12.04 토 - [MATLAB] WLAN lab

정보통신공학과

2017112207

이건우

```
%-----------------------------------------
%  INC4103: fragmentation & aggregation
%  by E.-C. Park
%-----------------------------------------
clear;
close all;

% frags for fragmentation and A_MSDU/A_MPDU
FRAG_ENABLE = 0;
A_MSDU_ENABLE = 1;
A_MPDU_ENABLE = 0;


BEB_ENABLE = 0;

%-----------------------------------------
% ASSUMPTIONS
%-----------------------------------------
% A1. Carrier sensing is perfect so that all the users can completely
% detect busy channel
% A2. The bit error rate is constant
% A3. All the users always have data to send

%-----------------------------------------
% PARAMETERS & VARIABLES
%-----------------------------------------
N_slot = 1000000;         % total number of slots
N_user = 3;             % number of users
%N_frag = 3*ones(1,N_user);     % number of fragments
N_frag = [2 3 4];

%N_agg = 4*ones(1,N_user);      % number of packets aggreagted
N_agg = [2 4 8];

%BER = 0;
BER = 10^-5;                % bit error rate


TX_rate = 24*10^6*ones(1,N_user);
                        % transmission rate

CW_min = 16*ones(1,N_user);             % minimum contention window

L_pkt = 1000*ones(1,N_user);
L_frag = L_pkt./N_frag;
                        % packet size (byte)

if (A_MSDU_ENABLE == 1)
    FER = 1-(1-BER).^(L_pkt.*N_agg*8)
elseif (A_MPDU_ENABLE == 1)
    FER = 1-(1-BER).^(L_pkt*8)    % frame error rate
elseif (FRAG_ENABLE == 1)
    FER = 1-(1-BER).^(L_frag*8)
end


% 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)
L_back = L_ack + 8;
Basic_rate = 6*10^6;    % basic rate for ACK (bit/sec)
T_ack = T_phyH + L_ack*8/Basic_rate;
                        % ACK transmission time (sec)
T_back = T_phyH + L_ack*8/Basic_rate;
if (FRAG_ENABLE == 1)
    T_data = T_phyH + (L_macH + L_frag)*8./TX_rate;
                        % data frame transmission time (sec)
    T_txslot = floor( N_frag.*(T_data + T_sifs + T_ack)/T_slot);
```

```
            % number of slots required to transmit one packet (consisting
            % of N fragmetns)
elseif (A_MSDU_ENABLE == 1)
    T_data = T_phyH + (L_macH + N_agg .* L_pkt)*8./TX_rate;
                        % data frame transmission time (sec)
    T_txslot = floor( (T_data + T_sifs + T_ack)/T_slot);
elseif (A_MPDU_ENABLE == 1)
    T_data = T_phyH + N_agg.*(L_macH + L_pkt)*8./TX_rate;
    T_txslot = floor( (T_data + T_sifs + T_back)/T_slot);
end
AIFSN = 3*ones(1,N_user);  % AIFSN for EDCA

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;


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 = zeros(N_slot,1); % 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)
                bc(j) = bc(j) -1;
            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);
                % 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)
                    if (FRAG_ENABLE == 1)
                        n_success(j) = n_success(j) + sum(rand(1,N_frag(j))>FER(j));
                    elseif (A_MSDU_ENABLE == 1)
                        flag_success = rand > FER(j);
                        n_success(j) = n_success(j) + flag_success;
                        if ( (flag_success == 0) && (BEB_ENABLE == 1))
                            CW(j) = min(CW(j) * 2, CW_max);
                        end
                    elseif (A_MPDU_ENABLE == 1)
                        n_success(j) = n_success(j) + sum(rand(1,N_agg(j))>FER(j));
                    end
                    % BEB here...
                    if (BEB_ENABLE == 1)
                        CW(j) = CW_min(j);
                    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;
                        % 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
        end % end for busy-channel

    end

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


    %-----------------------------
    % statistics
    %-----------------------------
    %n_access
    %n_collision
    if (FRAG_ENABLE == 1)
        per_user_th = (n_success .* L_frag * 8) / (N_slot*T_slot) / 10^6  % Mb/s
    elseif (A_MSDU_ENABLE == 1)
        per_user_th = (n_success .* N_agg .* L_pkt * 8) / (N_slot*T_slot) / 10^6  % Mb/s
    elseif (A_MPDU_ENABLE == 1)
        per_user_th = (n_success .* L_pkt * 8) / (N_slot*T_slot) / 10^6  % Mb/s
    end
    total_th = sum(per_user_th) % Mb/s
    %fairness_index = total_th^2 / (N_user * sum(per_user_th.*per_user_th))
        % fraction of time slot occupied without collision
    %collision_prob = sum(n_collision)/sum(n_access)
    %utilization = sum(sum(tx_state==1)) / N_slot
```

실습 1.1 - Fragmentation 여부에 따른 성능 비교

BER = 10^-6

fragmentation 없음, BER = 10^-6

```
FRAG_ENABLE = 1;
A_MSDU_ENABLE = 0;
A_MPDU_ENABLE = 0;

BEB_ENABLE = 0;

N_frag = [1 1 1];

BER = 10^-6;
```

```
FER =

    0.0080    0.0080    0.0080


per_user_th =

    4.6089    4.6329    4.5653


total_th =

   13.8071

>>
```

세 유저의 tput은 약 4.6Mbps로 동일하다.

전체 tput은 13.8071Mbps이다.

fragmentation [3 3 3], BER = 10^-6

```
FER =

    0.0027    0.0027    0.0027


per_user_th =

    3.1141    3.1135    3.0421


total_th =

    9.2696

>>
```

세 유저의 tput은 각각 약 3.1Mbps이다.

BER이 작으므로 FER이 작고, 재전송이 적게 발생한다.

DCF로 인한 오버헤드 때문에, 재전송이 적게 발생하는 환경에서 fragment 개수를 동일하게 한 경우 오히려 tput이 낮아졌다.

전체 tput은 9.2696Mbps이다.

fragmentation [2 3 4], BER = 10^-6

```
FER =

    0.0040    0.0027    0.0020


per_user_th =

    3.0849    3.0533    3.0191


total_th =

    9.1573
```

```
>>
```

BER이 작아서 FER이 작고 재전송이 적음.

fragment 개수가 다르지만, 재전송이 거의 일어나지 않았기 때문에 세 단말의 tput은 비슷하게 나타났다.

전체 tput은 9.1573Mbps로, 오버헤드로 인해 fragmentation을 하지 않은 경우에 비해 tput이 낮아졌다.

BER = 10^-4

fragmentation 없음, BER = 10^-4

```
FRAG_ENABLE = 1;
A_MSDU_ENABLE = 0;
A_MPDU_ENABLE = 0;

BEB_ENABLE = 0;

N_frag = [1 1 1];
BER = 10^-4;
```

```
FER =

    0.5507    0.5507    0.5507


per_user_th =

    2.0729    2.0862    2.0684


total_th =

    6.2276

>>
```

세 유저의 tput은 약 2Mbps로 비슷하다.

fragmentation [3 3 3], BER = 10^-4

```
FER =

    0.2341    0.2341    0.2341


per_user_th =

    2.3559    2.4006    2.3769


total_th =

    7.1333

>>
```

프레임을 분할하면 프레임을 분할하지 않은 것 대비 FER이 감소하므로, tput이 증가하였다.

fragmentation [2 3 4], BER = 10^-4

```
FER =

    0.3297    0.2341    0.1813
```

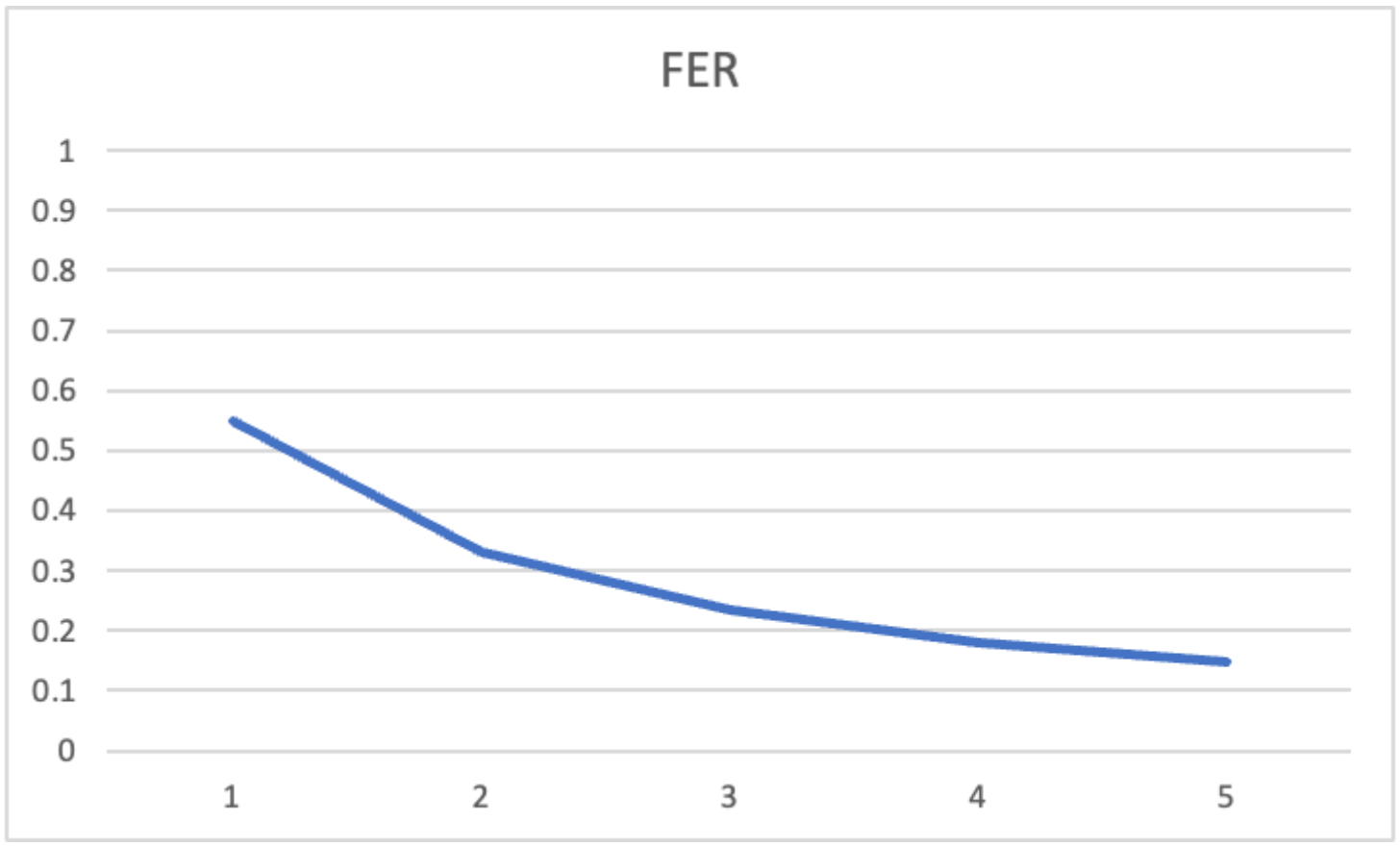
```
per_user_th =

    2.0516    2.2933    2.4847


total_th =

    6.8296

>>
```

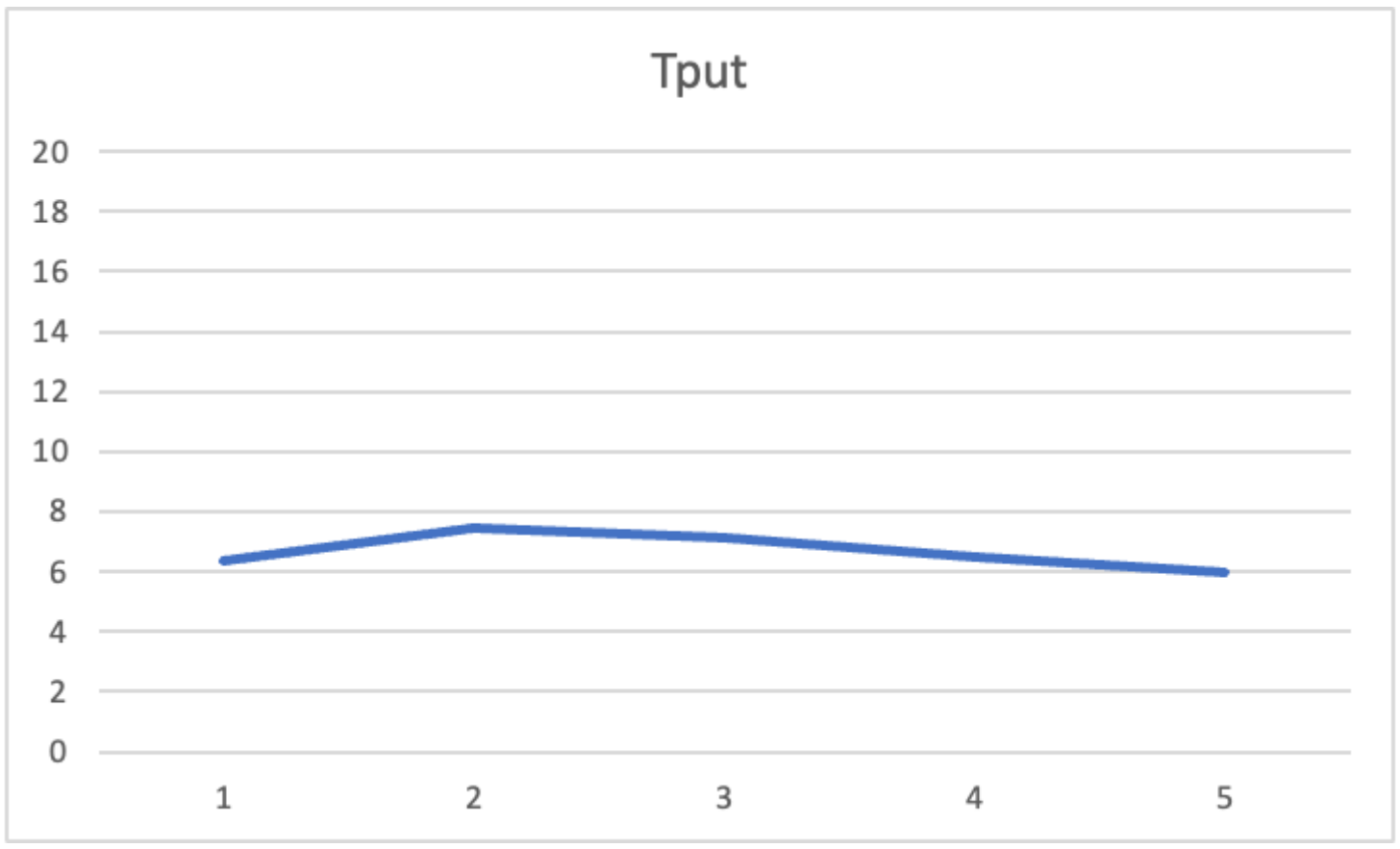
BER이 높아서 FER이 크다.

fragment가 큰 user3은 많이 쪼개서 전송하였으므로 FER이 가장 적어서 tput이 가장 크다.

실습 1.2 - Fragmentation이 FER과 throughput에 끼치는 영향

BER = 10^-4





프레임을 2개로 분할했을 때 tput이 가장 높게 나왔다.

fragmentation [1 1 1], BER = 10^-4

```
FRAG_ENABLE = 1;
A_MSDU_ENABLE = 0;
A_MPDU_ENABLE = 0;

BEB_ENABLE = 0;

N_frag = [1 1 1];

BER = 10^-4;
```

```
FER =

    0.5507    0.5507    0.5507


per_user_th =

    2.1387    2.1209    2.0898


total_th =

    6.3493

>>
```

fragmentation [2 2 2], BER = 10^-4

```
FER =

    0.3297    0.3297    0.3297


per_user_th =

    2.4809    2.4844    2.4831


total_th =

    7.4484

>>
```

fragmentation [3 3 3], BER = 10^-4

```
FER =

    0.2341    0.2341    0.2341


per_user_th =

    2.3879    2.3659    2.3641


total_th =

    7.1179

>>
```

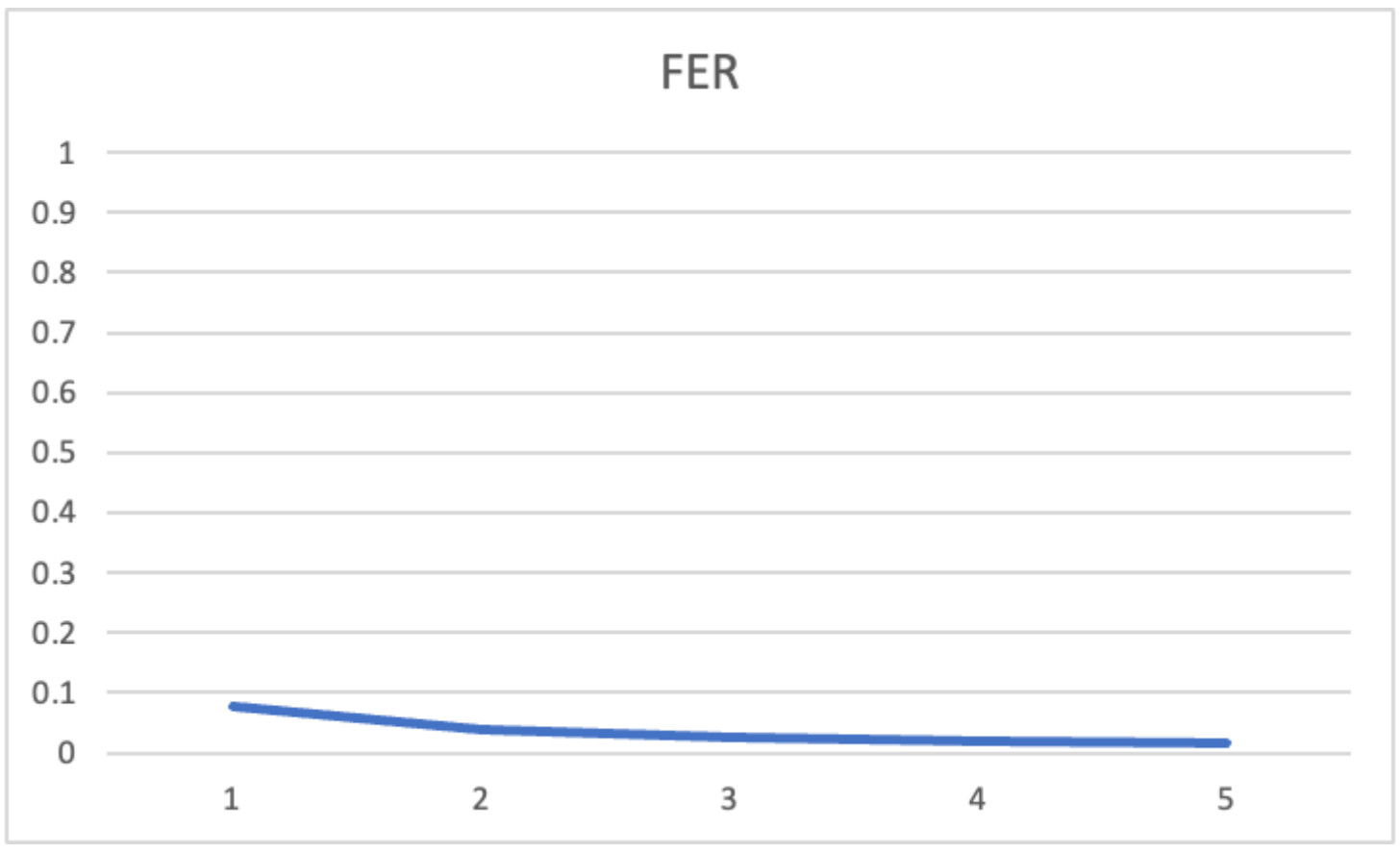
fragmentation [4 4 4], BER = 10^-4

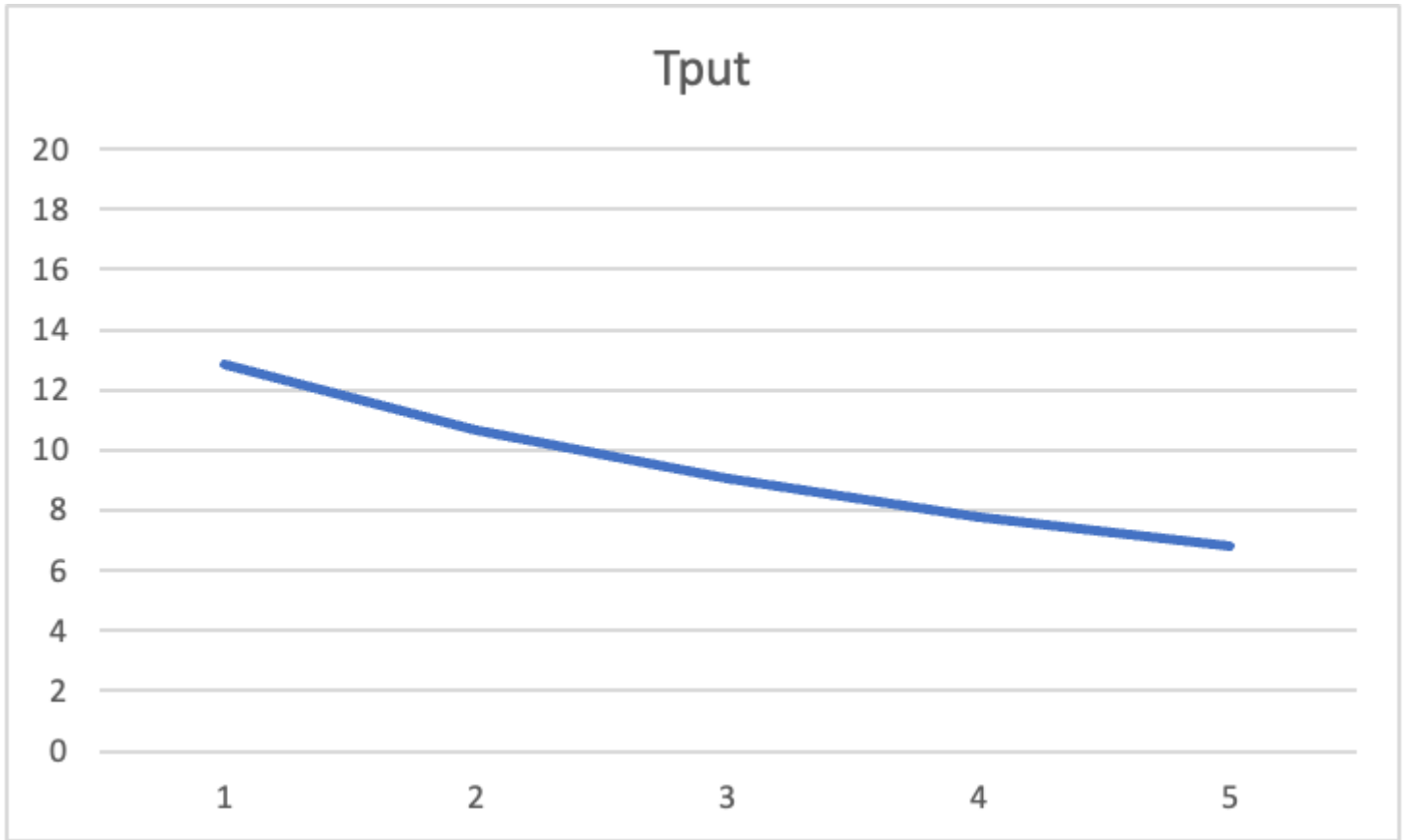
```
FER =

    0.1813    0.1813    0.1813


per_user_th =

    2.1420    2.1640    2.1858


total_th =

    6.4918

>>
```

fragmentation [5 5 5], BER = 10^-4

```
FER =

    0.1479    0.1479    0.1479


per_user_th =

    1.9732    2.0197    1.9735


total_th =

    5.9664

>>
```

BER = 10^-5





BER이 작을 때, fragmentation을 안 한 경우가 tput이 가장 컸다.

재전송이 잘 일어나지 않을 때는 fragmentation을 하면 오히려 오버헤드로 인해 tput이 감소한다.

fragmentation [1 1 1], BER = 10^-5

```
FRAG_ENABLE = 1;
A_MSDU_ENABLE = 0;
A_MPDU_ENABLE = 0;

BEB_ENABLE = 0;

N_frag = [1 1 1];

BER = 10^-5;
```

```
FER =

    0.0769    0.0769    0.0769


per_user_th =

    4.2978    4.2400    4.3262


total_th =

   12.8640

>>
```

fragmentation [2 2 2], BER = 10^-5

```
FER =

    0.0392    0.0392    0.0392


per_user_th =

    3.5418    3.5933    3.5427


total_th =
```

```
   10.6778

>>
```

fragmentation [3 3 3], BER = 10^-5

```
FER =

    0.0263    0.0263    0.0263


per_user_th =

    3.0284    3.0412    2.9781


total_th =

    9.0477

>>
```

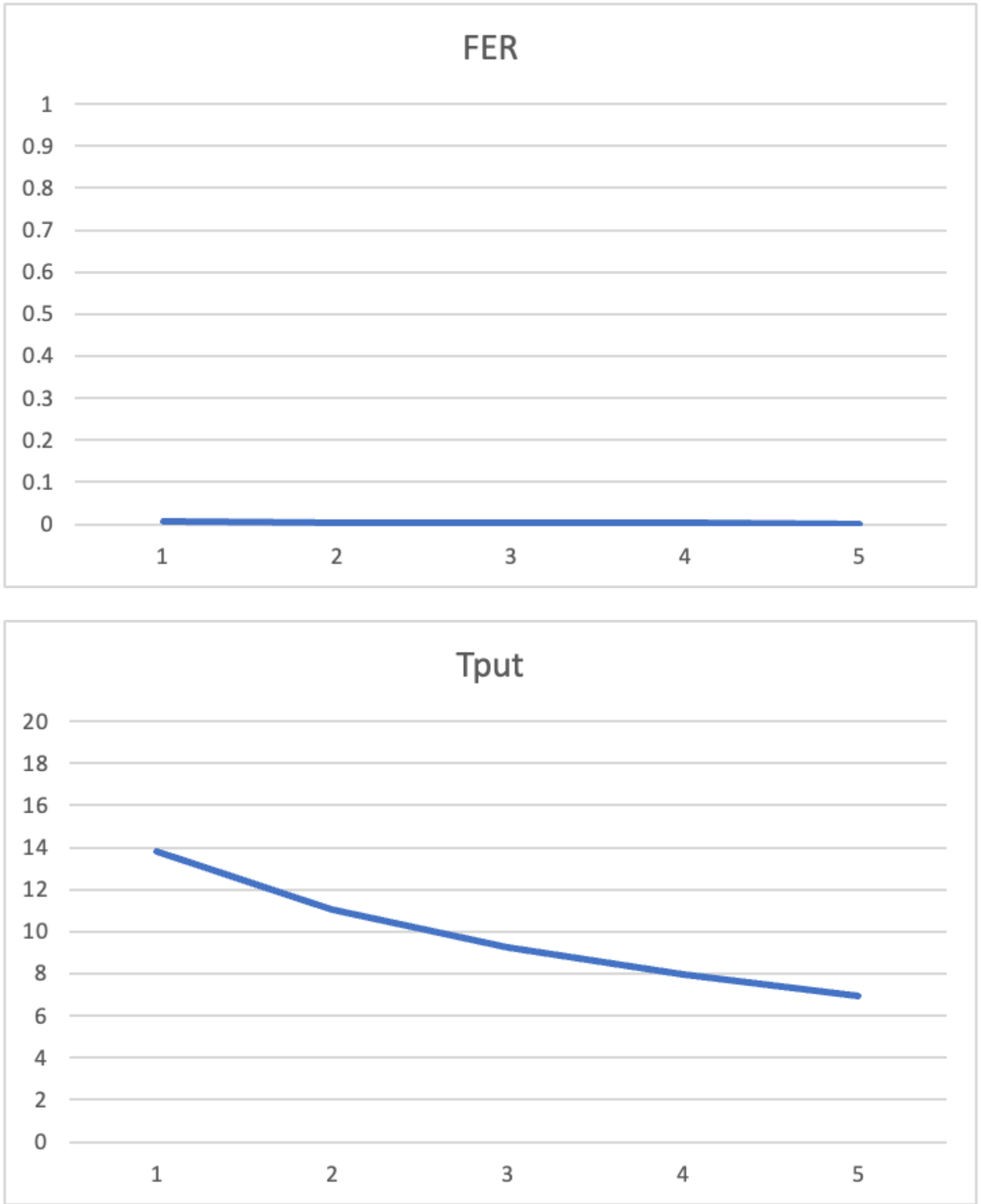
fragmentation [4 4 4], BER = 10^-5

```
FER =

    0.0198    0.0198    0.0198


per_user_th =

    2.5929    2.6202    2.5707


total_th =

    7.7838

>>
```

fragmentation [5 5 5], BER = 10^-5

```
FER =

    0.0159    0.0159    0.0159


per_user_th =

    2.2724    2.2976    2.2375


total_th =

    6.8075

>>
```

BER = 10^-6



BER이 매우 작을 때, fragmentation을 안 한 경우가 tput이 가장 컸다.

재전송이 거의 일어나지 않을 때는 fragmentation을 하면 오히려 오버헤드로 인해 tput이 감소한다.

fragmentation [1 1 1], BER = 10^-6

```
FRAG_ENABLE = 1;
A_MSDU_ENABLE = 0;
A_MPDU_ENABLE = 0;

BEB_ENABLE = 0;

N_frag = [1 1 1];

BER = 10^-6;
```

```
FER =

    0.0080    0.0080    0.0080


per_user_th =
```

```
    4.6116    4.5467    4.6293


total_th =

   13.7876

>>
```

fragmentation [2 2 2], BER = 10^-6

```
FER =

    0.0040    0.0040    0.0040


per_user_th =

    3.7187    3.6387    3.7253


total_th =

   11.0827

>>
```

fragmentation [3 3 3], BER = 10^-6

```
FER =

    0.0027    0.0027    0.0027


per_user_th =

    3.1132    3.0593    3.0601


total_th =

    9.2326

>>
```

fragmentation [4 4 4], BER = 10^-6

```
FER =

    0.0020    0.0020    0.0020


per_user_th =

    2.6600    2.6236    2.6578


total_th =

    7.9413

>>
```

fragmentation [5 5 5], BER = 10^-6

```
FER =

    0.0016    0.0016    0.0016


per_user_th =

    2.3465    2.2738    2.3214


total_th =

    6.9417

>>
```

실습 2.1 - A-MSDU 성능 평가

BER = 10^-6

A-MSDU(전체 재전송), aggregation [1 1 1], BER = 10^-6

```
FRAG_ENABLE = 0;
A_MSDU_ENABLE = 1;
A_MPDU_ENABLE = 0;

BEB_ENABLE = 0;

N_agg = [1 1 1];

BER = 10^-6;
```

```
FER =

    0.0080    0.0080    0.0080


per_user_th =

    4.6098    4.5476    4.6000


total_th =

   13.7573

>>
```

FER은 0.0080이다.

각 user의 tput은 약 4.6Mbps이다.

A-MSDU(전체 재전송), aggregation [2 2 2], BER = 10^-6

```
FER =

    0.0159    0.0159    0.0159


per_user_th =

    5.5129    5.4684    5.4916


total_th =

   16.4729
```

```
>>
```

FER은 0.0159이다.

각 user의 tput은 약 5.5Mbps이다.

2개씩 aggregation을 하였으므로 FER이 증가하였지만, 한 번에 전송하는 프레임의 사이즈가 커졌으므로 전체 tput이 증가하였다.

A-MSDU(전체 재전송), aggregation [4 4 4], BER = 10^-6

```
FER =

    0.0315    0.0315    0.0315


per_user_th =

    6.0622    6.0124    6.0942


total_th =

   18.1689

>>
```

FER은 0.0315이다.

각 user의 tput은 약 6.1Mbps이다.

4개씩 aggregation을 하였으므로 FER이 증가하였지만, 한 번에 전송하는 프레임의 사이즈가 커졌으므로 전체 tput이 증가하였다.

A-MSDU(전체 재전송), aggregation [2 4 8], BER = 10^-6

```
FER =

    0.0159    0.0315    0.0620


per_user_th =

    2.5351    4.8391   10.0053


total_th =

   17.3796

>>
```

BER이 매우 작아 재전송이 크게 발생하지 않는다.

각 user의 FER과 tput은 aggregation 크기와 같이 2:4:8 비율로 나타난다.

BER = 10^-5

A-MSDU(전체 재전송), aggregation [1 1 1], BER = 10^-5

```
FER =

    0.0769    0.0769    0.0769


per_user_th =

    4.2578    4.3120    4.2507
```

```
total_th =

   12.8204

>>
```

FER은 0.0769이다.

각 user의 tput은 약 4.3Mbps이다.

A-MSDU(전체 재전송), aggregation [2 2 2], BER = 10^-5

```
FER =

    0.1479    0.1479    0.1479


per_user_th =

    4.7253    4.8071    4.8480


total_th =

   14.3804

>>
```

FER은 0.1479이다.

각 user의 tput은 약 4.8Mbps이다.

2개씩 aggregation을 하였으므로 FER이 증가하였지만, 한 번에 전송하는 프레임의 사이즈가 커졌으므로 전체 tput이 증가하였다.

A-MSDU(전체 재전송), aggregation [4 4 4], BER = 10^-5

```
FER =

    0.2739    0.2739    0.2739


per_user_th =

    4.4124    4.5547    4.5333


total_th =

   13.5004

>>
```

FER은 0.2739이다.

각 user의 tput은 약 4.5Mbps이다.

2개씩 aggregation을 하였으므로 FER이 증가하였다.

전체 tput은 전체 재전송으로 인해 감소하였다.

A-MSDU(전체 재전송), aggregation [2 4 8], BER = 10^-5

```
FER =

    0.1479    0.2739    0.4727


per_user_th =

    2.1689    3.8293    5.3547
```

```
total_th =

   11.3529

>>
```

각 user의 FER은 aggregation 크기와 같이 2:4:8 비율로 나타난다.

전체 재전송으로 인해 aggregation이 클수록 MPDU에 비해 효율이 낮아진다.

실습 2.2 - A-MPDU 성능 평가

BER = 10^-6

A-MPDU(선택적 재전송), aggregation [1 1 1], BER = 10^-6

```
FRAG_ENABLE = 0;
A_MSDU_ENABLE = 0;
A_MPDU_ENABLE = 1;

BEB_ENABLE = 0;

N_agg = [1 1 1];

BER = 10^-6;
```

```
FER =

    0.0080    0.0080    0.0080


per_user_th =

    4.6133    4.6062    4.6000


total_th =

   13.8196

>>
```

FER은 발생하지 않았다.

각 user의 tput은 약 4.6Mbps이다.

전체 tput은 13.8196Mbps이다.

A-MPDU(선택적 재전송), aggregation [2 2 2], BER = 10^-6

```
FER =

    0.0080    0.0080    0.0080


per_user_th =

    5.4747    5.4711    5.4124


total_th =

   16.3582

>>
```

FER은 0.0080이다.

각 user의 tput은 약 5.5Mbps이다.

2개씩 aggregation을 하였으므로 FER이 증가하였지만, 한 번에 전송하는 프레임의 사이즈가 커졌으므로 전체 tput이 증가하였다.

A-MPDU(선택적 재전송), aggregation [4 4 4], BER = 10^-6

```
FER =

    0.0080    0.0080    0.0080


per_user_th =

    6.0089    6.0818    6.1031


total_th =

   18.1938

>>
```

FER은 0.0080이다.

각 user의 tput은 약 6.1Mbps이다.

A-MSDU와 달리 A-MPDU는 aggregation의 크기가 커짐에 따라 FER이 증가하지 않는다.

한 번에 전송하는 프레임의 사이즈가 커졌으므로 전체 tput이 증가하였다.

A-MPDU(선택적 재전송), aggregation [2 4 8], BER = 10^-6

```
FER =

    0.0080    0.0080    0.0080


per_user_th =

    2.4551    4.9991   10.0000


total_th =

   17.4542

>>
```

FER은 0.0080이다.

A-MSDU와 달리 A-MPDU는 aggregation의 크기가 커짐에 따라 FER이 증가하지 않는다.

선택적 재전송으로 인해 각 user의 tput은 aggregation 크기와 같이 2:4:8 비율로 나타난다.

BER = 10^-5

A-MPDU(선택적 재전송), aggregation [1 1 1], BER = 10^-5

```
FER =

    0.0769    0.0769    0.0769


per_user_th =

    4.2773    4.2498    4.3049
```

```
total_th =

   12.8320

>>
```

FER은 0.0769이다.

각 user의 tput은 약 4.3Mbps이다.

전체 tput은 12.8320Mbps이다.

A-MPDU(선택적 재전송), aggregation [2 2 2], BER = 10^-5

```
FER =

    0.0769    0.0769    0.0769


per_user_th =

    5.0498    5.1671    5.0658


total_th =

   15.2827

>>
```

FER은 0.0769이다.

각 user의 tput은 약 5.1Mbps이다.

A-MSDU와 달리 A-MPDU는 aggregation의 크기가 커짐에 따라 FER이 증가하지 않는다.

한 번에 전송하는 프레임의 사이즈가 커졌으므로 전체 tput이 증가하였다.

A-MPDU(선택적 재전송), aggregation [4 4 4], BER = 10^-5

```
FER =

    0.0769    0.0769    0.0769


per_user_th =

    5.6489    5.8107    5.4613


total_th =

   16.9209

>>
```

FER은 0.0769이다.

각 user의 tput은 약 5.6Mbps이다.

A-MSDU와 달리 A-MPDU는 aggregation의 크기가 커짐에 따라 FER이 증가하지 않는다.

한 번에 전송하는 프레임의 사이즈가 커졌으므로 전체 tput이 증가하였다.

A-MPDU(선택적 재전송), aggregation [2 4 8], BER = 10^-5

```
FER =

    0.0769    0.0769    0.0769
```

```
per_user_th =

    2.3067    4.5529    9.1902


total_th =

   16.0498

>>
```

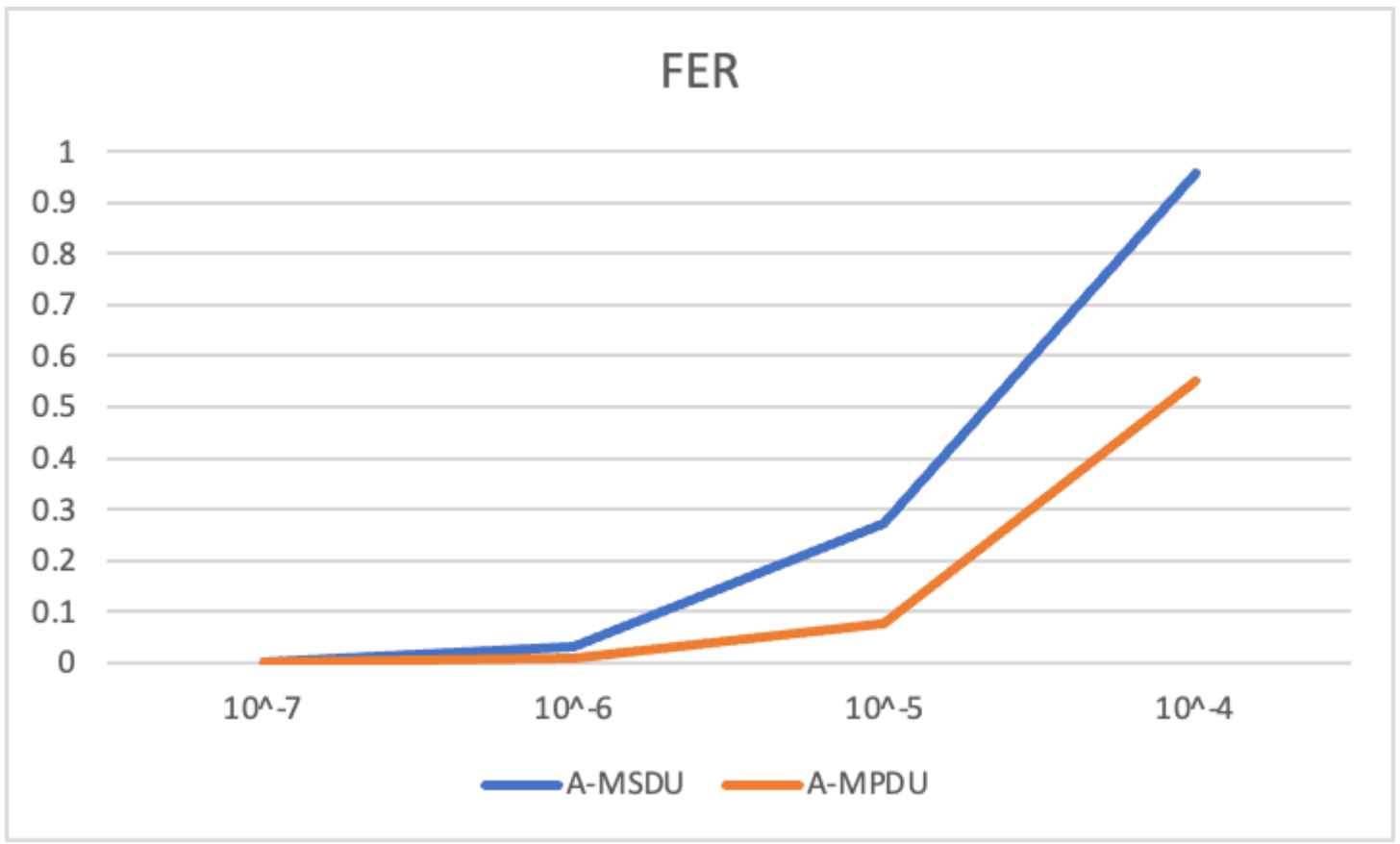
FER은 0.0769이다.

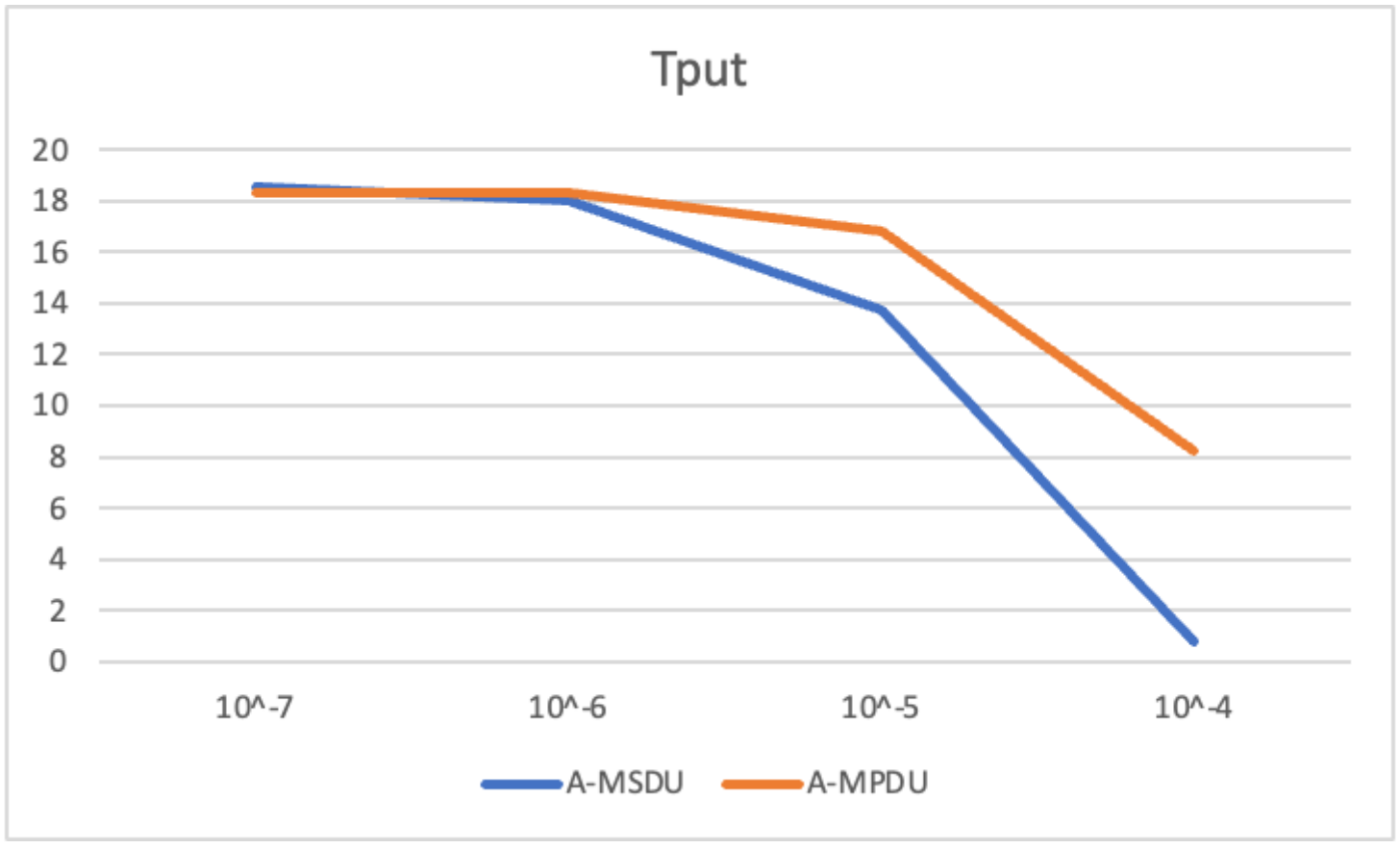
A-MSDU와 달리 A-MPDU는 aggregation의 크기가 커짐에 따라 FER이 증가하지 않는다.

선택적 재전송으로 인해 각 user의 tput은 aggregation 크기와 같이 2:4:8 비율로 나타난다.

실습 2.3 - BER 변화에 따른 A-MSDU와 A-MPDU 성능

N_agg = [4 4 4]





BER이 증가할수록 A-MPDU(선택적 재전송)가 A-MSDU(전체 재전송)보다 FE이 낮고 Tput이 높으므로 더 효율적이다.

A-MSDU와 A-MPDU는 프레임을 결합하기 때문에 각종 오버헤드가 감소한다는 장점이 있다. A-MSDU는 전송 실패 시 전체 재전송을 하므로 재전송해야 할 프레임의 크기가 크기 때문에 전송 소요 시간이 길어진다는 단점이 있다.

BER = 10^-7

A-MSDU

```
FER =

    0.0032    0.0032    0.0032


per_user_th =

    6.2756    6.1796    6.0693


total_th =

   18.5244

>>
```

A-MPDU

```
FER =

   1.0e-03 *

    0.7997    0.7997    0.7997


per_user_th =

    6.1884    6.0071    6.1102


total_th =

   18.3058

>>
```

BER = 10^-6

A-MSDU

```
FER =

    0.0315    0.0315    0.0315


per_user_th =

    5.9769    6.0942    5.9484


total_th =

   18.0196

>>
```

A-MPDU

```
FER =

    0.0080    0.0080    0.0080


per_user_th =

    6.0231    6.1458    6.1164


total_th =

   18.2853

>>
```

BER = 10^-5

A-MSDU

```
FER =

    0.2739    0.2739    0.2739


per_user_th =

    4.5333    4.6222    4.6080


total_th =

   13.7636

>>
```

A-MPDU

```
FER =

    0.0769    0.0769    0.0769


per_user_th =

    5.6551    5.5520    5.5724


total_th =

   16.7796

>>
```

BER = 10^-4

A-MSDU

```
FER =

    0.9592    0.9592    0.9592
```

```
per_user_th =

    0.2489    0.2276    0.2951


total_th =

    0.7716

>>
```

A-MPDU

```
FER =

    0.5507    0.5507    0.5507


per_user_th =

    2.8009    2.6418    2.7893


total_th =

    8.2320

>>
```