ECE 466/566
Advanced Computer Networks

Thinh Nguyen
Email: thinhq@eecs.orst.edu
Electrical Engineering and Computer Science
Oregon State University

TA: Monchai Lertsutthiwong
Email: lertsumo@engr.orst.edu
Office Hours: 3-5 PM.
Chapter 7 outline

- 7.1 Multimedia Networking Applications
- 7.2 Streaming stored audio and video
- 7.3 Real-time Multimedia: Internet Phone study
- 7.4 Protocols for Real-Time Interactive Applications
  - RTP, RTCP, SIP
- 7.5 Distributing Multimedia: content distribution networks
- 7.6 Beyond Best Effort
- 7.7 Scheduling and Policing Mechanisms
- 7.8 Integrated Services and Differentiated Services
- 7.9 RSVP
Improving QoS in IP Networks

Thus far: “making the best of best effort”
Future: next generation Internet with QoS guarantees
  - RSVP: signaling for resource reservations
  - Differentiated Services: differential guarantees
  - Integrated Services: firm guarantees

simple model
for sharing and congestion studies:
Principles for QOS Guarantees

- Example: 1Mbps, IP phone, FTP share 1.5 Mbps link.
  - bursts of FTP can congest router, cause audio loss
  - want to give priority to audio over FTP

Principle 1
packet marking needed for router to distinguish between different classes; and new router policy to treat packets accordingly
Principles for QOS Guarantees (more)

- what if applications misbehave (audio sends higher than declared rate)
  - policing: force source adherence to bandwidth allocations
- marking and policing at network edge:
  - similar to ATM UNI (User Network Interface)

Principle 2

provide protection (*isolation*) for one class from others
Principles for QOS Guarantees (more)

- Allocating *fixed* (non-sharable) bandwidth to flow: *inefficient* use of bandwidth if flows doesn't use its allocation

![Diagram showing network resources and flows]

**Principle 3**

While providing isolation, it is desirable to use resources as efficiently as possible.
Principles for QOS Guarantees (more)

- Basic fact of life: cannot support traffic demands beyond link capacity

Principle 4

Call Admission: flow declares its needs, network may block call (e.g., busy signal) if it cannot meet needs
Summary of QoS Principles

Let’s next look at mechanisms for achieving this ....
What Can a Basic Router do to Packets?

- Send it...
- Delay it...
- Drop it...
- How they are done impacts Quality of Service
  - Best effort? Guaranteed delay? Guaranteed throughput?

- Many variations in policies with different behavior
- Rich body of research work to understand them
- Limited Internet deployment
  - Many practical deployment barriers since Internet was best-effort to begin with, adding new stuff is hard
  - Some people just don’t believe in the need for QoS! Not enough universal support
Router Architecture Assumptions

- Assumes inputs just forward packets to outputs
  - Switch core is N times faster than links in a NxN switch
  - No contention at input, no head-of-line blocking
  - Remember homework 2?
- Resource contention occurs only at the output interfaces
- Output interface has classifier, buffer/queue, scheduler components
Internet Classifier

- A “flow” is a sequence of packets that are related (e.g. from the same application)
- Flow in Internet can be identified by a subset of following fields in the packet header
  - source/destination IP address (32 bits)
  - source/destination port number (16 bits)
  - protocol type (8 bits)
  - type of service (4 bits)
- Examples:
  - All TCP packets from Thinh’s web browser on machine A to web server on machine B
  - All packets from OSU
  - All packets between OSU and Berkeley
  - All UDP packets from OSU ECE department
- Classifier takes a packet and decides which flow it belongs to
**Buffer/Queue**

- **Buffer**: memory where packets can be stored temporarily
- **Queue**: using buffers to store packets in an ordered sequence
  - E.g. First-in-First-Out (FIFO) queue
Buffer/Queue

- When packets arrive at an output port faster than the output link speed (perhaps only momentarily)
- Can drop all excess packets
  - Resulting in terrible performance
- Or can hold excess packets in buffer/queue
  - Resulting in some delay, but better performance
- Still have to drop packets when buffer is full
  - For a FIFO queue, “drop tail” or “drop head” are common policies
  - i.e. drop last packet to arrive vs drop first packet in queue to make room

- A chance to be smart: Transmission of packets held in buffer/queue can be *scheduled*
  - Which stored packet goes out next? Which is more “important”?
  - Impacts quality of service
Fair Rate Computation

- Denote
  - $C$ - link capacity
  - $N$ - number of flows
  - $r_i$ - arrival rate

- Max-min fair rate computation:
  1. compute $C/N$
  2. if there are flows $i$ such that $r_i \leq C/N$, update $C$ and $N$
     \[ C = C - \sum_{i : s.t. r_i \leq C} r_i \]
  3. if no, $f = C/N$, terminate
  4. go to 1

- A flow can receive at most the fair rate, i.e., $\min(f, r_i)$
Example of Fair Rate Computation

- $C = 10; \ r1 = 8, \ r2 = 6, \ r3 = 2; \ N = 3$
- $C/3 = 3.33 \Rightarrow C = C - r3 = 8; \ N = 2$
- $C/2 = 4; \ f = 4$

$f = 4$:
- $\min(8, 4) = 4$
- $\min(6, 4) = 4$
- $\min(2, 4) = 2$
Max-Min Fairness

- Associate a weight $w_i$ with each flow $i$
- If link congested, compute $f$ such that

$$\sum_i \min(r_i, f \times w_i) = C$$

\[
\begin{align*}
(w_1 = 3) & : 8 \\
(w_2 = 1) & : 6 \\
(w_3 = 1) & : 2 \\
\end{align*}
\]

\[
\begin{align*}
(f = 2: & ) \\
\min(8, 2\times3) & = 6 \\
\min(6, 2\times1) & = 2 \\
\min(2, 2\times1) & = 2 \\
\end{align*}
\]
Scheduler

- Decides how the output link capacity is shared by flows
  - Which packet from which flow gets to go out next?

- E.g. FIFO schedule
  - Simple schedule: whichever packet arrives first leaves first
  - Agnostic of concept of flows, no need for classifier, no need for a real “scheduler”, a FIFO queue is all you need

- E.g. TDMA schedule
  - Queue packets according to flows
    - Need classifier and multiple FIFO queues
  - Divide transmission times into slots, one slot per flow
  - Transmit a packet from a flow during its time slot
TDMA Scheduling

Classifier

Buffer management

flow 1
flow 2
flow n

TDMA Scheduler

1
2
Priority Scheduling

**Priority scheduling**: transmit highest priority queued packet

- multiple *classes*, with different priorities
  - class may depend on marking or other header info, e.g. IP source/dest, port numbers, etc..
  - Real world example?

![Diagram](image)
Round Robin Scheduling

round robin scheduling:
- multiple classes
- cyclically scan class queues, serving one from each class (if available)
- real world example?
**Internet Today**

- FIFO queues are used at most routers
- No classifier, no scheduler, best-effort

- Sophisticated mechanisms tend to be more common near the “edge” of the network
  - E.g. At campus routers
  - Use classifier to pick out Kazaa packets
  - Use scheduler to limit bandwidth consumed by Kazaa traffic
Achieving QoS in Statistical Multiplexing Network

- We want guaranteed QoS
- But we don’t want the inefficiency of TDMA
  - Unused time slots are “wasted”

- Want to statistically share un-reserved capacity or reserved but unused capacity

- One solution: Weighted Fair Queuing (WFQ)
  - Guarantees a flow receives at least its allocated bit rate
WFQ Architecture
What is Weighted Fair Queueing?

- Each flow $i$ given a weight (importance) $w_i$
- WFQ guarantees a minimum service rate to flow $i$
  - $r_i = R * w_i / (w_1 + w_2 + ... + w_n)$
  - Implies isolation among flows (one cannot mess up another)
What is the Intuition? Fluid Flow

water pipes

water buckets
Fluid Flow System

- If flows can be served one bit at a time
- WFQ can be implemented using bit-by-bit weighted round robin
  - During each round from each flow that has data to send, send a number of bits equal to the flow’s weight
Fluid Flow System: Example 1

Flow 1 \((w_1 = 1)\) 100 Kbps

Flow 2 \((w_2 = 1)\)

<table>
<thead>
<tr>
<th></th>
<th>Packet Size (bits)</th>
<th>Packet inter-arrival time (ms)</th>
<th>Arrival Rate (Kbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow 1</td>
<td>1000</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>Flow 2</td>
<td>500</td>
<td>10</td>
<td>50</td>
</tr>
</tbody>
</table>

Flow 1 (arrival traffic) 1 2 3 4 5

Flow 2 (arrival traffic) 1 2 3 4 5 6

Service in fluid flow system

0 10 20 30 40 50 60 70 80
Fluid Flow System: Example 2

- Red flow has packets backlogged between time 0 and 10
  - Backlogged flow → flow's queue not empty
- Other flows have packets continuously backlogged
- All packets have the same size
Fluid Flow System

Packets of size 10, 20 & 30 arrive at time 0
Fluid Flow System

Packets: time 0 size 15
time 5 size 20
time 15 size 10
Fluid Flow System

Packets: time 0 size 15
  time 5 size 20
  time 15 size 10
  time 18 size 15
Implementation in Packet System

- Packet (Real) system: packet transmission cannot be preempted. Why?
  Solution: serve packets in the order in which they would have finished being transmitted in the fluid flow system
Packet System: Example 1

Select the first packet that finishes in the fluid flow system.

Service in fluid flow system

Packet system
Packet System: Example 2

- Select the first packet that finishes in the fluid flow system.
Implementation Challenge

- Need to compute the finish time of a packet in the fluid flow system...
- ... but the finish time may change as new packets arrive!
- Need to update the finish times of all packets that are in service in the fluid flow system when a new packet arrives
  - But this is very expensive; a high speed router may need to handle hundreds of thousands of flows!
Example

- Four flows, each with weight 1

Finish times computed at time 0

Finish times re-computed at time $\epsilon$
Solution: Virtual Time

- Key Observation: while the finish times of packets may change when a new packet arrives, the order in which packets finish doesn’t!
  - Only the order is important for scheduling

- Solution: instead of the packet finish time maintain the round # when a packet finishes (virtual finishing time)
  - Virtual finishing time doesn’t change when a packet arrives

- System virtual time $V(t)$ - index of the round in the bit-by-bit round robin scheme
Example

Suppose each packet is 1000 bits, so takes 1000 rounds to finish.

So, packets of F1, F2, F3 finishes at virtual time 1000.

When packet F4 arrives at virtual time 1 (after one round), the virtual finish time of packet F4 is 1001.

But the virtual finish time of packet F1,2,3 remains 1000.

Finishing order is preserved.
System Virtual Time (Round #): $V(t)$

- $V(t)$ increases inversely proportionally to the sum of the weights of the backlogged flows.
- Since round # increases slower when there are more flows to visit each round.

Diagram:

- Flow 1 ($w_1 = 1$)
- Flow 2 ($w_2 = 1$)

Graph:

- $V(t)$ vs. time
- $C$ and $C/2$ markers

7: Multimedia Networking 7-39
Virtual Time = Service Done

A work conserving is defined as

\[
\frac{W_i(t, t + dt)}{w_i} = \frac{W(t, t + dt)}{\sum_{j \in B(t)} W_j} \quad \forall i \in B(t)
\]

where

- \( w_i \) - weight of flow \( i \)
- \( W_i(t_1, t_2) \) - total service received by flow \( i \) during \([t_1, t_2]\)
- \( W(t_1, t_2) \) - total service allocated to all flows during \([t_1, t_2]\)
- \( B(t) \) - number of flows backlogged
System Virtual Time: $V(t)$

- Virtual time = Work ($W$)

\[
W_i(t, t + dt) = w_i \times \frac{W(t, t + dt)}{\sum_{j \in B(t)} w_j} \quad \forall i \in B(t)
\]

\[
\frac{\partial W_i}{\partial t} = \sum_{j \in B(t)} w_j \times \frac{\partial W}{\partial t} \quad \forall i \in B(t)
\]

\[
W_i(t_1, t_2) = w_i \times \int_{t=t_1}^{t_2} \left( \frac{1}{\sum_{j \in B(t)} w_j} \times \frac{\partial W}{\partial t} \right) dt \quad \forall i \in B(t)
\]
Service Allocation

- The service received by flow $i$ during an interval $[t_1, t_2)$, while it is backlogged is

\[ W_i(t_1, t_2) = w_i \times \int_{t_1}^{t_2} \frac{\partial V_{GPS}}{\partial t} dt \quad \forall i \in B(t) \]

\[ W_i(t_1, t_2) = w_i \times (V_{GPS}(t_2) - V_{GPS}(t_1)) \quad \forall i \in B(t) \]
Virtual Time Implementation of Weighted Fair Queueing

\[ V(0) = 0 \]
\[ V(t_j + \tau) = V(t_j) + \frac{\tau}{\sum_{i \in B_j} w_i} \]
\[ S_j^k = \max(F_j^{k-1}, V(a_j^k)) \]
\[ F_j^k = S_j^k + \frac{L_j^k}{W_j} \]

- \( a_j^k \) - arrival time of packet \( k \) of flow \( j \)
- \( S_j^k \) - virtual starting time of packet \( k \) of flow \( j \)
- \( F_j^k \) - virtual finishing time of packet \( k \) of flow \( j \)
- \( L_j^k \) - length of packet \( k \) of flow \( j \)
- \( B_j \) - backlog flow (flow with packets in queue)
- Packets are sent in the increasing order of \( F_j^k \)
Properties of WFQ

- Guarantee that any packet is transmitted within \( \frac{packet\_length}{link\_capacity} \) of its transmission time in the fluid flow system
  - Can be used to provide guaranteed services

- Achieve fair allocation
  - Can be used to protect well-behaved flows against malicious flows
Router Support For Congestion Management

- Traditional Internet
  - Congestion control mechanisms at end-systems, mainly implemented in TCP
  - Routers play little role

- Router mechanisms affecting congestion management
  - Scheduling
  - Buffer management

- Traditional routers
  - FIFO
  - Tail drop
Drawbacks of FIFO with Tail-drop

- Buffer lock out by misbehaving flows
- Synchronizing effect for multiple TCP flows
- Burst or multiple consecutive packet drops
  - Bad for TCP fast recovery
FIFO Router with Two TCP Sessions

Packet Sequence Number vs Time (sec)

Flow 1
Flow 2

Number of packets vs Time (sec)

Queue Size

7: Multimedia Networking 7-47
**RED (Random Early Detection)**

- **FIFO scheduling**
- **Buffer management:**
  - Probabilistically discard packets
  - Probability is computed as a function of average queue length (why average?)

![Diagram](image)

**Discard Probability**

```
0 1
```

```
min_th  max_th  queue_len  Average Queue Length
```
**RED (cont’d)**

- **min_th** - minimum threshold
- **max_th** - maximum threshold
- **avg_len** - average queue length
  - \( \text{avg_len} = (1-w) \times \text{avg_len} + w \times \text{sample_len} \)

![Graph showing discard probability vs. queue length with thresholds](image)
**RED (cont’d)**

- If (avg_len < min_th) → enqueue packet
- If (avg_len > max_th) → drop packet
- If (avg_len ≥ min_th and avg_len < max_th) → discard packet with probability P

![Discard Probability (P) vs Queue Length](chart.png)
**RED (cont’d)**

- \( P = \max_P \times (\text{avg\_len} - \text{min\_th}) / (\text{max\_th} - \text{min\_th}) \)
- Improvements to spread the drops
  \( P' = P / (1 - \text{count} \times P) \), where
  - \( \text{count} \) – how many packets were consecutively enqueued since last drop
RED Advantages

- Absorb burst better
- Avoids synchronization
- Signal end systems earlier
RED Router with Two TCP Sessions
Policing Mechanisms
Policing Mechanisms

Goal: limit traffic to not exceed declared parameters

Three common-used criteria:

- **(Long term) Average Rate:** how many pkts can be sent per unit time (in the long run)
  - crucial question: what is the interval length: 100 packets per sec or 6000 packets per min have same average!
- **Peak Rate:** e.g., 6000 pkts per min. (ppm) avg.; 1500 ppm peak rate
- **(Max.) Burst Size:** max. number of pkts sent consecutively (with no intervening idle)
Policing Mechanisms

1. Leaky Bucket Algorithm

2. Token Bucket Algorithm
The Leaky Bucket Algorithm

- The **Leaky Bucket Algorithm** used to control rate in a network.

- Implemented as a single-server queue with constant service time.

- If the bucket (buffer) overflows then packets are discarded.
The Leaky Bucket Algorithm

(a) A leaky bucket with water. (b) A leaky bucket with packets.
The Leaky Bucket Algorithm

- The leaky bucket enforces a constant output rate regardless of the burstiness of the input. Does nothing when input is idle.

- The host injects one packet per clock tick onto the network. This results in a uniform flow of packets, smoothing out bursts and reducing congestion.

- When packets are the same size (as in ATM cells), the one packet per tick is okay. For variable length packets though, it is better to allow a fixed number of bytes per tick.
**Token Bucket**

**Token Bucket:** limit input to specified Burst Size and Average Rate.

- bucket can hold $b$ tokens
- tokens generated at rate $r$ tokens/sec unless bucket full
- over interval of length $t$: number of packets admitted less than or equal to $(r \times t + b)$. 
Token Bucket

- Characterized by three parameters \((b, r, R)\)
  - \(b\) – token depth
  - \(r\) – average arrival rate
  - \(R\) – maximum arrival rate (e.g., R link capacity)

- A bit is transmitted only when there is an available token
  - When a bit is transmitted exactly one token is consumed
Characterizing a Source by Token Bucket

- Arrival curve - maximum amount of bits transmitted by time $t$
- Use token bucket to bound the arrival curve
Example

- Arrival curve – maximum amount of bits transmitted by time t
- Use token bucket to bound the arrival curve
Policing Mechanisms

- token bucket, WFQ combine to provide guaranteed upper bound on delay, i.e., QoS guarantee!

\[
D_{\text{max}} = \frac{b}{R}
\]

- \( r \): token rate
- \( b \): bucket size
- \( R \): per-flow rate

\( \frac{\text{arriving traffic}}{\text{WFQ}} \)

\( \frac{r_1}{b_1} \)

\( \frac{r_n}{b_n} \)

\( \frac{w_1}{w_n} \)
Leaky Bucket vs. Token Bucket

- With TB, a packet can only be transmitted if there are enough tokens to cover its length in bytes.

- LB sends packets at an average rate. TB allows for large bursts to be sent faster by speeding up the output.

- TB allows saving up tokens (permissions) to send large bursts. LB does not allow saving.