# ENTS 640 Networks and Protocols I
# Fall 2015
# Project

Assigned: November 5; Due: November 30, 5:00pm

## Overview

Write a distributed networking application in Java consisting of a transmitter and a receiver that can ensure reliable data transfer and cryptographic authentication. The application should use Java's UDP sockets (classes `DatagramPacket` and `DatagramSocket` and their methods) and provide the necessary reliable data transfer functionality on the top of UDP's unreliable communication services by implementing the data transfer protocol described below. The data transfer should be one-directional with data flowing from the transmitter to the receiver.

Let us define the maximum payload size (MPS) as the length of the largest data packet payload (*not* including the header) that can be sent by the protocol, and set its value to 30 bytes.

## Message Structure

The data packets sent by the transmitter should have the format shown in Figure 1.

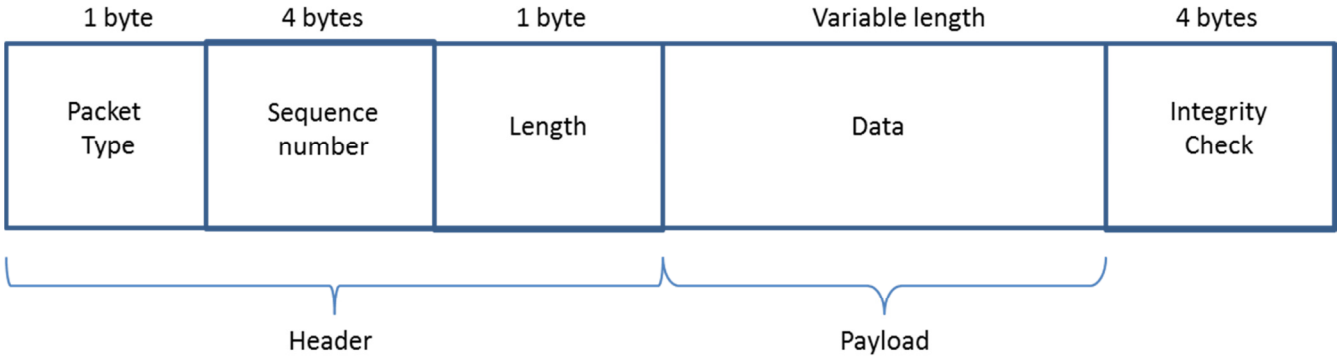


Figure 1: Data packet structure

The data packet should have the following fields ('h' stands for hexadecimal):

- Packet type (1 byte): This field describes the type of the packet. Its possible values are:
  55h : Data packet with additional packet(s) to be sent later
  aah : Last data packet (the payload contains the last part of the data)
- Sequence number (4 bytes): The ordinal number of the first sent byte in the payload of the current data packet. (The protocol counts bytes, and not packets.) It should start from a random value previously agreed by the transmitter and the receiver and be incremented based on the number of sent bytes for each sent packet.
- Length field (1 byte): The length of the data (payload) in bytes.
- Data (variable length): The carried payload as a sequence of bytes.
- Integrity check field (4 bytes): The integrity check value provides error detection and authentication ability. Its value should be calculated as described later.

Note: the multi-byte fields should be transmitted in big endian order, i.e. the most significant byte is transmitted first, and the least significant byte should be transmitted last.

The acknowledgment packets sent by the receiver should have the format shown in Figure 2.

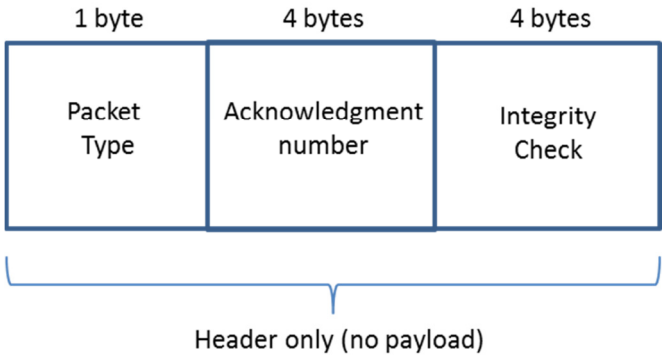


Figure 2: Acknowledgment packet structure

The acknowledgment packet should have the following fields:

- Packet type (1 byte): This field describes the type of the packet. It should be set to ffh.
- Acknowledgment number (4 bytes): The ordinal number of the next expected byte from the sender. The protocol uses cumulative acknowledgments, so this means that all data bytes with sequence numbers up to, but not including this acknowledgment number have been received correctly.
- Integrity check field (4 bytes): The integrity check value provides error detection and authentication ability. Its value should be calculated as described later.

## Protocol Operation

The system should work according to the following description. First, you should generate a 128-bit random secret key, and a 32-bit initial sequence number, and provide this information to both the transmitter and the receiver. You also need to set up the transmitter and receiver UDP socket parameters (IP addresses, port numbers etc.) so that the two sides could communicate with each other.

The transmitter should generate 500 bytes of random data and send it to the receiver using the following steps:

1. The transmitter should form the next data block form the data in such a way that the payload is as large as possible, but its length is less than or equal to MPS.
2. For each data block, it should create a data packet by prepending the payload with the packet header and appending the integrity check field. The integrity check field should be calculated over both the header and the payload and included in the packet at the end. The last packet should have a special packet type field (aah), informing the receiver that there are no more packets to send.
3. Each packet should be sent to the receiver using the stop-and-wait protocol. The transmitter should send a data packet, and wait until the sent data packet is acknowledged by the receiver before sending the next packet. An acknowledgment packet can only be accepted if:

   a) The locally calculated integrity check value for the acknowledgment packet is equal to the value of the integrity check field in the acknowledgment packet.
   b) It is an acknowledgment packet (packet type: ffh) with the correct acknowledgment number.
4. All other received packets should be discarded. The transmitter should also start a timer upon sending each data packet, and retransmit the packet if the timer expires. The initial timeout value should be set to 1 second and should be doubled at each timeout event. After the 4th timeout event, the transmitter should declare communication failure and print an error message. If an acknowledgment is received for a packet after some timeout events, the timeout value and the timeout counter should be reset to their initial values.
5. Steps 1. – 4. should be repeated until all the data has been transferred to the receiver.

The receiver protocol should operate according to the following description:

1) The receiver should start with the previously provided initial expected sequence number.
2) The receiver should receive each data packet, and send acknowledgment packets (packet type: ffh) for each correctly received data packet. A data packet is correctly received only if:
   a) The locally calculated integrity check value for the data packet is equal to the value of the integrity check field in the data packet.
   b) It is a data packet (packet type: 55h or aah) with the correct (in-order) sequence number.
   c) The length of the payload is less than or equal to MPS.
3) All other received packets should be discarded.
4) After receiving the last data packet (packet type: aah) and sending acknowledgment for it, the receiver should display the received byte sequence on the screen.

## Integrity Check

In this protocol, the integrity check serves two purposes: a) it provides a mechanism for bit error detection, and b) it provides authentication for the sender party by using encryption with a secret key that only the sender and the receiver know. This will make forging and/or replaying a message difficult, as the attacker does not know the secret key, so he/she cannot inject false messages into the data stream. (Note: this protocol does not provide secrecy, so an eavesdropper can still read and interpret the data if the packets are captured.) The integrity check calculation is based on the RC4 encryption algorithm, so please study the supplied additional reading material on this topic as you will need to write your own RC4 implementation.

The integrity check value should be calculated as follows:

1) A byte sequence should be formed from all bytes in the packet (both header and payload), except for the integrity check field.
2) The byte sequence should be zero-padded such that the length of the padded sequence is a multiple of four.
3) The padded byte sequence should be encrypted using the RC4 algorithm with the 128-bit secret key previously agreed on by the transmitter and the receiver.
4) The encrypted byte sequence should be compressed to 4 bytes. Let us denote the encrypted byte sequence by b[0], b[1], ...., b[N-1]. Then, C[0], the first byte of the integrity check value, should be calculated as the bit-wise exclusive or (XOR) of the bytes b[0], b[4], b[8], ..., i.e. every fourth byte starting from the first byte. C[1], the second byte of the integrity check

value, should be the XOR of the bytes b[1], b[5], b[9], ....(every fourth byte, starting from the second byte). And so on, the rest of the integrity check bytes C[2] and C[3] should be calculated similarly as the XOR of the bytes b[2], b[6], b[10], ... and the bytes b[3], b[7], b[11], ...., respectively. Figure 3 illustrates this process. The obtained C[0] will be the MSB and C[3] will be the LSB of the integrity check value.

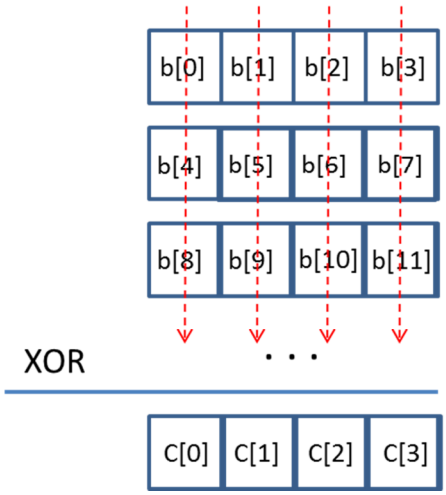


Figure 3: The integrity check calculation process

## Hints

1. The integer data types in Java are all signed, so be careful with converting between those types and performing arithmetic operations with those types.

## Deliverables and Deadline

Each goup needs to submit the Java source code (in the form of .java source files) implementing the above described protocol. In addition, each group should write a project report (10-15 pages) discussing the problem, their design and solution (for example, flow charts, block diagrams, UML class diagrams, state transition diagrams, etc.), and the application's output, demonstrating the implemented functionality.

The perfect solution will receive 50 points (50% of the final grade). All material shuld be submitted electronically as email attachment (to zsafar@umd.edu), one per group, by Novembre 30, 5:00pm in a single email. Late submissions will receive reduced points as follows:

- One day late (before December 1, 5:00pm): 50% reduction – maximum 25 points
- Two days late (before December 2, 5:00pm): 80% reduction – maximum 10 points
- More than two days late: 0 points

Please note that in addition to correctness and functionality, the Java code will also be evaluated for coding style. Thus, you should pay attention to software design/engineering issues: code modularization, class design, code block organization, class and variable naming, comments, etc.

The last note is on academic integrity: Each group is welcome to discuss the project in general terms with other groups or students. However, when it comes to a particular solution to a particular problem arising during the project work, it must be a result of the work of only the group members. **Copying code from other groups is strictly forbidden and will not be tolerated.**