how does client locate root server in 1st step?

- the client program is initialized with the IP address of a local name server
- a local name server has ≥1 root servers
  - 例 dns1.cs.princeton.edu → 198.41.0.4
    - 198.41.0.3
    - 198.41.0.2

cach(缓存)
- local server cashes responses ⇒ to resolve future queries (不必再问root)
- TTL: how long each record can be safely cached.
- caching 机制的好处: reduce the load on the root & TLD servers.

Exam 2, 2017, Dijkstra Question 4/(a)

□ has been visited

| V | A | B | C | D | E | min |
|---|---|---|---|---|---|---|
| A | [(A,0,-)] | (B,2,B) | (C,∞,A) | (D,6,D) | (E,8,E) | 2 → 选B |
| B | [...] | [(B,2,B)] | (C,4,B) | (D,6,D) | (E,3,B) | 3 → 选E |

can B reach C? Yes 即 A→B→C cost = 2+2 = 4 < ∞ ∴ choose 4

D? No 即 A→B→D cost = ∞
上一步知 A ⟶ D cost = 6
} min=6 ⇒ choose (D,6,A)

E? Yes 即 A→B→E cost = 3
上一步知 A ⟶ E cost = 8
} min=3 ⇒ choose (E,3,A)

| | | | | | |
|---|---|---|---|---|---|
| E | [...] | [...] | (C,4,B) | (D,5,B) | [(E,3,B)] → 4 → 选C |

can E reach C? Yes A→B→E→C cost = 3+2 = 5
上一步知 (C,4,B) 即 A→B→C cost = 4
} min=4 → 选(C,4,B)

D? Yes A→B→E→D cost = 3+2 = 5
上一步知 (D,6,D) 即 A→D cost = 6
} min=5 → 选(D,5,B)

| | | | | | |
|---|---|---|---|---|---|
| C | [...] | [...] | [(C,4,B)] | (D,5,E) | [ ] |

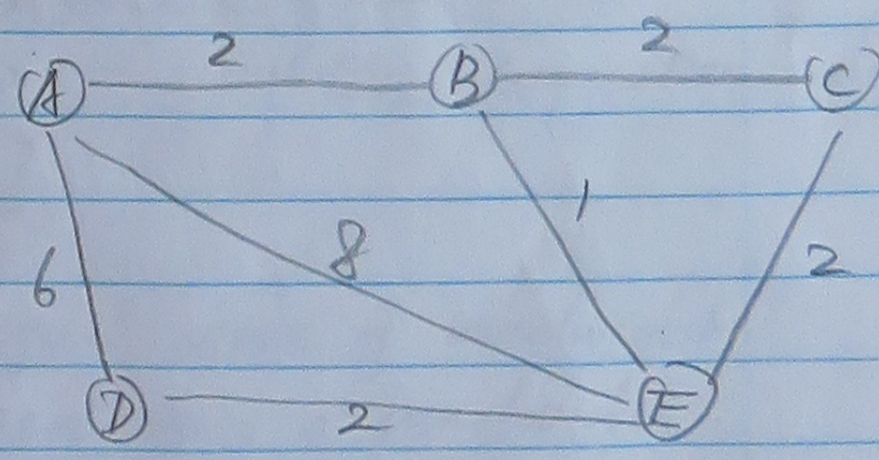
can C reach D? No
上一步知 (D,5,E)
cost = ∞
cost = 5
} min=5 → 选(D,5,E)

| | |
|---|---|
| D | [(D,5,E)] |

综上 (A,0,-) (B,2,B) (E,3,B) (C,4,B) (D,5,B).

不表示: A → B → E → C → D.
而表示: A → B ⟶ E → C
↘ D

题目 Dijkstra on node A.



书P288 Question 13.

X：不选
✓：选

| from LAN | to Root | possible paths | 经由几个 switch/bridge 即 cost | | ID比较 | decision |
|---|---|---|---|---|---|---|
| A | B1 | A→B7→B1 | 1 | 1<2 | 不用比 | ✓ |
| | | A→B2→B3→B1 | 2 | | | X |

∴ link A→B2 is cut off

| | | | | | | |
|---|---|---|---|---|---|---|
| B | B1 | B→B2→B7→B1 | 2 | 8<9 | | X |
| | | B→B5→B3→B1 | 2 | | | ✓ |

∴ link B→B5 is cut off.

| | | | | | |
|---|---|---|---|---|---|
| C | B1 | C→B1 | 0 | | ✓ |
| D | B1 | D→B2→B7→B1 | 2 | 1<2 | X |
| | | D→B3→B1 | 1 | | ✓ |
| E | B1 | E→B1 | 0 | | ✓ |
| F | B1 | F→B3→B1 | 1 | 1<3 | ✓ |
| | | F→B5→B2→B7→B1 | 3 | | X |
| G | B1 | G→B3→B1 | 1 | 1<3 | ✓ |
| | | G→B4→B6→B3→B1 | 3 | | X |
| I | B1 | I→B4→B3→B1 | 2 | 7<9 | ✓ |
| | | I→B6→B3→B1 | 2 | | X |
| J | B1 | J→B6→B3→B1 | 2 | 2<3 | ✓ |
| | | J→B6→B4→B3→B1 | 3 | | X |

Chapter 6

# Congestion Control & Resource Allocation

resource allocation
└ the process by which network elements try to meet the 竞争激烈的 demands that applications have for network resources
即 竞争 [ link bandwidth 或
└ buffer space in routers or switches.

Congestion control
└ effort made by network nodes to {prevent / respond to} overload conditions

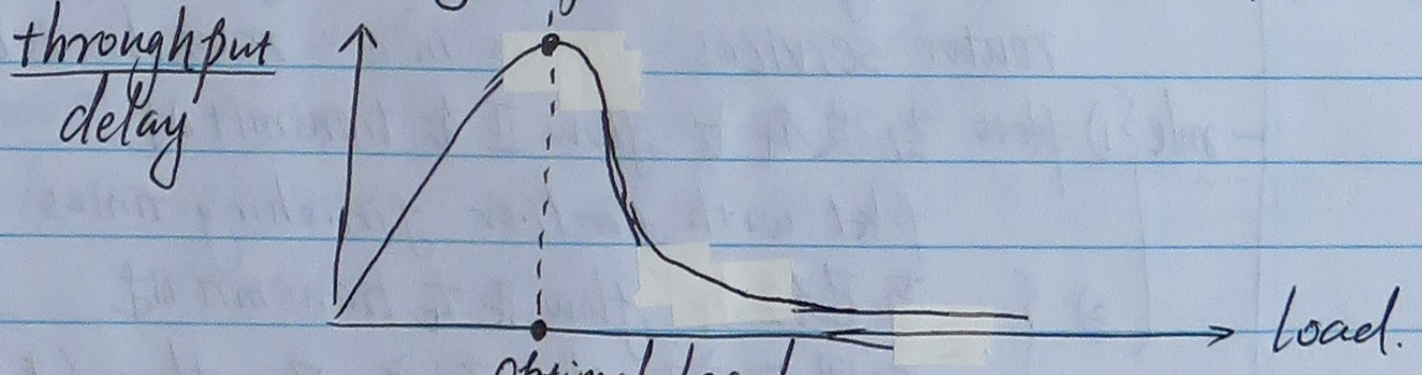
## 6.1.1 network model

problem: too high-speed link feeds in a low-speed link.
idea of flow: a sequence of pkts sent between a source/destination pair and following the same route through network.

## 6.1.3 evaluation criteria

how a network effectively & fairly allocate its resources.

- effective resource allocation.
  - throughput 越↑越好
  - delay 越↓越好
  - } 目标
  - ratio "power" = $\frac{\text{throughput}}{\text{delay}}$ power↑ 说明 effective↑
  - 目标: max "power"
  - 则需求: how much load you place on the network 使得 max. power.




  - [ stable heavy load ⇒ network 仍能传 pkt
  - [ not stable ⇒ 出现 congestion collapse
- fair resource allocation.
  - fair share of bandwidth: equal share of banelwidth.
    - each flow receives an equal share of bandwidth.
    - fair = equal

## 6.2 Queuing Disciplines

queuing algorithm
- allocate
  - bandwidth: which pkts get transmitted
  - buffer space: ..... discarded

### 6.2.1 FIFO

first in first out. 也是 first come first served.

- 机制
  - 1st pkt that arrives at a router is 1st pkt to be transmitted
  - when buffer space is full / a pkt arrives ⇒ router discard that pkt (tail drop)
- 变体一 priority queuing
  - pkts in lower classes are only served if there are no pkts in higher class waiting
  - mark each pkt with a priority, pkts are classfied
  - router always transmits pkt out of the highest-priority queue, if that queue is 非空的.
  - within each priority, pkts are managed by FIFO.
  - 缺点: low priority do not get served.
  - used in Differentiated Services Code Point.
- 缺陷 in FIFO: does not separate pkts according to which they belong ⇓ their traffic sources
  - ①
  - ② 某一病态 source flow 占有过大的 share ⇒ other pkts are discarded.

### 6.2.2 Fair Queuing FQ

when a flow sends pkts too fast, ⇒ its queue fills up
router services queues in a round-robin cycle.

- rule
  - 1) 当没有 ∀ flow 正在 transmit 时, pkt with earlier finishing times are sent first.
  - 2) 当已经有 flow 正在 transmit 时, 如果有新 pkt 到达另一 flow, (无论 transmisson 所需时间是长 or 短), 都继续传正在传的 flow. 当正在传的 flow 传完了它所有 pkt, 才开始传另一个 flow.
- 变体: weighted Fair Queuing WFQ
  - each flow has a weight
  - weight 决定了此 flow 在 bandwidth 中的占比
  - related to bit-by-bit round robin with # of bits served per class

## 6.3 TCP Congestion Control

idea — for each source to determine how much capacity is available in network.
⇒ knows how many pkts it can safely have in transit.

### 6.3.1 AIMD additive increase / multiplicative decrease

congestion window: used by source to limit how much data it is allowed to have in transit at a given time.

how to set congestion window
- additive increase
  - when level of congestion ↑ ⇒ ↓ congestion wnd
- multiplicative decrease
  - ..... ↓ ⇒ ↑ .....

how does source determine the network is congested?
- observe pkt loss
  - Because congestion in network ⇒ pkt is time out ⇒ pkt is dropped
- a pkt loss is detected ⇒ new cong wnd = old cong wnd $\times \frac{1}{2}$

  multiplicative decrease
- congestion wnd should ≥ 1 pkt.

  即 MSS maximum segment size.

MSS: max segment size 1个pkt/or segment 最多能容纳多少 bit/or bytes

additive increase
- when source get ACK in time ⇒ 说明 no pkt loss ⇒ no congestion.
  - then: new cong wnd = old cong wnd + 1
  - 即 increase cong wnd by 1 MSS every RTT.
  - equivalent: increase cong wnd by MSS $\times \frac{MSS}{congwnd}$ for each acked MSS.

### 6.3.2 Slow Start

对比
- AIMD: ↑ Cong Wnd linearly — cong wnd += 1
- slow start: ↑ ..... exponentially — cong wnd = $2^n$

ss threshold: slow start threshold 也称为 Congestion threshold.

AIMD总结
- when source get ACK in time (pkts are successfully transmitted)
  - additive increase: cong wnd += 1
- when source do not get ACK in time (即 pkt loss)
  - multiplicative decrease: cong wnd = cong wnd ÷ 2

① Cong Wnd = 1 (initialization)
② Cong Wnd ↑ exponentially : from 1 to $2^n$
③ when a pkt is lost
  ↳ [ new Cong Wnd = 1
  [ slow start threshold = ~~the~~ old Cong Wnd ÷ 2
④ Cong Wnd ↑ exponentially : from 1 to $2^n$
⑤ when Cong Wnd reaches slow start threshold
  ↳ additive increase 即 new Cong Wnd = old Cong Wnd + 1
⑥ when a pkt is lost
  ↳ loop from ③ 到 ⑤

slow start 在两种情况下会被使用.

6.3.3 Fast Retransmit & Fast Recovery.

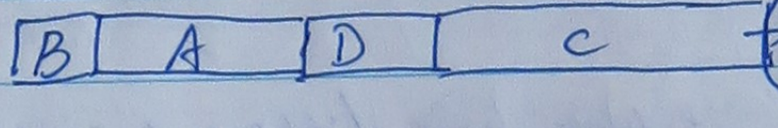
- 为了解决问题：当 pkt loss 发生，但 TCP 需要用 time out 去判断 pkt loss 是否发生，∴ TCP connection 要等很久时间且不敢传输 pkt ⇒ connection goes dead 在这段时间内.
- solution : fast retransmit.
  - trigger the retransmission of a dropped pkt sooner than the regular timeout mechanism.
- fast retransmit
  - when a pkt arrives out of order
    - 则 TCP resends the same ACK that it sent last time.
      duplicate ACK.
  - when sender sees a duplicate ACK.
    - 则 it knows that [an earlier pkt might have been lost.
      或 [ . . . . . delayed.
    - 为了甄鉴别是 lost 还是 delay,
      sender waits until it sees 3 duplicate ACKs
      - 当 sender see 3 duplicate ACKs,
        - sender retransmit the lost pkt.
- fast recovery
  - removes the slow start phase that happens between
    - between when fast retransmit detects a lost pkt
      and additive increase begins

- fast recovery avoids this slow start period instead
  - new Congestion Wnd = old Cong Wnd $\times \frac{1}{2}$
  - 恢复 additive increase
- slow start is only used in
  - the beginning of a connection
  - whenever a time out occurs
- at all other time, cong Wnd follows a pure addictive increase/ multiplicative decrease pattern.

## Scheduling

in what order are pkts served, after they enter a queue in a router?

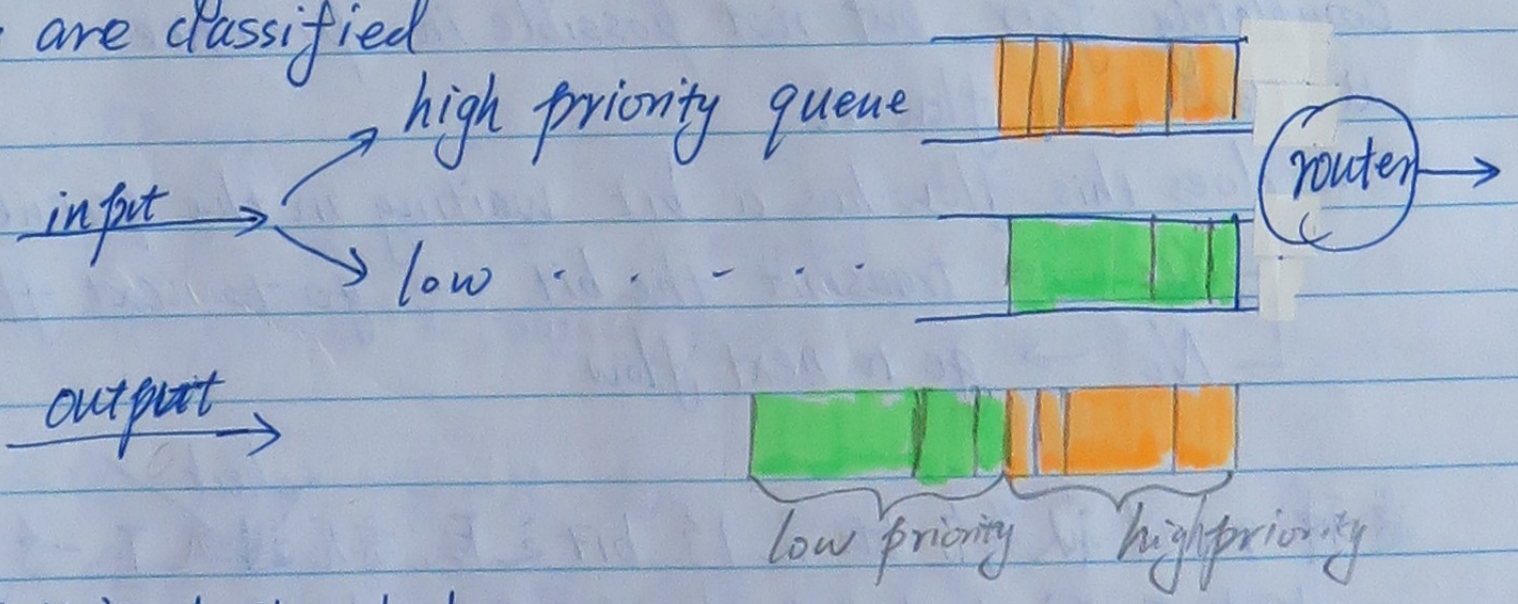
FIFO

- first in first out
  - ... come ... serve
  - [ 1st pkt that arrives at router is 1st pkt to be transmitted
    [ nth . . . . . nth . . . . . .
- no classification of pkt.
- tail drop: when buffer space is full, if a pkt arrives
  - ↳ router discard it
- no differential treatment
- input pkt: → 

- disadvantage: if a pkt is very large → it occupies a large fraction of network capacity → other pkts have to be discarded.

## Priority Queuing

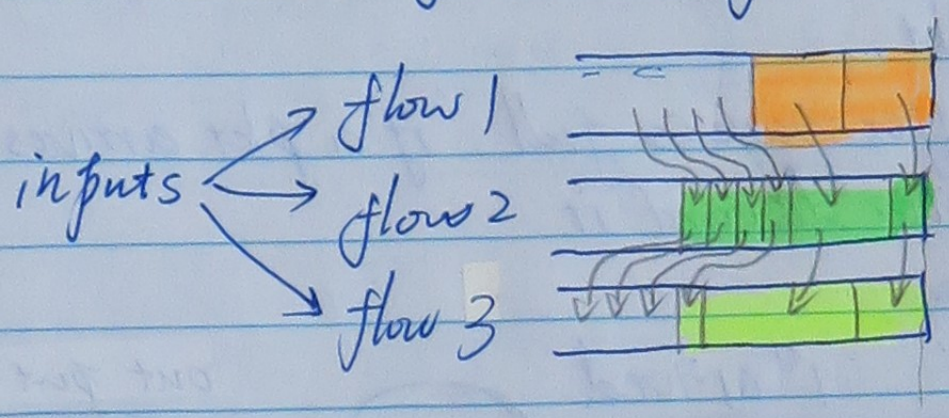
- pkts are classified

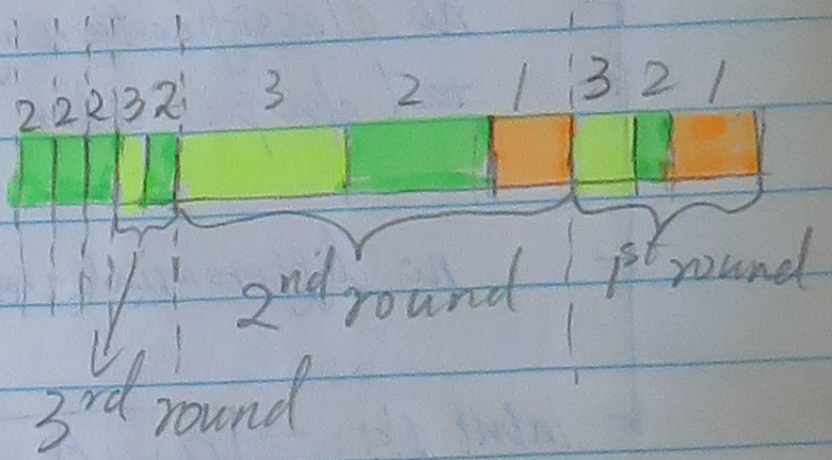



- 每传完一个 pkt, check:
  - ↳ does high priority queue has a pkt that is waiting to be transmitted?
    - [ if it has → transmit next pkt from high priority queue
    - [ if it doesn't have → . . . . pkt from low priority . . .
- within each priority queue, pkts are managed by FIFO.
- disadvantage: 只要 high priority queue 有一个 pkt 还没被传走, pkt in low . . . . . 就不会被传
  - ⇒ [ high priority queue 是长期占用 network capacity.
    [ low . . . . 可能不被 served.

## Round-Robin Scheduling

- pkts are classified.
- a round: 一个轮回，每个 queue 被依次遍历一次.
- 从 flow 1 到 flow n (in this case n=3)
  - does this flow has a pkt waiting in the queue?
    - Yes → transmit the pkt → go to next flow
    - No → go to next flow.




not completely fair, if pkts have different sizes.
one pkt from each class is served per "round".

## Bit-by-Bit Round

- one bit from each class is served per "round"
- pkts a classfied.
- completely fair, but not possible in practice.
- 从 flow 1 到 flow n
  - does this flow has a bit waiting in the queue?
    - Yes → transmit the bit → go to next flow
    - No → go to next flow.

这种 round 在每传完1个 bit 之后，就 check 下一个 flow 有无 bit，
⇒ 相当于同时传多个 flow.（只要此 flow 有 bit）