Tutorial 4 - CSMA/CD

Gidon Rosalki

2025-11-20

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

1 CSMA / Collision Detection

1.1 Introduction

We saw 2 variations of the ALOHA protocol, slotted which is (magically) synchronised, which had the better goodput of around 0.37, and pure, which is not synchronised, and had a goodput of 0.18. They have poor performance since they do not deal well with collisions. They can be improved by having a central manager, but in this course we are focusing on distributed protocols.

To improve these protocols, we can add some features, such as a node checking if it can transmit a message or if this would case a collision. If a node observes another transmission at the same time, then it can react. Consider a conversation, you do not just talk with probability p, but rather listen first, and speak if no one else is. We would like to add these two features.

When more than one node is using the same broadcast channel, then collisions are bound to occur, so to avoid these we can potentially check if the channel is free, schedule time for each node, or even add more channels (not in this course). For now, let us focus on checking the availability of the channel.

1.2 Carrier Sense Multiple Access (CSMA)

Let us suppose we simply test if the channel is occupied before we transmit. This is insufficient, since transmissions upon it are **not** instantaneous. There is a propagation time, so node A could transmit, and at the same time node B could test, and observe that the channel is unoccupied. Since it is unoccupied (from B's perspective), B begins to transmit, causing a collision with A's transmission.

Let us define T_{prop} to be the maximum propagation time in the channel, between any two nodes. A transmission may collide with another within the first T_{prop} seconds of it beginning to transmit. A node should decide what to do if the channel is occupied, and we iwll discuss the following 3 approaches:

Approach	What to do if free	What to do if occupied	Intuition
1-persistent	Send	Wait until the channel becomes idle, and then send immediately	High chance of collisions in loaded channels
Non-persistent	Send	Backoff and try again	Lower utilisation on low load
p-persistent	Send with probability p	Wait until the channel becomes idle, and then send with probability p	A compromise between the other two approaches

Table 1: Approaches

When detecting a collision, we want **all** the nodes to know about it, but then we have a few problems:

- How long does it take to detect a collision?
- How can we notify? Nodes A and B know that there was another transmission when they were transmitting, but node C just saw information on the channel, and does not know that this was not a legitimate transmission
- When should a node try and retransmit?

1.2.1 How long

If the network propagation time is T_{prop}, how long does it take to notice a collision?

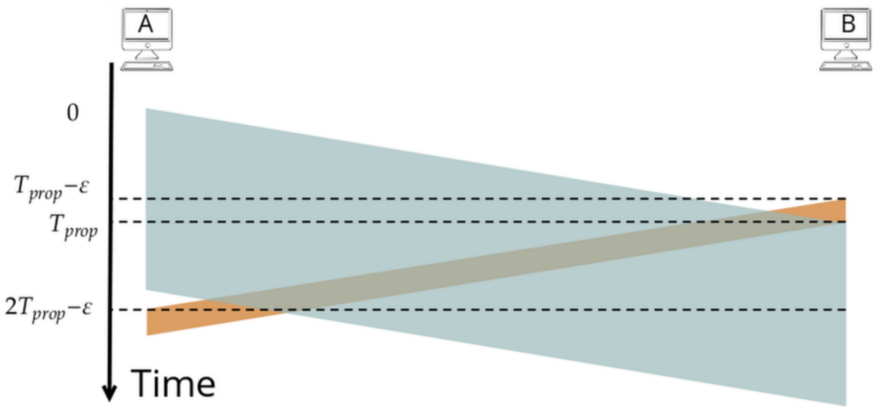

Figure 1: Collision time

Let us assume that the maximum one way propagation time of the network is between nodes A and B, and is T_{prop}. Node A begins transmitting at time 0, and B begins at $T_{prop} - \varepsilon$. Node A will only see the collision at time $2T_{prop} - \varepsilon$, so the amount of time it takes to notice a collision is $2T_{prop}$.
In conclusion, we need the frames of our CSMA/CD network to be long enough such that

$$T_{transmission} > 2T_{prop}$$

1.2.2 Jam signal

Other nodes listening will hear the collided signals, but not know if that was an intended transmission. They are notified that a collision occurred by the node that detected said collision sending a jam signal, after which all the nodes will be aware of the collision, and will now ignore the previous (corrupted) data.

1.2.3 1-persistent

In the case of collisions, we want to wait before trying again. There are a few ways to decide how long to wait, but the standard method is called "exponential backoff".
Exponential backoff depends on a parameter c, which we usually set to 2. After the kth attempt to transmit a frame, we will uniformly select j from $\{0, 1, \ldots, c^k - 1\}$, and wait $j \cdot T$ time. When a node transmits a frame successfully, then it resets its number of attempts counter (k) to 0. The attempt counters of other nodes remain unaffected.

1.2.4 p-persistent

Here, we have **no** exponential backoff. In the case of a collision, then the entire frame is wasted. The nodes instead simply wait until the channel is idle, and then send with probability p.

1.3 CSMA/CD p-persistent

1.3.1 Definitions

For networking, let:

- End to end propagation time (E2EP) - T_{prop}
- Number of stations - N
- Frame transmission time - T_{trans}

For the protocol, let:

- Time is divided into slots of size $2T_{prop}$ time units
- Nodes always have data to transmit
- The probability of a node transmitting is p

1.3.2 Analysis

So, our p-persistent protocol comes down to:

1. Sense: If busy, wait another time slot, and if free, send with probability p. Once sent, start the detection phase
2. Detecting: If detected a collision, stop transmitting, and try sending again in the next slot (with probability p)

The jam signal is still sent, and relevant, just not so interesting, and not mention here.

So the network may be in the following states:

- Idle - No node is using the network
- Transmitting - Some node succeeded in sending data, without a collision
- Contention - 2 nodes (or more) started sending at a difference of less that T_{prop}, and nodes will know of this only after at most $2T_{prop}$ time units have passed

Let us begin from the goodput of p-persistent: The probabilithy of a successful transmission is the same as in slotted ALOHA:

$$P_{suc}\left(p\right) = P\left(X_p = 1\right) = np\left(1-p\right)^{n-1}$$

The maximum of this function is at $p = \frac{1}{n}$, and so

$$\max_{p\in[0,1]} \left\{P_{suc}\left(p\right)\right\} = \left(1-\frac{1}{n}\right)^{n-1} = S$$

This is insufficient for calculating the goodput, since we need to account for the lost time of contention, and idle slots. Since each slot is of size $2T_{prop}$, the result is different from "regular" slotted ALOHA. This is like asking for the probability of k idle / contended intervals, followed by a successful transmission. This is a geometric distribution, with a parameter S. If the contended intervals is a geometric random variable, with parameter S, then its expectation is $\frac{1}{S}$, and so there are $\frac{1}{S}-1$ wasted slots. Each wasted slot costs $2T_{prop}$ time, so the expected wasted time is

$$2T_{prop}\left(\frac{1}{S}-1\right)$$

We can then calculate the goodput as

$$\frac{\text{Transmission time}}{\text{Transmission time + Wasted time}} = \frac{T_{trans}}{T_{trans} + 2T_{prop}\left(\frac{1}{S}-1\right)}$$

So as $n \to \infty$, the maximal goodput is

$$\frac{T_{trans}}{T_{trans} + 2T_{prop}\left(\frac{1}{S}-1\right)} = \frac{1}{1+\frac{2T_{prop}}{T_{trans}}}\left(e-1\right)$$

If our goodput is given as

$$\frac{1}{1+\frac{2T_{prop}}{T_{trans}}}\left(e-1\right)$$

Then the goodput will tend to 1 when either the transmission time tends to infinity, meaning that after the risky T_{prop}, no nodes will transmit, or when the propagation time tends to 0, meaning that the risky T_{prop} time will be insignificant.

Whereas in slotted ALOHA, without CSMA/CD, we calculated

$$S_{max} = \frac{1}{e}$$

2 Questions

Consider a CSMA/CD network with the following parameters:

- Link bandwidth: $10Mbps$
- Link propagation speed: $2\cdot 10^8 m/s$

We want to add a similar network, distanced $20Km$ from this one, connected by a medium of same properties The distance between two nodes within the same network is $<< 20Km$.

2.1 Question 1

Can a message of size 64Bytes be used in this type of network?

The time it takes for a single bit to cross between the two networks is

$$\begin{aligned} T_{prop} &= \frac{\text{distance}}{\text{propagation speed}} \\ &= \frac{20Km}{2\cdot 10^8 m/s} \\ &= 10^{-4}s \\ &= 100\mu s \end{aligned}$$

Where the time it takes to transmit 64B is

$$\begin{aligned} T_{trans} &= \frac{\text{bits}}{\text{bits per second}} \\ &= \frac{8 \cdot 64}{10Mb/s} \\ &= \frac{8 \cdot 64}{10^7} \\ &= 51.2\mu s \end{aligned}$$

Since $T_{trans} < 2T_{prop}$, we cannot use such a message in this network, and implement CSMA/CD. A station could finish sending a $64B$ frame before a worst-case collision has time to propagate back and be detected.

2.2 Question 2

What is the smallest change we need to make to solve the above problem?

We need to make the packet size large enough so that the time it takes to send is larger than $2T_{prop} = 200\mu s$. If we use $256B$ messages (4 times larger), this would give us a transmission time of $204.8\mu s$, which satisfies this.

2.3 Question 3

What would change if the link bandwidth was $100Mbs$?

Increasing the link bandwidth allows us to transmit at a higher rate, so T_{trans} is a factor of 10 smaller. We need to increase the packet size for the $T_{trans} \geq 2T_{prop}$ to hold, so since we have increased the link rate by 10, we would need a packet size of at least $2560B$.

2.4 Question 4

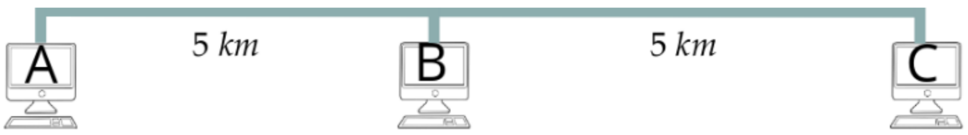

Figure 2: Network architecture

Let the propagation speed be

$$v_{prop} = 6 \cdot 10^7 m/s \tag{1}$$
$$B = 3Mb/s \tag{2}$$

For the CSMA/CD protocol to work properly, what is the minimum message size (x) that A can send C?

$$\begin{aligned} T_{prop} &= \frac{\text{distance}}{v_{prop}} \\ &= \frac{10Km}{6 \cdot 10^7 m/s} \\ &= \frac{1}{6} \cdot 10^{-3} s \\ T_{trans} &= \frac{x}{B} \\ &\geq 2T_{prop} \\ \Leftrightarrow \frac{x}{3 \cdot 10^6} &\geq \frac{1}{3} \cdot 10^{-3} \\ \Leftrightarrow x &\geq 10^3 bit \end{aligned}$$

For the CSMA/CD protocol to work properly, what is the minimum message size (x) that A can send B?

We must take into account the worst case scenario, so as above:

$$x = 10^3 bit$$

2.5 Question 5

Consider a network that runs **slotted ALOHA**, with $T_{trans} > 2T_{prop}$. Each slot is of size T_{trans}. Will adding **CSMA** improve the goodput? No, since if nodes sense the channel at the *beginning* of each slot, they will always sense that the medium is free, and will then transmit.

Consider a network that runs **slotted ALOHA**, with $T_{trans} > 2T_{prop}$. Each slot is of size T_{trans}. Will adding **CSMA/CD** improve the goodput? No, since even if a collision is detected, then the entire slot is already wasted.

Consider a network that runs **pure ALOHA**, with $T_{trans} > 2T_{prop}$. Each slot is of size T_{trans}. Will adding **CSMA** improve the goodput? Yes, this is a trivial CSMA improvement.

Consider a network that runs **pure ALOHA**, with $T_{trans} > 2T_{prop}$. Each slot is of size T_{trans}. Will adding **CSMA/CD** improve the goodput? Yes, since if collided transmissions will stop earlier, this will free the medium for other transmissions.

2.6 Conclusions of CSMA/CD

In CSMA/CD, we wait for the channel to become idle before sending, detect collisions and notify everyone of them when they happen (the jam signal), and wait a random time period before retrying (backoff). It makes sense to use CSMA/CD when the E2EP time is significantly shorter than the transmission time.

3 Errors Detection

When sending information, there is every chance (cosmic rays, noisy channels, etc.) that same information will not arrive in the end, some bits may be changed. We need to find a way to **detect** this corruption, and **recover** from them if possible.

Consider for example the Israeli ID number. The final digit is a checksum, computed by the Luhn algorithm:

1. Firstly, it alternates multiplying digits by 1 or 2
2. If a digit is multiplied by 2, and the result is greater than 9, then the digits of the doubled number are summed (e.g. $2 \cdot 7 = 14 \to 1 + 4 = 5$)
3. All the resulting digits (from the original digits, or their doubles) are then added together
4. The checksum digit is calculated as the difference between 10, and the last digit of the sum (sum modulo 10).

This is equivalent to the following Python:

```python
def luhn(tz_without_last_digit: int) -> int:
    digits = str(tz_without_last_digit)
    total = 0
    for index, string_digit in enumerate(digits):
        # Alternate multiplying by 1 and 2
        mult = 1 if index % 2 == 0 else 2
        digit = mult * int(string_digit)

        # Sum digits if result of multiplication is more than 1 digit long
        if digit > 9:
            digit = sum([int(sub_digit) for sub_digit in str(digit)])

        total += digit

    # Generate checksum from total
    last_digit = total % 10
    checksum = 10 - last_digit

    return checksum
```

So one can very quickly check if an entered value fits this formula, and verify that it is a legitimate ID number.

Similarly, to identify if some error occurred to our packet, since it is just a collection of bits, we can add additional, redundant data / bits that would allow us to observe changes in the original data. This would naturally have some cost, since more data is now being sent on the network.

3.1 Repetition code

Before we start with clever methods, consider the basic one of just repeating the packet. If one triples the packet, this would increase the packet size from n to $3n$, but for each bit, one may consider which bit occurs the most. If one suffered some corruption, and a bit was swapped, but the other 2 did not for this bit, it may be seen that two thirds agree that it is the other bit, and choose that bit. Essentially, each bit is chosen by majority vote of the repetitions. However, this comes with significant overhead:

$$\begin{aligned}\frac{\text{Actual - Original}}{\text{Original}} &= \frac{\text{Actual}}{\text{Original}} - 1 \\ &= \frac{\text{\# Redundant bits}}{\text{\# Original bits}}\end{aligned}$$

So repeating 3 times has the overhead of

$$\frac{3N}{N} - 1 = 2$$

3.2 1D parity

To reduce the additional of $2n$ bits, there are things to consider. If we were guaranteed that there could be at most a single bit being corrupted, then we can identify it through bit arithmetic. We can add a single bit indicating if there are an odd, or even number of 1s in the packet. That way, we can identify if there is an error, but not correct it. This has the overhead of

$$\frac{N+1}{N} - 1 = \frac{1}{N}$$

The cost of bit parity is a single bit per packet, which is not very much, but it cannot recover from failure. Additionally, the assumption that only one bit can change does not hold in practice. 2 bits changing can affect the corruption detection, so to overcome this we use 2D parity.

3.3 2D parity

Here, we convert the data into a rectangular form of a specific length. Parity bits are then added for both the rows, and the columns, for example, if there is a length of j, and data of size $i \cdot j$ bits, then the actual length of the packet is $i \cdot j + i + j + 1$ bits.

$$\begin{bmatrix} d_{1,1} & \dots & d_{1,j} \\ d_{2,1} & \dots & d_{2,j} \\ \vdots & \dots & \vdots \\ d_{i,1} & \dots & d_{i,j} \end{bmatrix} \implies \left[\begin{array}{ccc|c} d_{1,1} & \dots & d_{1,j} & d_{1,j+1} \\ d_{2,1} & \dots & d_{2,j} & d_{1,j+1} \\ \vdots & \dots & \dots & \vdots \\ d_{i,1} & \dots & d_{i,j} & d_{i,j+1} \\ \hline d_{i+1,1} & \dots & d_{i+1,j} & d_{i+1,j+1} \end{array}\right]$$

This way, we may identify errors from both the rows, and the columns, and thus correct them (with limitations). This approach can **detect** 1 bit errors, 2 bit, and 3 bit errors. Some 4 bit errors may be corrected, but others may not. For example, when a 4 bit 2x2 block is all swapped, this will not affect the overall sums in the rows and columns, and is thus a silent failure.

We may use this approach to recover lost data, and may recover a maximum of $\frac{\text{dimension}}{2}$ bits, so in this case: 1. What would be the overhead of 2D parity for a 100bit packet with $j, i = 10$?

$$\begin{aligned}\frac{i \cdot j + i + j + 1}{i \cdot j} - 1 &= \frac{100 + 10 + 10 + 1}{100} - 1 \\ &= \frac{121}{100} - 1 \\ &= 0.21\end{aligned}$$

So in summary:

Code	Overhead	Ability to correct	Ability to detect
Repetition (3 times)	2	1	2
1D Parity	$\frac{1}{N}$	0	1
2D parity	$\frac{i+j+1}{N}$	1	3

Table 2: Error detection and recovery summary

3.4 Cyclic redundancy checks (CRC)

Cyclic redundancy checks are commonly used, and types are referred to as crc-r, for example, Ethernet uses crc-32. This adds r bits to the overhead, and only fails to detect at probability $\approx 2^{-r}$. We treat the packet bits as a long binary number (or a polynomial over $GF(2)$), and divide it by a fixed generator polynomial. The remainder of this division is the CRC value, which will be added to the packet. It is used as follows:

1. The sender computes the CRC, and appends it to the frame
2. The receiver divides the received bits (data + CRC) bu the same polynomial
3. If the remainder is 0, we assume no error; otherwise, we detected corruption

Consider the following example of crc-3: We choose a simple generator, of degree 3:

$$G(x) = x^3 + x + 1 \Leftrightarrow G = 1011$$

We have the data to send $D = 1101$. We append $r = 3$ zero bits (the degree of G):

$$D \cdot x^3 = 1101000$$

and divide 1101000 by 1011 by using binary (XOR) long division, getting aa remainder $R = 001$. The sender appends the CRC:

$$\text{coded message} = D \| R = 1101001$$

The receiver divides 1101001 by 1011, and gets a remainder of 000, meaning no error was detected. If there was a bit flip, then division by 1011 results in a non-zero remainder, and an error is detected.