Data Link Layer

Lecture 7 Framing and introduction to CRC

Frame

In the data link layer, we try to "make sense" of the incoming bits

which bits together mean something in the stream of data.

↳ The resulting unit is called a frame.

In a frame, we first need to demarcate where the section begins/ends.

HDLC

High-Level Data Link Control (HDLC) is used as Layer-2 technology in Wide Area Networks.
(WANs)

The (HDLC) frame has a

begin sequence,	8 bits
header,	16 bits
body,	variable
CRC, and	16 bits
end sequence.	8 bits

The begin/end sequence are the same: 01111110

If there is nothing to send, we continuously send this

It also helps in clock synchronization.

Bit Stuffing

What if this sequence appears elsewhere?

We do bit stuffing. Say 01111110 is somewhere in the middle.

↳ we insert bits.

What HDLC does is:

→ If you see 5 consecutive 1s, insert a 0.

01111100011111100011111110

↓

01111100001111101000111110110

At the receiver,

we somehow have to remove these stuffed bits.

Wherever you see 5 consecutive 1s, remove the subsequent stuffed 0.

The end sequence still has 6 consecutive 1s.

But what if there is some bit error?

$$11111 \begin{cases} 0 \rightarrow \text{Remove (bit stuffing)} \\ 10 \rightarrow \text{Assume end sequence} \\ 11 \rightarrow \text{Assume error has occurred and discard the frame} \end{cases}$$

(We discard everything until we see the sequence again)

This is very barebones though, we need something better for errors.

Cyclic Redundancy Check (CRC)

CRC

We just append the k-bit CRC (=16 here) to the n-bit dataword to get a "codeword".

The space of datawords is the set of all 2^n bit words.

We keep it such that only 2^n of the 2^{n+k} $(n+k)$-bit strings are valid.

An issue only arises when the error is such that the erroneous string is a codeword as well.

$\Rightarrow$ We need to ensure that codewords are "far apart".

Given $v, w \in \{0,1\}^n$, the Hamming distance between v and w is

$$d(v,w) = \{i \in [n] : v_i \neq w_i\}.$$

Hamming distance

(number of positions they are distinct)

For a "code" $C \subseteq \{0,1\}^n$: the Hamming distance of C is

$$\min\{d(v,w) : v,w \in C,\ v \neq w\}.$$

We want this distance to be large.

$GF(2) = \mathbb{F}(2)$ is a finite field with elements $\{0,1\}$.

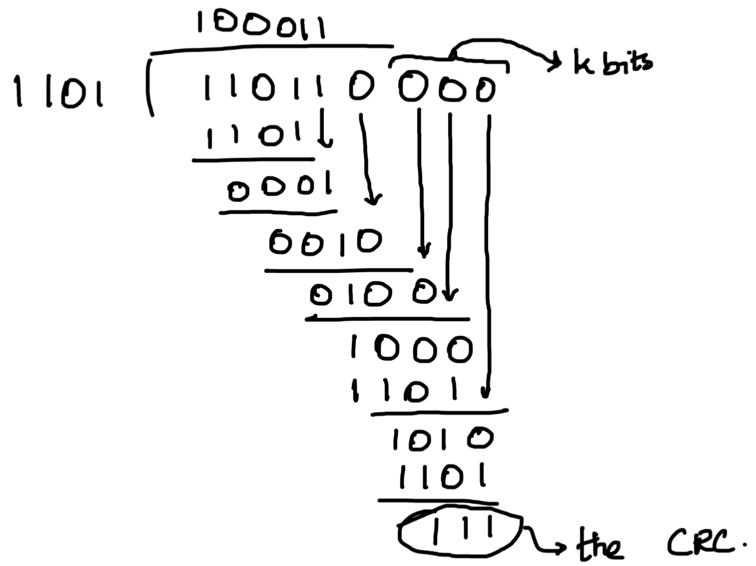
Addition $(+)$ has identity 0. (Note that $a+b = a-b$)

Multiplication $(\times)$ has identity 1.

CRC is based on a cyclic code
If v is a codeword, cyclic shifts of v also result in codewords.

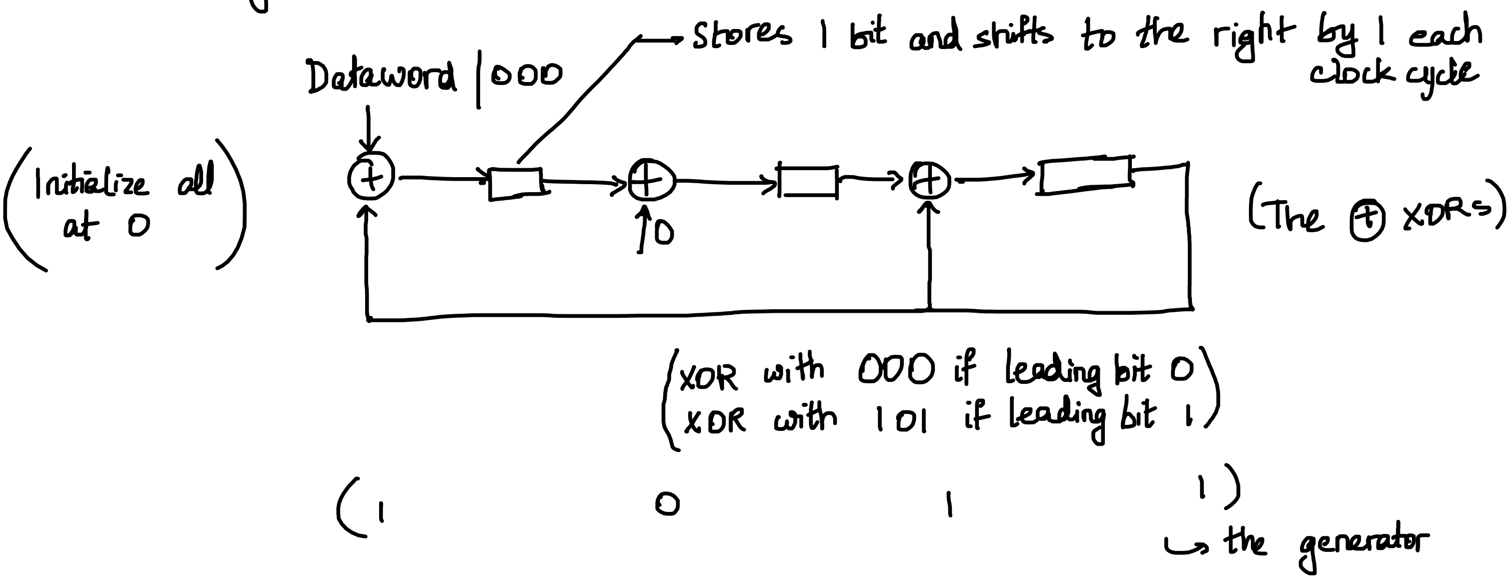
To generate a CRC, we use Long division (in $\mathbb{F}_2$)
The divisor/generator is of length $k+1$ bits.
For example, say $k=3$ and the generator or "divisor" is 1101 and
$n=6$ with dataword 110110

Generator
Divisor



The codeword is then $\underbrace{110110}_{\text{dataword}}\underbrace{111}_{\text{CRC}}$.

At the sender,
Long division for a given generator can be implemented using shift registers.
For generator 1101,



After Dataword|000 is emptied, the CRC is left in the shift registers.

At the receiver, either

1. pass the dataword with k 0s through the CRC circuit and verify that the CRCs match or
2. pass the entire received word through the CRC circuit and verify that you get all 0s.

Lecture 8 CRC Polynomial Arithmetic

What bit errors can the CRC detect?

We will represent the divisor as a polynomial

For example, $1101 \equiv 1 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0 = x^3 + x^2 + 1.$

$1101 \times 11 = 10111$

$\hookrightarrow x^4 + x^2 + x + 1.$

$(x^3 + x^2 + 1)(x+1) = x^4 + 2x^3 + x^2 + x + 1 = x^4 + x^2 + x + 1$

($2 = 0$ in $\mathbb{F}_2$)

So say we transmit some codeword with equivalent polynomial P(x). Let the error bitstring be E(x).

The received word is then P(x) + E(x).

We divide the received polynomial by C(x) and if the resultant is 0, we say there is no bit error

$\hookrightarrow$ poly. corresponding to generator.

We want

$$\frac{P(x) + E(x)}{C(x)} \neq 0 \quad \text{if } E(x) \neq 0.$$

→ Single bit errors.

$E(x) = x^i$ for some i.

$$\frac{P(x) + E(x)}{C(x)} = \cancelto{0}{\frac{P(x)}{C(x)}} + \frac{E(x)}{C(x)} = \frac{E(x)}{C(x)}.$$

If $C(x) = x^k + \underbrace{\cdots}_{0\text{s or }1\text{s}} + 1$, single bit errors can be detected.

$$E(x) = \underbrace{C(x) \cdot D(x)}_{\text{will have at least two non-zero powers of } x.}$$

(So for example, 1101 can detect single bit errors)

→ 2-bit errors.

$E(x) = x^j + x^i = x^i(x^{j-i} + 1)$ (suppose $j > i$)

Write each polynomial as a product of irreducible polynomials.

$$\frac{E(x)}{C(x)} = \frac{g_1(x) \cdots g_t(x)}{f_1(x) \cdots f_m(x)}$$

If C is of the form $x^k + \cdots + 1$, no $f_r(x)$ is of the form x^β.

$\Rightarrow$ no f_r will divide the x^i (if $i > 0$).

However, we could have

$$(x^k + \cdots + 1)(\cdots) = x^\gamma + 1 \text{ for some "large" } \gamma.$$

So if $j-i$ is large (the errors are far apart), the bit errors might not be detected

The smallest γ such that $C(x)$ divides $x^\gamma + 1$ is called its order.

It is known how to find $C(x)$ of the form $x^k + \cdots + 1$ such that it has order $2^k - 1$.

So while this limits the length of the codeword that can be transmitted, it gets the job done quite well.

So if we have a 16 bit CRC, we can transmit data with at most $2^{16} - 16$ bits while detecting 2-bit errors.

→ Odd-bit errors

$E(x) = x^{i_1} + x^{i_2} + \cdots + x^{i_{2r+1}}$.

If $C(x)$ has $(1+x)$ as a factor, it cannot divide x of this form. Indeed,

$$\underset{\substack{\uparrow \\ \text{substituting } x=1}}{(1+1)} D(1) = 0 \quad \text{but} \quad E(1) = 0.$$

$D(x) \rightarrow 000111111000$

$xD(x) \rightarrow \underline{001111110000}$

001000001000 ↳ each string of 1s in D two 1s in $(1+x)D(x)$

If $C(x)$ has an even number of terms (x^i), it can detect any error $E(x)$ with an odd number of terms

HDLC uses CRC-16-IBM

$C(x) = x^{16} + x^{15} + x^2 + 1$

CRC-32 has

$$C(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1.$$

Lecture 9 ARQ

How do we detect a burst of errors?
That is, a bunch of contiguous bits become erroneous.
The error is of the form

$$E(x) = x^{i+l-1} + x^{i+l-2} + \cdots + x^{i+1} + x^i$$

Let the CRC be of the form

$$C(x) = x^k + \underbrace{\cdots}_{\text{anything}} + 1$$

To be undetected, $C(x)$ divides $E(x)$.

$$\frac{E(x)}{c(x)} = \frac{x^i \overbrace{(x^{l-1} + x^{l-2} + \cdots + 1)}}{\underbrace{x^k + \cdots + 1}_{\text{no } x^l \text{ factor}}}$$

this is the part that matters

If $l-1 < k$, then $C(x)$ cannot divide $x^{l-1} + x^{l-2} + \cdots + 1$.
So bursts of length at most k can be detected.

—

ARQ

Another thing we shall study is ARQ - Automatic Repeat reQuest
Sometimes, if the link between two nodes is unreliable, some of the handling of reliability is done by the DLL. (TCP is quite slow)
WiFi uses ARQ.
(the wireless link is prone to errors)

Suppose the sender sends out a frame that the receiver receives δt later.
How do we know if the frame has got there reliebly?

speed of light delay

The receiver sends out an acknowledgement that is received after δt.
(ACK)

The ACK frame is quite small, it just acknowledges the specific frame.
How long should the sender wait for the ACK?
This is quite non-trivial in TCP because there are factors such as where you are located.
Here, however, we are just concerned about a single link.
We need to figure out what the RTT is.
(first bit of sent frame from sender to last bit of ACK frame to sender)
The distance between the (WiFi) access point and the device is usually a few 10s of meters.
It will take a couple of nanoseconds (not insignificant).
Wait 2–3 times the RTT and if ACK not received, retransmit.

Note that the time also depends on the size of the frame and ACK.

What if the ACK frame is in error / did not reach the sender? A

Timeout

timeout is said to occur. The sender assumes the worst case and resends the frame after the timeout (although the receiver has already received the frame).

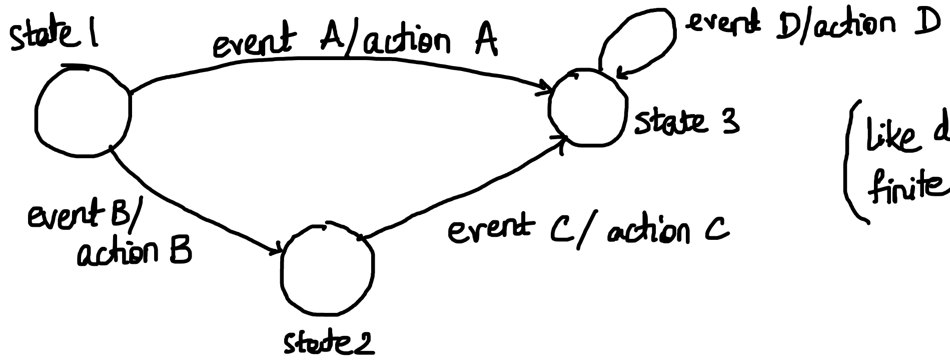
It is also possible to run into problems if we set the timer to be too small. We receive the ACK for a previous version of the (same) frame while we are waiting for the current frame.

If the timer is too large, then we waste a lot of time if there is an issue with the frame (or ACK).

We must set the timer appropriately to cut down on resources and redundancy.

We often represent protocols using state diagrams or flowcharts.

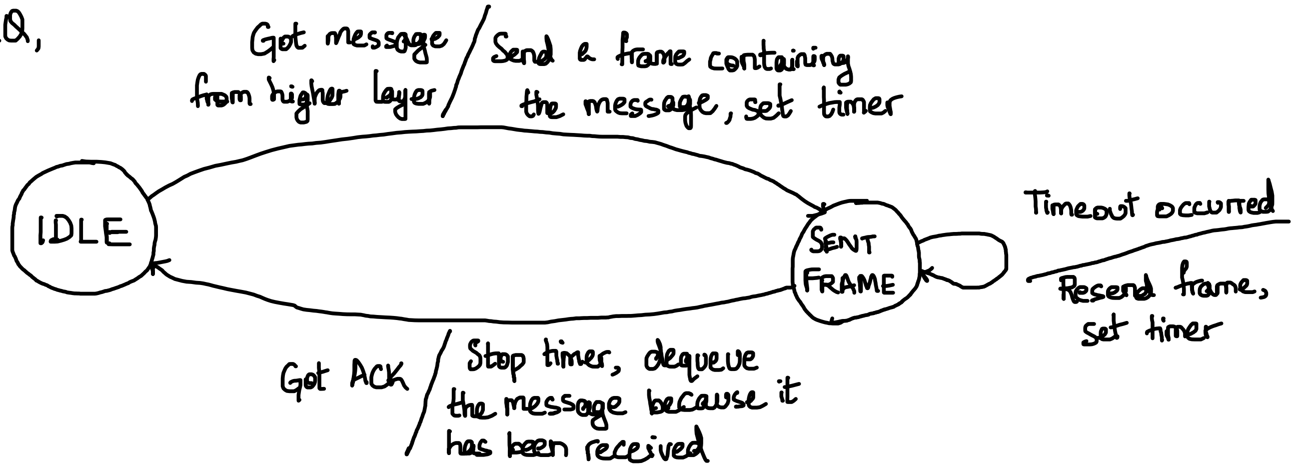
For example,



(like deterministic finite automata)

Each action is represented by a flowchart.

In ARQ,



Wireless systems usually use ARQ.

Lecture 10 Medium Access

Recall the issue of medium access that we had (briefly) looked at.
The wire there is the "medium". If everyone tries to use the medium at the same time, the messages will collide / interfere.
It is more problematic in the wireless case since the signal decays very fast with distance.

CSMA

For now, we shall look at Ethernet LAN, which uses CSMA - Carrier Sense Multiple Access.
There is a central bus (wire) and we want to prevent multiple people from simultaneously transmitting.

One idea is to have some central authority whose permission you need (this is used in wireless cellular networks).

Further, there is the issue of how this authority knows when someone wants to transmit. If we know how many nodes there are, we could do TDMA (Time Division Multiple Access) — give regular time slots during which (only) they can transmit. Obviously, this leads to massive inefficiency. What if a new person joins? What if the authority goes down?

Broadcast
Unicast
Multicast

If a node has data to send, it creates a frame and sends it.
Broadcast signals are sent to everyone on the network.
The most common are unicast signals - there is a single destination node. There are also multicast signals.
This part of the pertinent information should be mentioned as well.
In this scheme, what if there is a collision? Neither receiver receives the information If the information is sent by both users again, there could be another collision.
We also need a mechanism for collision detection.

If a collision is detected, we shouldn't have both senders try again immediately.
CSMA is a random access protocol — the schedule for transmission is not decided in advance.
At each sender node, we back off for some random amount of time before retransmitting.
The "Carrier Sense" means that if someone else is transmitting, we wait before sending our own message This resolves the issue that occurs when one person retransmits slightly before the other.
This could still be problematic if many people are trying to communicate simultaneously, so we need to refine this idea.

IEEE 802.3 uses CSMA - CD
↳ Collision Detection

To do carrier sense, we can just check if there is some reasonable amount of energy on the wire.
(sufficiently larger than the noise energy)

How do we perform collision detection?
In wireless, just check if ACK is received.
In wired, when the signals add up, we could do something with Fourier analysis. More simply however, we see that the energy of the signal increases after two signals add up.

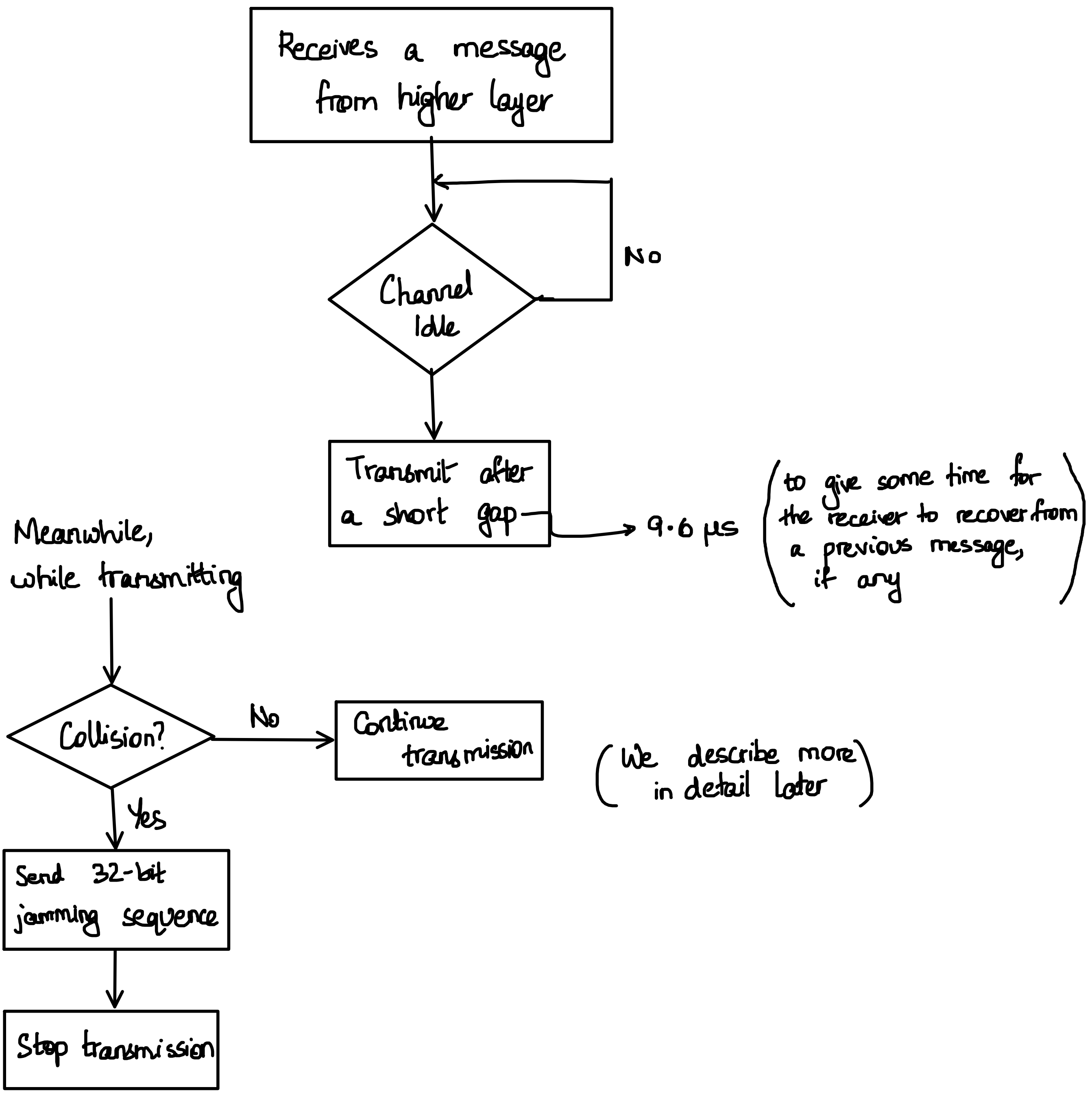
Keep another threshold. If the energy is greater, then declare that a collision has occurred.

So we have two thresholds – one for carrier sense and one for collision detection

Another issue is: what if one side detects the collision much earlier and stops transmitting so the other side does not detect a collision?

So after the first side realizes there is a collision, it sends out a (short) jamming signal so the other side detects a collision (it is guaranteed that detection will happen by the time the jamming signal ends)

It is worth noting that there may be multiple "other side"s.



Frame details in 802.3:

Preamble	Start Frame Delimiter	Destination MAC Address	Source MAC Address	Length	Payload	CRC
7 bytes	1 byte	6 bytes	6 bytes	2 bytes	46-1500 bytes	4 bytes

⟵ 64 – 1518 bytes ⟶

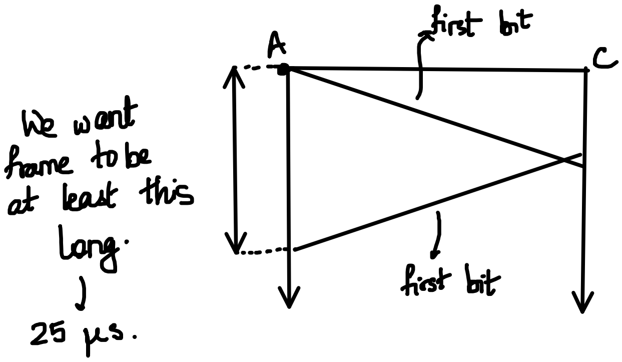
Why minimum frame size?

We assume that the WAN is at most 2500 m wide.

$$RTT \simeq \frac{5000}{2 \times 10^8} = 25\,\mu s$$

close to speed of light but not exactly

If the frame transmission time is (much) larger than the RTT, it is possible that we do not detect a collision before the frame finishes transmitting.



Along the way, we also have repeaters that reproduce the signal. This may add some more time, so let us conservatively take 50 µs (instead of 25 µs).

64 bytes at 10 Mbps → 51.2 µs.

If we have larger distances or higher rates, the minimum frame size would be made larger.

Why maximum frame size?

1. If we have a larger frame, the probability of getting an erroneous bit is high.

   If p is the probability of a particular bit going wrong (assume iid), the probability of getting an error is $1-(1-p)^n$ ← length

2. Other people do not get an opportunity to transmit for a long time.

3. Memory requirements at the NIC card go up (we need to store it for checking the CRC).

Finally, what distribution do we pick the random backoff from?

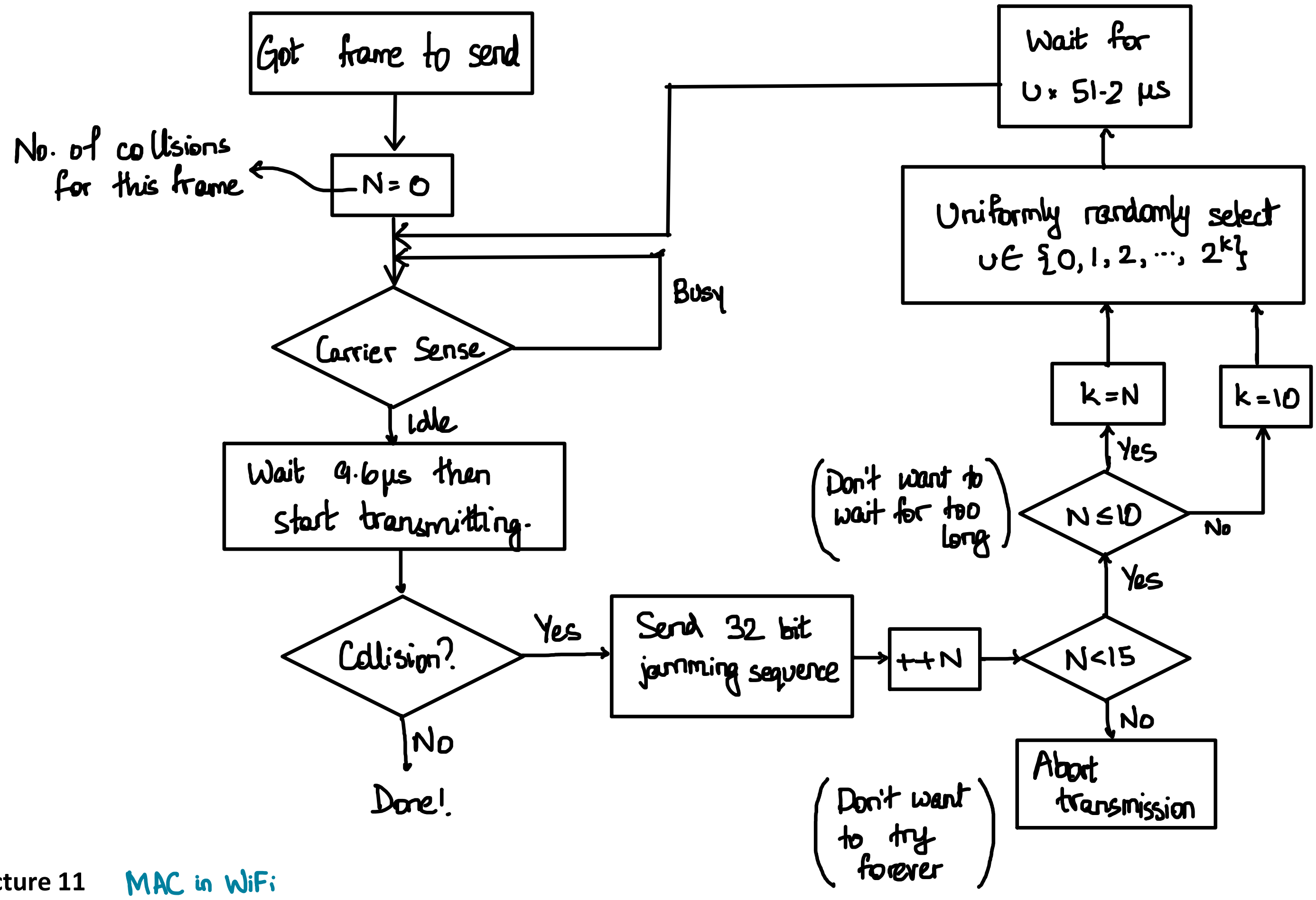
If Δ is the minimum wait time, we pick uniformly from $\{\Delta u : u \in \{0, 1, \cdots, 2^k\}\}$. We want k to be larger if more people have collided. How?

Backoff

What we do is that we initially pick some (small) k. If a collision occurs, we increase k (and recurse). This system is known as exponential back off.

If we use linear instead, we might end up waiting for a long time before no collision occurs

A more complete flowchart of the above information is given on the next page.



Lecture 11 MAC in WiFi

Wireless LANs have ideas similar to wired systems, but slightly modified.

Say we have two laptops connected to the same access point that want to communicate.

What if we use CSMA-CD? The problem here is that the signal decays very fast, so it might be interpreted as noise by other systems performing carrier sense (it decays as $d^{-\alpha}$ for some $2<\alpha<\infty$). For the same reason, collision detection is not very effective either.

So what do we do? Assigning frequencies might work, but who would assign frequencies?

Note that carrier sense could work if the senders are close by, so lets leave it in.

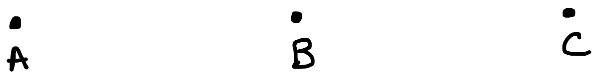
Can we try cancelling out our own signal? The received might not be very close to the transmitted signal due to noise (in case of a collision).

Why not use acknowledgements? First do carrier sense for some time. If free, start transmitting. Then just hope that the receiver has received it correctly and wait for an acknowledgement.

If there is no acknowledgement, assume a collision has occurred.

There is another issue called the hidden terminal problem.

Hidden Terminal



B can hear both A and C but A and C cannot hear each other.
If A→B and C→B, there is a collision at B. Neither receives an acknowledgment but a lot of time is wasted.

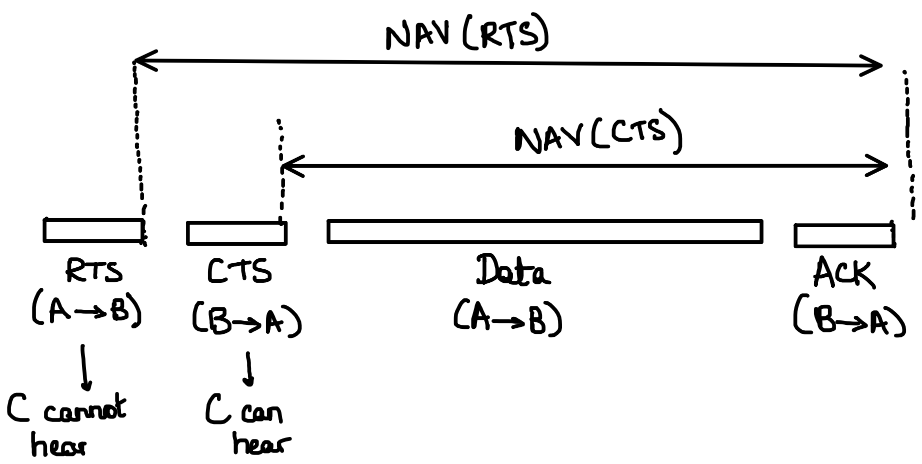
Virtual Carrier Sensing

WiFi uses Virtual Carrier Sensing. Maybe before A transmits to B, it sends a short request to send frame to tell B that it intends to send. B then tells its neighbours about the same then send A a short clear to send frame. Then when C hears the CTS, it knows that it should not transmit.

RTS
CTS

NAV

We can also include in the CTS how long the channel will be busy.
(time for A's frame + ACK to A)
This is known as the Network Allocation Vector. This tells B's neighbours how long to stay silent.
To do this, the RTS itself includes a NAV.



The rule is that anyone hearing a RTS or CTS remains silent for the corresponding NAV. We want silence around both the sender and receiver.
If A doesn't hear CTS, it doesn't transmit.

If there isn't much contention in the network, we might not use RTS/CTS to save time.

RTS/CTS are usually sent using lower modulation (like BPSK) so the duration might be large — we want it to be heard by a lot of people.

Lecture 12 Collisions in WiFi

CSMA-CA

The overall WiFi algorithm is called CSMA-CA
↳ Collision Avoidance

Exposed Terminal

Another problem is the exposed terminal problem.
Say we have

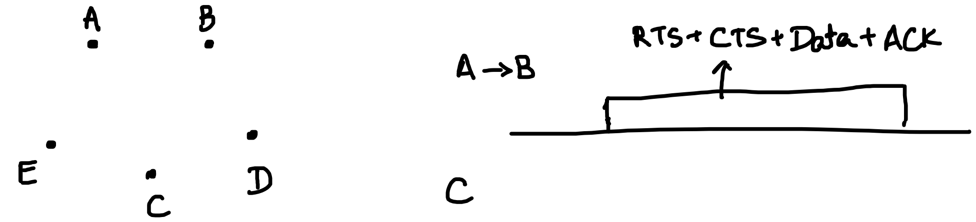
D A B C

Each can hear neighbours.

In theory A→D and B→C could be transmitted simultaneously
But Carrier Sense prevents this because A and B are exposed to each other.
(within hearing range)

Contention Window

Each will have a contention window (CW).



DIFS

Say A is transmitting to B, A has more data to transmit after the current frame, and C has data to transmit as well. After the frame is done, each waits for a fixed time Distributed Inter-Frame Spacing (DIFS).
This gives some notion of priority (if you wait less, you can transmit).

SIFS

The time gap between the RTS/CTS or CTS/Data or Data/ACK is called the Short Inter-Frame Spacing (SIFS).

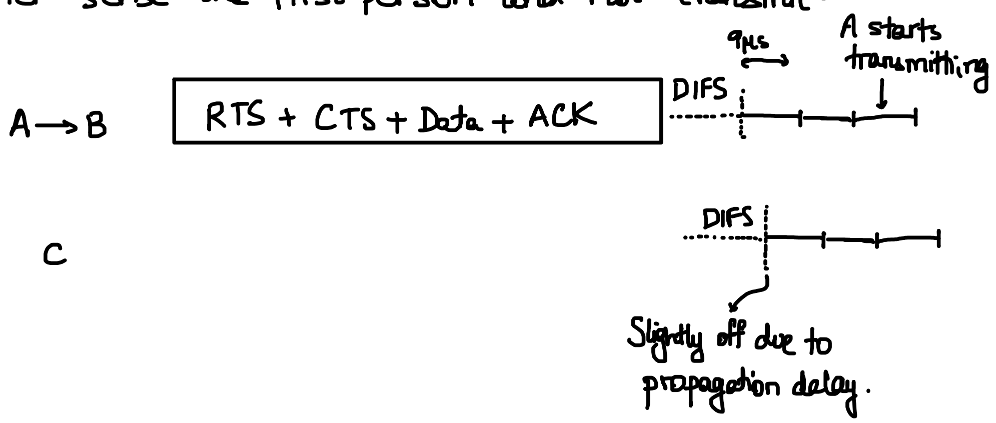
Obviously, the DIFS should be greater than the SIFS.

Why do we need SIFS?

- → Wait for processing (decoding and stuff — PHY/MAC(DLL) processing)
- → B hears RTS, then NIC switches from receive to transmit (which takes some time)
- → A sends RTS, change from transmit to receive.

After the DIFS, each waits for a random amount of time.

Divide time into slots (9 μs in WiFi). If someone starts transmitting in a slot, someone who is waiting for the next slot to transmit should be able to carrier sense the first person and not transmit.



So the slot length should take propagation delay, offset in slots, and time to carrier sense into account.

C must hear and carrier sense A within the slot if it starts transmitting.

Further, if A does not transmit, it needs time to switch from receive to transmit mode.

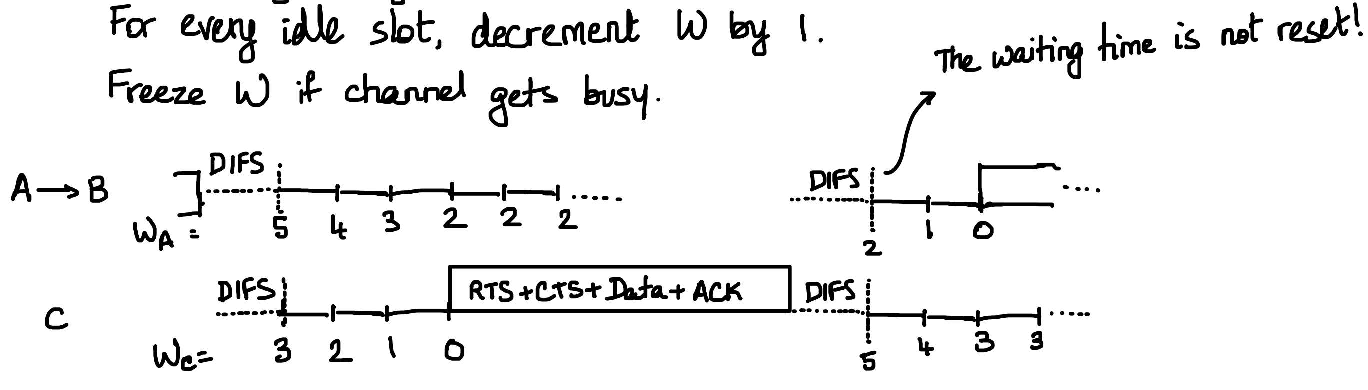
After the DIFS, we wait for some random number of slots. This is where contention window enters the picture.

Initialize $W = \text{Unif}(0, CW_{max})$

W → Remaining waiting time

For every idle slot, decrement W by 1.

Freeze W if channel gets busy.



Note that the waiting time is not decremented during the SIFS.
We detect a collision if → sent RTS, no CTS
→ sent data, no ACK.

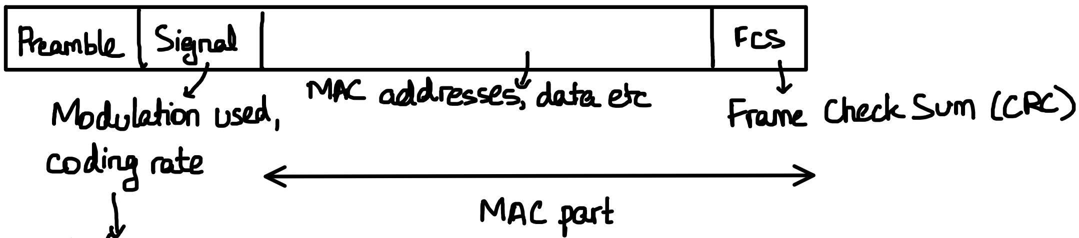
In this case, we do

$$CW_{max} = CW_{max} \times 2$$
$$CW_{max} = \min(CW_{max}, \text{Maximum allowed value})$$

and then choose the (random) waiting time again.

The name of this protocol is IEEE 802.11. (b, g, n, ac, ax)
ac ↳ common
ax ↳ latest

The frame has



We "code" the data before passing it to the channel to protect against bit errors.

Rate

The rate is the degree of redundancy $\frac{n}{k}$ (n → number of bits, k → number of information bits)
WiFi has ~ $\frac{3}{4}$ or $\frac{5}{6}$.

		rate	bit rate (during frame)	channel width
802.11 g	64-QAM	(3/4)	54 Mbps	20 MHz
802.11 n	64-QAM	(5/6)	150 Mbps	40 MHz
802.11 ac	256-QAM	(3/4) or (5/6)	866 Mbps	upto 160 MHz
802.11 ax	1024-QAM	(3/4) or (5/6)	1.2 Gbps (SISO)	upto 160 MHz

Codes help us detect and possibly correct codewords
In MIMO (Multiple Input Multiple Output), the sender/receiver have multiple antennas. The channel width is the width of the frequency bandwidth. Changing nothing else, bit rate ∝ channel width.
Using MIMO instead of SISO could drastically increase the bit rate.

Typically, DIFS = SIFS + 2×(slot time)
PIFS = SIFS + (slot time)

PIFS

↳ Point Coordination Function — intermediate priority
more centralized

DIFS uses the Distributed Coordination Function.

Now, what about the QoS? Is it possible to give higher priority depending on the application?

IEEE 802.11e was created for this. It does not give a hard guarantee.

Application	Voice	Video	Other
CW_{max}(initial)	3	7	15
CW_{max}(max)	7	15	1023

(Rarely used)

To know which packet is there, there is some cross-application interaction between the application layer and DLL.

WiFi is, in general, terrible for QoS.

SIFS = 10 μs for 11g, 11n
= 16 μs for 11ac etc.

Just like in ethernet, there is a maximum number of retries for collisions after which we give up.

WiFi protocols are usually backward compatible.

If we use the Distributed Coordination Function, WiFi is not too hard to set up and get working.

Lecture 12

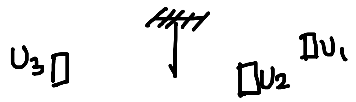
We have discussed CSMA-CD and CA. The government allots particular frequency bands and it is a free-for-all within those bands — anyone can try using them.

How would we ensure that say, our WiFi signals and our neighbour's WiFi signals do not interfere?

Since there is no centralized coordinator, we just use CSMA, a very decentralized protocol.

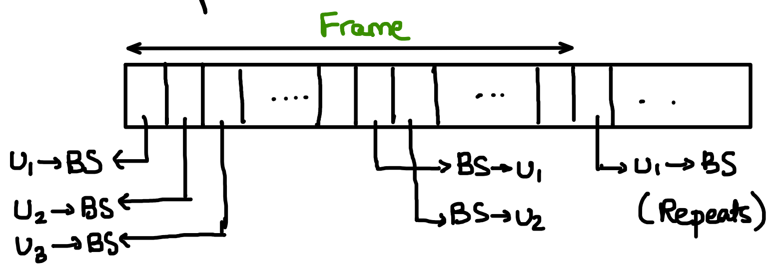
This is for unlicensed bands, what about Licensed bands? The govt auctions frequency bands for exclusive usage by some company.
CSMA isn't the best here because we do have some centralized coordinator

Let us look at cellular networks. The company can use a bandpass filter to ensure that other companies' signals do not interfere.
The geographical area is divided into "cells", each of which has a base station.
Let us look at a single cell for now. We must decide how to divvy up the resources among the mobile users so that they don't interfere.



One idea: Give different time slots to different users. This is TDMA - Time Division Multiple Access.

TDMA



(BS ≡ Base Station)

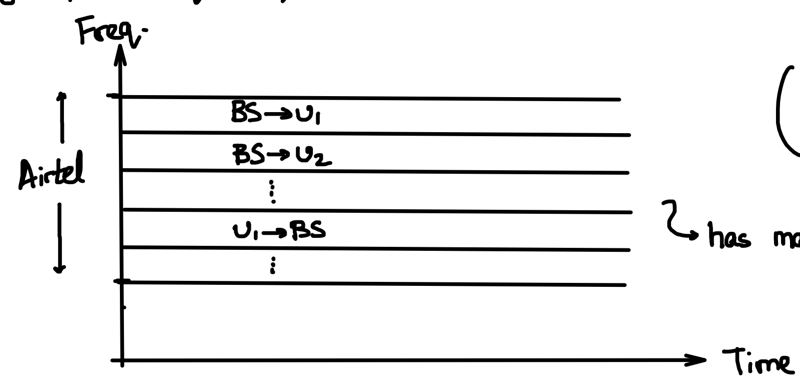
Downlink
Uplink

Some slots in the frame give the schedule — known as the downlink map and uplink map.

Guard
Band

To prevent interference, there is a guard band separating different companies' frequency bands

Can we similarly split frequency into slots?



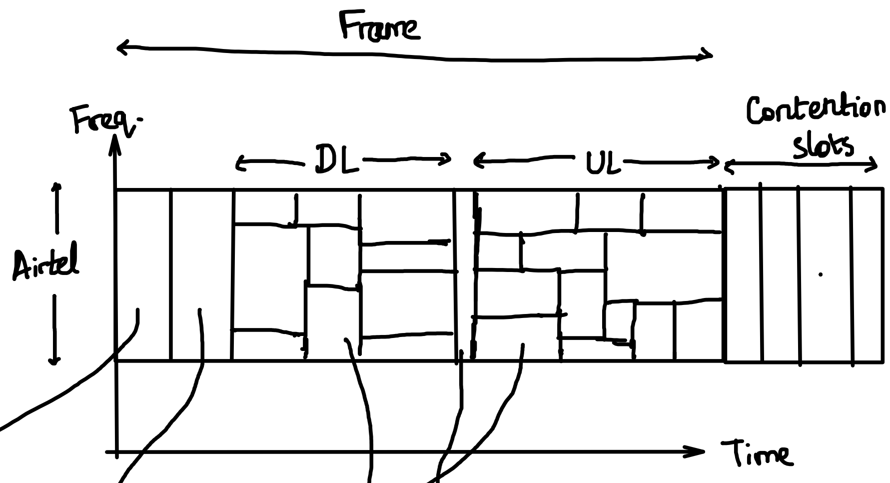
(Slots in the freq. domain)

has many guard bands

FDMA
OFDMA

This is FDMA - Frequency Division Multiple Access.
4G-LTE uses OFDMA - Orthogonal FDMA. It is a combination of TDMA and FDMA.

Tiles



Short gap (to allow switching b/w send and receive)

DL/UL map

Preamble:
- → Check synchronization with B.S.
- → Demarcate frame start
- → Estimate attenuation and phase changes

Tiles : multiple may correspond to a single user

We can't afford to have guard bands here because that would waste a lot. OFDMA essentially allows us to transmit without guard bands (we shall not study it)

It turns out that the attenuation in different tiles is different. Some slots at the end of the frame allow the users to tell the BS which tiles are good for them. This may not stay the same, a few frames later, the DL/UL map may change to benefit everyone.

We also only allocate slots to people who want to upload/download. This is where contention slots enter the picture, during which anyone can transmit. A user says they want to transmit. To do so, they use some sort of CSMA scheme
The DL/UL map could be used as a sort of acknowledgement or alternatively, the BS could just acknowledge during the DL window.
If a collision occurs during contention slots, they do a random backoff.

Orthogonal Frequency Division Multiplexing

In OFDM, in each narrow band, it is like we have a completely different center frequency f_c'. Distinct narrow bands can even use different modulations.

We can transmit along separate narrow bands at the same time.

Essentially multiple modulations on narrow bands.

OFDM is used in WiFi.

LTE uses both OFDM and OFDMA.

CDMA

Finally, let us look at CDMA - Code Division Multiple Access, which was used in 3G.

It allows multiple users to use the same time slot.

It uses a spreading code.

Say $u_1 \cdot c_1(t) \times A\cos(2\pi f_c t)$ $\quad 0 < t < \frac{N}{f_c}$ (c_1 and c_2 are spreading codes)

$U_2 : C_2(t) \times -A\cos(2\pi f_c t)$

Spreading Code

These codes are chosen such that c_1 and c_2 are orthogonal.

$$\int_0^T c_1(t)\, c_2(t)\, dt = 0.$$

(How do we generate? If we randomly generate bits (-1,1), then for a sufficiently long string, the codes are orthogonal with high prob.)

All the codes are mutually orthogonal. How do we decode?

$$r(t) = \sum_{j=1}^{n} s_i(t)\, dt.$$

Note that

$$S_1(t) \times c_1(t)\cos(2\pi f_c t) = A\cos^2(2\pi f_c t) \qquad (c_1^2(t) = 1)$$

$$= \frac{A}{2}\Big(1 + \underbrace{\cos(4\pi f_c t)}_{\text{removed by a low pass filter}}\Big)$$

$$S_2(t) \times c_1(t)\cos(2\pi f_c t) = -A\, c_1(t)\, c_2(t) \cos^2(2\pi f_c t)$$

$$= -\frac{A}{2}\, c_1(t)\, c_2(t)\Big(1 + \underbrace{\cos(4\pi f_c t)}_{\text{removed by a low pass filter}}\Big)$$

So we are left with something like

$$\frac{A}{2} + \underbrace{\left(\frac{-A}{2}\, c_1(t)\, c_2(t)\right)}$$

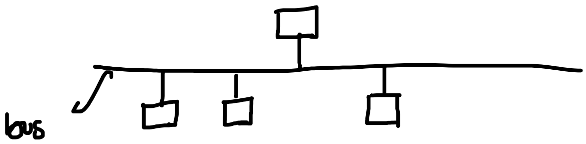
integrating from 0 to T removes this and just gives $\left(\frac{A}{2}\right) \cdot T$

$\Rightarrow$ Not too hard to get the original data back if you know the codes.

CDMA and OFDM are robust to multipath.

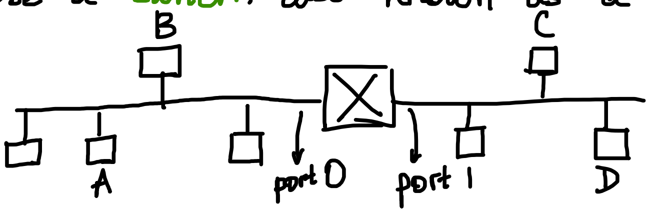
Lecture 14 Ethernet Switching

Layer 2 switches are in the ethernet and use the MAC address to switch.
Layer 3 switches (routers) use the IP address.



This topology does not easily scale to more than a few 10s of nodes.

$\Rightarrow$ We use a switch, also known as a bridge.



A switch selectively forwards intelligently.
$A \to C$ would transmit but not $A \to B$.

It has a forwarding table.

Destination	Port No
A	0
B	0
C	1
D	1

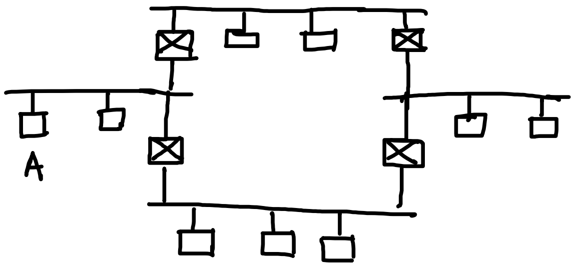
So if it hears a message for B at port 0, it does not forward.
How does it get to know the forwarding table? Automatic dynamic updation would be ideal.

Any device has a unique 6-byte MAC address hardcoded in the card. Each port of the switch has a MAC address as well.

Initially, the forwarding table is empty. We now connect it.
Say it hears A → B.
It now knows that A is at port 0 but since it doesn't know where B is, it decides to forward to port 1
We update the forwarding table based on the sender.
What if we unplug a device from one side and plug it on the other?
Each entry in the table has an expiring time. If it has been a while since we have heard from someone, we delete their entry.

The process is similar if we have more than two ports.
→ We send to all ports if the receiver is not in the table.
→ We update the sender's port in the table.
(further, there is an expiry time)

What about the following situation?



A frame from A could cycle endlessly, thus eating bandwidth.

We use the spanning tree protocol (created by Radia Perlman):

1. Elect a root bridge
2. Each bridge finds which port is closest to the root and assigns this port as the "root port".
   (what is the tiebreaking rule?)
3. Among all bridges connected to a LAN, elect one to forward frames on that LAN. ("designated port")
4. Any port which is neither a root port nor a designated port is disabled.

→ Each bridge has a bridge ID. The root is the bridge with the lowest bridge ID.

$$\underbrace{\text{Configurable part}}_{\text{2 bytes}} \quad \underbrace{\text{MAC address}}_{\text{smallest MAC}}$$

default : 32768

can be any multiple of 4096 from 0 – 16440.

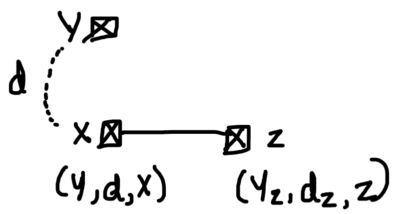
If we want a custom root bridge, we assign a lower value to it.

(the lower the value, the higher the priority)

Each bridge tells its neighbours

(Y, d, X)

- Y → smallest ID heard till now
- d → distance to Y
- X → my ID



if $Y < Y_z$,

$Y_z = Y$ and $d_z = d + dist(X,Z)$.

if $Y = Y_z$ and $d + dist(X,Z) < d_z$,

$d_z = d + dist(X,Z)$.

Repeat until everyone has the same root.

→ In the case of a tie where multiple ports have equal smallest distance to the root, choose that with the smallest ID. (tie for a port)

that ↳ (neighbour's port)

→ The designated port is that with the shortest distance to the root port. If tie, break based on ID. (tie for a LAN)

→ Disable ports that are neither root port nor designated port.

The distances can be tweaked manually.

Speed	Cost
10 Gbps	2
1 Gbps	4
100 Mbps	19
10 Mbps	100