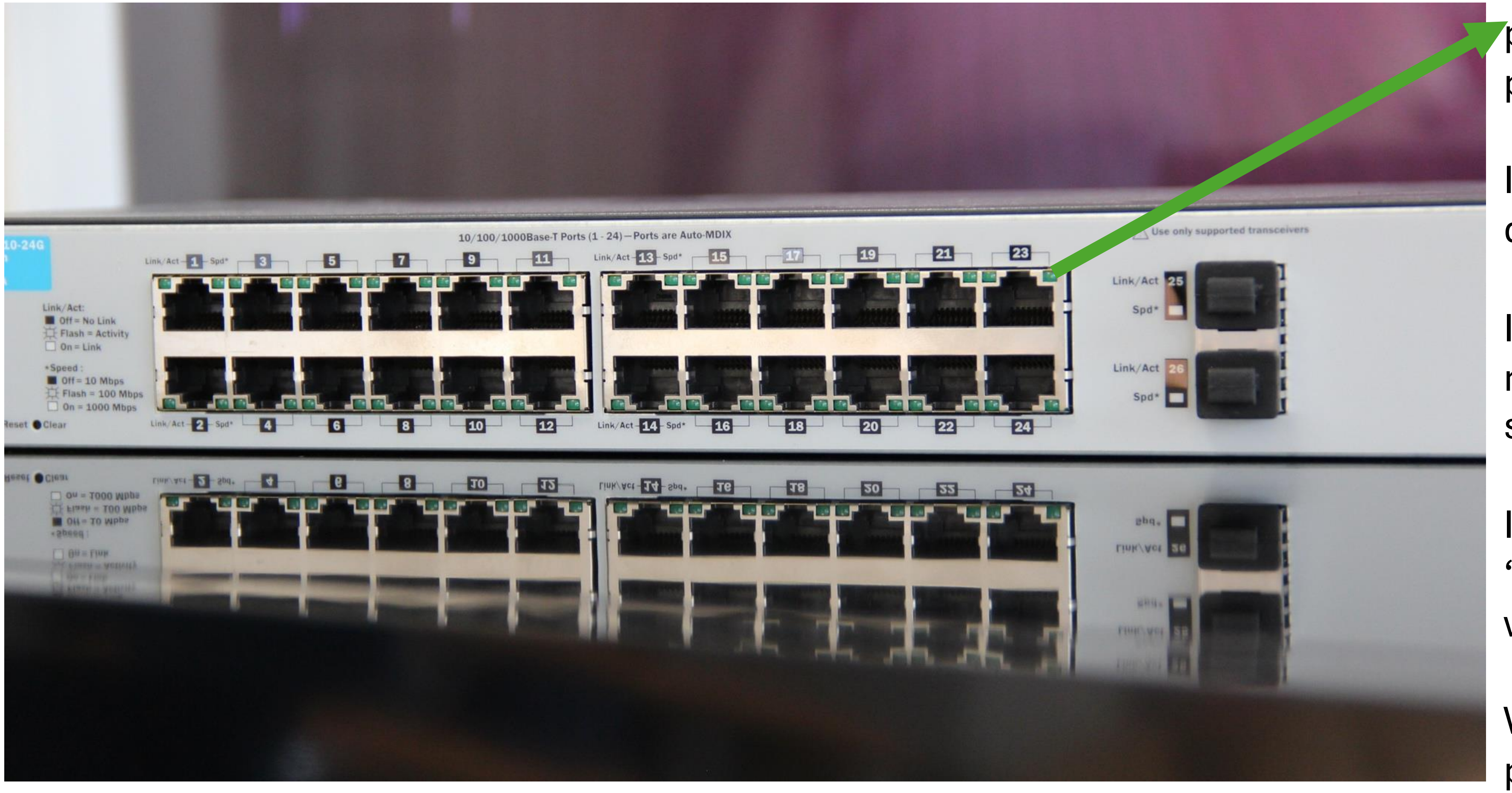
Lecture 18: Queuing disciplines + the application layer

Lecturer: Venkat Arun

Queuing Disciplines: an in-network bandwidth allocation mechanism (chapter 6.2)

- Implemented by routers
- When a router’s capacity is exceeded, which packets should it forward and which should it drop?
- In what order should it forward packets

Primer on router architecture



These are ethernet ports/interfaces. It reads every packet from every port.

It reads the header to get the destination IP addresses

It looks up this address in its routing table to decide which port it should forward it to

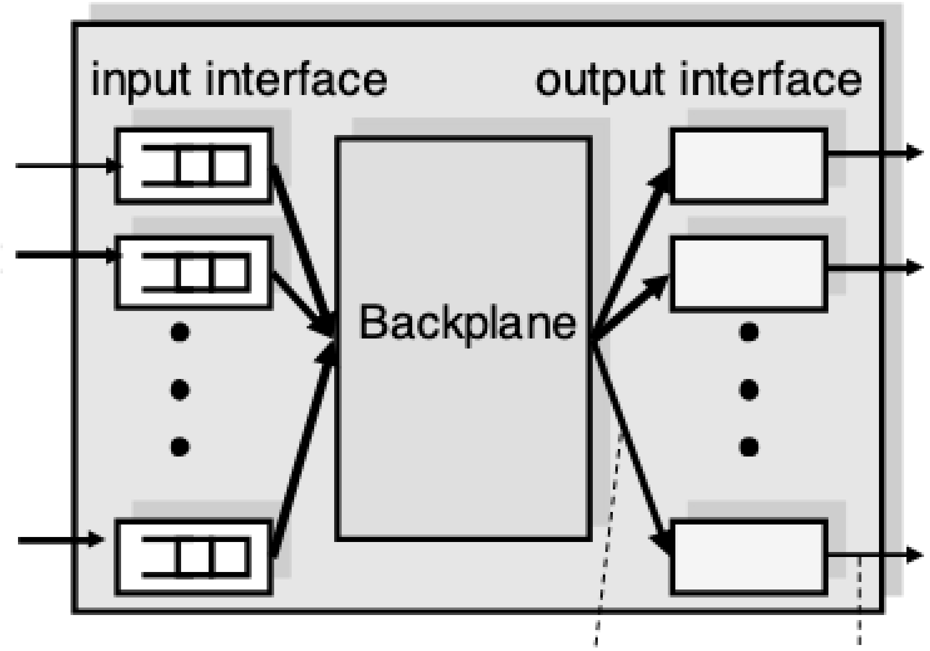
It then sends it to that port using a “switching mechanism” that we will study in the next slide

We can separate each physical port into an “input port” and an “output port”

Primer on router architecture: Input Queued Routers

Note: This is only one of many possible designs, but we will not discuss the others in this course

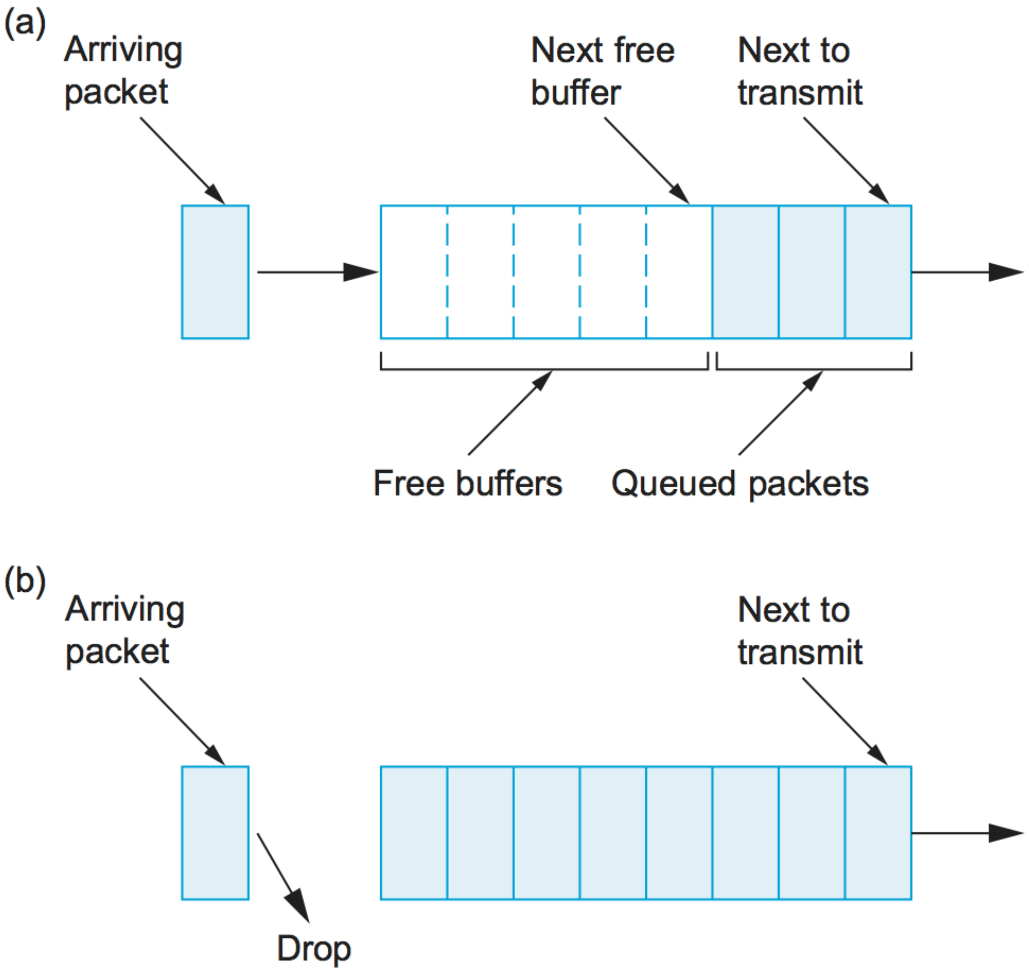
- Packets are queued only at the input ports
- Time is divided into slots. At every time slot, the backplane takes some bytes from the input interface/port to the output port. There is a lot of sophistication in this backplane that we will not discuss in this course
- The queuing disciplines we will discuss today are applied independently to queues in each input port



Picture taken from Scott Shenker and Ion Stoica's slides

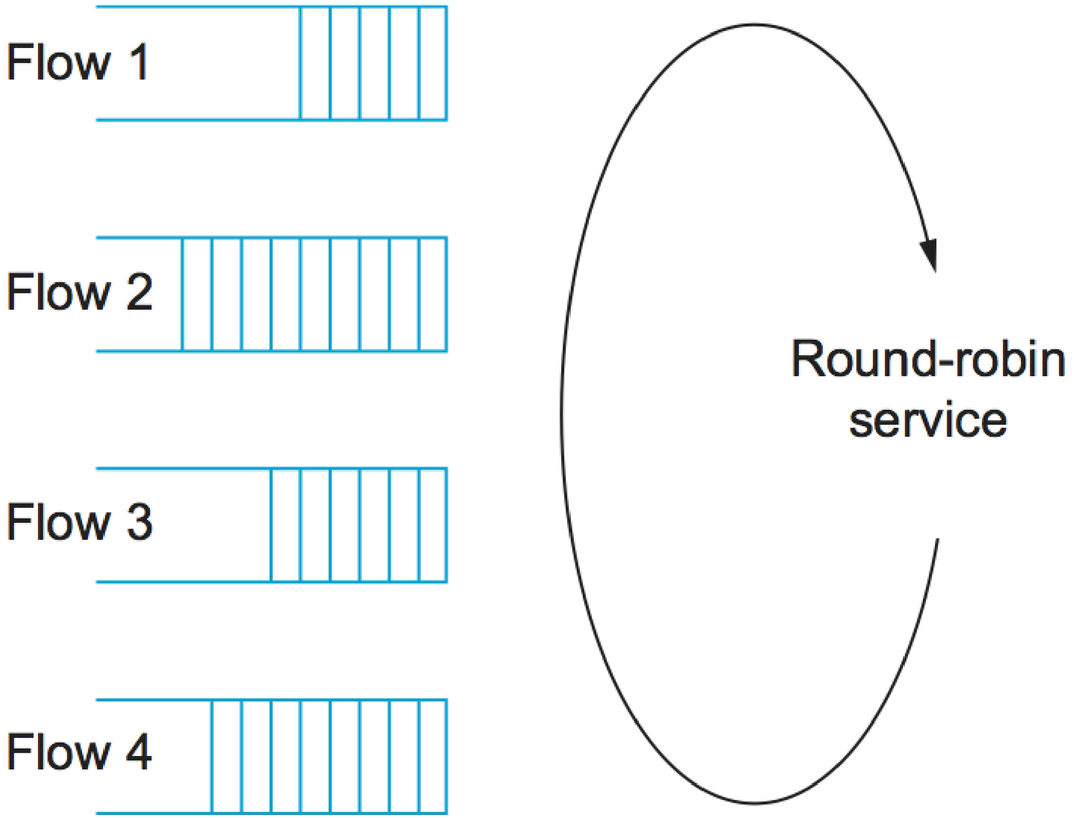
Queuing discipline: FIFO

- This is what we had assumed when we discussed AIMD congestion control
- *All* packets arriving at a port are enqueued here
- Advantage: simple to implement
- Disadvantage: Flows that send more get more bandwidth. They can go “rogue”, blast packets into the network and get rewarded for it (e.g. by modifying TCP congestion control)



Queuing discipline: Fair queuing

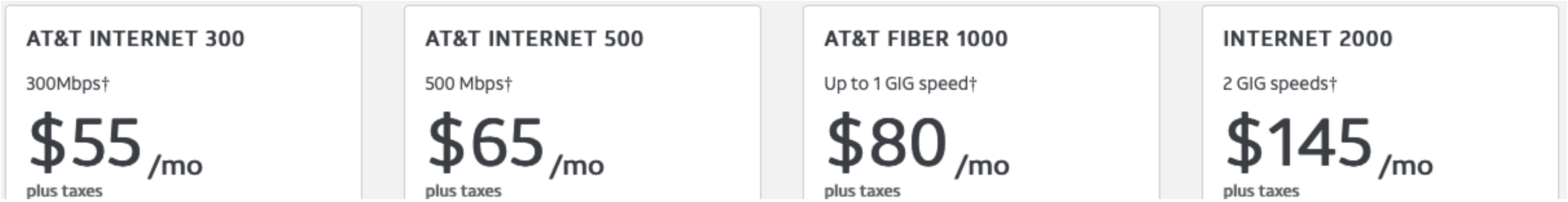
- Input queues discussed earlier are split further into multiple queues.
- Packets are classified into “flows” in some way and put into separate queues
  - E.g. using src/dst IP and port and the protocol number. This is commonly called a “5 tuple”
- Packets are dequeued so that each queue gets the same number of bytes per second
- Advantage: prevents rogue flows (Q. does it?)
- Disadvantage: much more complex. In particular, hardware is bad at maintaining a *variable* number of queues since the number of packets is not known a priori



Queuing discipline: artificial bottlenecks

- Sometimes routers are explicitly instructed to forward at a lower speed than what they are capable
- This is extremely simple to implement and ensures the congestion stays near the edges of the internet.
- Mechanisms like fair queuing are nearly impossible to implement near the core where millions of flows may go through the same link

Commercial artificial bottlenecks



AT&T INTERNET 300
300Mbps†
$55 /mo
plus taxes

AT&T INTERNET 500
500 Mbps†
$65 /mo
plus taxes

AT&T FIBER 1000
Up to 1 GIG speed†
$80 /mo
plus taxes

INTERNET 2000
2 GIG speeds†
$145 /mo
plus taxes

- This usually just means that there is some place between you and the network that is bottlenecked at the advertised speed.
- It caps the maximum, but says very little about what speed you will actually get
- When you change your plan, nobody is going out to install a new router or wire. It is just a software change that asks the router to change how much it is throttling your packets
- ISP contracts with commercial entities (e.g. UT Austin) are similar, except with bigger numbers

Application layer

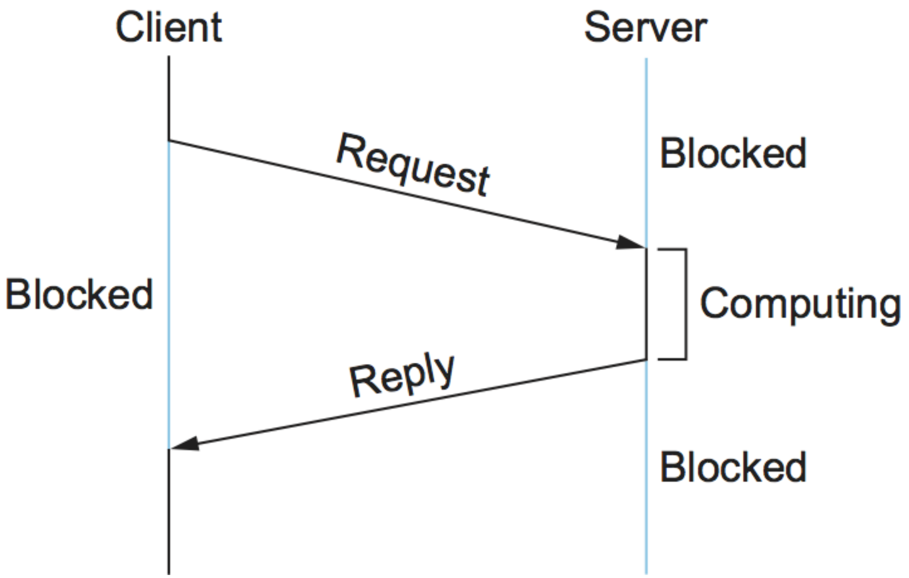
RPCs (chapter 5.3) and HTTP (chapter 5, perspective)

Why do we need a higher layer?

- TCP lets you transmit streams of bytes
- Imagine a computer just receiving some bytes. It doesn't know who is sending them, what the bytes mean or what they want it to do
    - Note: an IP address is only a rough clue for who is sending. E.g. it does not identify a specific user
- This is the job of higher layer protocols
- We will study HTTP as an example

Request-Response protocols

- Today, we will focus on the web
- Shown on the right is a request-response pattern commonly used in applications
- It could be to download a webpage, or to interact with it.



HTTP Format

START_LINE <CRLF>
MESSAGE_HEADER <CRLF> <CRLF>
MESSAGE_BODY <CRLF>

where <CRLF> is a special two-byte sequence (carriage return and line feed)

Start line formats:

For “get” requests:
GET /~venkatar/f24/assignments.html HTTP/1.1

For responses:
HTTP/1.1 200 OK

Or, if the server wants to say the page does not exist:
HTTP/1.1 404 Not Found

Examples of header fields:

Host: www.google.com
Date: Thu, 24 Oct 2024 17:14:44 GMT
Last-Modified: Mon, 07 Oct 2024 16:58:46 GMT
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8

Live demo 1: course website

- When you load a webpage, the browser sends a request asking for the HTML.
- The server responds with some HTML. The HTML tells the browser what other objects (if any) are needed to render the webpage. Usually, this includes videos, CSS, and javascript (executable code)

Live demo 2: Google’s autocomplete

- When the page loads, the browser first downloads some code from google that tells it what to do
- When you type something into google.com, the downloaded code asks the browser to send requests to the google server as your type into it.
- The server responds with autocompletion results