ET2440: "TCP/IP"

Project description

Blekinge Institute of Technology
School of Computing

1 Introduction

In this project, you will explore several of the control mechanisms used in transport layer protocols. You will implement a TCP-like transport protocol over User Datagram Protocol (UDP), and optionally compare the performance of your implementation of the various control mechanisms to the ones implemented in your operating system.

1.1 Control Mechanisms

The Transport Control Protocol (TCP) implements several control mechanisms to provide a reliable stream-oriented (or connection-oriented) data transport service. It also provides multiplexing between processes on participating hosts using *ports*. The most important control mechanisms are *error control*, *flow control* and *congestion control*.

1.1.1 Error Control

Error control in the transport layer (for a reliable transport protocol) is necessary as the network layer protocol (IP) does not provide this functionality. Error control entails making sure that the Transport Protocol (TP) can handle missing, corrupted, duplicated packets and packets arriving in the wrong order.

1.1.2 Flow Control

Flow control is the mechanism that makes sure that the receiving end of a transmission is not overloaded with more data than it can consume. Flow control is needed only when the sender sends data without being explicitly asked to do so (pushing).

1.1.3 Congestion Control

Congestion control is the mechanism by which the TP tries to avoid overloading the network. This can be done by either *avoiding* congestion, i.e., open-loop control, or by dealing with congestion after it has occurred, i.e., closed-loop control.

2 Project Task

Your task for this project is to implement and evaluate the BTH UDP-based Transport Protocol (BUTP), a simplified TCP-like protocol that provides flow, congestion and error control, using UDP as the underlying unreliable transport mechanism.

Unfortunately, the original protocol designer had to leave the protocol mostly unfinished, as he started a salmon farming business in the Arctic. However, he did provide a protocol header description, which is described in the next section.

The rest of the protocol is up to you to design and implement a proof-of-concept of, using what you've learned in the TCP/IP course. In particular, chapters 13 and 14 are useful for the theoretical foundations of TCP-like protocols, and lab 2 in this course should provide you with most of the socket code you will need for this project.

2.1 Protocol Header Description

The BUTP Protocol Data Units (PDUs) are called *chunks*, and the BUTP chunk format is defined in Figure 1. There are three protocol fields in the header. The BUTP header is always 10 bytes (or octets) long.

Sequence Number: 32 bits
The sequence number of the first data octet in this chunk.

Acknowledgment Number: 32 bits
This field contains the value of the next sequence number the sender of the chunk is expecting to receive.

Checksum: 16 bits
This field contains the 16-bit checksum of the *data* portion of the chunk. This is in contrast to the regular TCP, where it is calculated over the data plus a pseudo-header.

Options: 16 bits
This field is free to use for your own purpopses. Flag bits, etc.

Window, 32 bits
This field indicates the window size.

Data The remainder of the chunk is the data portion of the chunk.

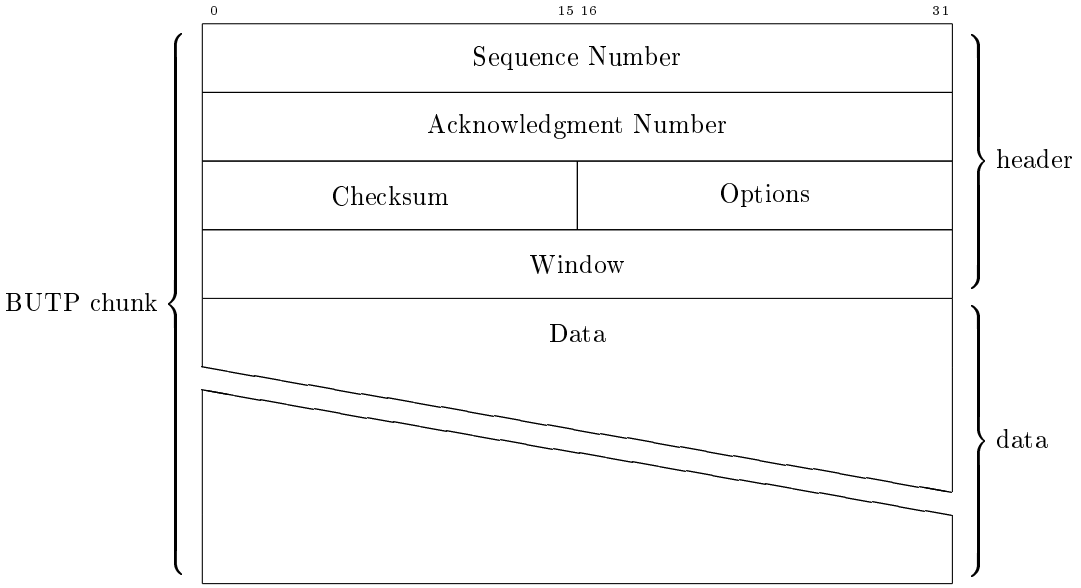

Figure 1: BUTP chunk format.

2.2 Implementation Notes

Implementing a reliable TP is not a trivial, and we will allow you to make simplifications.

1. Process multiplexing and demultiplexing is neither supported in the protocol header nor required.
2. There is no three-way handshake, as in TCP. You may assume that a connection has already been made, i.e., in the equivalent TCP state of `STATE_ESTABLISHED`.

3. Packet sizes never change when re-transmitted. This means that you don't have to handle overlapping packets/sequence numbers.

4. As it is highly unlikely that the network you'll be running your protocol over will experience any losses and significant delays, you'll need to fake this in your program to show how your control algorithms work. You may, e.g., randomly drop packets, randomly delay packets a random amount of time, randomly send corrupted data.

2.3 Project Deliverables

To pass the project, you *must* provide:

- Source code and any needed compilation instructions.
- A *brief*[1] implementation report, and description of the algorithms you've chosen, and any parameters for these algorithms, e.g., how does your congestion window grow and shrink, how long is the retransmission timer, etc. Using state diagrams are encouraged and recommended.
- Clear and unambiguous proof that your algorithms actually work.

You *may*, but are recommended to, also provide a performance study of your protocol, taking into account, e.g.,

- How does your protocol affect normal TCP flows?
- What are the performance benefits of your protocol, compared to TCP?

[1] According to the dictionary, *brief* means "concise in expression; using few words". This implies that you should be short, succinct, and to the point in writing. Recall that the important thing is to get your point across, not impressing with long words or sentences.