HIPSTER, the alternative file transfer protocol

1 Introduction

In this report we will describe the *Homework Inherent Protocol Such That Efficiency Rises* (HIPSTER), a file transfer protocol designed to work on top of the UDP transport layer. The aim of the protocol is to minimize transfer time over an unreliable `Transport Channel` (TC), which introduces delay and drops packets according to a certain model. In the first section, the protocol will be described. Then, we will show the results of some tests for the TC, demonstrating that it works as specified by the model. In the last section test results for HIPSTER protocol will be presented and we will show that the system reaches low transfer times.

2 Protocol description

```
0             15|16    26|27   31
|-------------------------------|
|     DESTINATION IP (ipv4)     |     // Destination:   0 to 31
|-------------------------------|     // Dst Port:      32 to 47
|   DST PORT    | LENGTH | CODE |
|-------------------------------|     // Length:        48 to 57
|        SEQUENCE NUMBER        |     // Code:          58 to 63
|-------------------------------|     // SN:            64 to 95
|                               |     // Data:          96 to (96+Length)
|             DATA              |
|                               |
|-------------------------------|
```

Figure 1: HIPSTER packet structure

The packet structure has been designed in order to add the minimum overhead, while providing all the functionality needed by the protocol.
The Destination IP and port are needed for packet forwarding within the channel and for the receiver to correctly reply to the sender. The first two fields are the same within different groups. Moreover, the size of the payload was included to ease the parsing of the packet at the receiving side. Finally, `CODE` and `SEQUENCE NUMBER` are used for signalling and error recovery. The following values are used for the code:

- 0 means a regular data packet;
- 1 means the packet is an ACK, and the sequence number is the same as the packet being ACKed. There is no payload;
- 2 signals the end of transmission (ETX). It is issued by the sender and ACKed by the receiver.

Before the actual implementation a theoretical model of the protocol and the channel was developed. Such model showed that the optimal packet length to minimize the transfer time is around 1000 bytes, so this protocol uses fixed-size packets whose payload length is 1000 bytes.
The behaviour of the `Data Sender` (DS) is depicted in Figure 2. It is organized into two threads: the ListenerThread receives, decodes and enqueues ACKs, while SendThread is in charge of sending all packets successfully and performing some basic congestion control using the enqueued ACKs.

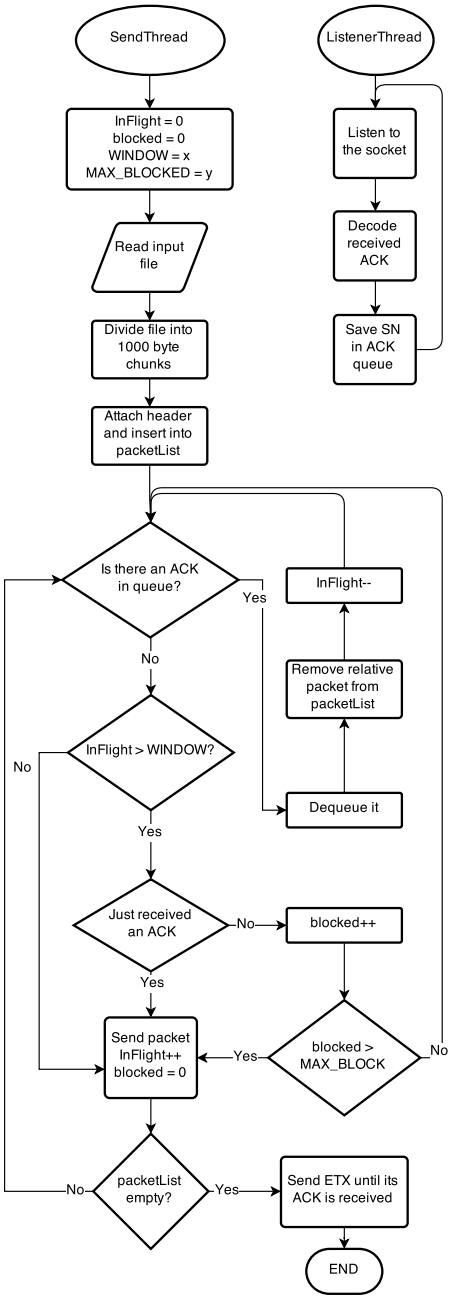

Figure 2: HIPSTER protocol DS

The `inFlight` counter in SendThread is incremented every time a packet is sent and decreased when an ACK is received. DS sends packets continuously until `inFlight` has reached the maximum allowed window. Once the window is full the sender will not send packets continuously, but will instead send a packet every time an ACK is received. The `blocked` counter is used to avoid deadlocks in case some ACKs are lost: it sets a maximum delay before a packet is sent regardless of the received ACK. Every time an ACK is dequeued the corresponding packet is removed from the list of packets. Only when the list is empty the transmission is considered complete. At this time DS sends the ETX packet, which signals to the receiver that the transmission is over. If the corresponding ACK is received the sender will exit otherwise if a 2.5 s timer expires the ETX is sent again.

The `Data Receiver` (DR) is in charge of handling ACK transmission and reordering the received packets. The protocol assumes the random delays and drops introduced by the channel would make in-order delivery extremely slow, so the decision of shifting the burden of reordering on the receiver was made. The DR listens continuously for new packets. Once it receives a packet, it checks its CODE. If the CODE is 0 (DATA), the DR stores the packet in a list and sends an ACK to the sender through the TC. In the case the received packet contains an ETX (end of transmission) code, the receiver acknowledges it, then proceeds to reorder the packets and finally writes the received file to disk. The ACKs crafted by the receiver contain an empty payload in order to minimize the channel drop probability, and the sequence number refers to the single packet the receiver intends to acknowledge, in order to inform the DS that retransmission of that portion is not necessary.

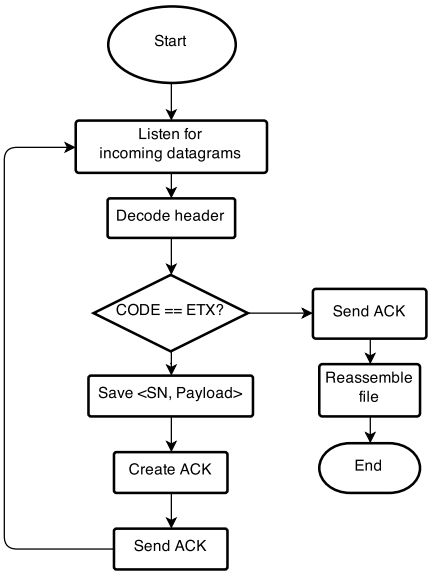

Figure 3: HIPSTER protocol DR

3 Transport Channel test results

The `Transport Channel` (TC) module simulates a bad channel among the DS and DR. It drops received UDP packets of length L with probability $P_{drop} = 1 - \exp(-L/1024)$ and forwards the remaining ones with a delay distributed according to an exponential random variable with mean $1024/\ln(L)$ ms. The TC was tested in loopback, by measuring the delay between the transmission of a packet by DS and its reception at DR. The ratio between sent and received packets was also measured. The payload of UDP packet used in this test is in the range from 12 byte (HIPSTER header length, thus the size of an ACK) to 1012 byte (actual size of HIPSTER packets). Each measurement was taken 10 times. The results are in Figures 4 and 5. The tests show that TC follows accurately the theoretical model, as the few discrepancies are related to the Java random number generator and the finiteness of measurements. The distribution of the delays introduced by the TC in a transmission fits the required exponential RV according to the Kolmogorov-Smirnov test and the comparison of the cumulative distributive functions (CDF) of the two can be seen in Figure 6.

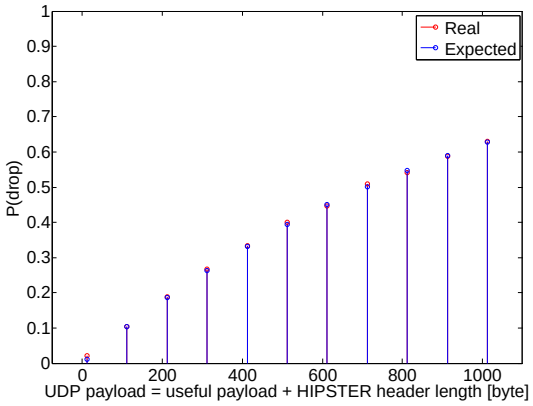

Figure 4: Dropping probability

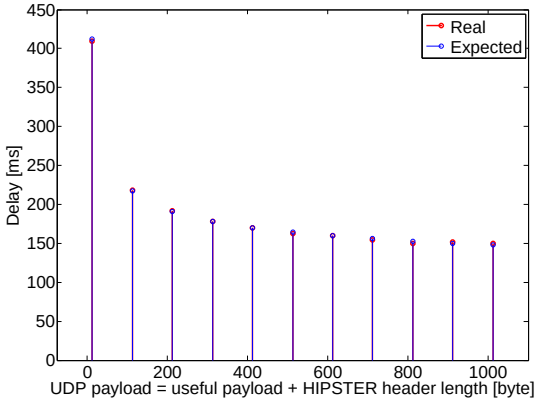

Figure 5: Mean TC delay

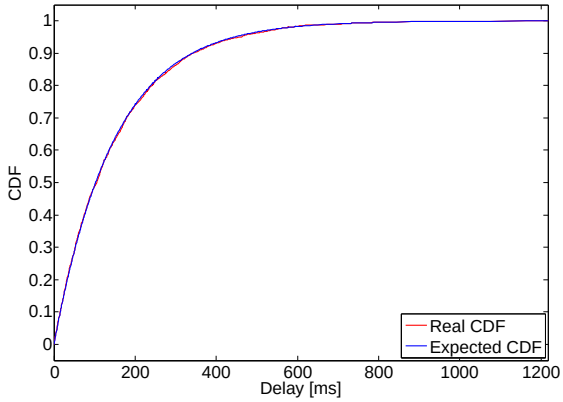

Figure 6: CDF of delays introduced by TC, $L_{UDP} = 1012$ byte

4 HIPSTER protocol test results

The transfer rate for files with different sizes was analyzed. Table 1 summarizes the mean over five runs for the transfer time and speed over `localhost`, while Table 2 reports the results of the same tests with another channel, borrowed from a different group. *Mean time* is the time since the beginning of the transmission at the sender until the reception of the ACK for the ETX packet, while *mean goodput* is file size over *mean time*.

Size (MB)	Mean time (s)	Mean goodput (MB/s)
500	64.92	8.07
200	23.86	8.78
100	13.48	7.78
50	7.83	6.7
10	2.57	4.07
5	1.55	3.39
1	0.67	1.56

Table 1: Mean transfer time and goodput with our channel

Size (MB)	Mean time (s)	Mean goodput (MB/s)
500	62.63	8.37
200	23.54	8.9
100	14.04	7.47
50	7.85	6.67
10	2.4	4.34
5	1.56	3.35
1	0.67	1.56

Table 2: Mean transfer time and goodput with a channel from a different group

Tests in a Local Area Network yield similar results. The transfer rate over different ADSL and WiMAX links has also been tested. In this scenario the algorithm needed some tuning of the `MAX_BLOCK` constant due to excessive congestion. Such variable has to be adapted to different networks. Bigger values (in the order of 10^5) are better suited for slow networks with longer RTTs. Smaller values instead (e.g. the default 128) are better suited for fast networks with RTT of about 1 ms.

The goodput at the Receiver over time is in Figures 7 and 8. They show the number of useful bits received every 500 ms, normalized to this sampling time, for a 200 MB file and 500 MB transmission. Both the figures show a common trend, with high goodput at the beginning, which decreases once transmission is almost completed, because of the retransmission mechanism and the long delays for the ACKs in the TC. The sudden drops of the goodput, instead, signal that congestion control mechanism has kicked in.

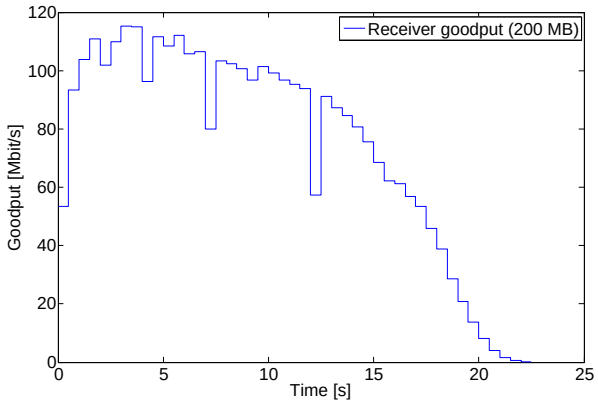

Figure 7: Goodput at the receiver, file size 200 MB

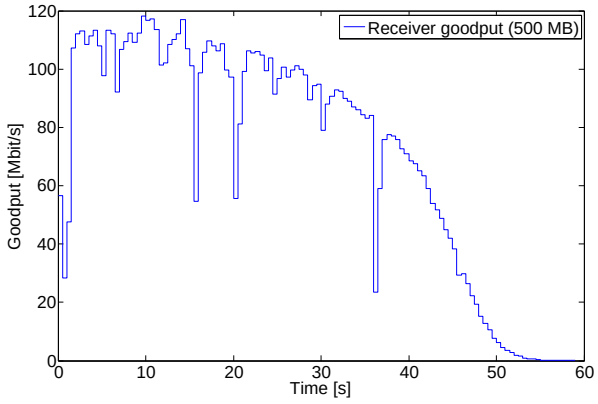

Figure 8: Goodput at the receiver, file size 500 MB

5 Conclusions

The HIPSTER protocol was designed in order to work upon networks with high round trip time and packet loss. Given the model of the HIPSTER sender, receiver and TC, we developed a theoretical analysis in order to tune packet length. The implementation of the TC was tested and performed as expected. Finally, tests with files of different sizes showed that the protocol can reach high goodput and low transfer time.