Michael Chen & George Dong

[Mchen96@gatech.edu](mailto:Mchen96@gatech.edu) & [gdong8@gatech.edu](mailto:gdong8@gatech.edu)

CS 3251 – Homework 4

4 November 2015

# Introduction

A reliable transport layer protocol specification was designed in order to create a reliable end to end protocol over an unreliable network layer. The protocol is a go-back-n pipelined protocol that features both positive and negative acknowledgements.

The protocol uses a connection-oriented design, in which the client sends a request to the server in order to initiate a connection. Once the connection is established, no distinction is made between the client and the server; both sides are capable of either transmitting or receiving. Since the protocol makes no distinction between the server and the client once a connection has been established, the protocol is inherently bidirectional.

To handle lost packets, a timeout mechanism is used. If the sender of a package does not receive a positive acknowledgement from the receiver within a preset timeout, the packet is assumed to be lost; to compensate, the lost package should be resent. The exact package lost is identified using package sequence numbers; a more specific description of this functionality has been included in the next sections.

If a duplicate package is received, the package is discarded. However, a positive acknowledgement is also sent to the sender in order to confirm that the package was successfully received. This scenario may occur when the acknowledgement package is lost.

If an out of order package is received, the package is discarded. Furthermore, a negative acknowledgement must be sent back in order to notify the sender that one of the packages was either lost or delayed and that it should be retransmitted.

A simple addition checksum is used to determine if a package has been corrupted. If a package is determined as corrupted via the checksum, a negative acknowledgement is sent. The sender, upon receiving the negative acknowledgement, should retransmit the corrupted package and every single package that followed it.

# Package Structure

The package header was designed in order to provide the functionality required by the project constraints. Figure 1, which is shown below, shows the structure of the package header.

|  |  |  |
| --- | --- | --- |
| Sequence Number (32 bits) | | |
| Acknowledgement Number (32 bits) | | |
| Payload Length (16 bits) | | Available Window Size (16 bits) |
| Reserved (12 bits) | Flags (4 bits) | Checksum (16 bits) |
| Payload… | | |

Figure 1: Package Structure

A 32 bit integer contains the sequence number, which identifies the position of the first payload byte if a payload is present. The next 32 bit integer contains the acknowledgement number, which is the position of next expected byte when receiving a package.

The following 16 bit short contains the effective length of the data payload only. The size of the header field does not count towards the payload length.

The following 16 bit short states the available size of the receiving window. The available window size is defined as the difference between the total window size and the space occupied by all the received bytes whose acknowledgements have been sent, from the receiver’s perspective.

After 12 bits that are reserved for future expansion, the next 4 bits contains flags that are used to signal a package’s purpose. The possible values for these flags are, from least significant to most significant bit, ACK (positive acknowledgement), NACK (negative acknowledgement), SYN (synchronize sequence numbers), and FIN (Finalize connection). Figure 2 shows the order of these flags.

|  |  |  |  |
| --- | --- | --- | --- |
| FIN | SYN | NACK | ACK |

Figure 2: Packet Flags

# Checksum Implementation

The checksum algorithm selected consists of adding together all the bytes in both the header and the package payload. Each pair of bytes in the header (excluding the checksum) and the payload will be read as a short and added to the results of the preceding addition. If the payload has an odd number of bytes, the last byte will be right padded with zeros before the addition is performed. The result of the addition, truncated to a short, is the value of the checksum.

On the received side, the client performs the exact same addition. If the sum obtained differs from the sum included in the package headers, the package must be discarded and a negative acknowledgement must be issued to notify the sender that the package was corrupted.

# Finite State Diagrams

In order to simplify the finite state diagrams, the diagrams have been divided into two segments.

**2- Reliable Transport Protocol**

The RxP design specification will need to specify *at least*the following:

* -  a high-level description of how RxP works and of any special features you have designed
* -  finite state-machine diagrams for the two RxP end-points
* -  a formal description of your API (the functions it exports to the application layer)
* -  algorithmic descriptions for any non-trivial algorithms in RxP (e.g., how you detect corrupted packets)

In the first page of the report (after the cover page), please provide clear answers to the following questions: 

Your protocol design must support all features to receive full credit (e.g. window-based flow control, bi-directional data transfer) even if you don't plan to completely implement them all.  
  
Please note: you may modify the RxP protocol after you submit your design report. Your design report will be graded based on what you submit by the first deadline. You will need to submit a revised design report together with the final implementation. If you make changes to your design during the implementation phase, please make sure that you highlight the differences between your original design report and the modified design report that describes what you have actually implemented.