University of Lleida

Master's Degree in Informatics Engineering

Higher Polythecnic School

Exercise 1

ICT Project: Communication Services and Security

Cèsar Fernández Camón

Albert Pérez Datsira

March 20, 2021

Table of contents

1 Problem 1 - RED congestion control (RED)

(Q1) Probability of a segment being dropped with a *AvgLen = 8*

According to the preventive mechanism of congestion avoidance, most commonly called *RED* reffering to *Random Early Detection.* We must apply the following formula:

$$P' = \max P \cdot \frac{AvgLen - MinTh}{MaxTh - minTh} \tag{1}$$

Now it is observed that we have all required parameters to solve *P'*.

But, it is needed to mention, we must look for the correct policy where our values can fit taking into account the two defined thresholds, *MinTh* and *MaxTh*.

Therefore, evaluating the situation it is concluded that the policy where our values fit, as it refers to *"Compute probability P/Drop out segment with probability P"*, is *MinTh <AvgLen <MaxTh*, so:

$$MinTh(4) < AvgLen(8) < MaxTh(10) \tag{2}$$

Then, by filling the first formula described above we obtain:

$$\begin{aligned} P' &= \max P \cdot \frac{AvgLen - MinTh}{MaxTh - minTh} = \\ &= 0.4 \cdot \frac{8-4}{10-4} = 0.2\widehat{6} \end{aligned} \tag{3}$$

To conclude, the resulting approximate probability is $\mathbf{0.2\widehat{6}}$

(Q2) Probability that 3 consecutive segments enter into the queue, assuming that all of them find the same average queu length (AvgLen) of 8

In this specific case we can say that the segment **1**, has a probability of entering the queue of:

$$P = 1 - \text{Probability of being dropped} \tag{4}$$

- Segment 1:

  Simply by applying the following,

  $$P = 1 - 0.2\widehat{6} = \mathbf{0.7\widehat{3}}$$

- Segment 2:

  Then, the probability of the second segment is that of the first one but twice,

  $$P = 0.7\widehat{3}^2 \approx \mathbf{0.54}$$

- Segment 3:

  Same as above, but three times

$$P = 0.7\widehat{3}^3 \approx \mathbf{0.40}$$

(Q3) Same probability as previous point (2) assuming a modified RED congestion control where the probability of a segment being dropped is

$$\frac{P}{-\, compt \cdot P} \tag{5}$$

being *P* the same probability computed at point (1). compt is the number of segments that entered into the queue from the last dropped segment. Assume compt = 0 for the first segment entering into the queue.

Assuming that *compt = 0* in the first segment that enters the queue we can use the given equation:

- Segment 1

$$\frac{P}{1 - compt \cdot P} = \frac{0.2\widehat{6}}{1 - (0 \cdot 0.2\widehat{6})} = \mathbf{0.2\widehat{6}}$$

We assume that for the second segment *compt=1* and for the third *compt=2*

- Segment 2

$$\frac{P}{1 - compt \cdot P} = \frac{0.2\widehat{6}}{1 - (1 \cdot 0.2\widehat{6})} \approx \mathbf{0.\widehat{36}}$$

- Segment 2

$$\frac{P}{1 - compt \cdot P} = \frac{0.2\widehat{6}}{1 - (2 \cdot 0.2\widehat{6})} \approx \mathbf{0.57}$$

2 Problem 2 - Weighted Fair Queueing (WFQ)

Write a Python script tha returns the transmission order sequence of a (WFQ) policy based on a list of triplets.

Input:

- File path name containing the list of triplets, reffering to the packets of the transmission ordered in time, to be scheduled where each triplet must represent:
  1. Arrival time (float)
  2. Packet length (float)
  3. Flow/Stream identifier (*integer* ≥ 1)
- Fraction of the bandwidth assigned to each flow (as a percentatge). Comma separated. As an example: 50,10,40

Output:

- The transmission sequence of the packets once the network scheduling algorithm has been applied.

A script using *Python 3* has been created that meets the above requirements printing by console the resulting sequence in the following format:

$$Result : [2, 1, 3, 4, 5, 6, 7, 8, 9]$$

In addition, several *.txt* files have been created so that they can be used for testing purposes.

If you want to run such a script in your python environment, simply follow the usage below:

```
(base) Alberts-MacBook-Air:Problem2 albert$ python script.py -h
usage: Network Scheduling Script [-h] [-v] file flows

Script implementing a packet-based network scheduling algorithm called Weighted Fair Queueing.

positional arguments:
  file          File path name containing the list of triplets to be scheduled where each one represents:
                   1. Arrival time (float)
                   2. Packet length (float)
                   3. Flow/Stream identifier (integer >= 1)
  flows         Fraction of the bandwidth assigned to each flow (as a percentage)(float). Comma separated. As an example: 50,10,40

optional arguments:
  -h, --help     show this help message and exit
  -v, --verbose  Display additional details about the execution such as the packets received and delivered at each time.
```

Figure 1: Python WFQ script usage

2.1 Scalable Version

In addition to the previously described script, a scalable version was made at the algorithm level, which was originally used to test results quickly between the Fair Queueing algorithm and its Weighted version.

This version has a slightly different argument model and a program structure based on a switcher which allows it to execute different algorithms according to the user's will.

Its code can be found in the *netfork.py* file, and its usage is shown below:

```
(base) Alberts-MacBook-Air:Problem2 albert$ python networkScheduler.py -h
usage: Network Scheduling Script [-h] [-s {FQ,WFQ}] [-f FLOWS] [-v] file

Script implementing a packet-based network scheduling algorithm called Fair Queueing and its Weighted version. Also it is prepared to manage other network
scheduling algorithms.

positional arguments:
  file           File path name containing the list of triplets to be scheduled where each one represents:
                    1. Arrival time (float)
                    2. Packet length (float)
                    3. Flow/Stream identifier (integer >= 1)

optional arguments:
  -h, --help     show this help message and exit
  -s {FQ,WFQ}    String to specify the network scheduling algorithm policy. If not specified will be executed the Fair Queueing as a default option.
                   · FQ = Fair Queueing
                   · WFQ = Weighted Fair Queueing.
  -f FLOWS       Fraction of the bandwidth assigned to each flow (as a percentage)(float). Comma separated. As an example: 50,10,40
                 Will only be taken into account when the policy {WFQ} is specified.
  -v, --verbose  Display additional details about the execution such as the packets received and delivered at each time.
```

Figure 2: Python scalable network scheduling script usage

3 Problem 3 - TCP Data Interchange

Write two Python scripts (transmitter and receiver) that using sockets emulates a TCP data interchange (half duplex) according the following rules:

- **General**
  - Use a *Karn/Partridge RTT* (Round Trip Time) estimator, without *Exponential Back-Off* and $\alpha = 0.8$
  - A Slow Start congestion control (CWMAX=4)
  - All transmitted segments have *MSS* bytes
  - Segment transmission time is 1 s.
  - ACK segment have size 0
  - No headers
  - Propagation delay is 0 s.
- **Transmitter**
  - Segments are numbered starting at 0. Acknowledgment based on segment number (no bytes)
  - Segments with a prime sequence number are considered lost (when transmitted for the first time)
  - Consider that transmitter has always enough information to send
  - Take an initial TimeOut value of 10 s.
- **Receiver**
  - Send ACK after 2 s. without a correct reception
  - Send ACK immediately when 3 or more segments in sequence
  - Receiver buffer size: 10 *MSS*

3.1 Implementation Main Aspects

Two scripts have been developed, one for the receiver and one for the transmitter based on the above requirements.

Their main functions/aspects are as follows:

Receiver

- **main Thread**
  - performs the reception of the segments as well as the checking of their validity, as well as the reading in case of 3 or more segments in sequence.
- **thread_timer()**
  - function that executes in the second thread and checks if a segment has been received correctly, otherwise it sends an ack every 2 s.

Transmitter

- **process_ack()**
  - function that is executed in a specific thread to process the segment ACK's that are received. Applying the corresponding *Karn/Patriage* and *Slow-Start*
- **send_buffer()**
  - function that executes the main thread of the program and carries out the transmission of the segments launching in each transmission the control thread for that.
- **time_out()**
  - function that performs the wait, for each transmission, to check whether timeOut has occurred or not. If so, the segment must be retransmitted.
- **send_retrans_buffer()**
  - function that performs retransmission of the segments that have timed out.

3.2 Datagram format

The datagram format is very simple. They are based on text strings separated by the '-' (dash) character.

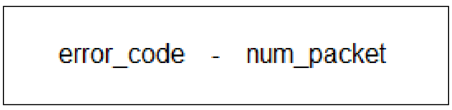

Figure 3: Datagram format

3.3 Plotting: *cwnd* and *sRTT*

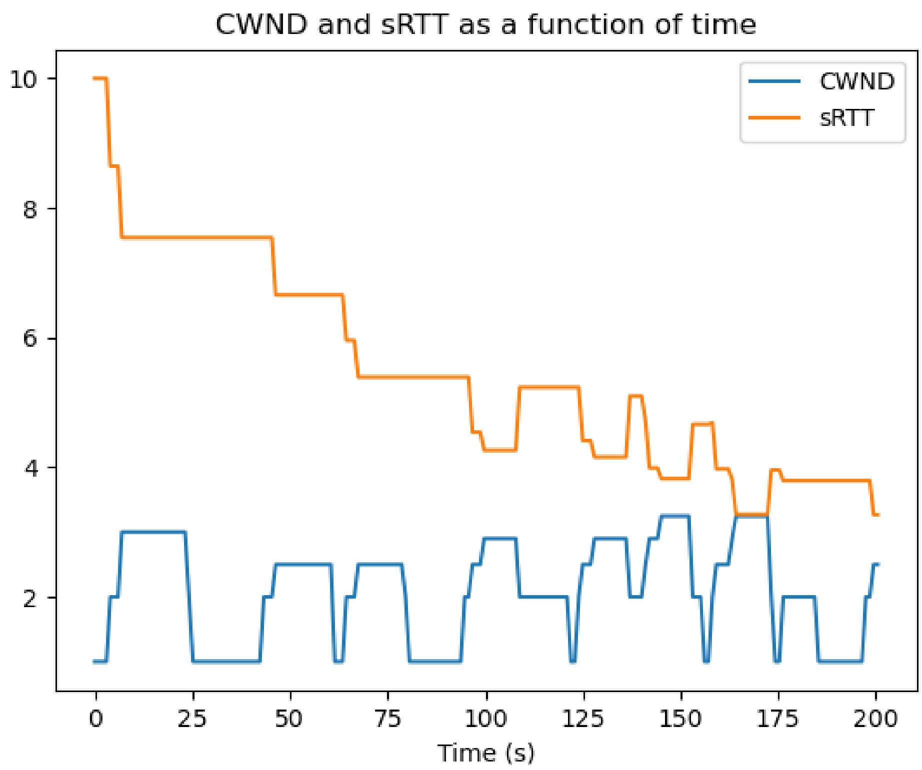

Figure 4: *CWND* and *sRTT* as a function of time

This figure shows how the sRTT becomes tighter and tighter as fewer packets are lost. And on the other hand, we observe that the network congestion makes the upstream and downstream as the Slow-Start algorithm is expected to behave.