SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

L18 – Quality of Service

50.012 Networks

Jit Biswas

Cohort 1: TT7&8 (1.409-10)

Cohort 2: TT24&25 (2.503-4)

Introduction

- This lecture:
  - Providing different service to different clients/protocols
  - Maintaining minimum quality for different applications
- Parts of these slides are based on Kurose & Ross Chapter 7

TCP Congestion Handling

General Problem Setting

- Internet and protocols such as TCP/IP are intended to be fair
  - Available resources are shared between all
  - Every incoming packet at a router is served equally
  - No sender or receiver should have an advantage over others

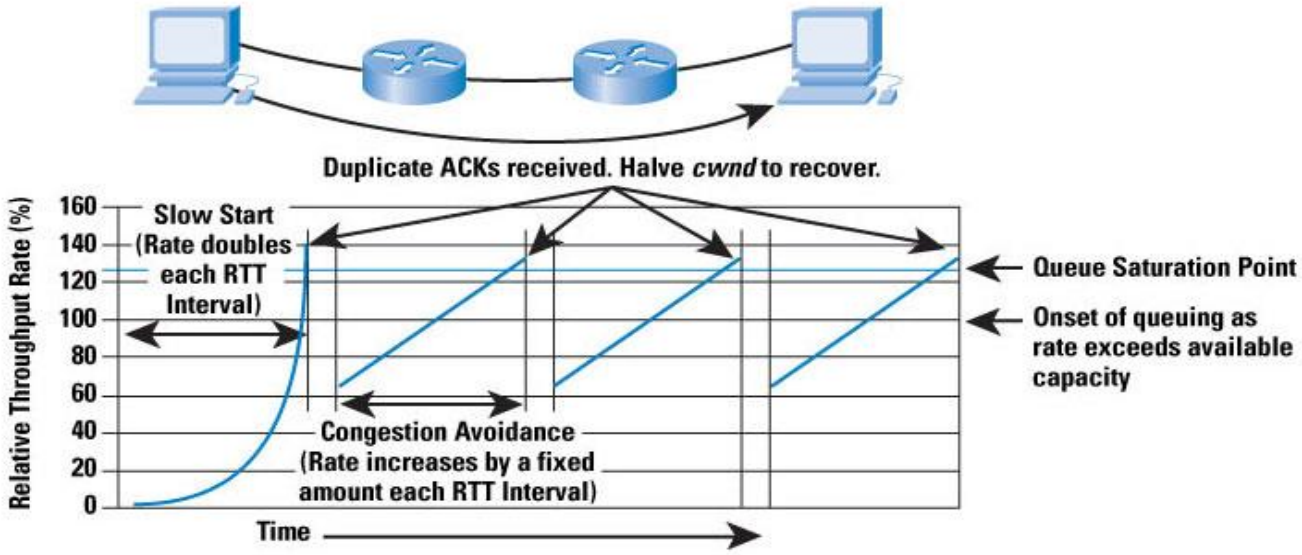
General Problem Setting

- Internet and protocols such as TCP/IP are intended to be fair
  - Available resources are shared between all
  - Every incoming packet at a router is served equally
  - No sender or receiver should have an advantage over others

- Problems
  - Users can get greater share by using parallel TCP streams
  - Delay might be more important than bandwidth for the user

Recap: Competing Sreams in TCP

- In TCP (Reno), congestion window determines how much un-ACK'ed data can be sent
- All TCP streams start in Slow Start mode
  - Window with low value, 1x Maximum Segment Size (536 B)
  - Sender increases window size by 1 MSS per ACK until loss
- When loss occurs, congestion avoidance mode
  - window size is cut in half, increased +1 MSS each RTT
- Every stream is trying to maximize throughput the same way



Source: Cisco

Problem with TCP rate adaptation

- TCP rate adaption means that each TCP stream has changing transmission rate all the time
- What if a specific Application Layer protocol needs reliable minimum rate?
  - HTTP video
  - Skype
  - VOIP
- What if a specific Application Layer protocol needs constant delay?
  - Games

Network support for multimedia

Approach	Granularity	Guarantee	Mechanisms	Complex	Deployed?
Making best of best effort service	All traffic treated equally	None or soft	No network support (all at application)	low	everywhere
Differentiated service	Traffic "class"	None of soft	Packet market, scheduling, policing.	med	some
Per-connection QoS	Per-connection flow	Soft or hard after flow admitted	Packet market, scheduling, policing, call admission	high	little to none

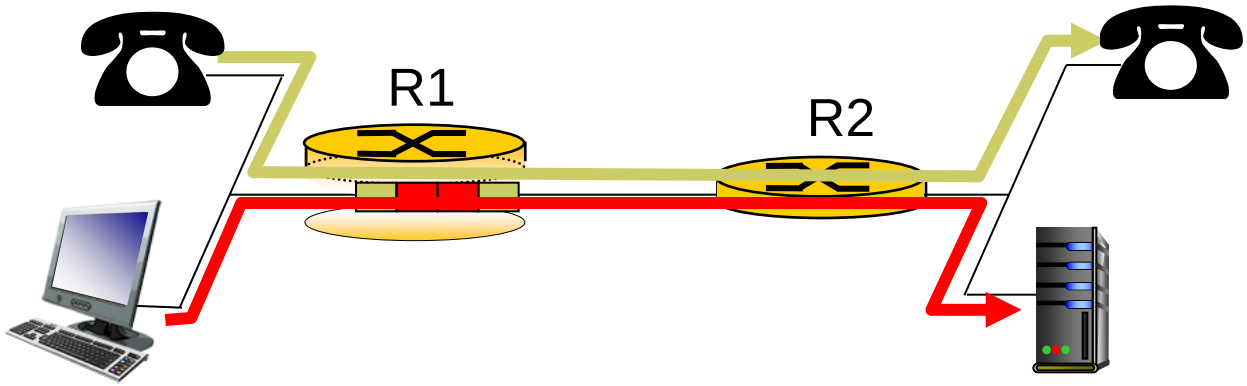
QoS

Quality of Service (QoS)

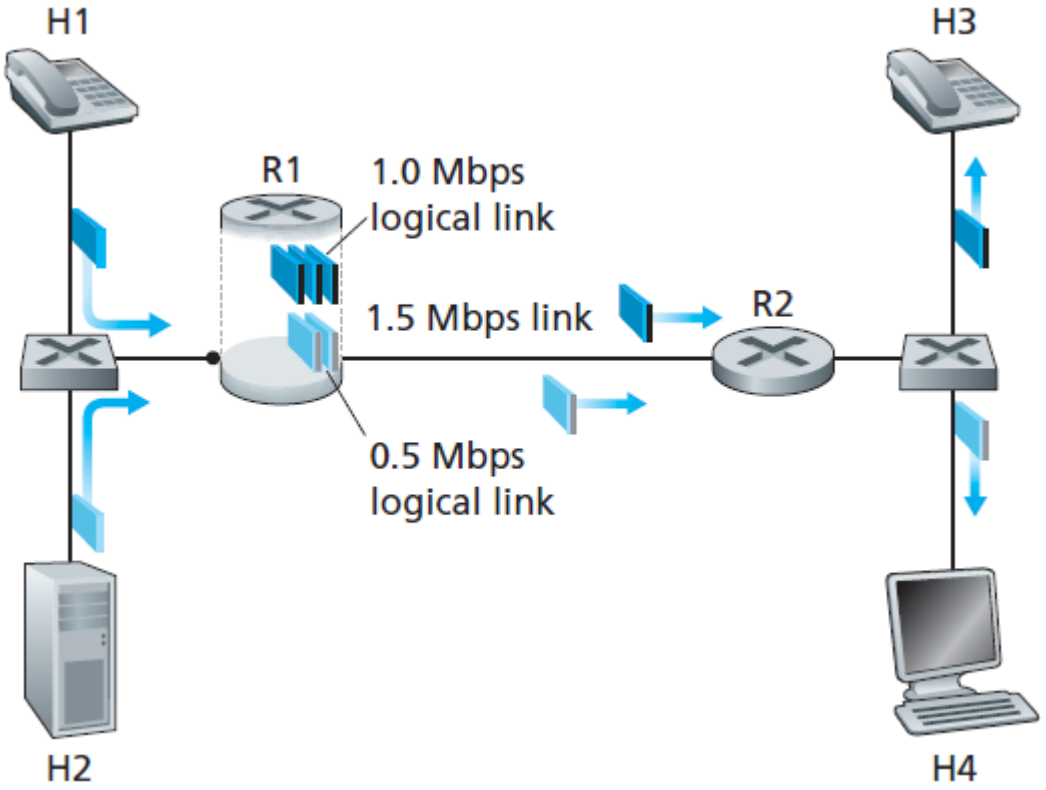
- Internet is modeled on network of peers
  - There is only *one class* of service
  - No guarantees on bandwidth
- *Quality of Service* introduces distinct traffic classes
  - Think *first class* on airplane
  - E.g. packets are using shorter queue
- Intuition: bring back some circuit-switching-style guarantees
- Dynamically adapting traffic conditions is also called *traffic shaping*

Example scenario: mixed HTTP and VoIP

- example: 1Mbps VoIP, HTTP share 1.5 Mbps link.

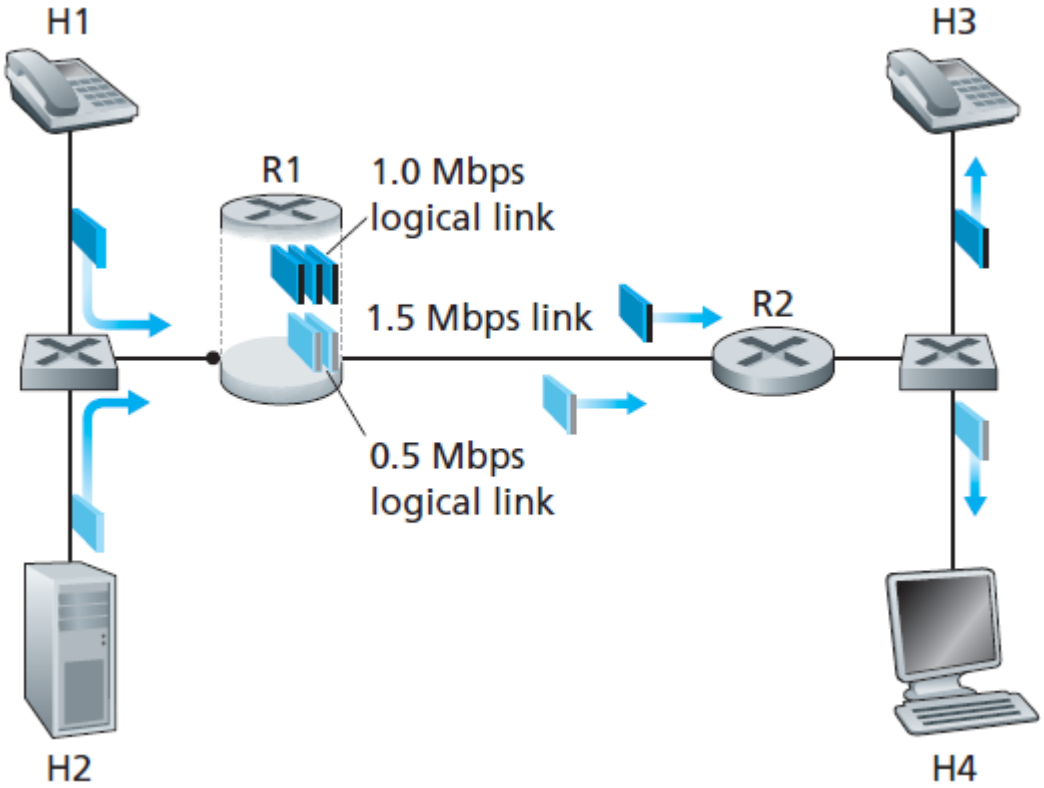


Example Scenario



- VOIP traffic sharing a low BW line with bursty HTTP traffic
- VOIP more sensitive to delay than HTTP
- How to ensure that VOIP can run uninterrupted?

Example Scenario



Insight 1:

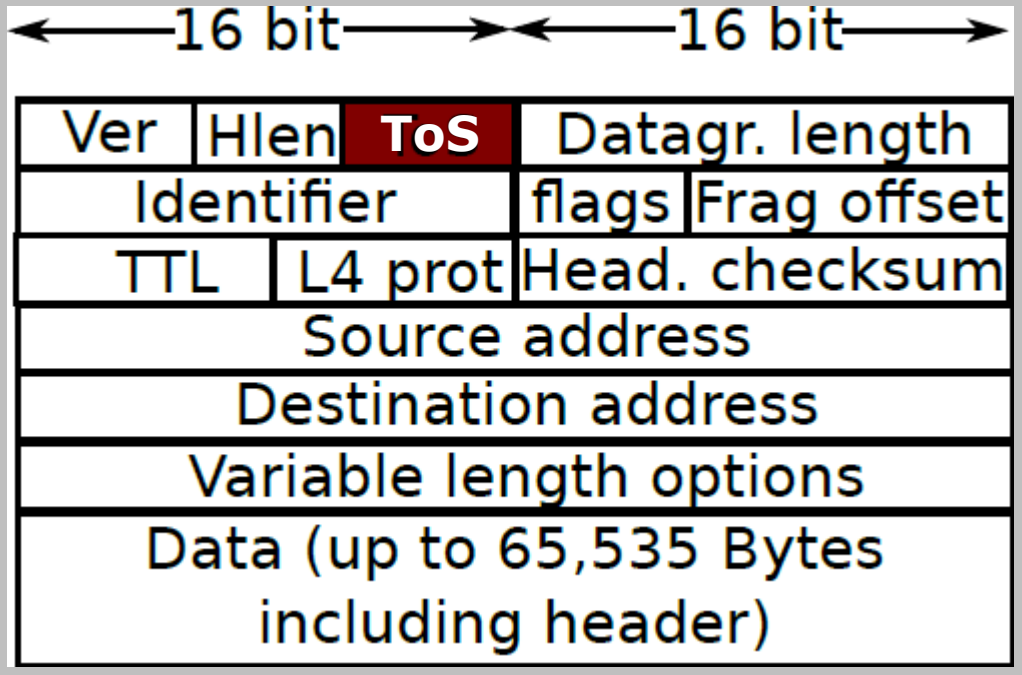
- **Packet marking** allows a router to distinguish among packets belonging to different classes of traffic

Packet Marking

- So far, routers treated each incoming packet equal
- How can the router know that Skype traffic should have priority?
- *VIP* traffic packets need to be marked for routers
- Routers need *policies* to decide to prioritize certain classes
  - We have seen something like this already, where was this?

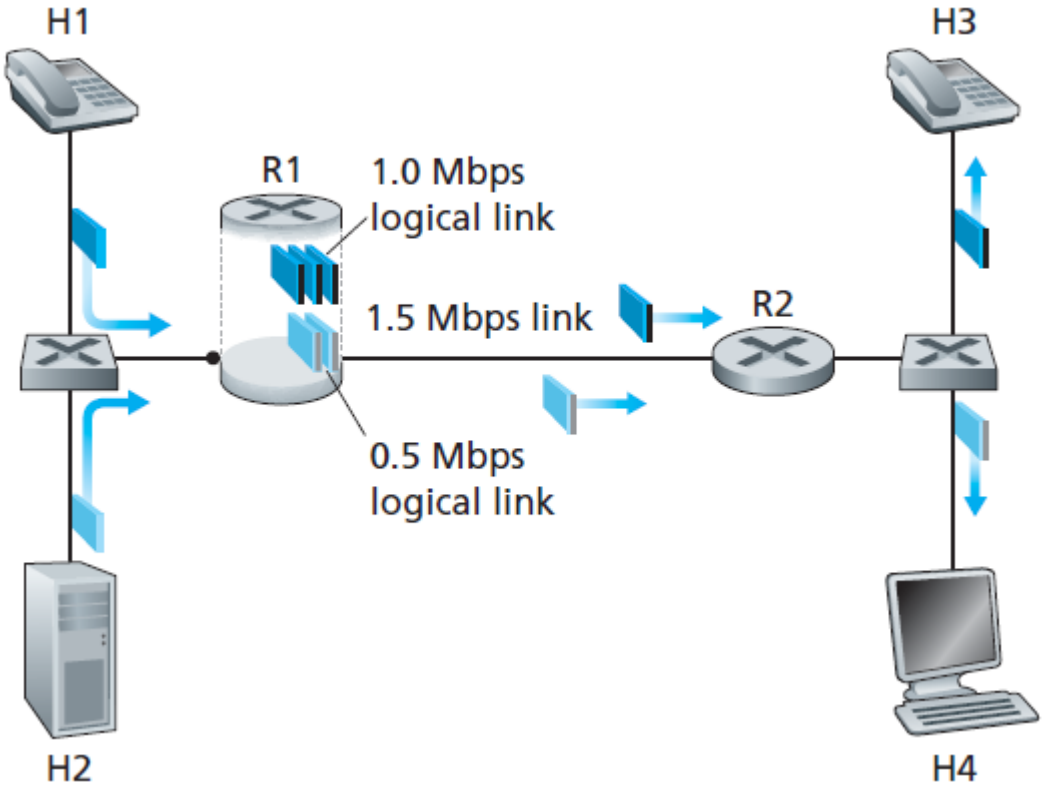
Packet Marking

- So far, routers treated each incoming packet equal
- How can the router know that Skype traffic should have priority?
- *VIP* traffic packets need to be marked for routers
- Routers need *policies* to decide to prioritize certain classes
  - We have seen something like this already, where was this?



The IP packet format

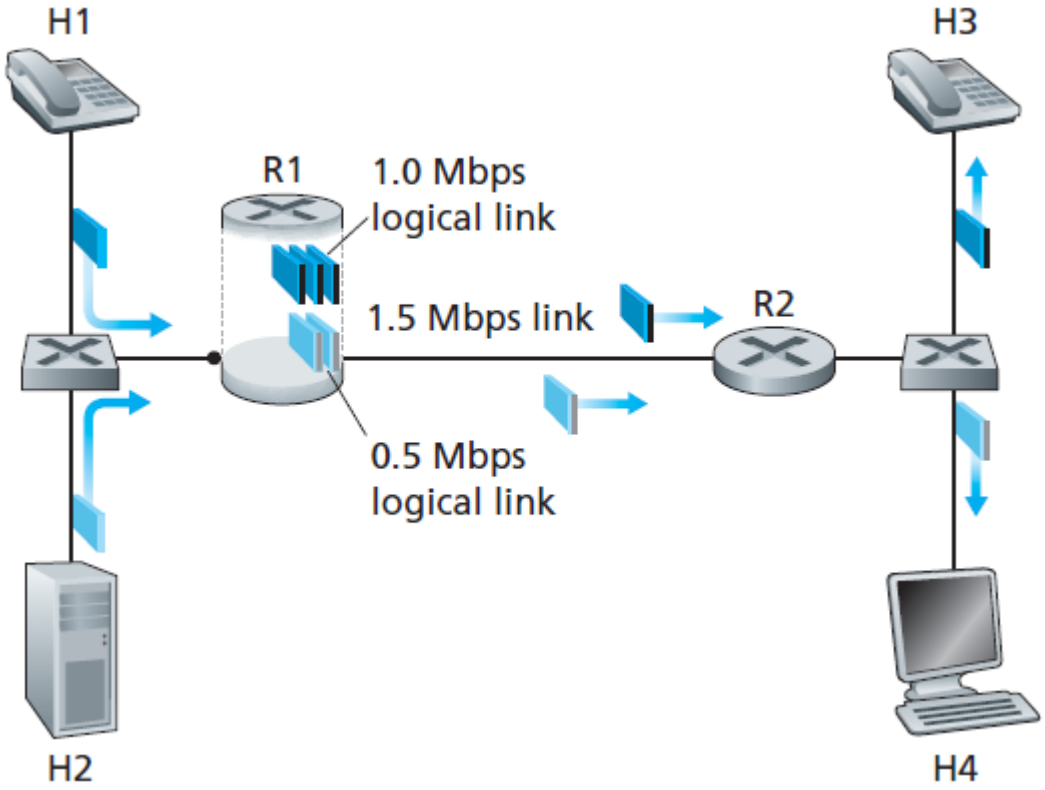
Example Scenario



Insight 2:

- It is desirable to provide a degree of **traffic isolation** among classes so that one class is not adversely affected by another class of traffic that misbehaves

Example Scenario



Insight 3:

- While providing isolation among classes, it is desirable to use resources such as link bandwidths and buffers as efficiently as possible.

Deep Packet Inspection

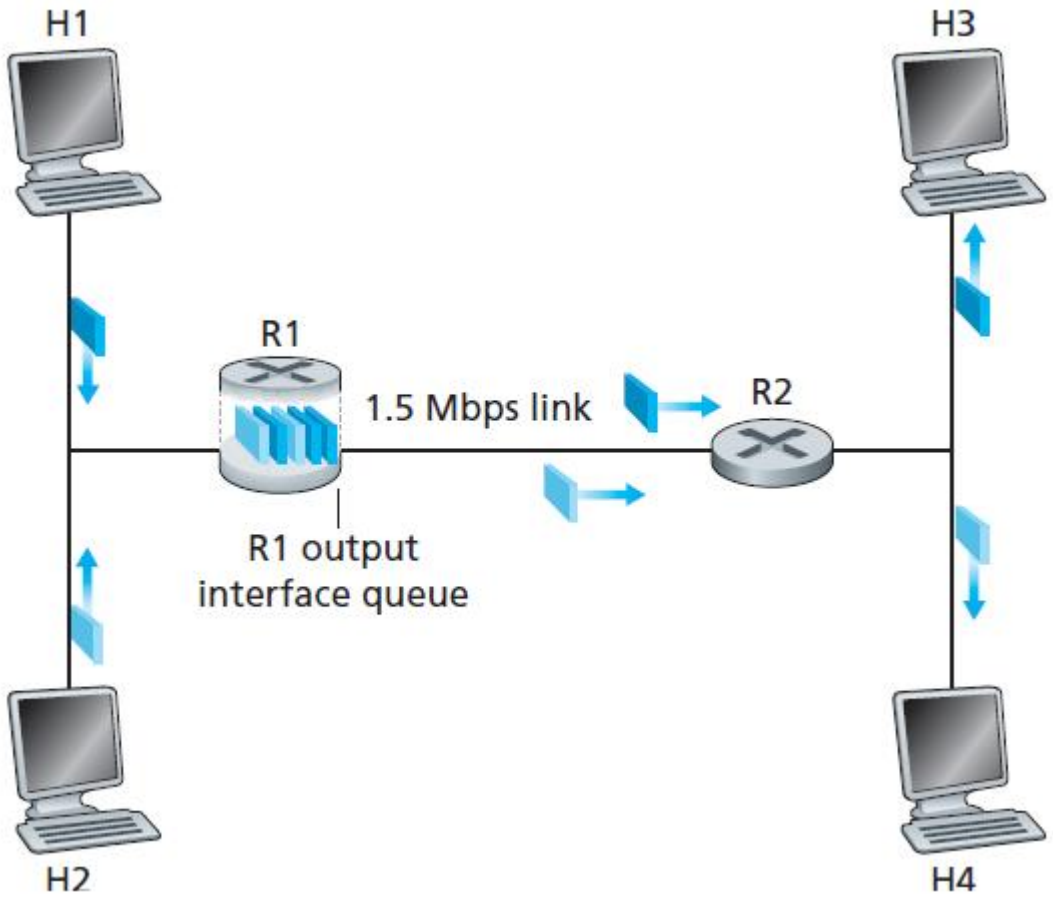
- What if packets are not explicitly marked for QoS?
- Lets just classify the traffic at router
  - But Network layer only contains little information on payload
- Look at application layer data, figure out what is going on
  - Also called *Deep Packet Inspection*
- Problem: High computational effort, additional delay, statefulness
- Ideally, we find some kind of bitmask to classify traffic
  - E.g. "If Bit 12 and 47 of payload is 1, classify as bittorrent"

Scheduling Mechanisms

- How can we reserve 20% of bandwidth for Skype
  - If not used can we still use the remaining bandwidth?
- Can be achieved by *scheduling policies* at routers
  - First-in-First-out: no classes
  - Priority queues: serve VIP first
  - Round-Robin: serve each class once each round
  - Weighted Fair Queuing: RR with weight for each class
- Default scheduling: packets are sent out in order of arrival

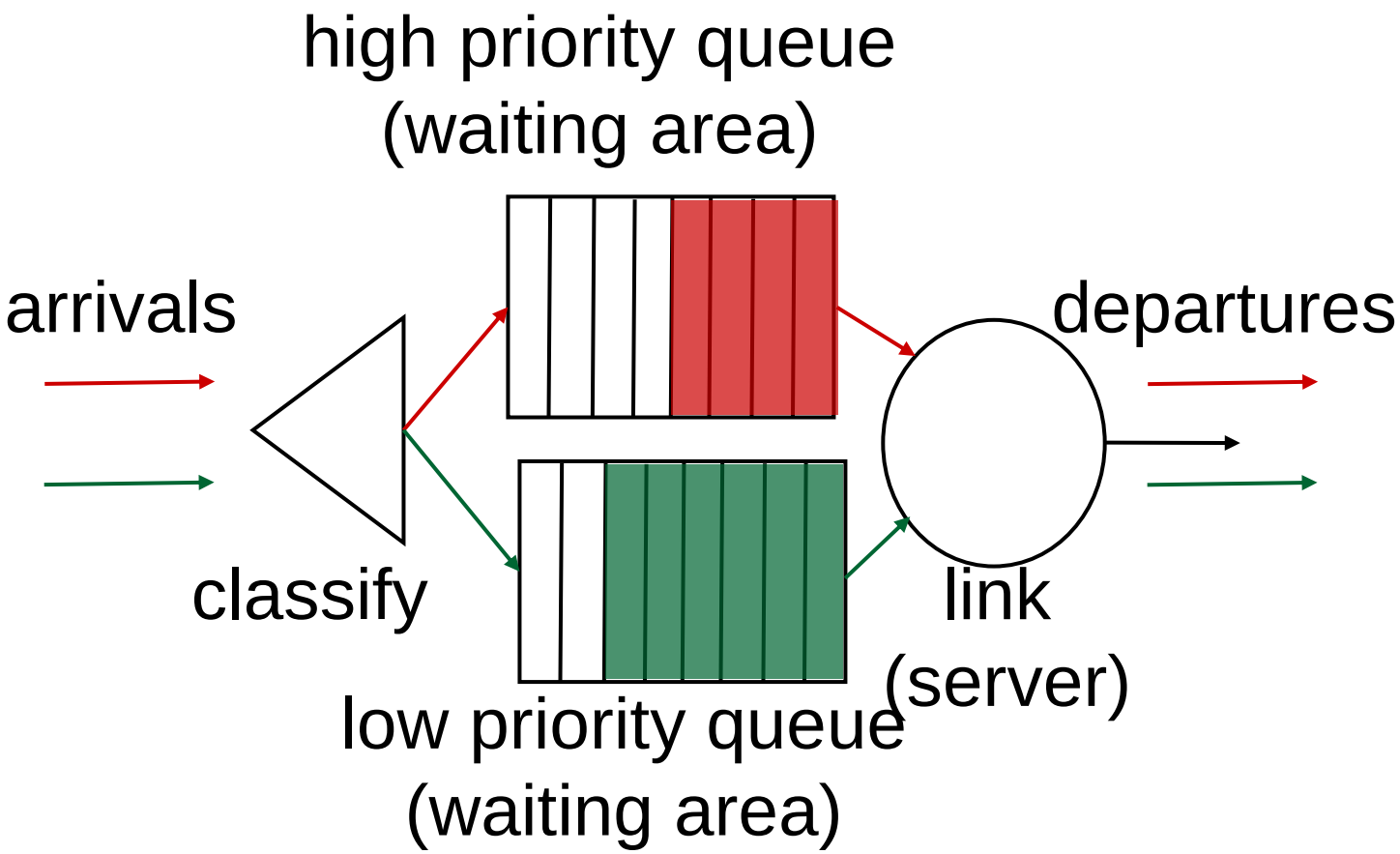
First-in-First-out (FIFO)

- No classes



Priority Queue (PQ)

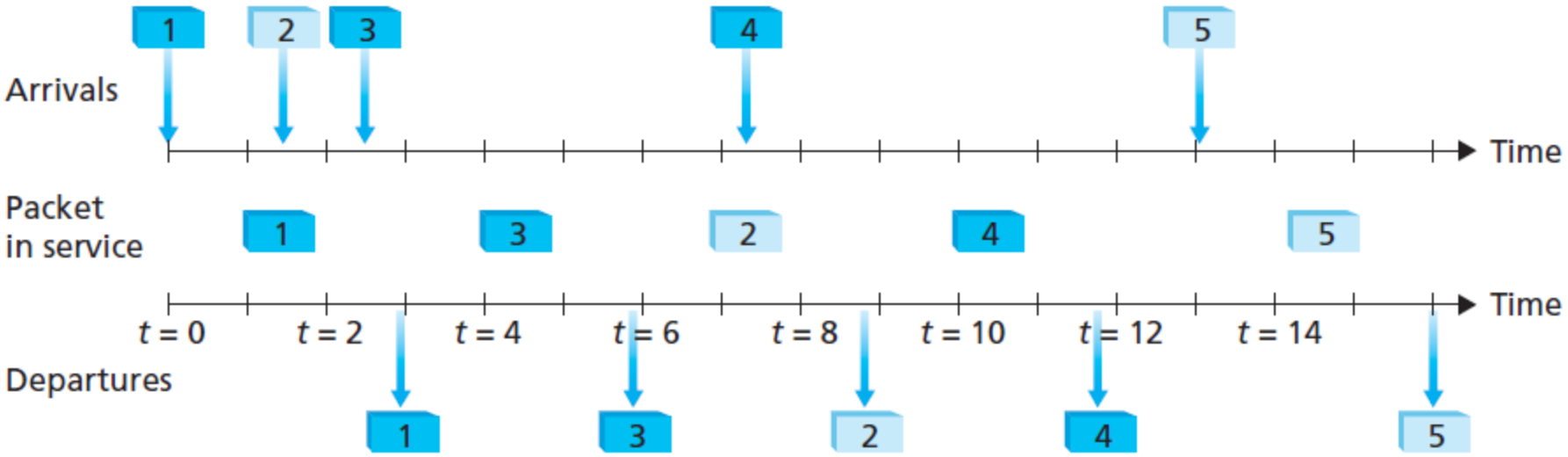
- Multiple queues, serve VIP/higher queues first



Scheduling policies: priority

priority scheduling: send highest priority queued packet

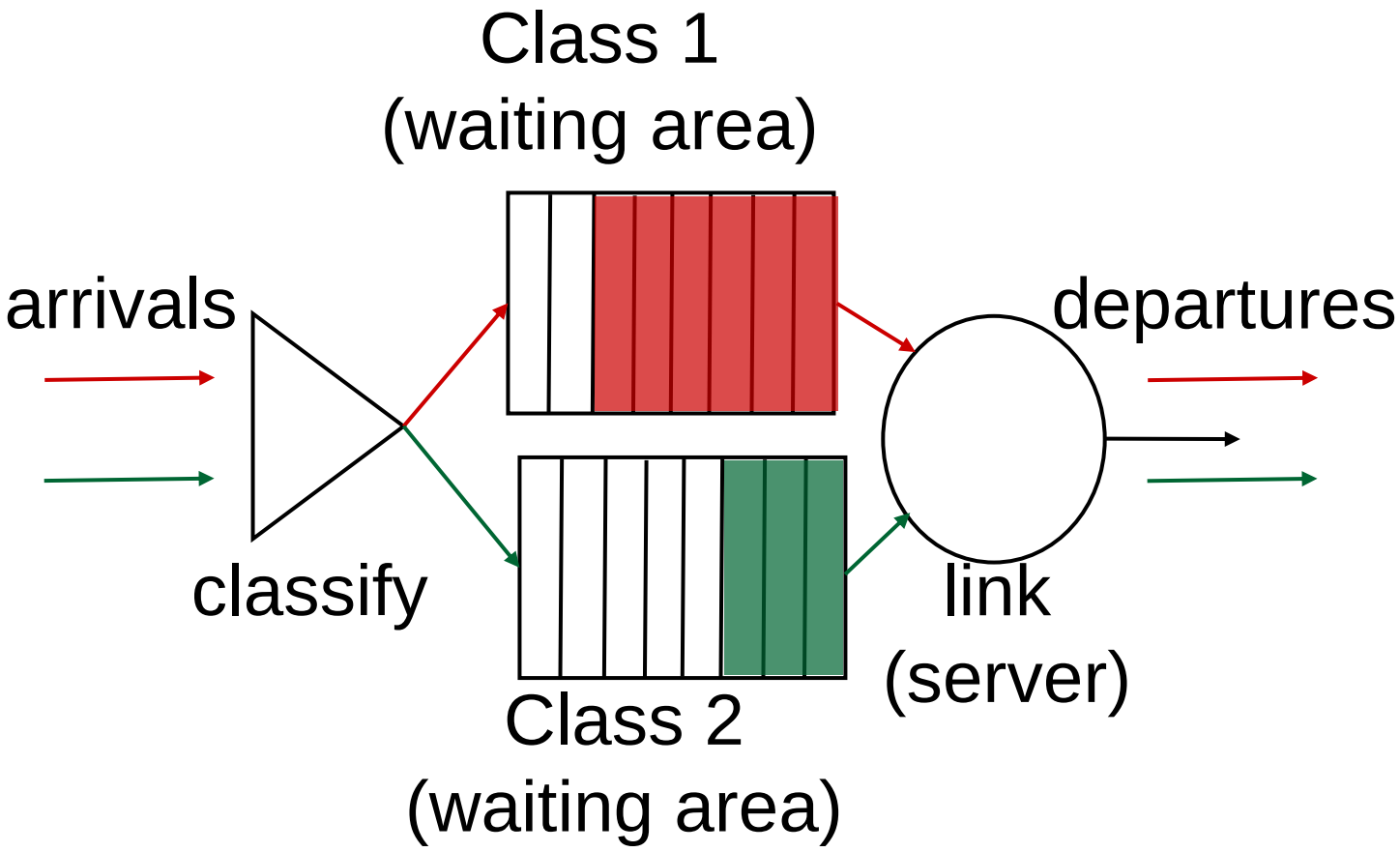
- multiple *classes*, with different priorities
  - class may depend on marking or other header info, e.g. IP source/dest, port numbers, etc.
  - real world example?



Source: Chapter 7 text + slide set, J.F Kurose and K.W. Ross, Ed 6

Round-Robin (RR)

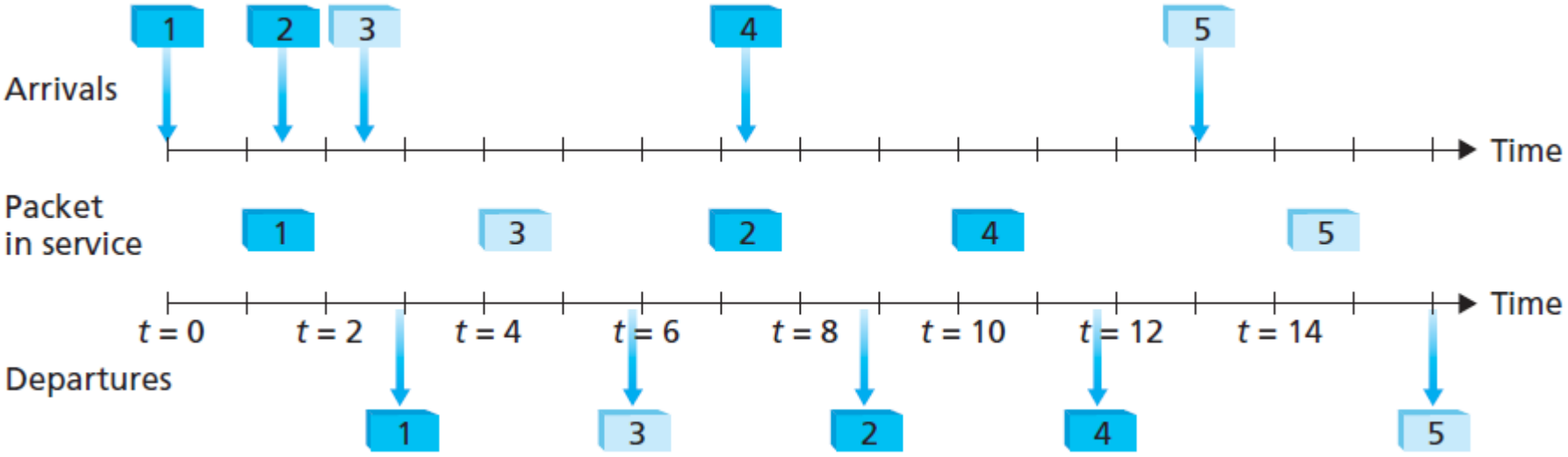
- Class 1 and Class 2 servers alternating
- Work conserving round robin discipline



Scheduling policies: still more

Round Robin (RR) scheduling:

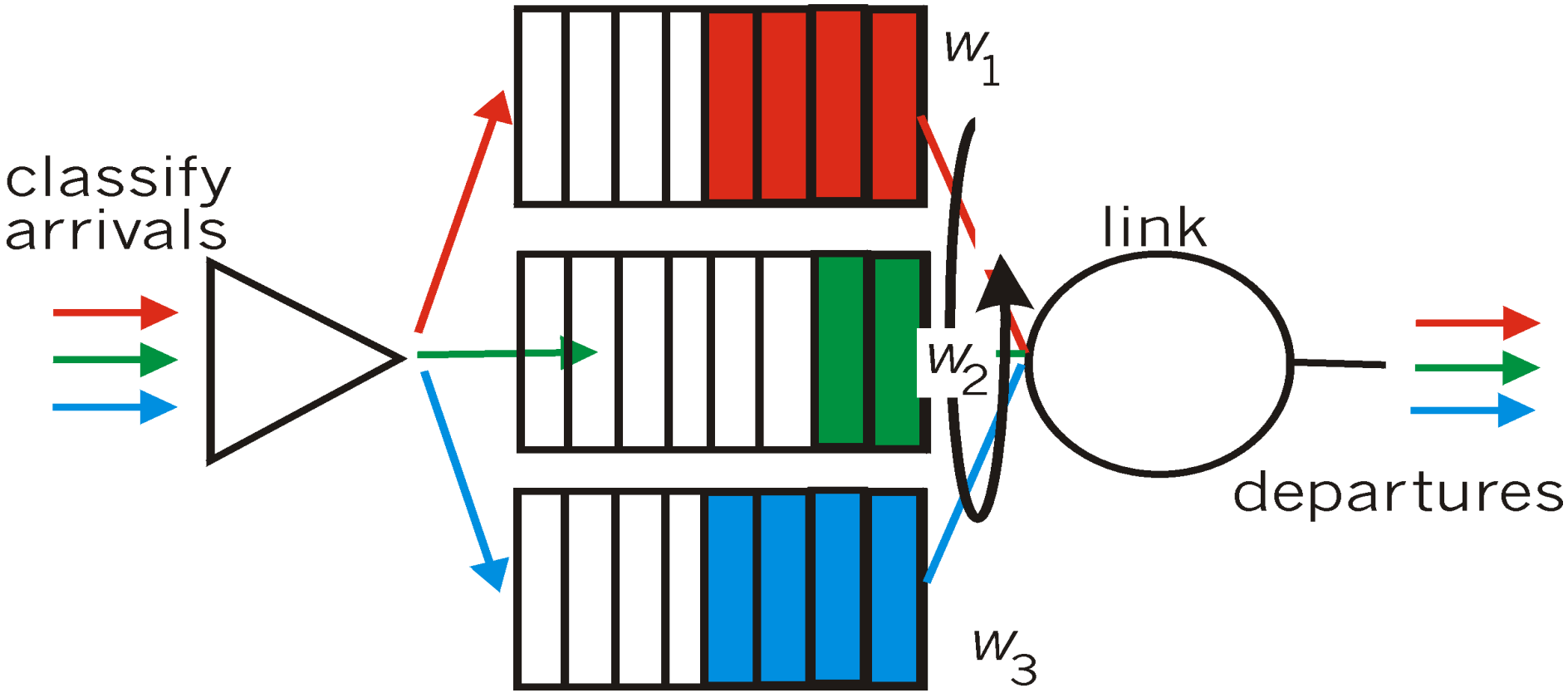
- multiple classes
- cyclically scan class queues, sending one complete packet from each class (if available)
- real world example?



Source: Chapter 7 text + slide set, J.F Kurose and K.W. Ross, Ed 6

Weighted Fair Queueing (WFQ)

- Multiple queues, one for each class
- Round Robin with weight for each class



Scheduling policies: still more

Weighted Fair Queuing (WFQ):

- generalized Round Robin
- each class gets weighted amount of service in each cycle
- real-world example?

Source: Chapter 7 slide set, J.F Kurose and K.W. Ross, Ed 6

Summary Scheduling

- Scheduling can be implemented using queues
- Allows to guarantee minimal transmission rate for classes
- Allows to distribute remaining bandwidth according to classes
- But scheduling cannot enforce limits on traffic in class

Packet Policing

- What if VIP packet grabs all bandwidth?
- QoS is intended to reserve certain share of connection
  - Not provide all bandwidth to VIP
- The routers also do *policing* in addition to marking
- The router will keep track of traffic generated by sources
  - E.g. reserve 50% for Skype traffic
- Effectively, QoS can isolate traffic classes from each other
  - Prevent classes from influencing other traffic

Policing mechanisms

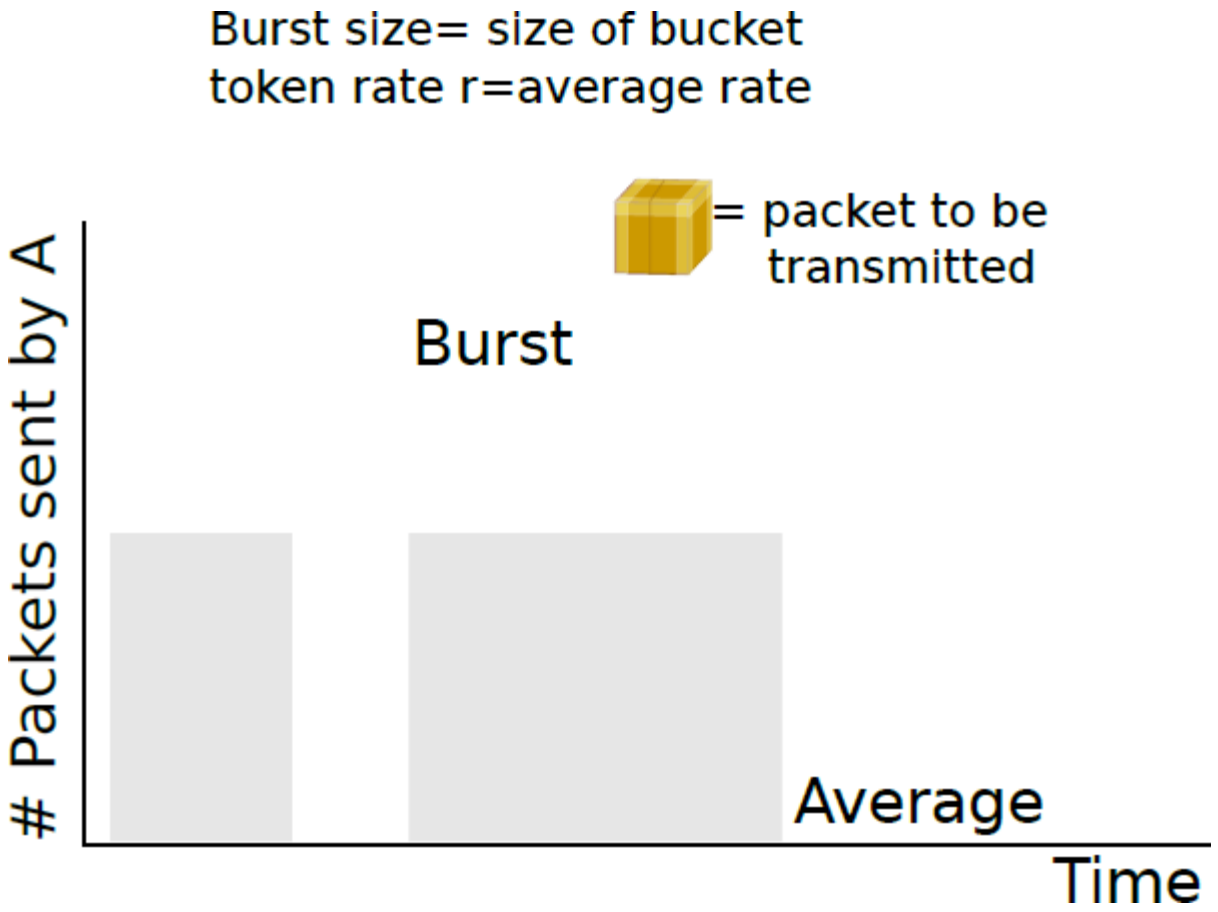
SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

goal: limit traffic to not exceed declared parameters

Three common-used criteria:

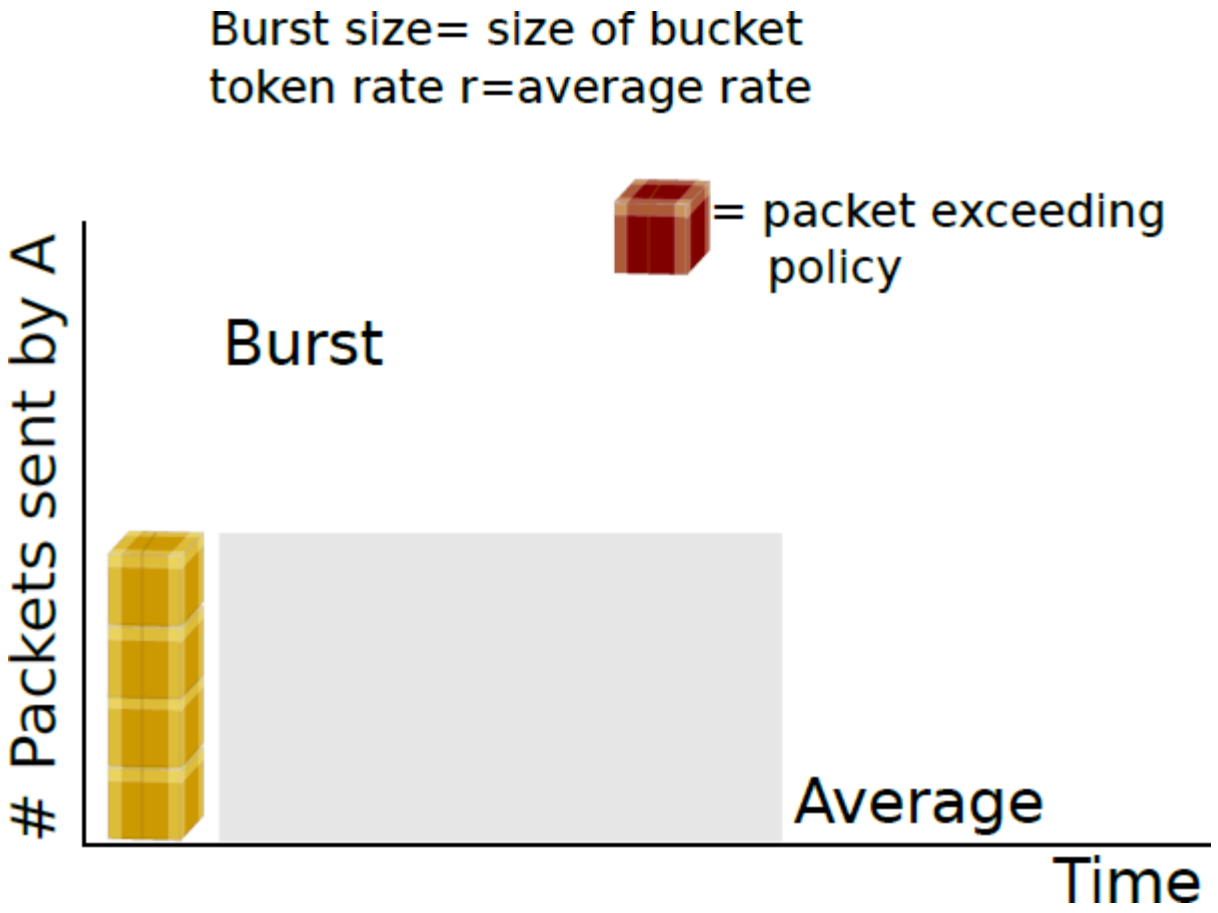
- *(long term) average rate:* how many pkts can be sent per unit time (in the long run)
  - crucial question: what is the interval length: 100 packets per sec or 6000 packets per min have same average!
- *peak rate:* e.g., 6000 pkts per min (ppm) avg.; 1500 ppm peak rate
- *(max.) burst size:* max number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

Average and Burst



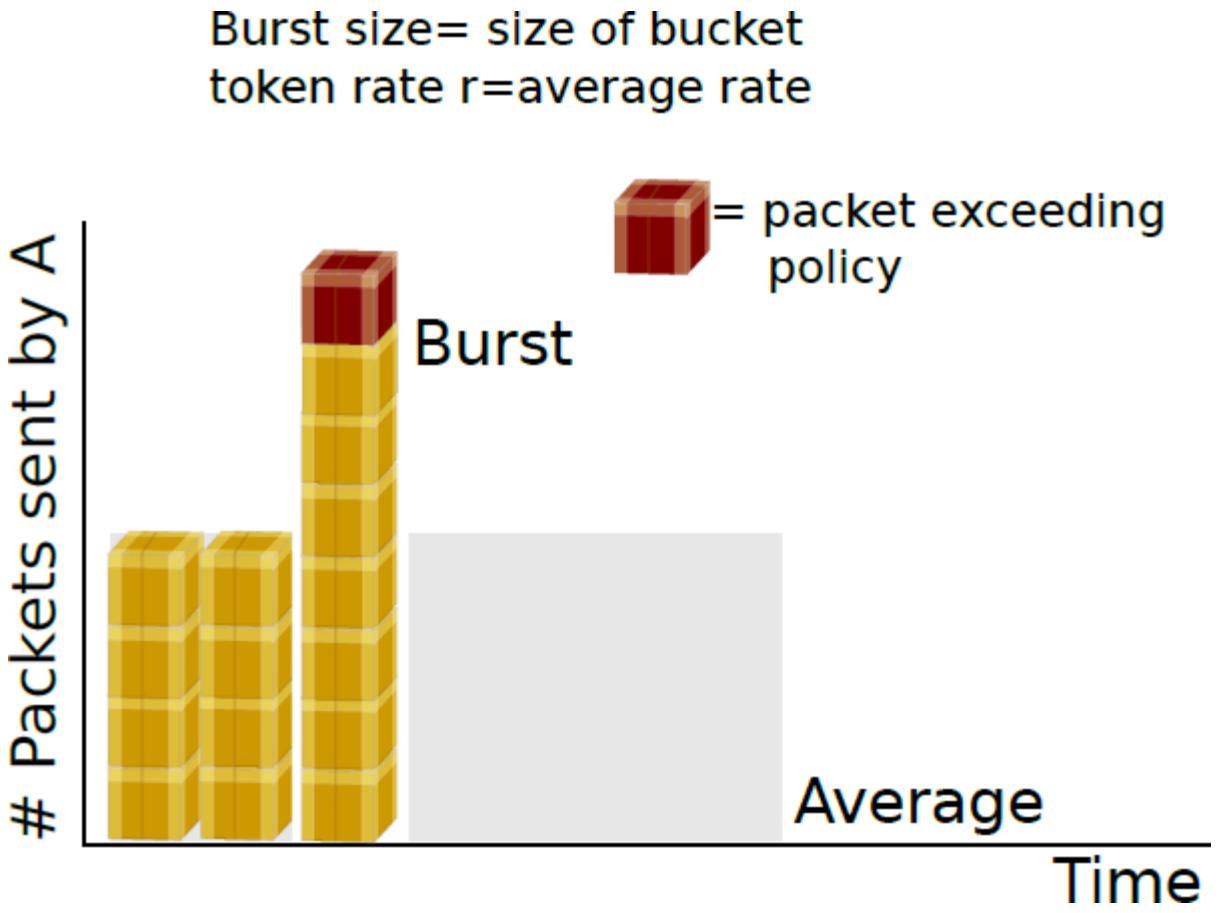
Average and Burst policies

Average and Burst



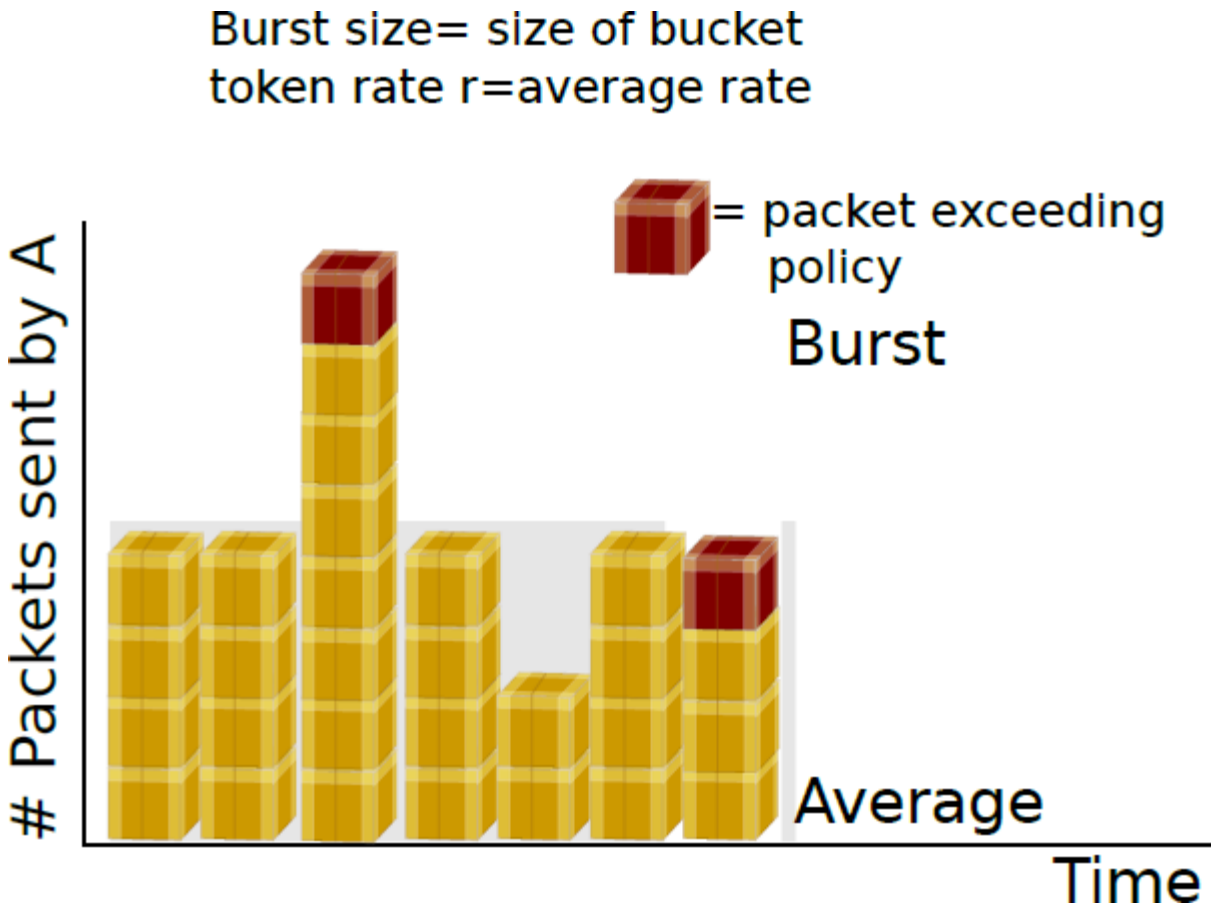
4 packets per slot fit in Average and Burst

Average and Burst



8 packets per slot exceed Burst policy

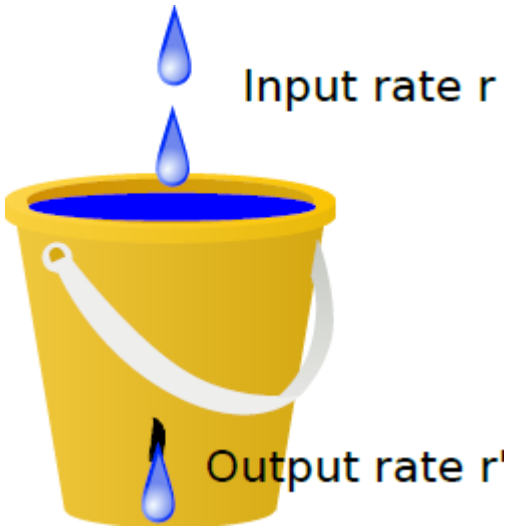
Average and Burst



29 packets exceed the Average Policy (28 over 7)

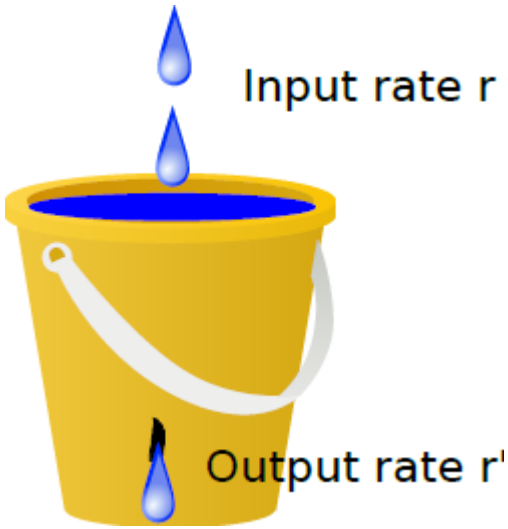
Leaky Bucket

- The *leaky bucket* algorithm is used in NICs or routers
  - Second variant: Token Bucket, discussed later
- Leaky Bucket is rate limited queue
  - Incoming packets arriving with rate *r*
  - Packets collected in the bucket of size *b*
  - Packets leaving the bucket with fix rate *r'*
  - If *r* > *r',* bucket begins to fill
  - If bucket full, drop next packet of class
  - This effectively limits rate of traffic class
  - *b* limits burst rate
  - *r'* limits average rate
- What is the problem with this approach?



Leaky Bucket

- The *leaky bucket* algorithm is used in NICs or routers
  - Second variant: Token Bucket, discussed later
- Leaky Bucket is rate limited queue
  - Incoming packets arriving with rate *r*
  - Packets collected in the bucket of size *b*
  - Packets leaving the bucket with fix rate *r'*
  - If *r* > *r',* bucket begins to fill
  - If bucket full, drop next packet of class
  - This effectively limits rate of traffic class
  - *b* limits burst rate
  - *r'* limits average rate
- What is the problem with this approach?



- Packets are wasting time waiting in queue because they are being sent out at a fixed rate

Example Leaky Bucket

Leaky Bucket with b=5, r'=3

Input	Bucket	Output	Dropped
P1,P2,P3	-	-	-
P4,P5,P6,P7	-	P1,P2,P3	-

3 packets will be sent to output one round later

Example Leaky Bucket

Leaky Bucket with b=5, r'=3

Input	Bucket	Output	Dropped
P1,P2,P3	-	-	-
P4,P5,P6,P7	-	P1,P2,P3	-
P8,P9,PA,PB	P7	P4,P5,P6	-
			-

4 Packets will fill up the bucket with 1 Packet

Example Leaky Bucket

Leaky Bucket with b=5, r'=3

Input	Bucket	Output	Dropped
P1,P2,P3	-	-	-
P4,P5,P6,P7	-	P1,P2,P3	-
P8,P9,PA,PB	P7	P4,P5,P6	-
PC,PD,PE,PF,PX	PA,PB	P7,P8,P9	-

4 Packets will fill up the bucket with 1 more Packet

Example Leaky Bucket

Leaky Bucket with b=5, r'=3

Input	Bucket	Output	Dropped
P1,P2,P3	-	-	-
P4,P5,P6,P7	-	P1,P2,P3	-
P8,P9,PA,PB	P7	P4,P5,P6	-
PC,PD,PE,PF,PX	PA,PB	P7,P8,P9	-
	PD,PE	PA,PB,PC	PF,PX

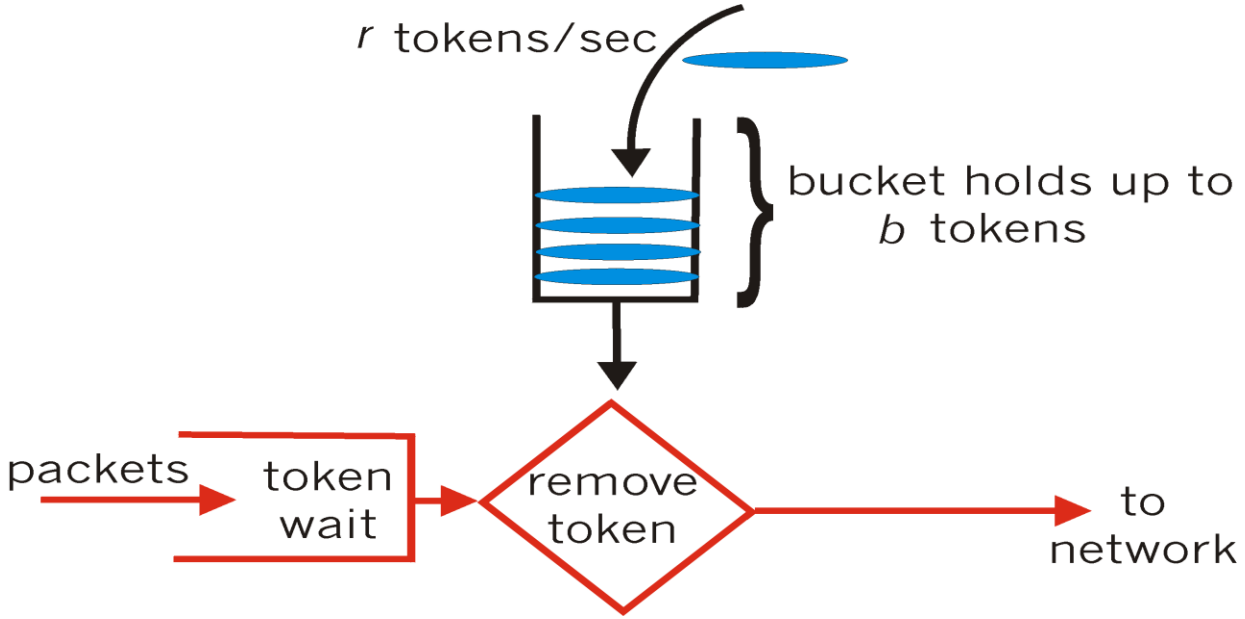
5 Packets will lead to overflow in that case, 2 packets are dropped

Token Bucket

- This variant allows non-uniform output rates and reduced delay
  - Bucket of size *b* is filled with tokens in rate *r',* at each time step *r'* tokens are added to the bucket.
  - If Bucket is full of tokens, newly generated tokens in a time-step are thrown away.
  - Each packet arriving with rate *r* attempts to remove token
  - If no token is left, packet waits in queue
  - If $r < r'$, bucket begins to fill
  - If bucket is full, larger bursts can be transmitted immediately
- In Token Bucket, no additional delay is incurred on packets

Policing mechanisms: implementation

token bucket: limit input to specified *burst size* and *average rate*



- bucket can hold b tokens
- tokens generated at rate *r token/sec* unless bucket full
- *over interval of length t: number of packets admitted less than or equal to (r t + b)*

Source: Kurose and Ross: Chapter 7 slide set

Some advantages of Token bucket over leaky bucket

- If bucket is full, token bucket discards tokens, not packets. However in leaky bucket, packets are discarded when the bucket is full.

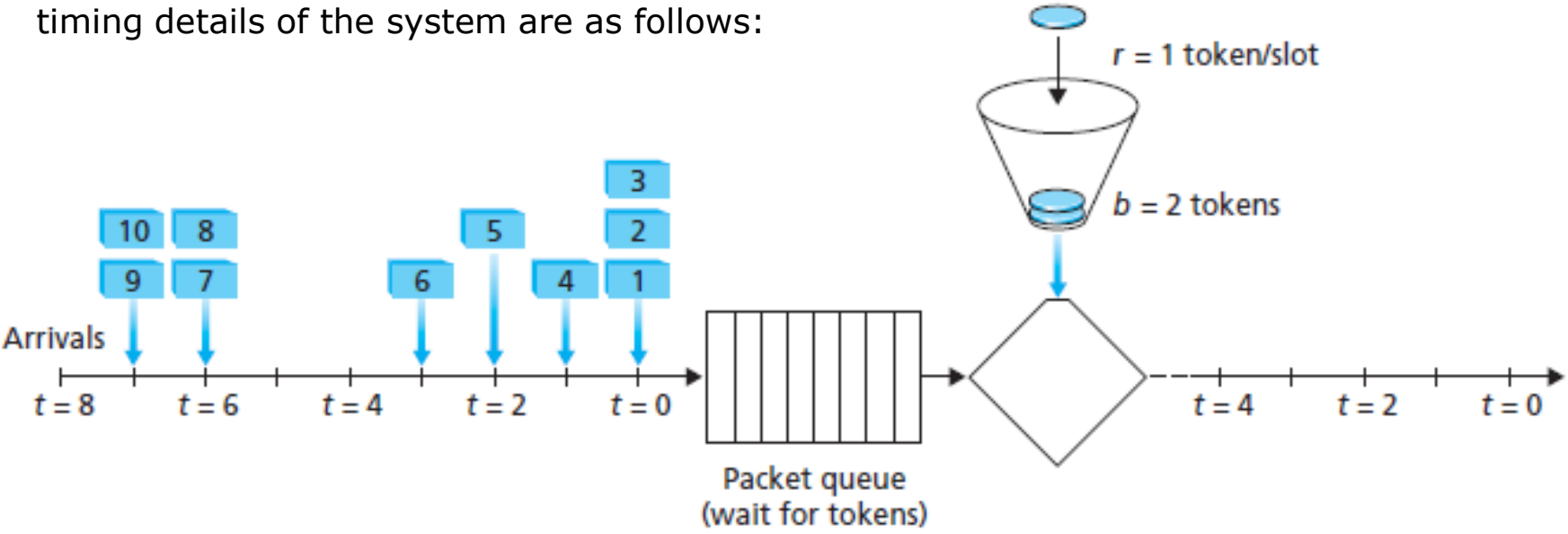
- Token Bucket can send large bursts at a faster rate while leaky bucket always sends packets at a constant rate.

Token Bucket example

A network uses a token bucket for traffic shaping. A new token is put into the bucket every 1 msec. Each token is good for one packet, which contains 100 bytes of data. What is the maximum sustainable (input) data rate?

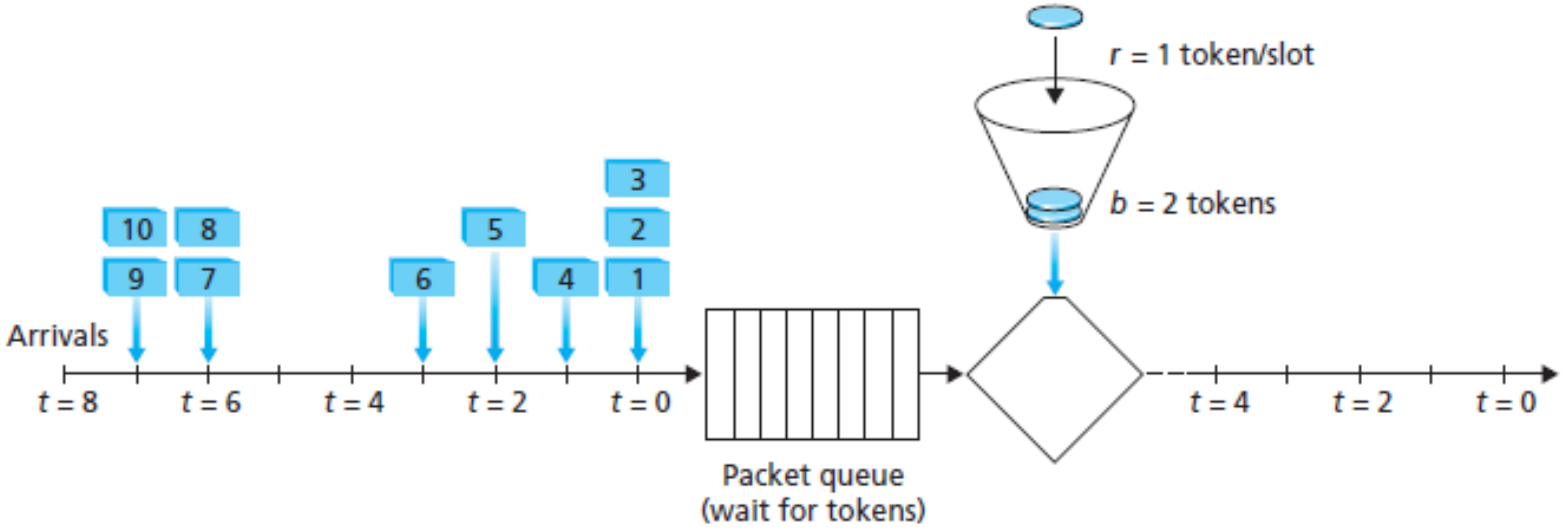
Homework 9: Token Bucket

Consider the token bucket policer shown below. The token buffer can hold at most two tokens, and is initially full at t = 0. New tokens arrive at a rate of **one** token per slot. The output link speed is such that if two packets obtain tokens at the beginning of a time slot, they can both go to the output link in the same slot. The timing details of the system are as follows:



1. Packets proceed towards the front of the queue in a FIFO manner.
2. At most two packets (depending on the number of available tokens) will each remove a token from the token buffer and go to the output link during a slot.
3. A new token is added to the token buffer if it is not full, since the token generation rate is r = 1 token/slot.
4. Time then advances to the next time slot, and these steps repeat.

Homework 9: Token Bucket (cont'd)



Answer the following questions:

a. For each time slot, identify the packets that are in the queue and the number of tokens in the bucket, immediately after the arrivals have been processed (step 1 above) but before any of the packets have passed through the queue and removed a token. Thus, for the t = 0 time slot in the example above, packets 1, 2 and 3 are in the queue, and there are two tokens in the buffer.

b. For each time slot indicate which packets appear on the output after the token(s) have been removed from the queue. Thus, for the t = 0 time slot in the example above, packets 1 and 2 appear on the output link from the leaky buffer during slot 0.

Combining Scheduling and Policing

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

- Use Weighted Fair Queues and Token Bucket metering
  - Token bucket with own *r'* in front of each queue
- This yields an upper and lower bound on available bandwidth
  - If too many packets arrive, they wait (bucket empty)
  - If too few packets arrive, bucket fills up
- The queues after Token Bucket are served by the WFQ algorithm
- -> Now we can provide our fair QoS regime!
  - *Relatively easy to implement in Linux using tc and iptables*

Conclusion

- We discussed
  - Quality of service
- TCP rate adaption
- When QoS is needed
  - Scheduling mechanisms
- Different queue serving mechanisms, e.g. WFQ
  - Policing mechanisms
- Leaky Bucket and Token Bucket