Flexible, efficient resource management for datagram networks

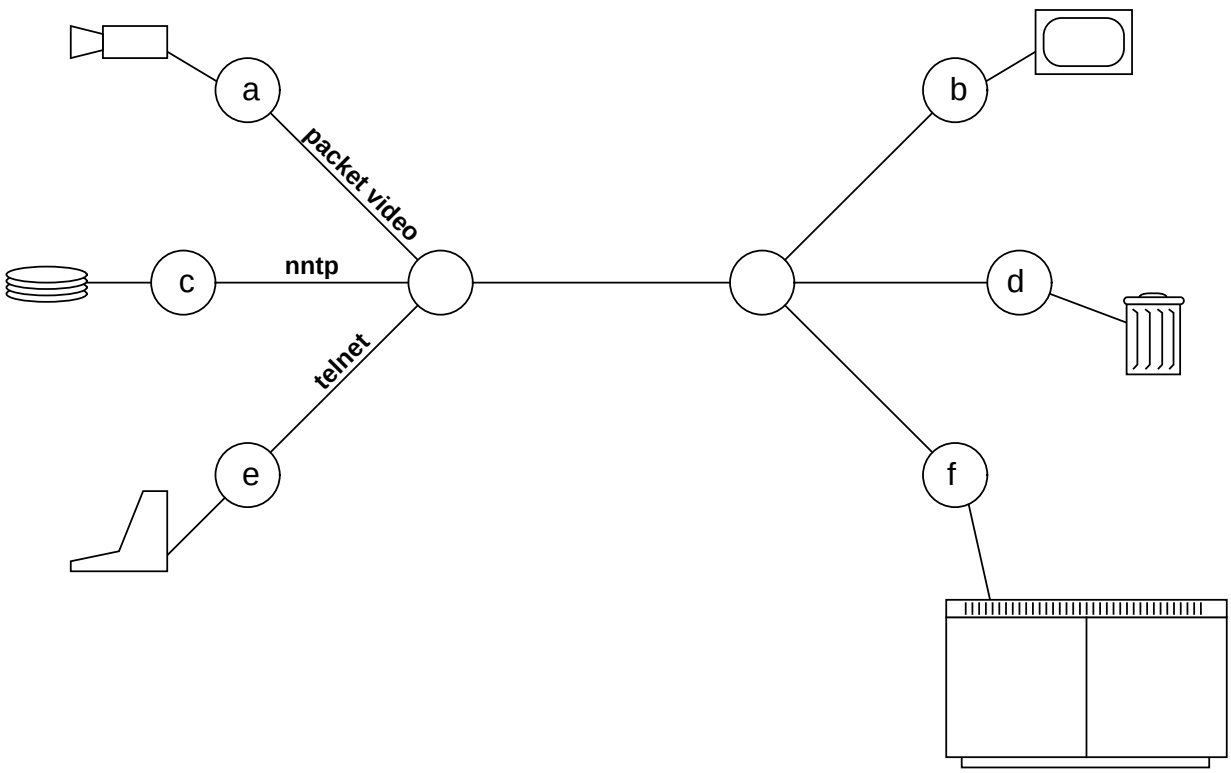
Van Jacobson
Lawrence Berkeley Laboratory
van@ee.lbl.gov

(joint work with Sally Floyd)

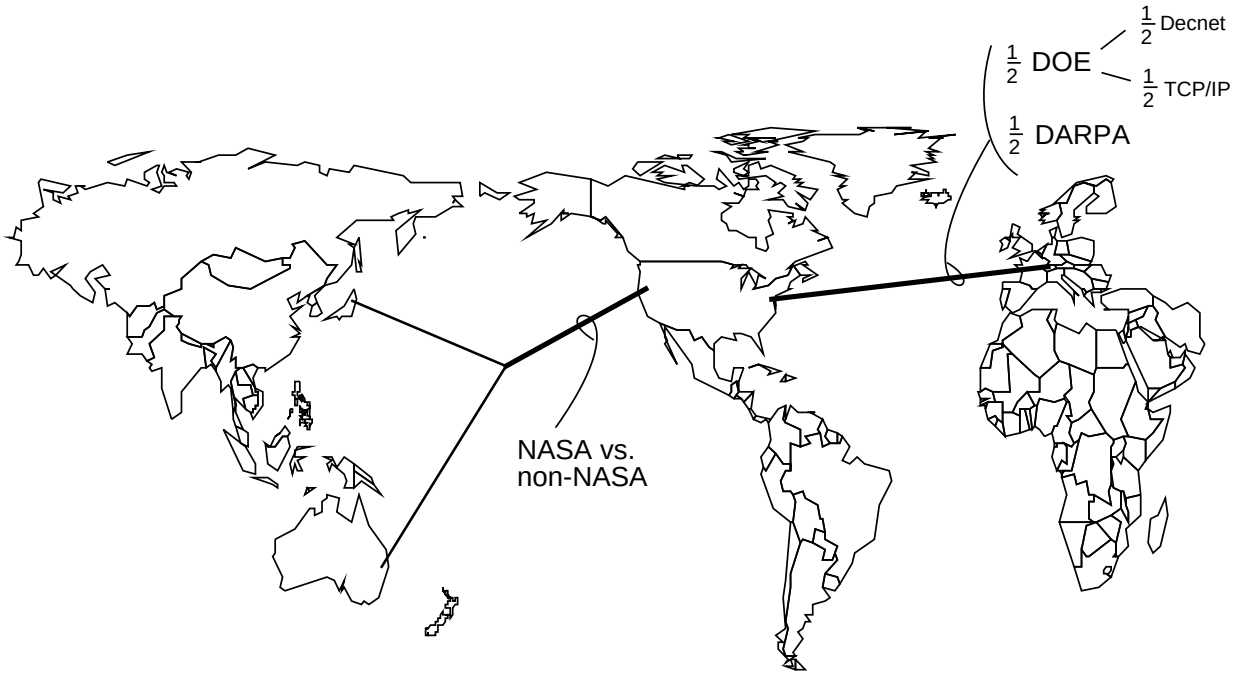
Sun Microsystems

October 12, 1993

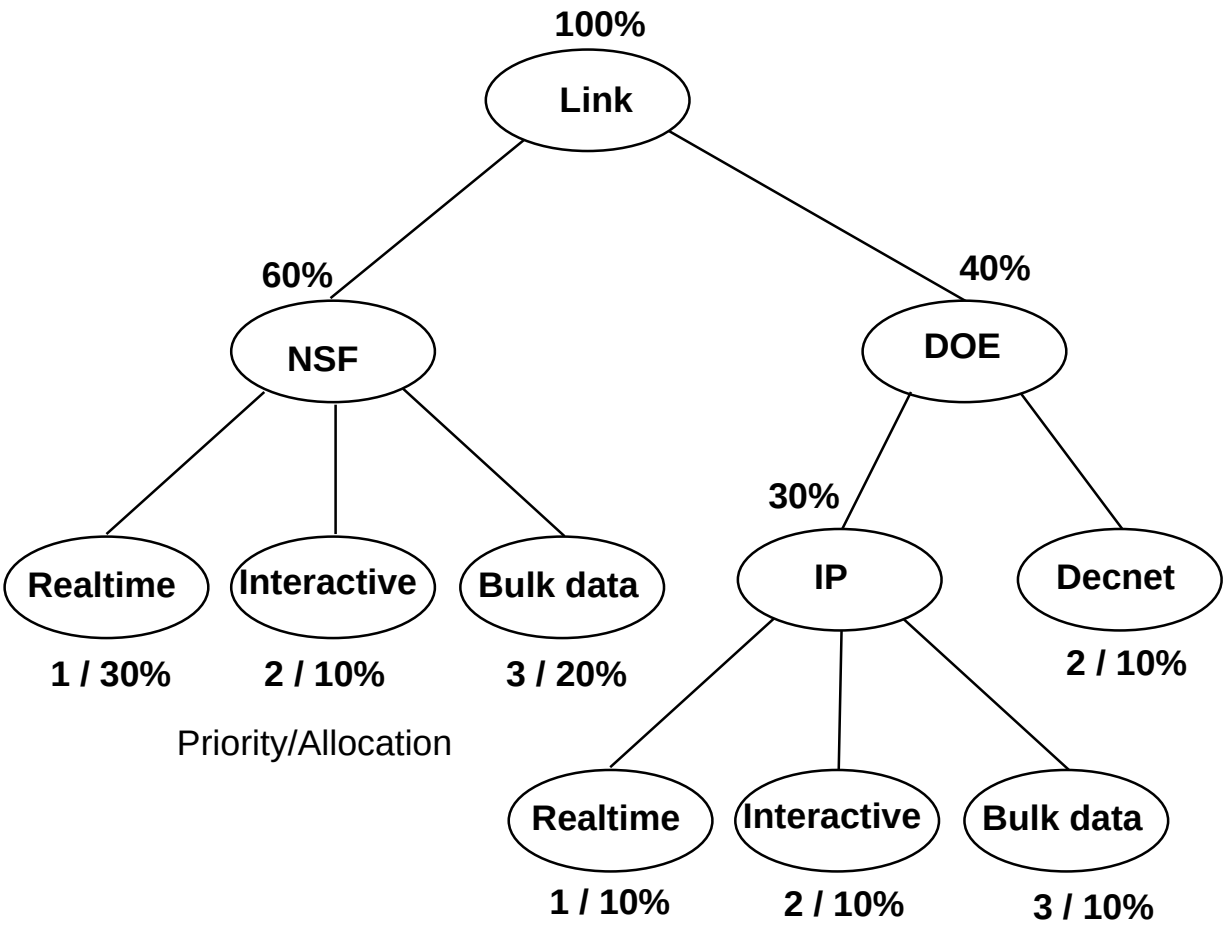
Need to support different qualities of service (real-time vs. interactive vs. bulk data)



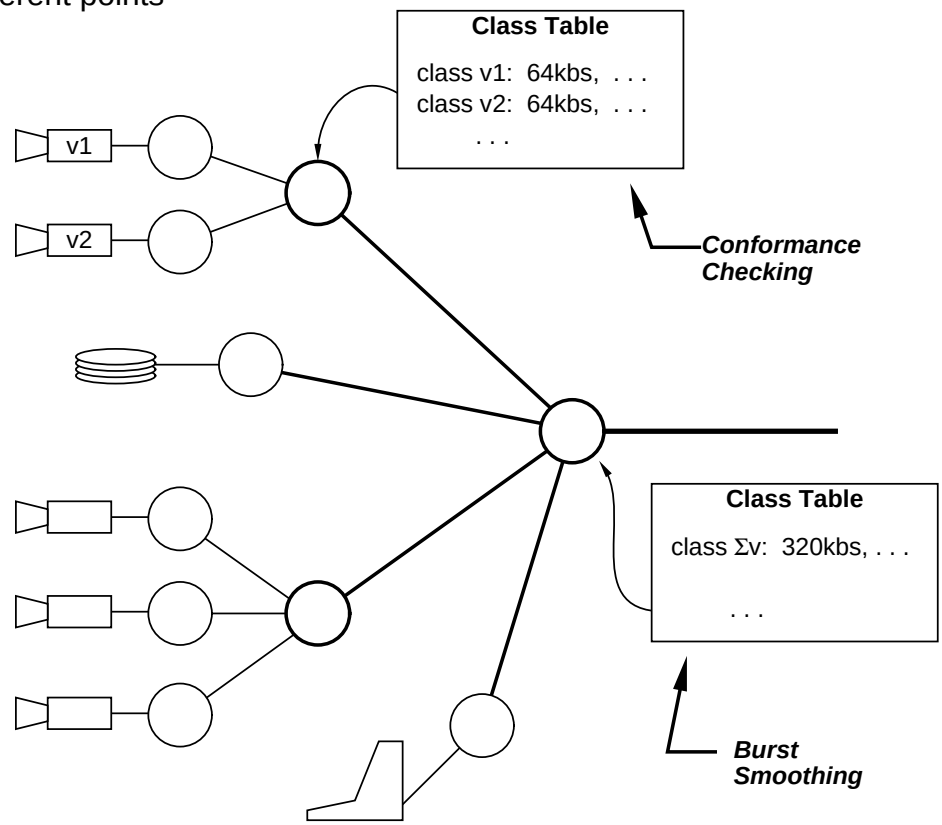
Need to accommodate ‘policy’ constraints. E.g., DOE & DARPA fund half of US–UK link. Each wants half if link loaded, wants excess if other guy isn’t using all his half. DOE also needs to divide link between DECNet & IP protocols using similar rules.



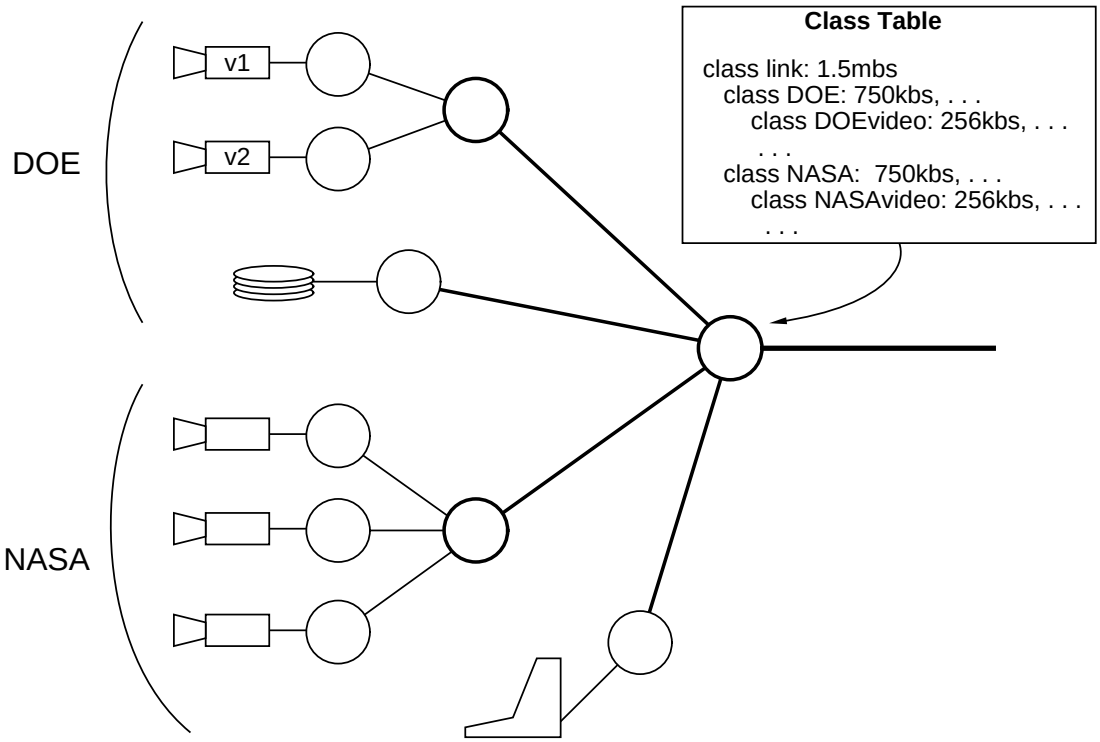
Combining policy with QOS suggests that traffic must satisfy a hierarchy of constraints:



Constraints (classes) may need to be different at different points in the network . . .



. . . and one type of traffic may
have to satisfy multiple constraints
at a single point in the network.



Goals

- Support traffic requiring bandwidth guarantees (e.g., packet voice and video).

- Support ‘quality of service’ (e.g., interactive telnet vs. bulk data ftp).

- Support flexible link sharing.

Non-goals

- Congestion control / congestion avoidance.

- 'Fair' sharing (for some definition of 'fair').

('Non-goals' doesn't mean these are non-problems—Other machinery, e.g., RED, solves these problems working inside this framework.)

Problems with ‘connections’

- Resources allocated only to transport endpoints (wrong service model, scaling problems).

- Resources are bound when transport connection established (wrong service model, scaling problems).

- State bound to a particular path through network (poor fault tolerance).

Root problem: Policy and mechanism mixed.

For a router, ‘bandwidth’, ‘delay’ and ‘conversations’ are abstractions.

Physically, routers can:

- classify packets based on their protocol headers, and
- chose to send, drop or delay any particular packet.

The success of IP over X25 taught us that a network is more flexible and scalable if its abstractions model physical reality rather than some committee’s idea of the correct way to communicate.

$\Longrightarrow$ Resource management should be canonical version of what routers can do.

Architecture

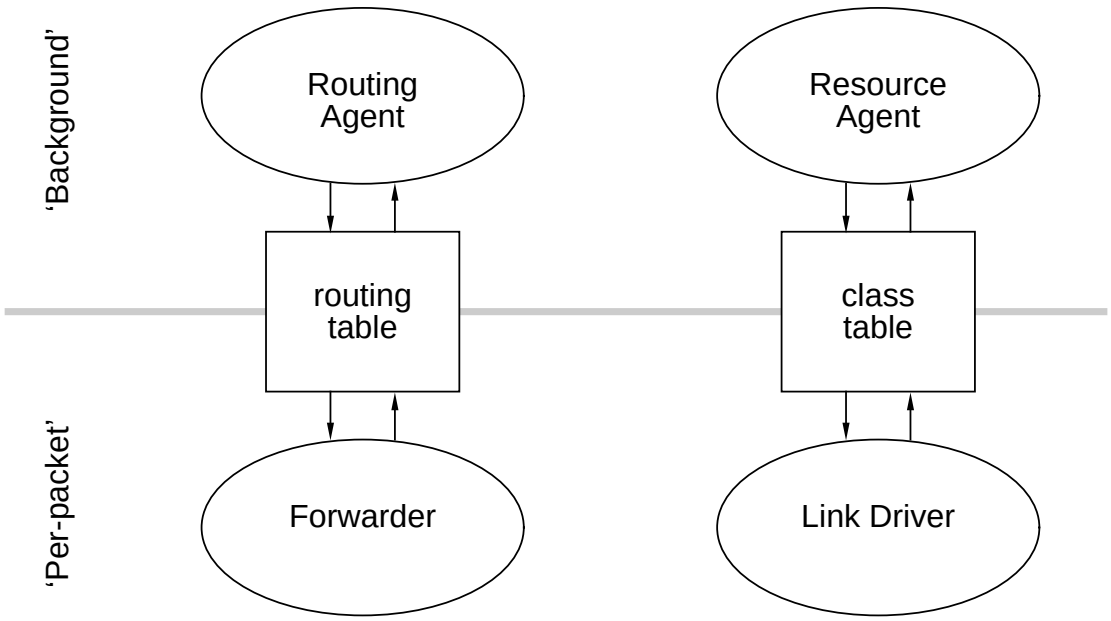
Another IP lesson is that per-packet processing (e.g., forwarding) should be separated from 'global coordination' (e.g., route exchange and route computation).

- helps performance [routes don't change per-packet]

- allows evolution [packet forwarding was well understood in 1978; in 1990 we started to dimly understand routing].

$\Longrightarrow$ RM implementation architecture should mirror the IP routing/forwarding split.

Gateway software structure (1)



One efficient mechanism that meets our goals has four pieces:

1. Classifier
2. Selector
3. Overlimit action
4. Estimator

Classifier

- Protocol dependent.

- Uses information in packet/cell to locate appropriate class structure.

- Classification can usually be done as part of normal forwarding decision.

  E.g., for IP class is carried in ‘route’ and classification is no-cost side effect of the route lookup.

Selector

- Protocol independent.
- Selects which packet to send next.
- Part of output link driver (run in response to completion interrupt, delay timer or packet arrival when link idle).

Selector's Algorithm

- Inspects classes in priority order.
- Inspects classes at same priority round-robin.
- Stops when it finds a packet to send (class under limit or can borrow from parents).
- If finds over limit class that can't borrow, invokes that class's **Overlimit Action**.

Overlimit Action

- Protocol dependent (in general).
- Routine called when a class goes over its utilization limit.
- A few 'generic' (protocol independent) actions: *delay, drop* and *reclass*.

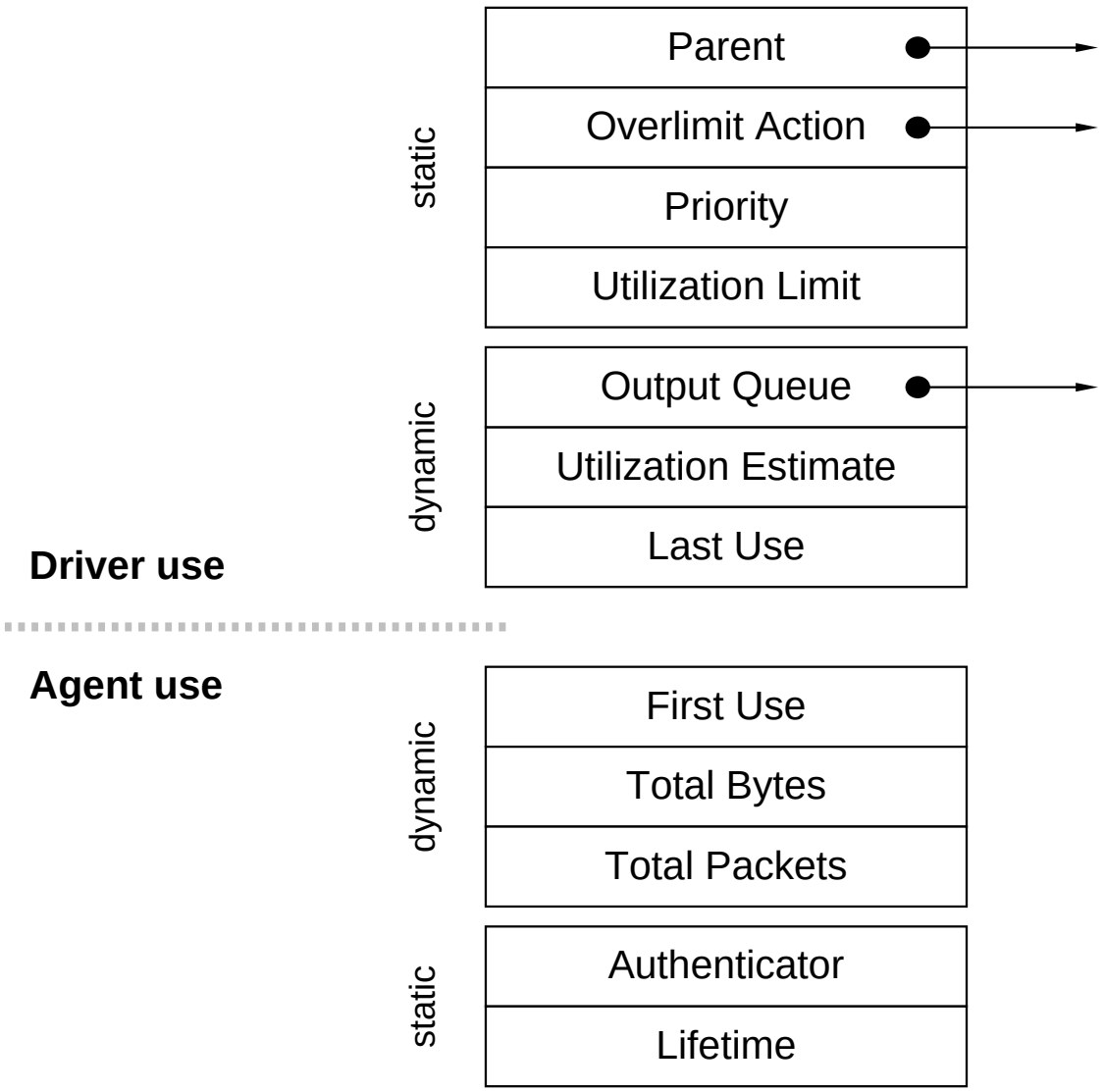
Most important action is probably *delay* (until below limit) which is used to (a) smooth bursts in real-time traffic, and (b) enforce limits at high load by rate-controlling classes.

Estimator

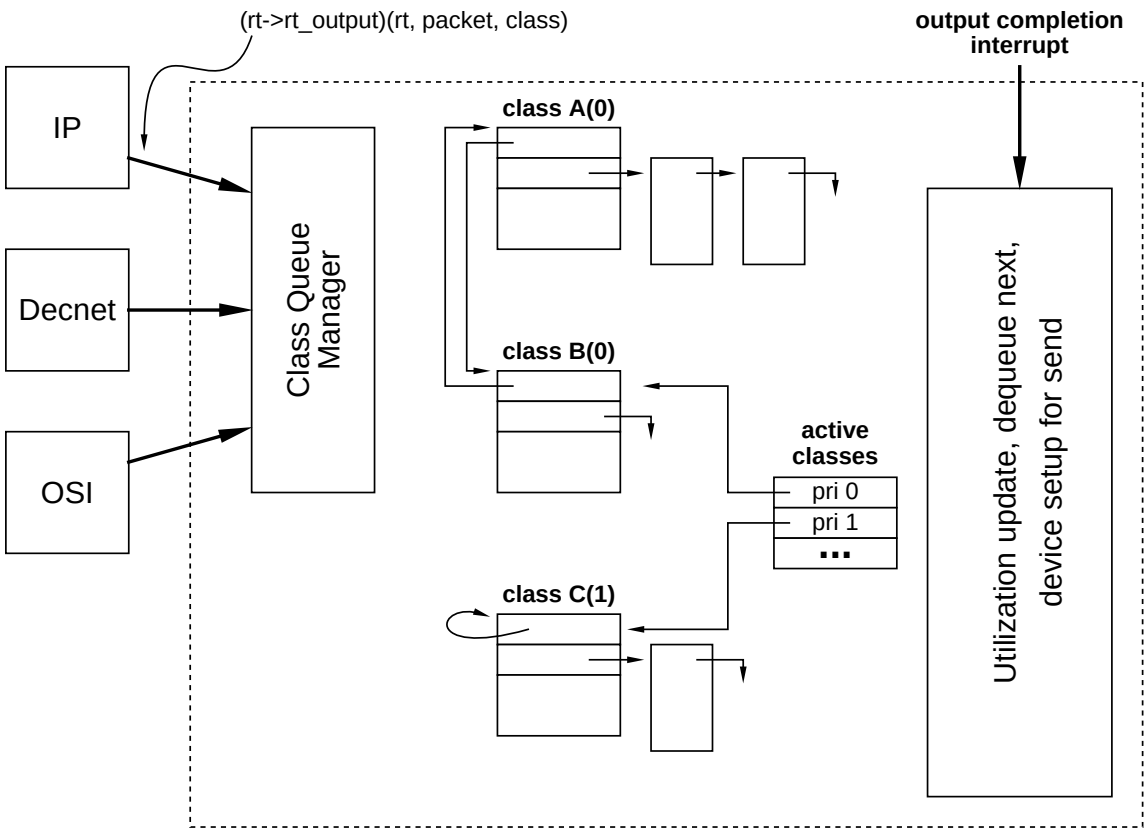
- Protocol independent.
- Part of output link driver (run in response to completion interrupt).
- Uses inter-departure time of packets in class to compute short-term average estimate of class's link utilization.
- Utilization updated for class and all its parents.

(One good estimator is very simple EWMA (low-pass filter): one shift and two adds.)

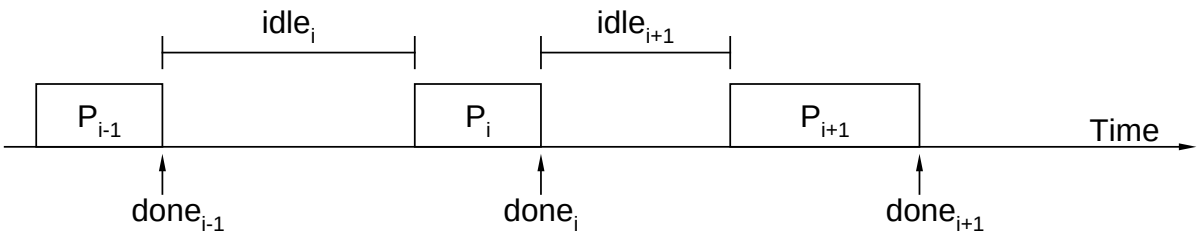
Class Table entries:
(per Traffic Class state)



Gateway software structure (2)



Computing Link Utilization (1)

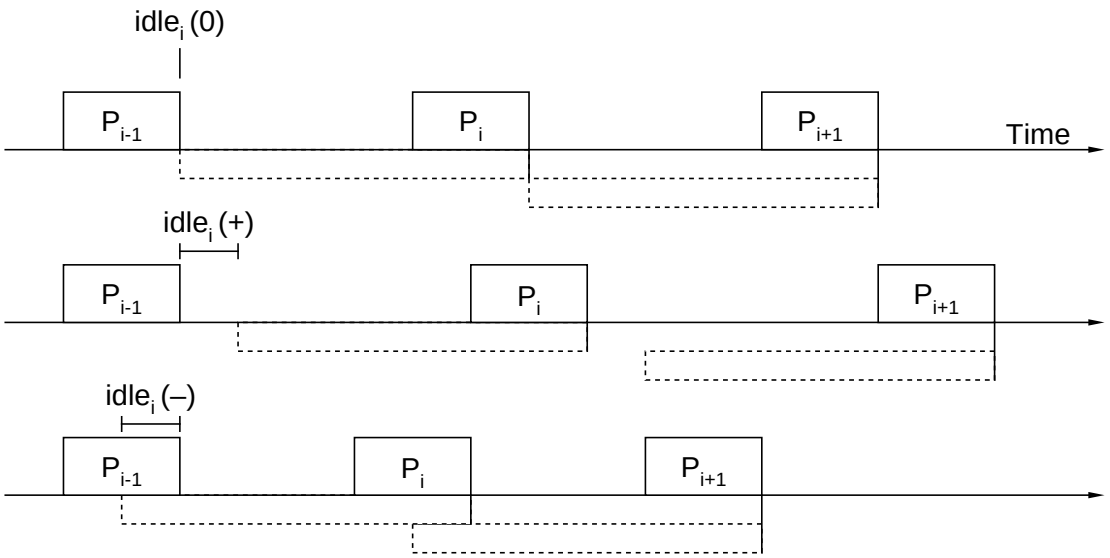


$$idle_i = done_i - done_{i-1} - (length_i / B)$$

(Instantaneous) Link idle time at completion of most recent packet:

idle = now – last – (length / B)

Computing Link Utilization (2)



Per-class (instantaneous) idle time:

$$idle = now - last_{class} - (length / B_{class})$$

Why this scheme instead of CSZ?

- This implements *all* of CSZ service model.
- This does things that CSZ can't.
- Priority and throughput can be assigned independently.
- Admissions control is optional and used, when desired, *within* a class. Priority & throughput guarantees are independent of admissions control.

Why? (cont.)

- 'Borrowing' of unused throughput is controlled by class hierarchy. (Among other things, this permits controlled link-sharing.)

- 'Guaranteed' service can be provided, if desired, by priority-1 classes.

- No per-packet or per-conversation state.

- Simple — < 200 lines of C.