Project 1 Report:
The Distributed Coordination Function (DCF) of 802.11

Shengxiang Zhu, Edward Sun,
Department of Electrical and Computer Engineering,
University of Arizona, Tucson, AZ 85721

1. Team members and responsibilities

We implemented a discrete time driven simulator in Python, which simulates the behavior of all nodes in the network in detail. It is written in OOP style and can easily scale to larger networks. All the test procedures are automated.

The team consists of two members: Shengxiang Zhu (Troy) who wrote the simulation program and analyzed the simulation results, and Edward Sun who drew the graphs of the simulated results and wrote the report.

2. Introduction of the simulator

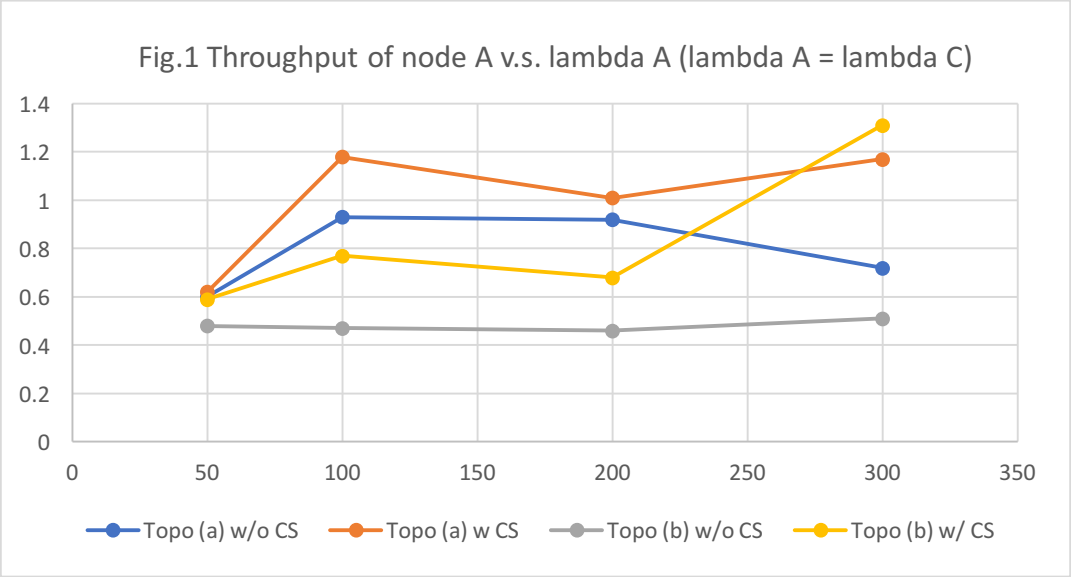
The simulator is contained in the folder named "MAP_Simulator", in which there are two files named "configuration.py" and "simulator.py" specifically. "configuration.py" is the configuration file which includes all the parameters used in the simulator. Please note the units of the parameters are defined in the code, which may not be standard. "simulator.py" is the main program executing the simulation. There is a folder named "logs", which stores all the simulation logs. The other ".pyc" files are compiled source codes automatically generated by Python interpreter for faster execution.

The main program consists of three parts, the class Tx, class Rx and class Simulator. Tx is the class for transmitter, Rx is the class for receiver, Simulator executes the simulation by passing messages among Tx's and Rx's.

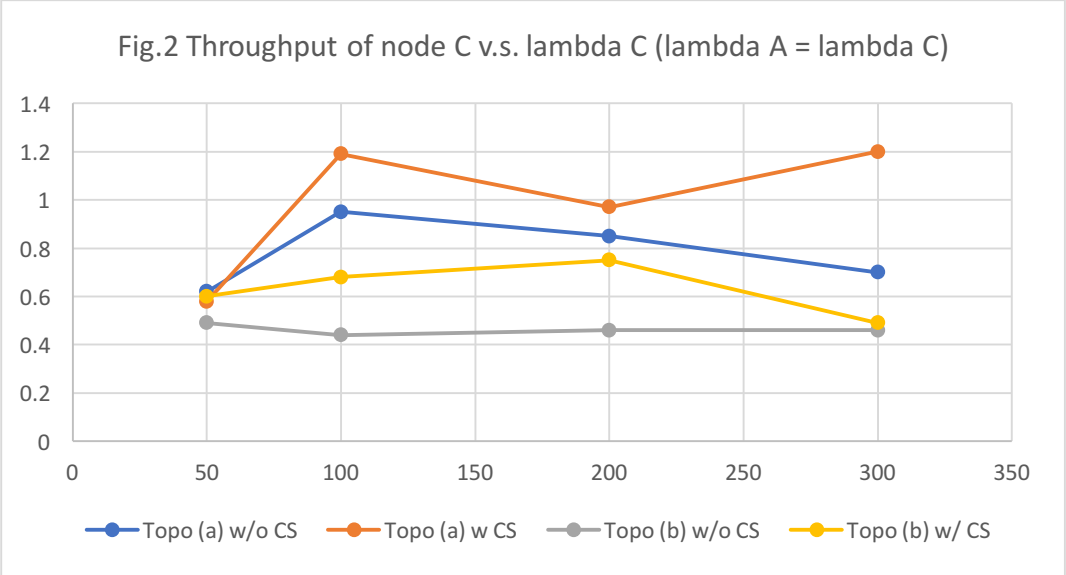
The simulator uses a discrete-time, time-driven simulation method. It generates several objects as nodes (Tx's and Rx's), which listen to messages from their common media and send out messages according to their own job queue. In abstract, these nodes maintain a finite state machine and reacts to the arriving packets and messages. The Simulator() object is to maintain a discrete time axis, which has a per-slot granularity. In each slot, the messages in the common medium are collected from those nodes sending messages. Then the messages will be sent to the nodes that are listening messages in this slot. To determine whether a node is listening or sending messages, we use a job queue to keep track of the state of the node. The behavior of each node is written in their class definition, according to different network protocols.

To run the simulator, "cd" to working directory and use "$python simulator.py" to execute. Runtime environment is Python 2.7.10 with basic libraries. To enable logging function to see the detailed log information, please comment out the logging configuration syntaxes. It may take longer to simulate if you enable log function.

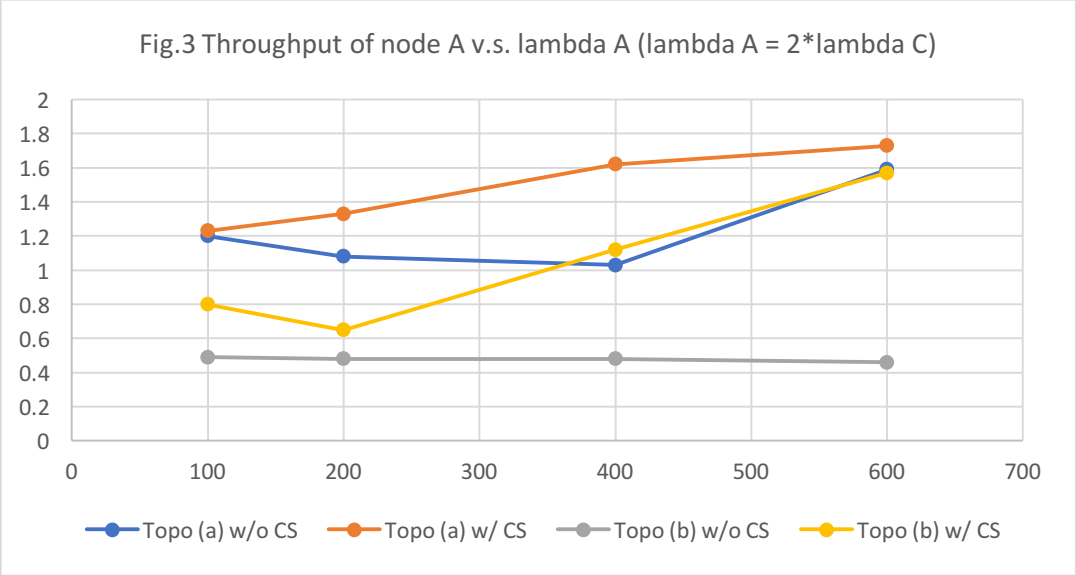
3. Results



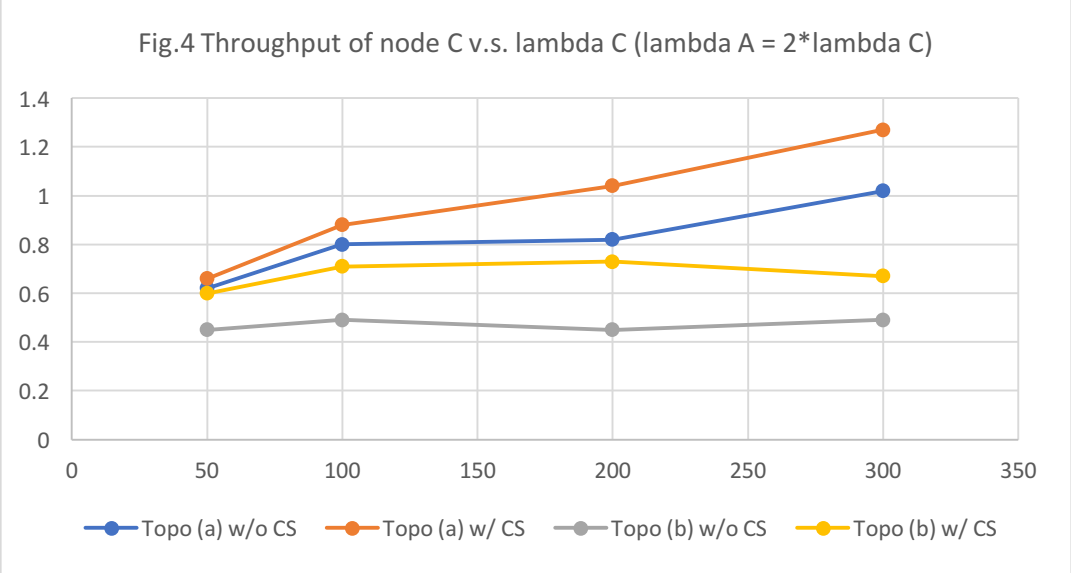
In this figure (Fig. 1), CS has higher throughput, the RTS/CTS process reduces a lot of collisions. Topology (a) has a higher throughput because all the nodes are in the same collision domain thus they can see all the messages in the common medium.



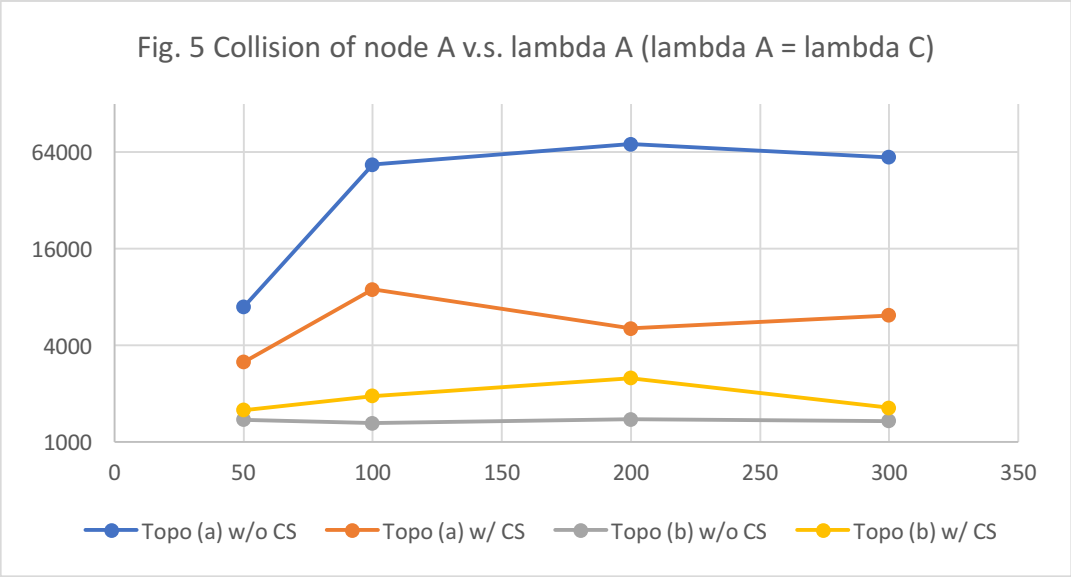
In this graph, similar to the previous one, node C has the same pattern as node A since they are similar to each other (same lambda, same topological structure).



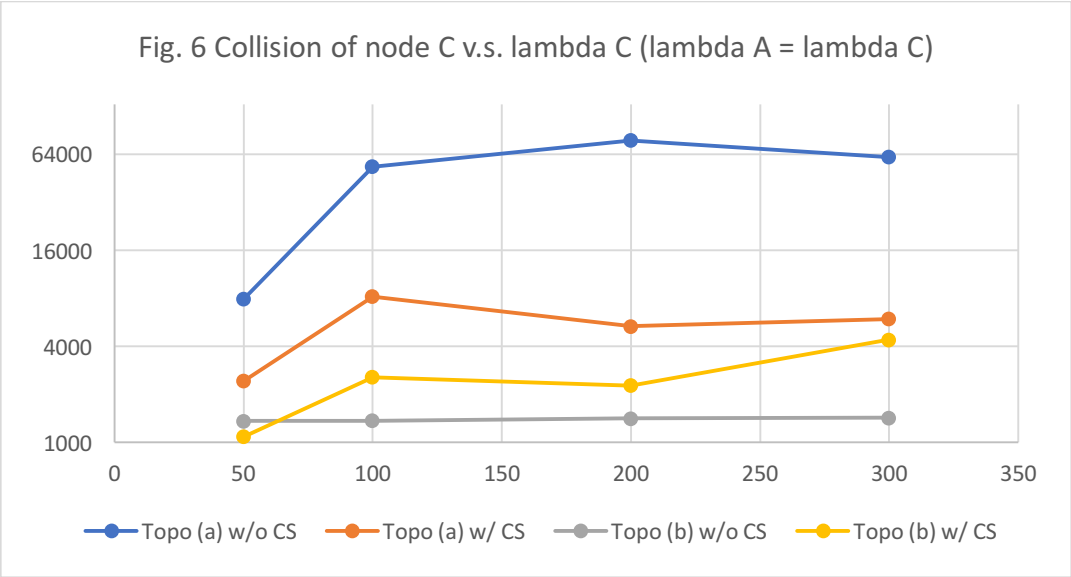
In this graph, the result is similar to the previous scenarios. CS has large benefit and the hidden terminal problem reduces the throughput of topology (b).



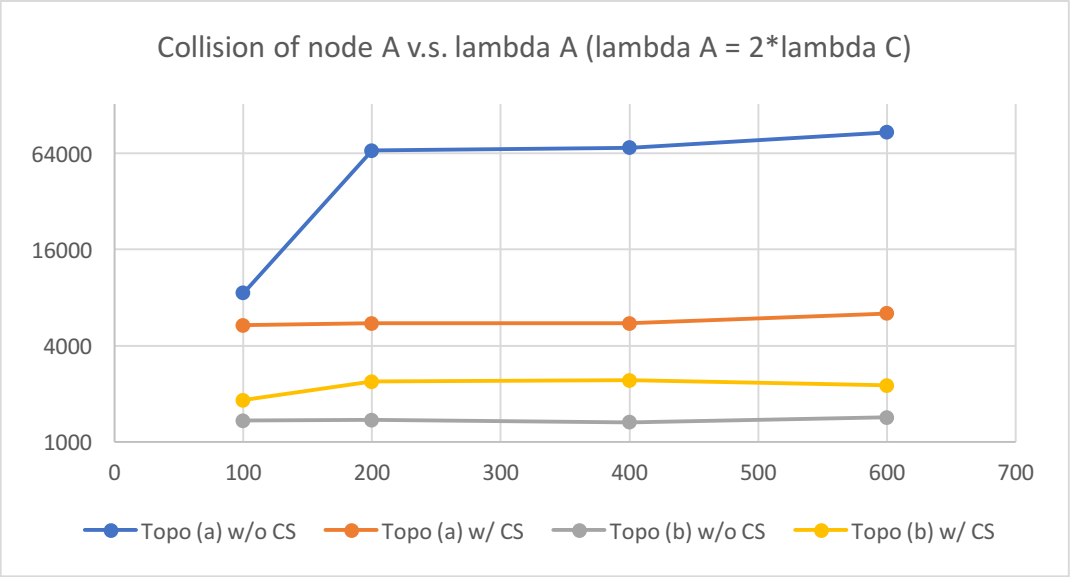
In this graph, the result is similar to the previous scenarios. We can see that the throughput of node C is about the half of node A since node A has twice the lambda value.



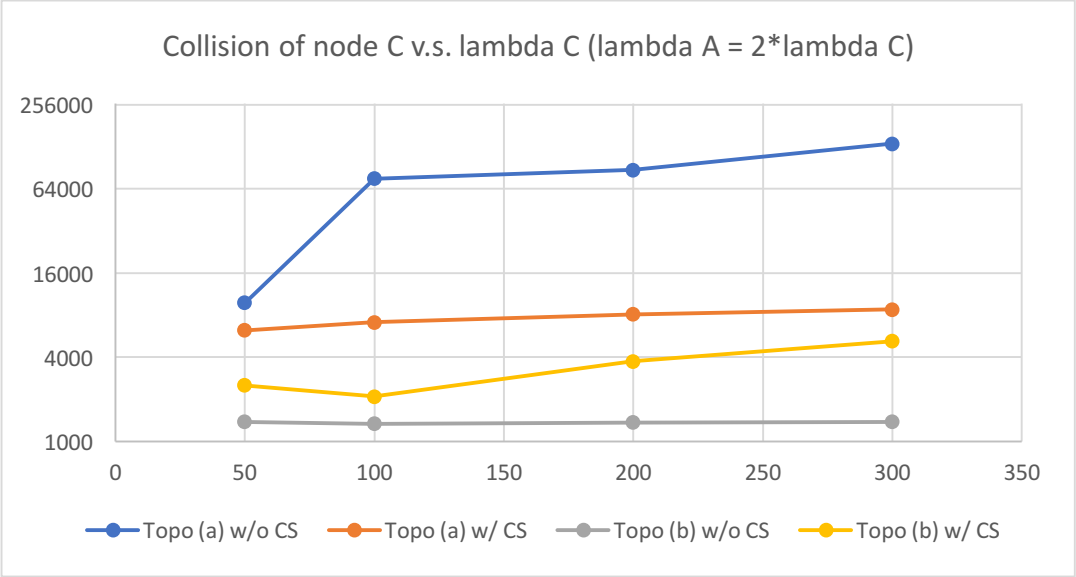
In this graph, node without CS has higher collision since they cannot terminate their transmission early by the RTS/CTS process. In addition, topology (a) has higher collision than topology (c). A possible reason is that in topology (a) all nodes are in the same collision domain so they can know collision instantly. However, in topology (c), a node cannot know its collision because it cannot see message coming from the other node. This node could not know its collision until the node misses ACK from the receiver. Another interesting thing is that with CS, node in topology (b) has a higher collision count. It is because in this scenario the throughput is much higer thus lead to higher number of collisions (refer to Fig. 3).



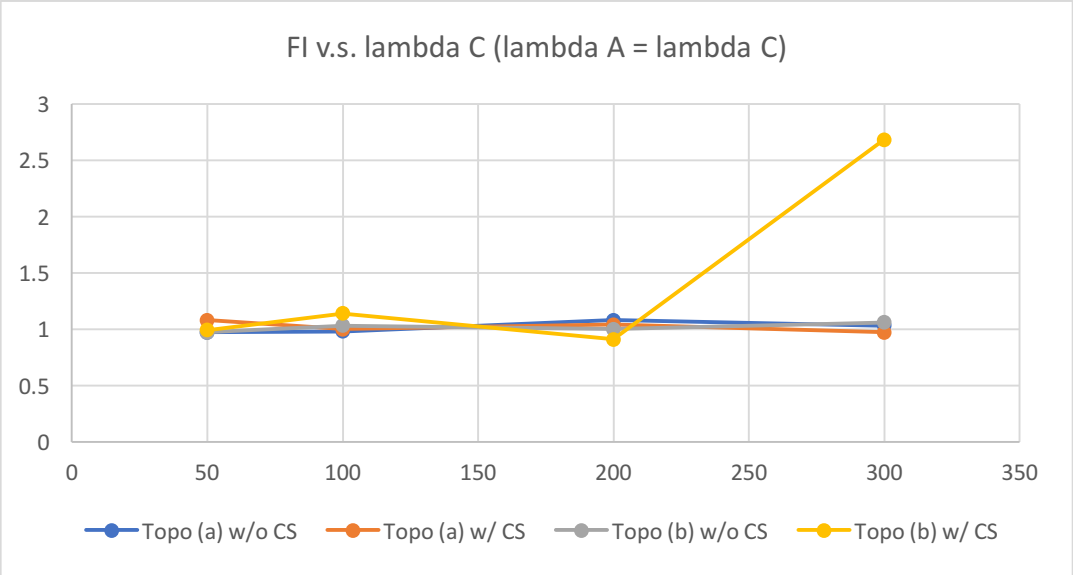
In this graph, the result is similar to the previous one. Node A and node C here are equivalent in their topological structure.



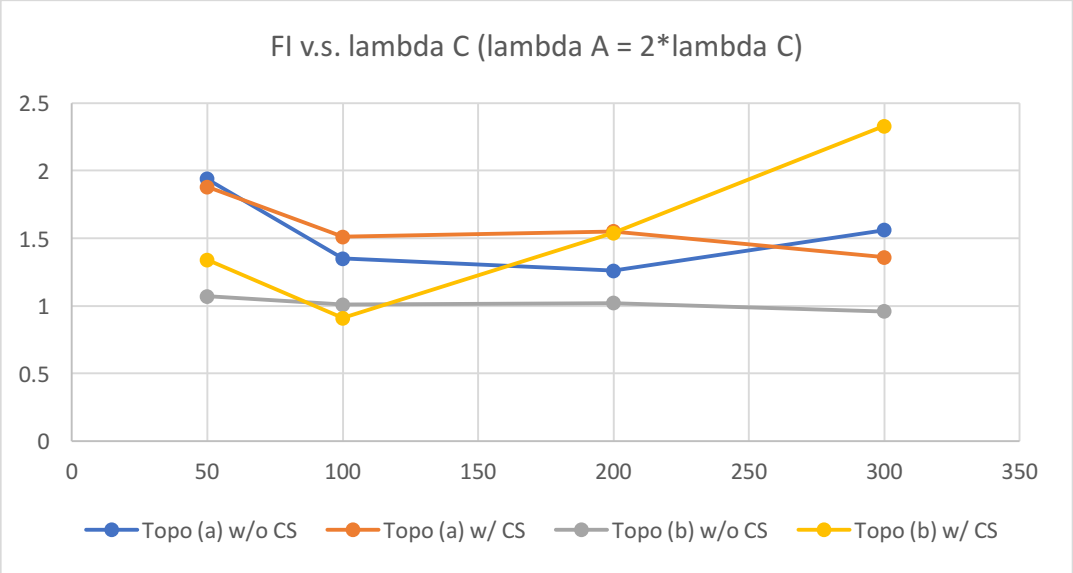
In this graph, the result is similar to the previous one. Interestingly the number of collision remains the same while lambda A doubles. A possible reason is that the traffic is high enough thus the transmitter works in a saturated status.



In this graph, the node C has a much higher collision count. It is possible that there is network capture effect where the node A dominates the network resources. Another reason is that the node A has much more packets in the medium so it dominates the network continuously. As a result, when node C tries to send out messages it often gets collisions.



In this graph, FI is close to 1.0 as expected. However, the FI rises above 2.0 in the yellow line, which need further analysis.



In this graph, FI decreases in topology (a) because with the increase of lambda, the network tends to saturate, resulting in the FI convergence to 1.0. The yellow line goes up with higher lambda, which needs further analysis.