The Problem

Given: A broadcast medium connecting N stations
- When one station transmits, every other station receives its transmission
  ...after some propagation delay
- If transmissions from 2 or more stations arrive at a point simultaneously, they are said to collide
- Medium may be 1, 2, or 3-dimensional

To do: Devise a distributed protocol for sharing the channel
- Deterministic: either all stations receive a message or none do
- Fairly: over time, each station gets the same share
- Efficiently: minimize wasted capacity
  • Axiom: any solution will result in some channel capacity being "wasted"
Signal Propagation

...in a one-dimensional medium (bus)
System Load

- Packets **arrive** from time to time (randomly) to stations
  - Each station is either **idle** or has a packet to transmit
- The **absolute offered load** on the system is the average rate at which traffic arrives (in bits/sec)
  - E.g. a 1000-byte packet arrives at some station every 5 sec ⇒ absolute offered load = 1600 bits/sec
  - The **arrival process** describes the manner in which traffic arrives more precisely (e.g. how bursty)
- The **normalized offered load** is the absolute load divided by the channel capacity
  - E.g. for a 1-Mbps channel, the normalized offered load above is .16%
Throughput vs. Offered Load

- **Throughput** is the rate at which traffic is successfully transmitted over the channel
  - Absolute: rate given in bits/second
  - Normalized: absolute rate divided by channel capacity
- Obviously throughput cannot exceed offered load
  - The system does not "create" traffic
- As normalized offered load grows, throughput should approach some maximum and level off
Delay vs. Throughput

- **Delay** experienced by a packet:
  - Interval between arrival time and when it is finally transmitted successfully

- Delay has two components:
  - Queueing delay = waiting for other packets that arrived in front of it to be transmitted
  - Protocol delay = waiting for the protocol to operate

- As normalized throughput approaches 1, delay becomes infinite, due to queueing
The Two Ways to Waste Capacity

1. Some station has a frame to transmit and the channel is idle
   (Given an "ideal" solution this would never happen)
   - Stations would transmit as soon as the channel is idle
     This shows up as delay under light load

2. Collisions
   Typically collisions happen at higher loads
   For some protocols, as more capacity is wasted due to collisions, throughput actually decreases at high loads
Taxonomy of Approaches

- Scheduling methods
  - Stations form a distributed queue, and "take turns" using the channel
  - Protocol designed to prevent collisions
  - Examples: polling, probing, token-passing

- Random-Access methods
  - Stations transmit without waiting for their "turn"
  - Protocol allows collisions
    - Specifies how to "try, try again"
  - Examples: ALOHA, CSMA/CD (Ethernet), CSMA/CA (WiFi)
Scheduling Methods: Polling

- Early MAC protocol
- Central **controller** (Note: non-symmetric protocol)
- **Stations** connected to **semi-broadcast** channels
  - All stations "hear" controller & vice versa
  - Stations cannot necessarily hear each other

```
C to #1: Anything to transmit?
#1 to C: Nothing to xmit

C to #2: Anything to transmit?
#2 to C: [Data]
```
Polling

- Early MAC protocol
- Central **controller** (Note: non-symmetric protocol)
- **Stations** connected to **semi-broadcast** channels
  - All stations "hear" controller & vice versa
  - Stations cannot necessarily hear each other

```
C to #3: Anything to transmit?
#3 to C: Nothing to xmit
C to #4: Anything to transmit?
#4 to C: Nothing to xmit
```
Polling — Where's the Waste?

Assume:
- Total time to poll all stations & get response $= \tau$
- Data header takes same time as neg. poll response
- Time to xmit data frame $= d$

Normalized throughput (this example): $\frac{d}{d + \tau}$
Polling — Where's the Waste?

Assume:
- Total time to poll all stations & get response = $\tau$
- Data header takes same time as neg. poll response
- Time to xmit data frame = $d$

Normalized throughput (this example): $\frac{d}{d + \tau}$

- Polling methods waste time when there is little data to transmit
  - Stations have to wait their turn, even though no other station has anything to transmit
Improvement: Probing

- Divide stations into groups
- **Poll group as a whole**
  - Stations respond by *transmitting their number* if they have a frame to transmit
  - Note: protocol differs according to whether stations can hear each other (*)

- **Three possible outcomes for each poll:**
  - **No station responds** (within time) → poll next group
  - **Exactly 1 station responds** → immediately poll that station*
  - **Collision** (more than 1 station responds) → poll group individually
Group A, anybody ready?
Group B, anybody ready?

#6, Go ahead

Group B, anybody ready?
Probing Example

Group A, anybody ready?
  yes, #4
  yes, #3

A

C

5
6
B
7
8
Probing Example

C to #1: ready?
A

#1 to C: No thanks

C to #2: ready?

#2 to C: No thanks

C to #3: ready?
Probing Example

#3 to C: [Data]
#4 to C: [Data]

C to #4: ready?
Probing Efficiency

- **Goal**: groups should be small enough that expected number of responses is $\approx 1$
- **Problem**: more groups $\Rightarrow$ more delay to find a "ready group"
- **Solution**: Dynamically adjust size of group polled
- **Tree Probing**
  - Logical tree structure, with stations as leaves
  - Subtrees $\leftrightarrow$ subgroups of different sizes
    - Each station knows the (log N) subgroups it belongs to
  - Controller probes different levels in tree according to load
Tree Probing

- Probe: Group A → Collision
- Probe: Group B → Collision
- Probe: Group D → 2: Data
- Probe: Group E → 4: Data
- Probe: Group C → 6: Data
Random Access Methods

• General Approach:
  – Stations access the channel at random times
  – Collisions may occur
    • Protocol may try to reduce likelihood of collisions
  – Stations re-try after colliding

• Tricky bits:
  – For stability, stations must "back off" after collisions
    • Otherwise, collisions recur forever after
  – How to ensure stations have the same idea of the state of the medium
Taxonomy of Random Access Methods

• Simplest: ALOHA
  – Stations transmit whenever a frame arrives
    • Frames collide if they overlap by even one bit!
  – Detect collisions by lack of (higher-level) ACK
  – Retransmit after collision
  – Better: Slotted Aloha (halves probability of collision)

• More polite: Carrier-Sense Multiple Access
  – "Listen before talking"
  – Small period during which collision can occur

• Even Better CSMA with Collision Detection
  – "Listen while talking"
  – If collision occurs, stop transmitting!