# 6.1800 Spring 2025

## Lecture #12: In-network resource management

continuing to share a network, this time with help from switches

Katrina LaCurts | lacurts@mit.edu | 6.1800 2025

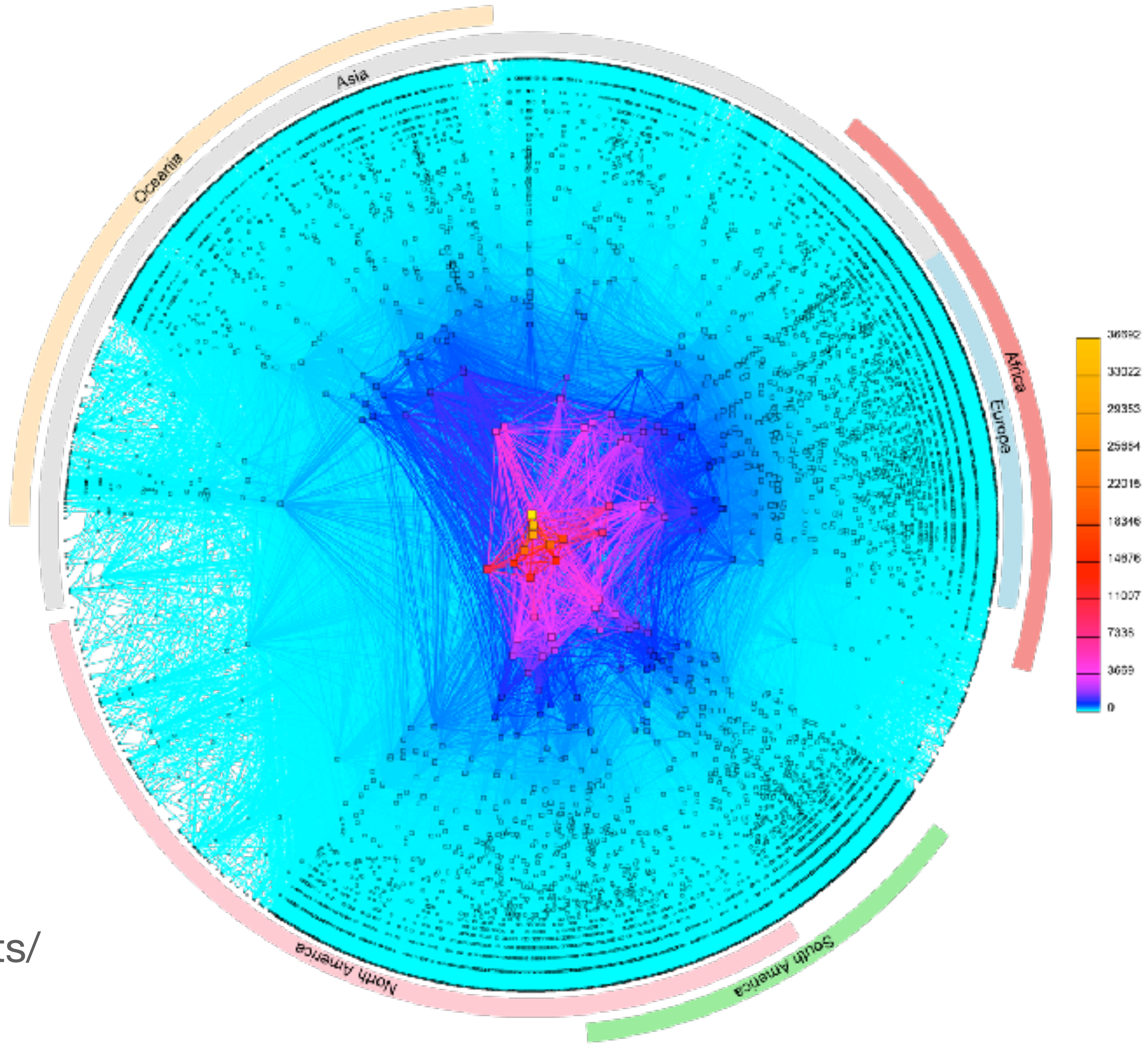
1970s: ARPAnet — hosts.txt

1978: flexibility and layering — distance-vector routing

early 80s: growth → change — **TCP**, UDP; OSPF, EGP, DNS

late 80s: growth → problems — congestion collapse (which led to **congestion control**); policy routing; CIDR

1993: commercialization




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)` <br> *examples: TCP, UDP* |
| `network` | `naming, addressing, routing` <br> *examples: IP* |
| `link` | `communication between two directly-connected nodes` <br> *examples: ethernet, bluetooth, 802.11 (wifi)* |

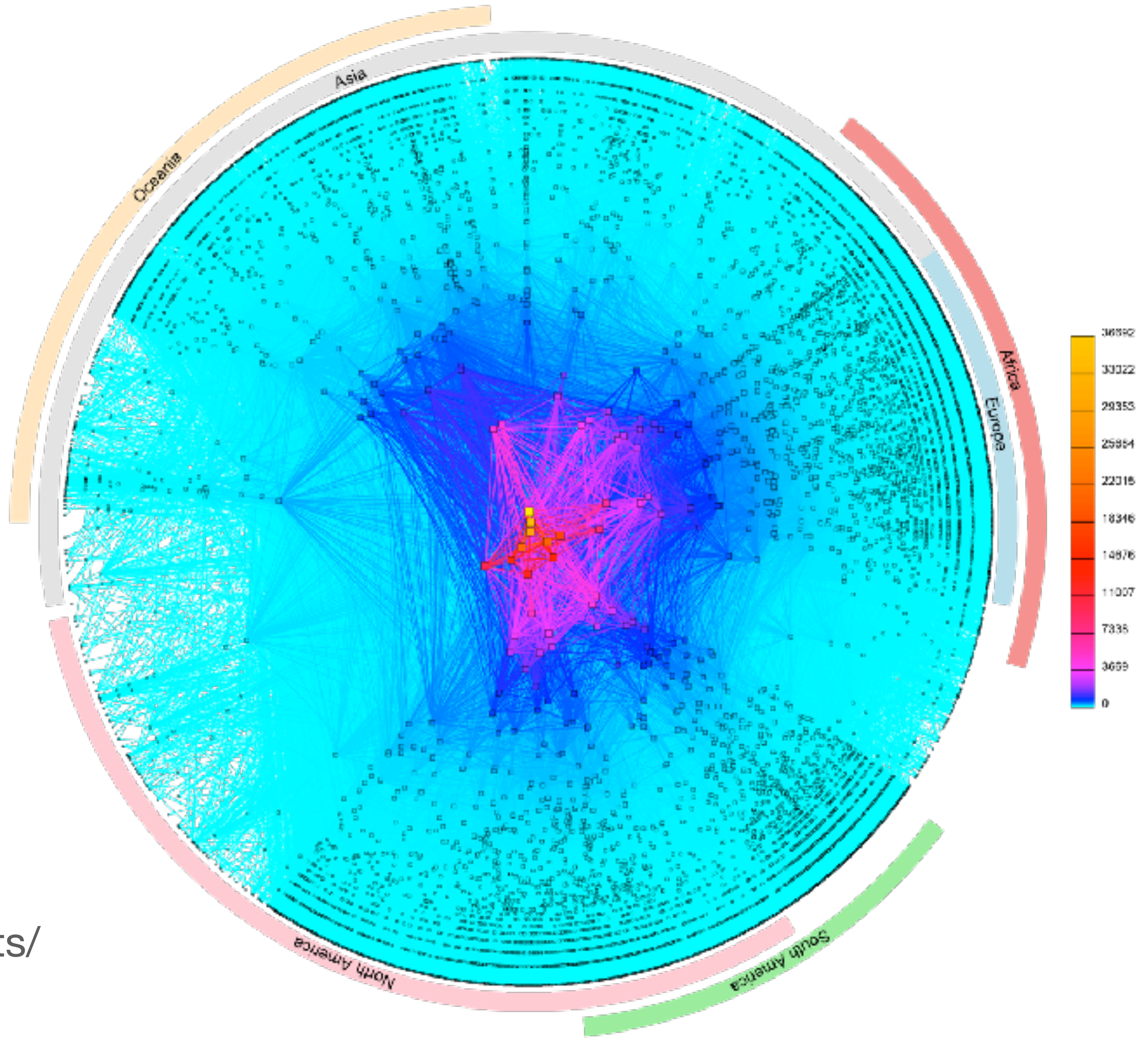
1970s: ARPAnet — hosts.txt

1978: flexibility and layering — distance-vector routing

early 80s: growth → change — TCP, UDP; OSPF, EGP, DNS

late 80s: growth → problems — congestion collapse (which led to congestion control); policy routing; CIDR

1993: commercialization




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

| | |
|---|---|
| **application** | the things that actually generate traffic |
| **transport** | sharing the network, reliability (or not)<br>*examples: TCP, UDP* |
| **network** | naming, addressing, routing<br>*examples: IP* |
| **link** | communication between two directly-connected nodes<br>*examples: ethernet, bluetooth, 802.11 (wifi)* |

**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?

**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?

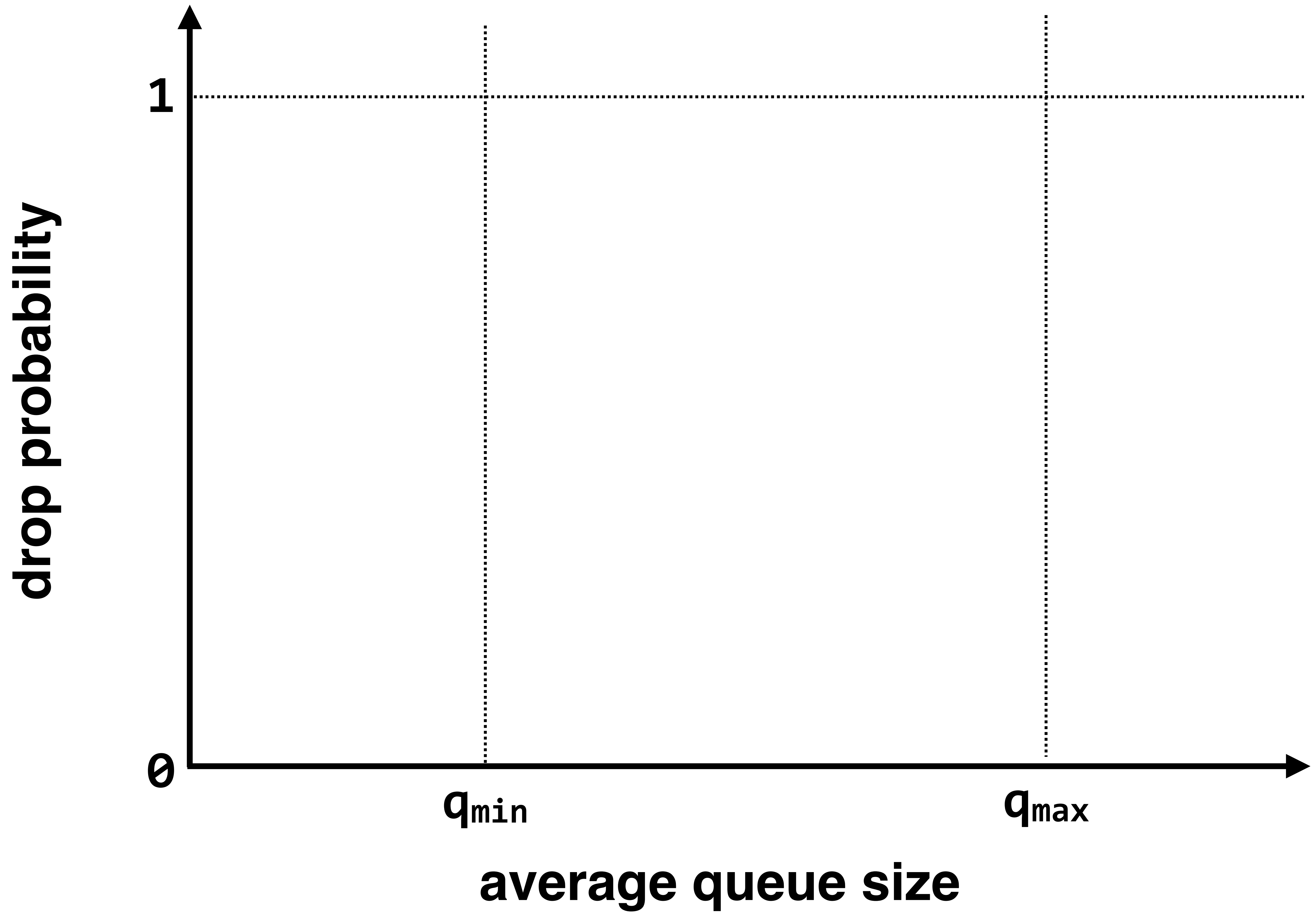
**droptail:** drop packets only when the queue is full. simple, but leads to high delays and synchronizes flows.

**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.

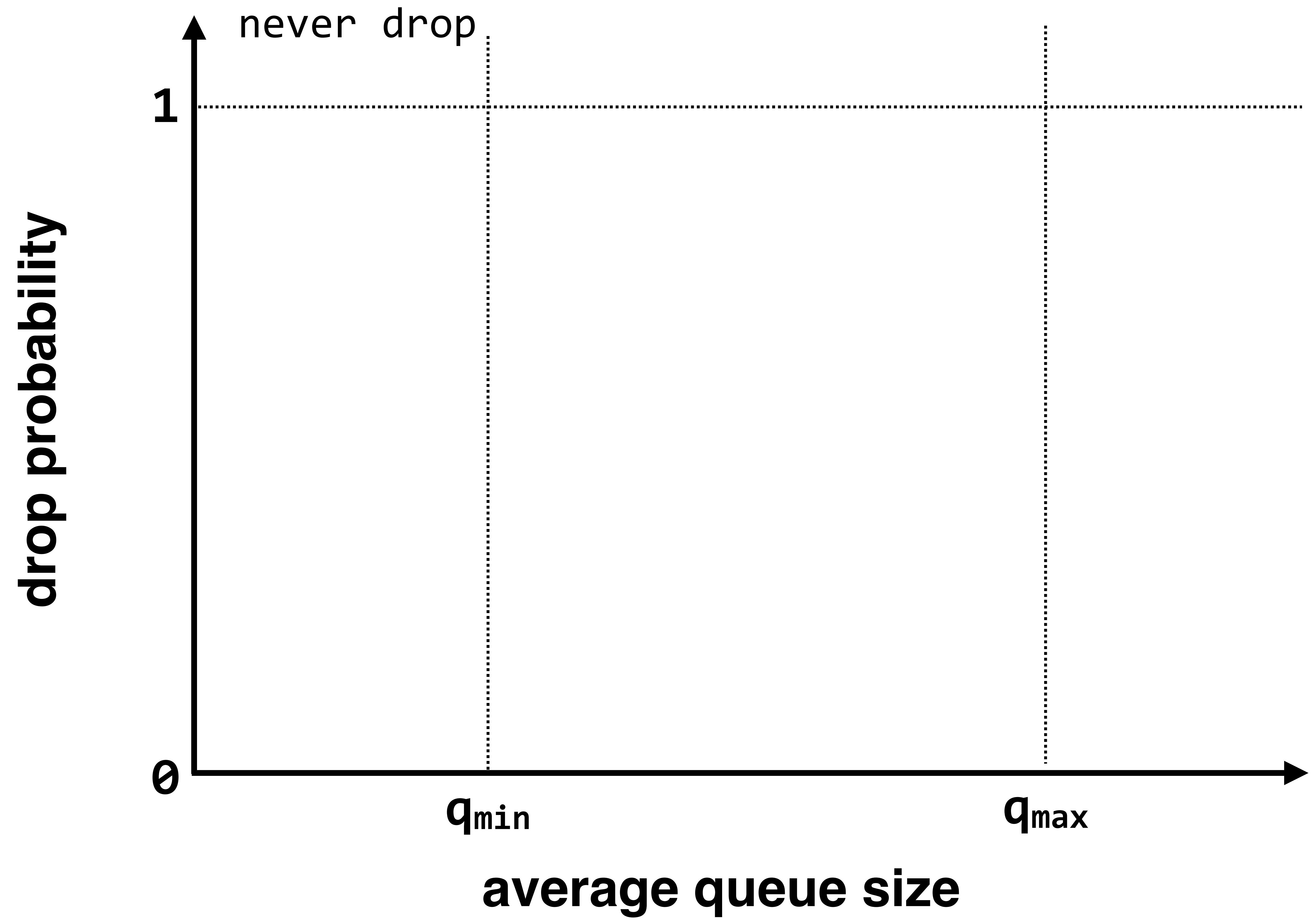
**RED:** drop packets before the queue is full, with increasing probability as the queue grows.



## **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.

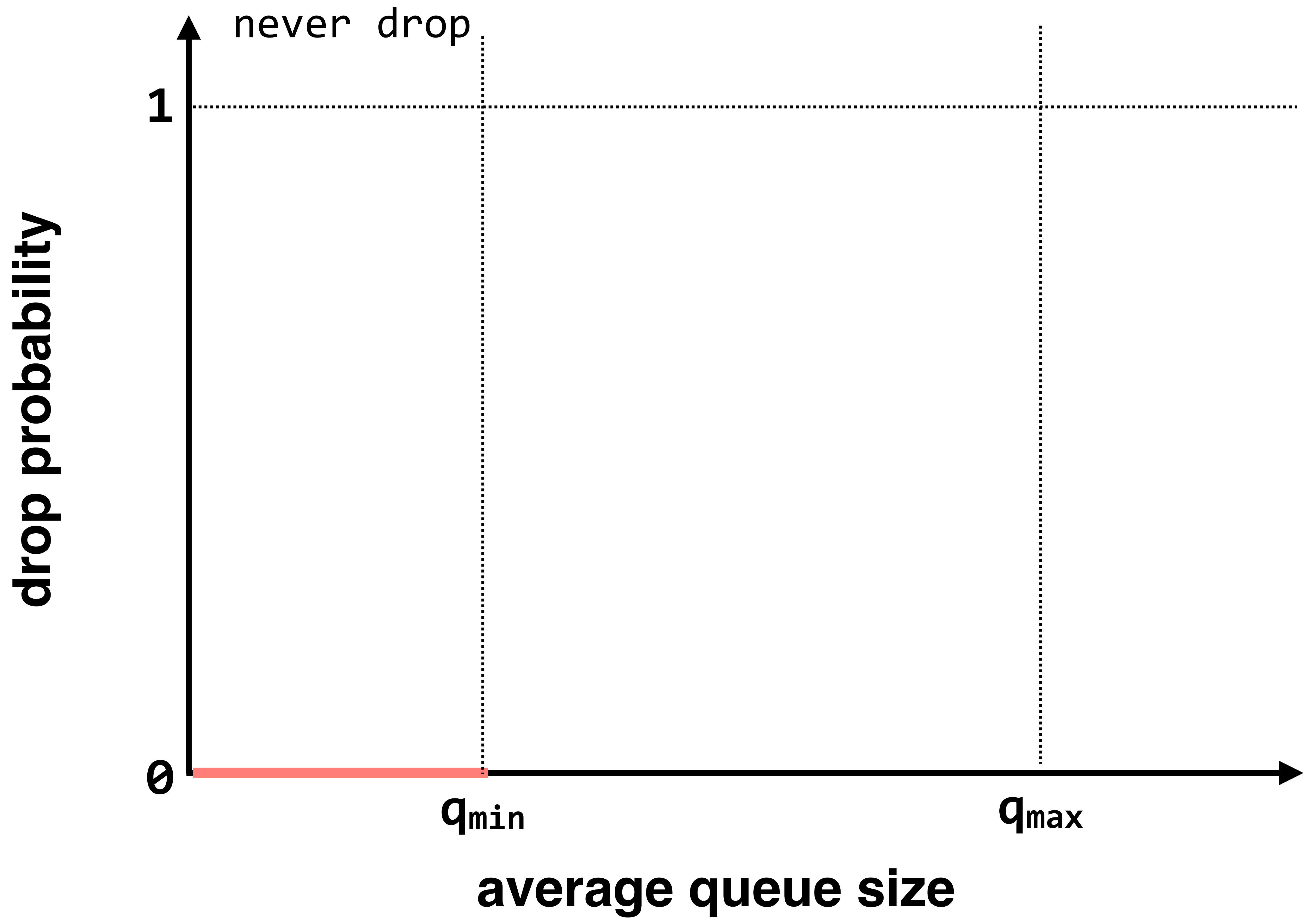
**RED:** drop packets before the queue is full, with increasing probability as the queue grows.



**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.

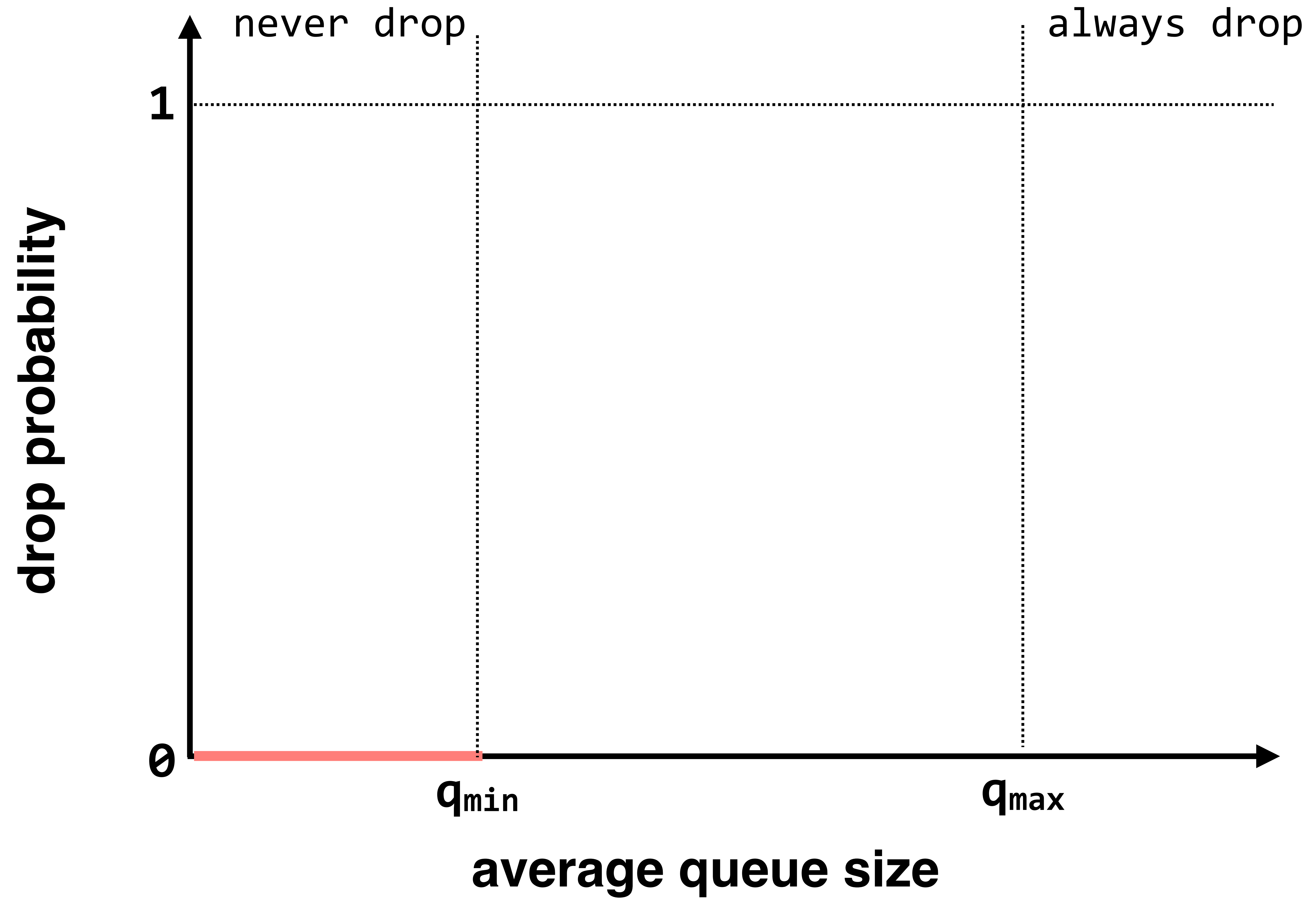
**RED:** drop packets before the queue is full, with increasing probability as the queue grows.



# **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.

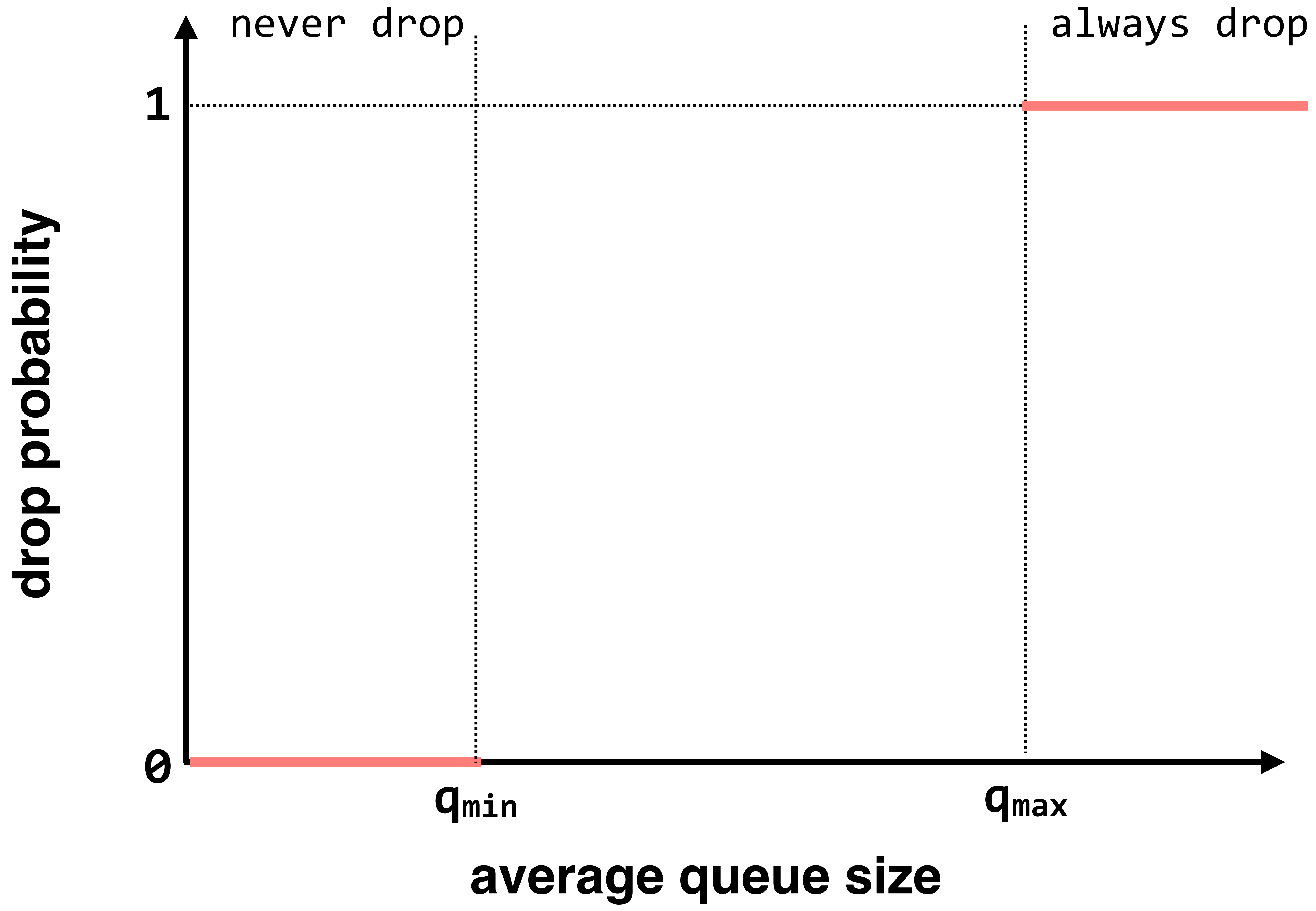
**RED:** drop packets before the queue is full, with increasing probability as the queue grows.



**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.

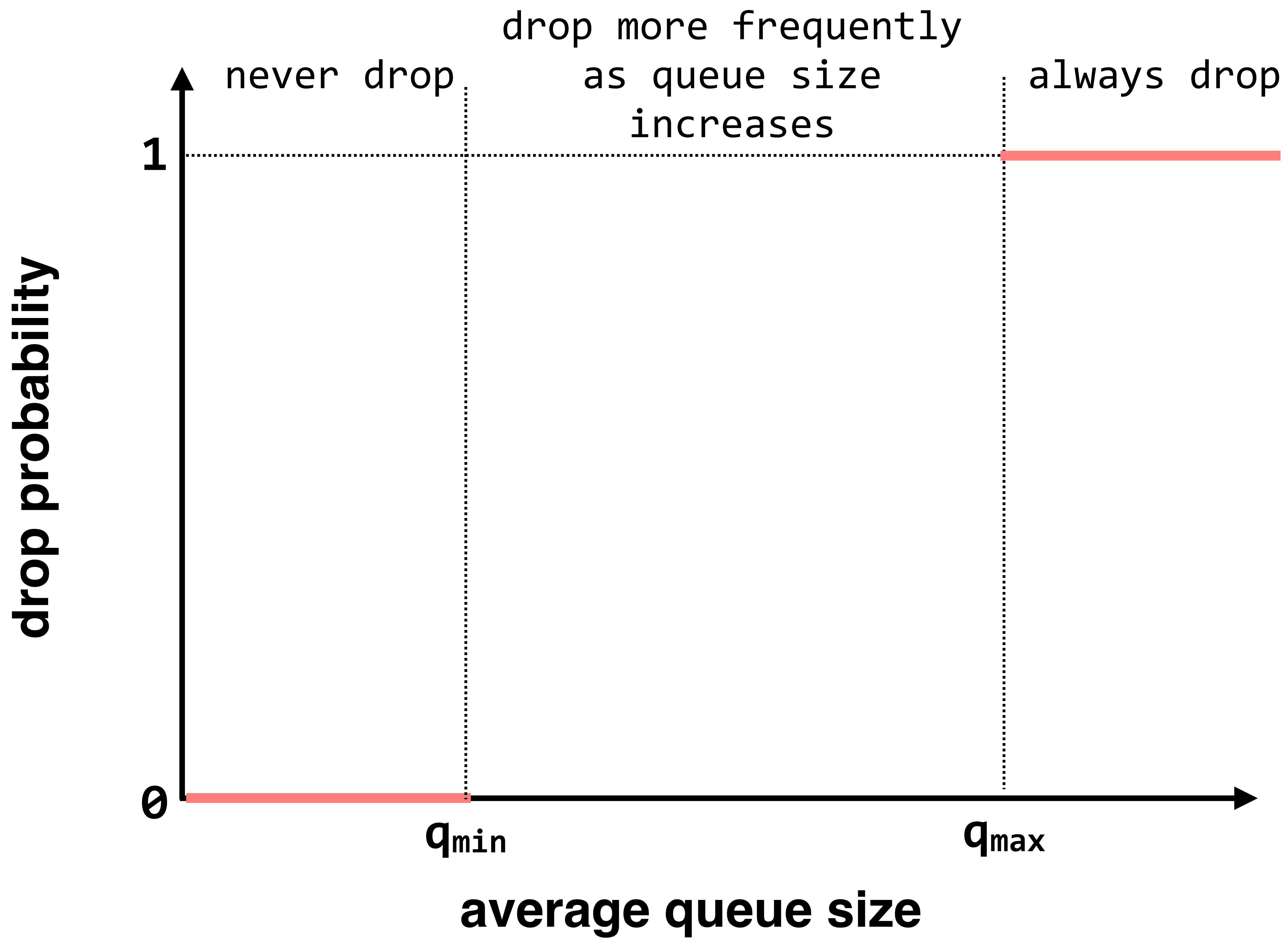
**RED:** drop packets before the queue is full, with increasing probability as the queue grows.



# **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.

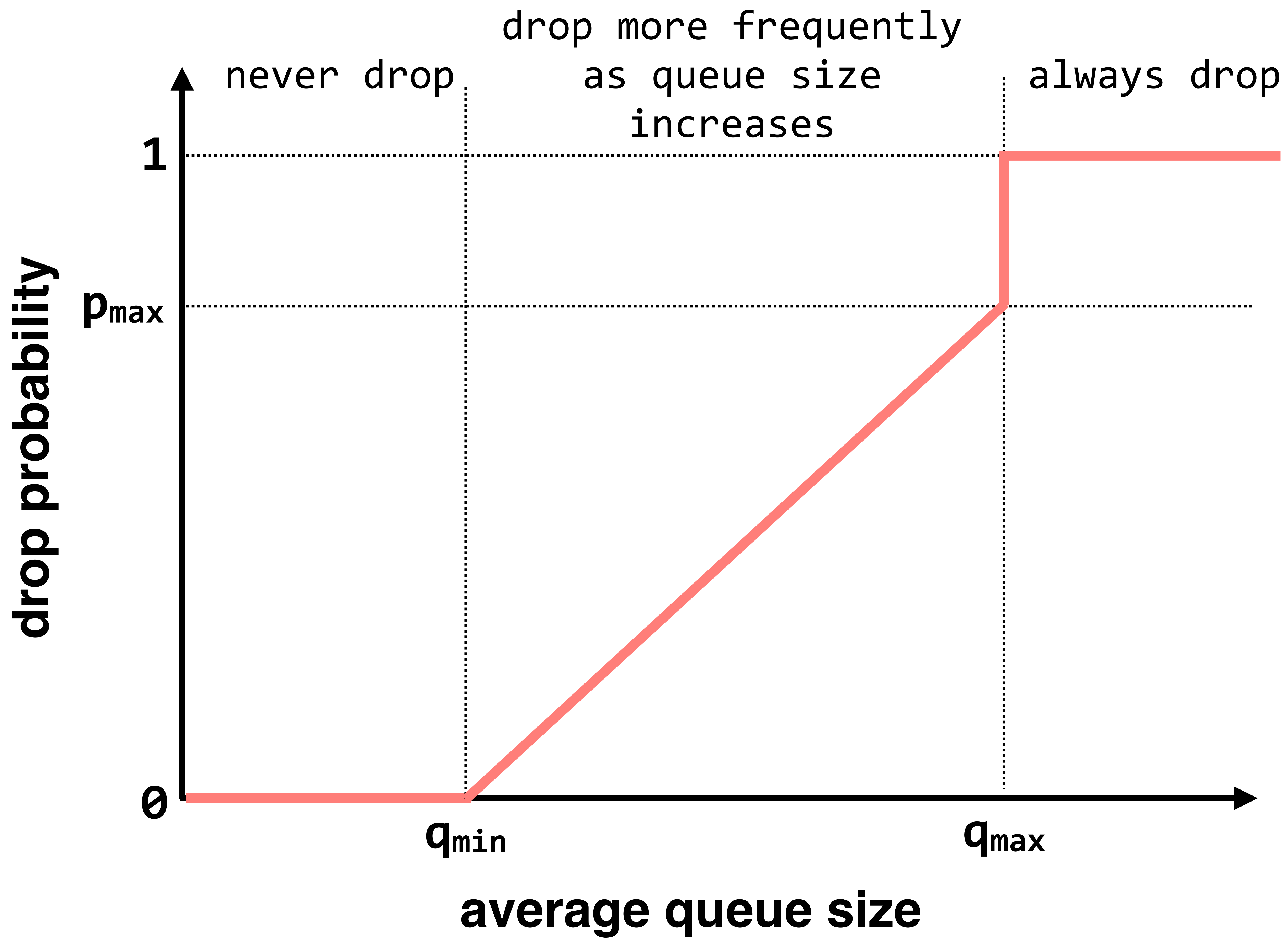
**RED:** drop packets before the queue is full, with increasing probability as the queue grows.



## **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.

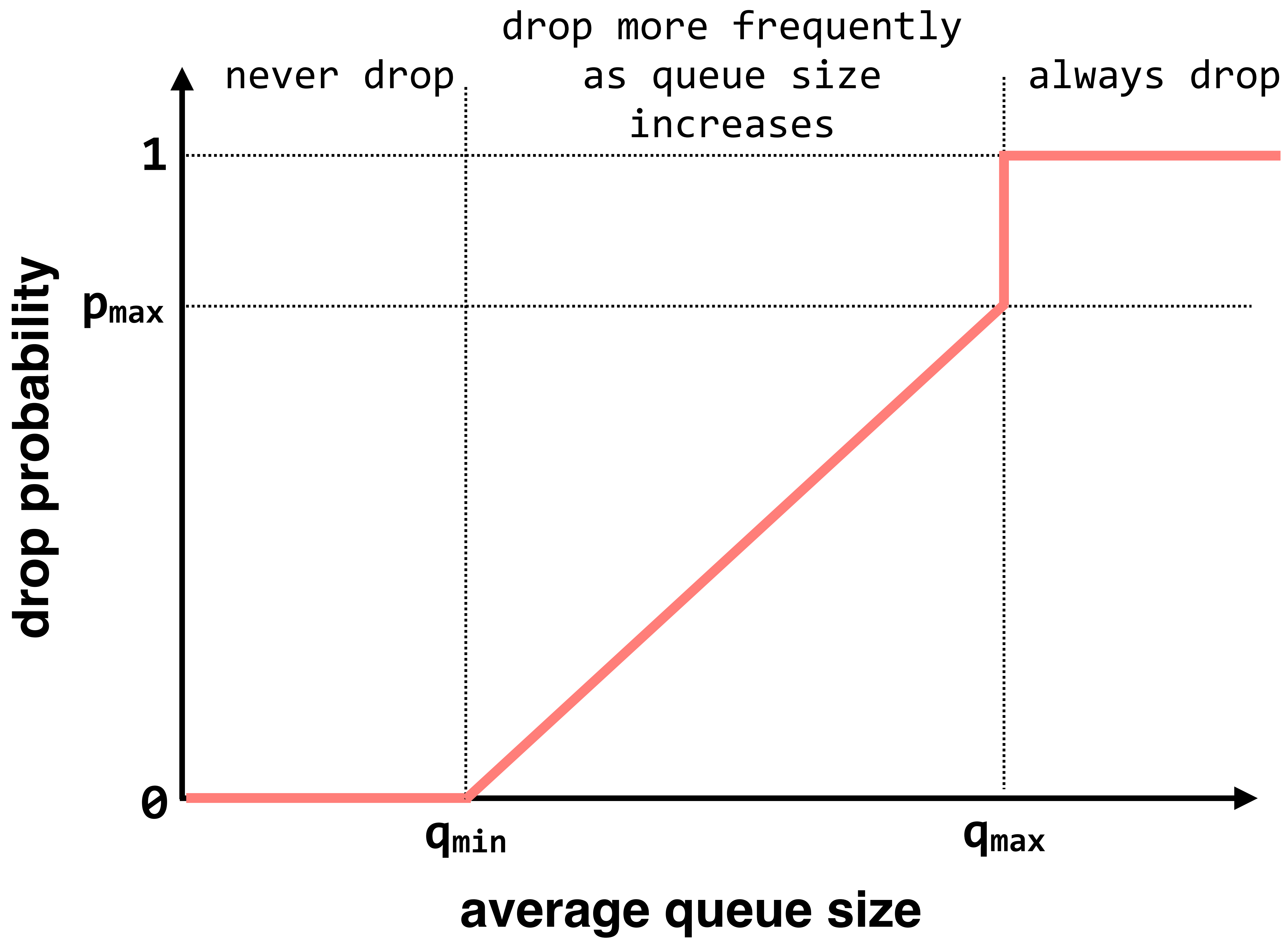
**RED:** drop packets before the queue is full, with increasing probability as the queue grows.



## 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.

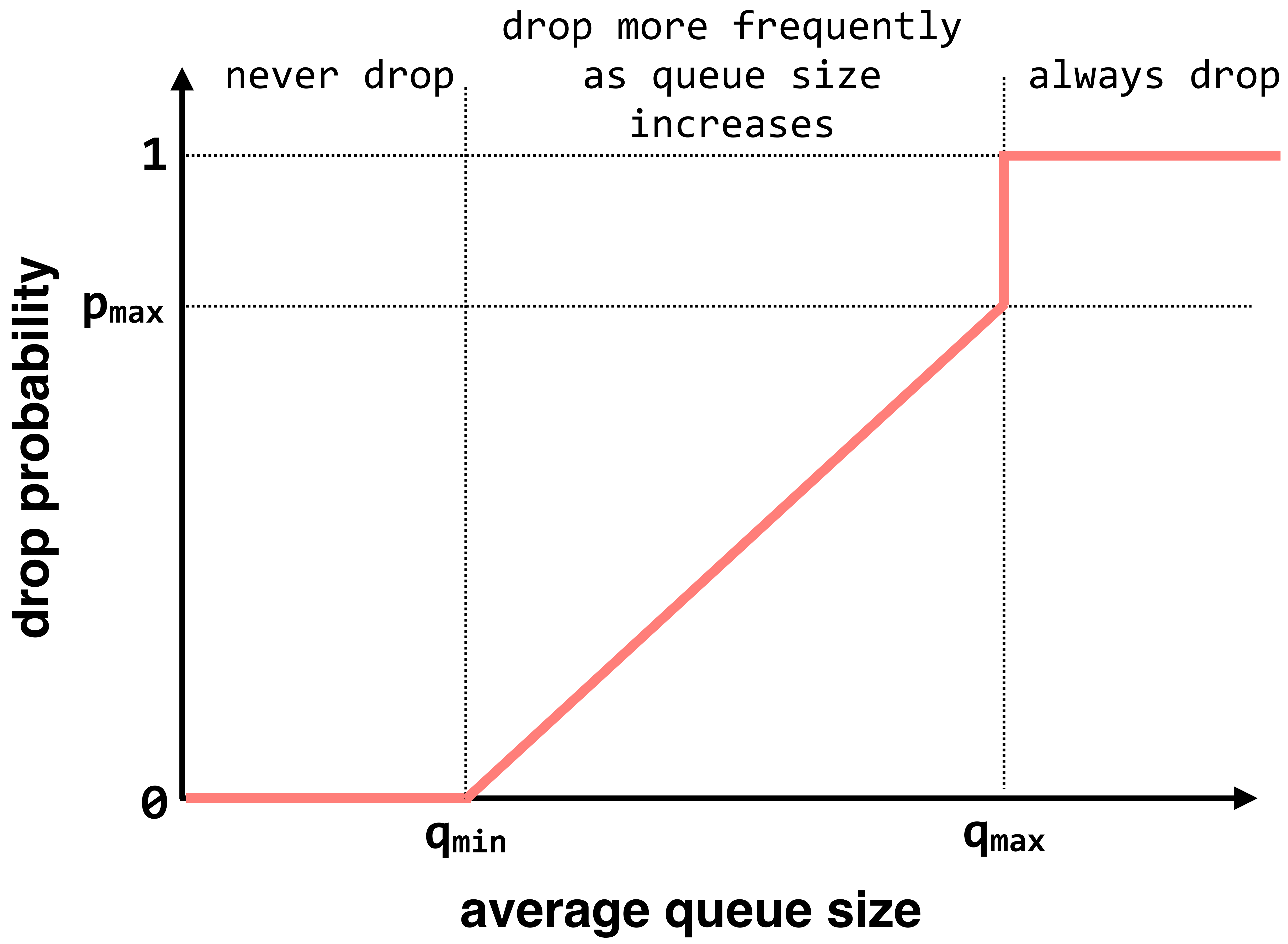
**RED:** drop packets before the queue is full, with increasing probability as the queue grows. prevents queue lengths from oscillating, decreases delay, flows don't synchronize.



# **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.

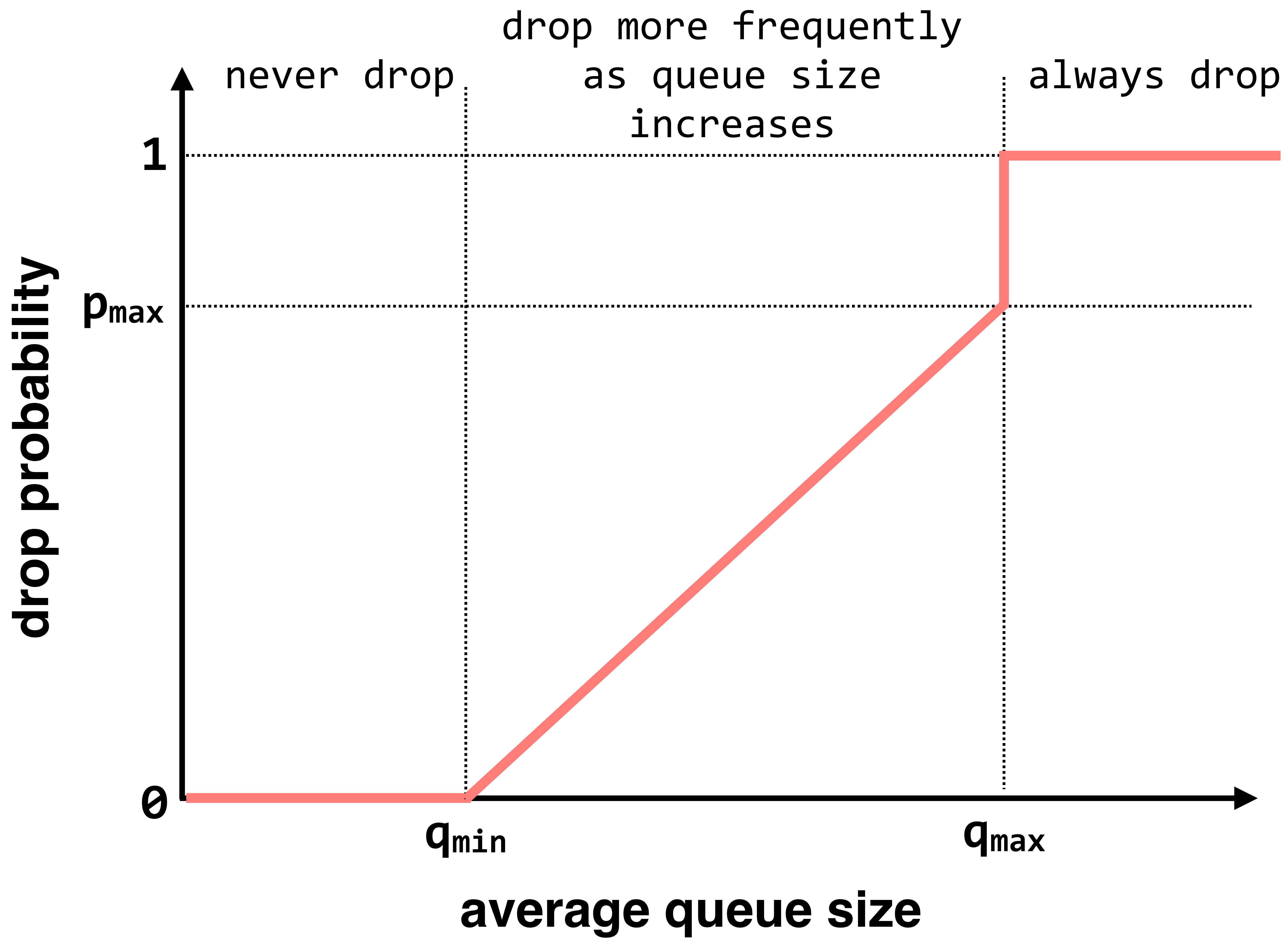
**RED:** drop packets before the queue is full, with increasing probability as the queue grows. prevents queue lengths from oscillating, decreases delay, flows don't synchronize.



# **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.

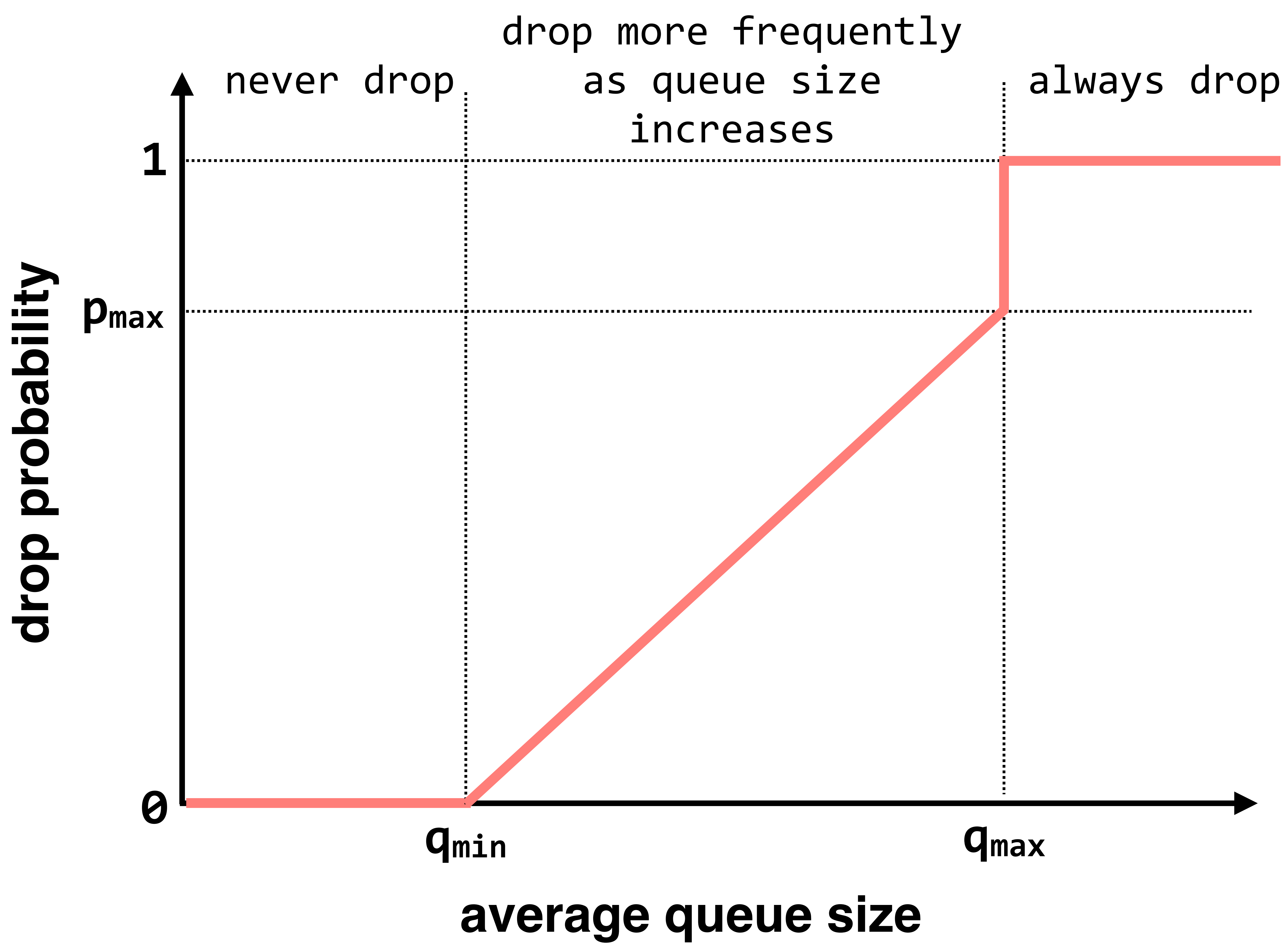
**RED (optionally with ECN):** drop (or mark) packets before the queue is full: with increasing probability as the queue grows. prevents queue lengths from oscillating, decreases delay, flows don't synchronize.



# 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.

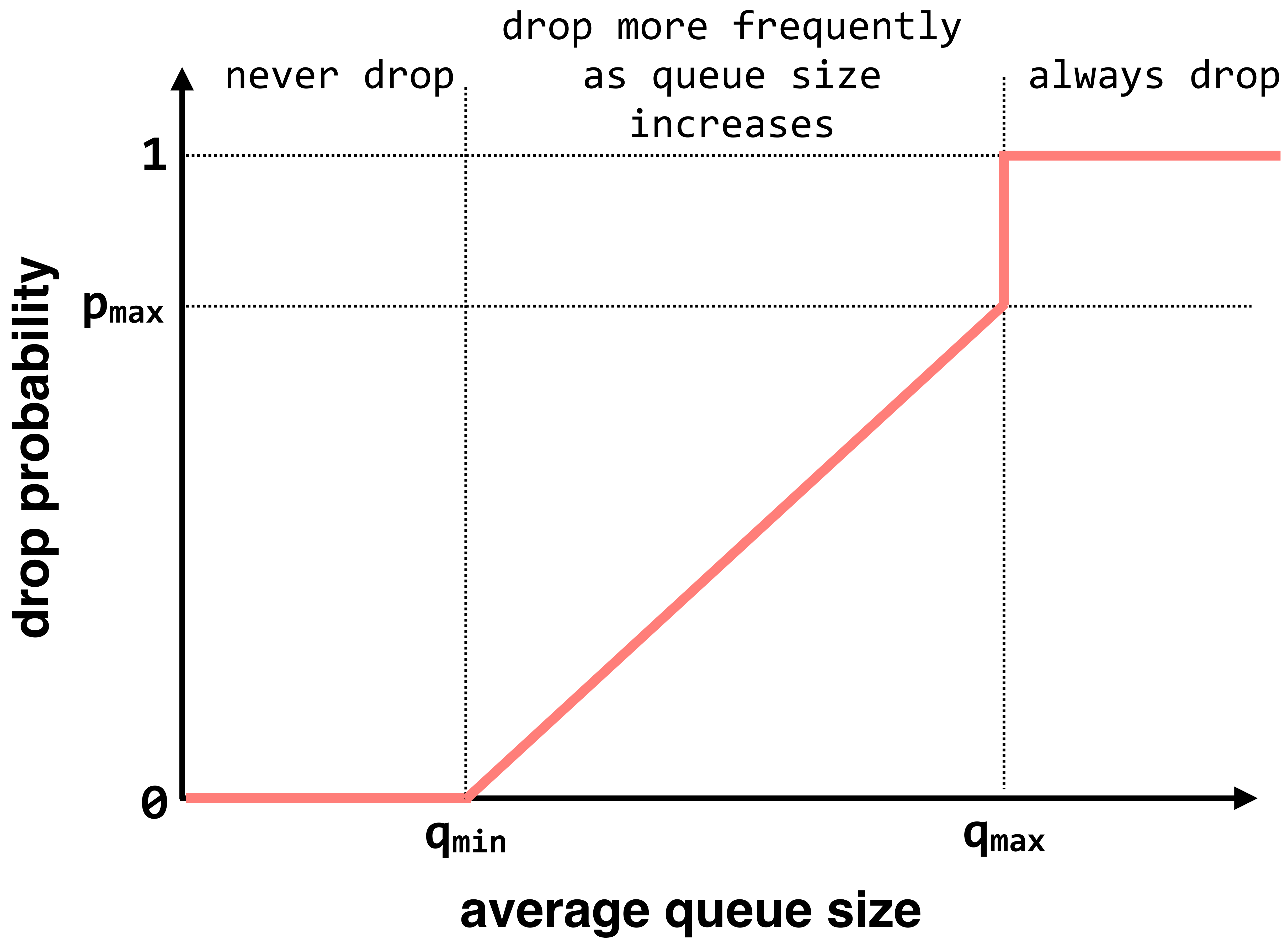
**RED (optionally with ECN):** drop (or mark) packets before the queue is full: with increasing probability as the queue grows. prevents queue lengths from oscillating, decreases delay, flows don't synchronize. *but* complex, hard to pick parameters



# **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 (optionally with ECN):** 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, 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?

## **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). 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 (can extend this idea to multiple queues/types of traffic). 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:`**

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

**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

deficit round robin also doesn't require a mean packet size, which is another good thing

```
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
```

**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

deficit round robin also doesn't require a mean packet size, which is another good thing

```
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

# in-network resource management

| type of management | what does this type of management allow a switch to do | example protocols | how the protocol works | pros/cons? |
|---|---|---|---|---|
| **Queue Management** | signal congestion, potentially before queues are full | `DropTail` | drop packets when the queue is full | simple, but queues get full (among other problems) |
| | | `RED (+ECN)` | drop (or mark) packets before the queue is full | gets TCP senders to react early, but complex, not clear how well it controls delay |
| | | `CoDel, PIE` | drop (or mark) packets before the queue is full (but in different ways than RED) | more robust than RED |
| **Delay-based Scheduling** | prioritize latency-sensitive traffic | `Priority Queueing` | serve some queues before others | prioritized queues can starve out the others |
| **Bandwidth-based Scheduling** | enforce (weighted) fairness among different types of traffic | `Round-robin` | try to give each type of traffic an equal share of bandwidth | can't handle variable packet sizes |
| | | `Weighted Round-robin` | round robin, but incorporate average packet size | have to calculate average packet size |
| | | `Deficit Round-robin` | round robin, but do a better job with packet sizes | pretty good! |

# in-network resource management

| type of management | what does this type of management allow a switch to do | example protocols | how the protocol works | pros/cons? |
|---|---|---|---|---|
| **Queue Management** | signal congestion, potentially before queues are full | `DropTail` | drop packets when the queue is full | simple, but queues get full (among other problems) |
| | | `RED (+ECN)` | drop (or mark) packets before the queue is full | gets TCP senders to react early, but complex, not clear how well it controls delay |
| | | `CoDel, PIE` | drop (or mark) packets before the queue is full (but in different ways than RED) | more robust than RED |
| **Delay-based Scheduling** | prioritize latency-sensitive traffic | `Priority Queueing` | serve some queues before others | prioritized queues can starve out the others |
| **Bandwidth-based Scheduling** | enforce (weighted) fairness among different types of traffic | `Round-robin` | try to give each type of traffic an equal share of bandwidth | can't handle variable packet sizes |
| | | `Weighted Round-robin` | round robin, but incorporate average packet size | have to calculate average packet size |
| | | `Deficit Round-robin` | round robin, but do a better job with packet sizes | pretty good! |

we didn't cover the highlighted protocols; this is just to give you a sense that there are other algorithms out there — in particular, in the case of AQM, there are algorithms that are more robust than RED

# in-network resource management

| type of management | what does this type of management allow a switch to do | example protocols | how the protocol works | pros/cons? |
|---|---|---|---|---|
| **Queue Management** | signal congestion, potentially before queues are full | `DropTail` | drop packets when the queue is full | simple, but queues get full (among other problems) |
| | | `RED (+ECN)` | drop (or mark) packets before the queue is full | gets TCP senders to react early, but complex, not clear how well it controls delay |
| | | `CoDel, PIE` | drop (or mark) packets before the queue is full (but in different ways than RED) | more robust than RED |
| **Delay-based Scheduling** | prioritize latency-sensitive traffic | `Priority Queueing` | serve some queues before others | prioritized queues can starve out the others |
| **Bandwidth-based Scheduling** | enforce (weighted) fairness among different types of traffic | `Round-robin` | try to give each type of traffic an equal share of bandwidth | can't handle variable packet sizes |
| | | `Weighted Round-robin` | round robin, but incorporate average packet size | have to calculate average packet size |
| | | `Deficit Round-robin` | round robin, but do a better job with packet sizes | pretty good! |

# in-network resource management

| type of management | what does this type of management allow a switch to do | example protocols | how the protocol works | pros/cons? |
| --- | --- | --- | --- | --- |
| **Queue Management** | signal congestion, potentially before queues are full | `DropTail` | drop packets when the queue is full | simple, but queues get full (among other problems) |
| | | `RED (+ECN)` | drop (or mark) packets before the queue is full | gets TCP senders to react early, but complex, not clear how well it controls delay |
| | | `CoDel, PIE` | drop (or mark) packets before the queue is full (but in different ways than RED) | more robust than RED |
| **Delay-based Scheduling** | prioritize latency-sensitive traffic | `Priority Queueing` | serve some queues before others | prioritized queues can starve out the others |
| **Bandwidth-based Scheduling** | enforce (weighted) fairness among different types of traffic | `Round-robin` | try to give each type of traffic an equal share of bandwidth | can't handle variable packet sizes |
| | | `Weighted Round-robin` | round robin, but incorporate average packet size | have to calculate average packet size |
| | | `Deficit Round-robin` | round robin, but do a better job with packet sizes | pretty good! |

**is in-network resource management a good idea on the Internet?**

# 6.1800 in the news

## *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.

Sally Floyd. "Her work on congestion control," a colleague said, helped keep the internet "working for everyone." Carole Leita

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."

https://www.nytimes.com/2019/09/04/science/sally-floyd-dead.html

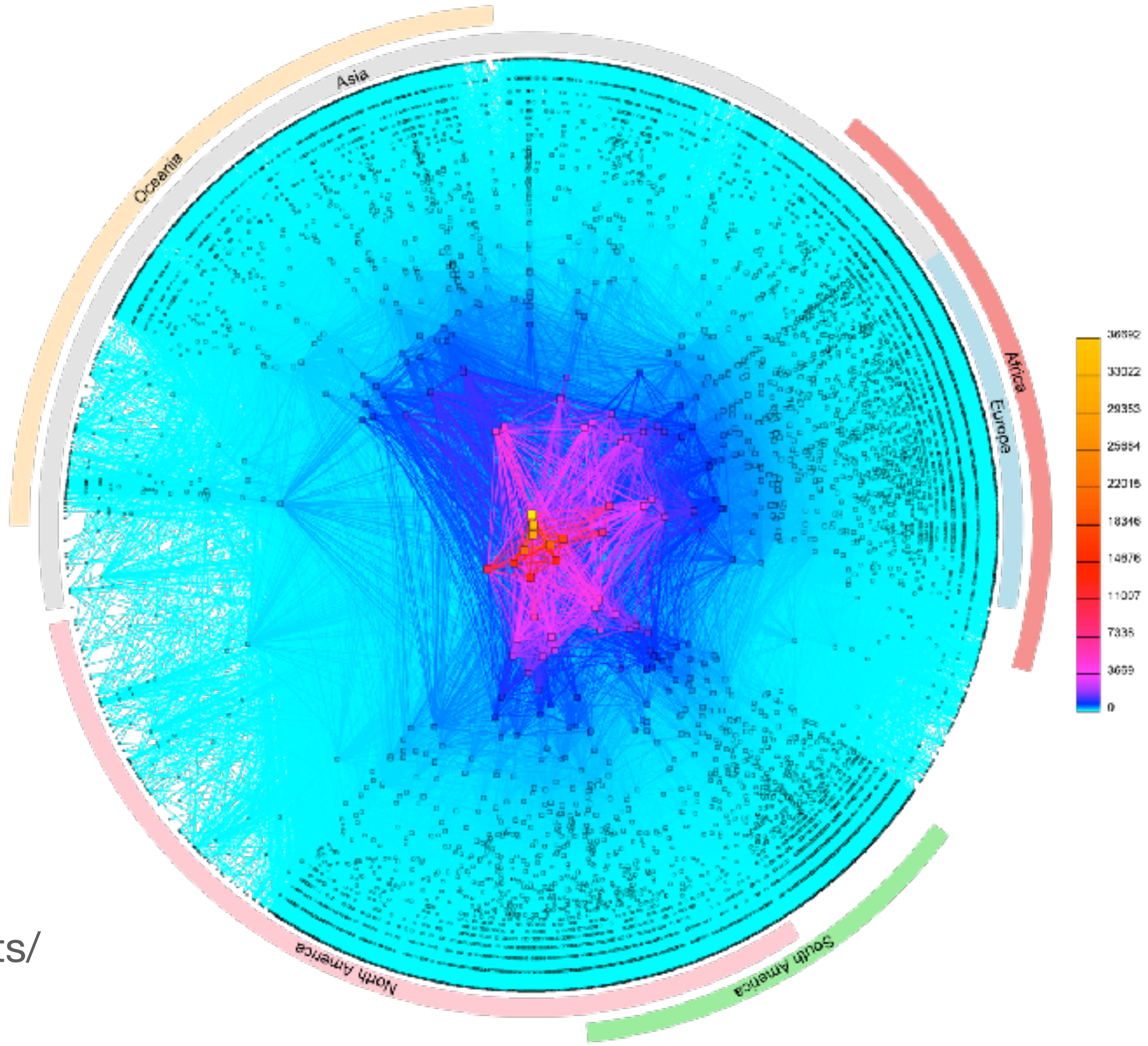
1970s: ARPAnet — hosts.txt

1978: flexibility and layering — distance-vector routing

early 80s: growth → change — TCP, UDP; OSPF, EGP, DNS

late 80s: growth → problems — congestion collapse (which led to congestion control); policy routing; CIDR

1993: commercialization




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

| | |
|---|---|
| **application** | the things that actually generate traffic |
| **transport** | sharing the network, reliability (or not)<br>*examples: TCP, UDP* |
| **network** | naming, addressing, routing<br>*examples: IP* |
| **link** | communication between two directly-connected nodes<br>*examples: ethernet, bluetooth, 802.11 (wifi)* |

**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**