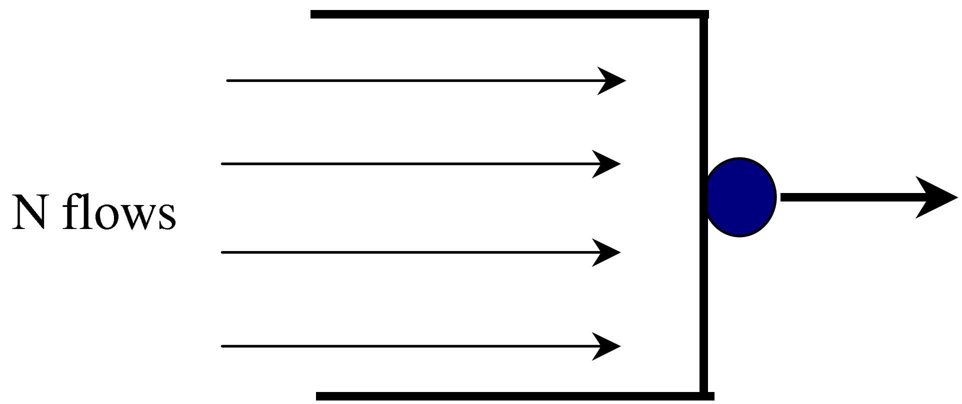
The Virtual Clock Protocol

Advanced Computer Networks

Multiplexors (a.k.a. Schedulers a.k.a. Servers)



- A multiplexor receives multiple input N flows and has an output channel.
- It is not cut-through
- It is work-conserving
- It preserves the packet order of each flow.

Notation

C_{out}: Capacity of the output channel

L_{max}: Maximum packet length allowed by multiplexor

R_f: Rate reserved by flow f

p_f^i: ith packet of flow f

L_f^i: Length of packet p_f^i

A_f^i: Arrival time into multiplexor of the last bit of p_f^i

E_f^i: Exit time from multiplexor of the last bit of p_f^i

T_f^i: Timestamp assigned to p_f^i

Δ_f^i: Delay of p_f^i at multiplexor, equal to $E_f^i - A_f^i$

Bounded Apetite Servers

- Before presenting the Virtual Clock Multiplexors, we introduce Bounded Appetite Servers (VC is a bounded appetite server)
- Consider a server, where each packet p_f^i has a *deadline* D_f^i.
- The server has bounded appetite iff, for any interval of time $[t, t']$, the total number of bytes of packets arriving during the interval and whose deadline is at most t' add to no more than

$$(t' - t) \cdot C_{out}$$

That is

$$\left(\sum\ i, f\ :\ A_f^i \in [t, t'] \wedge D_f^i \leq t'\ :\ L_f^i\right) \leq (t' - t) \cdot C_{out}$$

Lemma: Bounded Appetite Exit Time

Lemma 1 *Consider a server with bounded appetite. For every flow f and every i, $i \geq 1$,*

$$E_f^i \leq D_f^i + \frac{L_{max}}{C_{out}}$$

provided the server is work-conserving, and it forwards packets in order of deadline.

Proof of Lemma

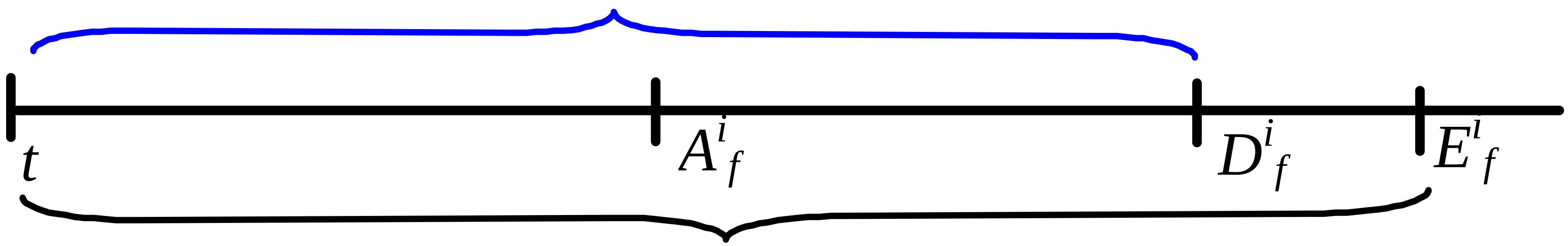
- We assume $E_f^i \geq D_f^i$, otherwise we have no proof obligation.
- We would like to find a time t, where $t \leq A_f^i$, such that

1. During the interval $[t, E_f^i]$, the queue is never empty
   - Item 1 above gives us a lower bound on the number of bytes sent before p_f^i exits
2. During the interval $[t, E_f^i]$, only packets with deadlines at most D_f^i arriving after t are forwarded.
   - Item 2 above gives us an upper bound on the number of bytes to be forwarded before p_f^i exits.
   - Note that the packets must arrive during the interval $[t, D_f^i]$. Why?

Proof Continued ...

1. In a nutshell, we want the following to be true from the definition of t.

Only packets with deadline $\leq D^i_f$ arriving during $[t, D^i_f]$ are forwarded during interval $[t, E^i_f]$

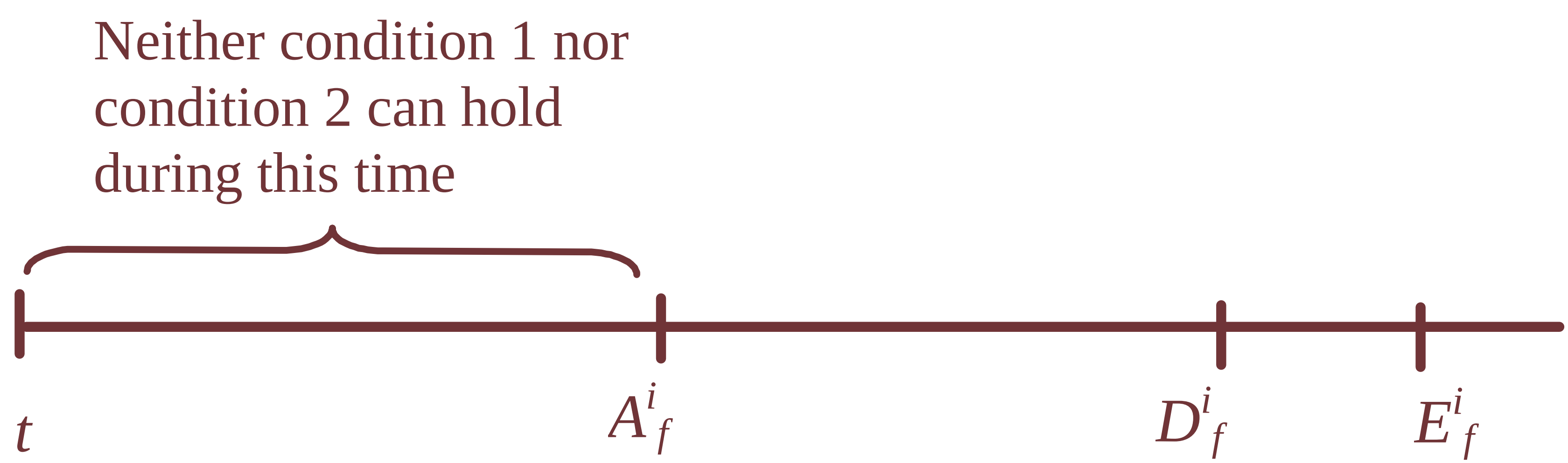


Queue is never empty

Proof Continued ... (def. of t)

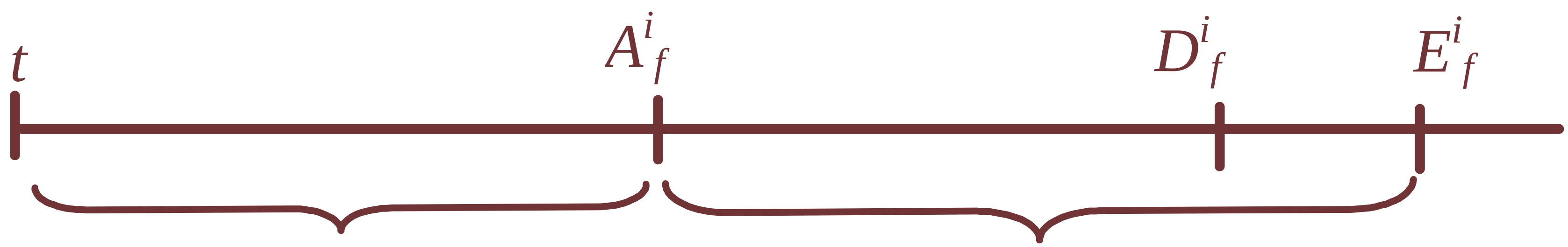
Consider packet p^i_f. Let t be the *latest* time such that $t \leq A^i_f$ and one of the following holds:

1. The queue of the multiplexor is empty at time t.
2. The multiplexor dequeues and forwards at time t a packet with deadline X such that $X > D^i_f$.



Proof Continued ... (queue not empty)

1. The queue of the multiplexor is never empty uring $[t, E^i_f]$.



Queue is never empty from t to A^i_f by the definition of t

Queue is never empty since p^i_f is in the queue already

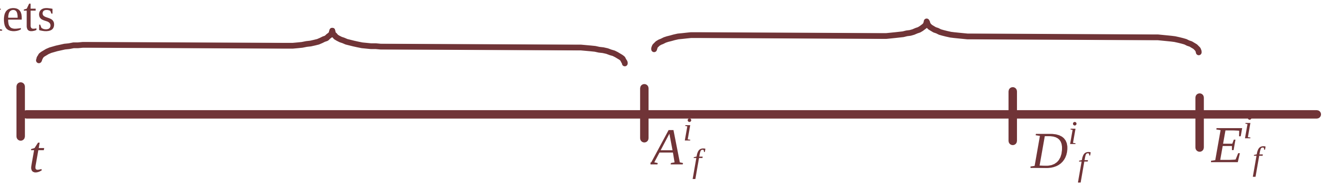
Proof Continued ... (only $\leq D_f^i$ forwarded)

1. During $[t, E_f^i]$, only packets that arrived during $[t, D_f^i]$ whose deadline is at most D_f^i are forwarded.

No packets with deadline > $D^i{}_f$ can be forwarded, according to the definition of *t*

At time $A^i{}_f$, $p^i{}_f$ is already in the system, so no packet with deadline > $D^i{}_f$ can be forwarded.

At time *t*, no packets are queued with deadline ≤ $D^i{}_f$ according to the definition of *t*



Proof Continued ...

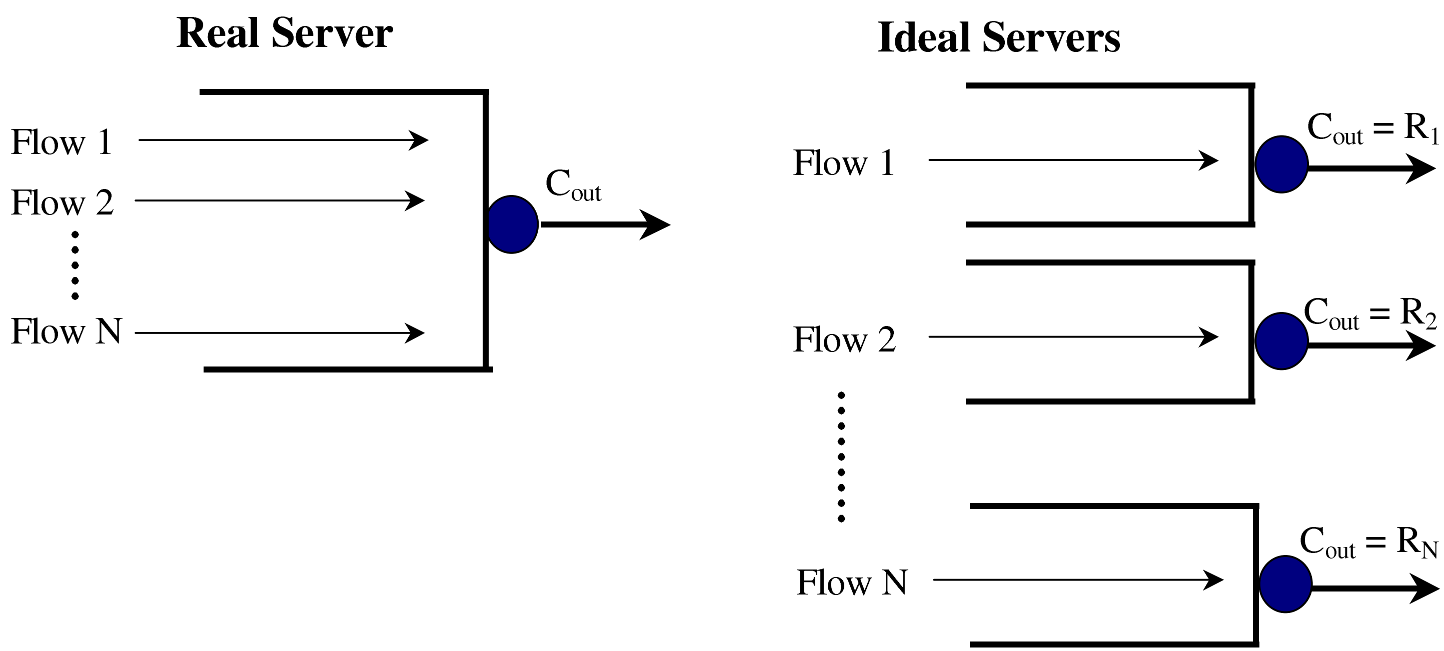
- Therefore,
  - During $[t, E_f^i]$, only packets with deadline at most D_f^i that arrive during $[t, D_f^i]$ are dequeued and forwarded.
  - From the bounded appetite property, these bytes add to at most

$$(D_f^i - t) \cdot C_{out}$$

  - The queue is always busy during time $[t, E_f^i]$, so it takes at most $(D_f^i - t)$ seconds to forward these packets, so p_f^i should exit by time D_f^i.
  - However, at time t, a packet with deadline $> D_f^i$ may be in the middle of being transmitted, and hence,

$$E_f^i \le D_f^i + \frac{L_{max}}{C_{out}}$$

VC Multiplexors



- VC multiplexer makes the real server behave as "good" as the ideal servers.
- Exit time from real server should be at most the exit time from ideal servers.

Method Overview

- Each arriving packet is assigned a timestamp with the time the packet exits the *ideal server*.
- Packets are maintained in a priority queue, whose priority is the timestamp of the packet.
- When the channel becomes idle, the packet with smallest timestamp is forwarded.
- Let M be the number of packets in the queue.
  - Inserting a packet takes $O(\log M)$ time.
  - Removing a packet takes $O(\log M)$ time.
  - Hence, we have higher complexity than FCFS, who has $O(1)$ overhead per packet.

Timestamp Computation

$$T_f^1 = A_f^1 + \frac{L_f^1}{R_f}$$

$$T_f^i = \text{max}(A_f^i, T_f^{i-1}) + \frac{L_f^i}{R_f}$$

- The ideal server of f serves a packet of size L in $\frac{L}{R_f}$ seconds.
- Thus, the first packet exits the ideal server at time $A_f^1 + \frac{L_f^1}{R_f}$.
- Packet p_f^i exits the ideal server when it begins service plus $\frac{L}{R_f}$ seconds.
- When does it begin service? At time $\text{max}(A_f^i, T_f^{i-1})$

The Main Theorem

Theorem 1 *For every flow f and every i, $i \geq 1$,*

$$E_f^i \leq T_f^i + \frac{L_{max}}{C_{out}}$$

Provided,

$$\sum_{g=1}^{N} R_g \leq C_{out}$$

Note that the above bound for flow f is *independent* of any other flow.

Lemma: VC has bounded appetite

Lemma 2 *A VC server, where*

$$\sum_{g=1}^{N} R_g \leq C_{out}$$

has bounded appetite, where $T_f^i = D_f^i$.

That is, for any interval of time $[t, t']$, the total number of bytes of packets arriving during the interval and whose timestamp is at most t' add to no more than

$$(t' - t) \cdot C_{out}$$

In formula,

$$\left(\sum \ i, f \ : \ A_f^i \in [t, t'] \wedge T_f^i \leq t' \ : \ L_f^i \right) \leq (t' - t) \cdot C_{out}$$

Proof of VC Bounded Appetite

- For any flow g, the appetite of g during the interval $[t, t']$ is at most $(t' - t) \cdot R_g$.
- Why?

Proof of VC Bounded Appetite (continued)

- Let p_g^j be the first packet of g after t, and p_g^n the last packet of g with $T_g^n \leq t'$.

$$\begin{aligned}
T_g^j &\geq t + \frac{L_g^j}{R_g} \\
T_g^{j+1} &\geq T_g^j + \frac{L_g^{j+1}}{R_g} \geq t + \frac{L_g^j}{R_g} + \frac{L_g^{j+1}}{R_g} \\
\dots & \\
t' \geq T_g^n &\geq t + \sum_{k=j}^{n} \frac{L_g^k}{R_g}
\end{aligned}$$

Thus,

$$\sum_{k=j}^{n} L_g^k \leq (t' - t) \cdot R_g$$

Proof of VC Bounded Appetite (continued)

- Summing over all g, the appetite of all the flows during an interval $[t, t']$ is,

$$\sum_{g=1}^{N}(t' - t) \cdot R_g = (t' - t) \sum_{g=1}^{N} R_g \leq (t' - t) \cdot C_{out}$$

- Hence, VC is a bounded appetite server.

Delay for a Bounded-Burstiness Flow

- Assume f is (R_f, B) constrained.
- Thus, the queue at a constant rate server of rate R_f is at most B.
- The delay of f through the constant rate server of rate R_f is at most B/R_f.
- Hence,

$$\Delta_f^i \leq \frac{B}{R_f} + \frac{L_{max}}{C_{out}}$$

One Timestamp Per Flow

- This is an alternative timestamping method to improve efficiency.
- If the queue of f is empty at time A_f^i, then

$$T_f^i = \text{max}(A_f^i, T_f^{i-1}) + \frac{L_f^i}{R_f}$$

- If the queue of f is not empty at time A_f^i, then

$$T_f^i = T_f^{i-1} + \frac{L_f^i}{R_f}$$

- This timestap is at most the regular timestamp (if smaller, no more than $\frac{L_{max}}{C_{out}}$ smaller.)

One Timestamp Per Flow (continued ...)

- This can be implementing with only one timestamp T_f per flow f.
- If the queue of f is not empty when p_f^i arrives, it is just appended to the queue of f.
- If the queue of f is empty when p_f^i arrives,

$$T_f := \text{max}(A_f^i, T_f) + \frac{L_f^i}{R_f}$$

- When a packet from f is forwarded, and the queue is not empty, then,

$$T_f := T_f + \frac{L_f^i}{R_f}$$

where p_f^i is the next packet of f.

Packet Timestamp

- The timestamp of packet p_f^i is the value of T_f when p_f^i becomes the head of the queue of f.
- Since we have only one timestamp per flow, enqueuing or dequeuing a packet takes $O(\log N)$ time.

VC Theorem

- Theorem 1 still holds (using the new timestamp). Why?
  - The difference in timestamp assignment is when the queue of f is not empty, and $A_f^i > T_f^{i-1}$.
  - The timestamp assigned in this case in the new method is the same as in the old method when p_f^i arrives at time T_f^{i-1}.
  - However, at time T_f^{i-1} packet p_f^{i-1} is still in the queue, and hence, packet p_f^i cannot be transmitted immediately (in the old system) if it arrives at time T_f^{i-1}.
  - Hence, the new system assigns timestamps and forwards packets in the same way as the old method where packets arrive a little earlier.

Flow Unfairness

- Consider two flows, f and g.
- $R_f = R_g = 100$ bytes/sec,
  $C_{out} = 200$ bytes/sec, $L = 100$ bytes.
- $[0, 100]$, f generates 2 packets/sec., and g is idle.
- Thus, 200 packets of f are forwarded, and $T_f = 200$
  $(L/R_f = 100/100 = 1)$.
- $[100, 200]$, f and g generate 2 packets/sec.
- For the first packet at time 100, $T_f = 201$ and $T_g = 101$.
- Thus, the next 100 packets transmitted are from g.
- $[100, 150]$, g dominates the output channel.

Flow Unfairness, remarks

- f is being *punished* for exceeding its reserved rate.
- However, the additional bandwidth used by f was *unused*, no other flow would use it.
- This is considered *unfair* to f.
- Note that the bounds of Theorem 1 still hold.

Active Flows

- When can the bandwidth of a flow f be *reused*? (assuming f sends no more packets).
- We introduce the notion of an *active* flow.

Definition 1 *A flow f is* active *at time t iff,*

$$t \leq T_f^i$$

where p_f^i is the last packet received from f at time t.

Active Flows (continued ...)

- Theorem 1 still holds.
- However, we replace

$$\sum_{g} R_g \leq C_{out}$$

by

$$\left(\forall\, t, 0 \leq t, \sum_{g \in V(t)} R_g \leq C_{out} \right)$$

where $V(t)$ is the set of active flows at time t.

Active Flows, remarks

- Note that being active is not related to the packets in the queue of the multiplexor, but on the value of the largest timestamp of the flow.
- f may be inactive and still have packets in the queue, and it may be active and have no packets in the queue.
- f being active implies that its *effects* on rate-reservation have not ended.
- Once f becomes inactive, its bandwidth may be given to new flows (or increase the reserved rate of existing flows).