Lecture #12: In-network resource management
continuing to share a network, this time with help from switches
reliable transport protocols deliver each byte of data exactly once, in-order, to the receiving application.

the sender is allowed to have $W$ outstanding packets at once, but no more.

**sequence numbers**: used to order the packets.

**acknowledgments ("ACKs")**: used to confirm that a packet has been received.

an ACK with sequence number $k$ indicates that the receiver has received all packets up to and including $k$.

**timeouts**: used to retransmit packets.

note that the sender could also infer loss because it has received multiple ACKs with sequence number 6, but none with sequence number $> 7$; we’ll come back to that.

this is known as a **sliding-window protocol**.

the window of outstanding (un-ACKed) packets slides along the sequence number space.
1970s: ARPAnet
1978: flexibility and layering
early 80s: growth → change
late 80s: growth → problems
1993: commercialization

hosts.txt distance-vector routing
TCP, UDP OSPF, EGP, DNS
(congestion collapse)
policy routing CIDR

CAIDA's IPv4 AS Core, January 2020
(https://www.caida.org/projects/cartography/as-core/2020/)

question: TCP congestion control doesn't react to congestion until after it's a problem; could we get senders to react before queues are full?

application: the things that actually generate traffic
transport: sharing the network, reliability (or not)
examples: TCP, UDP
network: naming, addressing, routing
examples: IP
link: communication between two directly-connected nodes
examples: ethernet, bluetooth, 802.11 (wi-fi)
question: TCP congestion control doesn’t react to congestion until after it’s a problem; could we get senders to react before queues are full?
queue management: given a queue, when should it drop packets?

- **droptail**: drop packets only when the queue is full. Simple, but leads to high delays and synchronizes flows.

- **RED**: drop packets before the queue is full, with increasing probability as the queue grows. Prevents queue lengths from oscillating, decreases delay, and flows don’t synchronize.
**queue management**: given a queue, when should it drop packets?

**droptail**: drop packets only when the queue is full.
queue management: given a queue, when should it drop packets?

**droptail**: drop packets only when the queue is full.
simple, but leads to high delays and synchronizes flows.
queue management: given a queue, when should it drop packets?

**droptail**: drop packets only when the queue is full. simple, but leads to high delays and synchronizes flows.

**RED**: drop packets before the queue is full, with increasing probability as the queue grows.
queue management: given a queue, when should it drop packets?

**droptail:** drop packets only when the queue is full. simple, but leads to high delays and synchronizes flows.

**RED:** drop packets before the queue is full, with increasing probability as the queue grows.
**Queue management:** given a queue, when should it drop packets?

**Droptail:** drop packets only when the queue is full. Simple, but leads to high delays and synchronizes flows.

**RED:** drop packets before the queue is full, with increasing probability as the queue grows.
queue management: given a queue, when should it drop packets?

**droptail:** drop packets only when the queue is full. Simple, but leads to high delays and synchronizes flows.

**RED:** drop packets before the queue is full, with increasing probability as the queue grows.

![Graph showing average queue size vs drop probability](image-url)
**queue management**: given a queue, when should it drop packets?

**droptail**: drop packets only when the queue is full. Simple, but leads to high delays and synchronizes flows.

**RED**: drop packets before the queue is full, with increasing probability as the queue grows.
**queue management:** given a queue, when should it drop packets?

**droptail:** drop packets only when the queue is full. simple, but leads to high delays and synchronizes flows.

**RED:** drop packets before the queue is full, with increasing probability as the queue grows.
queue management: given a queue, when should it drop packets?

**droptail**: drop packets only when the queue is full. simple, but leads to high delays and synchronizes flows.

**RED**: drop packets before the queue is full, with increasing probability as the queue grows.
**queue management**: given a queue, when should it drop packets?

**droptail**: drop packets only when the queue is full. Simple, but leads to high delays and synchronizes flows.

**RED**: drop packets before the queue is full, with increasing probability as the queue grows.
queue management: given a queue, when should it drop packets?

**droptail**: drop packets only when the queue is full. Simple, but leads to high delays and synchronizes flows.

**RED**: drop packets before the queue is full, with increasing probability as the queue grows. Prevents queue lengths from oscillating, decreases delay, flows don't synchronize.
queue management: given a queue, when should it drop packets?

**droptail**: drop packets only when the queue is full. simple, but leads to high delays and synchronizes flows.

**RED**: drop packets before the queue is full, with increasing probability as the queue grows. prevents queue lengths from oscillating, decreases delay, flows don't synchronize.
**queue management**: given a queue, when should it drop (or mark) packets?

**droptail**: drop packets only when the queue is full. simple, but leads to high delays and synchronizes flows.

**RED (drops) / ECN (marks)**: drop (or mark) packets before the queue is full: with increasing probability as the queue grows. prevents queue lengths from oscillating, decreases delay, flows don’t synchronize.
**queue management**: given a queue, when should it drop (or mark) packets?

**droptail**: drop packets only when the queue is full. simple, but leads to high delays and synchronizes flows.

**RED (drops) / ECN (marks)**: drop (or mark) packets before the queue is full: with increasing probability as the queue grows. prevents queue lengths from oscillating, decreases delay, flows don’t synchronize. but complex and hard to pick parameters.
queue management: given a queue, when should it drop (or mark) packets?

**droptail**: drop packets only when the queue is full. simple, but leads to high delays and synchronizes flows.

**RED (drops) / ECN (marks)**: drop (or mark) packets before the queue is full: with increasing probability as the queue grows. prevents queue lengths from oscillating, decreases delay, flows don’t synchronize. but complex and hard to pick parameters

as long as our switches are taking a more active role, let’s see what else they can do
(we’ll return to queue management later in the lecture)
delay-based scheduling: can we give latency guarantees for some types of traffic?
**delay-based scheduling:** can we give latency guarantees for some types of traffic?

**priority queueing:** put latency-sensitive traffic in its own queue and serve that queue first (can extend this idea to multiple queues/types of traffic)
**delay-based scheduling:** can we give latency guarantees for some types of traffic?

**priority queueing:** put latency-sensitive traffic in its own queue and serve that queue first (can extend this idea to multiple queues/types of traffic)

**question:** what could go wrong here?
**Priority Queueing:** put latency-sensitive traffic in its own queue and serve that queue first. Does *not* prevent the latency-sensitive traffic from "starving out" the other traffic (in other queues).

**Delay-based Scheduling:** can we give latency guarantees for some types of traffic?
**delay-based scheduling:** can we give latency guarantees for some types of traffic?

**priority queueing:** put latency-sensitive traffic in its own queue and serve that queue first. *does not* prevent the latency-sensitive traffic from “starving out” the other traffic (in other queues).

**as long as our switches are taking a more active role, let’s see what else they can do**

(we’ll return to priority queueing later in the lecture)
**bandwidth-based scheduling:** can we allocate specific amounts of bandwidth to some traffic?
bandwidth-based scheduling: can we allocate specific amounts of bandwidth to some traffic?

round robin: can’t handle variable packet sizes (and in its most basic form doesn’t allow us to weight traffic differently)
**bandwidth-based scheduling**: can we allocate specific amounts of bandwidth to some traffic?

**round robin**: can’t handle variable packet sizes (and in its most basic form doesn’t allow us to weight traffic differently)

**deficit round robin**: handles variable packet sizes (even within the same queue), near-perfect fairness and low packet processing overhead
**bandwidth-based scheduling**: can we allocate specific amounts of bandwidth to some traffic?

**round robin**: can’t handle variable packet sizes (and in its most basic form doesn’t allow us to weight traffic differently)

**deficit round robin**: handles variable packet sizes (even within the same queue), near-perfect fairness and low packet processing overhead

**in each round:**

q.credit += q.quantum

while q.credit >= size of next packet p:

q.credit -= size of p

send p
**bandwidth-based scheduling:** can we allocate specific amounts of bandwidth to some traffic?

**round robin:** can’t handle variable packet sizes (and in its most basic form doesn’t allow us to weight traffic differently)

**deficit round robin:** handles variable packet sizes (even within the same queue), near-perfect fairness and low packet processing overhead

**in each round:**

**for each queue q:**
**bandwidth-based scheduling:** can we allocate specific amounts of bandwidth to some traffic?

**round robin:** can’t handle variable packet sizes (and in its most basic form doesn’t allow us to weight traffic differently)

**deficit round robin:** handles variable packet sizes (even within the same queue), near-perfect fairness and low packet processing overhead

in each round:

for each queue q:

q.credit += q.quantum
**bandwidth-based scheduling:** can we allocate specific amounts of bandwidth to some traffic?

**round robin:** can’t handle variable packet sizes (and in its most basic form doesn’t allow us to weight traffic differently)

**deficit round robin:** handles variable packet sizes (even within the same queue), near-perfect fairness and low packet processing overhead

in each round:

for each queue q:

```
q.credit += q.quantum
```

while `q.credit` >= size of next packet `p`:

```
q.credit -= size of p
send p
```
**bandwidth-based scheduling:** can we allocate specific amounts of bandwidth to some traffic?

**round robin:** can’t handle variable packet sizes (and in its most basic form doesn’t allow us to weight traffic differently)

**deficit round robin:** handles variable packet sizes (even within the same queue), near-perfect fairness and low packet processing overhead

---

in each round:

for each queue q:

\[ q.credit += q.quantum \]

while \( q.credit \geq \text{size of next packet } p \):

\[ q.credit -= \text{size of } p \]

send p

---

deficit round robin also doesn’t require a mean packet size, which is another good thing
**bandwidth-based scheduling:** can we allocate specific amounts of bandwidth to some traffic?

**round robin:** can’t handle variable packet sizes (and in its most basic form doesn’t allow us to weight traffic differently)

**deficit round robin:** handles variable packet sizes (even within the same queue), near-perfect fairness and low packet processing overhead

---

**in each round:**

for each queue q:

\[ q.\text{credit} += q.\text{quantum} \]

while \[ q.\text{credit} \geq \text{size of next packet } p \]

\[ q.\text{credit} -= \text{size of } p \]

send \( p \)

---

**question:** suppose one of our queues is empty for many rounds. should it accumulate credit while empty?
**bandwidth-based scheduling**: can we allocate specific amounts of bandwidth to some traffic?

**round robin**: can’t handle variable packet sizes (and in its most basic form doesn’t allow us to weight traffic differently)

**deficit round robin**: handles variable packet sizes (even within the same queue), near-perfect fairness and low packet processing overhead

---

**in each round**:

for each queue q:

  if q is not empty:
    q.credit += q.quantum

  while q.credit >= size of next packet p:
    q.credit -= size of p
    send p

else:
  q.credit = 0

---

deficit round robin also doesn’t require a mean packet size, which is another good thing
**bandwidth-based scheduling**: can we allocate specific amounts of bandwidth to some traffic?

**round robin**: can’t handle variable packet sizes (and in its most basic form doesn’t allow us to weight traffic differently)

**deficit round robin**: handles variable packet sizes (even within the same queue), near-perfect fairness and low packet processing overhead

---

**in each round:**

for each queue q:

  if q is not empty:
    q.credit += q.quantum

    while q.credit >= size of next packet p:
      q.credit -= size of p
      send p
  else:
    q.credit = 0

---

now let’s start revisiting some of our previous strategies
**delay-based scheduling**: can we give latency guarantees for some types of traffic?

**priority queueing**: put latency-sensitive traffic in its own queue and serve that queue first. does *not* prevent the latency-sensitive traffic from “starving out” the other traffic (in other queues).
**delay-based scheduling:** can we give latency guarantees for some types of traffic?

**priority queueing:** put latency-sensitive traffic in its own queue and serve that queue first. does *not* prevent the latency-sensitive traffic from “starving out” the other traffic (in other queues).

can solve this problem by doing something similar to bandwidth-based scheduling across the two queues.
### Table: In-Network Resource Management

<table>
<thead>
<tr>
<th>Type of Management</th>
<th>What Does This Type of Management Allow a Switch to Do</th>
<th>Example Protocols</th>
<th>How the Protocol Works</th>
<th>Pros/Cons?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Queue Management</strong></td>
<td>signal congestion, potentially before queues are full</td>
<td>DropTail</td>
<td>drop packets when the queue is full</td>
<td>simple, but queues get full (among other problems)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RED, ECN</td>
<td>drop or mark packets before the queue is full</td>
<td>can keep queues from filling up, but complicated</td>
</tr>
<tr>
<td><strong>Delay-based Scheduling</strong></td>
<td>prioritize latency-sensitive traffic</td>
<td>Priority Queueing</td>
<td>serve some queues before others</td>
<td>prioritized queues can starve out the others</td>
</tr>
<tr>
<td><strong>Bandwidth-based Scheduling</strong></td>
<td>enforce (weighted) fairness among different types of traffic</td>
<td>Round-robin</td>
<td>try to give each type of traffic an equal share of bandwidth</td>
<td>can’t handle variable packet sizes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Weighted Round-robin</td>
<td>round robin, but incorporate average packet size</td>
<td>average packet size hard to get</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deficit Round-robin</td>
<td>round robin, but do a better job with packet sizes</td>
<td>honestly pretty good</td>
</tr>
</tbody>
</table>
### in-network resource management

<table>
<thead>
<tr>
<th>Type of Management</th>
<th>What Does This Type of Management Allow a Switch to Do</th>
<th>Example Protocols</th>
<th>How the Protocol Works</th>
<th>Pros/Cons?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Queue Management</strong></td>
<td>signal congestion, potentially before queues are full</td>
<td>DropTail</td>
<td>drop packets when the queue is full</td>
<td>simple, but queues get full (among other problems)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RED, ECN</td>
<td>drop or mark packets before the queue is full</td>
<td>can keep queues from filling up, but complicated</td>
</tr>
<tr>
<td><strong>Delay-based Scheduling</strong></td>
<td>prioritize latency-sensitive traffic</td>
<td>Priority Queueing</td>
<td>serve some queues before others</td>
<td>prioritized queues can starve out the others</td>
</tr>
<tr>
<td><strong>Bandwidth-based Scheduling</strong></td>
<td>enforce (weighted) fairness among different types of traffic</td>
<td>Round-robin</td>
<td>try to give each type of traffic an equal share of bandwidth</td>
<td>can’t handle variable packet sizes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Weighted Round-robin</td>
<td>round robin, but incorporate average packet size</td>
<td>average packet size hard to get</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deficit Round-robin</td>
<td>round robin, but do a better job with packet sizes</td>
<td>honestly pretty good</td>
</tr>
</tbody>
</table>

We didn’t cover weighted round-robin; this is just to give you a sense that there are algorithms that exist “between” round-robin and deficit round-robin.
### In-network Resource Management

<table>
<thead>
<tr>
<th>Type of Management</th>
<th>What Does This Type of Management Allow a Switch to Do</th>
<th>Example Protocols</th>
<th>How the Protocol Works</th>
<th>Pros/Cons?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queue Management</td>
<td>signal congestion, potentially before queues are full</td>
<td>DropTail</td>
<td>drop packets when the queue is full</td>
<td>simple, but queues get full (among other problems)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RED, ECN</td>
<td>drop or mark packets before the queue is full</td>
<td>can keep queues from filling up, but complicated</td>
</tr>
<tr>
<td>Delay-based Scheduling</td>
<td>prioritize latency-sensitive traffic</td>
<td>Priority Queueing</td>
<td>serve some queues before others</td>
<td>prioritized queues can starve out the others</td>
</tr>
<tr>
<td>Bandwidth-based Scheduling</td>
<td>enforce (weighted) fairness among different types of traffic</td>
<td>Round-robin</td>
<td>try to give each type of traffic an equal share of bandwidth</td>
<td>can’t handle variable packet sizes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Weighted Round-robin</td>
<td>round robin, but incorporate average packet size</td>
<td>average packet size hard to get</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deficit Round-robin</td>
<td>round robin, but do a better job with packet sizes</td>
<td>honestly pretty good</td>
</tr>
</tbody>
</table>
## in-network resource management

<table>
<thead>
<tr>
<th>type of management</th>
<th>what does this type of management allow a switch to do</th>
<th>example protocols</th>
<th>how the protocol works</th>
<th>pros/cons?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Queue Management</strong></td>
<td>signal congestion, potentially before queues are full</td>
<td>DropTail</td>
<td>drop packets when the queue is full</td>
<td>simple, but queues get full (among other problems)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RED, ECN</td>
<td>drop or mark packets before the queue is full</td>
<td>can keep queues from filling up, but complicated</td>
</tr>
<tr>
<td><strong>Delay-based Scheduling</strong></td>
<td>prioritize latency-sensitive traffic</td>
<td>Priority Queueing</td>
<td>serve some queues before others</td>
<td>prioritized queues can starve out the others</td>
</tr>
<tr>
<td><strong>Bandwidth-based Scheduling</strong></td>
<td>enforce (weighted) fairness among different types of traffic</td>
<td>Round-robin</td>
<td>try to give each type of traffic an equal share of bandwidth</td>
<td>can't handle variable packet sizes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Weighted Round-robin</td>
<td>round robin, but incorporate average packet size</td>
<td>average packet size hard to get</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deficit Round-robin</td>
<td>round robin, but do a better job with packet sizes</td>
<td>honestly pretty good</td>
</tr>
</tbody>
</table>

**is in-network resource management a good idea on the Internet?**
1970s: ARPAnet
1978: flexibility and layering
early 80s: growth → change
late 80s: growth → problems
1993: commercialization

<table>
<thead>
<tr>
<th>hosts.txt</th>
<th>distance-vector routing</th>
<th>TCP, UDP</th>
<th>OSPF, EGP, DNS</th>
<th>congestion collapse</th>
<th>policy routing</th>
<th>CIDR</th>
</tr>
</thead>
</table>

question: TCP congestion control doesn’t react to congestion until after it’s a problem; could we get senders to react before queues are full? **yes, if switches take a more active role**
Sally Floyd, Who Helped Things Run Smoothly Online, Dies at 69

In the early 1990s, Dr. Floyd was one of the inventors of Random Early Detection, which continues to play a vital role in the stability of the internet.

One byproduct of Dr. Floyd’s work reflected her passion for keeping things fair to all internet users. “Her work on congestion control was about keeping it working for everyone,” Dr. Kohler said. “For people with fast connections, and for people with slow connections.”