CMPT 434 Assignment 3
Winter 2019
Due Date: Monday March 18th
rdm659 11165820

Part B

(1) (2 marks) Give the parameters of a token bucket that limits the long term average rate to 40 Mbps (i.e., 40e6 bits per second) and the maximum duration of back-to-back packet transmissions to 500 microseconds, assuming a link of capacity of 1 Gbps.

Parameters of a token bucket are B, the capacity of the bucket, and R, the rate at which new tokens are 'added' or 'dispensed'.

If we want to limit the long term average rate to 40 Mbps, then we should set $R = 40Mbps$ to prevent the sender from ejecting more than 40 Mb a second.

If we want to limit the maximum duration of back to back packet transmission to 500 Microseconds, then we want our bucket to run out of tokens after 500 microseconds of consuming them at 1 Gbps. We can use the following formula to calculate B:

$$B + RS = MS$$

where M is $1\text{e}9\text{bit}\,\text{s}^{-1}$, R is $40\text{e}6\text{bit}\,\text{s}^{-1}$, and S is 5.0e−4s. To solve for B, the capacity of the bucket, we can re-write our formula as $B = MS - RS \iff B = S(M - R)$. This gives:

$$B = 5.0\text{e}{-4}\text{s}(1.0\text{e}9\text{bit}\,\text{s}^{-1} - 4.0\text{e}7\text{bit}\,\text{s}^{-1})$$
$$B = 5.0\text{e}{-4}\text{s} * 9.6\text{e}8\text{bit}\,\text{s}^{-1}$$
$$B = 4.8\text{e}5\text{bit}$$

Here, we see that a token bucket with parameter of $R = 40\text{Mbit}\,\text{s}^{-1}$ and $B = 480000\text{bit}$ achieves this.

(2) (8 marks) For each of the following destination IP addresses, state whether an incoming IP datagram would be forwarded on an outgoing interface using a destination link layer address belonging to a next hop router, or a link layer address belonging to the IP datagram's destination host. In the former case, state which next hop router. Make sure you justify your answers.

```
Entry 1:
address prefix: 10.11.43.128/26
    address: 00001010 00001011 00011011 10000000
    mask:    11111111 11111111 11111111 11000000
    prefix:  00001010 00001011 00011011 10000000
address of next hop router: 10.11.32.7
interface: A
Entry 2:
address prefix: 0.0.0.0/0
address of next hop router: 10.11.0.73
interface: B
Entry 3:
address prefix: 10.11.42.0/23
address of next hop router: 10.11.47.127
    address: 00001010 00001011 00011010 00000000
    mask:    11111111 11111111 11111110 00000000
    prefix:  00001010 00001011 00011010 00000000
interface: A
Entry 4:
address prefix: 10.11.32.0/20
    address: 00001010 00001011 00010000 00000000
    mask:    11111111 11111111 11110000 00000000
    prefix:  00001010 00001011 00010000 00000000
interface: A
```

(a) 10.11.43.127
00001010 00001011 00011011 01111111
This address does not match the prefix of Entry 1. It does match with Entry 3 and 4. However, it has a longer match Entry 3 than with entry 4. Therefore it will be forwarded to the next hop router at 10.11.47.127 over interface A.

(b) 10.11.43.160
00001010 00001011 00011011 10100000
This address matches the prefix of Entry 1. It will be forwarded to the next hop router at 10.11.32.7 over interface A.

(c) 10.11.49.123
00001010 00001011 00011011 01111011
This doesn't match Entry 1. It has its longest prefix match with Entry 3, though it matches 4 as well. It will be forwarded to the next hop router at 10.11.47.127 over interface A.

(d) 10.11.44.222
00001010 00001011 00011011 11011110
This address has its longest prefix match with entry 3. It will be forwarded to the next hop router at 10.11.47.127 over interface A.

(3) (2 marks) A TCP sender's value of SRTT is 125 milliseconds, but then a routing change occurs, after which all measured RTTs are 80 milliseconds. How many measurements of the new RTT are required before SRTT drops below 100 milliseconds? Assume that a weight of 0.125 is used for the new sample, and a weight of 0.875 for the old value, when updating SRTT. Make sure to show how you got your answer.

The formula for calculating the SRTT based on rount-trip times is:

$$SRTT = \alpha SRTT + (1 - \alpha) * R$$

If we were to script a solution to this...

```
ALPHA = 7/8
INITIAL_SRTT = 125

def new_srtt(old_srtt, alpha, r):
    return alpha*old_srtt + (1-alpha)*r

i = 0
srtt = INITIAL_SRTT
while(srtt > 100):
    print("Iteration " + str(i) + ": " + str(srtt))
    srtt = new_srtt(srtt, ALPHA, 80)
    i += 1

print("Iteration " + str(i) + ": " + str(srtt))
```

... and then run it, we would get the following output, showing that it would take 8 iterations to bring the value of SRTT below 80:

Iteration 0: 125
Iteration 1: 119.375
Iteration 2: 114.453125
Iteration 3: 110.146484375
Iteration 4: 106.378173828125
Iteration 5: 103.08090209960938
Iteration 6: 100.1957893371582
Iteration 7: 97.67131567001343

(4) (2 marks) Suppose that a TCP connection's RTT is 125 milliseconds except for every N'th RTT, which is 500 milliseconds. Note that if N is sufficiently large, the 500 millisecond RTT will cause a timeout, since that RTT value will be highly "unexpected" (assume here that RTOmin is less than 500 milliseconds), while if N is sufficiently small, it won't cause a timeout. What is the largest N for which the 500 millisecond RTT won't cause a timeout? (Consider only the behavior after there have been many measured RTTs following this pattern, and the initial values chosen for SRTT and RTTVAR no longer have any significant impact.) Assume that the same weights are used as in question 3 when updating SRTT, and weights of 0.25 (new sample) and 0.75 (old value) when updating RTTVAR. Make sure to show how you got your answer.

The formula for calculating the $SRTT$ based on rount-trip times is:

$$SRTT = \alpha SRTT + (1 - \alpha) * R$$

For calculating $RTTVAR$, we use:

$$RTTVAR = \beta RTTVAR + (1 - \beta) * |SRTT - R|$$

Then, we set the retransmission time RTO as

$$RTO = SRTT + 4 * RTTVAR$$

A timeout will be caused if the measured RTT is greater than the RTO.

If we were to script a solution to this, part of it would look like...

```
# Run simulation for increasing values of N
for n in range(1, 20):
    print("***** N = " + str(n) + " *****")
    print('{:8}{:14}{:14}{:14}{:14}'
        .format("N", "RTT", "SRTT", "RTT_VAR", "RTO"))
    for i in range(WARMUP + TEST):
        rtt = get_rtt(i, n, RTT, N_RTT)
        # Record the value of N when a timeout occurs
        # But only after warmup iterations have expired
        if (i >= WARMUP and rtt > rto):
            print(" ***** !!!TIMEOUT OCCURRED!!! (N = "
                + str(n) + ") ***** ")
        srtt = new_srtt(srtt, ALPHA, rtt)
        rttvar = new_rttvar(rttvar, srtt, BETA, rtt)
        rto = new_rto(srtt, rttvar)
        print('{:<8d}{:<14f}{:<14f}{:<14f}{:<14f}'
            .format(i, rtt, srtt, rttvar, rto))
```

... and then run it, we would find that values of N greater than 5 cause timeouts to fire. Here is a sample of the output:

```
...
***** N = 6 *****
...
N       RTT           SRTT          RTT_VAR       RTO
95      125.000000    168.618326    70.823094     451.910702
96      500.000000    210.041035    125.607062    712.469282
97      125.000000    199.410906    112.808023    650.642997
98      125.000000    190.109543    100.883403    593.643154
99      125.000000    181.970850    89.905264     541.591908
100     125.000000    174.849494    79.891322     494.414781
101     125.000000    168.618307    70.823068     451.910579
 ***** !!!TIMEOUT OCCURRED!!! (N = 6) *****
102     500.000000    210.041018    125.607046    712.469204
...
```

Here we can see that on iteration 102 an RTT of 500ms is measured, but the RTO we had set was only 451ms. This would cause a retransmission timeout to fire.

(5) (14 marks) Suppose that a sender using TCP Reno is observed to have the following congestion window sizes, as measured in segments, during each transmission "round" spent in slow start or additive increase mode...

(a) Identify the transmission rounds when TCP is in slow start mode.
rounds 1-7, rounds 22-26

(b) After the 13th transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?
By a triple duplicate ACK, as the *cwnd* was cut only in half.

(c) After the 20th transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?
After round 21, segment loss is detected by a timeout. This can be determined by observing that *cwnd* has fallen to 1 in round 22.

(d) What is the initial value of the slow start threshold ssthresh at the first transmission round?
This value must be 64, as that is the value of *cwnd* after which additive increase begins at round 8.

(e) What is the value of ssthresh at the 18th transmission round?
Assuming *ssthresh* is updated to be equal to the last highest value of *cwnd* acheived before packet loss, *ssthresh* must be 70.

(f) What is the value of ssthresh at the 24th transmission round?
ssthresh was likely updated to a value of 42 after packet loss was detected in round 21 with *cwnd* equal to 42.

(g) Assuming that a segment loss is detected after the 26th round by the receipt of a triple duplicate ACK, what will be the new value of ssthresh and cwnd?
ssthresh will be set to 16, the last highest value of *cwnd* before packet loss, and *cwnd* will be set to 8, half the previous value of *cwnd*.