Assignment 2

Computer Networks (CS 456), Fall 2017

Reliable Data Transfer

Due Date: Oct 31, 2017 11:59pm

TAs: Nashid Shahriar, Elaheh Jalalpour, Anthony Ang

Use piazza for all communication

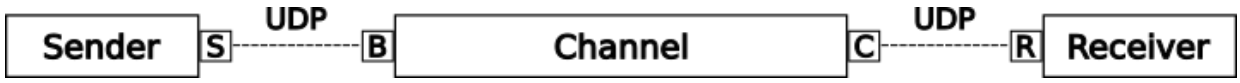
TA office hours: Thursdays, 3.00-4.00pm, Room: DC 3509

Work on this assignment is to be completed individually

In this assignment, you are provided with an unreliable channel emulator. You must implement both the *Go-Back-N* and *Selective Repeat* versions of pipelined reliable data transfer, as well as a simple file transfer application that can transfer any file (including binary files) over the unreliable channel emulator. The reliable transfer protocol must be able to handle network errors such as packet loss, duplication, and reordering. For simplicity, the protocol only needs to be unidirectional, i.e., data flows in one direction (from sender to receiver) and acknowledgements (ACKs) in the opposite direction. To implement the protocols, you need to write two programs: a sender and a receiver with the specifications given below. Communication to and from the channel emulator uses UDP. You can implement your programs in any of the following programming languages: C, C++, Java, Python, Ruby. However, do not use any libraries that substantially alter and/or enhance the basic socket interface as discussed in class. If in doubt, consult with the instructor.

Overview

The overall setup is shown in the diagram below with 'S', 'B', 'C', and 'R' denoting the various UDP sockets. See addressing below for further information.



When the sender needs to send packets to the receiver, it sends them to the channel emulator at 'B'. The channel emulator forwards the packets to the receiver at 'R'. However, it may randomly delay or discard packets. The receiver sends ACKs back to the channel at 'C', which may also randomly discard or delay ACKs before forwarding them to 'S'.

Details

Packet Format

All packets exchanged between the sender and the receiver must adhere to the following format:

Packet Type	32 bit unsigned integer, big endian (network order)
Packet Length	32 bit unsigned integer, big endian (network order)
Sequence Number	32 bit unsigned integer, big endian (network order)
Payload	byte sequence, maximum 500 bytes

The *Packet Type* field indicates the type of the packet. It is set as follows:

Type	Explanation
0	Data Packet
1	Acknowledgement (ACK) Packet
2	End-Of-Transfer (EOT) Packet (see below for details)

The *Packet Length* field specifies the total length of the packet in bytes, including the packet header. For ACK and EOT packets, the size of the packet is just the size of the header.

For data packets, the *Sequence Number* is the modulo 256 sequence number of the packet, i.e., the sequence number range is [0...255]. For ACK packets, *Sequence Number* is the sequence number of the packet being acknowledged.

Sender Program

You must implement both Go-Back-N and Selective Repeat in your sender program that takes three arguments: i) protocol selector – 0 for Go-Back-N or 1 for Selective Repeat, ii) the value of a timeout in milliseconds, iii) the filename to be transferred. The sender then transfers the file reliably to the receiver program. The timeout is used as the timeout period for the reliable data transfer protocol. During the transfer, the sender program must create packets as big as possible, i.e., containing 500 bytes payload, if enough data is available. After all contents of the file have been transmitted successfully to the receiver and the corresponding ACKs have been received, the sender must send an EOT packet to the receiver. The sender must exit after receiving the response EOT from the receiver. You can assume that EOT packets are never lost.

Receiver Program

You must implement both Go-Back-N and Selective Repeat in your receiver program. The receiver program takes two arguments: i) protocol selector – 0 for Go-Back-N or 1 for Selective Repeat, ii) the filename to which the transferred file is written. When the receiver program receives the EOT packet, it sends an EOT packet back and exits.

Both sender and receive will run using the same protocol: Go-Back-N on both ends, or selective repeat on both ends. (So, the sender should not try to send using Go-Back-N, while the receiver is expecting selective repeat).

Output

For both testing and marking purposes, your sender and receiver program must print log messages to standard output - a line for each data packet being sent and received (including duplicates) in the following format. You **must** follow this format to avoid problems during testing and marking:

```
PKT {SEND|RECV} {DAT|ACK|EOT} <total length> <sequence number>
```

For example:

```
PKT SEND DAT 512 17
PKT RECV ACK 12 17
```

Further, before your program executes a potentially blocking system call, it should print a message to standard output, describing the system call being made. The format of this message is not important, but it should **not** contain the string "PKT". If the program ever hangs, this output can help determine why.

Go-Back-N

In the Go-Back-N variant, if there is data available for sending, the sender sends data according to the current status of its send window. The send window size must be set to a fixed value of 10 packets. The sender uses a single timer. ACKs are cumulative up to the sequence number in the ACK packet. Follow the description of Go-Back-N as discussed in class.

Selective Repeat

In the Selective Repeat variant, if there is data available for sending, the sender sends data according to the current status of its send window. The send window size must be set to a fixed value of 10 packets. A separate logical timer must be kept for each packet. ACKs are not cumulative and only acknowledge the sequence number in the ACK packet. Follow the description of Selective Repeat as discussed in class.

Channel Emulator

The channel emulator is started with the following syntax:

- Linux binary:

  ```
  channel <max delay> <discard probability> <random seed> <verbose>
  ```

- Java version:

```
java -jar channel.jar <max delay> <discard probability> <random seed>
<verbose>
```

- All Data and ACK packets are subject to a random delay, uniformly distributed between 0 and `<max delay>` milliseconds.
- All Data and ACK packets are subject to random discard with a probability of `<discard probability>`.
- If `<random seed>` is set to a non-zero value, this seed is being used to initialize the random number generator. Multiple runs with the same seed produce the same channel behaviour. If `<random seed>` is set to zero, the random number generator is seeded with the current system time.
- If `<verbose>` is set to a non-zero value, the channel emulator outputs information about its internal processing.

Addressing

In order to keep addressing simple, but enable running the programs in a shared environment, the following addressing scheme is used with OS-assigned port numbers. The receiver program is started first and must write its 'R' socket address information (hostname and port number) into a file `recvInfo` that is read by the channel emulator. The channel emulator is started next and uses this information to send packets towards the receiver. The same mechanism is used between the sender and the emulator, i.e., the emulator writes its 'B' addressing information into a file `channelInfo` which is then read by the sender. All files are read and written in the current directory. The contents of each file are the IP address (or hostname) and port number, separated by space. Example:
`ubuntu1604-006.student.cs 38548`

Additional Comments/Hints

1. Unlike A1, A2 will be tested on a single machine i.e., the sender, the receiver, and the channel emulator will be executed on the same machine. The correctness of your programs will be evaluated on two perspectives:
    I. Executing *diff* on the sent and received files. If your protocol is correct, sent and received files will be the same even if the channel emulator adds delays and/or drops packets.
    II. Depending on the *timeout* of the sender and *max delay* and *discard probability* of the channel emulator, the number of sent packets will vary for the same file. For example, if the sent file's size is 10000 bytes, *timeout* = 5, *max delay* = 0, and *discard probability = 0.5,* the sender with selective repeat protocol should show *approximately* 80 PKT SEND attempts. Similarly, for *timeout* = 1, *max delay* = 200, and *discard probability = 0,* the sender with selective repeat protocol should show 1000-1200 PKT SEND attempts for the same file due to aggressive timeouts. In both cases, the sender with Go-back-N protocol should show roughly 10x PKT SEND attempts compared to selective repeat. Note that the numbers mentioned here are estimated ones. They may vary depending on the behavior of the channel emulator.

2. There are different ways to concurrently wait for an incoming packet and/or timer expiration. These include busy-looping, multi-threading, or using I/O multiplexing system calls, such as `select()`, `poll()`, or `epoll()`. You can choose either of these methods. Use any of the above methods to block while there is nothing to do immediately.
3. Avoid duplication of code. Instead, create reusable system modules that can be shared between the Go-Back-N and Selective-Repeat variants of the sender and receiver respectively.
4. Since UDP is used for data transmission, delivery to and from the channel emulator is not guaranteed. This should not matter for data or ACK packets and gracefully handled by the reliable data transfer. You can ignore the residual probability that this could result in a loss of an EOT packet.
5. Be aware that the channel emulator only forwards two EOT packets (one in each direction) and then exits!
6. **IMPORTANT:** Follow the hand-in instructions below carefully - or risk loosing a substantial number of marks!

Download

The channel program is provided in two versions: The Linux binary is the reference program for the `linux.student.cs` environment. The Java archive is provided for your convenience.

- Linux binary
- Java archive

Procedures

What to hand in

- Hand in source code files, including appropriate comments. All files must be stored in the same directory. **There should be** no directory hierarchy **or package definitions**. Your assignment **must** come with a Makefile. The targets in the Makefile must include:
  - 'clean' to remove all object files, addressing files, as well as all log and temporary files; and
  - 'all' to build all object and executable files
- Put all the above files into .tar in the format of <student_id>.tar
- After executing 'make all', the following executables (or start scripts) must exist in the current directory. The sender programs must accept a timeout in milliseconds and a filename as arguments. The receiver program must accept a filename:

```
Receiver <protocol selector> <filename>
Sender <protocol selector> <timeout> <filename>

<protocol selector> _ 0 for Go-Back-N and 1 for Selective Repeat
```

There must be two executable files with an exact name as above, i.e., Receiver and Sender.

- Your code can make use of standard libraries for your programming language. However the reliable data transfer protocols must not be taken from third-party code. Clearly document what you do. If in doubt, consult with the instructor.
- 'README' file (in plain ASCII text): Report which machines your program was built and tested on. Also, document which parts of the assignment have or have not been completed. Describe the basic design ideas/justifications for your program(s). This file does not need to be long, but should succinctly provide all the requested information.
- 10 marks will be deducted if you do not follow the instructions above.

Evaluation

The assignment is to be done individually. Your program **must work** in the `linux.student.cs` environment. Your program should not silently crash under any circumstances. At the very least, all fatal errors should lead to an error message indicating the location in the source code before termination. Marks will be assigned as follows:

- Functionality and Correctness: 80%
- Code Quality, Modularity, and Documentation: 20%

Submission Instructions

After you have completed testing your code, hand it in using the dropbox feature of the Learn environment. Combine all files into a zip/tar/gzip/bzip2 archive with any of the following corresponding names: a2.{zip,tgz,tbz}. Make sure to execute 'make clean' before submitting, and do not include temporary or object files!