Medium Access Control

Course comments: A course with less organized discussion encouraged. The main goal of the course is to find good theoretical work, the right assumptions and metrics.

Readings:
Gallager paper:
Komlos-Greenberg paper:
Background: Vaidya notes on MAC layer
Schiller chapter 3

Next time:
Brenner slides on 802.11.
MACAW paper

1 What is the MAC Layer?

MAC = Media ACcess (wireless channel access)

Purpose of the MAC layer:
Achieve relatively reliable local communication among nearby devices over wireless channel.
→ Reasonable throughput (capacity).
→ Reasonable fairness to each transmitter and receiver.
→ Both unicast (point-to-point) and broadcast.

Recall last time, we described the Physical Layer.
Start with physics: Devices, signals, antennas, signal propagation.
Communicate using analog signals.
The job of the Physical Layer protocols is to produce digital broadcast communication, from each sender to nearby receivers.
So the Physical Layer protocols have to modulate/encode digital data into analog signals at one end, demodulate/decode back to digital signals at the other end.

Problem: the communication achieved by the Physical Layer is not necessarily very reliable.
A message might not get through, because of channel noise or interference (collisions).
The MAC layer is supposed to improve the reliability.
Note: It still will not be perfect—in spite of all the tricks we are going to see, we still may not succeed in delivering all messages. But it should be better.

Q: What are reasonable statements of the guarantees of a MAC layer? Probabilistic delivery guarantees? Conditional?
Q: Are the guarantees of current MAC layers adequate for programming over the layers? (Hari

Balakrishnan always comments on the limitations of current MAC layers)

The main job of the MAC layer is to manage contention among nearby transmitters and receivers.

2 Overview of MAC Layer strategies

Compiled from Hari Balakrishnan's notes for 6.829, lecture 11; Schiller Chapter 3; and Vaidya's MAC chapter.

2.1 Carrier Sense Multiple Access (CSMA)

2.1.1 CSMA for Ethernets

The obvious place to look for channel-sharing strategies is traditional protocols for sharing wired channels (e.g., Ethernet).
These generally use a strategy called CSMA = Carrier-Sense Multiple Access.
Sensing the carrier means that, before transmitting, a sender listens first on the same frequency to see if other transmissions are currently going on.
If so, the sender defers its transmission.
If not, the sender sends, perhaps with some probability p, based on some estimate of the presence and number of other senders.

Two transmissions can still collide, if two senders happen to sense sufficiently close together in time, and then both start to send.
So this requires the transmitters to continue listening as they transmit data, to see if their transmissions sound OK (collision detection).

2.1.2 CSMA for Wireless Settings

However, Ethernet-style CSMA doesn't work so well in the wireless setting:
For one thing, Ethernet collision detection requires the ability to simultaneously send and receive messages on the same channel.
This isn't easy in the wireless setting—requires expensive hardware (more on this below).

Not only that, in the wireless setting, not everyone receives the same signal.
The transmitter actually wants to know what the signal sounds like at the receiver end, not the sender end.
That is, it wants to detect collisions at the receiver.
In an Ethernet, everyone should be receiving the same thing, but not in a wireless setting.

Example: Hidden terminals
Recall from last time: A, B, C, in a line: $A \rightarrow B \leftarrow C$.
A and C cannot hear each other's transmissions, but B can hear both. If A is broadcasting (or trying to unicast to B), and C wants to transmit, C does not hear A when C tries to sense the carrier. So C starts to transmit.
C hears its own signal just fine, so C doesn't detect a collision.

But B, an intended recipient of both messages, doesn't receive either transmission correctly.

Example: Exposed terminal
A, B, C, D; B sends to A, C wants to send to D: $A \leftarrow B\ C \rightarrow D$.
If C senses, it hears B's transmission, which tells C to defer.
But that's not necessary, if C cares only about sending to D. No conflict at D.

To detect collisions at the receiver, the sender would have to get some kind of feedback from the receiver.
Need additional mechanisms, e.g., some kind of acks, nacks (acknowledgements) from the receiver to the sender.

Example: busy-tones = Physical carrier sensing
One way to sense activity on a wireless channel is to add an extra (narrow-band) channel for "busy-tones".
When device B begins receiving transmission from A, sends busy-tone on the special channel.
Another potential transmitter C senses busy-tone and defers transmitting.

Issues with busy-tones:
Requires complex hardware to allow receiving data and sending busy tone at the same time.
Fading: Busy-tones sent on narrow-band channels, which are prone to fading.
Works best if communication is symmetric. So, try to avoid channels with different characteristics: Should have similar gains, nearby frequencies.

Only a very simple busy-tone for everybody. There may be more sophisticated hardware allowing for unique busy-tones, but expensive and not considered here.

For unicast communication only, busy-tones would solve the exposed terminal problem, as C would see that D doesn't send a busy-tone.

2.2 Link-level acknowledgements

A basic technique for adding some reliability in CSMA strategies is to add, within the MAC layer protocol, a protocol where the receiver sends an Ack for each data message, and non-receipt of an Ack causes the sender to retransmit.
This makes sense only for unicast messages, intended for a single receiver.

Good to retransmit within the MAC-layer protocol, to reduce amount of retransmission needed by upper layers, which would be more expensive, with large time-outs.

Thus, a transmitter who has sent a unicast message waits for an Ack.
If the sender doesn't receive an Ack, it retransmits the message.
It should do this only a limited number of times: it might be fruitless to continue, because the device could be turned off, or could have moved away.

The transmitter can use the non-receipt of an Ack, by a certain timeout deadline (after some retries), as a crude way of "detecting collisions": at least, it gives a rather good indication that the message hasn't gotten through—one likely reason is that it collided with another message.

Of course, it could be that the message got through but the Ack didn't, so this isn't a perfect test.

2.3 Reservations

Strategy used in MACA, MACAW.
For unicast transmission only.

Assumes that data to be sent can be fairly long (so that conflicts are likely).
Introduce some control messages, which are short, hence less likely to collide.

Each data transmission is preceded by a handshake to reserve the channel for a period of time.

RTS/CTS/Data (Request to send, Clear to send, Data):
Sender A sends RTS (Request to Send). Contains name of sender and receiver, and length of data to be sent (the amount of time the data transmission will take).
Receiver B responds with CTS (Clear to send), containing the same information.
If sender A gets CTS, sends actual data.

The RTS/CTS/Data protocol can be interrupted (might not complete):
If Sender A hears another CTS, with a planned data-transmission time that would overlap the new transmission, then A doesn't transmit.
This is because some receiver in A's range has already OK'd another transmission.
So A's transmission might collide with that other transmission.

Also, if the receiver B hears another RTS, with a planned data-transmission time that overlaps the proposed one by A, then B doesn't respond with the CTS.
Because the new transmission would collide.

This approach does not require carrier-sensing support—depends on explicit messages.
Reduces collisions, because now the data messages shouldn't collide.
But the RTS and CTS transmissions could still collide, if they are sent simultaneously.
However, they are shorter, so less likely to collide.

RTS/CTS/Data Protocol solves the hidden terminal problem:
CTS by B to A is heard by other potential sender C, so C won't send to B. (C now knows that B will be receiving from A.)

Also handles exposed terminal problem:
Sender C sends RTS to D.
D hasn't heard the RTS from B to A (not in range), so D is free to respond to C with a CTS.
Then C can transmit to D, knowing that D won't be receiving any other transmission.

Q: But, what about the following problem? C has to get a CTS back from D. Maybe it won't hear it, because of interference from B's transmission.
Also, B must get a CTS from A, and might not hear it because of interference from C's transmission. ???
Maybe less likely to collide because the CTS messages are short? Not a perfect solution.

With link-layer acks:
If the data transmission is followed by link-layer Acks, then we have to make sure that the Acks succeed also, not just the data packets.
Because if an Ack gets lost, the data would be retransmitted (expensive).
So, expand the proposed reserved transmission time to include enough time for the Ack too.
We essentially have a RTS/CTS/Data/Ack protocol.

Since this now involves communication in the opposite direction (receiver to sender), we have to treat the link symmetrically.
Simplest version: Let both sender and receiver defer if they hear either an overlapping RTS or CTS.

RTS/CTS/Data or RTS/CTS/Data/Ack protocol reduces likelihood of collision: Now collisions can occur only during the (short) handshake.
E.g., two senders send RTS at the same time for the same receiver.
Probability lower, since RTS messages are short.

Note: If receiver detects the collision, and receives at least one of the RTS messages, it could send CTS to one of the senders.

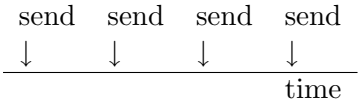
Evaluation:
RTS/CTS scheme imposes high overhead if data packets are short or collisions are infrequent,
low overhead if data packets are long and collisions frequent.
Assumes symmetrical transmission/reception conditions.
Good only for unicast. Doesn't guard against newcomers. New nodes may have to listen for a while before attempting to send.
In exposed terminal problem, the nodes do not transmit, which is the intended outcome, since B and C now need protection as receivers.

Vaidya describes RTS/CTS strategy as "virtual carrier sensing", since it provides an implicit way of telling if the channel is busy.

2.4 CSMA with Collision Avoidance (CSMA/CA)

Try to improve the basic CSMA strategy by adding some features to try to avoid collisions.
Slots, probabilities for transmission, backoff,...

2.4.1 Slots



First, it helps to divide time into *slots*.
Usually equal length.
But the nearby nodes must agree on where the slot boundaries are.
Requires some kind of time synchronization.
(I've made time synch a later topic, but perhaps it, or at least some basic version, needs to come first.)

Senders try to send only at the very beginnings of slots.
This reduces the likelihood of two devices transmitting in such as way as to collide:
E.g., suppose for simplicity that the transmission time d is approx. equal to the slot length.
Then if all nodes transmit on slot boundaries, a particular transmission overlaps only with others sent in the same slot—an interval of length d.
But if the nodes transmit at arbitrary times, a transmission overlaps with any that begin between $(t-d, t+d)$—an interval of length $2d$.
This idea can be used to show that (under reasonable assumptions), the likelihood of collision is halved.

"Slotted Ahoha".

Vaidya analyzes the "throughput" of a slotted system, measured as the expected number of successful transmissions per slot.
He gets $1/e$ in the limit (check it out), as the number of contending nodes grows to infinity.
Compares with unslotted: $1/2e$.
Thus, improves throughput by a factor of 2.

Slotted $1/e$ analysis:
Suppose we have n devices, each tries to transmit at the start of each slot, with probability p.
What is the probability that a particular host i successfully transmits in a particular slot?
$p(1-p)^{n-1}$. (This is the probability that i transmits, times the probability that nobody else transmits).
Multiplying this by n gives the expected number of successful transmissions in the slot:
$np(1-p)^{n-1}$.
This is maximized when $p = 1/n$, so the max throughput is $(1-1/n)^{n-1}$.
Taking the limit when $n \to \infty$ yields $lim(1-1/n)^n(1-1/n)^{-1}$, $= 1/e \times 1 = 1/e$.

Unslotted $1/2e$ analysis:
Treats it as a slotted system with unsynchronized slots at different nodes.
Now the probability that i transmits successfully in its k^{th} slot is $p(1-p)^{2(n-1)}$: the probability that none of the others transmits either in i's slot k or $k-1$.
Then for n devices, the probability that someone transmits successfully in *its* slot k is $np(1-p)^{2(n-1)}$.
Again taking limits, we get $1/2e$.

Q: What if messages are of different sizes?
Could use larger slot lengths.

Evaluation:
The scheme is a bit inflexible, if variations are too great.

The scheme requires synchronized clocks. What if the clocks aren't perfectly, but maybe approximately, synchronized? Suppose they have some bounded skew?
Again, can increase the slot time to accommodate the skew.

2.4.2 Choosing probabilities for transmission

We've said that devices choose to transmit in an idle slot with some probability p. How should we choose p?
If there are n stations around, and all would like to transmit in the same slot, then ideally, everyone would try with probability $p = 1/n$.
However, the devices don't know n—would have to estimate it somehow.

2.4.3 Backoff

After a collision happens, it's reasonable to assume that there are other devices around contending for the channel. The more collisions, the more other devices.
We might want devices to lower their expectations for how quickly they will succeed in transmitting, based on the number of other devices around.
But they have no way of finding out how many other devices are around—all they can do is detect collisions.
So, they simply "back off", delaying their retransmissions based on the number of collisions that they see.

This strategy doesn't use the probability p discussed above.

Node maintains BO, an integer, representing the largest number of slots to wait for retransmission.
Initially some known $BOmin$.
To transmit: Pick a number in range $[0, BO]$, uniformly at random.
Use this to initialize a counter.
Keep trying to sense (listen to communications), counting down the counter only when the channel appears to be idle (let it stall when the channel seems to be busy).
When counter reaches 0, transmit the data.

This strategy ensures that, within a bounded number of detected idle intervals, the device will in fact transmit.
Won't skip an arbitrarily large number of valid opportunities.

Now, if a collision is detected (somehow—note failure to transmit, e.g., by not receiving a link-level Ack), increase BO and start over, choosing random number,...
When data is successfully transmitted, decrease BO.
Common strategy is to increase by doubling (called "binary exponential backoff"), and decrease by resetting to $BOmin$.
This strategy is rather unfair: A device that fails will keep increasing its BO, which gives it fewer and fewer chances.
This tends to allow devices who are transmitting to keep transmitting, and to lock out those who have failed.

Q: What is a good strategy for increasing/decreasing BO?

Random vs. fixed choices:
Why choose a random number in the interval, rather than just a fixed number?
This is to avoid looping behavior—synchronization of transmission attempts—among processes that

have the same BO.
Once they collide, they would keep colliding...

2.4.4 Combining backoff with RTS/CTS

Use backoff to tell the device when to start an entire RTS/CTS/Data dialog, not when to transmit actual data.
Use backoff only for beginning the dialog, not for the rest—once it gets started, keep going.
Of course, look out for collisions.

Recent work proposes an alternative strategy. Decrease BO if collision detected, as it may mean that another contending node succeeded in transmission.

2.5 Time-Division Multiple Access (TDMA)

All the strategies we've just discussed are classified as TDMA, because they time-share the channel.
There are also more extreme versions of TDMA, with more fixed allocations.
The preceding ones may be called "dynamic allocation" TDMA.

E.g., Bluetooth:
Master elected, gets alternate time slots, gives permission to particular "slaves" to transmit in the other slots.

Typical solution for cellular phone systems:
Base station acts as the master, assigning time slots to specific mobile devices.

Fixed TDMA makes sense at high loads.
Simple, predictable delays.
At lower loads, slots are wasted.
Inefficient for bursty data.

2.6 Discussion

In the wireless networks we are considering—both ad hoc and cellular—no one starts out knowing what devices are participating.
It seems that, no matter what we do, there is some possibility of collisions.
All we can do is reduce the probability.
A theorem here?

In an ad hoc network, we can reduce to short intervals—time to exchange control packets RTS/CTS.
In a cellular network, a base station can schedule data transmission.
Here, all we have to worry about is the very small probability that a device will be blocked from announcing its presence to the base station.

CSMA protocols, with collision avoidance, can work quite well at low loads.
Little overhead, low likelihood of colliding.

Reservation-based protocols work better when there is more congestion.
But they work only for unicast.

(Hari) No completely adequate MAC protocols today., for wireless networks.

3 Gallager paper

A Persective on Multiaccess Channels, 1985

For the rest of today and next time, we will review/discuss three papers—two older ones on general problems of multiaccess channels, one newer one describing the MAC layer protocols MACA and MACAW. Also, we will describe IEEE 802.11, from Brenner's slides. Start with Gallager.

3.1 Overview

Considers a simplified setting:
Static, with several transmitters and one receiver.
Problems: Noise, interference
Assumes messages arrive at transmitters from outside sources, at random times.

E.g., this can model uplink for satellite, traffic to central node in telephone network, or traffic to one receiver in a fully-connected wireless radio network.

He points out that there are two different communities working on aspects of the problem, notably:

1. Collision resolution research.

   E.g., what is covered in the Komlos/Greenberg paper.
   Here, the focus is on collisions between transmission attempts.
   The messages arrive from the sources randomly, sometimes in bursts.
   Typically, researchers assume:
   –If exactly one transmitter transmits at a time, the message succeeds.
   –If more than one transmits, a collision happens and communication fails.

   This oversimplifies: Collisions aren't the only cause of communication failure. This ignores noise and other aspects of the real communication process.

2. Multiaccess information theory:

   Here, the focus is on using coding techniques.
   This work arose from the community studying coding techniques for individual point-to-point channels in wired networks.

   Models typically capture the underlying communication channel behavior appropriately, by modeling noise, including probabilistic failure, etc.
   But ignore random nature of message arrivals. (That is, assume everyone always has something to send).

> In the study of point-to-point channels, it's generally assumed that the sender always has something to send—the focus is on getting the transmission to be reliable enough. More precisely, it is on getting the fastest rate possible of sufficiently reliable communication, on the given channel.
>
> Ignoring random arrivals makes sense in the study of point-to-point channels, but it's not appropriate in situations involving multiple senders where each sender has nothing to send most of the time.
> A key problem is how to share the channel among the users that are currently busy, without wasting time on users that aren't busy.

Throughout the paper, he focuses on just the 2-sender case, but says that the results extend.

3.2 The information-theoretic approach

First, the coding theory.
Section II deals with some basic existence results—existence of codes sufficient to achieve certain combinations of rates from two senders.
Section III deals with problems in converting the existence theorems to practical protocols.

3.2.1 The set-up

Model:
Discrete time slots.
Physical Channel given, with two inputs and one output.
Transmitter 1's inputs are symbols from alphabet X, trans 2's inputs are symbols from W.
Since we're neglecting random arrivals, we assume that both transmitters send symbols every time slot.
And, in each time slot, the channel outputs some symbol in the channel output alphabet Y.

To model channel unreliability, we assume that given inputs x and w produce different possible outputs y, each with some probability, written as $P(y|xw)$.
The channel works independently at each time slot (so we can multiply probabilities in various calculations).

Now add encoders and decoders to the picture:

Assume a source at transmitter 1 provides an input integer in $\{1, \ldots, M\}$, and likewise a source at 2 provides an input in $\{1, \ldots, L\}$.
The goal is now to use the given channel, but now to have the same inputs be produced as outputs at the receiver end.

A simple block code can be used to encode each of the transmitter inputs as a code word of length N, for some N: (N, M, L) code.
More precisely, each of the M possible inputs for 1 is encoded as a length N code word over alphabet X, and each of the L inputs for 2 is encoded as a length N code word over W. N times lots used.

Decoding at the other end is done with simple maximum-likelihood decoding:
Given a length N vector of Y values, $\bar{y}$, the decoder chooses (m, l) to maximize $P(\bar{y}|ml)$.
That is, output the pair of inputs that has the highest probability of generating the given output vector $\bar{y}$.

Depending on reliability of the physical channel, this can give some erroneous outputs.
Consider P_e, the probability of an erroneous output for one vector-transmission (which takes N slots).
Depends on the channel characteristics, captured by the probabilities P, and by the choice of code.

In any case, we can talk about the transmission rates – what rates are achievable?
Define rate R_1, the transmission rate for source 1, to be $(\ln M)/N$, that is (essentially) the number of bits sent by source 1 per time slot (it takes around M time slots for $\ln M$ bits.
Likewise, $R_2 = (\ln L)/N$.

3.2.2 An existence theorem

Theorem 1: (Ahlswede, Liao) is an existence theorem, saying that certain combinations of rates (R_1, R_2) are achievable.

Basically, they define a region $\mathcal{R}$ of 2-space describing combinations of rates (R_1, R_2).
They give a tight bound:
For any rate pair inside the region, for any error probability ϵ, and for certain relationships between N, M, and L, coding is possible with the given error probability.
And this is tight—outside the region, this isn't possible (again, for certain relationships between N, M, and L).

3.2.3 Producing actual codes

Theorem 1 gives an existence theorem.
The next theorem, by Slepian and Wolf, produces particular codes with the properties asserted in Theorem 1.
Namely, they produce a particular collection of (N, M, L) codes, by choosing code words randomly (and using maximum likelihood decoders). Are the codes practical?

Error classification:
Type 1: Decode input 1 incorrectly, input 2 correctly
Type 2: Input 1 correct, 2 incorrect
Type 3: Both incorrect

Gallager proves bounds on the three kinds of error probabilities (for randomly-chosen code words), derived from equations in his book.

Theorem 2: (Slepian, Wolf)
For these randomly-chosen codes, we get error bounds as in the previous three inequalities.

The math works out nicely—analyzes the "shape" of the bounds.
Similar behavior as for the single-input case.

But, are these codes practical?
Random arrivals are being ignored—can we do better if we take those into consideration?

3.2.4 Considering collisions

Now he tries to extend the coding approach to allow channel collisions.
Gets some partial results, in the same style.

New model:
Still assumes only 2 senders.
Not handling random arrivals of inputs.

Now one of the elements of X and W is a special "idle" value 0.
Assumes:
If both transmit (non-idle message) at once, then receiver gets a collision indicator c.
If only one, receiver gets the correct data.
If neither, receiver learns this.

Now the transmitters get to choose when to send a "real" element of X or W, or an idle value.
Again, he analyzes the achievable rate region (for transmission of actual data from the outside source).
The analysis is a bit messier now.
He discusses the technical problems of finding the achievable rate region.
Illustrates why it's harder to analyze rates vs. error probabilities for multiaccess channels than for single input channels.

Conclusion: Coding theory is hard to extend to random arrivals.

3.2.5 Need for practical coding technology

Need practical coding technology (as opposed to theorems about existence).
Techniques should apply to any number of transmitters, not just 2.
And they should handle random arrivals of messages from an outside source.
Including the case where a small, but variable subset want to use the channel at any time (random arrivals).

In Section III, Gallager discusses several possible approaches, still emphasizing coding issues.

In some ways, the model is oversimplified:
Might like to include more channel complications, like carrier phase and sampling time.
Might want to allow feedback from receiver to sender—this might increase the achievable rates.

Concludes more research is needed on coding/decoding for multiaccess channels.

3.3 Collision resolution

Now he switches gears, to work that is more like what we discussed earlier today: work on allocating the channel among competing senders.
Weakness: Ignores some aspects of real communication channels that could be important.

3.3.1 Assumptions

1. Slotted system, with message transmission time equal to slot length.

   Senders synchronized, on slot boundaries.
   Needs synchronized clocks, guard space betwen packets, timing feedback.
   Not considering strategies involving combinations of short and long packets, e.g., sending short packets to reserve time for long messages.

2. Collision or perfect reception (receiver-side info).

   If exactly one sends, received with no errors.
   If more than one sends, collision, receiver gets no info about the packets sent, but does know a collision happened.
   These assumptions remove noise and communication aspects from the model/problem, which can lead to unrealistic solutions as well as limit the class of strategies/tradeoffs that are considered.

   Q: Does receiver know if the slot is idle? I assume that's also assumed here.
   But, it's not always realistic to assume receiver can distinguish idle slot from collision.

3. Infinite set of transmitters.

   Assume (for math reasons) that each new packet arrives at a completely new transmitter.
   Avoids consideration of queueing.
   Precludes use of TDM.

4. Poisson arrivals, rate λ.

5. $(0, 1, c)$ immediate feedback (sender-side info).

   Each sender learns *immediately* whether 0, 1 or more packets were sent in that slot.
   A simplifying assumption, but algorithms designed for this assumption can often be extended to case of delayed feedback.

   Assumes transmitters are always listening for feedback.
   Even when they are idle.

The rest of the paper gives specific examples of multiaccess strategies.

3.3.2 Slotted Aloha

Packets sent at slot boundaries only.
Transmit with fixed probability p.

We discussed this above.

Gallager's analysis: Gives a Markov chain model for the number of "backlogged" packets at time t.
Shows: If the system becomes sufficiently backlogged, it drifts in the direction of becoming more and more backlogged.

Choosing p:
Large p makes it easy to enter unstable region (with a big backlog).
But small p causes large delay for collided packets in the stable region.
It might be nice to change p in response to changes in the size of the backlog. But this size isn't known. Must estimate somehow.

Slotted Aloha doesn't use all the feedback info we've assumed:
Doesn't need to distinguish idle slot from collision; enough to distinguish successful transmission from idle/collision.

Throughput upper-bounded by $1/e$, as we analyzed.
Compare with pure Aloha:
Probability of collision is higher, throughput is halved.
Same stability issues as slotted Aloha.
But can handle packets of different lengths.

3.3.3 Splitting algorithms

A collection of papers developed "tree algorithms".
The idea seems to be to try to obtain a sort of first-come first-served behavior, postponing transmission of newly-arrived packets, while trying to service all the packets that were already involved in a collision.

Consider the set of packets involved in a collision.
Split them (probabilistically) into a "transmitting set" and a "nontransmitting set", while making new packets wait.
In fact, split repeatedly, into a kind of tournament.
A bit complicated...
Use this tree to resolve collisions, and service all the previously- collided packets.
Algorithm spends many slots resolving collisions, based on this tree of sets.
Collision mode, vs. normal mode.

After the colliding packets in the tree are all sent, we go back to normal mode.
But now we have a backlog of new arrivals.
If they all transmitted at the next slot, we would have a collision again.
So maybe it's better to eliminate the normal mode—just manage the tree forever.

Gallager modified this idea further, eliminating the tree structure: FCFS splitting algorithm

3.3.4 Carrier sensing

Assumes that an idle slot can be detected somehow, during the slot (different from earlier assumptions 1-5).
Say, within a short time α, shorter than the normal slot length.
Then allows everyone to agree to shorten an idle slot, to length α.
This can increase the throughput.
Analyzes this case, using Markov chains.

3.4 Discussion

The paper has an interesting discussion of the reasonableness of various assumptions, and describes and analyzes a variety of strategies.

It advocates more work on strategies for settings with:
Realistic communication assumptions (noise, error probabilities).
Allowance for random/bursty arrivals.
Unknown, dynamically-changing set of participants.
Only a few contending at a time.

Of course, with good math analysis.

4 Komlos, Greenberg

An Asymptotically Nonadaptive Algorithm for Conflict Resolution in Multiple-Access Channels, 1985

A nice little 4-page math paper.
They discuss the tree algorithms that Gallager describes.
Their main contribution is that they achieve the same bound (on time to resolve collisions) as the best known tree algorithm; however, they do not require any feedback whereas the tree algorithm does.
On the other hand, their algorithm needs to know k, which is a bound on the number stations that actually want to transmit.

4.1 The problem

Describe a more general problem than the one they actually solve, so they can compare their algorithm to others'.
n total devices.
Slots numbered $1, 2, 3, \ldots$.
Some number k of the devices have messages to send.
Regard this as an input set I, of size k, $|I| = k$.
Assume this input arrives at the beginning, that is, the devices in I learn at the beginning if they have something to transmit.

In the algorithm, at each step, some stations among those in I can try to transmit:

- If no devices try, then no packets are transmitted.
- If exactly one tries, the packet is successfully bcast.
- If two or more try, collision, no packet is successfully bcast.

Moreover, in any case, all stations receive feedback $(0, 1, c)$, immediately.

Can regard this as capturing the conflict resolution phase of an algorithm that alternates normal phases (anyone can try), and conflict resolution phase (just those involved in a recent conflict can try).

At each slot, the algorithm running on the stations tells some of the stations to try to transmit.

Cost measure: Worst-case number of slots before every station succeeds in transmitting.

Tree algorithm: Gives good worst-case bound, $O(k + k \log n/k)$ as in this paper. But, the nodes use feedback—using $(0, 1, c)$ results from earlier slots to decide on whether to transmit in next slot.

4.2 The nonadaptive case

In this paper, consider nonadaptive case.
That is, the "algorithm" is simply a list $Q_1, Q_2, \ldots, Q_t$ of query sets, chosen ahead of time, independently of the input set I.

Note: The transmitting stations at a particular slot are *not* chosen to be exactly those in the corresponding query set.
Rather, they are those that have not succeeded in transmitting (alone) at an earlier slot.
Thus, although the list of query sets is chosen ahead of time, independently of the input set I, the actual transmission sets, say $I_0, I_1, I_2, \ldots$ do depend on I.
Thus, in a sense, the stations are using a little feedback—each station uses knowledge of whether or not it has succeeded in the past to decide whether it will try again.

A bit more formally: They define I_j to be the set of contenders that still remains to transmit, after slot j.
Thus, I_0, the set that wants to transmit at the beginning, is defined by $I_0 = I$.
But then the later sets are defined recursively, by:
$I_j = I_{j-1} - Q_j$ if $|I_{j-1} \cap Q_j| = 1$, and
$I_j = I_{j-1}$ otherwise.

If $|I_{j-1} \cap Q_j| = \{x\}$, a single station, we say that slot j *isolates* x.
That means it succeeds in letting it transmit alone.
The goal is to isolate all the stations in I.

Cost measure is the total length t of a sequence that always succeeds, for any I, in isolating all the stations in I.

4.3 A fast nonadaptive algorithm

Their result is a nonadaptive algorithm (sequence of query sets) with the same bound as the known adaptive algorithm; that is, the length t of the sequence is $O(k + k \log n/k)$.
Well, the theorem here actually simply asserts the *existence* of such a nonadaptive algorithm—it does not actually find one!

The algorithm relies on knowledge of k.
However, they claim it can be modified to an algorithm that doesn't know k, by a standard successive doubling trick. Sounds right.

As an intermediate step in the construction, it's useful to define what it means for a list of queries to isolate some fraction of the stations, not necessarily all of them:

Definition:
A list of queries is (α, k, n)-*universal* if it isolates all but at most $(1 - \alpha)k$ of the members of I for every I with $|I| \leq k$.

The overall goal can be restated as constructing a $(1, k, n)$-universal list of queries, of length $O(k + k \log n/k)$.

Suppose for simplicity that k divides n, and k is even (these aren't essential).

Suppose we choose a sequence of query sets $Q_1, Q_2, \ldots, Q_p$ independently, with each set chosen uniformly at random from the set of all subsets of the n stations of size exactly n/k.
Moreover, the number of sets $p = k/2$.

The first lemma says that any particular query set Q_j in the sequence isolates some member of the input set, with at least a certain probability. This probability taken over the choices of query set Q_j only—it works whatever the choice of the previous query sets.

Lemma 1: For any I, and any $Q_1, \ldots, Q_{j-1}$, the probability that Q_j isolates some member of the input set I is $> 1/(2e^2)$.

Proof:
By brute-force calculation.
The case where $k = n$ is an easy boundary case.
($|Q_j| = 1$, and at least half of the entire set of n processes are still trying to transmit. So the chance is at least $1/2$ of choosing one of these.)

So we can assume that $2 \leq k \leq n/2$.

The first expression gives probability of isolating some member of the still-active set I_{j-1}.
Here x denotes $|I_{j-1}|$.
The denominator, $nchoose(n/k)$, is simply the total number of ways to choose Q_j.
The numerator represents the number of choices of Q_j that successfully isolate some member of I.
In the numerator, the first factor x gives the number of choices for the station to isolate.
Suppose that is in Q_j.

The second factor, $((n-x)choose(n/k-1))$, is the number of ways to choose the remaining $n/k-1$ elements of Q_j from among the non-transmitting stations.
Those can be stations that already succeeded in transmitting, or stations that weren't in the input set I to begin with.

After this, the analysis involves calculation.
Expand the binomial coefficients into factorials,
use the fact that each fraction in the resulting expansion is $\leq 1/2$, and the fact that for $u \leq 1/2$, we have $1 - u > e^{-2u}$.
The result is a lower bound that is a harmonic series.
Use an integral approximation for the resulting harmonic series,
and finally, use the approximation expression we have seen before, for $(1 - 1/n)^{n-1}$.

QED

So, we see that, regardless of what has happened before, the probability of the next query set isolating some station is bounded from below. Now use this repeatedly, for the whole series of $p = k/2$ queries:

Lemma 2: For some constants b and c, the probability that $Q_1, Q_2, \ldots, Q_p$ isolates at least ck members of the input set I is $> 1 - 1/e^{bk}$.

Proof: Simple application of Lemma 1, with binomial distribution analysis.
QED

Next, convert Lemma 2, about probabilities, into a theorem about certainties, for isolating a certain fraction of the inputs.
However, this statement just says "there exists" some sequence of queries that yields the needed guarantee—it doesn't actually produce one.

Theorem 1: For every k, n, $2 \leq k \leq n$ (with the two other assumptions about k and n), there exists a (c, k, n)-universal list of query sets of length $O(k + k \log n/k)$, where c is the constant in Lemma 2.

Proof: The idea is to use enough queries so that, for every I, the probability that I is a "bad input set" (does not get enough stations isolated) is strictly less than $1/(nchoosek)$.
This probability is taken over the random choices of the query sets, as usual.
Then the expected number of bad input sets is the number of input sets I, which is $(nchoosek)$, times this probability.
This implies that the expected number of bad input sets is < 1.

If the expected number of bad input sets is less than one (expectation taken over the query lists), then there must be some particular query list (but we don't know what it is, only that it exists) that guarantees that there are no bad input sets.

To get the probability that each I is a bad input set to be sufficiently low ($< 1/(nchoosek)$), we use Lemma 2 repeatedly, on m batches of queries, each of length $p = k/2$.

This upper-bounds the probability that every batch fails to isolate enough stations, by e^{-bmk}.

(See this by multiplying probabilities that all fail. Actually, this assumes that the input set I is restored after each batch. But they claim that if they don't do this, the probability of failing to isolate becomes even less.)

By choosing m as indicated in the proof, approximately $log(nchoosek)/bk$, we get this probability to be $< 1/(nchoosek)$, as needed.
And the length of the list isn't too great: $t = mp$ gives $log(nchoosek)/bk$ times $k/2$, which is $O(k + klog(n/k))$.

(Use Stirling's formula to show $(nchoosek) \leq (en/k)^k$.
Then take logs, getting $\log nchoosek \leq k(\log e + \log(n/k))$, which is equal to $k + k\log(n/k)$.)
QED

Finally, we concatenate (c, k_i, n)-universal lists, in a geometric progression of lengths, to get a $(1, k, n)$-universal list.

Theorem 2: For every k, n, there is a $(1, k, n)$-universal list of length $O(k + k\log n/k)$.

Proof: Start with $k_1 = k$, get a universal sequence.
Then use $k_2 = k(1 - c)$, since we've already removed kc from the set...
Repeat, each time multiplying by $(1 - c)$.
The sequences are of successively decreasing lengths.
When we add them up, we still remain within the same (order of magnitude) bound.
And they isolate enough to get < 1 remaining, which means they must isolate everything.
QED

4.4 Discussion

They give directions for further work:
Tighter analysis.
Constructive proof—actually produce an algorithm instead of just asserting one exists.
Close gap between upper and lower bounds.

In view of Gallager's remarks, it would also be nice to weaken the assumption that a single transmitter must succeed.