• In-network resource management
• Queue management schemes
• Traffic differentiation
Problems to Solve

How do we **route** (and address) scalably, while dealing with issues of policy and economy?

How do we **transport** data scalably, while dealing with varying application demands?

How do we **adapt** new applications and technologies to an inflexible architecture?

**Hierarchy of routing**
(link-state or distance-vector within an AS, BGP across ASes)

**TCP**
How do we route (and address) scalably, while dealing with issues of policy and economy?

How do we transport data scalably, while dealing with varying application demands?

How do we adapt new applications and technologies to an inflexible architecture?

Hierarchy of routing (link-state or distance-vector within an AS, BGP across ASes)

TCP, in-network resource management
**problem:** TCP reacts to drops, and packets aren’t dropped until queues are full
Queue Management

given a queue, when do we drop packets?

1. droptail
   drop packets only when the queue is full
Queue Management

given a queue, when do we drop packets?

1. droptail

drop packets only when the queue is full. simple, but leads to high delays and synchronizes flows.
Queue Management

given a queue, when do we drop packets?

1. droptail

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

2. RED

   drop packets before the queue is full
graph showing the relationship between drop probability and average queue size.
average queue size

drop probability

never drop

$q_{min}$

$q_{max}$

0

1
The diagram illustrates the relationship between drop probability and the average queue size. The vertical axis represents the drop probability, ranging from 0 to 1, with 'never drop' at 0 and 'always drop' at 1. The horizontal axis represents the average queue size, ranging from $q_{\text{min}}$ to $q_{\text{max}}$. The graph shows that as the average queue size increases, the drop probability increases, transitioning from 'never drop' to 'always drop'.
The graph illustrates the relationship between average queue size and drop probability. It shows three regions:

1. **Never drop**
   - For queue sizes less than or equal to $q_{\text{min}}$.

2. **Drop more frequently as queue size increases**
   - For queue sizes greater than $q_{\text{min}}$ but less than or equal to $q_{\text{max}}$.

3. **Always drop**
   - For queue sizes greater than $q_{\max}$.

The y-axis represents drop probability, ranging from 0 to 1. The x-axis represents the average queue size, ranging from $q_{\text{min}}$ to $q_{\text{max}}$. The graph visually demonstrates how the drop probability changes with varying queue sizes.
The diagram illustrates the relationship between the average queue size and the drop probability. As the average queue size increases, the drop probability increases linearly from 0 at \( q_{\text{min}} \) to \( \rho_{\text{max}} \) at \( q_{\text{max}} \). The graph shows:

- **Never drop** when the queue size is below \( q_{\text{min}} \).
- **Drop more frequently as queue size increases** between \( q_{\text{min}} \) and \( q_{\text{max}} \).
- **Always drop** when the queue size is equal to or exceeds \( q_{\text{max}} \).
Queue Management

given a queue, when do we drop packets?

1. droptail

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

2. 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 do we drop (or mark) packets?

1. droptail

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

2. 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 do we drop (or mark) packets?

1. droptail

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

2. 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
what if we want to give latency guarantees to certain types of traffic?
(or at least try to prioritize latency-sensitive traffic)
Delay-based Scheduling

how could we give latency guarantees for some traffic?

1. **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

how could we give latency guarantees for some traffic?

1. 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).
what if we want to allocate different amounts of bandwidth to different types of traffic?
Bandwidth-based Scheduling

how can we allocate a specific amount of network bandwidth to some traffic?

1. round-robin
Bandwidth-based Scheduling

how can we allocate a specific amount of network bandwidth to some traffic?

1. 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

how can we allocate a specific amount of network bandwidth to some traffic?

1. round-robin

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

2. weighted round-robin

can set weights and deal with variable packet sizes
Weighted Round Robin

in each round:
Weighted Round Robin

in each round:

for each queue q:
    q.norm = q.weight / q.mean_packet_size
Weighted Round Robin

in each round:

for each queue q:
    q.norm = q.weight / q.mean_packet_size

min = min of q.norm’s over all flows
Weighted Round Robin

in each round:

for each queue q:
  \( q.\text{norm} = q.\text{weight} / q.\text{mean}_\text{packet}_\text{size} \)

\( \text{min} = \text{min of } q.\text{norm}'s \) over all flows

for each queue q:
  \( q.\text{n}_\text{packets} = q.\text{norm} / \text{min} \)
Weighted Round Robin

in each round:

for each queue q:
    \( q.\text{norm} = \frac{q.\text{weight}}{q.\text{mean_packet_size}} \)

\( \text{min} = \text{min of } q.\text{norm}'s \text{ over all flows} \)

for each queue q:
    \( q.\text{n_packets} = \frac{q.\text{norm}}{\text{min}} \)
    send \( q.\text{n_packets} \) from queue q
Bandwidth-based Scheduling

how can we allocate a specific amount of network bandwidth to some traffic?

1. round-robin

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

2. weighted round-robin

can set weights and deal with variable packet sizes, but needs to know mean packet sizes
Bandwidth-based Scheduling

how can we allocate a specific amount of network bandwidth to some traffic?

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

2. **weighted round-robin**
   
can set weights and deal with variable packet sizes, but needs to know mean packet sizes

3. **deficit round-robin**
Deficit Round Robin

in each round:
in each round:
  for each queue q:
    q.credit += q.quantum
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

how can we allocate a specific amount of network bandwidth to some traffic?

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

2. **weighted round-robin**
   - can set weights and deal with variable packet sizes, but needs to know mean packet sizes

3. **deficit round-robin**
   - doesn’t need mean packet sizes. near-perfect fairness and low packet processing overhead
Delay-based Scheduling
how could we give latency guarantees for some traffic?

1. 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
In-network Resource Management

**Queue Management**
- Switches can signal congestion before queues are full

**Delay-based Scheduling**
- Switches can prioritize latency-sensitive traffic

**Bandwidth-based Scheduling**
- Switches can enforce (weighted) fairness among different types of traffic
In-network Resource Management

**Queue Management**
- switches can signal congestion before queues are full

**Delay-based Scheduling**
- switches can prioritize latency-sensitive traffic

**Bandwidth-based Scheduling**
- switches can enforce (weighted) fairness among different types of traffic

**Techniques**
- DropTail
- RED
- ECN
- Priority Queueing
- Round-robin
- Weighted Round-robin
- Deficit Round-robin

**In-network resource management: a good idea?**
• Active **queue management schemes**, such as RED or ECN, drop or mark packets before a queue is full, in hopes of getting TCP senders to react earlier to congestion. They are difficult to get to work on the Internet-at-large, but the ideas can be useful in other types of networks.

• **Traffic differentiation** requires a scheduling discipline, such as *weighted round robin* or *deficit round robin*. The goal of these schemes is to give weighted fairness in the face of variable packet sizes while having low processing overhead.

• Both of these are examples of **in-network resource management**