12.11 토 - [MATLAB] WLAN enhancement lab

```
%-----------------------------------------
%  INC4103: IEEE 802.11 DCF & EDCA
% (1) Derive throghput model of DCF
% (2) Understand service differentiation of EDCA
% (3) Understand throughput-fairness of DCF
%-----------------------------------------
clear; clc;
close all;

BEB_ENABLE = 0;

%-----------------------------------------
% ASSUMPTIONS
%-----------------------------------------
% A1. Carrier sensing is perfect so that all the users can completely
% detect busy channel
% A2. There is no transmssion failure due to poor quality of wireless
% channel, i.e., transmission failure only occurs due to collision
% A3. All the users always have data to send

%-----------------------------------------
% PARAMETERS & VARIABLES
%-----------------------------------------
N_slot = 1000000;         % total number of slots
N_user = 2;             % number of users

R_data = 24*10^6*ones(1,N_user);
%R_data = [48 24 12]*10^6;
                        % transmission rate

%CW_min = 16*ones(1,N_user);             % minimum contention window
CW_min = [8 4];

AIFSN = 3*ones(1,N_user);  % AIFSN for EDCA, default = 3 considering DIFS
AIFSN = [2 4];

L_data = 1000*ones(1,N_user);
                        % data frame size (byte)
%L_data = [2000 1000 500];

% 802.11g MAC/PHY spec.
T_slot = 9*10^-6;      % time slot = 20 us
L_macH = 28;            % length of the MAC Header (bytes)
T_phyH = 44*10^-6;     % PHY Header transmission time (sec)
T_sifs = 10*10^-6;      % SIFS time (sec)
L_ack = 14;             % length of ACK (byte)
R_ack = 6*10^6;    % basic rate for ACK (bit/sec)
T_ack = T_phyH + L_ack*8/R_ack;
                        % ACK transmission time (sec)
T_data = T_phyH + (L_macH + L_data)*8./R_data;
                        % data frame transmission time (sec)

T_txslot = floor( (T_data + T_sifs + T_ack)/T_slot);
            % number of slots required to transmit one data frame

CW_max = 1024;
            % when BEB is enabled, CW is doubled if collision occurs
            % its maximum value is CW_max
CW = CW_min.*ones(1,N_user);               % initial contention window


%initial backoff counter & aifs
bc = ceil(rand(1,N_user).*CW);
aifs = AIFSN;
aifs_reset = zeros(1,N_user);

tx_state = zeros(N_slot,N_user);
% tx_state(i,j) = transmission status at time slot i for user j
STATE_BC  = 0;   % backoff state
STATE_TX  = 1;   % transmission state (without collision)
STATE_CS  = 2;   % carrier-sensing state
STATE_COL = 3;   % collision state

n_txnode = 0; % number of TX nodes
```

```
n_access = zeros(1,N_user);
    % number of channel access per user
n_collision = zeros(1,N_user);
    % number of collisions per user
n_success = zeros(1,N_user);
    % number of successful channel access without collision
    % n_access = n_collision + n_success

i=2;

%---------------------------------------------------------------------
% START SIMULATION
%---------------------------------------------------------------------
while(i < N_slot-1)

    % check if channel is idle
    if (n_txnode == 0)
        for j=1:N_user
            if (aifs(j) > 0)
                aifs(j) = aifs(j) - 1;
                % wait for AIFS
            end
        end
        for j=1:N_user
            if (aifs(j) == 0)
                %if (aifs_reset(j) == 1)
                    bc(j) = bc(j) -1;
                %elseif (aifs_reset(j) == 0)
                %    bc(j) = 0;
                %end
            end

    end
    % decrement backoff counter by 1 for users after waiting for AIFSN
end
% if channel is busy, do not change aifs & backoff counter

% check whether BC = 0
for j=1:N_user
    if (bc(j) == 0)
        tx_state(i:(i+T_txslot(j)-1),j) = STATE_TX;
        % set sate from i to i+T_txslot-1 = STATE_TX
        n_txnode = n_txnode + 1;
        n_access(j) = n_access(j)+1;

        bc(j) = ceil(rand*CW(j));
        aifs(j) = AIFSN(j);
        % re-select a new random backoff & aifs
    end
end

% update state
% if channel is busy
if (n_txnode ~= 0 )
    % if at least one node is in transmision state
    max_duration = max( T_txslot.*(tx_state(i,:)==STATE_TX) );
    for (j=1:N_user)
        if (tx_state(i,j) ~= STATE_TX)
            tx_duration = i+max_duration-1;
            tx_state(i:tx_duration,j) = STATE_CS;
            % set state = carrier sensing
            aifs(j) = AIFSN(j);
            %aifs_reset(j) = 1;
            % reset aifs after sensing busy channel
        end
    end

    % check collision
    if (n_txnode == 1)
        % only one node accesses channel, i.e., no collision
        for (j=1:N_user)
            if (tx_state(i,j) == STATE_TX)
                n_success(j) = n_success(j) + 1;
                %aifs_reset(j) = 0;
                % BEB here...
                if (BEB_ENABLE == 1)
                    CW(j) = CW_min(j);
                end
            end
        end
    elseif (n_txnode > 1)
        % more than two nodes access channel => collision
        for (j=1:N_user)
            if (tx_state(i,j) == STATE_TX)
                tx_duration = i+T_txslot(j)-1;
```

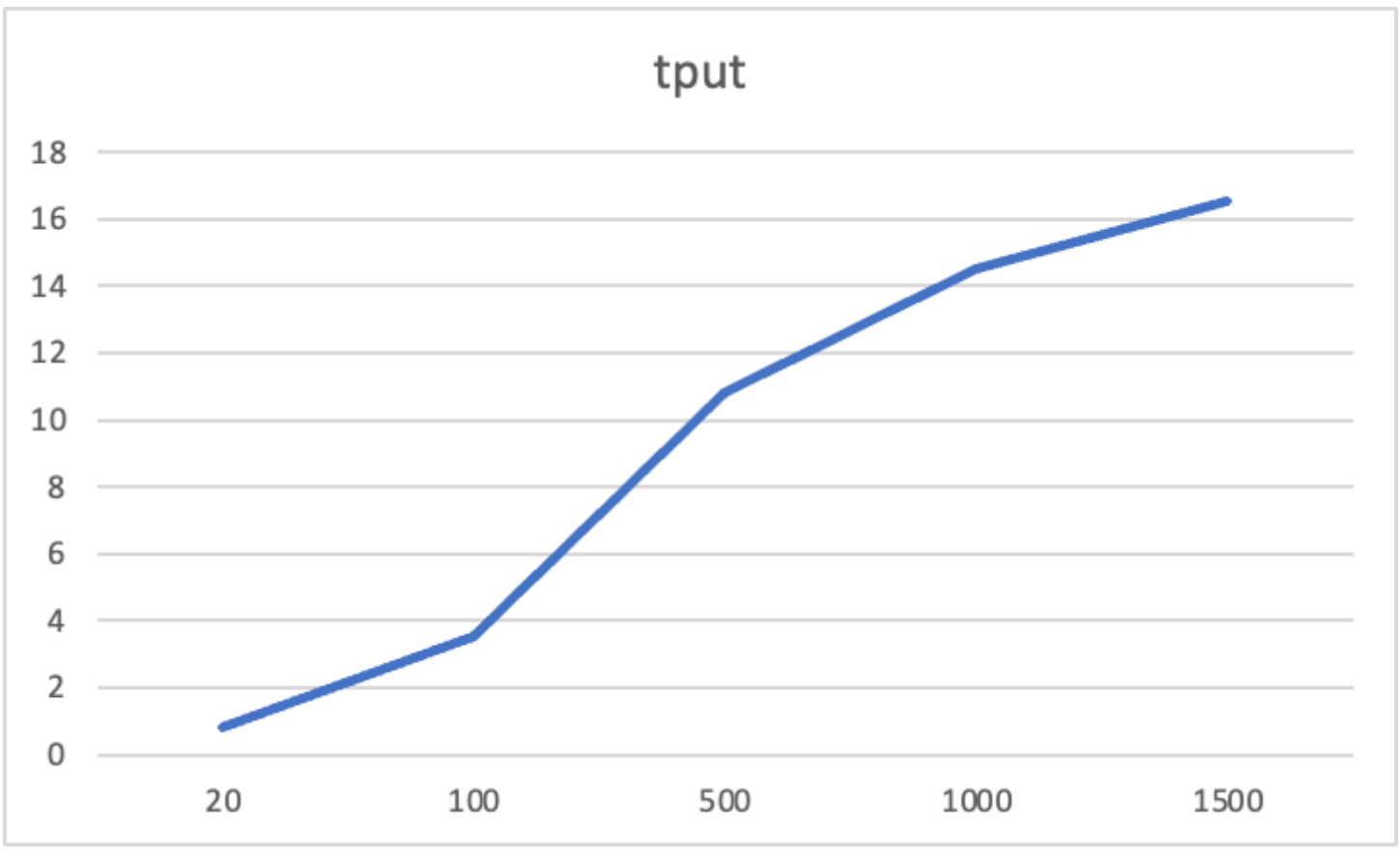
```
                    tx_state(i:tx_duration,j) = STATE_COL;
                    % set state = collision
                    n_collision(j) = n_collision(j)+1;
                    %aifs_reset(j) = 1;
                    % BEB here.....
                    if (BEB_ENABLE == 1)
                        CW(j) = min(CW(j) * 2, CW_max);
                    end
                end
            end
        end % end for collision-check

        i = i + max_duration+1;   % increase time index by T_txslot
        n_txnode = 0;
    else
        i=i+1;  % increase time index by 1 (if n_txnode = 0)
    end

end

%-------------------------------------------------------------------


%-----------------------------
% statistics
%-----------------------------
n_access
t_access = n_access.*T_data
    % channel access time
%n_collision
%n_success
%per_user_access = n_access/(sum(n_access))

per_user_th = (n_success .* L_data * 8) / (N_slot*T_slot) / 10^6  % Mb/s
total_th = sum(per_user_th) % Mb/s
fairness_index = total_th^2 / (N_user * sum(per_user_th.*per_user_th))

%collision_prob = sum(n_collision)/sum(n_access)
%utilization = sum(sum(tx_state==1)) / N_slot
```

실습 1.2 - 프레임 크기 (Ldata)가 throughput에 끼치는 영향

R_data = 24Mbps

CW_min = 16

AIFSN = 3



L_data = 20

```
per_user_th =

    0.4067    0.4065


total_th =

    0.8133
```

프레임사이즈가 작기 때문에 오버헤드가 상대적으로 커져 tput이 전송률에 비해 매우 낮다.

L_data = 100

```
per_user_th =

    1.7771    1.7592


total_th =

    3.5363
```

L_data = 500

```
per_user_th =

    5.4244    5.3853


total_th =

   10.8098
```

L_data = 1000

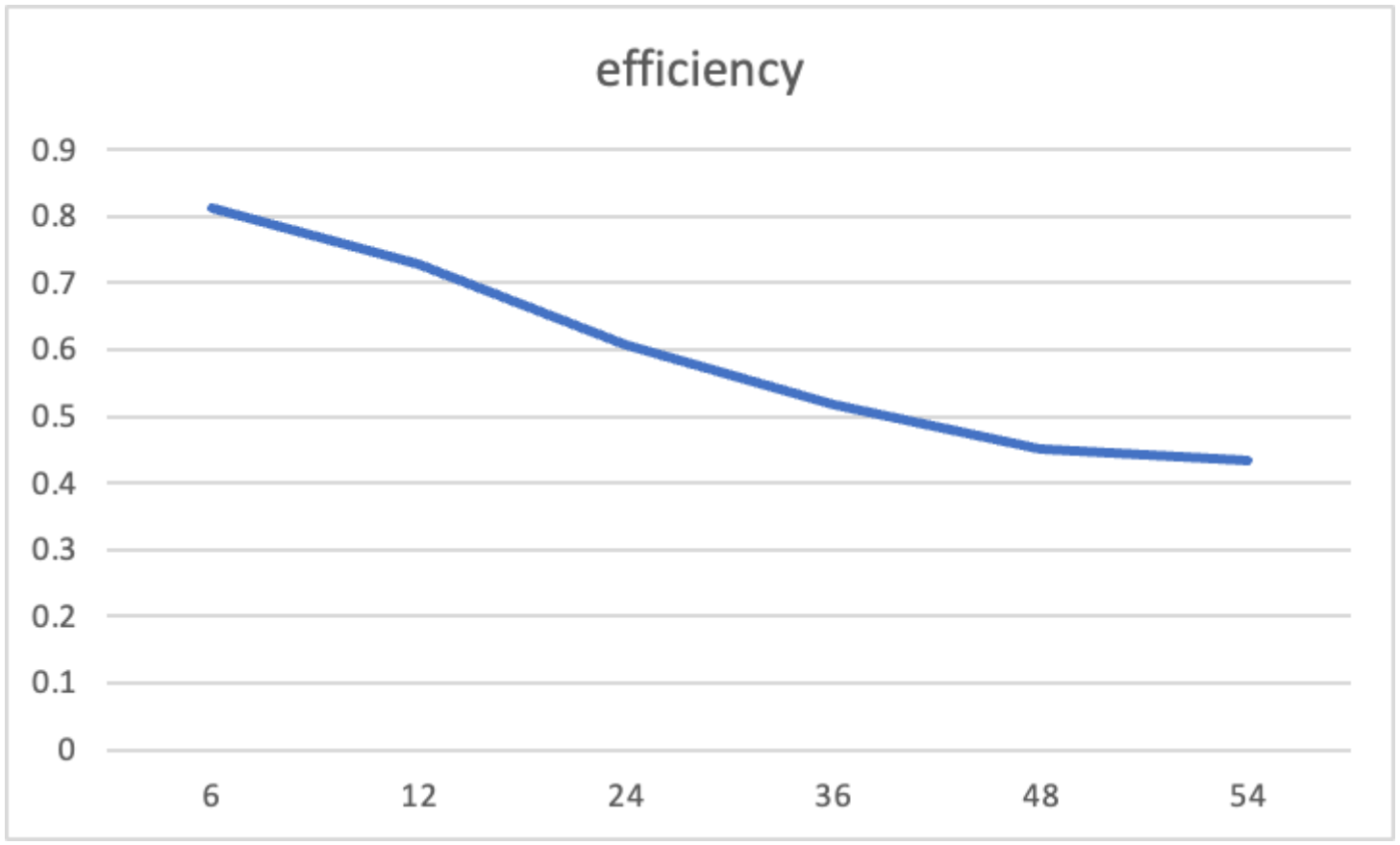
```
per_user_th =

    7.2604    7.2560


total_th =

   14.5164
```

L_data = 1500

```
per_user_th =

    8.3013    8.2507


total_th =

   16.5520
```

전송 속도는 24Mbps이지만 실제 속도는 총 14Mbps이다. 오버헤드로 인해 tput은 24Mbps가 되지 못한다.

실습 1.2 - 데이터 전송 속도 (Rdata)의 영향



R_data = 6

```
per_user_th =

    2.4142    2.4516


total_th =

    4.8658
```

total tput은 4.8658Mbps로, 이는 전송률의 약 81.1%이다.

오버헤드가 상대적으로 크지 않아, 효율이 좋은 편이다.

R_data = 12

```
per_user_th =

    4.3244    4.4160


total_th =

    8.7404
```

R_data = 24

```
per_user_th =

    7.3262    7.2551


total_th =

   14.5813
```

R_data = 36

```
per_user_th =

    9.2720    9.3760


total_th =

   18.6480
```

R_data = 48

```
per_user_th =

   10.8044   10.8124


total_th =

   21.6169
```

R_data = 54

```
per_user_th =

   11.7653   11.6996


total_th =

   23.4649
```

total tput은 23.4649Mbps로, 이는 전송률의 약 43.45%이다.

전송률의 절반도 채 안 되는데, 이는 전송 속도가 빨라질수록 오버헤드로 인해 tput이 전송률을 따라가지 못하기 때문이다.

실습 2.1 - CW 차별화와 AIFS 차별화 비교

CWmin = [16 16 16], AIFSN = [4 4 4]

```
per_user_th =

    4.5636    4.5351    4.5733


total_th =

   13.6720
```

단말별로 tput이 비슷함

CWmin 값 다르게 - CWmin = [8 16 32], AIFSN = [4 4 4]

```
per_user_th =

    8.0960    3.7796    1.8142
```

```
total_th =

   13.6898
```

채널 접속 확률은 CW_min 크기에 반비례한다. CW_min이 작을수록 경쟁에서 이길 확률이 높아진다.

tput은 8, 3.8, 1.8 Mbps로 약 4:2:1의 비율로 나타난다.

AIFSN 값 다르게 - CWmin = [16 16 16], AIFSN = [2 4 8]

```
per_user_th =

    8.2889    4.9804    1.0720


total_th =

   14.3413
```

데이터 하나 보내기 전에 여러번 처리해야한다. 다른 단말 전송이 감지되면 멈췄다가, 쉬고, 다시 쉬고 반복한다.

tput은 8.4, 4.9, 1.1Mbps로 CWmin을 차별화한 경우보다 단말별 tput이 더욱 차별화되었다.

실습 2.2 - 절대적인 채널 접속 차별화

단말 1- 높은 우선순위

단말2 - 낮은 우선순위

Policy 1: AIFSN 차별화

CWmin = [8 8], AIFSN = [2 4]

```
per_user_th =

   10.6080    3.9387


total_th =

   14.5467
```

CWmin = [8 8], AIFSN = [2 8]

```
per_user_th =

   14.5947    0.4036


total_th =

   14.9982
```

위의 경우보다 차별화가 더 되었다.

CWmin = [8 8], AIFSN = [2 12]

```
per_user_th =

   15.7360         0


total_th =

   15.7360
```

단말 2는 아예 access하지 못하였다.

Policy 2: CW 차별화

CWmin = [8 16], AIFSN = [2 2]

```
per_user_th =

   10.0613    4.7547


total_th =

   14.8160
```

CWmin = [8 32], AIFSN = [2 2]

```
per_user_th =

   12.3867    2.7564


total_th =

   15.1431
```

CWmin = [8 64], AIFSN = [2 2]

```
per_user_th =

   13.8444    1.5573


total_th =

   15.4018
```

tput 차별화가 AIFSN 차별화를 한 경우에 비해 잘 되지 않았다.

- Policy1에서 단말1의 절대적인 채널 접속 우선순위 (즉, 단말 1과 단말2가 채널 접속 경쟁을 할 때 항상 단말1이 먼저 채널을 접속함)를 보장하는 AIFSN2의 최소값은 얼마인가?

  AIFSN2 = 10

  ### CWmin = [8 8], AIFSN = [2 10]

  ```
  n_access =

         17696           0


  t_access =
  ```

```
    6.8425         0


per_user_th =

   15.7298         0


total_th =

   15.7298


fairness_index =

    0.5000
```

- Policy2에서 단말1의 절대적인 채널 접속 우선순위를 보장할 수 있나? 있다면 CWmin,2의 조건을 구하고 없다면 그 이유는?

  CWmin.2 = 262144 (=2^18)

  ### CWmin = [8 262144], AIFSN = [2 2]

  ```
  n_access =

         17702           0


  t_access =

      6.8448         0


  per_user_th =

     15.7351         0


  total_th =

     15.7351


  fairness_index =

      0.5000
  ```

  가능은 하다. 다만, CWmin=262144는 값이 너무 크므로 프로토콜 규격에 맞지 않을 것이다.

- (AIFSN2 = 4, CWmin,2 = 8)인 경우와, (AIFSN2 = 8, CWmin,2 = 4)인 경우 어느 경우가 단말1의 채널 접속 우선순위를 높일 수 있나? 그 이유는?

  ### AIFSN2 = 4, CWmin,2 = 8

  CWmin 차별화 → 효과 미미함

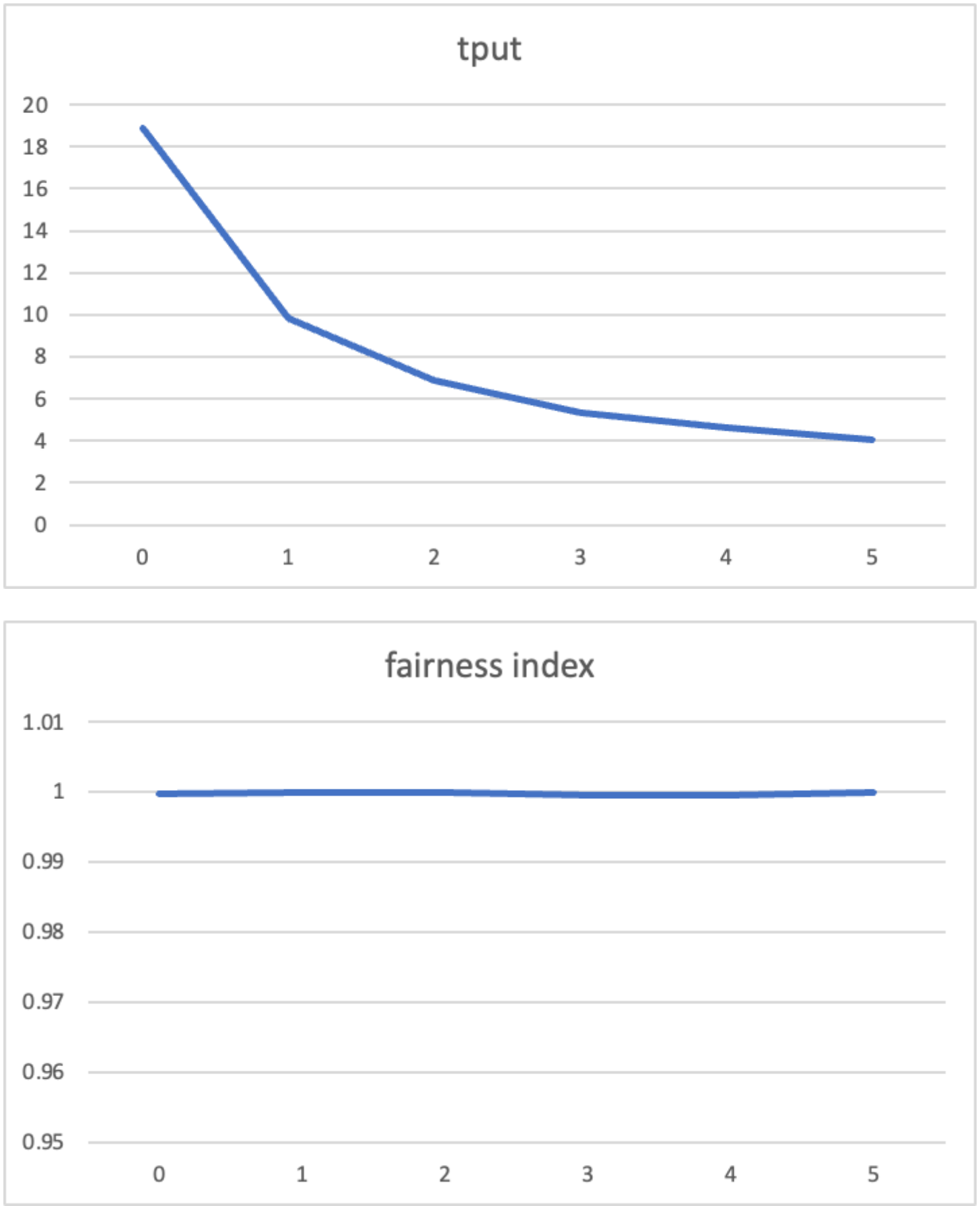
  ```
  per_user_th =

     10.4871    4.0373


  total_th =

     14.5244
  ```

  ### AIFSN2 = 8, CWmin,2 = 4

  AIFSN 차별화 → 접속 우선순위를 높이는데 효과가 좋음

```
per_user_th =

   13.7956    0.7156


total_th =

   14.5111
```

실습 3.1 - multi-rate WLAN 환경에서의 fairness



N_user = 5

채널 상태 나쁜 단말 - 6Mbps

채널 상태 좋은 단말 - 48Mbps

CW_min = 16

AIFSN = 3

L_data = 1000

채널 상태 나쁜 단말 0개

```
per_user_th =

    3.8364    3.7138    3.7218    3.7484    3.8436


total_th =

   18.8640

fairness_index =

    0.9998
```

채널 상태 나쁜 단말 1개

```
per_user_th =

    1.9804    1.9520    1.9920    1.9742    1.9440


total_th =

    9.8427


fairness_index =

    0.9999
```

채널 상태 나쁜 단말 2개

```
per_user_th =

    1.3751    1.3849    1.3831    1.3787    1.3511


total_th =

    6.8729


fairness_index =

    0.9999
```

채널 상태 나쁜 단말 3개

```
per_user_th =

    1.0738    1.0480    1.0329    1.0978    1.0827


total_th =

    5.3351


fairness_index =

    0.9995
```

채널 상태 나쁜 단말 4개

```
per_user_th =

    0.9280    0.9556    0.8898    0.9173    0.9191


total_th =

    4.6098


fairness_index =

    0.9995
```

채널 상태 나쁜 단말 5개

```
per_user_th =

    0.8187    0.8142    0.8142    0.8080    0.8080


total_th =

    4.0631


fairness_index =

    1.0000
```

- 전송 속도의 차이에도 불구하고 모든 단말이 비슷한 throughput을 가지는 이유와 k 값의 증가에 따라 각 단말의 throughput이 낮아지는 이유를 설명

  전송속도가 낮은 단말은 같은 크기의 프레임 전송에 많은 시간이 소요되기 때문에, 전송 속도가 높은 다른 단말의 전송 기회가 감소하여 전체 처리율이 저하된다.

실습 3.2 - multi-rate WLAN 환경에서의 throughput 저하 해결

해결 방안: 각 단말의 채널 접속 시간을 비슷한 수준으로 조절하여 tput이 전송 속도에 비례하도록 함

N_user = 3

R_data = [12 24 48] → 전송률은 1:2:4 비율이다.

방안 1: CWmin 값 차별화 - 채널 접속 횟수를 전송 속도에 비례하도록 설정

CWmin = [32 16 8] → R_data와 반비례하게 설정해야함

```
n_access =

        3242        6337       11885


t_access =

    2.3645    2.4503    2.5592


per_user_th =
```

```
    1.9920    4.1493    8.7671


total_th =

   14.9084


fairness_index =

    0.7556
```

n_access = 3200, 6300, 12000 로 약 1:2:4 횟수로 할당되었음

t_access = 2.4, 2.5, 2.6 으로 비슷한 시간 할당됨

tput = [2.0, 4.1, 8.8] 로 약 1:2:4 비율로 나옴

방안 2: 채널 접속 시간 차별화 - 단말이 채널 접속 기회를 가졌을 때, 채널을 점유하는 시간을 비슷하도록 설정

CWmin = [16 16 16]

L_data = [500 1000 2000] 프레임 aggregation → R_data와 비례하게 설정해야함.

```
n_access =

        6694        6690        6658


t_access =

    2.6508    2.5868    2.5434


per_user_th =

    2.3182    4.6240    9.1804


total_th =

   16.1227


fairness_index =

    0.7803
```

n_access = [6700, 6700, 6700]으로 비슷함

t_access = 2.6, 2.6, 2.6 초 할당됨

tput = [2.3, 4.6, 9.3]로 전송속도와 비례하게 약 1:2:4 비율로 나옴

같은 시간 동안 high-rate 단말은 low-rate 단말보다 더 많은 프레임을 전송할 수 있다. TXOP로 frame을 합치면 전송 오버헤드가 감소하는 효과가 있어, 전체 처리율이 향상되고 단말별 tput은 data rate와 비례하게 된다.