Lecture 6.3

Transport Layer: Quality of Service

Dr. Vandana Kushwaha

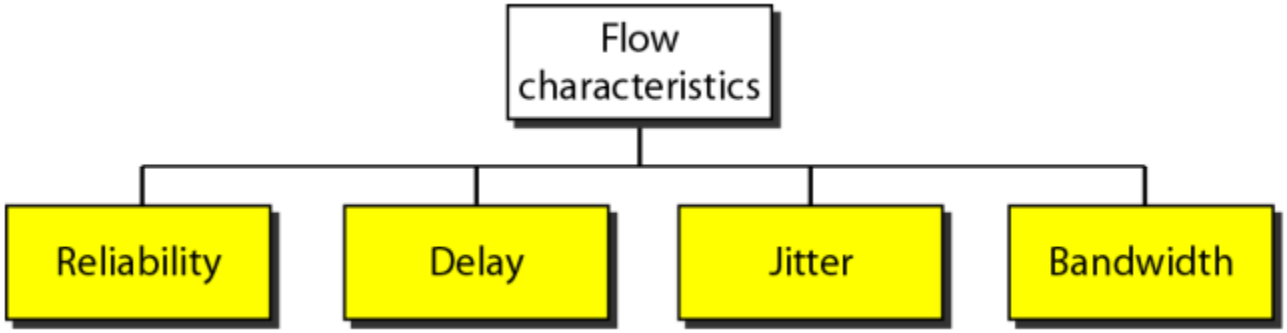
Department of Computer Science

Institute of Science, BHU, Varanasi

Quality of Service

- **Quality of Service (QoS)** refers to the **capability of a network** to provide **better service** to **selected network traffic** over various technologies.
- The **primary goal** of **QoS** is to provide **priority** including dedicated **bandwidth**, **controlled jitter** and **latency** (required by some real-time and interactive traffic), and improved **loss characteristics.**
- One **important issue** is **making sure** that providing **priority** for one or more **flows** does not make other flows **fail.**
- **A flow** can be **defined** as a **combination** of **source** and **destination addresses**, **source** and **destination socket numbers** etc.

Flow Characteristics



1. Reliability

- **Reliability** is a **characteristic** that a **flow** needs.
- **Lack** of **reliability** means **losing a packet** or **acknowledgment**, which entails **retransmission.**
- However, the **sensitivity** of **application programs** to **reliability** is **not** the **same.**
- For **example,** it is **more important** that **electronic mail**, **file transfer** have **reliable transmissions** than **telephony** or **audio conferencing.**

Flow Characteristics

2. Delay

- **Different applications** can **tolerate delay** in **different degrees.**
- In this case, **telephony, audio conferencing, video conferencing**, and **remote log-in** need **minimum delay**, while **delay** in **file transfer** or **e-mail** is **less important.**

3. Jitter

- **Jitte**r is the **variation in delay** for **packets** belonging to the **same flow.**
- For **example**, if **four packets** depart at times **0, 1, 2, 3** and arrive at **20, 21, 22, 23**, all have the **same delay**, **20 units** of time.
- On the other hand, if the above **four packets** arrive at **21, 23, 21**, and **28,** they will have **different delays**: **21,22, 19**, and **24.**

Flow Characteristics

- For **applications** such as **audio** and **video**, the **first case** is **completely acceptable**; the **second case** is **not.**
- For **audio** and **video applications**, it does not matter if the packets arrive with a **short** or **long delay** as long as the **delay** is the **same** for **all packets.**
- **Jitter** is defined as the **variation in the packet delay**.
- **High jitter** means the **difference** between **delays** is **large**; **low jitter** means the **variation** is **small.**

4. Bandwidth

- Different **applications** need **different bandwidths.**
- In **video conferencing** we need to send **millions of bits per second** to refresh a **colour screen** while the total number of bits in an **e-mail** may not reach even **a million.**

TECHNIQUES TO IMPROVE QoS

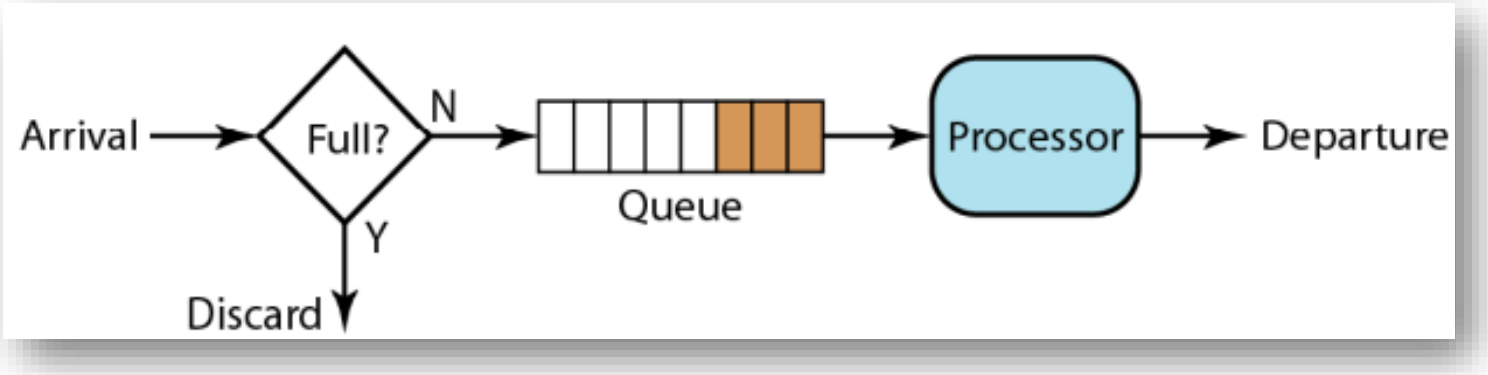
- Some common methods to **improve QoS** are:
    1. ***Scheduling,***
    2. ***Traffic shaping,***
    3. ***Admission control,***
    4. ***Resource Reservation.***

1. Scheduling

- **Packets** from **different flows** arrive at a **switch** or **router** for **processing.**
- A **good scheduling technique** treats the **different flows** in a **fair** and **appropriate** manner.
- Several **scheduling techniques** are designed to **improve** the **quality of service**.
- The **three** of them here: **FIFO queuing**, **Priority queuing**, and **Weighted fair queuing(WFQ).**

Scheduling: FIFO Queuing

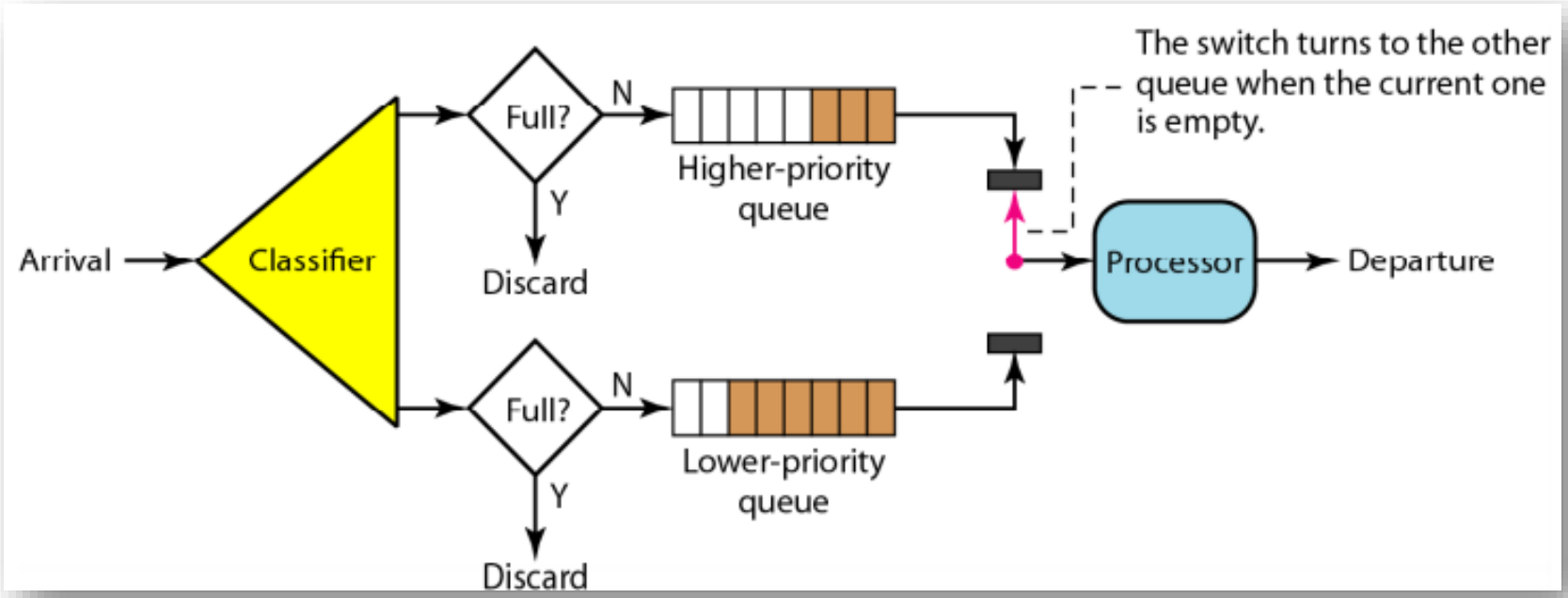
- In **first-in, first-out (FIFO) queuing**, **packets** wait in a **buffer (queue)** until the **node (router or switch)** is ready to **process** them.
- If the **average arrival rate** is **higher** than the **average processing rate**, the **queue** will **fill up** and **new packets** will be **discarded.**



Priority Queuing

- In **priority queuing**, **packets** are first **assigned** to a **priority class.**
- **Each priority class** has **its own queue.**
- The **packets** in the **highest-priority queue** are **processed first.**
- **Packets** in the **lowest-priority queue** are **processed last.**
- Note that the **system** does **not stop** serving a **queue until it is empty.**
- A **priority queue** can provide **better QoS** than the **FIFO queue** because **higher priority traffic**, such as **multimedia**, can reach the destination with **less delay**.
- However, there is a **potential drawback**. If there is a **continuous flow in a high-priority queue**, the **packets** in the **lower-priority queues** will **never have a chance to be processed.**
- This is a condition called ***starvation.***

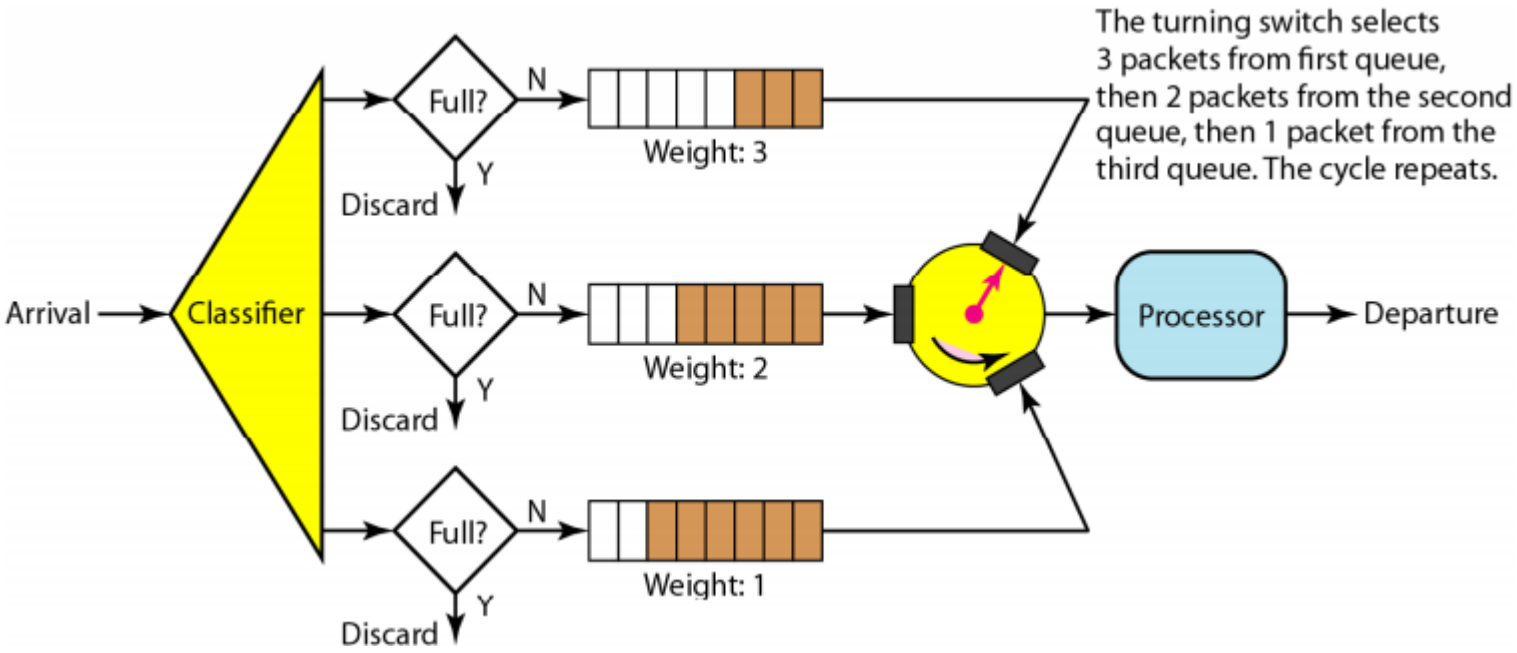
Priority Queuing



Weighted Fair Queuing

- In this **technique**, the **packets** **are still assigned** to **different classes** and **admitted** to **different queues.**
- The **queues,** however, are **weighted** based on the **priority** **of the queues**; **higher priority** means a **higher weight**.
- The **system processes packets** in each **queue** in a **round-robin fashion** with the n**umber of packets** selected from **each queue** based on the **corresponding weight**.
- For **example**, if the **weights** are **3, 2**, and **1**, **three packets** are **processed** from the **first queue**, **two** from the **second queue**, and **one** from the **third queue**.
- If the **system** does not impose **priority** on the **classes**, all **weights** can be **equal** In this way, we have **fair queuing** without priority.

Weighted Fair Queuing



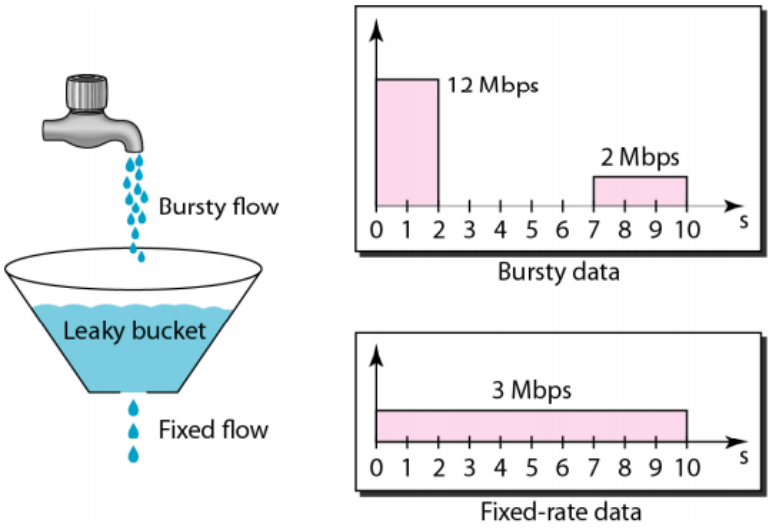
2.Traffic Shaping

- **Traffic shaping** is a **mechanism** to **control** **the amount** and the **rate of the traffic** sent to the **network.**
- **Two techniques** can shape traffic: **Leaky bucket** and **Token bucket.**

Leaky Bucket

- If a **bucket** has a **small hole** at the **bottom**, the **water leaks from the bucket** at a **constant rate** as long as there is **water** in the **bucket.**
- The **rate** **at which the water leaks** **does not depend** on the **rate** **at which the water is input** to the **bucket** unless the **bucket is empty.**
- The **input rate** **can vary**, but the **output rate** **remains constant**.
- Similarly, in **networking**, a technique called **leaky bucket** can **smooth out** **bursty traffic.**
- **Bursty chunks** are **stored** in the **bucket** and **sent out** at an **average rate.**

Leaky bucket



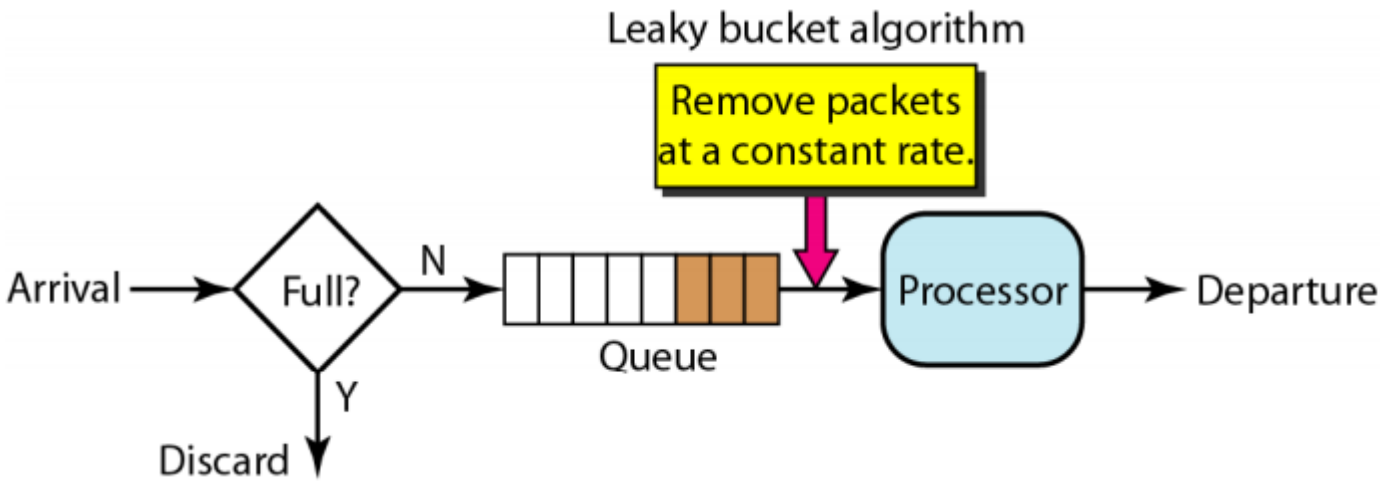
- Assume that the **network** has **committed** a **bandwidth of 3 Mbps** for **a host**.
- The use of the **leaky bucket shapes** the **input traffic** to make it conform to this **commitment.**
- The **host sends** a **burst of data** at a rate of **12 Mbps for 2 s**, for a total of **24 Mbits** of **data.**
- The **host is silent for 5 s** and then **sends data** at a rate of **2 Mbps for 3 s,** for a total of **6 Mbits** of **data.**

Leaky bucket

- In **all**, the **host** has **sent 30 Mbits** of **data** in all.
- The **leaky bucket smooths** the **traffic** by **sending out data** at a **rate** of **3 Mbps** during the same **10s.**
- **Without** the **leaky bucket**, the **beginning burst** may have **hurt the network** by **consuming more bandwidth** than is set aside for this host.
- We can also see that the **leaky bucket** may **prevent congestion.**

Leaky bucket implementation

- A simple **leaky bucket implementation** is shown in **Figure** below.
- A **FIFO queue** holds the **packets.**
- If the **traffic consists of fixed-size packets** (e.g., cells in ATM networks), the process **removes** a **fixed number of packets** from the **queue** at **each tick of the clock.**
- If the **traffic consists of variable-length packets**, the **fixed output rate** must be **based** on the **number of bytes or bits**.



Leaky bucket implementation

The following is an **Algorithm** for **variable-length packets:**

1. Initialize** a **counter** to **n** at the **tick** of the **clock** as **counter=n

2.** If **packet size < counter**, **send the packet** and **decrement the counter** by the **packet size** **as** **counter=n-packet size

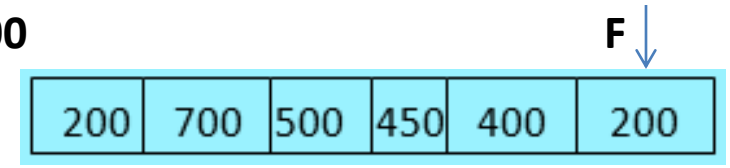
Repeat** this **step** **until** **packet size>counter.

3. Reset** the **counter=n** and **go to** **step 1.

Leaky bucket Example

- Let **n=1000**

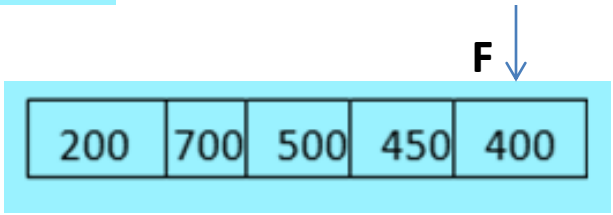
  Packets =



- Since **n> front of Queue** i.e. **n>200**

  Therefore, **n=1000-200=800**

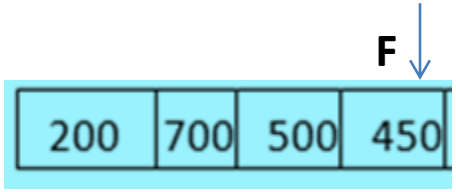
  Packet of **size 200** is **sent** to the **network.**



- Now Again **n>front of the queue** i.e. **n > 400**

  Therefore**, n=800-400=400**

  Packet of **size 400** is **sent** to the **network.**



- Since **n< front of queue**

  Therefore, the **procedure** is **stop.**

- Initialize **n=1000** on **another tick of clock**.

- This **procedure** is **repeated until all the packets** are **sent** to the **network**.

Numerical: Leaky bucket

Question 1: Consider a host using **leaky bucket strategy** for traffic shaping. The host send a burst data at a rate of 15 Mbps for first 3 sec and remains silent for 2 sec. Then again a burst data at a rate of 6 Mbps is sent for next 2 sec and then the host remains silent for next 2 sec. Now again the host send data at a rate of 5 Mbps for next 3 sec. What will be the output data rate of the leaky bucket?

Solution:

Data sent is 15x3 + 6x2 + 5x3=72 Mb

During 3+2+2+2+3=12 sec

So, the **output data rate** is 72/12 = **6 Mbps**

Numerical: Leaky bucket

Question 2: Consider a frame relay network having bucket capacity of 1Mb and data is **input** at the **rate** of **25mbps**.Calculate

i. What is the time needed to fill the bucket.

ii. If the output rate is 2 mbps , the time needed to empty the bucket.

Solution:

Here , C is Capacity of bucket = 1Mb , Data input rate = 25 mbps , Output rate = 2mbps.

i. T = C/input rate = 1/25 = 40 msec

ii. T = C/output rate = ½ = 500 msec

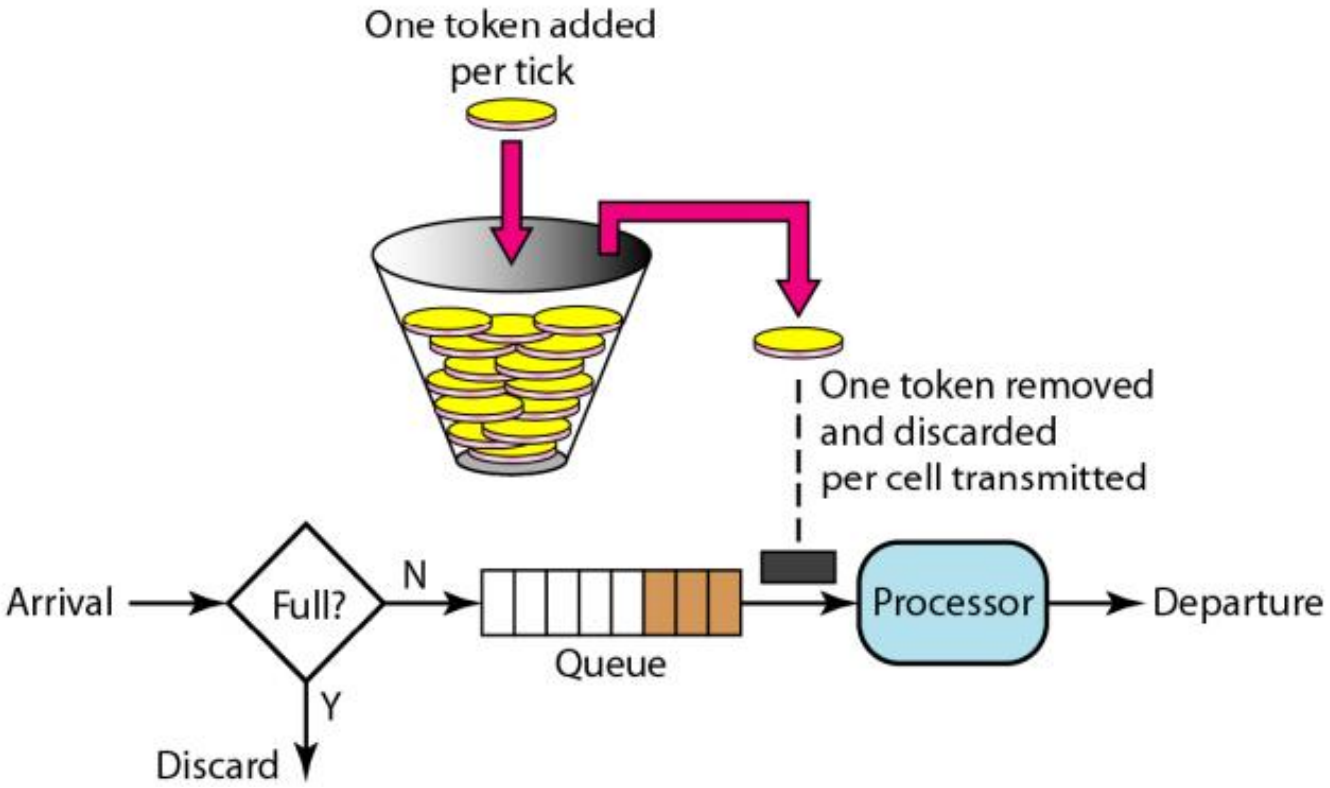
Token Bucket

- The **leaky bucket** is **very restrictive as** It does **not credit an idle host**.
- For **example**, if a **host is not sending for a while**, its bucket becomes **empty**.
- Now if the host has **bursty data,** the **leaky bucket allows only an average rate**.
- The **time when the host was idle** is **not taken into account.**
- On the other hand, the **token bucket algorithm** allows **idle hosts to accumulate credit** for the **future in the form of tokens**.
- For **each tick of the clock**, the **system sends *n* tokens *to the bucket***.
- *The **system removes one token** for* **every byte** of **data sent.**
- For **example**, if ***n*** *is* ***100*** *and the* ***host is idle for 100 ticks,*** the **bucket collects 10,000 tokens.**

Token Bucket

- Now the **host can consume all these tokens** in **one tick** with **10,000 bytes**, or the host takes **1000 ticks** with **10 bytes per tick**.
- In other words, the **host can send bursty data as long as the bucket is not empty.**
- The **token bucket** can easily be **implemented** with a **counter.**
- The **token** is **initialized to zero**.
- Each time a **token is added**, the **counter is incremented by 1**.
- Each time **a unit of data is sent**, the **counter is decremented by 1**.
- When the **counter is zero**, the **host cannot send data.**

Token Bucket



Token Bucket: Algorithm

1. A **token** is **added** to the **bucket** every **1/r seconds**, i.e. **token generation rate** is **r tokens/second.**

2. The **bucket** can hold at the most **B tokens**. If a **token** arrives when the **bucket** is full, it is **discarded.**

3. When a **packet** of ***n* bytes** arrives,

- if **at least *n* tokens** are in the bucket, ***n* tokens are removed** from the **bucket,** and the **packet is sent** to the network.
- if **fewer** than ***n* tokens** are available, **no tokens** are **removed** from the **bucket**, and the **packet** is considered to be **non-conformant.**
- The **non-conformant packets** may either be **dropped** or **queued** for subsequent transmission when **sufficient tokens** have **accumulated** in the **bucket.**

Token Bucket: Algorithm

- For these **non-conformant packets**, there are **two common options:**
- **Dropped (discarded):** The packet is **removed immediately** to enforce the **traffic rate limit.**
- **Queued (delayed):** The packet is **stored temporarily** until **enough tokens accumulate,** allowing it to be **sent later** without violating the rate limit.

- **Basic token update rule**
- Between events, **tokens increase linearly**:

$$T(t_{\text{new}}) = \min\left(B,\ T(t_{\text{old}}) + r \cdot \Delta t\right)$$

- When a **packet of size** S is transmitted, **tokens reduce**: **T = T–S**

Token Bucket: Example

Question: *Consider the following parameters and packet arrival details. Find the packet processing sequence using Token Bucket.*

- **Parameters (units = bytes):**
  - Token generation rate **r**=1000 bytes/s
  - Bucket capacity **B**=2000 bytes
  - Initial tokens **T(0)**=2000 (bucket full)
- **Packet arrivals (time in seconds):**
  - **Packet 1** (P1): size 1500 bytes at **t=0.0**
  - **Packet 2** (P2): size 1200 bytes at **t=0.5**
  - **Packet 3** (P3): size 800 bytes at **t=1.0**

Token Bucket: Example

- **1) t=0.0 : P1 arrives (1500 B)**
  - Initial tokens = 2000.
  - 2000 ≥ 1500 → **conformant** → **send immediately.**
  - **Tokens after sending**: 2000−1500=500
- **2) t=0.5 : P2 arrives (1200 B)**
  - Time since last event = 0.5 s → **tokens accumulate**: **r·Δt**=1000×0.5=**500**
  - **Tokens before decision**: **min**(2000, 500+500)=**1000**
  - Need **1200**, but only **1000 available** → **non-conformant** (short by 200 bytes).
- Now any one of **two policies** can be followed:

Token Bucket: Example

Policy A — Queue (delay until tokens available)

- Shortfall = $1200 - 1000 = 200$ bytes.
- With rate 1000 B/s, time to accumulate 200 bytes = $200/1000 = 0.2$ s.
- So P2 will be sent at $t = 0.5 + 0.2 = 0.7$ s.

At send moment $t = 0.7$:

- Tokens reached 1200, send P2 → tokens after sending = $1200 - 1200 = 0$.

Token Bucket: Example

Next, P3 at $t = 1.0$ (0.3 s after 0.7):

- Tokens accumulated in 0.3 s = $1000 \times 0.3 = 300$.
- Tokens available = 0 + 300 = 300.
- P3 size 800 → non-conformant (short by 500).
- Wait time to get 500 bytes = $500/1000 = 0.5$ s → P3 sent at $t = 1.5$ s.
- After sending P3 at 1.5 s, tokens = 0.

Token Bucket: Example

Policy B — Drop (discard non-conformant immediately)

- At $t = 0.5$, P2 is dropped. Tokens remain 1000.
- At $t = 1.0$ (0.5 s later), tokens accumulate: $1000 + 1000 \times 0.5 = 1500$ (below capacity).
- P3 arrives (800 B): 1500 ≥ 800 → **conformant**, send immediately.
- Tokens after sending = $1500 - 800 = 700$.

TECHNIQUES TO IMPROVE QoS

3. Resource Reservation

- A flow of data needs **resources** such as a **buffer**, **bandwidth**, **CPU time**, and so on.
- The **quality of service** is **improved** if these resources are **reserved beforehand.**
- There are some **QoS model** called **Integrated Services Model** and **Differentiated Services** , provide **resource reservation** to improve the quality of service.

4. Admission Control

- Admission control refers to the **mechanism** used by a **router,** or a switch, to **accept** or **reject a flow** based on **predefined parameters** called **flow specifications.**
- Before a **router accepts** a **flow** for **processing,** it checks the **flow specifications** to see if its **capacity** (in terms of bandwidth, buffer size, CPU speed, etc.) and its **previous commitments** to other flows can **handle** the **new flow.**