EE122: Communication Networks **S'12**

Project: Part 1/2 — Due April 2

Lecturers: Shyam Parekh & Jean Walrand *GSI: Shicong Yang*

This project should be done in teams of up to three students. You will be asked to present your work in one-by-one interviews in addition to turning in a description and the code. We believe this project to be reasonable in scope and amount of work. It should give you the basic ideas of how to use sockets to write applications over the internet. At the same time, you will learn features of adaptive protocols.

The socket programming project that we propose is based on the Optimized Edge Routing protocol of Cisco (see link below):
`http://www.cisco.com/en/US/products/ps6628/products_ios_protocol_option_home.html`

We suggest that you check that description of OER, even though we consider a simplified version of it. The idea is that some user has two paths to connect to the internet and the protocol selects the path that seems to be better, probably because it happens to be less congested.

After the second part of the project, you will have built a system that emulates two paths. At a given time, the user is using one path and is probing the alternate path periodically. When the user estimates that the alternate path is better, she switches to that path and the roles of the two paths are reversed. For us, a measure of quality of a path is the average round-trip time.

In the first part of the project, which we consider here, you build only one path and the delay estimator. That is, there is one path from a host S to a destination D. That path goes through a queue A. The queue emulates the delays that occur along a more complicated path from S to D. As you will see, the queue is a first-come, first-served queue with service times that are independent and identically distributed and have a probability distribution function that depends on a parameter $a(t) = a$. In the second part of the project, we will change the parameter $a(t)$ to reflect changes in operating conditions of the path.

For the first part of the project, we consider a source S, a queue A, and a destination D. The idea is to build the path from S to A and from A to D, and a path from D to S for acknowledgments (see Figure 1). You should consider the three different links of Figure 1 as UDP transmissions.

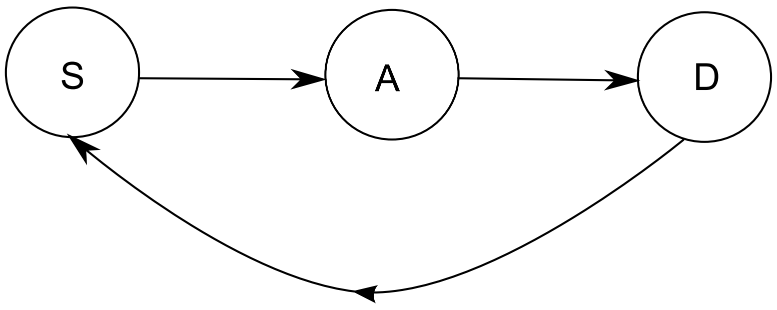

Figure 1. Description of the path

In the following, we explain what happens at the different nodes.

Source S**:**
We consider that the source S sends packets at regular intervals (every T seconds) to the queue A. We will choose $T = 1$ second. The goal is to estimate an average round-trip time value, i.e., the average time for an acknowledgment to come back to the source after the source has sent a packet. There are a few ways to estimate that delay. One approach is to keep track of the delay for all packets. A simpler approach that uses only one timer at a time works as follows. Consider the first packet. When the source S sends that packet, it stores the time. When S receives an acknowledgment of that packet from D, it notes the delay T_1 between the transmission time of the packet and the reception time of its ACK. We then repeat the operation for the next packet that S sends after it received the acknowledgment from that first packet. In the example given by Figure 2 the source keeps track of the delays for packets 1, 3, and 5.

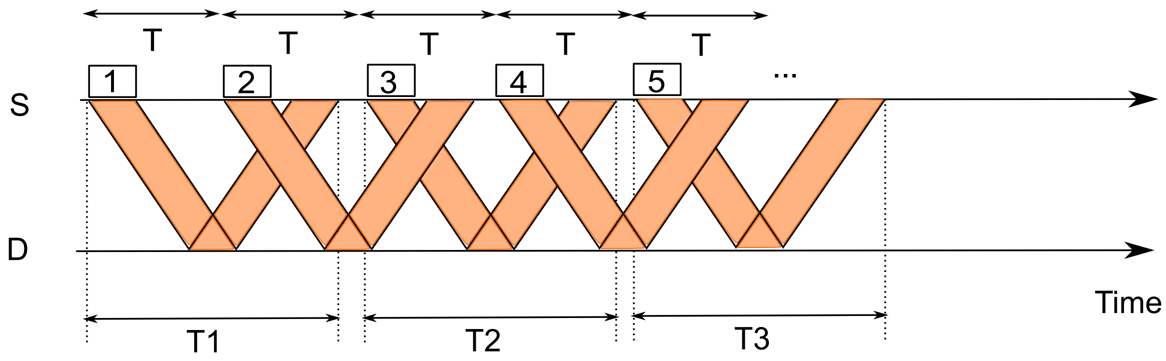

Figure 2. Sender S sends packets every T seconds

The source calculates the average value A_n of the round-trip times $\{T_1, \ldots, T_n\}$ recursively as exponential averages defined as follows ($q < 1$ is a parameter of the algorithm):

$$A_n = (1 - q)A_{n-1} + qT_n.$$

Observe that this basic scheme would fail if an acknowledgment were to not come back, which is possible as paths are not fully reliable. To handle such situations, we define a timeout value after which we stop waiting for an acknowledgment. For simplicity, let *Timeout* $= 2 \times A_n$. You need to give A_0 a reasonable initial value.

When the round trip time of a packet is greater than *Timeout*, we consider that the packet is lost and we stop its timer. This is shown in Figure 3. For simplicity, the source S does not retransmit packets for which the AKCs fail to arrive after a timeout. Of course, it may happen that this ACK arrives after *Timeout* and your algorithm must be able to handle this case.

Note that you can just send numbers as the payload in the packets, e.g., 1, 2, 3... , and also in the ACK packets that D sends to S. That way, it is easy to know which packet is

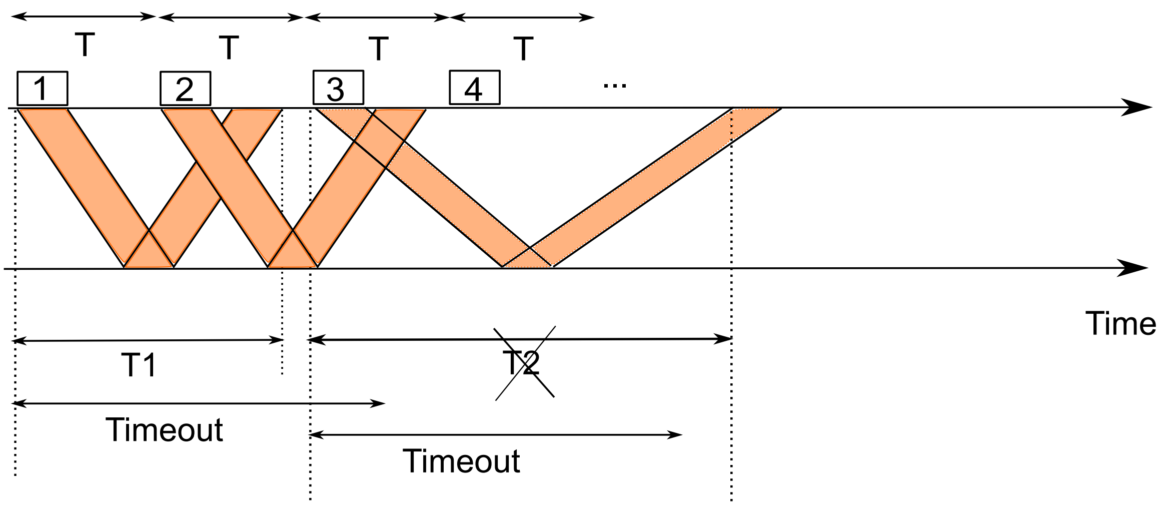

Figure 3. When the round trip time of a packet is greater than the Timeout, we consider that the packet is lost and we dismiss the estimated time.

ACKed by destination D. Given the gaps between packets, adding a dummy payload to make the packets longer does not add much realism to the system.

Observe that the source S must read from the socket $D \to S$ while it runs a process that waits until the next packet transmission. This simultaneous waiting for a read and waiting for some time can be implemented either using multi-threading or with the *SELECT()* function.

Queue A:

The queue A receives the packets from the source S and stores them. The queue A forwards the packets on a first-come first-served basis. The queue transmits the packets to their destination D after independent exponentially distributed random service times Q_n for the successive packets; these random variables have rate $a(t)$ at time t. See Figure 4. For this part of the project consider $a(t)$ to be a constant value $a(t) = a$. In the second part of the project, we will change $a(t)$ at some times to reflect the changes in the path conditions.

The queue works as follows. Say that the queue just transmitted its n-th packet a time τ_n. If the queue is not empty, then it will transmit its $(n+1)$-st packet at time $\tau_n + Q_{n+1}$. If the queue is empty at time τ_n, then it will transmit its $(n+1)$-st packet at time $\alpha_{n+1} + Q_{n+1}$ where α_{n+1} is the arrival time of the $(n+1)$-st packet at the queue.

Note that the system should work when packets fail to arrive at the queue. Accordingly, the queue must keep track of the sequence numbers of the packets that have arrived, and not only of the number of packets in the queue. If you transmit numbers as the packet content, the sequence can be implied from the packets' content.

Here again, the queue A must read from the incoming socket. But we do not want the program to be blocked and waiting to read from the socket because we also need to forward packets if the queue is not empty. Thus, *SELECT()* or multi-threading should

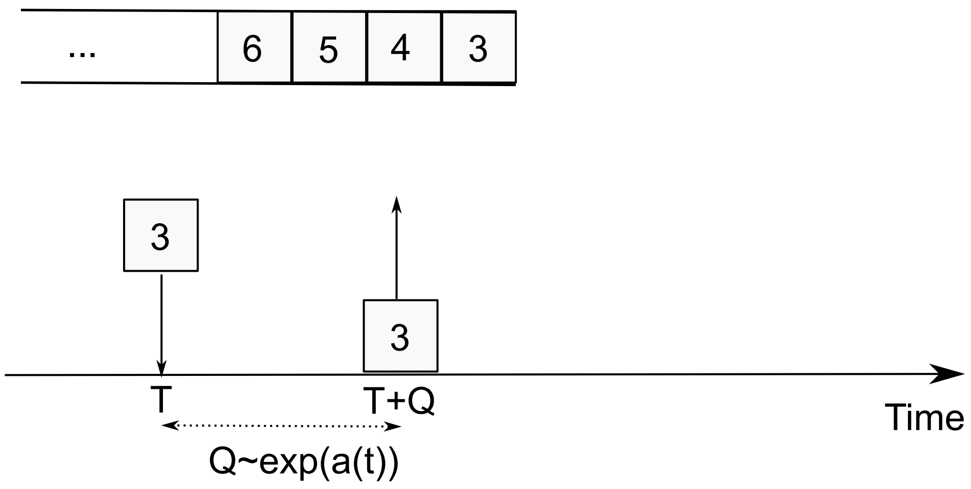

Figure 4. Queue A, assuming that the queue was empty when packet 3 arrived.

be used.

Destination D:

Every time the destination D receives a packet from the queue A, it sends immediately an acknowledgment to the source S.

Project report

There is no page limit on the report. Credits will be given based on how well and clearly you explain things. You need submit a report which consists at least the following parts:

1. On title page, make a table to explain each team member's contribution to this project. Everyone who agrees with it should sign after each person's name.
2. Briefly describe and explain your code. Clearly indicate how to run your program.
3. Study the relation between average delay and queue's servicing rate. Plot a graph that shows average delay as a function of the queue's servicing rate. Explain why the graph should look like the way it is.
4. Study the relation between average delay and the parameter of the exponential averaging. Plot a graph that shows average delay as a function of the parameter of the exponential averaging. Explain why the graph should look like the way it is.
5. Study the relation between queue dynamics and queue's servicing rate. You can plot how the queue length evolves for a given a(t) (a(t)=a in part 1 of the project) over some duration of time. Plot at least 3 graphs when a(t) = 0.5, 1, and 10 [packets/s]. You can choose other rates too. Explain why the graphs should look like the way they are.