

## 電腦網路導論 Assignment 2: Go-Back-N Report

外文五 楊晴雯 b06102020

\* Due to ignorance of the time unit, “time unit” is used instead in this report.

\* A is sender; B is receiver.

\* Usage: `gcc hw2.c -o hw2 -std=gnu11`

### 1. Go-Back-N

(a) `./{student-id} 10 0 0 10 2`

Although Go-Back-N is said to “keep one timer for the sender (A) side,” the implementation is, however, keeping a timer for each packet sent (by A), but the timeout event is considered a real timeout when the timeout packet is the base packet (if (index == A.base)).

Observe in the log that when base is 7, pkt 2 times out but no retransmission occurs.

```
EVENT time: 64.682739, type: 0, timerinterrupt entity: 0
→ MESSAGE: A timeout (index: 2)

EVENT time: 68.891106, type: 2, fromlayer3 entity: 0
→ MESSAGE: A receive (seq: 0 ack: 7)
..... START TIMER: starting timer for index: 8 at 68.891106,
..... will timeout at 100.227356.
```

(b) `./{student-id} 20 0.1 0 10 2 (loss prob = 0.1)`

Event: In this simulation, pkt 5 is lost twice during transmission. Below is the first loss.

```
EVENT time: 32.231213, type: 1, fromlayer5 entity: 0
→ MESSAGE: A output (eeeeeeeeeeeeeeeeeeee)
→ MESSAGE: A send (seq: 5 ack: 0)
..... TOLAYER3: packet being lost
```

Sender (A) is unaware of the loss and keeps sending next packet (pkt 6) since the N = 8 window size is not yet full.

```
EVENT time: 51.620754, type: 1, fromlayer5 entity: 0
→ MESSAGE: A output (ffffffffffffffffffff)
→ MESSAGE: A send (seq: 6 ack: 0)
```

Receiver (B) receives packet 6, yet it is expecting pkt 5, so it sends the highest-in-order ack (ack 4). For all the subsequent packets that are not in the right order, it answers ack4.

```
EVENT time: 57.572548, type: 2, fromlayer3 entity: 1
→ MESSAGE: B receive (seq: 6 ack: 0)
→ MESSAGE: B send (seq: 0 ack: 4)
```

Sender (A): **first timeout on pkt 5**; timer starts on **pkt 5** again and packets are retransmitted (from **pkt 5** to the last sent packet **pkt9**). But **pkt 5** is lost again.

```
EVENT time: 92.183914, type: 0, timerinterrupt entity: 0
→ MESSAGE: A timeout (index: 5)
→ START TIMER: starting timer for index: 5 at 92.183914,
will timeout at 131.813599. MESSAGE: A send (seq: 5 ack: 0)
→ TOLAYER3: packet being lost
→ MESSAGE: A send (seq: 6 ack: 0)
→ MESSAGE: A send (seq: 7 ack: 0)
→ TOLAYER3: packet being lost
→ MESSAGE: A send (seq: 8 ack: 0)
→ MESSAGE: A send (seq: 9 ack: 0)
```

B receives an out-of-order packet (pkt 6), and it sends ack 4 to demand **pkt 5** again. In the meanwhile, A is extending the window size (from 5 to 12) and the timer at **pkt 5** is ticking.

```
EVENT time: 101.126633, type: 2, fromlayer3 entity: 1
→ MESSAGE: B receive (seq: 6 ack: 0)
→ MESSAGE: B send (seq: 0 ack: 4)
```

Sender (A): **second timeout on pkt5**; timer starts on **pkt 5** and **pkt 5** to **pkt 12** are retransmitted.

```
EVENT time: 131.813599, type: 0, timerinterrupt entity: 0
→ MESSAGE: A timeout (index: 5)
→ START TIMER: starting timer for index: 5 at 131.813599,
will timeout at 195.945343. MESSAGE: A send (seq: 5 ack: 0)
→ MESSAGE: A send (seq: 6 ack: 0)
→ MESSAGE: A send (seq: 7 ack: 0)
→ MESSAGE: A send (seq: 8 ack: 0)
→ MESSAGE: A send (seq: 9 ack: 0)
→ MESSAGE: A send (seq:10 ack: 0)
→ MESSAGE: A send (seq:11 ack: 0)
→ MESSAGE: A send (seq:12 ack: 0)
```

Receiver (B) finally receives **pkt 5** and acks it by sending **ack 5**. Sender (A) receives **ack 5** and advances its base to 6, starting the system timer on **pkt 6**.

```
EVENT time: 139.508224, type: 2, fromlayer3 entity: 1
→ MESSAGE: B receive (seq: 5 ack: 0)
→ MESSAGE: B input (eeeeeeeeeeeeeeeeeeee)
→ MESSAGE: B send (seq: 0 ack: 5)

EVENT time: 142.460648, type: 2, fromlayer3 entity: 1
→ MESSAGE: B receive (seq: 6 ack: 0)
→ MESSAGE: B input (ffffffffffffffffffff)
→ MESSAGE: B send (seq: 0 ack: 6)

EVENT time: 143.077148, type: 2, fromlayer3 entity: 0
→ MESSAGE: A receive (seq: 0 ack: 5)
→ START TIMER: starting timer for index: 6 at 143.077148,
will timeout at 207.208893.
```

1b) pkt 5 送2次都失敗

Sequence diagram illustrating packet transmission and acknowledgment between two hosts, A and B.

**Host A:**

- starttimer(4)
- base=4
- starttimer(5)
- time=53.3
- base=5
- t=92.2
- timeout on pkt 5
- starttimer(5)
- Full window
- t=131.6
- timeout on pkt 5
- starttimer(5)
- starttimer(6)

**Host B:**

- out-of-order
- out-of-order
- out-of-order

**Transmissions:**

- A sends pkt 5 (failed, X)
- A sends pkt 6 (out-of-order)
- A sends pkt 7 (out-of-order)
- A receives ack 4
- A sends pkt 8 (out-of-order)
- A receives ack 4
- A sends pkt 9
- A times out on pkt 5 (t=92.2)
- A retransmits pkt 5 (failed, X)
- A retransmits pkt 6 (out-of-order)
- A retransmits pkt 7 (out-of-order)
- A retransmits pkt 8 (out-of-order)
- A receives ack 4
- A enters Full window
- A sends pkt 10 (out-of-order)
- A sends pkt 11 (failed, X)
- A sends pkt 12 (failed, X)
- A receives ack 4
- A times out on pkt 5 (t=131.6)
- A retransmits pkt 5 (failed, X)
- A retransmits pkt 12 (out-of-order)
- A receives ack 4
- A receives ack 5

Receiver (B) acks 9 but its ack is corrupted, so technically Sender (A) does not receive ack 9.

```
EVENT time: 78.582726, type: 2, fromlayer3 entity: 1
|→ MESSAGE: B receive (seq: 9 ack: 0)
|→ MESSAGE: B input (iiiiiiiiiiiiiiiiiiii)
|→ MESSAGE: B ... send (seq: 0 ack: 9)
|...|...|TOLAYER3: packet being corrupted

EVENT time: 83.568939, type: 2, fromlayer3 entity: 0
|→ MESSAGE: A receive a corrupted packet!
```

Since A keeps sending next packets, B receives pkt 10 and acks 10 (sending ack 10).

From B's perspective, everything goes smoothly (no packet is lost or corrupted for B). This time A receives an out-of-order ack 10 without receiving ack 9 beforehand. Fortunately, **Go-Back-N's mechanism implies that once an ack  $x$  is received by A, all packets before  $x$  have received by B without problems**, so A does not need to retransmit pkt 9 to get ack 9; on the contrary, A can safely ignore the unreceived ack 9 and slide the base index forward to  $10+1=11$  and start the timer on pkt11.

```
EVENT time: 94.178238, type: 2, fromlayer3 entity: 1
→ MESSAGE: B receive (seq:10 ack: 0)
→ MESSAGE: B input (jjjjjjjjjjjjjjjjjjjj)
→ MESSAGE: B send (seq: 0 ack:10)

EVENT time: 102.111412, type: 2, fromlayer3 entity: 0
→ MESSAGE: A receive (seq: 0 ack:10)
→ START TIMER: starting timer for index: 11 at 102.111412,
→ will timeout at 130.484818.
```

(d)./{student-id} 20 0.1 0.1 10 2

Since corrupted and loss probabilities are turned to 0.1, packet transmission is greatly delayed; there are times for A that its window size is full and has to refuse data. When A refuses data, it does not buffer the message passed from layer5 but let it disappears (assuming A's window size is A's buffer size). This leads to a situation that the predefined sending packet number does not match the final reception number (in this case, B only receives 13 packets). One way to guarantee all messages from layer5 are passed correctly is to have a boolean return value for `A_output()` to indicate output success/failure, and use a conditional statement in line 296 to decrement `nsim` if fails.

```
EVENT time: 123.718544, type: 1, fromlayer5 entity: 0
→ MESSAGE: A output (mmmmmmmmmmmmmmmmmm)
Window size is full. The data is refused.
```

```
nsim++;
if (eventptr->evententity == A) {
    printf("\tMESSAGE: A output (%s)\n", msg2give.data);
    A_output(msg2give);
    // window size sliding issue?
    // If holding the data when output fails:
    // if (!(A_output(msg2give))){nsim--;}
}
```

## 2. EWMA

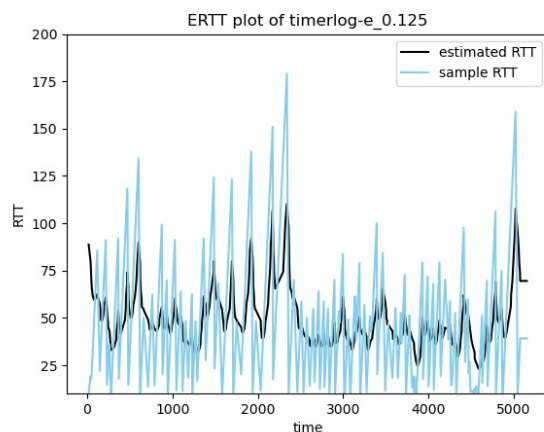
The below chart is estimated with: `./{student-id} 1000 0.1 0.1 10 2`  
(`#define BUFFSIZE 1024`)

$$EstimatedRTT = (1 - \alpha) \times EstimatedRTT + \alpha \times SampleRTT$$

EMWA is one way to estimate the roundtrip time/the time to wait before a timeout. It is clear from the figures below that the more the  $\alpha$  is set, the more variations in estimated RTT and the more similar estimated RTT and sample RTT are, since the estimated RTT will reflect more of the current congestion. On the other hand, setting  $\alpha$  lower smooths out the estimated RTT.

EMWA in TCP implementation details:

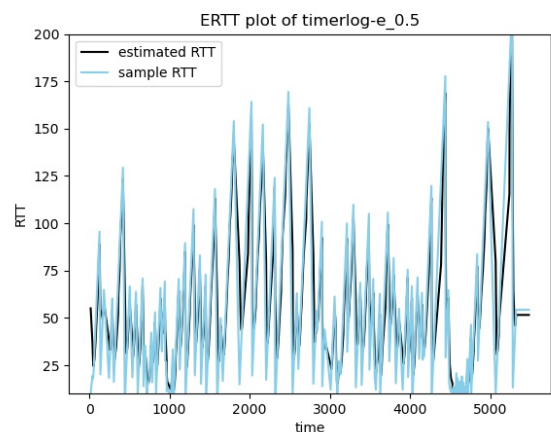
- (1) For sample RTT, only measure the segments that have been retransmitted ONCE. The reason is specified [here](#). Since doing so will result in low update rate, the code implements updates in both first-time transmission and retransmission.
- (2) Don't try to access a corrupted segment's ACK for it violates the above rule and it returns wrong value.



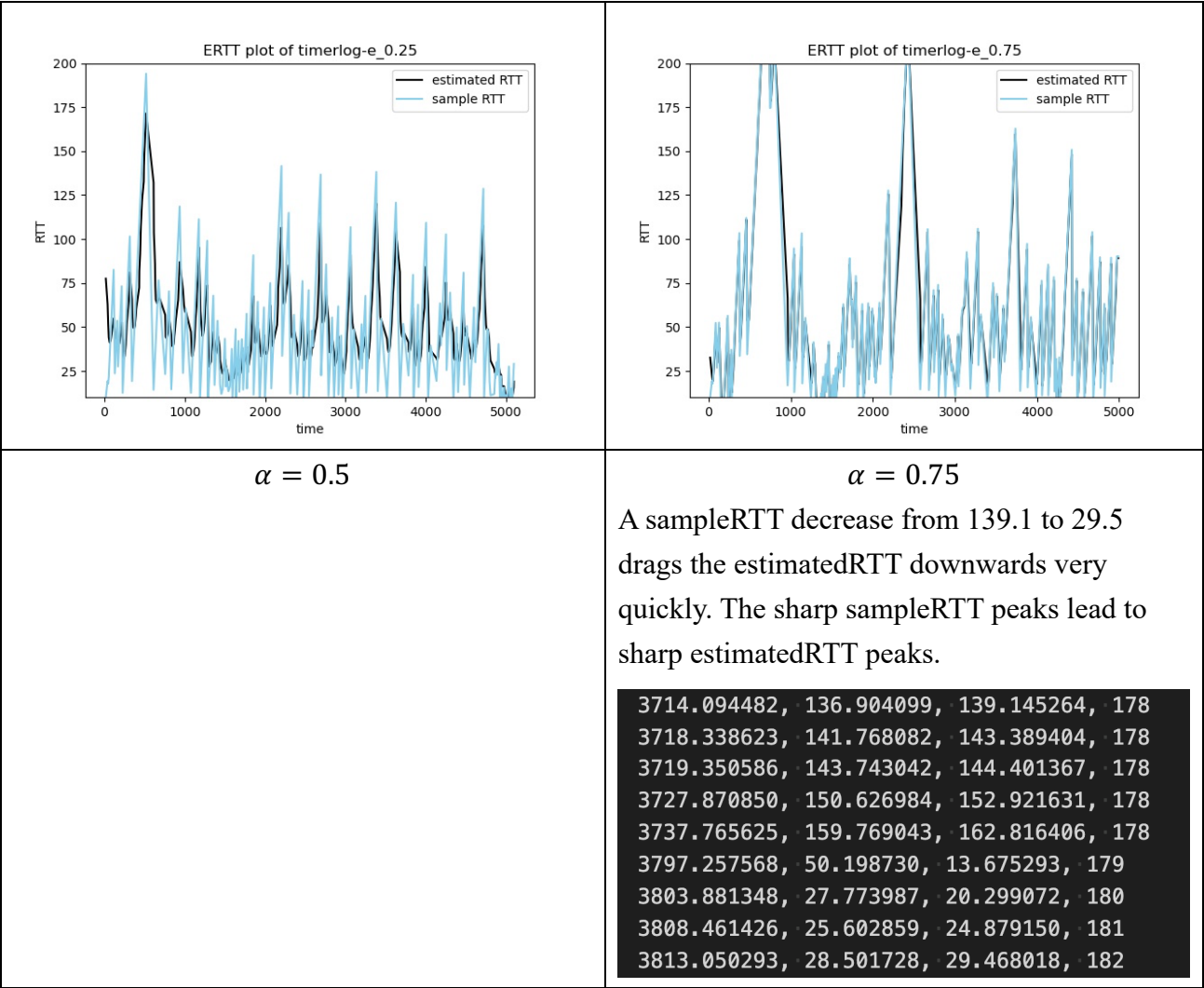
$\alpha = 0.125$  (recommended by [rfc6298](#))

A part of the output log shows that a dramatic increase in sampleRTT (from 54.2 to 148.5) does not affect the estimatedRTT dramatically (slight increase, from 42.5 to 93.0).

```
545 4915.510254, 42.512726, 54.206543, 250
546 4924.033203, 45.039822, 62.729492, 250
547 4939.584473, 49.194939, 78.280762, 250
548 4964.362305, 55.927895, 103.058594, 250
549 4968.397461, 62.323627, 107.093750, 250
550 4983.504883, 69.808319, 122.201172, 250
551 4991.586426, 77.367615, 130.282715, 250
552 4999.700195, 84.996223, 138.396484, 250
553 5009.801758, 92.933952, 148.498047, 250
```



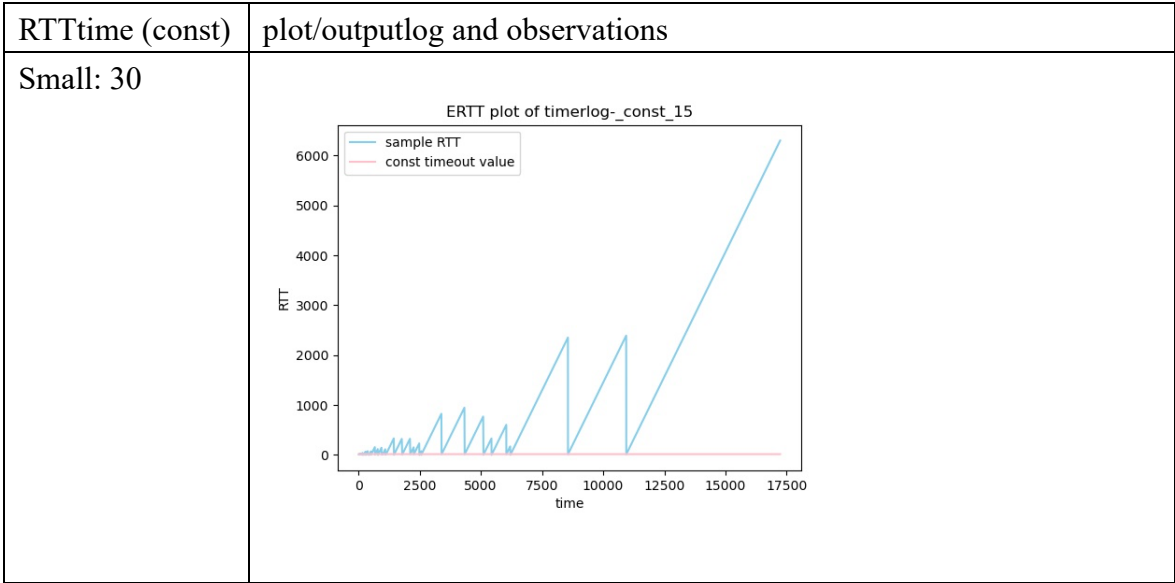
$\alpha = 0.25$

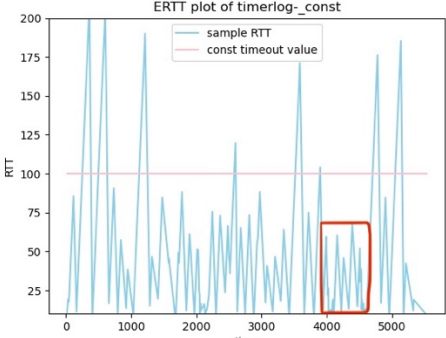
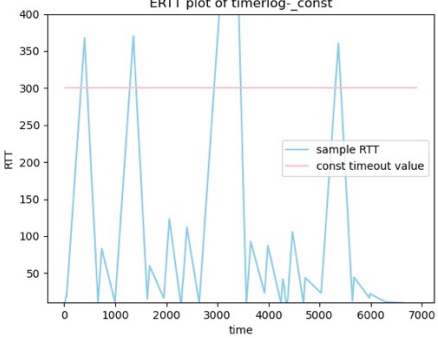


### 3. Bonus: constant timeout value

Argument Setting: 500 0.1 0.1 10 2

Const RTT time: 30, 100, 300



	<p>A timeout value too small results in early timeout. Observe that B is sending ack9 at 157.3 (and it arrives at 165.2 at A's side), but A times pkt9 out too early at 157.8 and results in many retransmitted packets (which will aggravate congestion). The above plot (sampleRTT) also illustrates the serious congestion at the end of the transmission.</p> <pre> EVENT time: 151.397858, type: 2, fromlayer3 entity: 0 MESSAGE: A receive (seq: 0 ack: 9) START TIMER: starting timer for index: 10 at 151.397858, will timeout at 166.397858.Warning: attempt to start a timer that is already running </pre> <pre> EVENT time: 157.308075, type: 2, fromlayer3 entity: 1 MESSAGE: B receive (seq: 9 ack: 0) MESSAGE: B send (seq: 0 ack: 9) </pre> <pre> EVENT time: 157.754959, type: 0, timerinterrupt entity: 0 MESSAGE: A timeout (index:10) START TIMER: starting timer for index: 10 at 157.754959, will timeout at 172.754959. MESSAGE: A send (seq:10 ack: 0) TOLAYER3: packet being lost MESSAGE: A send (seq:11 ack: 0) MESSAGE: A send (seq:12 ack: 0) MESSAGE: A send (seq:13 ack: 0) MESSAGE: A send (seq:14 ack: 0) </pre> <pre> EVENT time: 165.152634, type: 2, fromlayer3 entity: 0 MESSAGE: A receive (seq: 0 ack: 9) START TIMER: starting timer for index: 10 at 165.152634, will timeout at 180.152634.Warning: attempt to start a timer that is already running </pre>
Medium: 100	 <p>It looks nice generally, except for the red-marked part when RTTtime is obviously smaller than 100 time units. Sender (A) may have saved a packet loss/corruption event if it triggers a timeout earlier in that small interval.</p>
Large: 300	 <p>Observe that pkt69 is lost.</p>

```
MESSAGE: A . . . send (seq:69 ack: 0)
. . . . . TOLAYER3: packet being lost
```

B receives out-of-order spkt72 and acks the highest-in-order-received segment number: ack 68, but since the timeout does not occur timely, B keeps received more out-of-order segments from A and desert them.

```
EVENT time: 4708.105957, type: 2, fromlayer3 entity: 1
MESSAGE: B receive (seq:72 ack: 0)
MESSAGE: B send (seq: 0 ack:68)
```

```
Updated estimatedRTT, samplerTT: 34.381966, 36.064941
EVENT time: 4715.588867, type: 2, fromlayer3 entity: 1
  MESSAGE: B receive (seq:74 ack: 0)
  MESSAGE: B send (seq: 0 ack:68)
```

```
EVENT time: 4732.615234, type: 2, fromlayer3 entity: 1
  MESSAGE: B receive (seq:76 ack: 0)
  MESSAGE: B send (seq: 0 ack:68)
  ....TOLAYER3: packet being lost
```

On the other hand, A suffers from window size problem because B does not ack on pkt69.

```
Window size is full. The data is refused.  
EVENT time: 4871.461914, type: 1, fromlayer5 entity: 0  
+ MESSAGE: A output (rrrrrrrrrrrrrrrrrrrr)  
Window size is full. The data is refused.  
EVENT time: 4878.018066, type: 1, fromlayer5 entity: 0  
+ MESSAGE: A output (sssssssssssssssssss)  
Window size is full. The data is refused.  
EVENT time: 4887.592285, type: 1, fromlayer5 entity: 0  
+ MESSAGE: A output (ttttttttttttttttttt)  
Window size is full. The data is refused.  
EVENT time: 4901.492676, type: 1, fromlayer5 entity: 0  
+ MESSAGE: A output (uuuuuuuuuuuuuuuuuuuu)  
Window size is full. The data is refused.  
EVENT time: 4907.055176, type: 1, fromlayer5 entity: 0  
+ MESSAGE: A output (vvvvvvvvvvvvvvvvvvvvv)  
Window size is full. The data is refused.
```

When finally pkt69 timeout, 300 time units have already wasted.

```
EVENT time: 5007.742676, type: 0, timerinterrupt entity: 0
MESSAGE: A timeout (index:69)
```