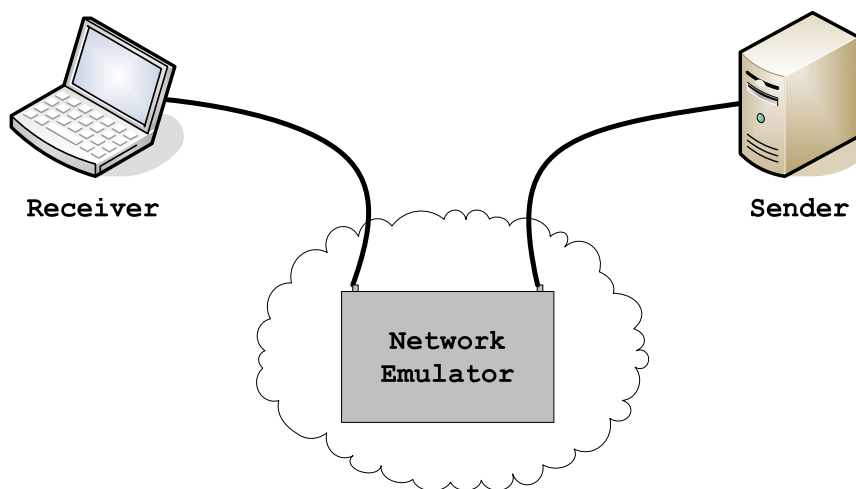


Computer Communications and Networks (COMN) Course, Fall 2009

Coursework 2

The overall goal of this coursework is to develop application layer support for end-to-end reliable data delivery using the sliding window approach. In particular, you will implement three different sliding window protocols – *Stop-and-Wait*, *Go Back N* and *Selective Repeat* – in Java at the application layer using socket API over the unreliable datagram protocol (UDP). Note that the stop-and-wait protocol can be viewed as a special kind of sliding window protocol in which both sender and receiver window sizes are equal to 1. For each of the three sliding window protocols, you will implement the two protocol endpoints referred to as *sender* and *receiver* respectively; these endpoints also act as application programs. Data communication is unidirectional, *requiring transfer of a large file from the sender to the receiver* as a sequence of smaller messages.

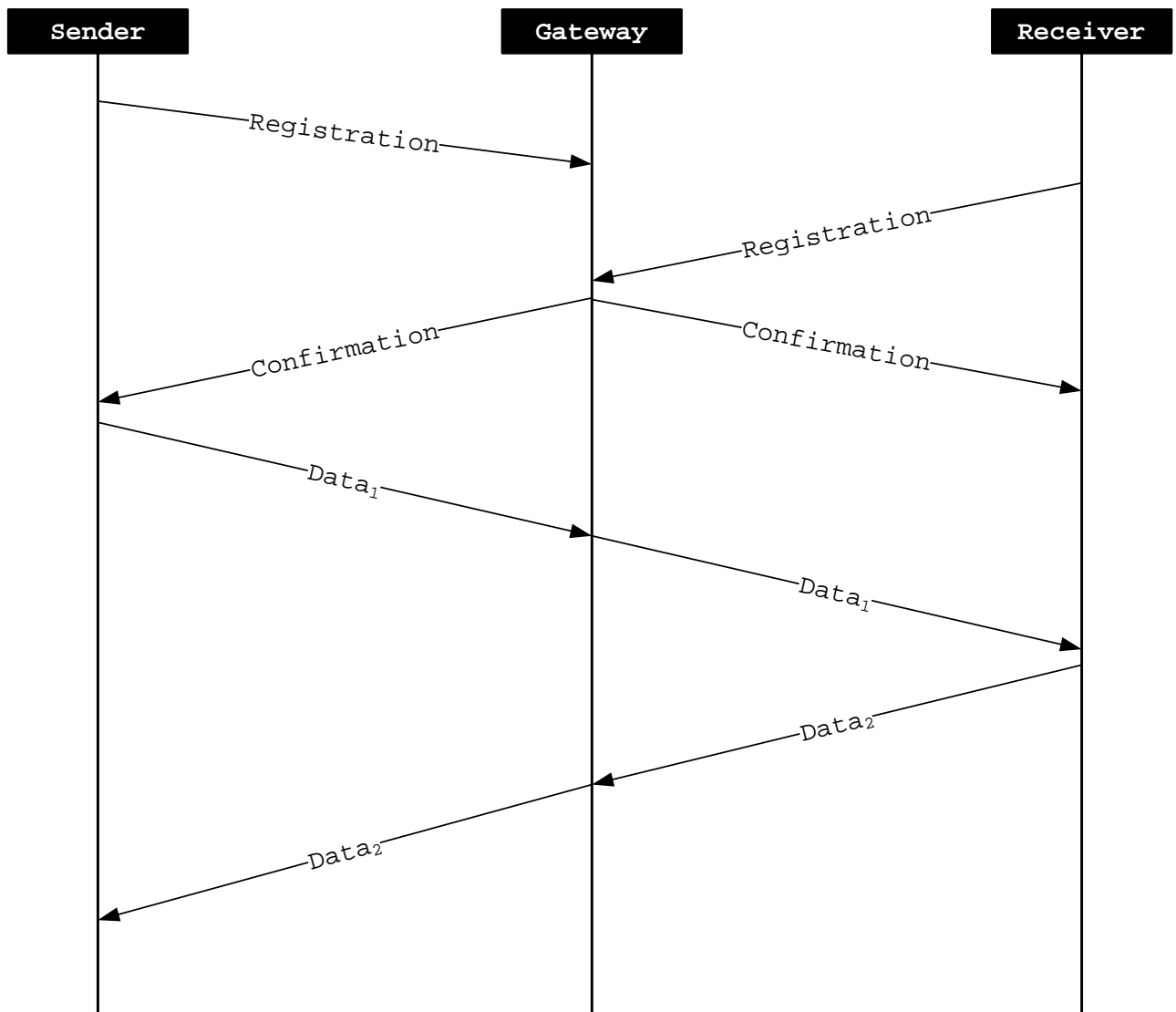
To test your protocol implementations and study their performance in a controlled manner in a laboratory environment, you are provided with a UDP-based application layer program that acts as a simple Internet emulation gateway¹ between the sender and the receiver, each of the three programs possibly running on a different machine (as illustrated below). Depending on the target network to be emulated, the gateway is configured using a set of values for parameters such as delay, packet loss rate and packet duplication rate; all packets between application endpoints flowing via the gateway are subject to delay, loss and duplication as per the gateway configuration. Essentially, the gateway can emulate a wide range of network conditions, such as introduce delay as in the Internet or satellite link or model packet loss and duplication of a faulty or busy link.



¹ Note that the terms “emulator” and “gateway” are used interchangeably in this document. Similarly, the terms “packet” and “message” are used interchangeably.

The “SimpleNetEmulator” downloadable from [1] is written in C, and already compiled for Linux. It is ready to use on any DICE machine and you do not need to have root privileges in order to run it. When executed without any command line arguments (as in “./SimpleNetEmulator”), it listens for incoming packets on port 54321. Alternatively, you can have the emulator use a specific port number by giving it as a command line argument.

To use the emulator after it is started, both the application endpoints (sender and receiver) must first register with the emulator using a registration-confirmation handshake as shown below. Note that the gateway must receive registration messages from both sender and receiver before it can respond with a confirmation message to either of them.



Registration message is a UDP packet from an endpoint to the emulator (identified by the host running it and the port number it uses). This message must contain a string of the form:

```
%%GatewayRegistration <endpoint> <unique_name> [delay <p>] [drop <q>] [dupl  
<r>] [bandwidth <s>] [timeout <t>]
```

Where,

- <endpoint> is a string that must be either SENDER or RECEIVER depending on the endpoint that is registering.
- <unique_name> is an arbitrary unique identifier string. Both endpoints must use exactly same string so that the network emulator can match them.
- Parameters within square brackets are optional.
- <p> is the one-way propagation delay (as an integer in millisecond units) of the network emulated by the gateway. Default value set to 10ms.
- <q> is the percentage of packets to drop (as an integer between 0-100, inclusive). Default value set to 0.
- <r> is the percentage of packets to duplicate (as an integer between 0-100, inclusive). Default value set to 0.
- <s> is the bandwidth of the emulated network in bits per second. Default value set to 1000000 (i.e., 1Mbps). Note that this bandwidth along with size of a packet is used in the gateway to emulate the “transmission” delay of the packet.
- <t> is the timeout value (as an integer in millisecond units) to represent the validity period for a registration. Emulator cancels the registration if no activity occurs for a period longer than the timeout period. Timeouts also apply to incomplete registrations (e.g., one side registers, but the other does not). Default value for the registration timeout set to 60000 (i.e., 1 minute).

For example, a valid string within a sender’s registration message may look like:

```
%%GatewayRegistration SENDER test delay 100 drop 1 bandwidth 10000000
```

In the above example, sender asks the gateway to emulate a network with 100ms (one-way) propagation delay, 10Mbps bandwidth and 1% loss rate. Also note that the unspecified parameters such as duplication rate take on the default values.

Note that the gateway you are provided with only supports symmetric networks, i.e., where the network path from sender to receiver and the one from receiver to sender have identical characteristics. So you must use identical registration strings from both sides except for the <endpoint> string (which will be SENDER in the sender’s registration message and RECEIVER in the receiver’s registration message). Also for simplicity, we assume that packets are not corrupted in transit (i.e., no bit errors), so there is no need to implement error detection functionality at the endpoints (e.g., checksum). Note, however, that whole packets can be dropped as specified by the drop rate to the emulator.

After both endpoints have sent their registration message to the emulator, they will receive a UDP confirmation message with the following format:

```
%%%GatewayConfirmation
```

Note that if the gateway gets an invalid registration string from either of the endpoints (e.g., parameter values falling outside the acceptable range) then it exits with an error message. In that case you need to restart the gateway again followed by sender and receiver.

Also note that the above registration procedure with the emulator, even though based on UDP, implicitly assumes that the underlying local area network is fairly reliable.

Once the sender receives a confirmation message in response to its registration message, it can start sending data to the receiver.

Your implementation needs to follow the sequence of steps described below.

Step 1: Basic framework (large file transmission under ideal conditions):

Implement sender and receiver endpoints for transferring a large file given at [2] from the sender to the receiver over UDP as a sequence of small messages with 1KB payload (note: 1KB = 1024 bytes) via the network emulation gateway configured with 1Mbps bandwidth, 10ms one-way propagation delay, 0% drops and 0% duplicates (following the gateway registration procedure outlined earlier). The exchanged data messages must also have a header part for the sender to include a 16-bit message sequence number (for duplicate filtering at the receiver in subsequent steps) and a bit to indicate the last message (i.e. end-of-file). Name the sender and receiver developed in this step as `Sender1.java` and `Receiver1.java` respectively.

Step 2: Stop-and-Wait

Extend sender and receiver applications from the previous step to implement a stop-and-wait protocol as described in [4], specifically `rdt3.0` (Call the resulting two applications `Sender2.java` and `Receiver2.java` respectively). This step requires you to define an acknowledgement message that the receiver will use to inform the sender about the receipt of a data message. Filtering out duplicates using sequence numbers is also needed. Test the duplicate detection functionality by specifying a non-zero duplicate percentage to the gateway or using a small retransmission timeout at the sender.

Using a 1% packet drop rate and rest of the emulator configuration parameters as in the previous step (i.e., 1Mbps bandwidth, 10ms one-way propagation delay and 0% duplicates), experiment with different retransmission timeouts. Tabulate the observed

number of retransmissions in the space provided under Question 1 in the results sheet provided in [3].

Also study the impact of retransmission timeouts on the average throughput and answer Question 2 in [3]. For this, modify your sender implementation to measure average throughput (in KB/s) which is defined as the ratio of file size (in KB) to the transfer time (in seconds). Transfer time in turn can be measured at the sender as the interval between first message transmission time and acknowledgement receipt time for last message. Before the sender application finishes and quits, print the average throughput value to the standard output.

Step 3: Go-Back-N

Extend `Sender2.java` and `Receiver2.java` from the previous step in order to incorporate the Go-Back-N strategy as described in [5], by allowing the sender window size to be greater than 1. Use the “optimal” value for the retransmission timeout obtained from the previous step. Experiment with different window sizes at the sender (increasing powers of 2 starting from 1) and different one-way propagation delay values (10ms, 100ms and 1000ms) in the emulator. Across all these experiments, use the following values for the other emulator parameters: 1Mbps bandwidth, 1% drop rate and 0% duplicates. Tabulate your results under Question 3 and answer Question 4 in [3]. Name the sender-receiver pair implemented in this step as `Sender3.java` and `Receiver3.java`.

Step 4: Selective Repeat

Extend `Sender3.java` and `Receiver3.java` to implement selective repeat strategy as described in [6]. Call the resulting two applications as `Sender4.java` and `Receiver4.java` respectively. By configuring the emulator gateway with 1Mbps bandwidth, 100ms one-way propagation delay, 1% drops and 0% duplicates, experiment with different window size values and complete the table under Question 5 and answer Question 6 in [3].

Implementation Guidelines

The programs must adhere to the following standard:

- The sender program must be named as specified below and must accept the following options from the command line:
`java SenderX <Hostname> <Port> <Filename>`
where `<Hostname>` is the name of the host on which the network emulator is running. `<Port>` is the port number where the network emulator is listening on. `<Filename>` is the file to transfer. For example:
`java Sender1 localhost 54321 sfile`
- The receiver program must be named as specified below and must accept the following options from the command line:
`java ReceiverX <Hostname> <Port> <Filename>`
`<Hostname>` is the name of the host on which the network emulator is

running. <Port> is the port number where the network emulator is listening on. <Filename> is the name to use for the received file to save on local disk. For example:

```
java Receiver1 localhost 54321 rfile
```

- Please start each source file with the following comment line:

```
/* Forename Surname MatriculationNumber */
```

For example:

```
/* John Doe 1234567 */
```

- Please use comments in your code!

Submission and Assessment

The deadline for this coursework is 4pm on Friday, 27 November 2009. No late submissions are allowed, except under extenuating circumstances. You must submit an electronic version of your implementations for steps 1 – 4 (including Sender1.java, Receiver1.java, Sender2.java, Receiver2.java, Sender3.java, Receiver3.java, Sender4.java, Receiver4.java), corresponding executables and completed results sheet [3] (as DOC, ODT or PDF) using the submit command as follows: `submit cs3 comn cw2 <directory-name>`. You are expected to work on this coursework on your own. Any kind of copying will result in punitive action.

This coursework accounts for 60% of the coursework mark (or, 24% of the overall course mark). Distribution of marks among the various steps is as follows:

- 5% of the coursework mark for successful completion of step 1
- 20% of the coursework mark for successful completion of steps 1 and 2
- 40% of the coursework mark for successful completion of steps 1, 2 and 3
- 60% of the coursework mark for successful completion of steps 1, 2, 3 and 4

References

1. Application Gateway for Network Emulation:
http://www.inf.ed.ac.uk/teaching/courses/comn/coursework/cwk2_gateway.tar.gz
2. Test file to be used for this coursework:
http://www.inf.ed.ac.uk/teaching/courses/comn/coursework/cwk2_testfile.jpg
3. Results Sheet:
http://www.inf.ed.ac.uk/teaching/courses/comn/coursework/cwk2_results.doc
4. Section 3.4.1 in J. F. Kurose and K. W. Ross, "Computer Networking: A Top-Down Approach" (4th edition), Pearson Education, 2008.
5. Section 3.4.3 in J. F. Kurose and K. W. Ross, "Computer Networking: A Top-Down Approach" (4th edition), Pearson Education, 2008.
6. Section 3.4.4 in J. F. Kurose and K. W. Ross, "Computer Networking: A Top-Down Approach" (4th edition), Pearson Education, 2008.