COE 4DN4 ADVANCED INTERNET COMMUNICATIONS

ASSIGNMENT #2 Weighted Fair Queueing

Jingnan Chen (1073196), Wei Zhang (0951321)
chenj39@mcmaster.ca, zhangw32@mcmaster.ca

Contents

Part (A)

I used 4 flows with fixed packet rates, fixed packet sizes and equal weights.

In the main function:

```
NUM_FLOWS  = 4;          % 4 flows
NUM_PKTS     = 100;
LINK_RATE    = 1000;  % 1 kbits per second

% define some global data-structures, visible in the functions
global NUM_PKTS;
global PACKET_ATIMES;
global PACKET_BITS;
global LINK_RATE;

%-----------------------------------------------------------------------------
---
% define several matrices, to store data on packets and flows
% store packet arrival times, bits per packet,
%----------------------------------------------------------
PACKET_ATIMES   = zeros(NUM_FLOWS, NUM_PKTS);
PACKET_BITS      = zeros(NUM_FLOWS, NUM_PKTS);

FLOW_MEAN_RATES     = zeros(1, NUM_FLOWS);
FLOW_MEAN_BITS      = zeros(1, NUM_FLOWS);
FLOW_WEIGHTS    = zeros(1, NUM_FLOWS);

FLOW_MEAN_RATES     = [100   200  55 30];
FLOW_MEAN_BITS      = [20   15  40   60];
FLOW_WEIGHTS        = [ 1   1   1     1];
```

In the generate_packets.m, I updated the code as follows:

```
% new function to Generate Packet arrivals for one flow
function [ATIMES, BITS] = generate_packets(Num_Pkts, mean_rate, mean_bits)

ATIMES  = zeros(1,Num_Pkts);
BITS     = zeros(1,Num_Pkts);

current_time = 0;

    for j=1:Num_Pkts
        mean_time    = 1/mean_rate;

        ATIMES(1,j)  = current_time;
        BITS(1,j)    =  mean_bits;  % fiexd packet size

        current_time = current_time + mean_time;    %fix arrival time

    end  % for j=…
end  % end function
```

Flow 1 has a fixed rate = 100 pkts/sec, and a fixed packet size of 20 bits, fixed rate = 2000 bps
Flow 2 has a fixed rate = 200 pkts/sec, and a fixed packet size of 15 bits, fixed rate = 3000 bps
Flow 3 has a fixed rate = 55 pkts/sec, and a fixed packet size of 40 bits, fixed rate = 2200 bps
Flow 4 has a fixed rate = 30 pkts/sec, and a fixed packet size of 60 bits, fixed rate = 1800 bps

Part (B)

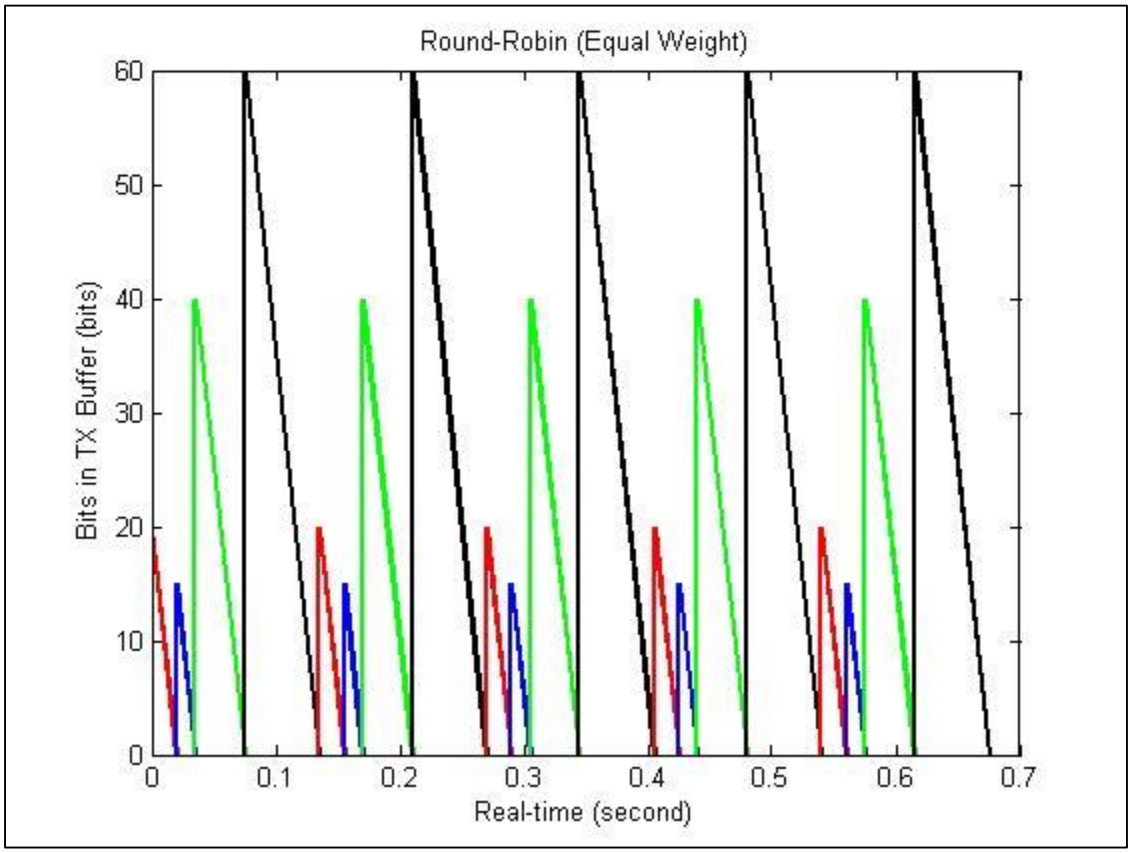

Figure 1: WRR Server with Equal Weight Flows

The WRR service is unfair as you can see in the Figure 1. The packet transmission is in the following order: packets from Flow 1 (red), packets from Flow 2 (blue), packets from Flow 3 (green), and packets from Flow 4 (black). Since the packets from different flows with different sizes, the flow with largest packet size gets most service (i.e. Flow 4), while the flow with smallest packet size gets least service (i.e. Flow 2).

If we set up the flow weight as follows:

```
FLOW_WEIGHTS          = [ 2    2    1      1];
```

We can have Figure 2:

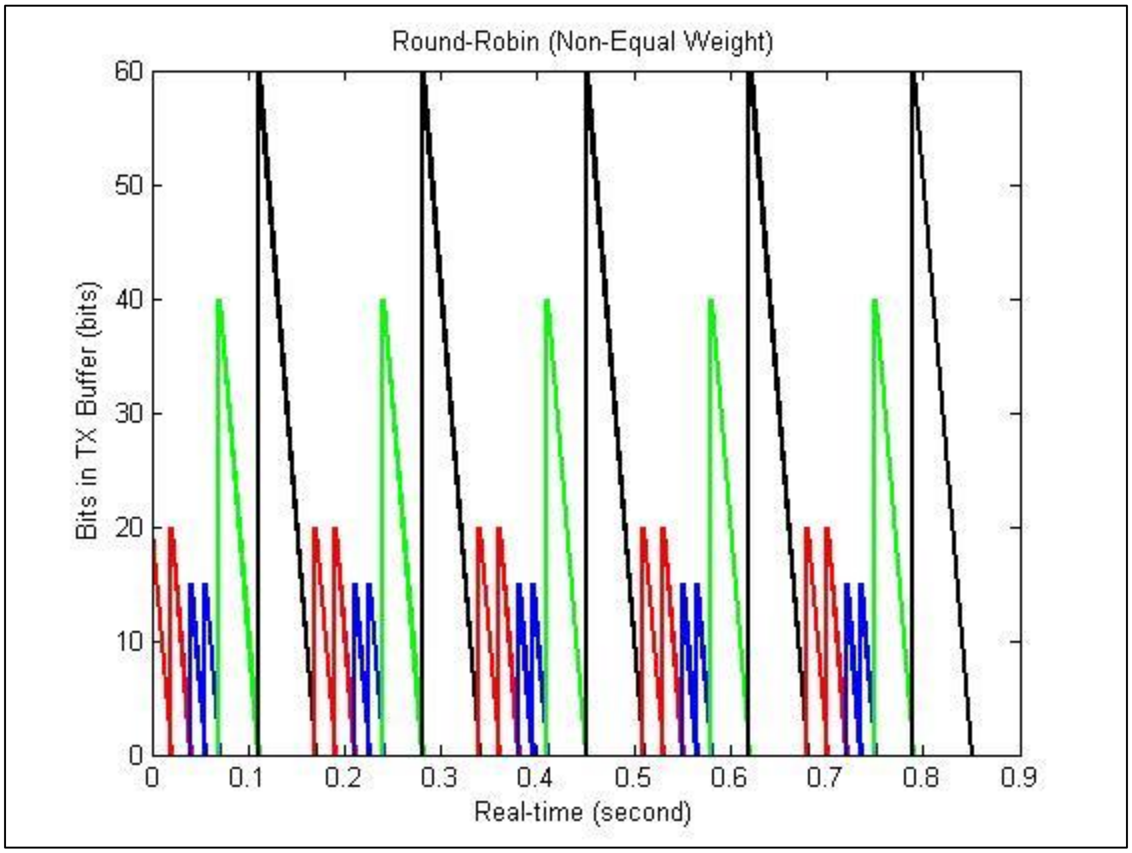

Figure 2: WRR Sever with Non-Equal Weight Flows

Part (C)

With all weights equal to 1, after 5 rounds, each flow transmits 5 packets (See Figure 1). Now we can calculate how many bits are transmitted for each flow:

Red Flow:	5 packets * 20 bits per packet = 100 bits
Blue Flow:	5 packets * 15 bits per packet = 75 bits
Green Flow:	5 packets * 40 bits per packet = 200 bits
Black Flow:	5 packets * 60 bits per packet = 300 bits

As you can see the WRR service is not fair. The WRR service will be more unfair if we increase the packet size of one of the flows. We can quantify the unfairness. We can record how many bits served for each flow per round, and we can report how many bits have been served for each flow. I have added the code in the given source code to record the total transmitted bits for each flow when we have a packet arrival to transmit:

```
FLOW(flow).transmitted_bits = FLOW(flow).transmitted_bits + bits;
```

And the result is as follows:

```
Flow 1 transmitted 100 bits.

Flow 2 transmitted 75 bits.

Flow 3 transmitted 200 bits.

Flow 4 transmitted 300 bits.
```

For non-equal weight flows, we have the following data:

```
Flow 1 transmitted 200 bits.

Flow 2 transmitted 150 bits.

Flow 3 transmitted 200 bits.

Flow 4 transmits 300 bits.
```

Part (D)

Flow Chart for WFQ Server

```
PACKET_ATIMES    = zeros(NUM_FLOWS, NUM_PKTS);
PACKET_BITS       = zeros(NUM_FLOWS, NUM_PKTS);

FLOW_MEAN_RATES      = zeros(1, NUM_FLOWS);
FLOW_MEAN_BITS       = zeros(1, NUM_FLOWS);
FLOW_WEIGHTS         = zeros(1, NUM_FLOWS);
VFT                  = (inf)*ones(3,NUM_PKTS);

FLOW_MEAN_RATES      = [100   200  55 30];
FLOW_MEAN_BITS       = [20   15  40   60];
FLOW_WEIGHTS         = [ 1   1   1     1];
```

Comment 1: we have to initialize the packet sizes, packet arrival rates and weights for four flows, and set the virtual finishing time (VFT) to infinity at the beginning.

```
      % generate packet arrivals for 4 flows.
    [atimes, bits] = generate_packets(NUM_PKTS, flow_rate, flow_bits);
    FLOW(flow).Queue = [];        % Queue
    FLOW(flow).QSIZE = 0;         % Queue Size
    FLOW(flow).total_bits = 0;    % Total number of bits transmitted
```

Comment 2: then we generate 100 packet arrivals for each flow and create a FLOW structure to store the queue and queue size and total number of bits that have been transmitted.

```
TX_start   = 0; % transmission start time
TX_end     = 0; % transmission end time
TX_busy    = 0; % transmission status
TX_flow    = 0; % flow being transmitted
TX_pkt_num = 0; % packet number being transmitted
```

Comment: We have these variables for the transmission queue. The transmission queue is busy when TX_busy equals to 1. TX_start records start time and TX_end records records the finishing time of the transmission. TX_flow and TX_pkt_num record the packet information.

```
round = 1;
% implement using realtime
for real_time = 0:3000
```

Comment 1: we implement the WFQ server using real time.

```
    for flow = 1: NUM_FLOWS
        pkt_num = check_for_arrivals(flow,real_time/1000);
```

In check_for_arrival.m:

```
    smallest_VFT = inf;
    for flow = 1:NUM_FLOWS
        for pkt_num = FLOW(flow).Queue
            if (VFT(flow, pkt_num) < smallest_VFT) && (VFT(flow,
pkt_num) >= real_time)
                smallest_VFT = VFT(flow, pkt_num);
                TX_flow = flow;
                TX_pkt_num = pkt_num;
            end
        end
    end
```

Comment 2: In the check_for_arrival.m function, we find the next packet with the smallest VFT.

```
      if (pkt_num > 0)
            bits = PACKET_BITS(flow, pkt_num);
            weight = FLOW_WEIGHTS(flow);
            if(FLOW(flow).QSIZE == 0)
                FLOW(flow).Queue = pkt_num;
                FLOW(flow).QSIZE = 1;
                VFT(flow,pkt_num) = round + bits/weight;
            else
                FLOW(flow).Queue = [FLOW(flow).Queue, pkt_num];
                FLOW(flow).QSIZE = FLOW(flow).QSIZE + 1;
                VFT(flow,pkt_num) = VFT(flow,pkt_num - 1) + bits/weight;
            end
            PACKET_ATIMES(flow, pkt_num) = inf;
        end
    end
```

Comment 3: first of all, we check for next arrived packet based on the current time. If we can get a packet, we check if the queue of corresponding flow is empty. If so, we can update the VFT base on the round + bits/weight; if not, we update the VFT base on the previous packet’s VFT + bits/weight.

```
if (TX_busy ==0)
        [TX_flow, TX_pkt_num] = find_smallest_VFT(real_time/1000);
        if (TX_pkt_num > 0)
            bits = PACKET_BITS(TX_flow, TX_pkt_num);
            TX_busy    = 1;
            TX_start   = real_time/1000;
            TX_end     = TX_start + bits/LINK_RATE;
            plot_transmission_2015(TX_flow, TX_pkt_num, TX_start,
bits/LINK_RATE);
            FLOW(TX_flow).total_bits = FLOW(TX_flow).total_bits + bits;
        end
```

Comment 1: if the transmission queue is idle, we find the next packet with smallest VFT based on the real time. Then we calculate the TX_end and set TX_busy to 1. Also accumulate the total bits for the corresponding flows.

```
    elseif (TX_busy ==1)&&(TX_end<=real_time/1000)
        VFT(TX_flow, TX_pkt_num) = inf;
        FLOW(TX_flow).QSIZE = FLOW(TX_flow).QSIZE - 1;
        TX_busy = 0;
        TX_end = 0;
        TX_flow = 0;
        TX_pkt_num = 0;
    end
```

Comment 2: If the queue is busy, we don't do anything until the packet is done transmitting. Then we reset the transmission queue to idle state.

```
    % increatment the 'round'
    delta_t = 0;
    for flow = 1:NUM_FLOWS
        if(FLOW(flow).QSIZE > 0)
            delta_t = delta_t + FLOW_WEIGHTS(flow);
        end
    end
    round = round + 1/delta_t;
```

Comment 3: since the main loop is using real time, we have to update the 'round' based on the real time. We check the queue size for each flow. If it is greater than 0, then increase the delta time based on the weight of that flow. We update the round with calculated delta time.

Part (E)

Using the same traffic pattern as in Part (B), so we have 4 flows with equal weight:

Flow 1 has a fixed rate = 100 pkts/sec, and a fixed packet size of 20 bits, fixed rate = 2000 bps
Flow 2 has a fixed rate = 200 pkts/sec, and a fixed packet size of 15 bits, fixed rate = 3000 bps
Flow 3 has a fixed rate = 55 pkts/sec, and a fixed packet size of 40 bits, fixed rate = 2200 bps
Flow 4 has a fixed rate = 30 pkts/sec, and a fixed packet size of 60 bits, fixed rate = 1800 bps

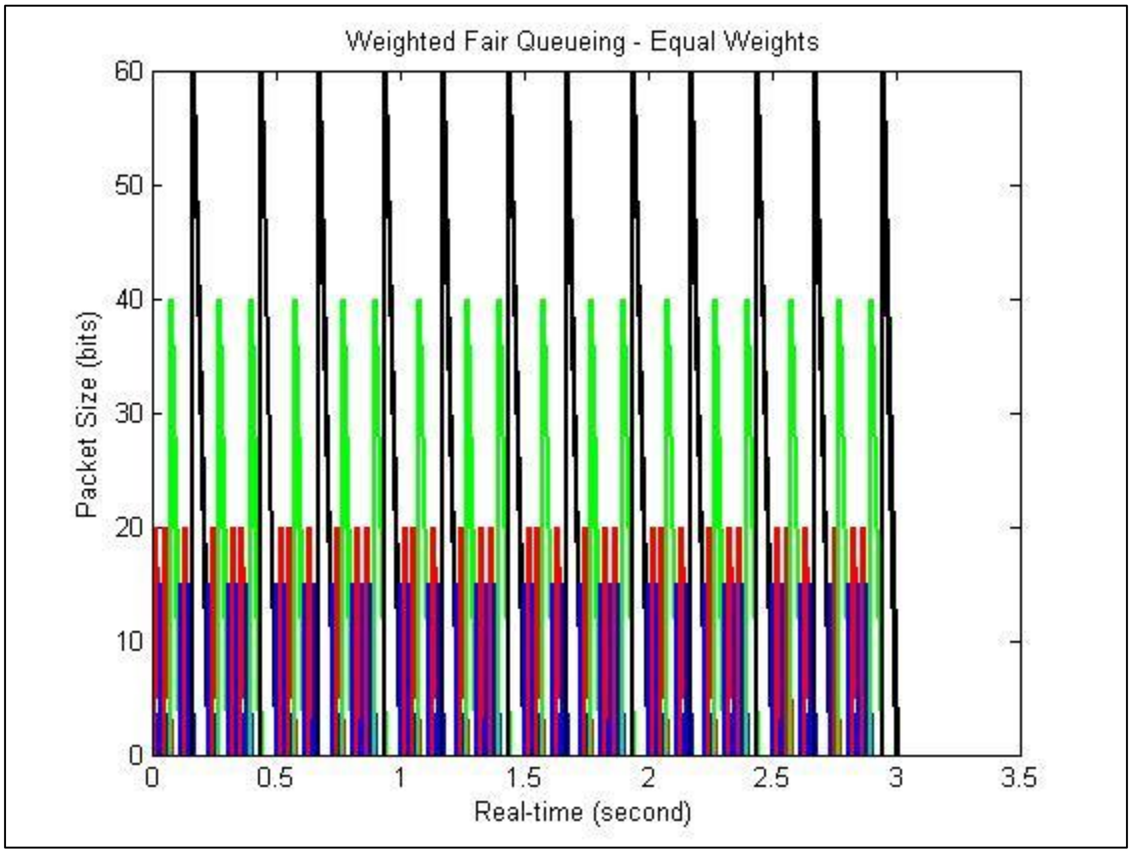

Figure 3: GPS/WFQ Sever with Equal Weight FLows

Part (F)

As you can see the GPS/WFQ server is fair. Fairness can be measured by how many bits are being transmitted from each flow. If any flow has a relatively lower transmitting rate, then it would indicate unfairness.

As a proof of fairness in GPS/WFQ, We recorded how many bits served for each flow per round, and the results for the simulation of equal weight:

```
Flow 1 transmitted 720 bits.

Flow 2 transmitted 720 bits.

Flow 3 transmitted 720 bits.

Flow 4 transmitted 720 bits.
```

Compared with the Part (B), we can see all the flows has been served with same amount of bits.

If we set up the flow weight as follows:

```
FLOW_WEIGHTS         = [ 2   2   1     1];
```

We can have the following data and Figure 4:

```
Flow 1 transmitted 960 bits.
Flow 2 transmitted 960 bits.

Flow 3 transmitted 480 bits.

Flow 4 transmitted 480 bits.
```

Flow 1 and Flow 2 get 2 times services as Flow 3 and Flow 4, since Flow 1 and Flow 2 have a larger weight, which is 2.

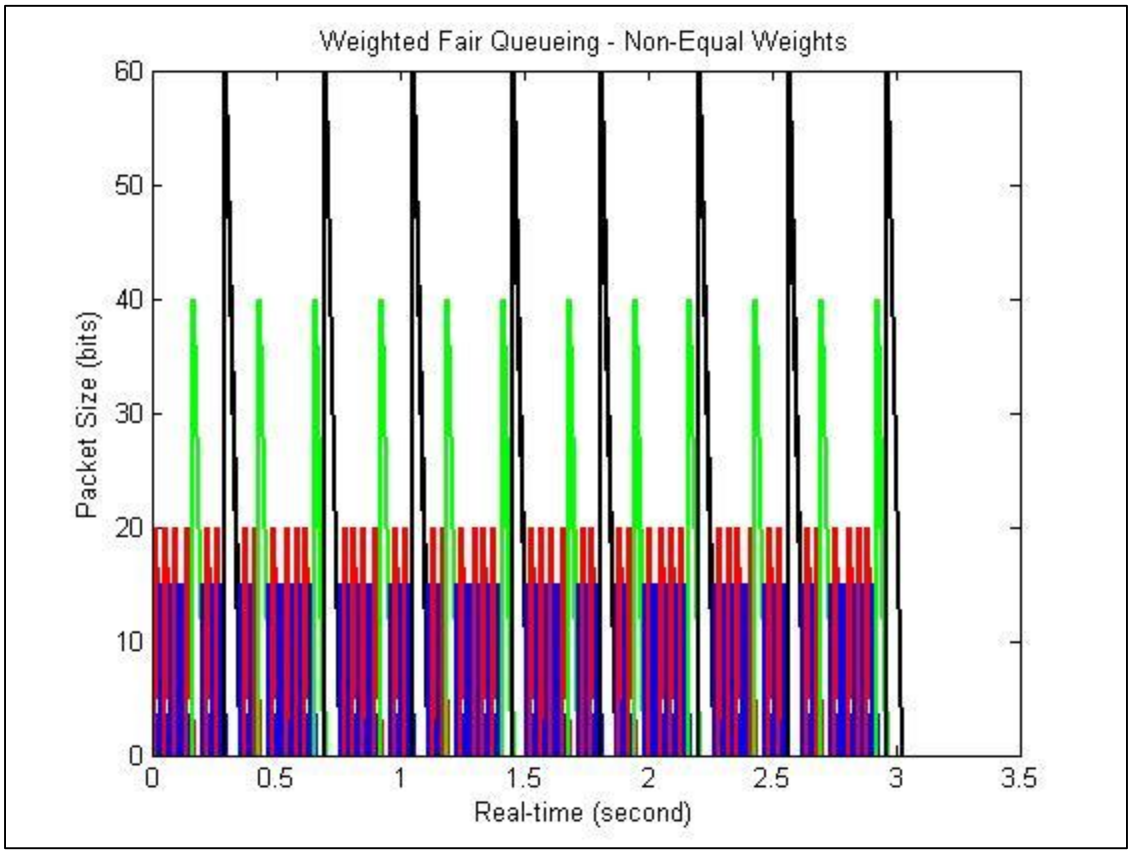

Figure 4: GPS/WFQ Sever with Non-Equal Weight Flows