Project II WiFi DCF Protocol

by Jason Zhao 4405 3635

I. Project Description

DCF is a contention-based multiple access scheme (aka. best effort access). The scheme adopts carrier sense multiple access with collision avoidance (CSMA/CA) protocol. The CSMA/CA scheme is described as below.

1) A station that has frames ready to send much first sense the channel to find out whether it is currently empty or occupied by the frames transmitted by other stations.

2) If the channel is detected as busy status nodes will perform the defer process.

3) If the channel is detected as free for a period of inter-frame space (IFS) duration, the back-off (BO) counter start counting down and the node whose BO counter goes to zero starts to transmit.

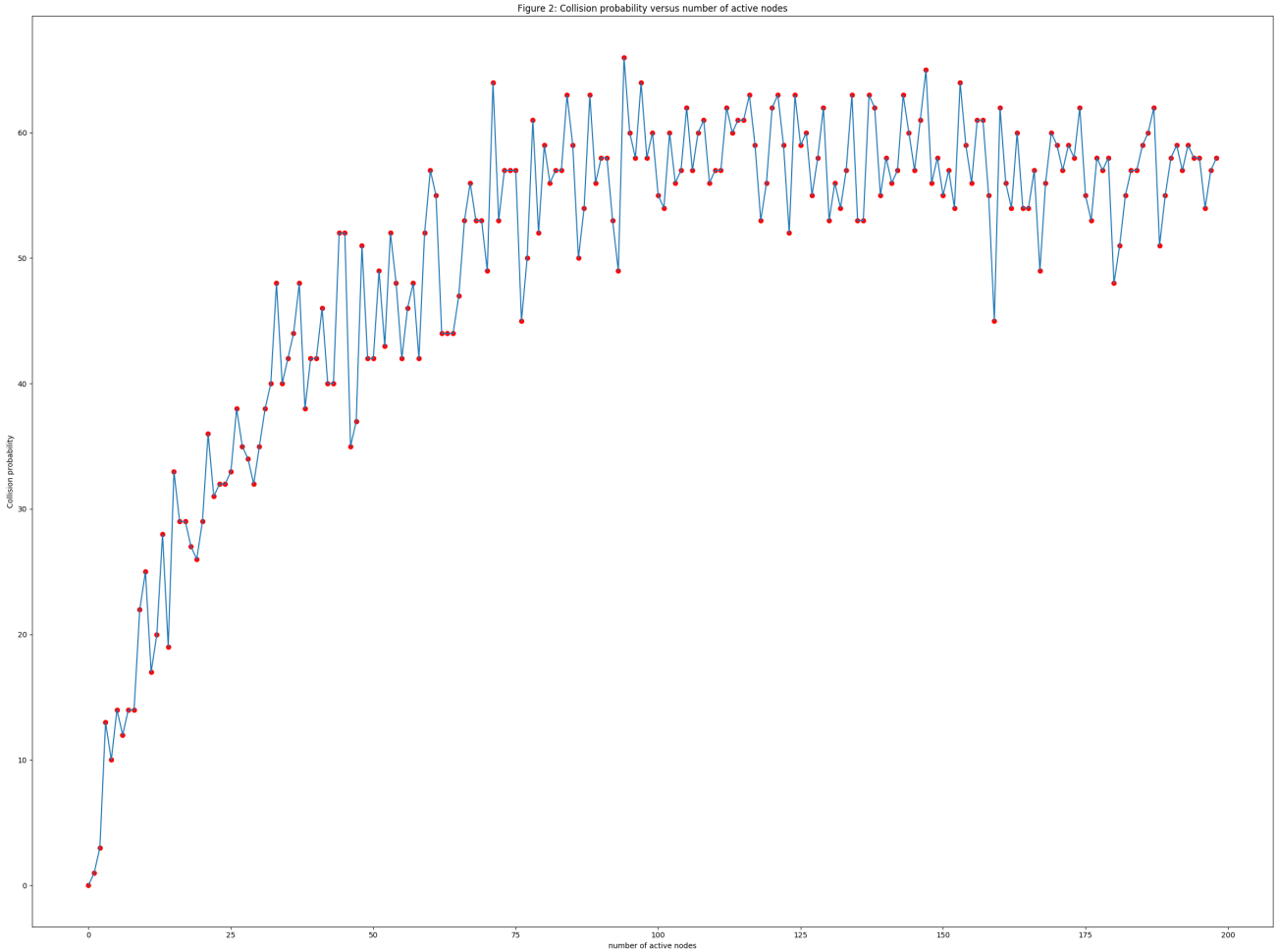
4) After transmission, the transmission node will wait for an acknowledgement (AKC) message from destination node. If ACK is received the transmission is completed and the node will content for next frame transmission if the traffic is assumed to be saturated meaning there are infinite number of frame to be sent. If it not received it within a given duration (time out), collision is assumed the increase the retrials counter and perform defer process.

5) During the defer process, the node will be randomly assigned an integer between $[0,\ 2\ \mathrm{n}\ -\ 1]$ where n is the number of retrials.

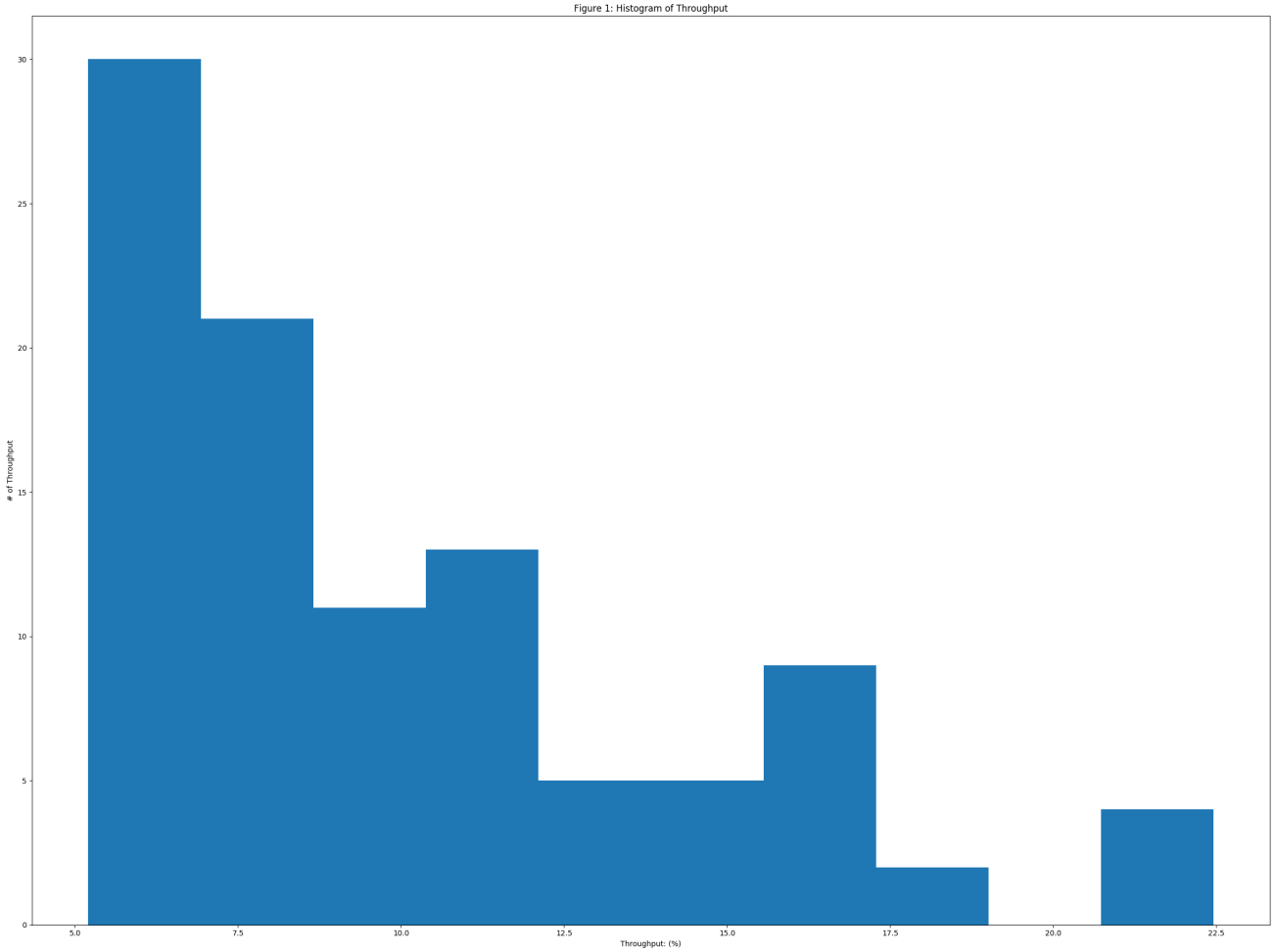
II. Project Outputs

Number of user (nodes): 200

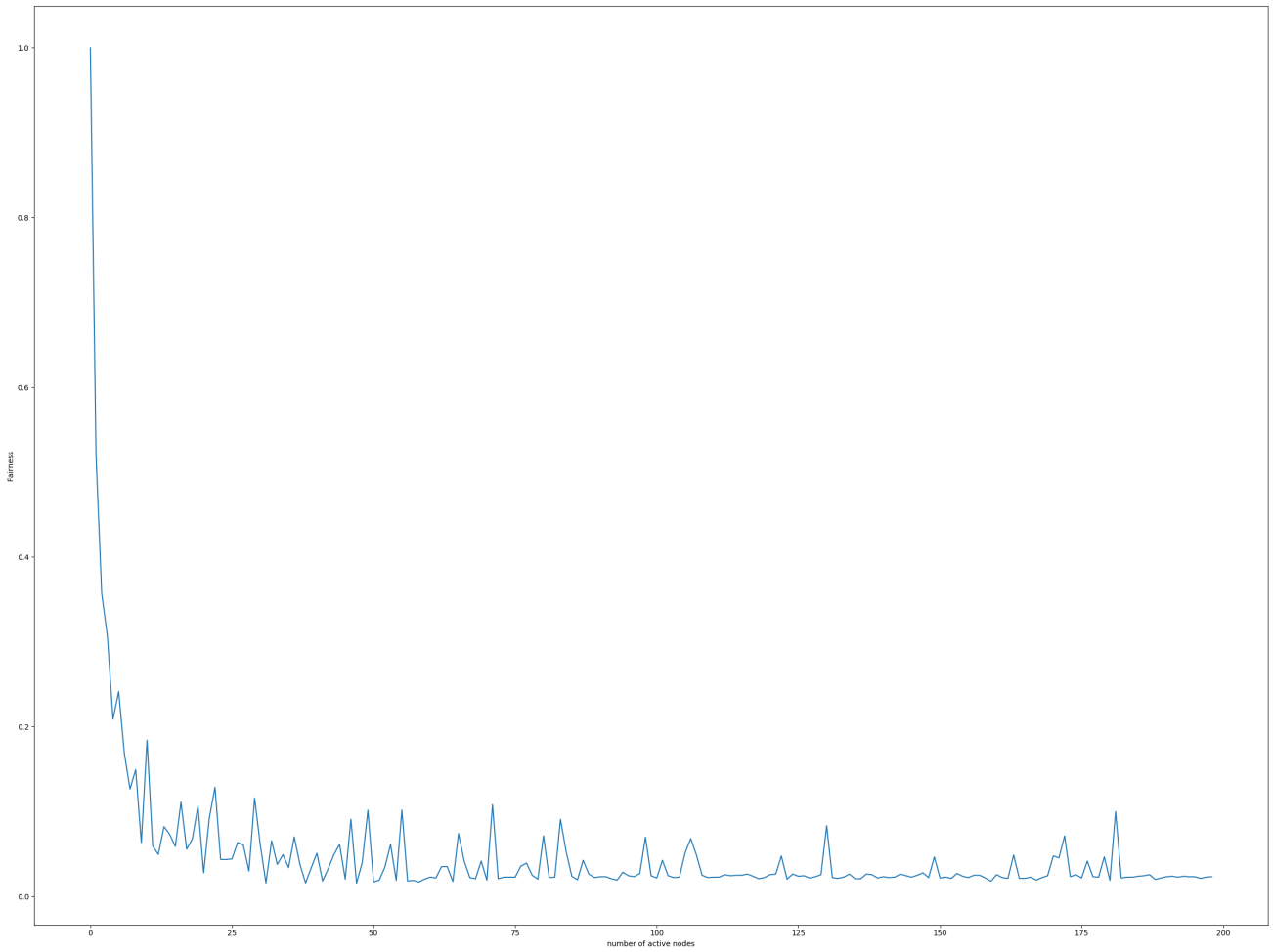
1) A figure of collision probability versus number of active users.



2) The histogram plot of throughput when there is one active user where N = 2048 and Rb = 6M bits/s.



3) A figure of fairness versus number of active users.



III. Appendix: Sourse Codes

```
import numpy as np
from numpy import zeros, argmin
from numpy.random import randint
import matplotlib.pyplot as plt

SIFS = 10  # us
slot = 20  # us slot time
DIFS = 2 * slot + SIFS
Rb = 6
Tb = 1 / Rb
CWmin = 32  # %minimal Contention Window
CWmax = 1024  # maximal Contention Window
RTS = round(20 * 8 * Tb)  # 20Byte for RTS
CTS = round(14 * 8 * Tb)  # 14Byte for CTS
ACK = round(14 * 8 * Tb)  # 14Byte for ACK
# Nnode = 2  # number of active nodes
fair = []
pc = []

# get collision probability
def get_collision(CWmin, n):
```

```
    s = 0
    for i in range(1, CWmin - 1):
        si = (i / 32) ** (n - 1)
        s += si
    return 1 - n / 32 * s


nodes, p, p1 = [], [], []
for Nusr in range(1, 50):  # Num of users
    Ncln = 0
    Nlp = 50
    # print(Nusr)
    for i in range(Nlp):
        BC = np.random.randint(CWmin, size=Nusr)
        minBC = min(BC)
        BC = BC - minBC
        ind = np.where(BC == 0)
        if len(ind[0]) > 1:
            Ncln += 1
    nodes.append(Nusr)
    p.append(Ncln / Nlp * 100)
    # p1.append(get_collision(CWmin, Nusr) * 100)


for Nnode in range(1, 200):

    thr = []
    # print('Nnode = %d' % Nnode)
    # assume both nodes are saturated with packets to be sent out
    # node status loop
    cnt = zeros(Nnode, dtype=int)
    Ndata = zeros(Nnode, dtype=int)
    BC = zeros(Nnode, dtype=int)
    ch, cln, sent = [], [], []
    first, Ncln = 1, 0

    Nloop = 100
    Npkg = 1
    npkg = 1

    for loop in range(int(Nloop)):
        ch = []
        # if loop % 100 == 0:
        #     print('loop %d times' % loop)
        if first:
            first = 0
            BC = randint(CWmin, size=Nnode)  # set back-off counter;
            # Ndata=np.random.randint(29,2347,size=Nnode)*8*Tb #random data length
            Ndata = randint(32, 33, size=Nnode) * 8 * Tb  # set a data length
            Ndata = Ndata.astype(int)
```

```
        # wait for DIFS to start out
        ch += [0] * DIFS
        [mBC, ind] = min(BC), argmin(BC)
        cln = []

        for i in range(Nnode):
            once = 1
            if i != ind and BC[i] == mBC:
                cln.append(i)
                cnt[i] += 1
                if once:
                    cnt[ind] += 1
                    once = 0
            # for j in range(len(cln)):
            #     print('collision nodes: {}'.format(cln))

        ch += [0] * (mBC * slot)
        # print(BC)
        BC = BC - mBC  # adjust BO values
        # print(BC)

        if len(cln) == 0:
            cnt[:Nnode] = 0
            # print('the node %d win the channel access' % ind)
            Npkg += 1
            print('Npkg: %d' % Npkg)
            if ind == 0:
                npkg += 1
                print('npkg: %d' % npkg)
            sent.append(ind)
            BC[ind] = np.random.randint(CWmin)
        else:
            Ncln += 1
            BC[ind] = randint(CWmin * 2 ** min(5, cnt[ind]))
            for i in range(len(cln)):
                BC[cln[i]] = randint(CWmin * 2 ** min(5, cnt[ind]))

        ch += [1] * RTS
        ch += [0] * SIFS
        ch += [1] * CTS
        ch += [0] * SIFS
        ch += [1] * Ndata[ind]
        ch += [0] * SIFS
        ch += [1] * ACK

        # thr.append(Ndata[0] / len(ch) * 100)
    pc.append(Ncln / Nloop * 100)
    fair.append(npkg / Npkg)


# plt.plot(ch)
```

```
# plt.grid(True)
# plt.show()
# fig.savefig('1', dpi=300)

# plt.plot(pc, 'ro')
# plt.plot(range(1, 10), fair)


# plt.hist(thr)
# plt.title('Figure 1: Histogram of Throughput')
# plt.xlabel('Throughput: (%)')
# plt.ylabel('# of Throughput')
# plt.show()
# fig.savefig('Histogram of Throughput')

fig1 = plt.figure(1, figsize=(24, 18))
plt.plot(pc, 'ro')
plt.plot(pc)
plt.title('Figure 2: Collision probability versus number of active nodes')
plt.xlabel('number of active nodes')
plt.ylabel('Collision probability')
plt.show()
fig1.savefig('Probability of collision')


fig2 = plt.figure(1, figsize=(24, 18))
plt.plot(fair)
plt.xlabel('number of active nodes')
plt.ylabel('Fairness')
plt.show()
fig2.savefig('Fairness')
```