eecs489 COMPUTER NETWORKS

Lecture 37:
PA4 Walk-Through

Lab 7 Token-Bucket Filter

Client communicates flow rate to server

`IMGDB_BPTOK` specifies number of bytes covered by each token

Token bucket size, in tokens, computed from `rwnd` and `mss`

Token generation rate, in tokens/sec, computed from user specified flow rate (`frate`), in Kbps

Token accounting is fractional

3 lines of code

Lab 7 Token-Bucket Filter

Server can only send if there's enough token in bucket to cover a segment

Server "consumes" the tokens needed to send a segment

If there isn't enough token accumulated, sleep for a certain amount of time

- amount of sleep time should be long enough to generate sufficient tokens to cover 1 segment of data, plus some random fraction of bucket size worth of tokens

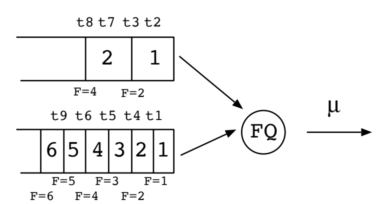
Assume no token accumulation during transmission

About 12 lines of code

Weighted-Fair Queueing

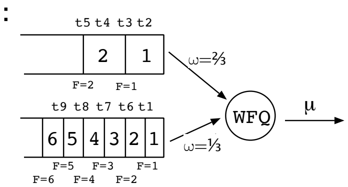
Fair Queueing (FQ):

- compute F: finish round, the round a packet finishes service
- simulates fluid-flow RR in the computation of F's
- serve packets with the smallest F first



Weighted-Fair Queueing (WFQ):

- generalized Round Robin
- each VC/flow/class gets weighted amount of service in each cycle
- $P^{\alpha}_{i} = L^{\alpha}_{i}/(\omega\mu)$, L^{α}_{i} size of packet



Lab 8 Weighted-Fair Queueing

Server accepts download requests from multiple clients

Command line options: `-l`: linkrate (Mbps), `-g`: wait for minimum number of flows before starting transmission (gated start)

Transmission also started if total reserved rate equals link rate

No rate nor flow control

Given n flows with total reserved rate of R, and link rate of μ, each client is served $(r/R)*\mu$ with WFQ

Compute service time (in rounds) of packet with length L as $P = L/(r*\mu/R)$

Lab 8 Weighted-Fair Queueing

Serve packet with smallest finish round first

Total reserved rate R changes as flow enters and departs

Assume no packet pre-emption

Flow departure/arrival checked after sending each packet

Finish round computed for each flow after every packet transmission

Assume no idle flow (doesn't perform round catch up)

In total, less than 25 lines of code

PA 4 Link Virtualization

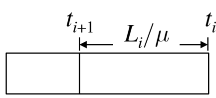
Virtual link indistinguishable from physical link

Client allocated a 1 Mbps virtual link on a 100 Mbps physical link receives at most 1 Mbps even if the rest of the physical link is idle

Work conserving vs. non-work conserving scheduler

No max-min fair sharing!

PA4: Transmission Rate and Inter-packet Gap

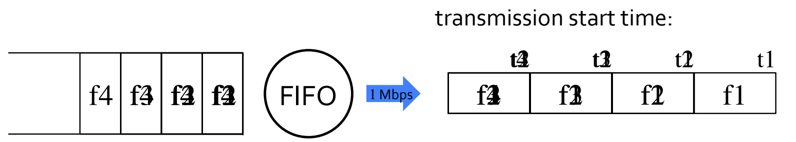


The gap between starts of transmission for packet i (t_i) and $i+1$ (t_{i+1}), sent back-to-back, is the service/transmission time of packet i: L_i/μ, where L_i is the size of packet i, and μ is the link bandwidth

- the inter-packet gap of back-to-back packets of a flow belonging to a FIFO client j with virtual link rate μ_j, is L_i/μ_j

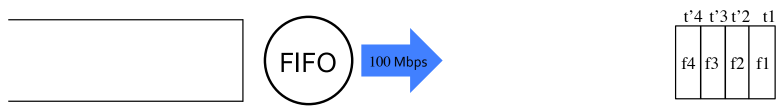
To simulate a lower transmission rate, we introduce idle/sleep time between packets

PA4: Virtualizing a Lower BW Link

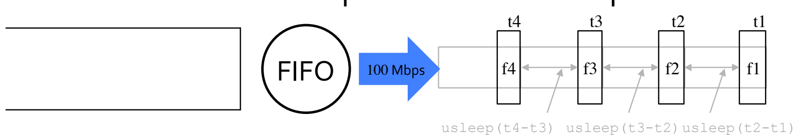


Packet spacing on a 100 Mbps link?

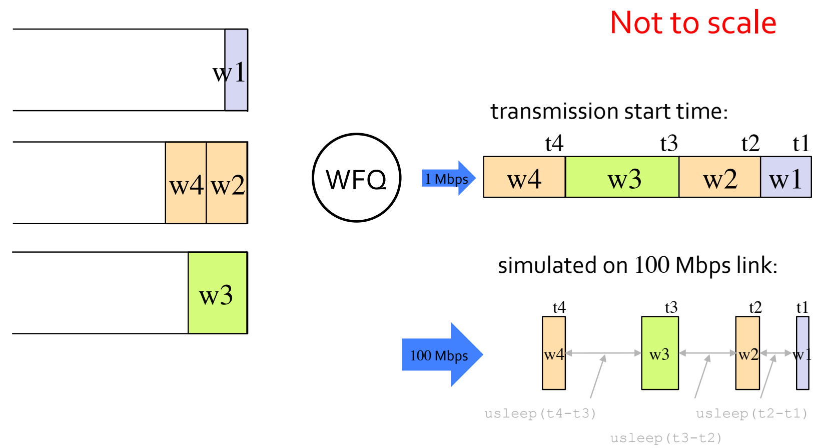
Not to scale



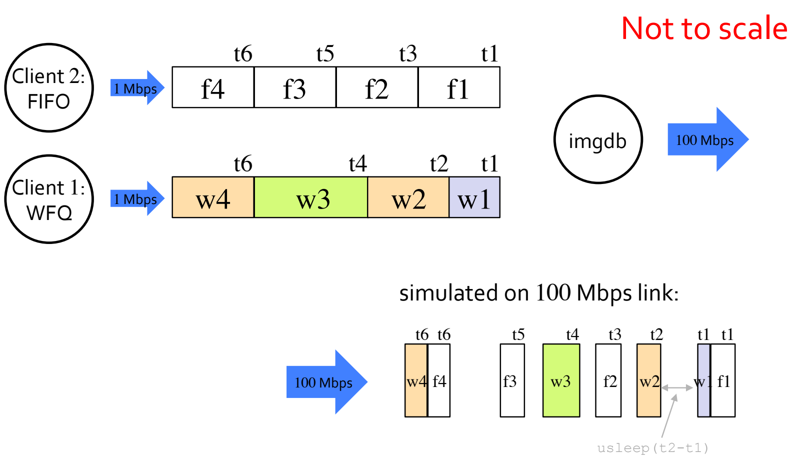
How to emulate a 1 Mbps link on a 100 Mbps link?



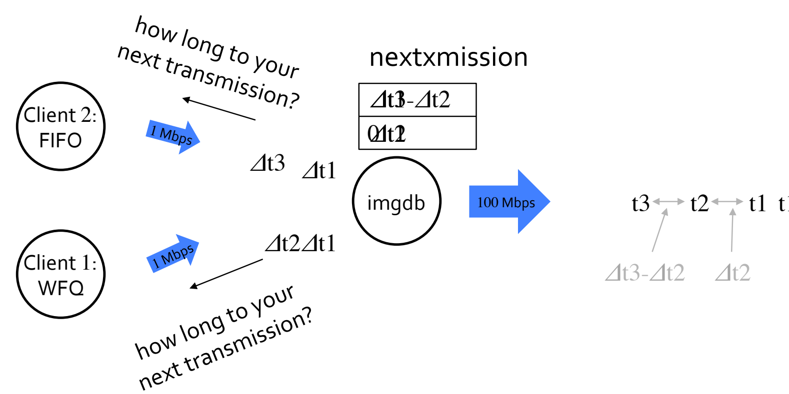
PA4: Virtualizing a Lower BW Link



PA4: Virtualizing 2 Clients



PA4: Virtualization Implementation



Note: only works because we assume flows don't go idle

PA4: Assumptions

No client management:
- hard-code 2 clients, one FIFO, one WFQ
- refactor Lab7 and Lab8 `imgdb`s into 3 classes: `FIFO`, `WFQ`, `imgdb`, instantiate `FIFO` and `WFQ` within `imgdb` (about 30 lines in `imgdb.h`)
- `imgdb` command line option `-f`: fraction of link bandwidth allocated to the `WFQ` client (default: `IMGDB_INITFRAC`)

No flow-setup protocol:
- assume `FIFO` client supports only 1 flow
- `FIFO` client/flow accepted only if at least `IMGDB_MINFRAC` fraction of physical link rate is still available
- use `netimg -r 0` hack to specify `FIFO` client's flow

PA4: Assumptions

For token accumulation, add accumulation time to inter-packet gap
- assume no token accumulation during simulated packet transmission time

Virtual link scheduling takes about another 30 lines of code