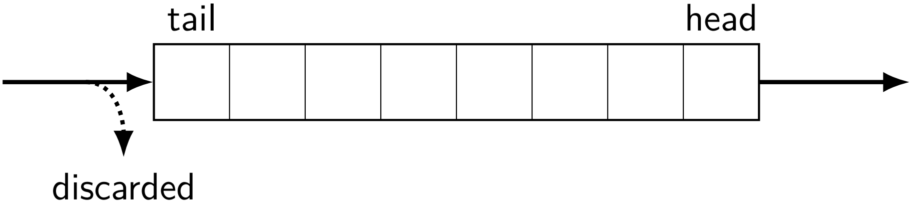
FIFO + drop-tail

scheduling policy: first-in first-out (FIFO)

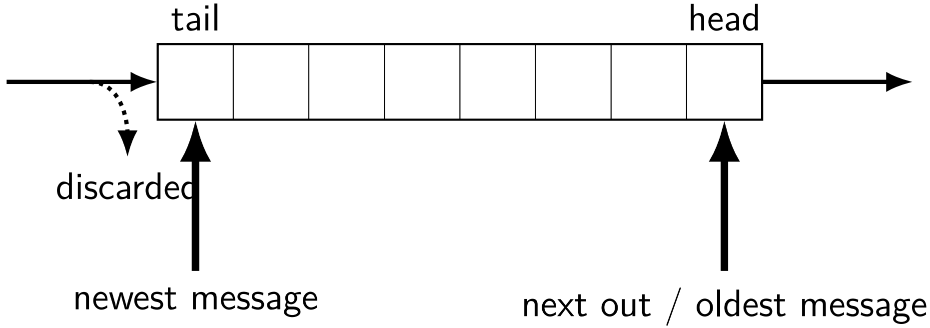
drop policy: drop tail



FIFO + drop-tail

scheduling policy: first-in first-out (FIFO)

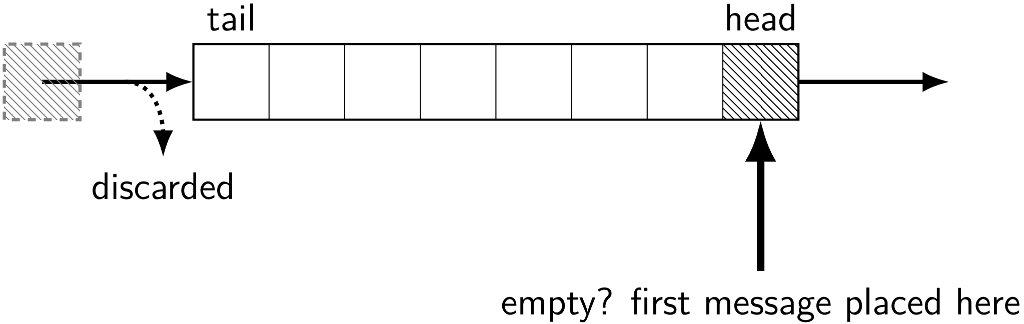
drop policy: drop tail



FIFO + drop-tail

scheduling policy: first-in first-out (FIFO)

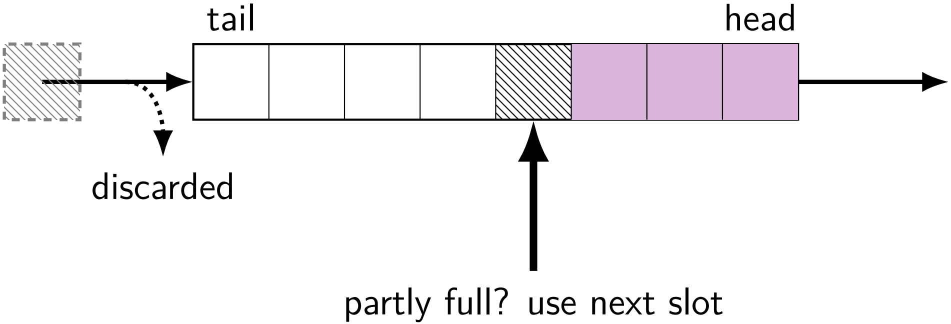
drop policy: drop tail



FIFO + drop-tail

scheduling policy: first-in first-out (FIFO)

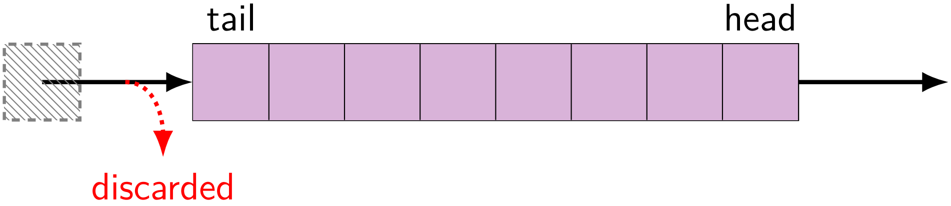
drop policy: drop tail



FIFO + drop-tail

scheduling policy: first-in first-out (FIFO)

drop policy: drop tail



full? discard new packets

queue scheduling

"what goes out on network first?"

simple policy: first-in, first-out (FIFO)

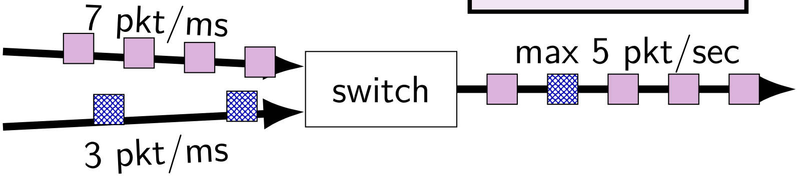
FIFO minimizes per-packet average latency, but...

doesn't distinguish between different flows

unfair dequeue/drop

7 pkt/ms into switch

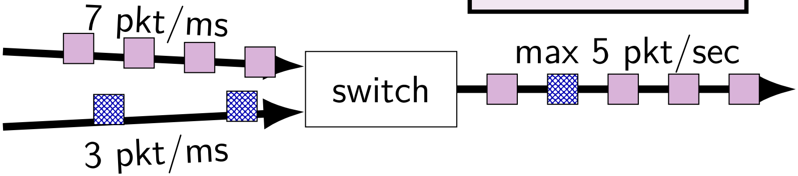
3.5 pkt/ms out
50% packet loss



unfair dequeue/drop

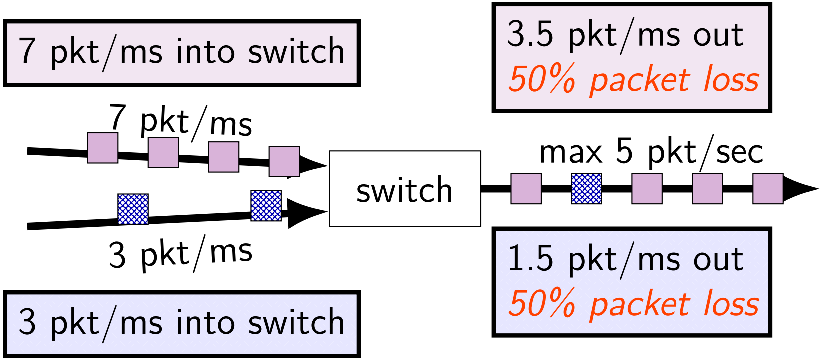
7 pkt/ms into switch

3.5 pkt/ms out
50% packet loss



intuitive notion of 'fair' would mean:
top flow gets 2.5 pkts/sec out
bottom flow gets 2.5 pkts/sec out

unfair dequeue/drop

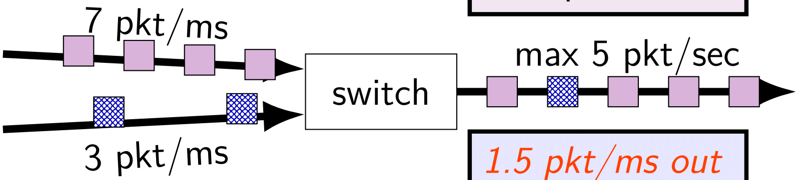


FIFO + drop-tail does not distinguish flows
so both flows have same drop rate

unfair dequeue/drop

7 pkt/ms into switch

3.5 pkt/ms out
50% packet loss

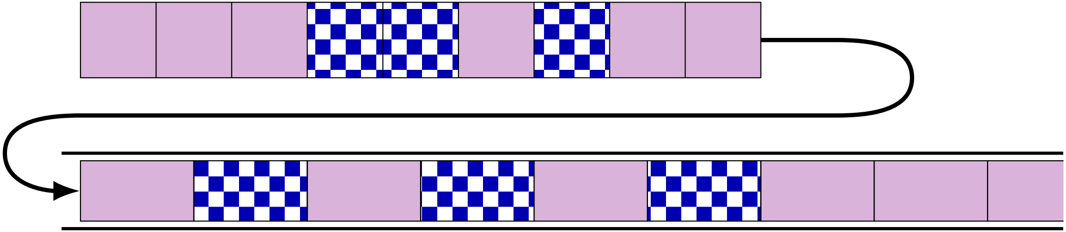


3 pkt/ms into switch

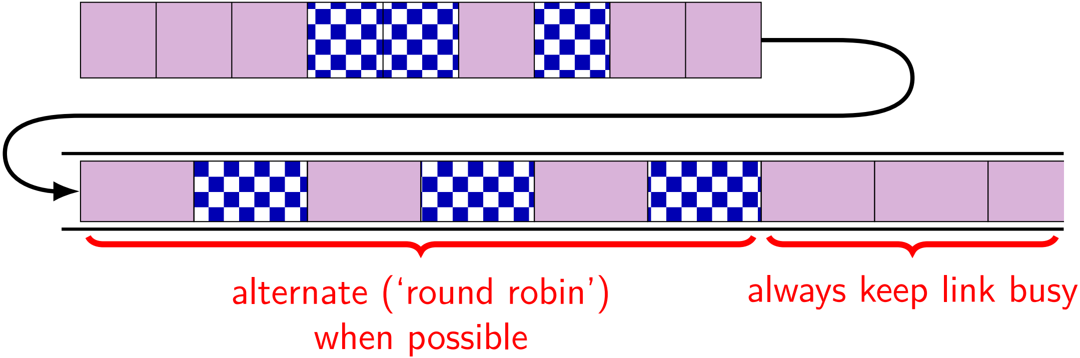
1.5 pkt/ms out
50% packet loss

preserve ratio of input demand
far from even ('intuitive fair') split

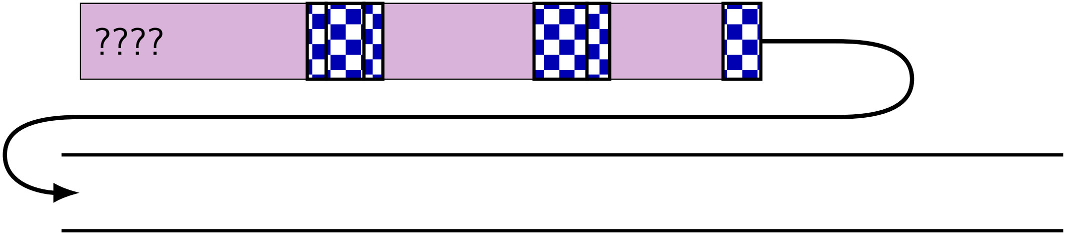
fair (de)queuing



fair (de)queuing



problem 1: variable packet size



just alternating packets doesn't work with variable sizes

need to send more packets from flows with faster packets

problem 2: packets arriving late

time =	queue	
	A1, B1, B2, B3	
0.0	B1, B2, B3	start sending A1
1.0	B1, B2, B3	finish sending A1
1.0	B2, B3	start sending B1
2.0	B2, B3	finish sending B1
2.0	B3	start sending B2
2.1	B3, A2	receive A2
2.9	B3, A2, A3	receive A3
3.0	B3, A2, A3	finish sending B2

problem 2: packets arriving late

time =	queue	
	A1, B1, B2, B3	
0.0	B1, B2, B3	start sending A1
1.0	B1, B2, B3	finish sending A1
1.0	B2, B3	start sending B1
2.0	B2, B3	finish sending B1
2.0	B3	start sending B2
2.1	B3, A2	receive A2
2.9	B3, A2, A3	receive A3
3.0	B3, A2, A3	finish sending B2

flow A missed a turn because packet was just a little late

intuition: should get extra turn to make up for this?

theoretical model: alternating bit

let's say we can split up packets into individual bits

fair sharing:
one bit from A, one bit from B, one bit from A, one bit from B, one bit from A, etc.

preview:

figure out what timing would look like if we could do this

use this timing to decide what packets to send
(the way we can actually send packets)

alternating bit ordering

sending A1 (2 bit), A2 (1 bit), A3 (2 bit), B1 (4 bit), B2 (1 bit)

step	—
1	send A1 bit 1
2	send B1 bit 1
3	send A1 bit 2
4	send B1 bit 2
5	send A2 bit 1
6	send B1 bit 3
7	send A3 bit 1
8	send B1 bit 4
9	send A3 bit 2
10	send B2 bit 1

alternating bit ordering

sending A1 (2 bit), A2 (1 bit), A3 (2 bit), B1 (4 bit), B2 (1 bit)

step	—
1	send A1 bit 1
2	send B1 bit 1
3	send A1 bit 2
4	send B1 bit 2
5	send A2 bit 1
6	send B1 bit 3
7	send A3 bit 1
8	send B1 bit 4
9	send A3 bit 2
10	send B2 bit 1

packet	finish step
A1	3
A2	5
A3	9
B1	8
B2	10

alternating bit ordering

sending A1 (2 bit), A2 (1 bit), A3 (2 bit), B1 (4 bit), B2 (1 bit)

step	—
1	send A1 bit 1
2	send B1 bit 1
3	send A1 bit 2
4	send B1 bit 2
5	send A2 bit 1
6	send B1 bit 3
7	send A3 bit 1
8	send B1 bit 4
9	send A3 bit 2
10	send B2 bit 1

packet	finish step
A1	3
A2	5
A3	9
B1	8
B2	10

sort

packet	finish step
A1	3
A2	5
B1	8
A3	9
B2	10

alternating bit ordering

sending A1 (2 bit), A2 (1 bit), A3 (2 bit), B1 (4 bit), B2 (1 bit)

step	—
1	send A1 bit 1
2	send B1 bit 1
3	send A1 bit 2
4	send B1 bit 2
5	send A2 bit 1
6	send B1 bit 3
7	send A3 bit 1
8	send B1 bit 4
9	send A3 bit 2
10	send B2 bit 1

packet	finish step
A1	3
A2	5
A3	9
B1	8
B2	10

sort

packet	finish step
A1	3
A2	5
B1	8
A3	9
B2	10

using this order

computed the order packets “should” be received in

goal: achieve same order but without bit-by-bit

...by sending packets in desired receive order

full packet and ordering

sending A1 (2 bit), A2 (1 bit), A3 (2 bit), B1 (4 bit), B2 (1 bit)

bit-by-bit time

packet	finish at
A1	3
A2	5
B1	8
A3	9
B2	10

full packet and ordering

sending A1 (2 bit), A2 (1 bit), A3 (2 bit), B1 (4 bit), B2 (1 bit)

bit-by-bit time

packet	finish at
A1	3
A2	5
B1	8
A3	9
B2	10

whole-packet schedule

step=	
1	start sending A1
2	A1's last bit
3	start sending A2
3	A2's last bit
4	start sending B1
7	B1's last bit
8	start sending A3
9	A3's last bit
10	start sending B2
10	B2's last bit

full packet and ordering

sending A1 (2 bit), A2 (1 bit), A3 (2 bit), B1 (4 bit), B2 (1 bit)

bit-by-bit time

packet	finish at
A1	3
A2	5
B1	8
A3	9
B2	10

whole-packet schedule

step=	
1	start sending A1
2	A1's last bit
3	start sending A2
3	A2's last bit
4	start sending B1
7	B1's last bit
8	start sending A3
9	A3's last bit
10	start sending B2
10	B2's last bit

actual time

packet	finish at
A1	2
A2	3
B1	7
A3	9
B2	10

uneven finishing

sending A1 (2 bit), A2 (1 bit), B1 (4 bit), B2 (1 bit), C1 (2 bit)

step	—
1	send A1 bit 1
2	send B1 bit 1
3	send C1 bit 1
4	send A1 bit 2
5	send B1 bit 2
6	send C1 bit 2
7	send A2 bit 1
8	send B1 bit 3
9	send B1 bit 4
10	send B2 bit 1

packet	finish step
A1	4
C1	6
A2	7
B1	9
B2	10

uneven finishing

sending A1 (2 bit), A2 (1 bit), B1 (4 bit), B2 (1 bit), C1 (2 bit)

round	step	—
1	1	send A1 bit 1
1	2	send B1 bit 1
1	3	send C1 bit 1
2	4	send A1 bit 2
2	5	send B1 bit 2
2	6	send C1 bit 2
3	7	send A2 bit 1
3	8	send B1 bit 3
4	9	send B1 bit 4
5	10	send B2 bit 1

packet	finish step	*finish round*
A1	4	*2*
C1	6	*2*
A2	7	*3*
B1	9	*4*
B2	10	*5*

rounds / virtual time

alternating bit:

several rounds where we transmit 1 bit from each pending flow

can think of round number as a weird notion of time

“virtual clock” that advances at variable speed

called *virtual time*

rounds / virtual time

alternating bit:

several rounds where we transmit 1 bit from each pending flow

can think of round number as a weird notion of time

"virtual clock" that advances at variable speed

called *virtual time*

can quickly compute virtual time packet should finish

enough to tell us when to send each packet

computing virtual time

sending A1 (2 bit), A2 (1 bit), B1 (4 bit), B2 (1 bit), C1 (2 bit)

'current' virtual time $= 0$

packet	virtual finish time

virtual time $=$ time for packet alone $+$ max $\{$last finish time from flow, current time$\}$

computing virtual time

sending *A1 (2 bit)*, A2 (1 bit), B1 (4 bit), B2 (1 bit), C1 (2 bit)

'current' virtual time $= 0$

packet	virtual finish time
A1	2 (= current + size)

virtual time = time for packet alone + max {last finish time from flow, current time}

computing virtual time

sending A1 (2 bit), *A2 (1 bit)*, B1 (4 bit), B2 (1 bit), C1 (2 bit)

'current' virtual time = 0

packet	virtual finish time
A1	2 (= current + size)
A2	3 (= A1's time + size)

virtual time = time for packet alone + max {last finish time from flow, current time}

computing virtual time

sending A1 (2 bit), A2 (1 bit), *B1 (4 bit)*, B2 (1 bit), C1 (2 bit)

'current' virtual time = 0

packet	virtual finish time
A1	2 (= current + size)
A2	3 (= A1's time + size)
B1	4 (= current + size)

virtual time = time for packet alone + max {last finish time from flow, current time}

computing virtual time

sending A1 (2 bit), A2 (1 bit), B1 (4 bit), *B2 (1 bit)*, C1 (2 bit)

'current' virtual time $= 0$

packet	virtual finish time
A1	2 (= current + size)
A2	3 (= A1's time + size)
B1	4 (= current + size)
B2	5 (= B1's time + size)

virtual time = time for packet alone + max {last finish time from flow, current time}

computing virtual time

sending A1 (2 bit), A2 (1 bit), B1 (4 bit), B2 (1 bit), *C1 (2 bit)*

'current' virtual time = 0

packet	virtual finish time
A1	2 (= current + size)
A2	3 (= A1's time + size)
B1	4 (= current + size)
B2	5 (= B1's time + size)
C1	2 (= current + size)

virtual time = time for packet alone + max {last finish time from flow, current time}

late packets

sending A1 (2 bit), B1 (2 bit), B2 (2 bit)
and A2 (2 bit) but that arrives between steps 5+6:

vtime	step	—
1	1	send A1 bit 1
1	2	send B1 bit 1
2	3	send A1 bit 2
2	4	send B1 bit 2
3	5	send B2 bit 1 (get A2 here)
4	6	send A2 bit 1
4	7	send B2 bit 1
5	8	send A2 bit 1
6	9	send A2 bit 1

pkt	virtual finish time

late packets

sending A1 (2 bit), B1 (2 bit), B2 (2 bit)
and A2 (2 bit) but that arrives between steps 5+6:

vtime	step	—
1	1	send A1 bit 1
1	2	send B1 bit 1
2	3	send A1 bit 2
2	4	send B1 bit 2
3	5	send B2 bit 1 (get A2 here)
4	6	send A2 bit 1
4	7	send B2 bit 1
5	8	send A2 bit 1
6	9	send A2 bit 1

pkt	virtual finish time
A1	2 (= current + size)

late packets

sending A1 (2 bit), B1 (2 bit), B2 (2 bit)
and A2 (2 bit) but that arrives between steps 5+6:

vtime	step	—
1	1	send A1 bit 1
1	2	send B1 bit 1
2	3	send A1 bit 2
2	4	send B1 bit 2
3	5	send B2 bit 1 (get A2 here)
4	6	send A2 bit 1
4	7	send B2 bit 1
5	8	send A2 bit 1
6	9	send A2 bit 1

pkt	virtual finish time
A1	2 (= current + size)
B1	2 (= current + size)

late packets

sending A1 (2 bit), B1 (2 bit), B2 (2 bit)
and A2 (2 bit) but that arrives between steps 5+6:

vtime	step	—
1	1	send A1 bit 1
1	2	send B1 bit 1
2	3	send A1 bit 2
2	4	send B1 bit 2
3	5	send B2 bit 1 (get A2 here)
4	6	send A2 bit 1
4	7	send B2 bit 1
5	8	send A2 bit 1
6	9	send A2 bit 1

pkt	virtual finish time
A1	2 (= current + size)
B1	2 (= current + size)
B2	4 (= B1's time + size)

late packets

sending A1 (2 bit), B1 (2 bit), B2 (2 bit)
and A2 (2 bit) but that arrives between steps 5+6:

vtime	step	—
1	1	send A1 bit 1
1	2	send B1 bit 1
2	3	send A1 bit 2
2	4	send B1 bit 2
3	5	send B2 bit 1 (get A2 here)
4	6	send A2 bit 1
4	7	send B2 bit 1
5	8	send A2 bit 1
6	9	send A2 bit 1

pkt	virtual finish time
A1	2 (= current + size)
B1	2 (= current + size)
B2	4 (= B1's time + size)
A2	6 (= (current AKA 4) + siz

virtual time algorithm

track:

virtual start time of last sent packet
virtual finish time of last packet for each flow

on receive packet:

compute virtual start time of packet = max of
virtual finish time from last packet in same flow
virtual start time of last sent packet + adjustment for how much sent

compute virtual finish time of packet =
virtual start time + packet size

when sending, send with lowest virtual finish time

controlled fairness

fair queuing says with 2 flows, each gets half bandwidth

maybe we have other goals:

flow A gets 80% of bandwidth

flow B gets 20% of bandwidth

controlled fairness

fair queuing says with 2 flows, each gets half bandwidth

maybe we have other goals:

flow A gets 80% of bandwidth

flow B gets 20% of bandwidth

instead of 1-bit at a time, imagine:

A sends 4 bits per round, B sends 1 bit per round

weighted fair queuing

change fair queuing to add 'weights'

say N bits takes N/weight rounds to send

virtual finish time = virtual start time + size/weight

instead of just + size

upcoming assignment

fixed size packets, but...

part 1: priority queue

part 2: weighted fair queue

- one flow gets twice weight of other
- current version has fixed size packets (undecided if I'll change before out)

exercise

suppose A gets 3 shares, B 1 share

get packets:

A1, A2, A3, A4, A5 (each 2000 bits)

B1 (500 bits), B2 (600 bits), B3 (500 bits), B4 (400 bits)

best order?

late arriving packets (1)

suppose we transmit 1 bit per wallclock time unit:

wallclock time 0: A1 (1000 bit), A2 (500 bit)

wallclock time 800: B1 (250 bit), B2 (250 bit)

want finish virtual times:

A1 (1000), A2 (500)
B1 (800 + 250), B2 (800 + 250)

...because we're on round 800 when B1, B2 received

late arriving packets (2)

suppose we transmit 1 bit per wallclock time unit:

wallclock time 0: A1 (1000 bit), B1 (1100 bit)

wallclock time 800: C1 (500 bit)

if we were going bit-by-bit we'd have transmitted 400 bits of A1 and 400 bits of B1

late arriving packets (2)

suppose we transmit 1 bit per wallclock time unit:

wallclock time 0: A1 (1000 bit), B1 (1100 bit)

wallclock time 800: C1 (500 bit)

if we were going bit-by-bit we'd have transmitted 400 bits of A1 and 400 bits of B1

so we want finish virtual times:

A1 (1000), B1 (1100), C1(*400*+500=900)

late arriving packets (2)

suppose we transmit 1 bit per wallclock time unit:

wallclock time 0: A1 (1000 bit), B1 (1100 bit)

wallclock time 800: C1 (500 bit)

if we were going bit-by-bit we'd have transmitted 400 bits of A1 and 400 bits of B1

so we want finish virtual times:

A1 (1000), B1 (1100), C1(*400*+500=900)

problem: we can't stop transmitting A1 to transmit C1!

tracking round numbers

suppose packets for A, B queued at time 0–100

- virtual time 0–100 (full speed)

then packets for A, B, C queued at time 100-200

- virtual time 100–133 (one-third speed)
- ...and last packet for A finish at time 200...

then packets for B, C queued at time 200-400

- virtual time 133–183 (one-half speed)
- ...and last packet for B finishes at time 400

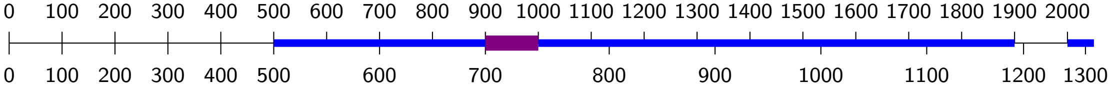
then packets for C queued at time 400-500

- virtual time 183-283 (full speed)

example

example: A1(1000 bit) @ 0, B1(900 bit) @ 500, C1(600 bit) @ 900, C2(100 bit) @ 2000, B2(200 bit) @ 2000

wallclock time	virtual time	active packets (virt time, * = sending)
0	0	A1 (1000*)
500	500	A1 (1000*), B1 (1400)
900	700	A1 (1000*), B1 (1400), C1 (1300)
1000	733	B1 (1400*), C1 (1300)
1900	1183	C1 (1300*)
2000	1283	C1 (1300*), C2 (1400), B2 (1483)
...	...	...



practical weighted-fair-queuing

gets complex to calculate with lots of different flows

so common to use various approximations

example: deficit round robin

- take turns betwen flows
- keep count of 'extra turns' based on size/etc.

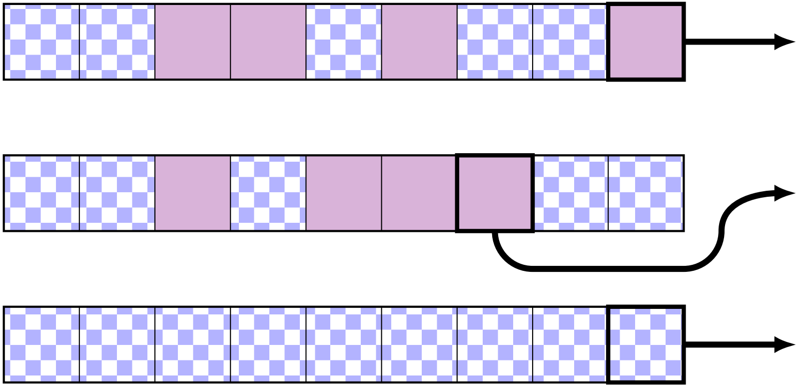
priority queue

assign priorities to flows

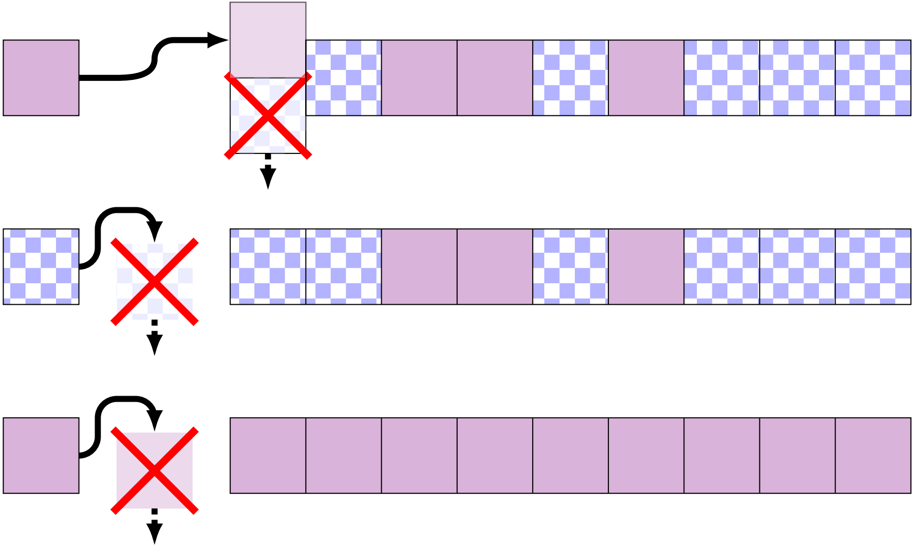
drop packet from lowest priority flow possible

on ties, choose other drop strategy

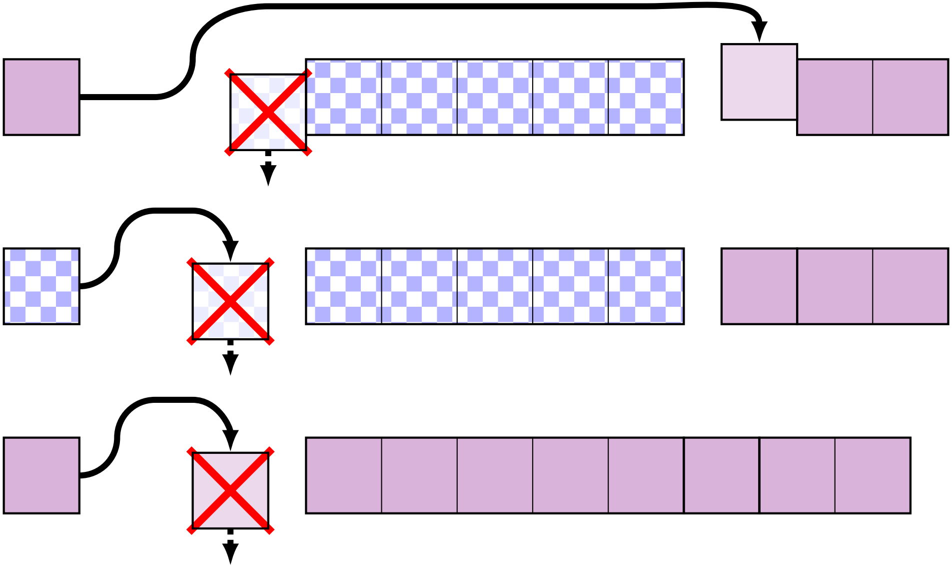
priority-based dequeue



priority-based dropping



priority-based dropping (alt)



priority: all or nothing

if flow A has greater priority than flow B

and both can use full available bandwidth

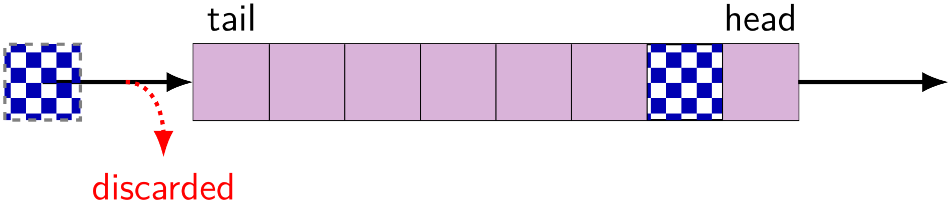
flow A gets full available bandwidth

flow B gets no bandwidth (everything dropped)

drop tail — unfair discard

drop tail is "unfair"

flow not responsible for most usage can be dropped

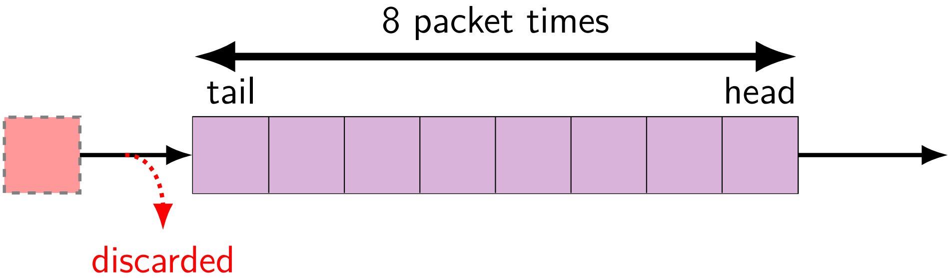


drop tail — long waits

switch now 'knows' flow is going to lose packet

but receiver won't know until packets in queue are sent first

might've been better to drop first packet



early detection

really want to drop earlier packets

- trigger congestion control to reduce send rate faster
- should reduce total number of dropped packets

but don't want to waste queue capacity

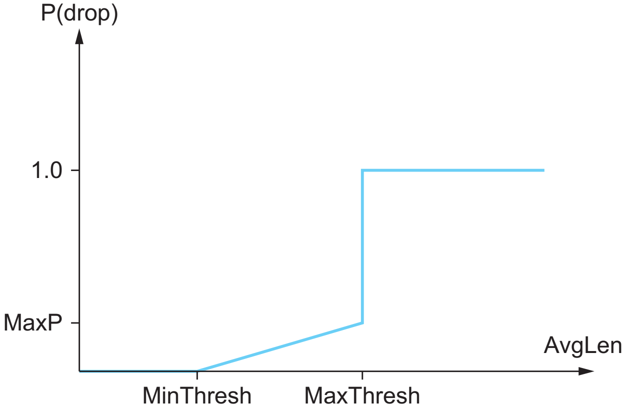
- (otherwise, just use smaller queue)

so drop 'some' packets early

random early detection (RED)

compute queue length

drop with probability based on queue length:

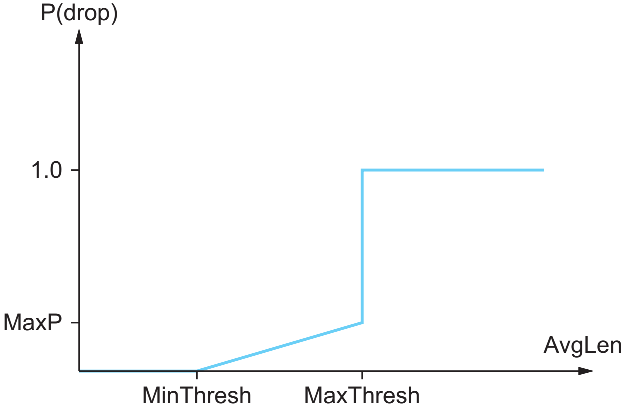


Peterson and Davie, *Computer Networks: A Systems Approach*, Fig 169 (sec 6.4)

random early detection (RED)

compute queue length

drop with probability based on queue length:



Peterson and Davie, *Computer Networks: A Systems Approach*, Fig 169 (sec 6.4)

measuring queue length

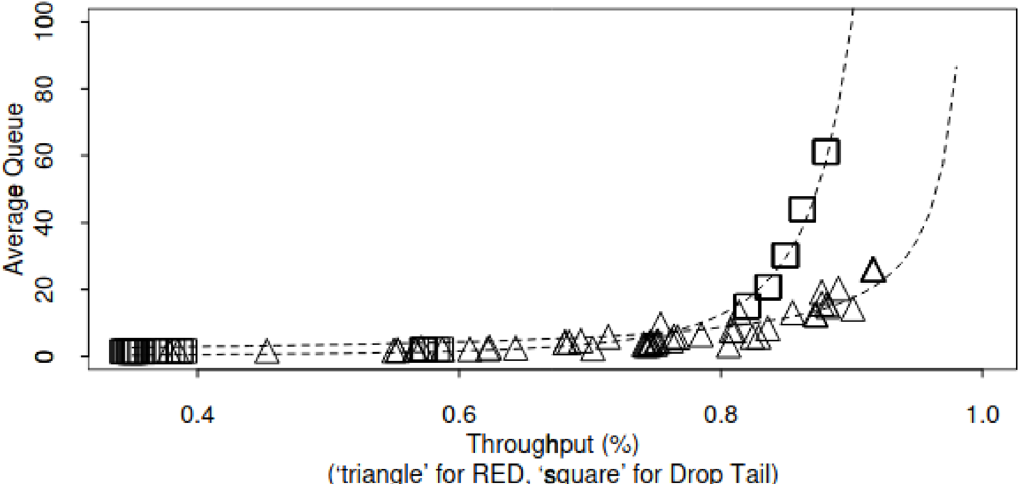
use moving average instead of actual value

reduce sensitivity to short ‘bursts’ of packets

spread packet-dropping ‘signal’ among all active connections

early detection results

From Floyd and Jacobson, "Random Early Detection Gateways for Congestion Avoidance"



active queue management

algorithms that go beyond drop-tail, FIFO called "active queue management"

many variations besides RED, fair queuing

changing random early detection?

some things seem suspicious in RED:

- moving average of queue length limits response time
- how moving average computed can matter
- doesn't choose specific target queue length
- choice of drops doesn't take fairness into account

various proposals to tweak and address these issues

selected other possible goals

weighted fair policy for random early detection

specific bandwidth limits

- if strict priority/weights not good enough
- common algorithm: "token bucket"

backup slides