Data Link Layer

Lecture 7   Framing and introduction to CRC

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.

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: 0 1 1 1 1 1 1 0
If there is nothing to send, we continuously send this
It also helps in clock synchronization.

What if this sequence appears elsewhere?
We do bit stuffing. Say 0111 111 0 is somewhere in the middle.
　　　　　↳ we insert bits.
What HDLC does is:
　　　→ If you see 5 consecutive 1s, insert a 0.

0 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 0
　　　　　　　↓
0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 1 0 0 0 1 1 1 1 1 0 1 1 0

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?

$$11111 \begin{cases} 0 \longrightarrow \text{Remove (bit stuffing)} \\ 10 \longrightarrow \text{Assume end sequence} \\ 11 \longrightarrow \text{Assume error has occurred and discard the frame} \\ \qquad \text{(We discard everything until we see the sequence again)} \end{cases}$$

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

Cyclic Redundancy Check (CRC)

We just append the k-bit CRC to the n-bit dataword to get a
(=16 here)   "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.
⇒ We need to ensure that codewords are "far apart".

Given $v, w \in \{0,1\}^n$, the Hamming distance between v and w is
$$d(v,w) = \{i \in [n] : v_i \neq w_i\}.$$
(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 $a + b = a - b$)
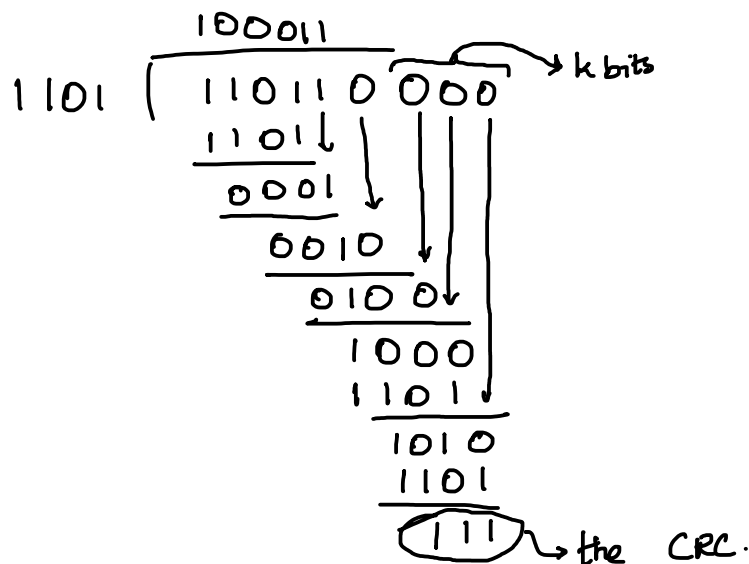Multiplication $(\times)$ has identity 1.

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 $\mathbb{F}_2$)
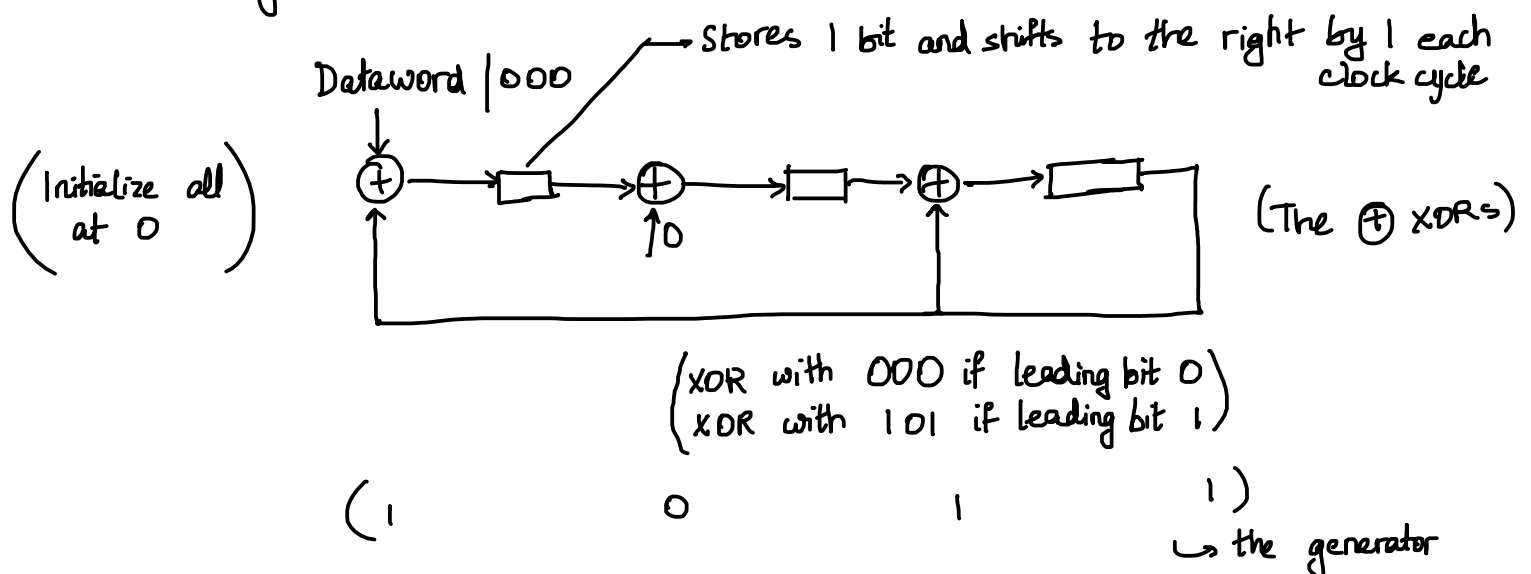
The divisor/generator is of length k+1 bits.

For example, say k=3 and the generator or "divisor" is 1101 and n=6 with dataword 110110

```
              100011
        ┌─────────────────
  1101  │ 110110 000  → k bits
          1101
          ────
          0001
           0010
            0100
             1000
             1101
             ────
              1010
              1101
              ────
               111  → the CRC.
```

The codeword is then 110110 111.
                      ‾‾‾‾‾‾ ‾‾‾
                      dataword CRC

At the sender,

Long division for a given generator can be implemented using shift registers.

For generator 1101,

→ Stores 1 bit and shifts to the right by 1 each clock cycle

Dataword | 000

(Initialize all at 0)

(The ⊕ XORs)

(XOR with 000 if leading bit 0
 XOR with 101 if leading bit 1)

(1   0   1   1)

→ the generator

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

At the receiver, either

1. pass the dataword with k Os 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 Os.

Lecture 8   CRC Polynomial Arithmetic

What bit errors can the CRC detect?

We will represent the divisor as a polynomial

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

$1101 \times 11 = 10111$

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

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

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

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

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

$\rightarrow$ Single bit errors.

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

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

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

$$E(x) = \underbrace{C(x) \cdot D(x)}$$

will have at least two non-zero powers of x.

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

→ 2-bit errors.

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

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.
⇒ no f_r will divide the x^i (if $i > 0$).
However, we could have

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

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

$$D(x) \rightarrow 0 0 0 1 \, 1 \, 1 \, 1 \, 1 \, 0 0 0$$
$$x D(x) \rightarrow \underline{0 0 1 1 1 1 1 1 0 0 0 0}$$
$$\quad\quad 0 0 1 0 0 0 0 1 0 0 0 \longmapsto \text{each string of 1s in D}$$
$$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \text{two 1s in } (1+x)D(x)$$

If $C(x)$ has an even number of terms $\nearrow x^i$, 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 $E(x)$.

$$\frac{E(x)}{c(x)} = \frac{x^i \; \overbrace{(x^{\ell-1} + x^{\ell-2} + \cdots + 1)}}{\underbrace{x^k + \cdots + 1}_{\text{no } x^0 \text{ factor}}}$$ → this is the part that matters

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

—

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 reliably? ↙ 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.
　　$\left(\begin{array}{l}\text{first bit of sent frame from sender to} \\ \text{last bit of ACK frame to sender}\end{array}\right)$
　　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 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).
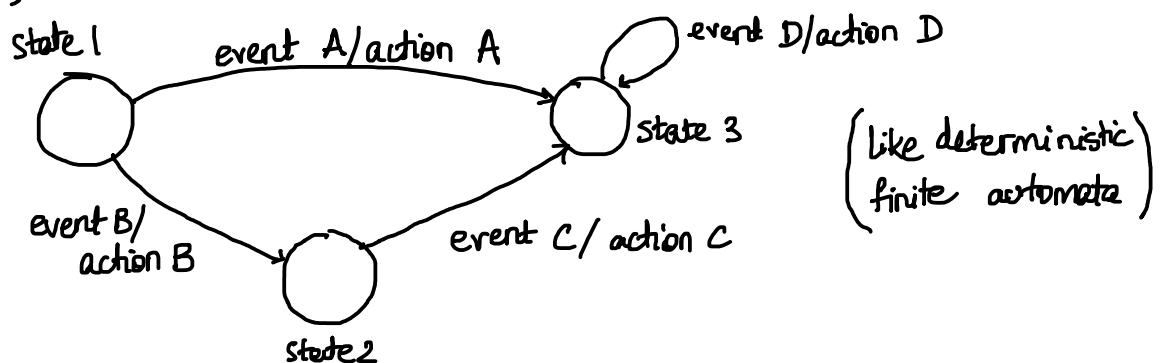
It is also possible to run into problems if we set the timer to be too small. We receive the ACK for a previous version of the (same) frame while we are waiting for the current frame.
If the timer is too large, then we waste a lot of time if there is an issue with the frame (or ACK).
We must set the timer appropriately to cut down on resources and redundancy.

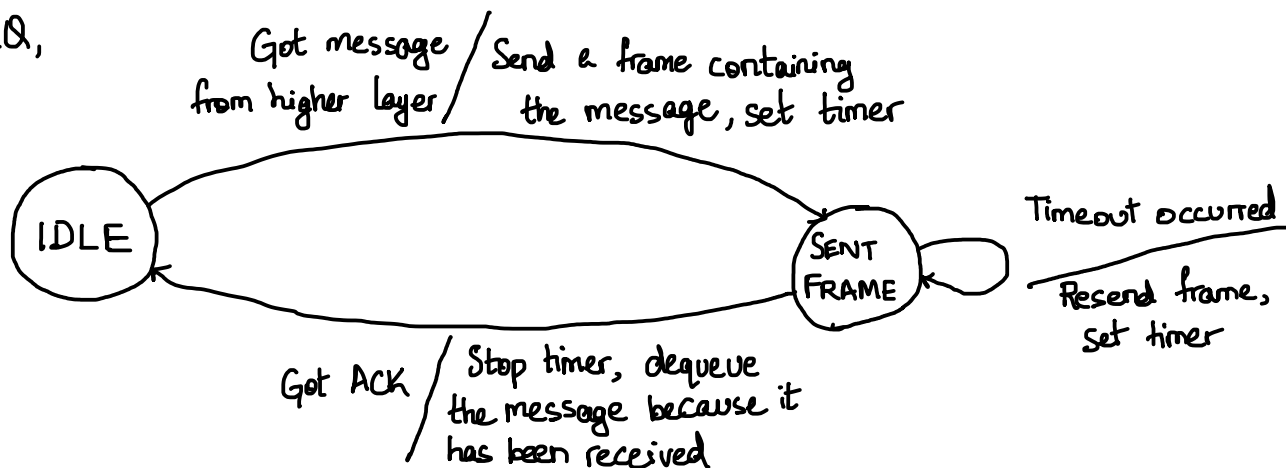We often represent protocols using state diagrams or flowcharts.
For example,

Each action is represented by a flowchart.

In ARQ,

Got message from higher layer / Send a frame containing the message, set timer

IDLE

SENT FRAME

Got ACK / Stop timer, dequeue the message because it has been received

Timeout occurred / Resend frame, set timer

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.

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?

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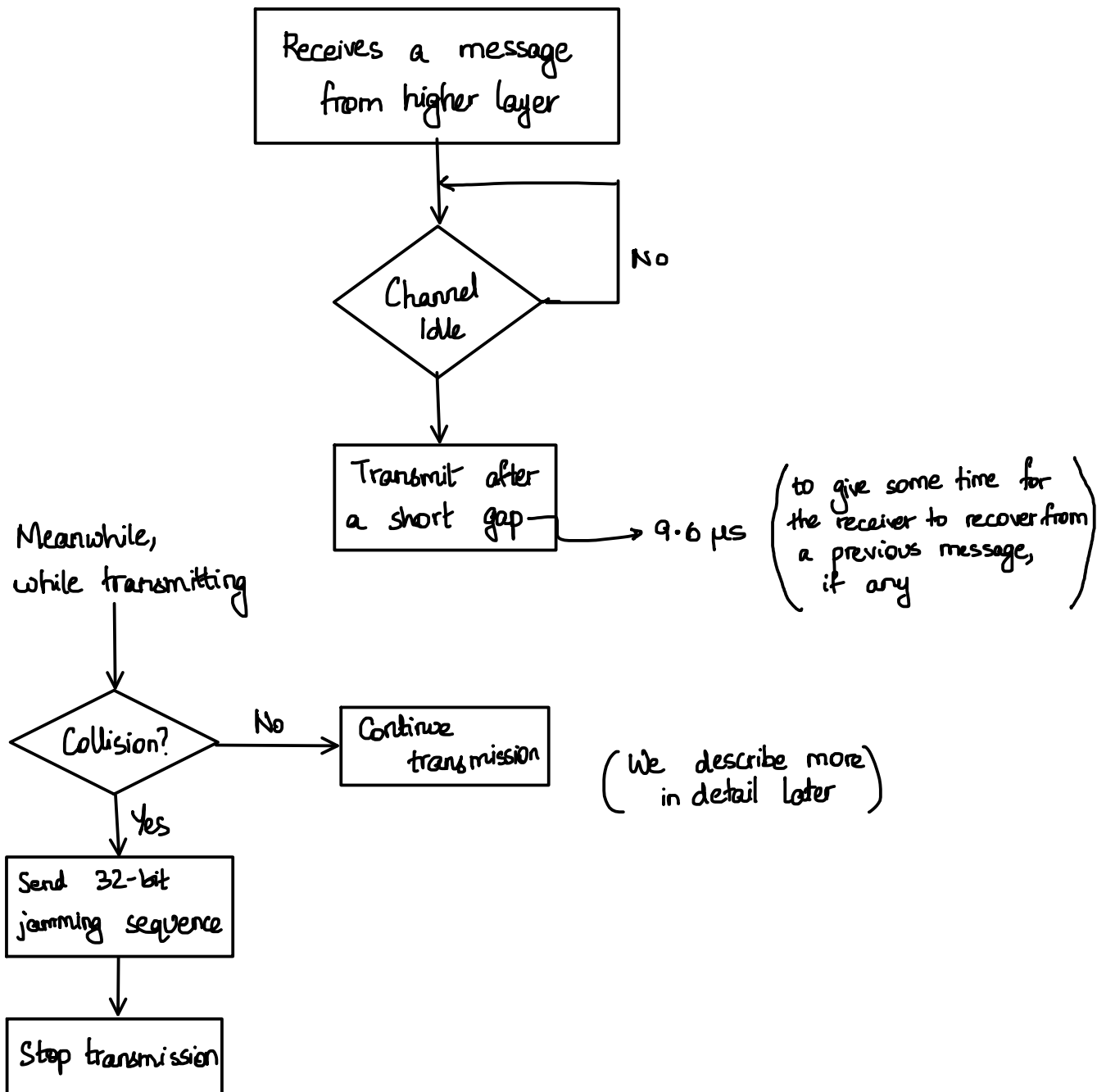
Keep another threshold. If the energy is greater, then declare that a collision has occurred.

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.

```
┌─────────────────────────┐
│   Receives a message    │
│   from higher layer     │
└─────────────────────────┘
            │
            ▼  ◄────────────┐
         ╱────────╲         │ No
        ╱ Channel  ╲────────┘
        ╲  Idle    ╱
         ╲────────╱
            │
            ▼
┌─────────────────────┐
│  Transmit after     │
│  a short gap  ──────┼──────►  9.6 μs
└─────────────────────┘
```

(to give some time for the receiver to recover from a previous message, if any)

Meanwhile, while transmitting

```
         ╱──────────╲      No    ┌──────────────┐
        ╱ Collision? ╲──────────►│  Continue    │
        ╲            ╱           │  transmission│
         ╲──────────╱            └──────────────┘
            │ Yes
            ▼
┌──────────────────┐
│  Send 32-bit     │
│  jamming sequence│
└──────────────────┘
            │
            ▼
┌────────────────────┐
│ Stop transmission  │
└────────────────────┘
```

(We describe more in detail later)

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

$\longleftrightarrow$ 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.

We want frame to be at least this long.
) 25 μs.

first bit

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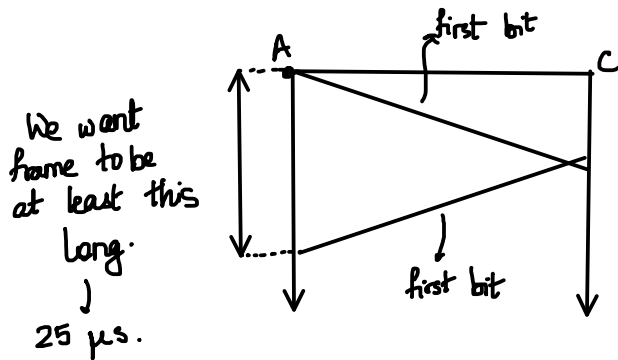
Along the way, we also have repeaters that reproduce the signal. This may add some more time, so let us conservatively take 50 μs (instead of 25 μs).

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 $1 - (1-p)^{n \cdot \text{length}}$

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 $\{\Delta v : v \in \{1, \cdots, 2^k\}\}$. We want k to be larger if more people have collided. How?

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 backoff.

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.

Got frame to send

No. of collisions for this frame ← N = 0

Carrier Sense → Busy

Idle

Wait 9.6μs then start transmitting.

Collision? —Yes→ Send 32 bit jamming sequence → ++N → N<15

No

Done!

N<15 —Yes→ N≤10

(Don't want to try forever)

N<15 —No→ Abort transmission

N≤10 —Yes→ k=N

(Don't want to wait for too long)

N≤10 —No→ k=10

k=N → Uniformly randomly select $U \in \{1, 2, \cdots, 2^k\}$

k=10 → Uniformly randomly select $U \in \{1, 2, \cdots, 2^k\}$

Uniformly randomly select $U \in \{1, 2, \cdots, 2^k\}$ → Wait for $U \times 51.2$ μs