ECE 358: Computer Networks

Lab 2

CSMA/CD Performance Evaluation

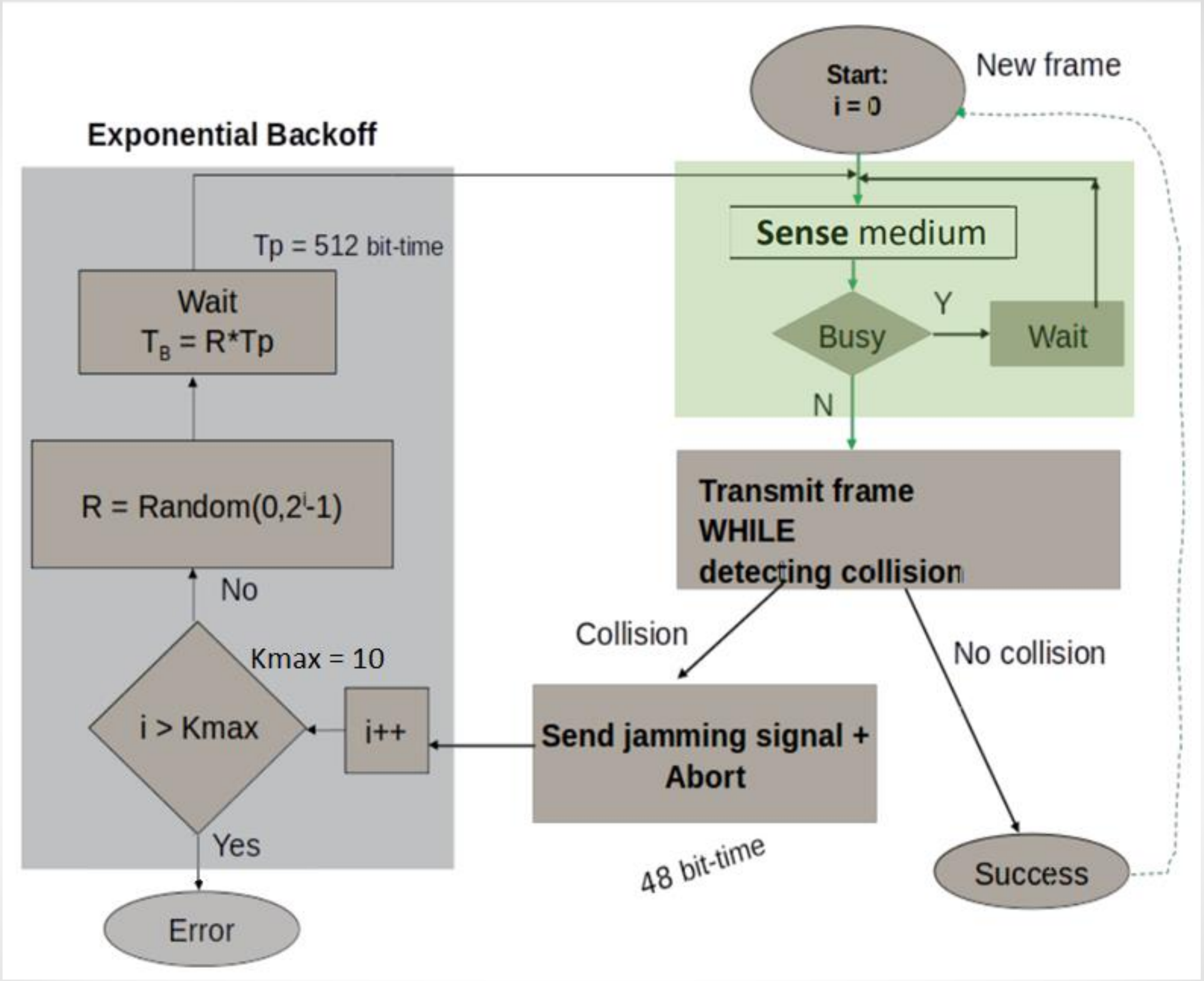
Presented by Hamidreza Nafissi
Spring 2020

Goals

- In this lab, you will once again be implementing a Discrete Event Simulator (similar to what you did in Lab 1)
- You will be simulating the behavior of a CSMA/CD LAN
- You will be monitoring the performance of the LAN

How does CSMA/CD work?

- Listen to the network, and wait for it to be idle
- Transmit a frame, and while transmitting, check for collisions
- If no collisions, the frame was sent successfully
- If there were collisions, back off for a bit and then try again


Exponential Backoff
Start: i = 0
New frame
Sense medium
Busy
Y
Wait
N
Tp = 512 bit-time
Wait $T_B = R*Tp$
$R = Random(0,2^i-1)$
No
Kmax = 10
i > Kmax
i++
Yes
Error
Transmit frame WHILE detecting collision
Collision
No collision
Send jamming signal + Abort
48 bit-time
Success

Exponential Backoff

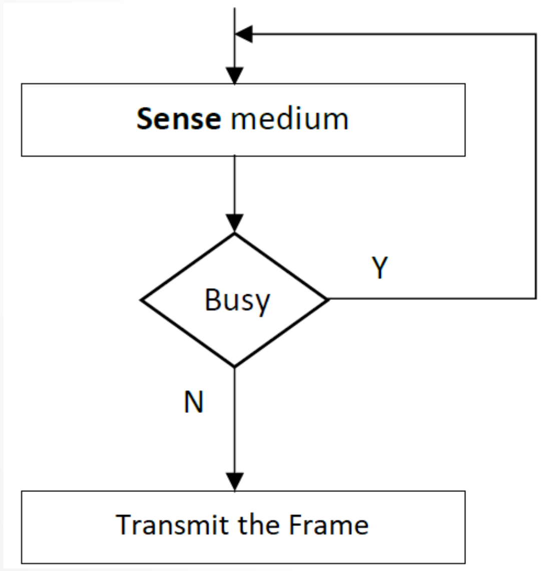
- If everyone on the LAN tries to transmit as soon as the medium is available, you'll get a lot of collisions
- Instead, we have the nodes wait a random amount of time before retransmitting
- The random amount of time increases exponentially, so when network isn't busy you don't wait long
- Specifically, wait *Random(0,* $2^i - 1$*)* 512* bit-times where *i* is the number of times you've retried
- Retry up to K_{max} times, and discard the frame if you go past that (K_{max} == *10* in this lab)

Sensing the Medium

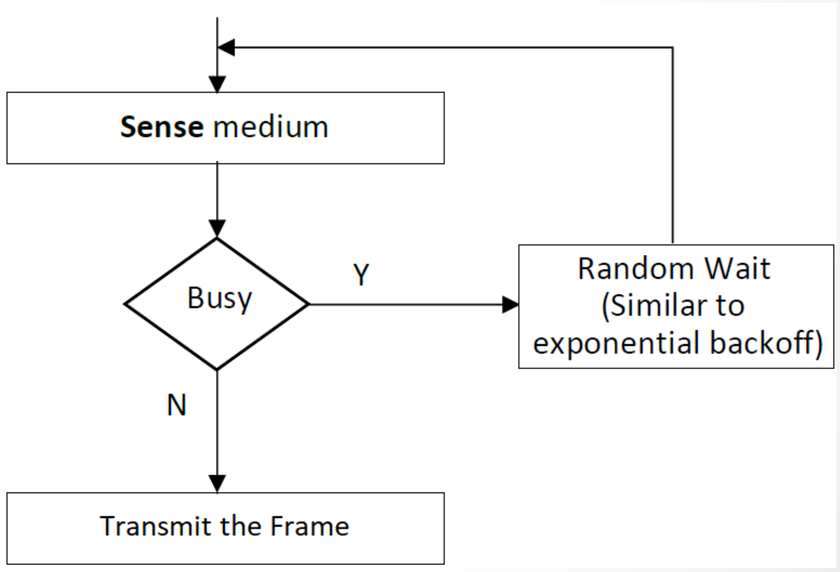
- There are two alternatives when it comes to sensing whether the medium is busy
- Persistent
  - If the channel is busy, continue sensing until it becomes idle
- Non-persistent
  - If the channel is busy, wait a while before checking again
  - By "a while" we mean a random period of time
  - For this lab, use the same algorithm as for exponential backoff (previous slide)
- Persistent mode is "greedy", non-persistent is friendlier

Sensing the Medium

Persistent



Non-Persistent



Simplifying Assumptions

- Real-world implementations can be quite complex
- To make it easier, you can assume the following:
    - There is no need to simulate the "jamming signal" shown in the figure
    - All nodes see a collision at the same time (aside from propagation delay)
    - The distance between consecutive nodes on the network is the same
    - All the packets are the same length

What you need to do

- You need to create an application in C, C++, Python or Java that simulates a CSMA/CD-based network
- We provide *R*, the speed of the network in bits/second, *L* the length of the packet, *D*, the distance between nodes and *S*, the propagation speed (these are all constant throughout your simulations)
- You will simulate both a 1-persistent and a non-persistent CSMA-CD protocol
- For each, you will **vary the number of nodes *N* and the average packet arrival rate** (using the same Poisson process as for Lab 1), **compute the efficiency and throughput, and plot the results**

How to do it

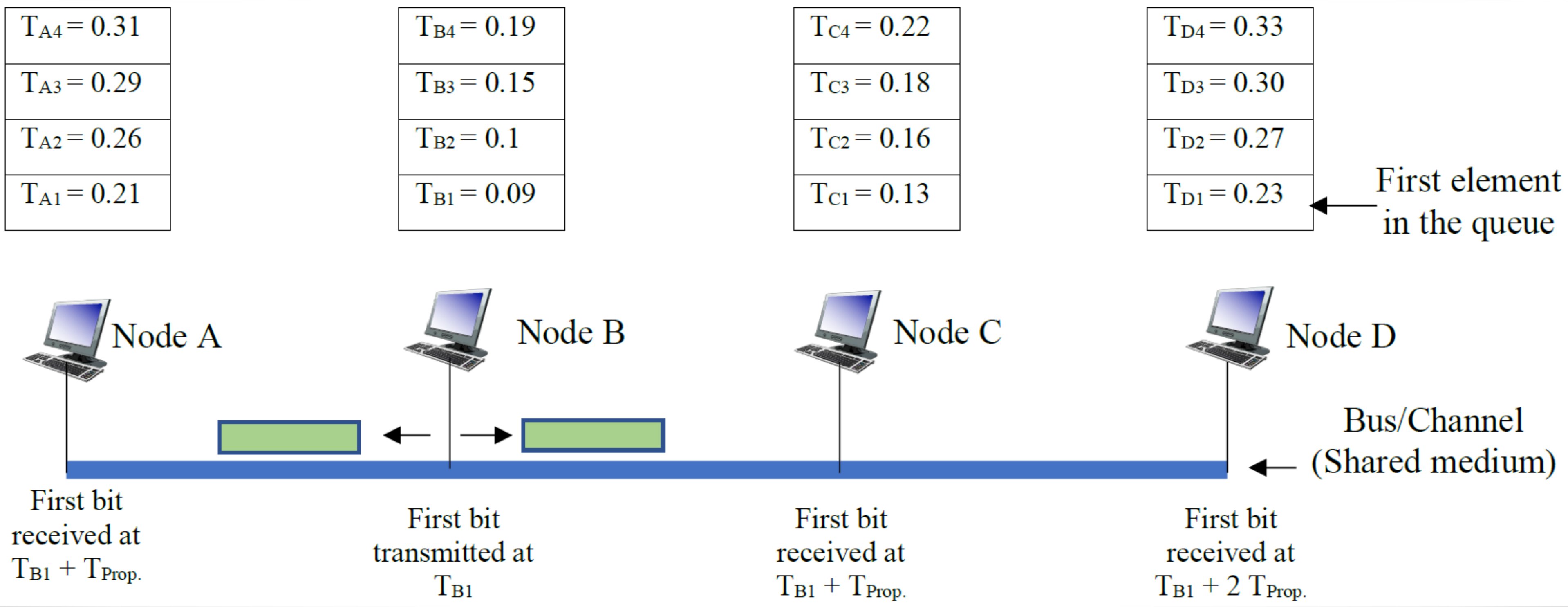
- You can implement this lab however you like
- What follows is a description of one possible approach
- If you choose a different approach, that's fine, but be sure to *document* the approach you're using, how it works and why you chose it.

Setup and transmission

- Setup
  - Create a queue of packets for each node
  - Each packet has an arrival timestamp
  - Populate the queues, generating the timestamps the same way you did for Lab 1
  - Select a simulation time T_{sim} (the same way you did for Lab 1)

- Sending a packet
  - Determine which node transmits next (the one whose queue head has the lowest timestamp)
  - Set the current simulation time to the timestamp of that packet
  - In other words, the time of the next discrete event in the simulator is the time of the next packet (no matter which queue it's in)
  - The packet is transmitted at that time (i.e. the time that the first bit of the packet is sent)

How to do it: an example
Persistent Mode



Propagation Delay

- The first bit of the packet arrives at the immediately adjacent nodes after some fixed propagation time T_{prop}, and later than that at more distant nodes (always some multiple of T_{prop})
- You can also compute $T_{transmission}$ which is the length of time the bits of the packet are being transmitted, based on the (fixed) length of a packet L and the (fixed) transmission rate R

Detecting Bus Busy

- When node B sends a packet at time T_{B1}, the bus is busy (at node C) between $T_{B1} + T_{prop}$ and $T_{B1} + T_{prop} + T_{transmission}$
- If the arrival time of the first packet in the queue at C is greater than $T_{B1} + T_{prop}$ and less than $T_{B1} + T_{prop} + T_{transmission}$, C will see the bus as busy and will wait
- "Waiting" means updating the arrival times of C's queued packets so that any packets that were due to be sent before the end of B's packet transmission are rescheduled to the end of the packet (i.e. to $T_{B1} + T_{prop} + T_{transmission}$)

Detecting Collisions

- Case I: $T_{C1} < T_{B1} + T_{prop}$, then C does not know that there is a packet on the bus already and will therefore start transmitting => **collision**
- Case II: $T_{B1} + T_{prop} < T_{C1} < T_{B1} + T_{prop} + T_{transmission}$, a collision has occurred and you need to **use exponential backoff**
- Reschedule the packets in the queue at C as discussed on the previous slide, setting their time to the end of the waiting period

Exponential Backoff

- When you transmit a packet for the first time, you reset a counter to zero
- Every time you try to transmit and encounter a collision, you increment the collision counter
- If the counter reaches ten, the packet is discarded
- Each time you increment the counter (call it *iteration*) you generate a random waiting period between zero and $(2^{iteration}-1) * 512$ *bit-times* (the time it takes to transmit a bit, based on R)

Non-Persistent Mode

- The non-persistent case is exactly the same as the persistent case we just described, except that instead of setting the waiting time to be the end of the packet from B it is set to a random value (same way you do the backoff, so **you need a separate iteration counter**)

Efficiency and Throughput

- The **efficiency (η)** is the number of successfully transmitted packets divided by the total number of packets pulled from the queues
- The **throughput** (in *Mbps*) can be computed using the number of successfully transmitted packets, the packet length L, and the simulation time T_{sim}

What You Need to Find

- You are given fixed values for the packet length *L*, the transmission rate *R*, the distance between adjacent nodes *D* and the propagation speed *S*
- You will run the simulation multiple times, using three different values for the packet arrival rate *A* (7, 10 and 20) and varying the number of nodes *N* from 20 to 100 inclusive in steps of 20
- You are expected to compute the efficiency and throughput for both the persistent and non-persistent modes
- You are expected to graph the **efficiency (η) vs number of nodes (*N*)** for each of the three arrival rates

Marking Criteria

- You will be marked based on the correctness of your results (e.g. shape of the curves)
- You will also be marked based on your written report and the description you provide of how your simulation works
- The written report is especially important if you choose an approach to simulation that differs from the one described here

Notes

- Programming languages: you can use C, C++, Python, Java or other programming language that is available on eceUbuntu
- Your code must compile and run on the eceUbuntu servers since that's where the TAs will be testing it
- You must submit your code and your report on Learn individually
- You should provide a single command that builds and runs your application (**makefile**) and describe (in your report) how to use it (**readme.txt**).

Questions?