Lecture #12: router architecture design

WPI CS4516 Spring 2019 D term

Instructor: Lorenzo De Carli (ldecarli@wpi.edu)

(slides include material from Christos Papadopoulos, CSU)

Structure of this lecture

- **Part 1: challenges in router design**
  - Techniques for designing routers supporting multi-Gb packet forwarding
- **Part 2 - Queue management**
  - Techniques for queue management that help ensuring fairness and avoiding congestion

Part 1 – challenges in router design

Nick Mckeown's paper

The original version of this paper appears in Business Communication Review.

WHITE PAPER:

A Fast Switched Backplane for a Gigabit Switched Router

by Nick McKeown (tel: 650/725-3641; fax: 650/725-6949; email: nickm@stanford.edu) a professor of electrical engineering and computer science at Stanford University. He received his PhD from the University of California at Berkeley in 1995. From 1986-1989 he worked for Hewlett-Packard Labs, in their network and communications research group in Bristol, England. In spring 1995, he worked for Cisco Systems as an architect of the Cisco 12000 GSR router. Nick serves as an editor for the *IEEE Transactions on Communications*. He is the Robert Noyce Faculty Fellow at Stanford and recipient of a fellowship from the Alfred P. Sloan Foundation. Nick researches techniques for high-speed networks, including high-speed Internet routing and architectures for high-speed switches. More recently, he has worked on the analysis and design of cell-scheduling algorithms, memory architectures and the economics of the Internet. His research group is currently building the Tiny Tera; an all-CMOS Terabit-per-second network switch.

(most of the material in the first part of this lecture is based on the paper above)

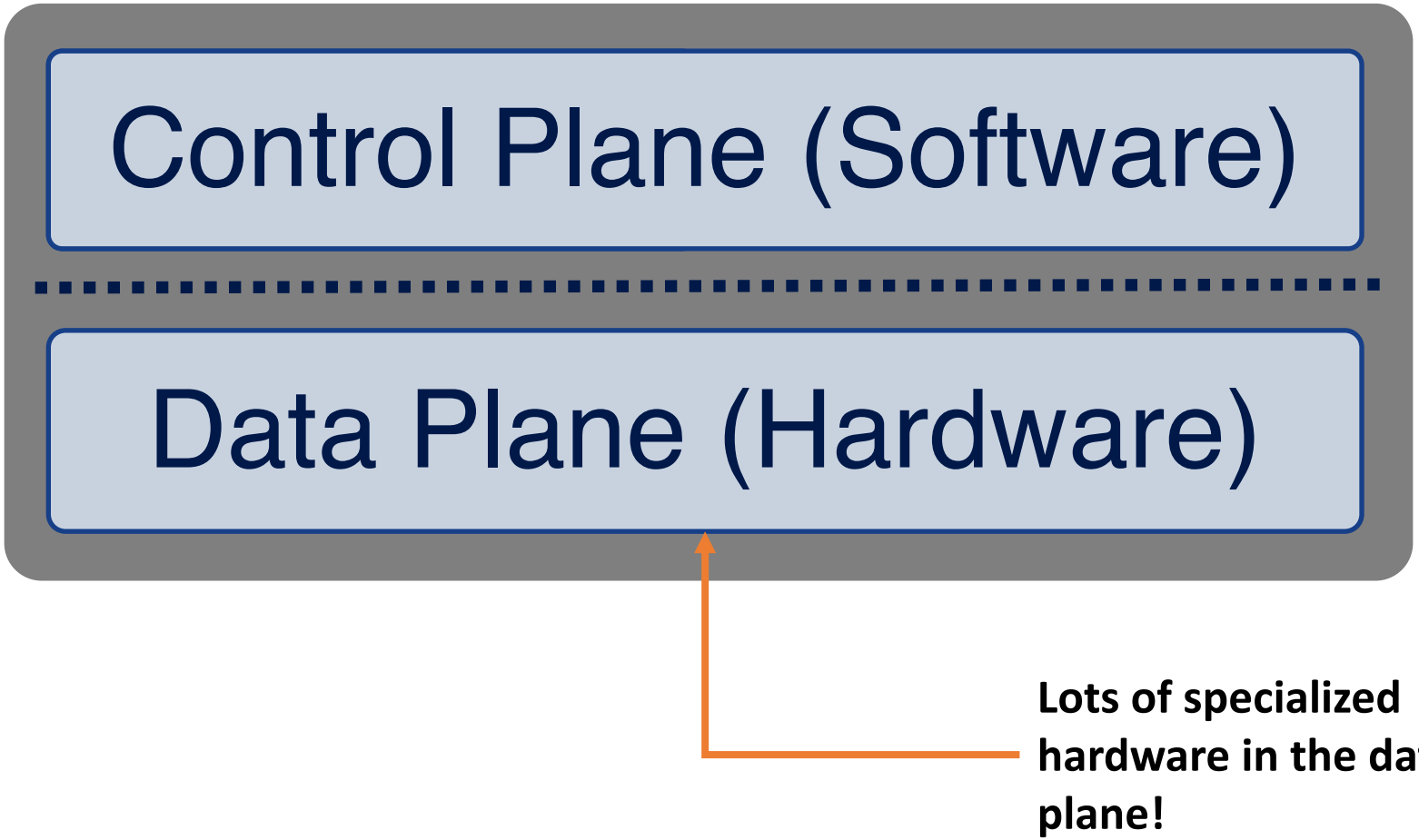
Router design requirements

- Simplifying, a router's goal is to **move incoming packets closer to their destination**
- In practice, this consists in:
  1. Receiving packets on an input interface
  2. **Selecting the appropriate output interface** based on destination IP
  3. **Moving packets to the appropriate output interface**
- The router may also perform simple packet modifications (e.g., TTL decrease)

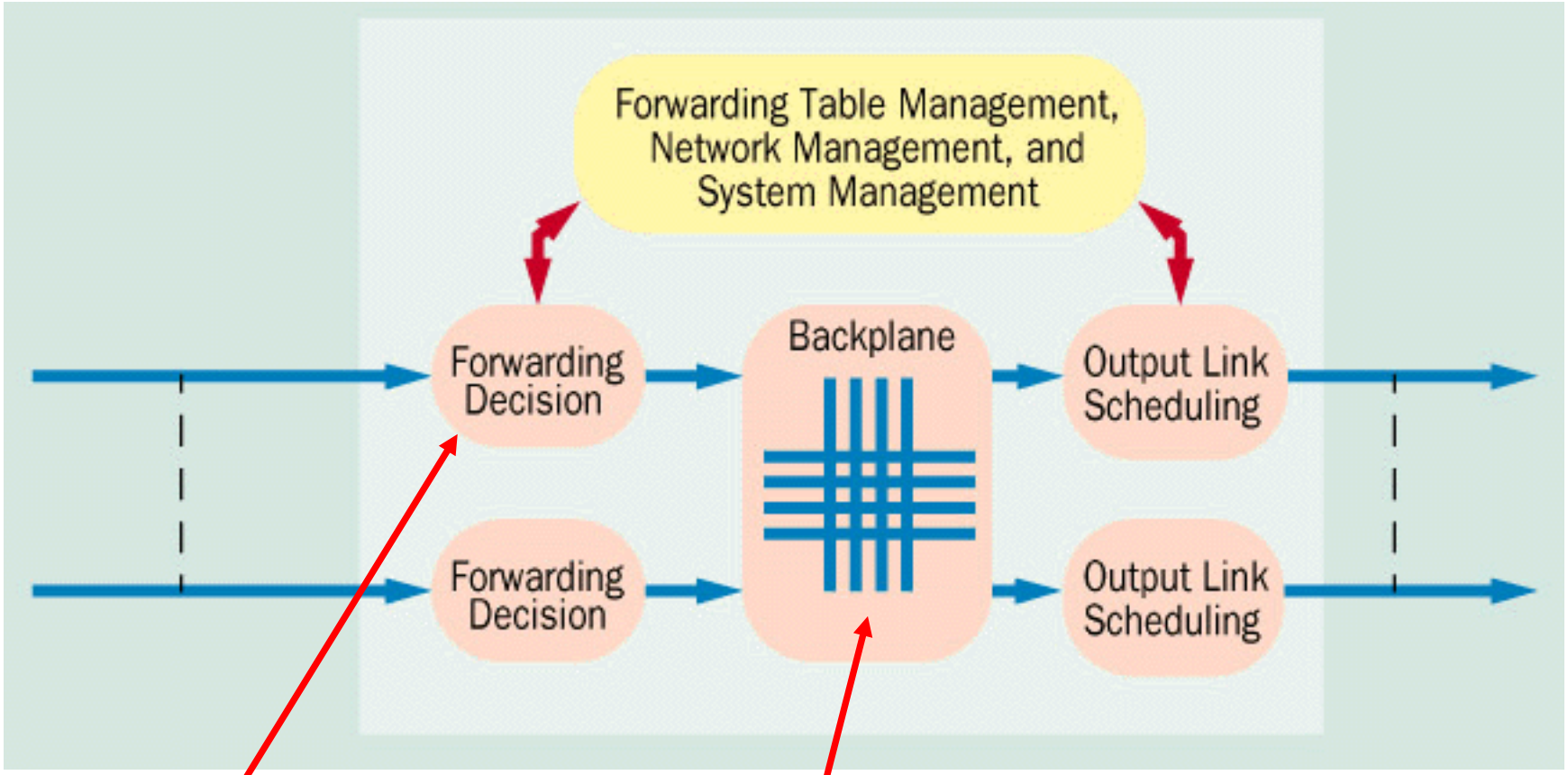
Router design requirements/2

- **Throughput:** data transfer must operate at **full B/W between all pairs of interfaces**
  - Regardless of the communication pattern!
- **Latency:** processing delay must be:
  - Kept low
  - As **predictable** as possible (to avoid introducing jitter)
- At the same time, routers have to perform **complex lookup operations** for each packet

So how do we do it?



More specifically…

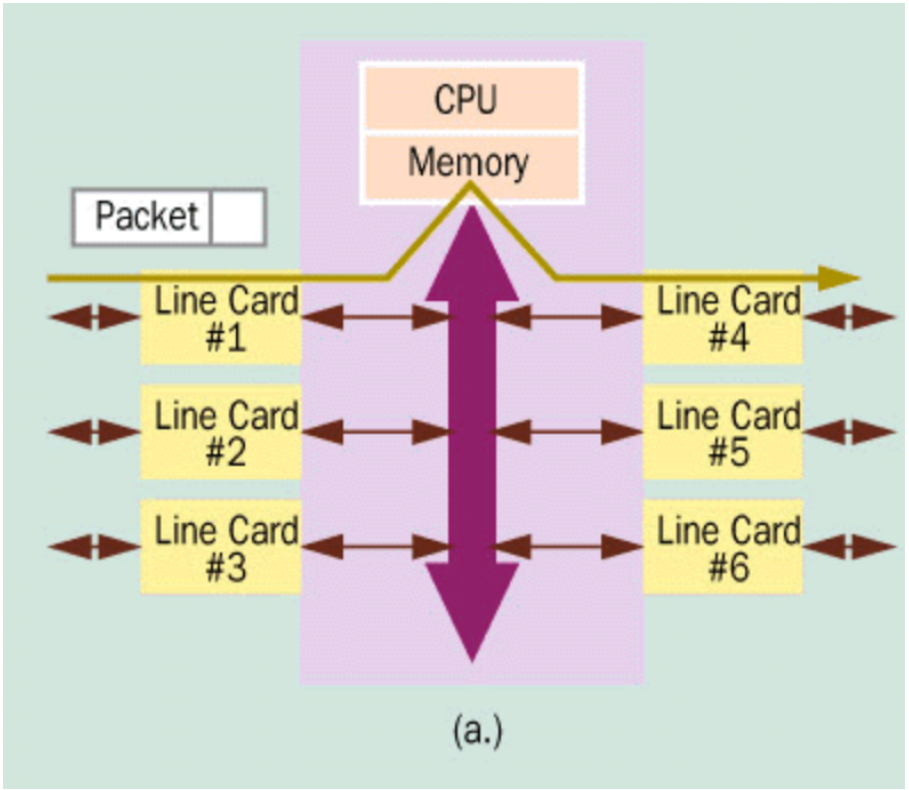


"Something" makes a decision on the packet

"Something" moves the packet towards the output

Router architecture history #1

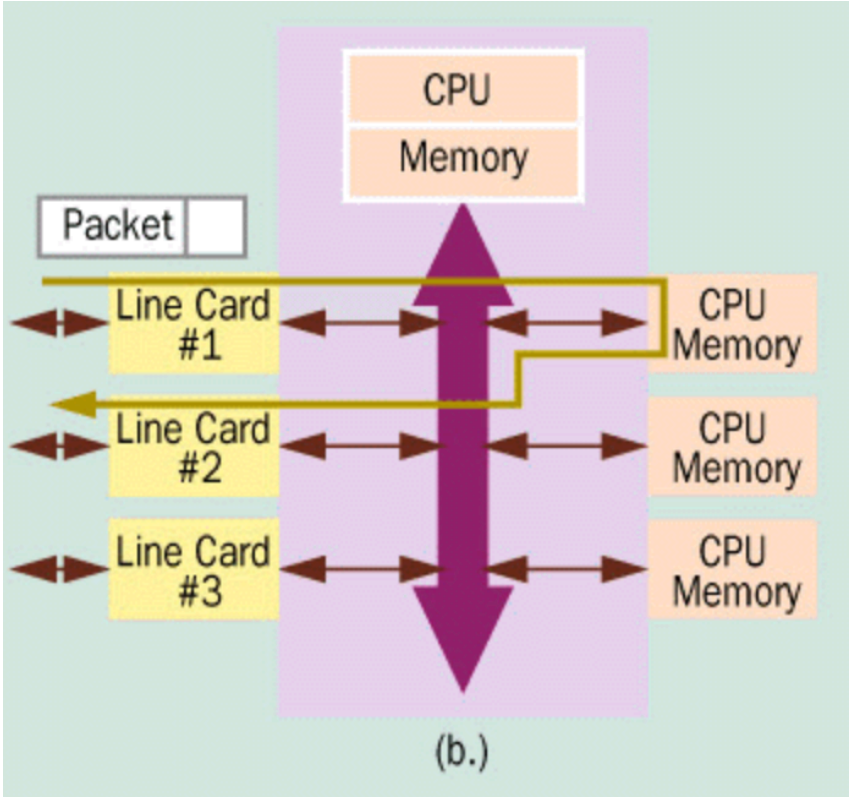
- Approach #1: **CPU-based**



- **Simple**
  - No need for specialized hardware
  - Good enough for a small network/low bandwidth
- **Does not scale**
  - Limited by CPU
  - Limited by bus bandwidth

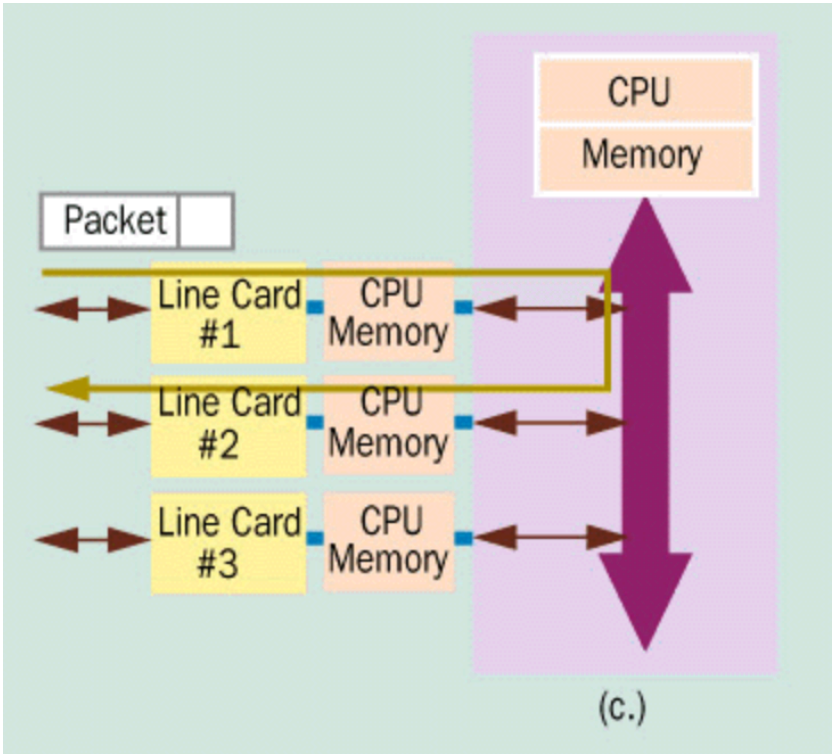
Router architecture history #2

- Approach #2: **multiple CPUs**
  - Allows **load-balancing** across multiple CPUs
  - **Still limited by bus bandwidth**



Router architecture history #3

- Approach #3: **Processors on line cards**

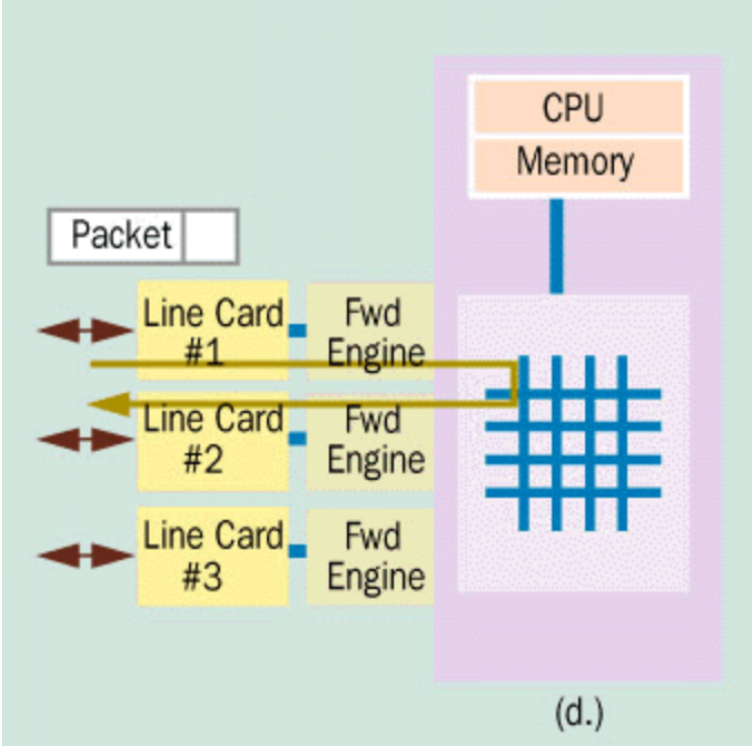


- **Streamlines internal communication**
  - Forwarding decision is made **locally**, then packet is moved
  - **Less pressure on bus bandwidth** (less data movement)
  - …alleviates the shared bus chokepoint, but **does not solve the issue**

Router architecture history #4

- Approach #4: **crossbar switch + forwarding ASIC**

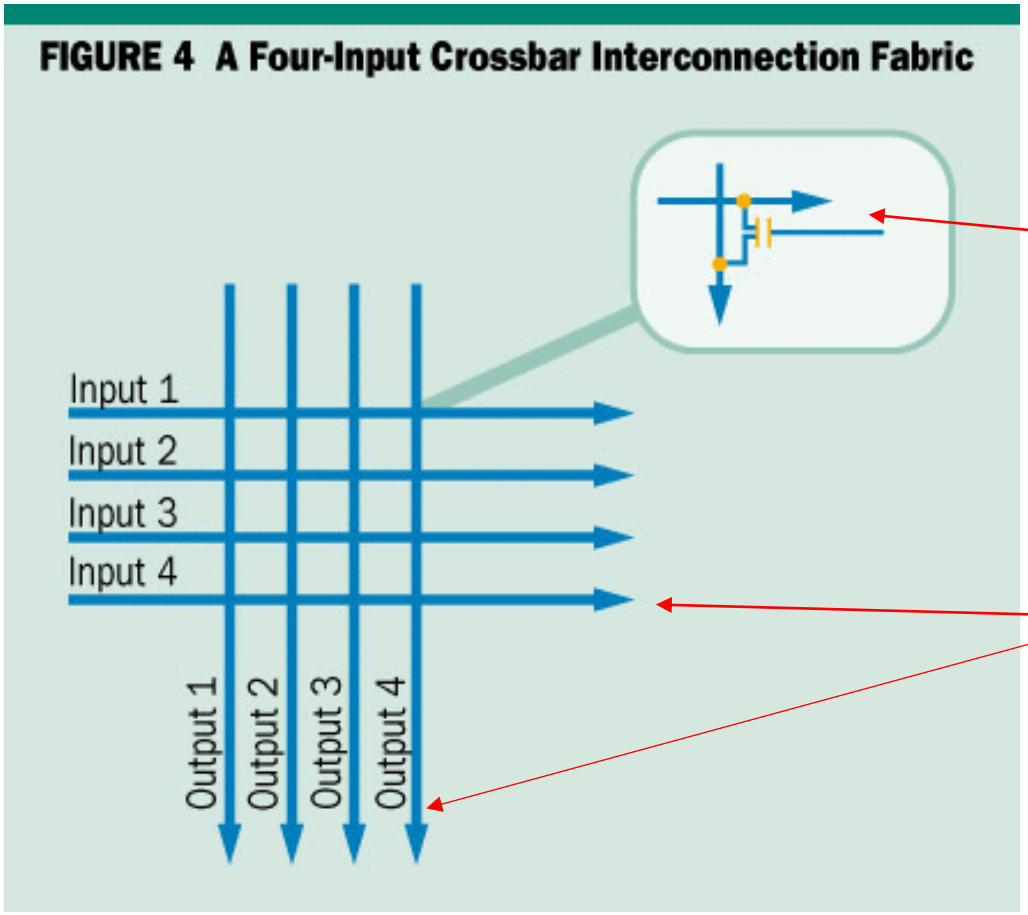
  - Replace serial bus with **specialized crossbar** allowing **multiple transactions in parallel**
  - Replaces CPU with **dedicated forwarding ASIC** (more on this in next lecture)



Crossbar switch

- High-performance **switched backplane**
- Every input has a **wire connection** to every output
  - Simplifies design, signaling compared to bus
- **Non-blocking:** every input can communicate with a different output simultaneously
- **Expensive** in terms of #components, power

Crossbar switch – how does it work?

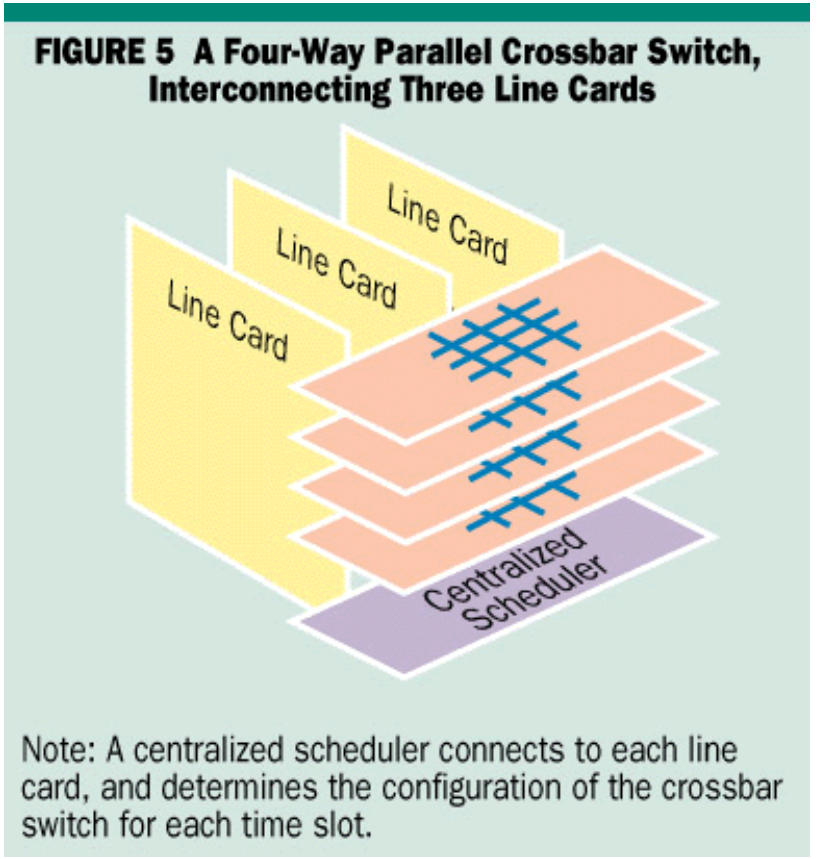


Tri-state buffers allow a scheduler to decide which input is connected to which output

Inputs & outputs have **dedicated wires**

More on crossbar switch design

- Basically, it replaces a bus with a wire
  - Much easier to ensure **high-quality signals** (limited reflections, noise, interference)
  - A single crossbar only supports 1-bit transfer – need **several** of those **operating in parallel**!



Crossbar scheduling

- While a crossbar is in principle non-blocking, it is only so if **configured appropriately** for the data waiting to be transferred
- Basic idea: divide time in **discrete slots**; have scheduling algorithm determine **optimal configuration for each slot**
- Non trivial! Algorithm must **prevent starvation**, be **fair**, and execute **quickly**!
- We'll talk about the **iSLIP** algorithm later

Variable- vs fixed-size cells

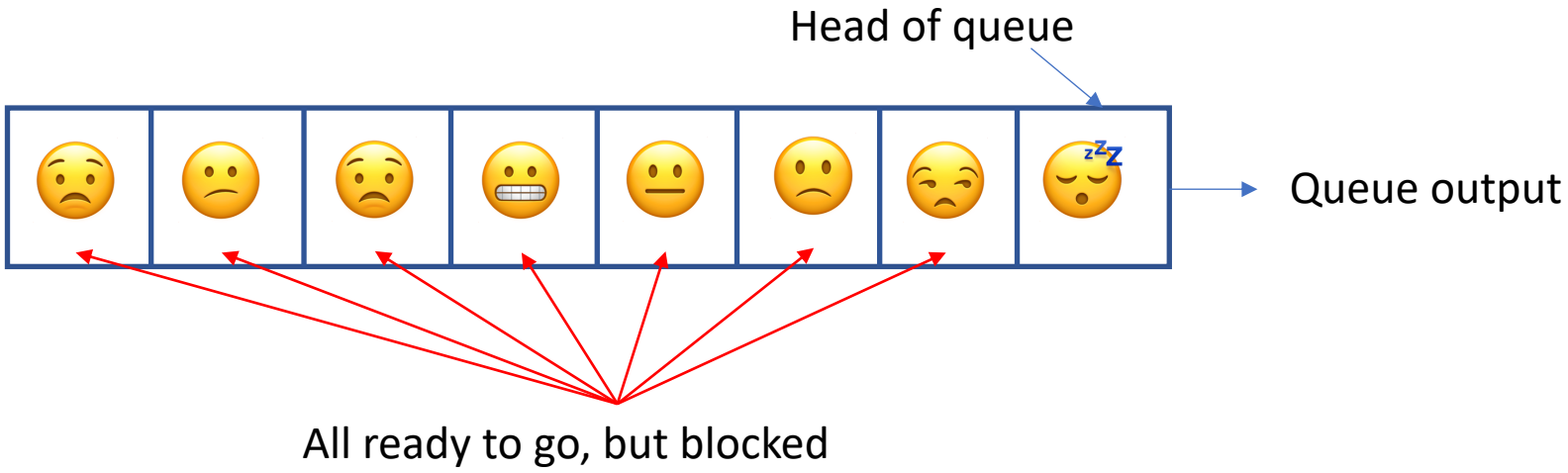
- In general, network packets have varying size between 40 and 1500 bytes
- Allowing transfers of arbitrary size would **complicate the scheduling algorithm**, since it makes **guaranteeing fairness** more complicated, and it makes it harder to figure out **when the bus is going to become available again**
- **Solution:** break packets in fixed-length cells, transfer cells

The problem with throughput

- Even with a switched backplane that can support full B/W across all pairs of interfaces, there are issues limiting throughput
- If care is not taken, certain incoming traffic patterns can cause bandwidth to be used inefficiently
- Most significant problem: **head-of-line (HOL) blocking**

Head-of-line blocking

- Core issue in packet scheduling
- **The problem:**
  - consider a FIFO queue with *N* packets waiting to be sent.
  - Suppose the first packet is blocked because assigned to an output interface which is busy, while the other *N-1* packets are assigned to available output interface and could be transmitted immediately.
  - **Because the FIFO nature of the queue, all *N* packets are blocked.**



Head-of-line blocking /2

- HOL can significantly affect router throughput by **unnecessarily penalizing packets queued behind blocked ones**
- Solving the issue requires the ability to **“peek” in the queue beyond the first element** (e.g. iSlip algorithm)

Input and output blocking

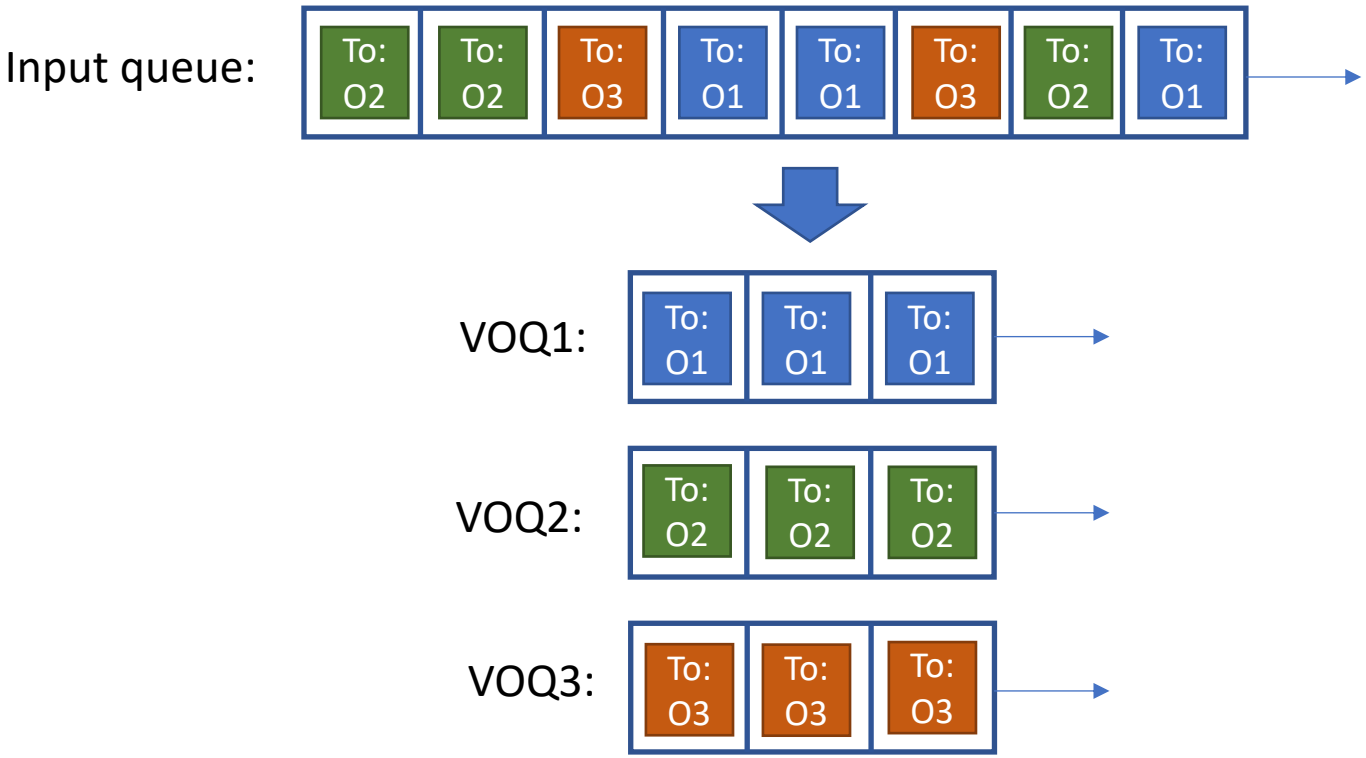
- Event if HOL is circumvented, other issues may still appear:
  - **Input blocking:** multiple cells can be transferred from a given input, but the input interface can put at most one cell on the crossbar in each time slot
  - **Output blocking:** multiple cells from different inputs need to be transferred to the same output, but the output can receive from the crossbar only one cell per time slot
  - These issue can **increase delay** of queued packets **in unpredictable ways**!

The iSlip algorithm

- **Crossbar switch scheduling algorithm** by Nick McKeown
- Designed to achieve **efficient crossbar utilization** in high-performance routers
- Insight: partition the queues at each input interface into **virtual queues** (one virtual queue for each output)
- Used in several models of Cisco routers

Virtual Output Queues (VOQ)

- Prerequisite for iSlip: each input must maintain separate queues for each output (use **virtual output queues**)



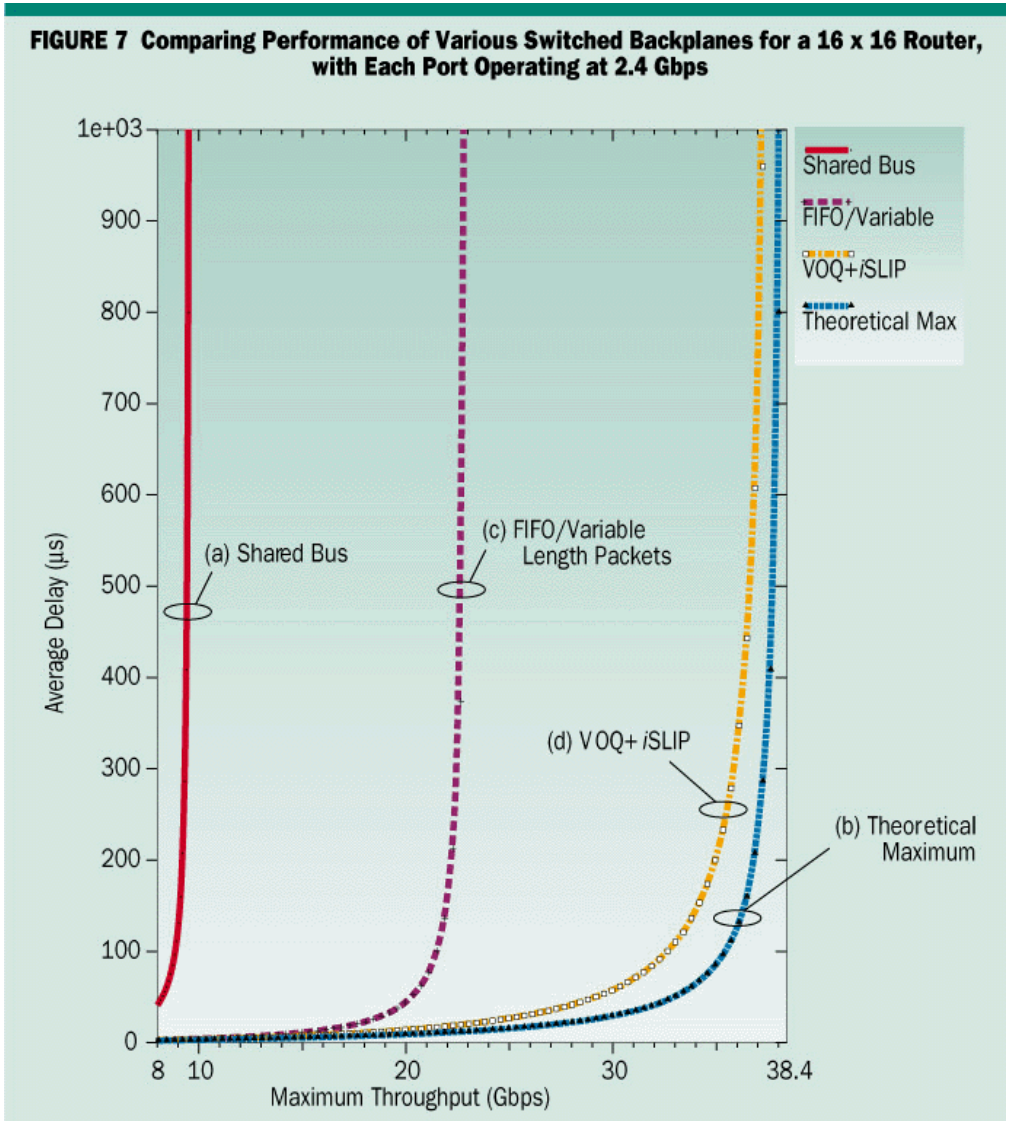
(Simplified) iSlip overview

1. **Request:** every input issues a request for every output for which there is a queued cell
2. **Grant:** each output examines all the requests received in step 1 and chooses the one with the highest priority
   - Priority is given by a round-robin schedule (once an input has been serviced, it becomes the lowest-priority one)
3. **Accept:** each input examines all granted requests and chooses the one with the highest priority (same concept of priority applies here)

iSlip properties

- **High throughput:** 100% throughput achievable for uniform/uncorrelated arrival (benign, random traffic pattern)
- **Starvation-free:** every input queue is eventually serviced
- **Terminates** at most after *N* iterations
- Designed to be implementable as **ASIC**

iSlip results



Dealing with delays

- Scheduling algorithms like iSlip can drive achievable bandwidth near the theoretical maximum, however they **do not explicit control delays**
- Incoming packets need to contend at:
  - **The input interface**, since each input has multiple VOQs competing for service
  - **The output interface**, since there may be multiple input interfaces wanting to transfer cells to the same output interface
- **Contention** depends on the packets queues at each interface and it is very hard to model/predict!

Dealing with delays /2

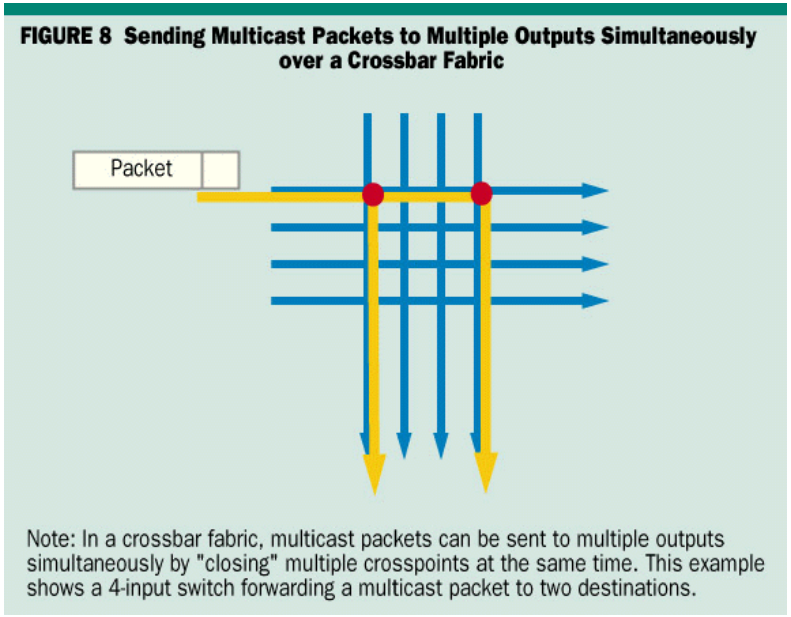
- **Solution 1:** good ol' **priority classes**
  - Further split each VOQ in four **queues w/ different levels of priority**
  - Service high-priority queues first
  - Can be used to implement QoS
- **Solution 2: speedup**
  - **Use a higher B/W on crossbar links than on input/output ports**
  - Allows multiple cells to be transferred internally in the time it takes to receive or transmit a cell externally
  - Obviously beneficial, but **expensive**!

Multicast support

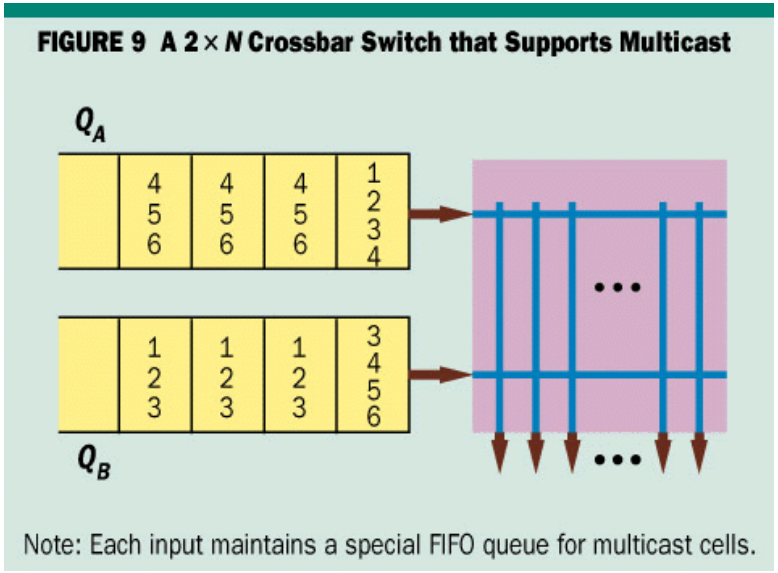
- **Multicast** requires the ability to forward one input cell to multiple output
- Naïve implementation: separately copy input cell to every output port
  - **Very inefficient!**
- Can you think of something better?

(Better multicast suppot)

- A crossbar switch can **naturally implement multicast**:

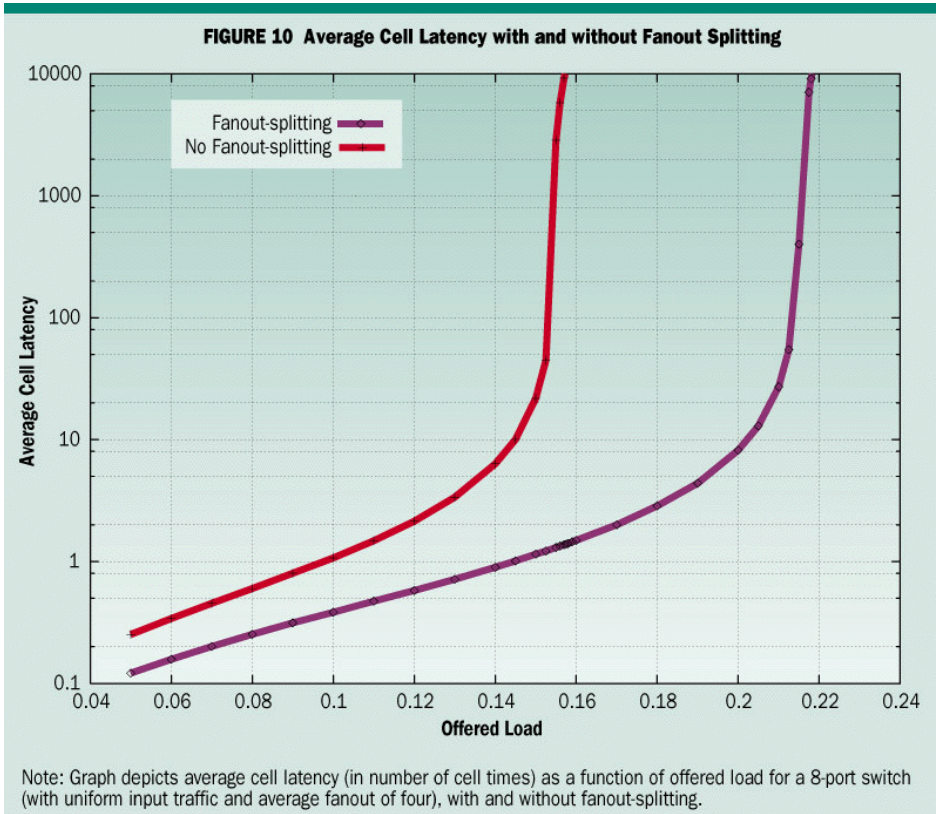


- To avoid HOL blocking due to multicast cells, these have their **own queues**:

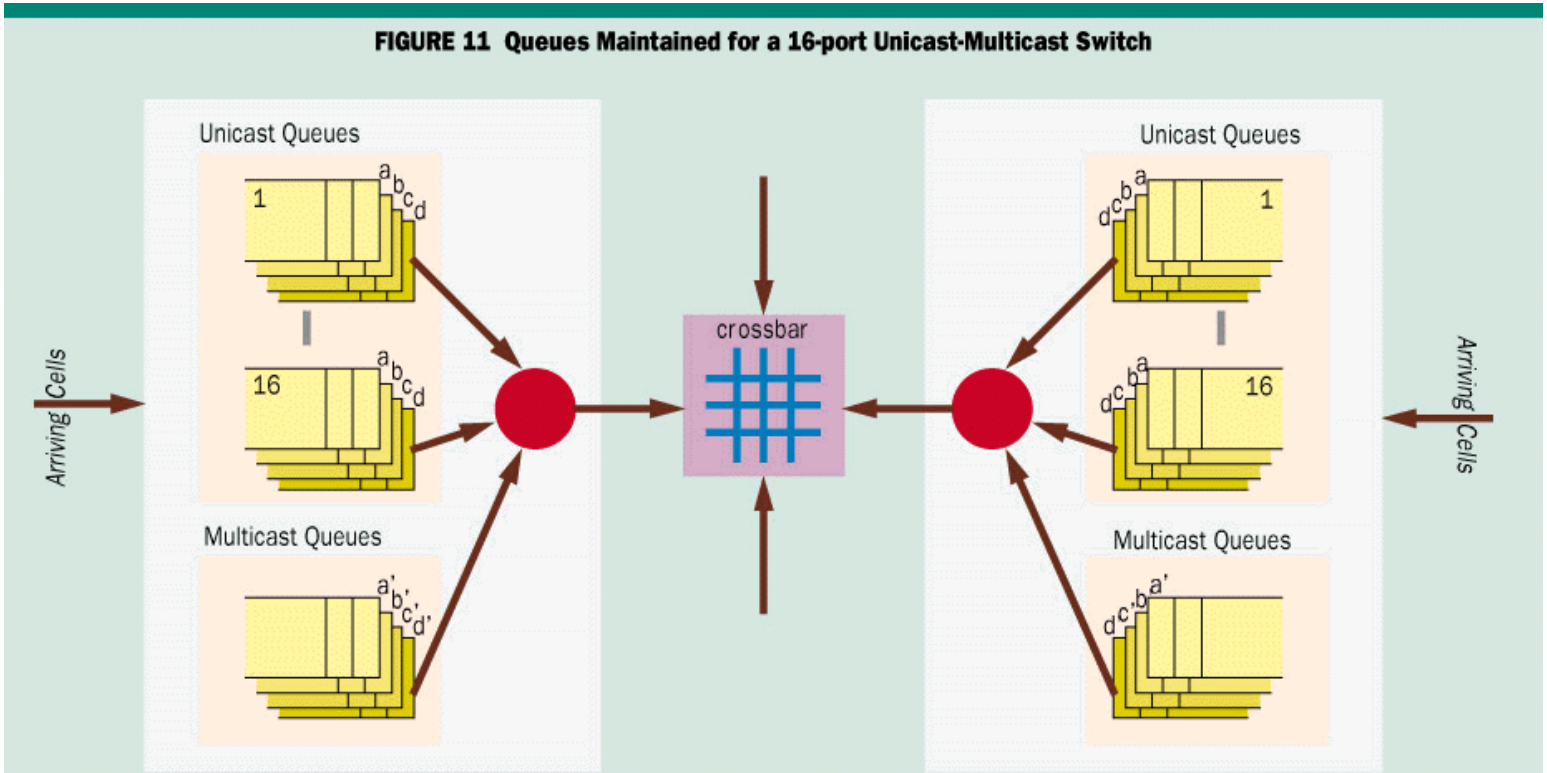


Multicast – fanout splitting

- Particularly for cells that need to be multicasted over a large number of output, waiting for all outputs to be available may entail **significant delay**
- **Better solution:** allow a multicast cell to be transmitted over crossbar **over multiple periods**
  - In every period, transfer cell to as many available output as possible – but do not require that all outputs are available



Putting it all together



Part 2 – queue management in routers

Congestion Control vs. Resource Allocation

- One of a network's key role is to **allocate its transmission resources to users or applications**
- Two sides of the same coin
  - Let network do resource allocation (e.g., virtual circuits)
    - Difficult to do allocation of distributed resources
    - Can be wasteful of resources
  - Let sources send as much data as they want
    - Then recover from congestion when it occurs
    - **Easier to implement, but may lose packets**
- **It is still useful if the network attempts to balance resource usage**

Connectionless Flows

- How can a connectionless network allocate anything to a user?
  - It doesn't know about users or applications!
- **Flow:**
  - a sequence of packets between same source - destination pair, following the same route
- **Flow is visible to routers** - it is **not a channel**, which is an end-to-end abstraction
- Routers may maintain **soft-state** for a flow

Flow control - Taxonomy

- Router-centric v.s. Host-centric
  - **router-centric**: address the problem from inside network - routers decide what to forward and what to drop
  - **host centric**: address problem at the edges - hosts observe network conditions and adjust behavior
    - not always a clear separation: hosts and routers may collaborate, e.g., routers advise hosts

Taxonomy /2

- Reservation-based v.s. Feedback-based
  - **Reservations**: hosts ask for resources, network responds yes/no
    - implies router-centric allocation
    - E.g., RSVP protocol
  - **Feedback**: hosts send with no reservation, adjust according to feedback
    - either router or host centric: explicit (e.g., ICMP source quench) or implicit (e.g., loss) feedback

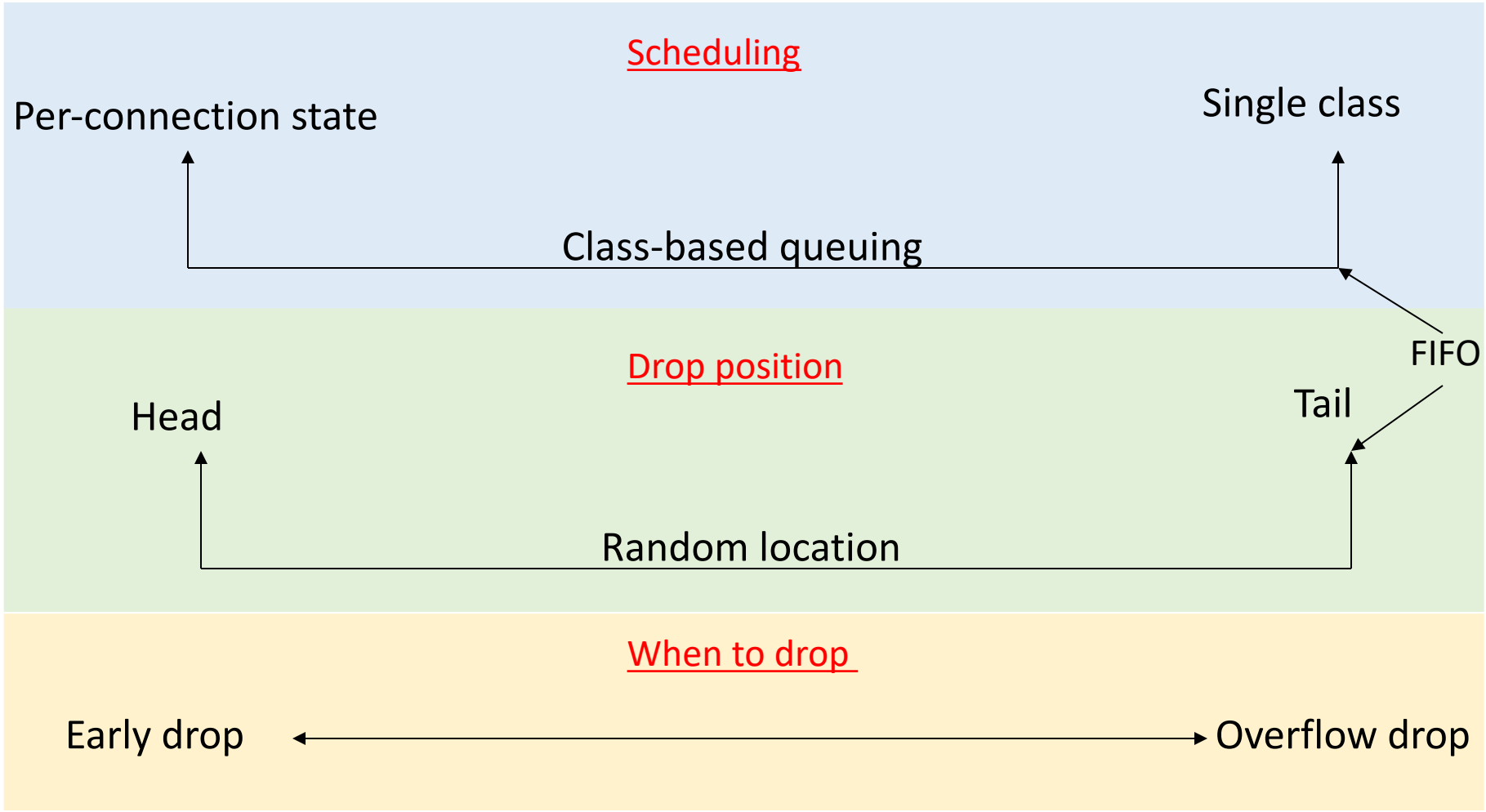
Service Models

- **Best-effort networks**
  - Mostly host-centric, feedback, window based
  - TCP is *the* example
- Networks with **flexible Quality of Service**
  - Router-centric, reservation, rate-based

Queuing Disciplines

- Each router MUST implement some **queuing discipline** regardless of what the resource allocation mechanism is
- Queuing allocates bandwidth, buffer space, and promptness:
  - **bandwidth**: which packets get transmitted
  - **buffer space**: which packets get queued/dropped
  - **promptness**: when packets get transmitted

Dimensions


Scheduling
Per-connection state
Single class
Class-based queuing
FIFO
Drop position
Head
Tail
Random location
When to drop
Early drop
Overflow drop

FIFO Queuing

- **FIFO: first-in-first-out** (or FCFS: first-come-first-serve)
- Arriving packets get **dropped when queue is full** regardless of flow or importance - implies drop-tail
- Important distinction:
  - FIFO: **scheduling discipline** (which packet to serve next)
  - Drop-tail: **drop policy** (which packet to drop next)

FIFO Queuinq /2

- **FIFO + drop-tail is the simplest queuing algorithm**
  - used widely in the Internet
- Leaves responsibility of congestion control to edges (e.g., TCP)
- FIFO lets large user get more data through but shares congestion with others
  - does not provide **isolation** between different flows

But what does “fairness” mean?

- In this discussion we are going to use **max-min fairness**:
  - Suppose we have an amount μ of a resource R to be shared between N users
  - We define the **fair share** of r as $\mu_f = \mu / N$
  - Every user requests a certain amount of R; ρ_i is the amount of resource requested by the *i-th* user
  - **Max-min fairness is achieved if:**
    - All users requesting an amount $\rho_i \leq \mu_f$ receive the **desired amount**
    - The remaining amount of resource is **distributed evenly** among all other users requesting $\rho_i > \mu_f$

Max-min fairness example

- Suppose I have a link bandwidth of 10Kb/s to distribute across 5 users
- Fair share $\mu_f = 10\ Kb/s\ /\ 5 =$ ***2 Kb/s***

Requested:

A: 1 Kb/s,
B: 2 Kb/s
C: 20 Kb/s
D: 1 Kb/s
E: 4 Kb/s

Assigned:

A: 1 Kb/s,
B: 2 Kb/s
C: 3 Kb/s
D: 1 Kb/s
E: 3 Kb/s

Q: is it possible to enforce fairness in routers?

Analysis and Simulation of a Fair Queueing Algorithm

Alan Demers
Srinivasan Keshav[U]
Scott Shenker

Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304

(Originally published in Proceedings SIGCOMM ë89,
CCR Vol. 19, No. 4, Austin, TX, September, 1989, pp. 1-12)

Abstract

We discuss gateway queueing algorithms and their role in controlling congestion in datagram networks. A fair queueing algorithm, based on an earlier suggestion by Nagle, is proposed. Analysis and simulations are used to compare this algorithm to other congestion control schemes. We find that fair queueing provides several important advantages over the usual first-come-first-serve queueing algorithm: fair allocation of bandwidth, lower delay for sources using less than their full share of bandwidth, and protection from ill-behaved sources.

1. Introduction

Datagram networks have long suffered from performance degradation in the presence of congestion [Ger80]. The rapid growth, in both use and size, of computer networks has sparked a renewed interest in methods of congestion control [DEC87abcd, Jac88a, Man89, Nag87]. These methods have two points of implementation. The first is at the source, where flow control algorithms vary the rate at which the source sends packets. Of course, flow control algorithms are designed primarily to ensure the presence of free buffers at the destination host, but we are more concerned with their role in limiting the overall network traffic. The second point of implementation is at the gateway. Congestion can be controlled at gateways through routing and queueing algorithms. Adaptive routing, if properly implemented, lessens congestion by routing packets away from network bottlenecks. Queueing algorithms, which control the order in which packets are sent and the usage of the gatewayís buffer space, do not affect congestion directly, in that they do not change the total traffic on the gatewayís outgoing line. Queueing algorithms do, however, determine the way in which packets from different sources interact with each other which, in turn, affects the collective behavior of flow control algorithms. We shall argue that this effect, which is often ignored, makes queueing algorithms a crucial component in effective congestion control.

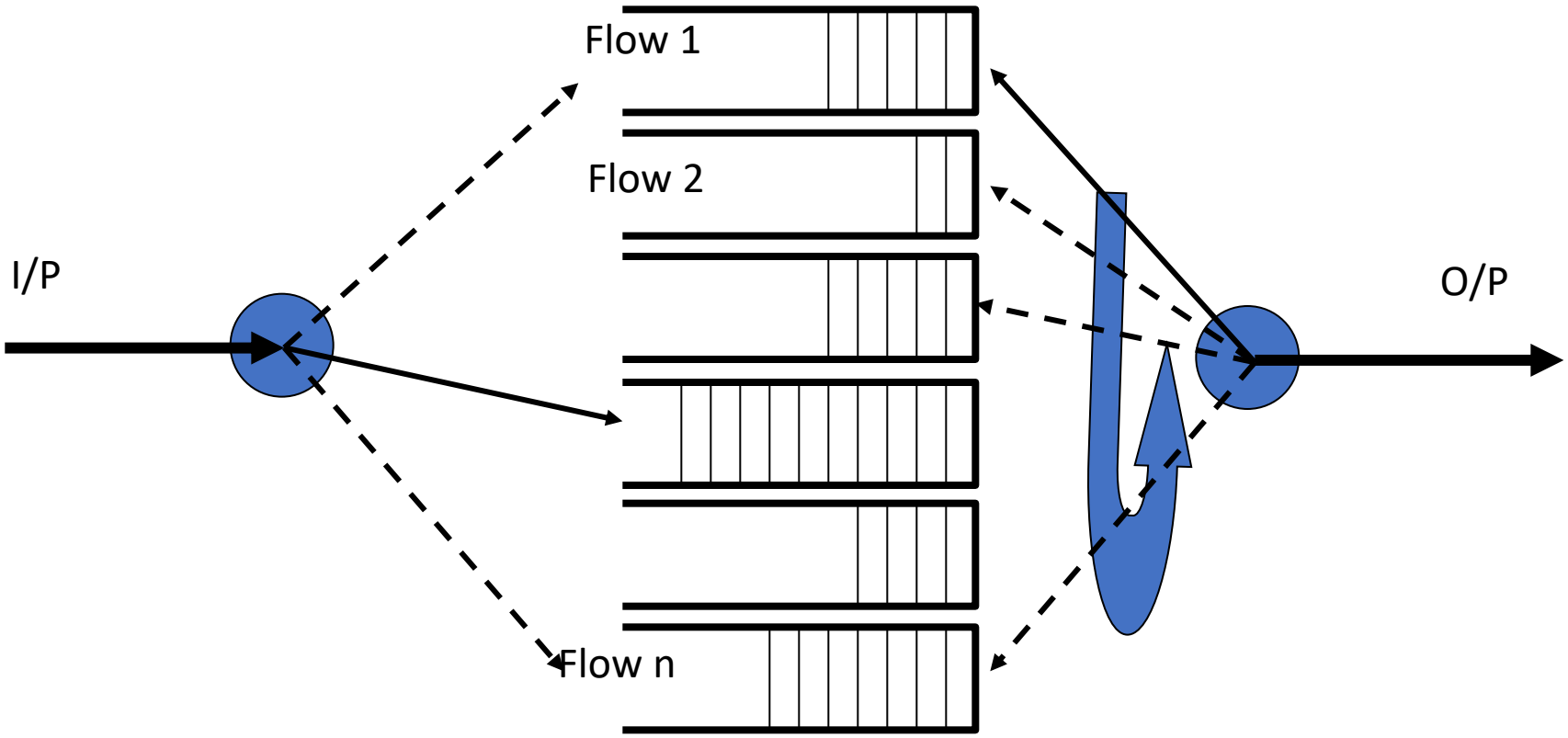
Queueing algorithms can be thought of as allocating three nearly independent quantities: bandwidth *which* packets get *transmitted*), promptness (*when* do those packets get *transmitted*), and buffer space (*which* packets are *discarded* by the gateway). Currently, the most common queueing algorithm is first-come-first-serve (FCFS). FCFS queueing essentially relegates all congestion control to the sources, since the order of arrival completely determines the bandwidth, promptness, and buffer space allocations. Thus, FCFS inextricably intertwines these three allocation issues. There may indeed be flow control algorithms that, when universally implemented throughout a network with FCFS gateways, can overcome these limitations and provide reasonably fair and efficient congestion control. This point is discussed more fully in Sections 3 and 4, where several flow control algorithms are compared. However, with todayís diverse and decentralized computing environments, it is unrealistic to expect universal implementation of any given flow control algorithm. This is not merely a question of standards, but also one of compliance. Even if a universal standard such as ISO [ISO86] were adopted, malfunctioning hardware and software could violate the standard, and there is always the possibility that individuals would alter the algorithms on their own machine to improve their performance at the expense of others. Consequently, congestion control algorithms should function well even in the presence of ill-behaved

[U] Current Address: University of California at Berkeley

Fair Queuing

- Main idea:
  - maintain a **separate queue** for each flow through router
  - router services queues in **Round-Robin fashion**
- Changes interaction between packets from different flows
  - Provides **isolation between flows**
  - Ill-behaved flows **cannot starve well-behaved flows**
  - **Allocates** buffer space and bandwidth **fairly**

FQ Illustration



Issues

- **What constitutes a user?**
  - **Several granularities** at which one can express flows:
    - **Source (= source IP address):** restricts sources which consume high bandwidth (e.g. file storage service)
    - **Receiver: (= destination IP address)** expose nodes to denial of service (just send a node unwanted traffic to fill its share)
    - **Process: (= source IP + TCP source port):** easily sidestepped by opening multiple copies of same process
    - **Source+destination (= source IP + destination IP):** can consume boundless BW by sending traffic to many destinations
    - **Conversation (=source IP+TCP source port+ destination IP+TCP destination port)**
  - The paper assumes conversation as the scheduling granularity, but **the choice is not critical for the functioning of the algorithm**

Issues - II

- Packets are of **different lengths**
  - **Source sending longer packets can still grab more than their share of resources**
  - We really need bit-by-bit round-robin
    - But not feasible to interleave bits!
  - **Fair Queuing *simulates* bit-by-bit RR**

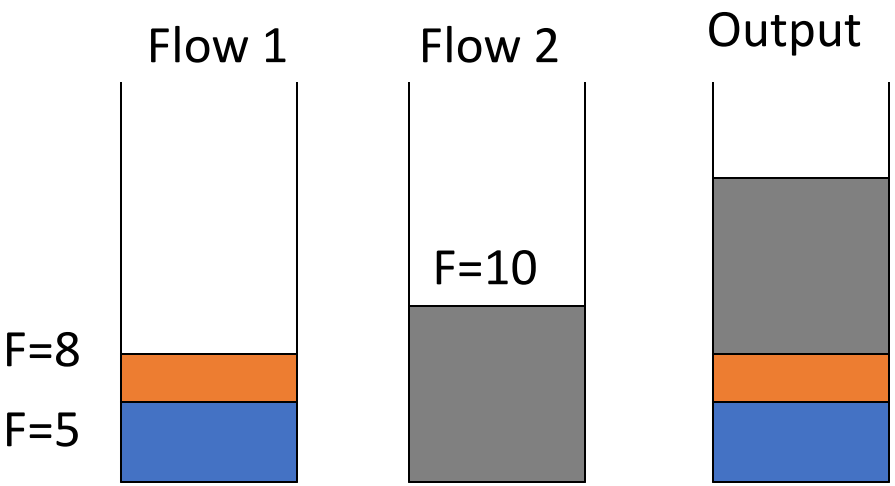
Bit-by-bit RR

- Router maintains **local clock**
- Single flow: suppose clock ticks when a bit is transmitted. For packet *i*:
  - P_i: length, A_i = arrival time, S_i: begin transmit time, F_i: finish transmit time. $F_i = S_i + P_i$
  - $F_i = max\ (F_{i-1}, A_i) + P_i$

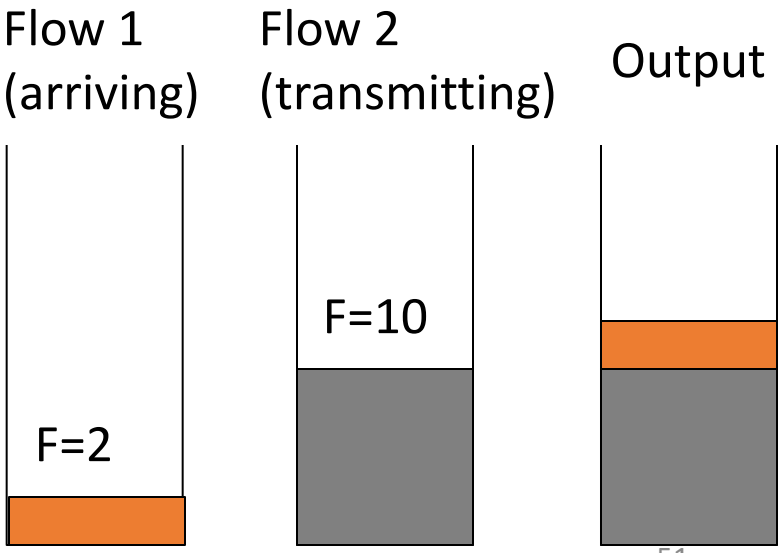
Fair Queuing

- While we cannot actually perform bit-by-bit interleaving, can compute (for each packet) F_i. Then, **use F_i to schedule packets**
  - **Transmit earliest F_i first**
- Still not completely fair
  - But **difference** now **bounded by the size of the largest packet**

Fair Queuing Example



Cannot preempt packet currently being transmitted



Delay Allocation

- Aim: **give less delay to those using less than their fair share**
- Adjust finish times for **sources whose queues drain temporarily**
- $B_i = P_i + max\ (F_{i-1}, A_i - \delta)$
  (compare with $F_i = max\ (F_{i-1}, A_i) + P_i$ *for default FQ)*
- Schedule earliest B_i first

Allocate Promptness

- $B_i = P_i + max\ (F_{i-1}, A_i - \delta)$
- **δ gives added promptness:**
  - if $A_i < F_{i-1}$, **conversation is active and δ does not affect it:** $F_i = P_i + F_{i-1}$
  - if $A_i > F_{i-1}$, conversation is inactive and δ determines how much history to take into account
  - This strategy makes the scheduling decision dependent on the previous packet finishing round, as long as it was not too long ago
  - How long ago? That depends on the choice of δ (0 → no history, ∞ → no bound on history)

Notes on FQ

- FQ is a **scheduling policy**, not a drop policy
- FQ is *work conserving* (transmission never idle if there are packets in any queue)
- WFQ (weighted fair queuing) is a possible variation

More Notes on FQ

- Router does not send explicit feedback to source - **still needs e2e congestion control**
  - FQ isolates ill-behaved users by forcing users to share overload with themselves
  - user: flow, transport protocol, etc
- But, **maintaining *per flow state* can be expensive**
  - Flow aggregation is a possibility

RED queuing

Random Early Detection Gateways for Congestion Avoidance

Sally Floyd and Van Jacobson*

Lawrence Berkeley Laboratory
University of California
floyd@ee.lbl.gov
van@ee.lbl.gov

To appear in the August 1993 IEEE/ACM Transactions on Networking

Abstract

This paper presents Random Early Detection (RED) gateways for congestion avoidance in packet-switched networks. The gateway detects incipient congestion by computing the average queue size. The gateway could notify connections of congestion either by dropping packets arriving at the gateway or by setting a bit in packet headers. When the average queue size exceeds a preset threshold, the gateway drops or *marks* each arriving packet with a certain probability, where the exact probability is a function of the average queue size.

RED gateways keep the average queue size low while allowing occasional bursts of packets in the queue. During congestion, the probability that the gateway notifies a particular connection to reduce its window is roughly proportional to that connection's share of the bandwidth through the gateway. RED gateways are designed to accompany a transport-layer congestion control protocol such as TCP. The RED gateway has no bias against bursty traffic and avoids the global synchronization of many connections decreasing their window at the same time. Simulations of a TCP/IP network are used to illustrate the performance of RED gateways.

1 Introduction

In high-speed networks with connections with large delay-bandwidth products, gateways are likely to be designed with correspondingly large maximum queues to accommodate transient congestion. In the current Internet, the TCP transport protocol detects congestion only after a packet has been dropped at the gateway. However, it would clearly be undesirable to have large queues (possibly on the order of a delay-bandwidth product) that were full much of the time; this would significantly increase the average delay in the network. Therefore, with increasingly high-speed networks, it is increasingly important to have mechanisms that keep throughput high but average queue sizes low.

In the absence of explicit feedback from the gateway, there are a number of mechanisms that have been proposed for transport-layer protocols to maintain high throughput and low delay in the network. Some of these proposed mechanisms are designed to work with current gateways [15, 23, 31, 33, 34], while other mechanisms are coupled with gateway scheduling algorithms that require per-connection state in the gateway [20, 22]. In the absence of explicit feedback from the gateway, transport-layer protocols could infer congestion from the estimated bottleneck service time, from changes in throughput, from changes in end-to-end delay, as well as from packet drops or other methods. Nevertheless, the view of an individual connection is limited by the timescales of the connection, the traffic pattern of the connection, the lack of knowledge of the number of congested gateways, the possibilities of routing changes, as well as by other difficulties in distinguishing propagation delay from persistent queueing delay.

The most effective detection of congestion can occur in the gateway itself. The gateway can reliably distinguish between propagation delay and persistent queueing delay. Only the gateway has a unified view of the queueing behavior over time; the perspective of individual connections is limited by the packet arrival patterns for those connections. In addition, a gateway is shared by many active connections with a wide range of roundtrip times, tolerances of delay, throughput requirements, etc.; decisions about the duration and magnitude of transient congestion to be allowed at the gateway are best made by the gateway itself.

The method of monitoring the average queue size at

*This work was supported by the Director, Office of Energy Research, Scientific Computing Staff, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

Random Early Detection (RED)

- **Motivation:**
  - TCP detects congestion from loss - after queues have built up and increase delay (not good if goal is to keep queue utilization low!) (**full queue problem**)
- **Aim:**
  - keep throughput high and delay low
  - accommodate bursts
- **Approach:**
  - Probabilistically drop packets **before** congestions occurs
  - No per-flow state

Solving the Full Queues Problem

- **Drop packets before queue becomes full** (early drop)
- Intuition: **notify senders of incipient congestion**

RED Operation