Challenge 3: Implementing a medium access control protocol

1 Challenge

Your task is to design and implement a medicum access control protocol to share a time-slotted medium fairly and efficiently among four nodes.

Periodically, a timeslot will be announced in which each node can decide whether or not to transmit. Only if exactly one node transmits (out of four nodes sharing the channel), the transmission will be successful; otherwise, the slot is wasted (collision or idle). Each successful transmission may also contain some control information, which the nodes can use to optimize their sharing of the channel.

Your task is to design and implement a medium-access mechanism for this system; i.e., an algorithm that for each node and each timeslot decides whether or not to transmit. The goal is to optimize this for both *efficiency* (i.e., not wasting time slots) and *fairness* (nodes shouldn't be able to hog the channel causing other nodes to have to wait long).

2 Details

Periodically, the physical layer will announce a new timeslot to your protocol. With every announcement, also the status of the previous timeslot (success, idle, collision) and, if succesful, the control information contained in the slot will be given. Furthermore, the number of packets waiting in the queue for transmission is announced. Packets arrive to the queue at random times, and one packet leaves the queue if a node transmits succesfully. Your algorithm then, at each timeslot, has to decide whether to transmit, and what control information to send with the transmission.

Practically, this is implemented by writing a method (Java) or function (C++) which gets called (by an underlying framework) once per timeslot. This method/function should return one of the following:

- if it has data in the queue and wants to transmit: `(TransmissionType.Data, c)`, where `c` is a Java or C++ integer containg 32 bits of control information
- if it has no data in the queue and but wants to transmit anyway, only for control information: `(TransmissionType.Data, c)`
- if it decides not to transmit: `(TransmissionType.Silent, 0)`

If a data transmission was successful, the number of packets in the queue will be decremented by 1. Note that your code does not handle the data itself (as with last week's reliable data transfer challenge); your code only needs to decide whether and what to transmit.

In order to run your protocol, you will need to compile and run the class `protocol.Program`, which will connect to our server. The challenge server emulates the behavior of the medium, and pushes packets onto your queue. The rate at which the server generates packets at nodes will differ between nodes, and vary over time. The server will only generate packets for a limited time period. After that period, your protocol must continue to transmit until the queues are empty. The server keeps track of what happened, both by calculating efficiency and fairness scores, and by giving you a trace showing exactly what happened in each timeslot, to help you to improve your algorithm or debug the implementation.

The protocol will eventually have to run on a network of 4 nodes, but you are free to experiment with other network sizes (up to 6 nodes, to prevent excessive server load).

3 Hints for getting started

You are supplied with a Java or C++ framework which **must not** be modified, with the exception of the configuration parameters in `Program.java`. Please change the group ID to one of your student numbers, and change the default password.

A basic protocol, which is provided as a starting point, is a simplified version of slotted Aloha. The class `protocol.SlottedAlohaSimplified` provides you with example code, which shows you how to interact with the server emulating the medium and packet arrivals. Start with compiling and running 4 instances of this protocol. At the webpage `http://netsys.student.utwente.nl:8000/mac/`, you will then find a trace of what happened on the medium during this run. Try to see what caused the poor performance of the protocol.

Next, think about how to improve this, by creating your own implementation of `protocol.IMACProtocol`. Try to think of better rules for accessing the medium, and possibly exchanging some control information. Keep in mind that control information will also be lost if the slot in which it was transmitted resulted in a collision.

4 C++ getting started

From challenge 2 you should have Cygwin installed with gdb and gcc-g++. Now you can simply open `.cproject` with a text editor, such as Notepad++, and change the paths on lines 26 - 31. On these lines you need to change 'C:\cygwin64' to the folder in which you have installed Cygwin and 'x86_64-pc-cygwin\4.9.3' to what is available on your system. After you have changed `.cproject` you can simple start Eclipse and choose File -> Import -> General -> Existing Projects into Workspace, and then select the folder which contains `.cproject`.

5 Submission and grading

Your group's scores will be automatically maintained by the server. To finalize your challenge, you will also have to hand in a 1 page design document, in which you describe the operation of your protocol. Your grade will be determined by the performance of your protocol implementation, as well as the quality of the report on your design choices.

6 Learning objectives

- Understanding medium-access mechanisms
- Insight in their performance