6.02 - MAC Protocols

Lecture #18

Katrina LaCurts, lacurts@mit.edu

==

6.02 Part 3 - Networks

==

So far, 6.02 has given you a fairly complete picture of how to get data from one machine to another one that is directly connected:

- We start by encoding the data in a particular way (Huffman, LZW), typically with the goal of being efficient in terms of the number of bits we use.
- Then, we might add error correction into that data stream (Hamming codes, convolutional codes), to make the transmission more robust to noise.
- That data is modulated and sent across the channel, where it is affected by noise.
- On the receiving end, data is demodulated and filtered, and an approximation of the original *signal* is recovered.
- Then, we use threshold detection to determine what constitutes a 0 and what constitutes a 1, and decode the signal back into a bitstream.
- Finally, those bits are decoded with whatever method is appropriate (syndrome decoding, Viterbi).

The third part of 6.02 will deal with how we handle communications between multiple machines, i.e., networking. The next lecture will give a more complete overview of networking and the questions therein. Today's lecture is more of a bridge between Parts 2 and 3. Along the way I will introduce some networking terms, which you'll see again in the next lecture. I've also included a dictionary along with these notes.

==

Introduction to Media Access Control (MAC)

==

Shared Media

"Shared media" refers to communication channels where multiple "nodes" can all be connected and hear each other's transmissions.

This should be remarkably intuitive: we're all sitting in lecture sharing this (802.11) wireless network. There are lots of other examples:

- Satellite networks: many nodes on land can send their data to a satellite, which will then send the data back down to the destination.

- Cellular networks: your phones connect to a cell tower in your vicinity, and the tower is then connected to the rest of the phone network (we don't care about the details of cell networks in 6.02); multiple phones can connect to the same tower.

- PS6: PS6 presented a shared medium. You sent two messages on two different frequencies across the same channel. It's a little different because the two messages came from the same node, but you could imagine extending PS6 to work across machines, and allowing multiple machines to transmit at once, on different frequencies (in fact we'll come back to this idea).

We can even talk about shared media with things other than communication networks. For instance, roads form a network, and those roads act as a shared medium (cars are sharing the channel).

Necessary Abstractions

We're going to talk about shared media in general, not for any one particular abstraction. The ideas we'll talk about today are applicable to any of the above examples (except, perhaps, the road network).

Here is our abstraction:

1. Nodes transmit units of data called packets. A lot of times, this is about 1.5KB of data (that number is not important in 6.02 and can change; I'm giving it to you strictly to make the notion of a packet more concrete). If we wanted to transmit more data than that, we'd just break it into multiple packets.

2. Nodes interface with the channel in some way. This is what connects the queue of packets to the shared channel.

3. The channel is shared. When two or more nodes are transmitting at the same time, the packets will collide and be lost. This is a collision. Even if only part of a packet is involved in a collision, the entire packet will be lost (you will see later in these notes how it is possible for "part" of a packet to collide).

4. Nodes can tell (can "detect") when a collision has occurred, and re-transmit the packets later.

5. On the shared channel, nodes may be able to hear each other perfectly, partially, or not at all. If node A hears node B sending a packet, then A will wait to send its packet (and thus avoid a collision with B). But this won't be possible if A can't hear B, or if A and B start sending at the same time.

Goals

To deal with this shared medium, we're going to design a communications protocol. The goals of the protocol are two-fold:
1. To avoid collisions
2. To ensure good performance

(Without the second goal, it's very easy to satisfy the first: don't allow any communication whatsoever).

What do we mean by "good performance"? There are many facets of performance. Here is the first:

---------------------------- Utilization ----------------------------

1. Utilization: the channel is a shared resource, so we should use it efficiently. Ideally, we'd use 100% of its capacity (if we don't, we could be sending more data, and aren't).

We calculate utilization the following way:

```
U_channel =   total throughput over all nodes
            -----------------------------------
              maximum data rate of the channel
```

Obviously, at this point I need to tell you what throughput is. Throughput is the amount of data a node transfers over a certain period of time (PRO TIP: throughput is a *unit of data* over a *unit of time*, e.g., megabits-per-second, bytes-per-second, etc. If you are calculating throughput, please check your work; if you end up with something like seconds-per-byte, byte-per-megabit, seconds-per-minute, etc., something has gone horribly wrong).

We will do a lot more with throughput in a few lectures.

Example: Suppose we have a channel capable of sending 10 megabits-per-second (Mbps). Four nodes are connected to it, and they are sending with throughputs 1Mpbs, 2Mpbs, 2Mpbs, and 3Mpbs. What is the utilization? (1 + 2 + 2 + 3) / 10 = 0.8

Note: 0 <= U <= 1:
- U cannot be greater than 1; the channel cannot send more data than it is capable of sending.
- U cannot be less than 0: that doesn't even make sense.
- U = 1 is perfect utilization (the channel is 100% utilized)
- U can be less than 1 for a variety of reasons:
  - The nodes have packets to transmit, but the protocol is inefficient
  - Nodes don't have enough packets to transmit to use the full capacity of the channel

(Some more terms: nodes that have packets in their queue are backlogged. The packets available for transmission constitute the offered load; when there is insufficient offered load, U is less than 1)

----------------------------- Fairness ------------------------------

We could achieve 100% utilization by always letting node A use the channel (assuming it has infinite data to send; if not, maybe we let node B jump in until node A has data). But that certainly seems like an undesirable outcome. So we'll add a second goal:

2. Fairness: ideally, we'd divide the capacity equally among the nodes requesting to use it. But there are all sorts of problems here: not all nodes need to send data all of the time, we need a way to determine *who* has data to send, some data is more important than other data (e.g., maybe calls to 911 should get priority on the cell network), ... (really I could go on for days). In 6.02 we're going to concern ourselves mostly with the first one, but just know that fairness is an Issue in networking.

How do we calculate fairness? That's a complicated question; there are actually multiple definitions of fairness. But we have a particular one we'll use in 6.02.

Suppose we measure the throughput x_i that node i achieves under our protocol, and assume there are N nodes. If things are perfectly fair, each x_i will be equal. The less fair the protocol, the more spread out this distribution is.

We capture that notion with the following fairness index:

F = (sum_i=1^N x_i)^2   / (N * sum (x_i^2))

(The N in the denominator lets us normalize as the number of nodes varies)

From this, you can see that 1/N <= F <= 1. If F = 1/N, then a single node is getting all of the throughput. If F = 1, the protocol is perfectly fair (Work these out yourself! You should understand how I came to that conclusion).

One tricky aspect of this is: over what period should we calculate the throughput used to make these calculations? A few seconds? Hours? We'll consider both.

There are other goals that we care about, and I'm going to talk about those at the end of class today, but for now, utilization and fairness are the two big ones. They are often at odds with each other.

==
Time Division
==

It's time to analyze some specific Media Access Control (MAC) protocols. The utilization/fairness calculations will come in during this analysis; it's how we'll determine whether a protocol enforces "good" performance.

(Note: If you have experience in cryptography, you may know MAC as an acronym for Message Authentication Code. Media Access Control has nothing to do with that; it's an unfortunate collision of acronyms. If you have experience in computer systems, you may have heard of a MAC Address. Media Access Control *does* have something to do with that -- a MAC Address is a Media Access Control Address -- but we are not going to talk about MAC Addresses in 6.02.)

Frequency Division vs. Time Division

You've already shared a channel in PS6. There, multiple senders used *frequency division* to share the channel. Each picked a different frequency on which to send, and the receiver was able to recover both transmissions. You saw that this required some finesse: the channels needed a gap between them, smart filtering had to be used, etc. And it only got us so far; we couldn't handle much more than six messages transmitting at once.

Today, we're going to share the channel in a different way: instead of dividing up frequency, we're going to divide up time.

Necessary Abstractions for Time Division
----

1. Time is divided into slots of equal length.

2. Each node can start a transmission of a packet only at the
   beginning of a time slot.

```
3. All packets are of the same size and hence take the same amount of
   time to transmit, equal to some integer multiple of time slots
   (e.g., 1 time slot, 2 time slots, ...)

Now recall our channel abstractions.
- Collisions happen when two senders send at the same time
- The channel is idle when no one sends
- A transmission is successful when exactly one sender sends in a
  time slot

Remember that even if a collision involves only part of a packet, the
entire packet is assumed to be lost.  Also remember that nodes can
detect when a collision happened (and then will retransmit the
packet).

To calculate the utilization of these protocols, we need to be precise
about how we calculate throughput.  It is the number of *uncollided*
packets per time interval.

======================================================================
Protocol 1: Time Division Multiple Access (TDMA)
======================================================================

----------------------------------------------------------------------
Protocol
----------------------------------------------------------------------

1. For N nodes, give each node a unique index in the range [0, N-1].
   Assume each slot is numbered starting at 0.

2. Node i gets to transmit in time slot t iff t mod N = i.  So a
   particular node transmits once every N time slots.

The way that we would make this work technically would be to have a
centralized "resource allocator" and a way to ensure time
synchronization between nodes.  We'll just assume this exists for now,
and not worry about the details in 6.02.

----------------------------------------------------------------------
Analysis
----------------------------------------------------------------------

Let's see how well this protocols meet our goals: eliminating
collisions, providing high utilization and fairness.

Are there collisions?  No!  Assuming everything works correctly, there
are no collisions.

Is it fair?  Seems pretty fair; every node gets an equal share of the
channel.

In fact, if we assume all nodes always have data to send, and the
throughput of the channel is X_c, then each individual node has
throughput x_i = X_c / N.  And then:

    F = (sum_i=0^{N-1} x_i)^2 / (N * sum_i=0^{N-1} (x_i^2) )
      = (N * X_c/N)^2 / (N * N * X_c^2 / N^2)
      = 1

So it's perfectly fair by our definition, and under this assumption
about nodes having infinite offered load.
```

But what about utilization? Well, if nodes have infinite offered load, then utilization is 100%. But if they don't, what happens?

There are wasted time slots! If node 3 doesn't have very much data to send, its slots will usually be wasted. It seems silly not to allow another node to send in those wasted slots.

So TDMA is particularly poor when nodes have different offered loads, or when data is sent in bursts (i.e., nodes have a lot of data to send at once, and then a period of no data to send, rather than having a constant stream of data to send).

```
======================================================================
ALOHA
======================================================================
```

We are going to improve on TDMA with a protocol where allocation is *not* pre-determined (this type of protocol is known as a contention protocol). That will allow us to improve performance when the offered load of the nodes differs.

```
----------------------------------------------------------------------
ALOHA History
----------------------------------------------------------------------
```

This protocol -- ALOHA -- comes from ALOHAnet, designed by Norm Abramson. ALOHAnet was a satellite network that connected computers on the Hawaiian islands. One frequency was used to send data from the island stations to the satellite, another frequency was used to send data from the satellite to the island stations.

The stations could only hear the satellite, and so had to independently decide when it was their turn to transmit.

(Sidenote: ALOHA should be in all-caps, but it is not an acronym. We will not take points off from any assignments if you typeset it incorrectly :))

```
----------------------------------------------------------------------
Protocol 2: Slotted ALOHA
----------------------------------------------------------------------
```

We will first study "Slotted ALOHA", which assumes that each packet takes exactly one time slot to transmit.

The protocol is dead simple: If a node is backlogged (i.e., has packets to send) it sends a packet in the next time slot with probability p. Since slots aren't pre-allocated, we don't have the problem where a node is meant to send but has no data.

But this leads to an obvious question: what should p be? If p is very large, we will see lots of collisions. If p is very small, the network will be under-utilized.

So let's figure out the utilization given a particular probability p. Assume that N nodes are backlogged.

In this case, the utilization when N nodes are backlogged is equivalent to the probability that exactly one node sends a packet.

```
   P(send a packet) = p
   P(don't send a packet) = 1-p
   P(successful transmission) = P(there is only one sender)
                              = P(one node sends and all others
                                 *don't* send)
                              = p * (1-p)^(N-1)

   How many ways can we have exactly one sender? (N choose 1) = N.

So U_{slotted ALOHA) = N * p * (1-p)^(N-1)

   Aside: What we've calculated here is, in some sense, the
   theoretical utilization of ALOHA.  Any particular run of the ALOHA
   protocol could have a different utilization, because the run may
   not reflect our assumptions (e.g., that every node transmits with
   exactly the value p).  We'll come back to this.

Now, what should p be?  We would like to maximize our utilization, so
by taking the derivative and setting it equal to zero, we'll find a
maximum at p=1/N.

When p=1/N, utilization is:

    N * (1/N) * (1 - (1/N))^(N-1) = (1 - 1/N)^(N-1)

What happens as N grows large?  As N -> infinity, (1 - 1/N)^(N-1) ->
1/e \approx 37%.

37% is not that great!  But note that Slotted ALOHA is extremely
simple.  A lot of times we're willing to trade simplicity for
performance.

Regardless, we have another problem: How do the nodes know the value
of 1/N?  Even if the satellite communicates the value of N (and thus
1/N) to them, what if N changes?  Remember that N is the number of
backlogged nodes; that value will almost certainly change over time
(even if N were the total number of nodes, we would like a protocol
that allowed for nodes to join and leave the network).

The fact that the nodes can't independently set p to the optimal value
is the real problem with Slotted ALOHA, not its low utilization.

-----------------------------------------------------------------------
Protocol 3: Stabilized ALOHA
-----------------------------------------------------------------------

To deal with this, we will allow nodes to dynamically adjust p.  Let's
say that a node sends a packet, and it collides with another node's
transmission.  What should happen to p?  It should probably decrease;
collisions are indicative of there being too much traffic in the
network.

What about if the node's transmission is successful?  We could try
keeping p the same; after all, it seems to be working.  But now
imagine that in the next slot we experience a collision; p will
decrease.  There might be a few successful transmissions (and a few
idle slots), and then another collision; p will decrease again.  p is
going to get smaller and smaller, and never increase.

So when a node's transmission is successful, we will *increase* p.

What we're doing here is trying to stabilize the system; we're looking
to make sure it's operating at or near a desired operating point (in
this case, at or near p = 1/N).
```

We will start out with the following decrease/increase rules:

1. When a transmission collides, set p = p/2. This is known as multiplicative decrease.

   If there are k collisions in a row, p will be decreased by a factor of 2^{-k}; this is known as binary (2) exponential (k) back-off (smaller p).

2. When a transmission is successful, set p = min(2*p, 1)

   An alternate idea is to just set p=1. This might seem dramatic, but it will ensure that no slots go idle. We're not going to analyze this case, though.

Trying these rules, we will find that for some nodes, a series of successive failures will result in a very small p, which results in very few transmission attempts. Effectively, nodes that experience this are starved out of the system.

We'd like to prevent p from getting too small. This is easy to enforce; we'll set some limit p_min, and then our rules become:

1. On collision, p = max(p_min, p/2)
2. On success, p = min(2p, 1)

With these rules, we get high fairness and reasonable utilization (the smaller p_min, the higher utilization you'll get; in fact, you can get much higher than our "theoretical maximum". As I mentioned before, the utilization that we calculated was based on all nodes sending at exactly the same p. This is not the case here!).

But we see something concerning: some nodes "capture" the channel for a period, starving out the others. This is known as the capture effect. It leads to significant short-term unfairness, which is undesirable for a few reasons:

- At a high-level, it makes it more difficult for a node to have constant communication; they're communicating in bursts. Think about how that might affect interactive communications such as phone calls.

- This type of short-term unfairness is also bad for particular Internet protocols that we won't discuss in 6.02 (basically: if there is too much time between successive transmissions, these protocols will "time out"). It also causes problems with queues, which we *will* talk about in 6.02, but not for a few lectures.

The capture effect occurs when one node maintains a particularly high p (near 1). We'll deal with this the same way we dealt with the starvation issue:

1. On collision, p = max(p_min, p/2)
2. On success, p = min(2p, p_max)

The analysis we've done for this protocol has been done via simulation. You'll see more of this in PS7. To do a more formal analysis, take 6.041 or 6.008 and learn about Markov chains.

==
Protocol 4: CSMA

```
========================================================================

Earlier we mentioned that nodes using a shared channel could hear each
other completely, partially, or not at all.  Our discussion of ALOHA
was built upon the premise that each node could hear the satellite,
and the satellite could hear the nodes, but in networks in general,
it's not true that all nodes can hear each other all of the time.

However, if nodes *can* hear each other, we can utilize this property:
if the a node can first listen on the channel before attempting
transmission, it can avoid the transmission if it hears another node
(and thus avoid a collision).  This is known as carrier sense.

We can modify Stabilized ALOHA to use what is called CSMA: Carrier
Sense Multiple Access.  You will investigate this process in PS7.
Notice that just because we use CSMA doesn't mean that we will
experience 100% utilization: there could still be idle slots, or two
nodes could start transmitting at the same time.

========================================================================
The rest of networking
========================================================================

------------------------------------------------------------------------
Goals
------------------------------------------------------------------------

In today's lecture, we dealt a lot with fairness and utilization.
Those aren't the only relevant goals for good performance in networks.

Some applications, like phone calls, are particularly sensitive to
delay in the network.  E.g., you don't want a 3-second delay on a
phone call -- it will become unusable -- or most video -- the video
will be jittery, and if it's live TV, you'll be three seconds behind
the rest of the world!  So, a third goal:

3. Bounded wait: We'd like an upper bound on the wait before a
   successful transmission.

There are also some high-level systems goals we care about:

4. Dynamism: Our protocol should be able to handle variability in user
   behavior.  For instance, it should not rest on an assumption like:
   "All nodes should have data to send at all times"; that's not a
   reasonable assumption.

5. Scalability: Our protocol should be suitable for a large number of
   users; it should "scale well".  (in the advent of Big Data, talking
   about scale has become somewhat buzzword-y, but it's actually
   incredibly important; see 6.033 for more details).

------------------------------------------------------------------------
Things we don't care about today
------------------------------------------------------------------------

There were a few things we did not care about today:
- How nodes in a network know where to send (e.g., if I want to send a
  message to George, how does my computer know where George's computer
  is?)
- How data moves within a network.  In ALOHA, it's up to the
  satellite, and then back down, but what about more complicated
  topologies?
```

```
- How can we guarantee that data makes it from the transmitter to the
  receiver?

These are all things that matter in a network, and we'll deal with
some of them in future lectures.
```