Course Transport Protocol (CTP)

CPSC 3780, Instructor: R. Benkoczi

Spring 2021

1 Acknowledgment

The reference implementation for this protocol as well as the design for the protocol is the creation of Dr. Olivier Tilmans, formerly with UC Louvain, now at Nokia Bell Labs. This project description is translated and adapted from a project description by the same author, in French.

2 Project description

Your task is to implement the *course transport layer* protocol, CTP, described below. The protocol allows the rapid and reliable transfer of one file from one computer to another. CTP uses the *selective repeat* sliding window algorithm. The protocol does not use special acknowledgments to specify the out of sequence segments received, but can use negative acknowledgments for this purpose. CTP uses UDP segments over IPv4.

CTP is inspired by a proposal for a network stack optimized for data centres which appeared in a research article [2][1]. A *special feature* of the network layer in the proposal is to strip off the payload from data segments when the buffers of the network switches get congested and to deliver only the headers which occupy much less space. This situation happens, for example, during bursts in traffic which are frequent in data centres. For this project, truncating data segments is simulated by a proxy server application called `link_sim`, whose binary is available on the course space in the school network, which can be accessed using your CPSC 3780 computer account.

A *Wireshark* dissector is also available from the course space in the school network, and will help with debugging. A reference implementation of the protocol as a binary and a sample PCAP capture file are also available to use and experiment with.

The project must be completed in teams of two members. You must implement two programs, a `sender` and a `receiver`. The sender and receiver transfer one file between two computers connected to the internet.

3 Specification

3.1 Protocol

The structure of the segments for CTP is shown in Fig. 1. A segment contains the following fields:

[1]The paper is an interesting read, but it is not necessary for the completion of this project.

Type This field is encoded on 2 bits. It indicates the type of the segment. Three types are possible:

(i) `PTYPE_DATA` = 1, indicates a packet containing data.

(ii) `PTYPE_ACK` = 2, indicates an acknowledgment packet.

(iii) `PTYPE_NACK` = 3, indicates a negative acknowledgment of a data packet whose data was stripped off (thus the data packet had the field TR set to 1).

Any other segment containing a different value **MUST** be ignored.

TR This field is represented by 1 bit and indicates that the network has stripped off the payload from a data segment `PTYPE_DATA`: receiving a `PTYPE_DATA` segment with field TR set to 1 requires the receiver to confirm with a `PTYPE_NACK` packet. Notice that the sender will never set the TR field of a `PTYPE_DATA` packet to 1 explicitly since this field is set by the network (simulated by `sim_link` in our case). Any packet other than `PTYPE_DATA` with TR set to 1 **MUST** be ignored.

Window This field is an integer represented on 5 bits in the range [0 .. 31]. It indicates the size (expressed as the number of segments) of the receiving window of the source host. More precisely, the field indicates the number of **empty or available segments** in the receiving window of the source host. This value can change with time. If the source has no receiving window, the value of field Window **MUST** be zero.

The sender CAN SEND a NEW data packet only if the destination host has previously replied with a `PTYPE_ACK` or `PTYPE_NACK` with NON ZERO Window field. When a new connection is created (and the sender sends the first `PTYPE_DATA` packet), the sender **MUST** *assume* that the receiver has advertised an initial Window value of 1 (one)[2].

Seqnum This field is an 8 bit integer with value in the interval [0 .. 255]. Its meaning depends on the type of packet.

`PTYPE_DATA` It corresponds to the sequence number of the data packet. The sequence number of the first segment of a connection is 0 (zero). If this sequence number does not belong to the set of sequence numbers within the sliding window of the destination host, the destination host **MUST** ignore this packet.

`PTYPE_ACK` It corresponds to the number of the next sequence expected, i.e., the last sequence number received + 1) (mod 2^8). It is possible to send a single `PTYPE_ACK` packet to acknowledge several `PTYPE_DATA` packets (the principle of cumulative acknowledgments).

`PTYPE_NACK` It corresponds to the sequence number of the truncated `PTYPE_DATA` packet received. If this sequence number is not in the range of the sequence numbers covered by the sender's sliding window, the packet **MUST** be ignored.

Once the source of the packet reaches sequence number 255, the following sequence number is 0 (zero).

[2]This means that the sender can only send one `PTYPE_DATA` packet initially until it receives an acknowledgment with an updated Window field.

Length This field is a 16 bit integer in network-byte-order, with the value in the range [0 .. 512]. It represents the number of bytes in the payload of the packet.

A `PTYPE_DATA` packet with value 0 for this field for which the sequence number equals the sequence number of the last acknowledgment sent by the receiver indicates to the receiver that the data transfer is completed. A possible truncation of the packet (TR field equals 1) will not change the value of the Length field and will still indicate the end of the transfer. If Length is greater than 512, the packed **MUST** be ignored.

Timestamp This field takes 4 bytes and represents an unspecified value with unspecified endianness. For each `PTYPE_DATA` packet sent, the sender chooses a value for this field. For a `PTYPE_ACK` packet, this value contains the Timestamp value from the **last** `PTYPE_DATA` packet received[3].

The significance of this field is at the discretion of the programmers.

CRC1 This field is represented over 4 bytes, in network-byte-order. It contains the value obtained by applying the CRC32 algorithm[4] to the header **with field TR set to 0** [5]. When a packet is received at the destination, the CRC1 field **MUST** be calculated with field TR set to 0 and the packet **MUST** be ignored if the calculated and stored values differ.

Payload This field uses maximum 512 bytes. It contains the data transferred by the protocol. If TR is zero, the size of Payload is given by field Length, otherwise, its size is 0.

CRC2 This field is represented over 4 bytes in network-byte-order. It contains the result of the CRC32 function over the payload, if present, before the packet is transmitted. The field is present only if the packet contains a payload and it is not truncated (the type is `PTYPE_DATA` and TR must be 0). When a packet is received and this field is present, the CRC32 value must be calculated over the payload and the packet **MUST** be ignored if the calculated and the stored CRC32 value are different.

3.2 Network

The following assumptions about the network can be made:

1. A segment once sent will be received at most once (**segments may be lost** but there are no duplicate segments).
2. The network may **corrupt the segments** in an arbitrary fashion.
3. The network may truncate the payload from `PTYPE_DATA` segments in an arbitrary fasion.

[3]The meaning of the word **last** is vague in the original documentation. Is it last as in last sequence number, or is it last as in most recently received?

[4]The divisor is the polynomial $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$, which is represented, in network-byte-order by the hex value $0 \times 04C11DB7$. CRC32 is implemented in `zlib.h`, but other implementations exist.

[5]Generally, `PTYPE_DATA` packets are truncated by the network switches/routers but the CRC1 field is calculated by the source host and it is always 0 originally. Without this requirement, the network switches would need to recalculate the CRC1 field when they truncate the packet.

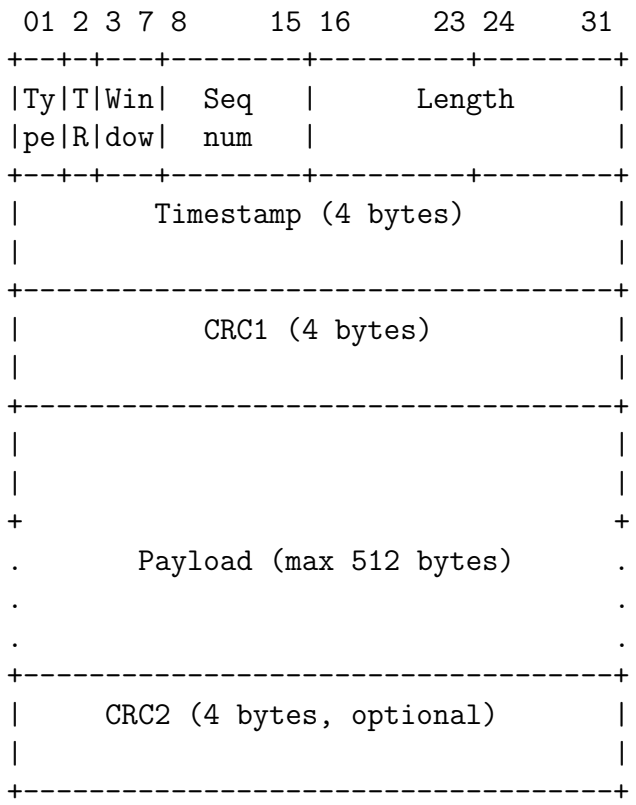

Figure 1: CTP packet format

4. Suppose two segments S_1 and S_2 are transmitted one before the other. The **order** in which S_1 and S_2 are received **is not guaranteed**.

5. The network **delay** to deliver a packet varies in the interval [**0 .. 2000 ms**].

3.3 Executables

To implement the protocol, you will create two computer programs named `sender` and `receiver`. The programs should accept the following command line arguments:

```
sender [-f data_file] host port
receiver [-f data_file] port
```

host: an IPv4 address or domain name. For the sender, it is the address of the receiver.

port: The port number used by the protocol. For the sender, it is the port number where UDP packets will be sent. For the receiver, it is the port number where the receiver process listens to. The Wireshark dissector provided will examine packets sent or received from ports 64341, 64342, and 12345, so use one of these ports if you plan to capture and examine the packets with Wireshark.

data_file: An optional parameter containing data to be transferred. The file may be a binary file.

Examples (run in separate command windows on the same host):

```
receiver -f data-received.txt 64341
sender -f data.txt 127.0.0.1 64341
```

Notes IPv4 address 127.0.0.1 is the loop-back address. It is an address that refers to the own host, so the example shows the case when the sender and the receiver are on executed on the same host.

3.4 The reference implementation and project resources

(1) Location: log into the CS network at school using the credentials that were e-mailed to you. The resources for this project (reference implementation of the protocol, wireshark dissector, sample capture file, some sample C++ threaded code examples) are available from `/home/lib3780/project/`

(2) Running the reference implementation. The reference implementation is currently implemented so that it runs on IPv6 only. The implementation will be modified so that it works with IPv4. Once the porting is complete, this will be announced on the course mailing list.

Details about executing the reference implementation will be posted here.

3.4.1 Simulating an imperfect network

TBD.

4 Schedule and deliverables

Checkpoint 1: Using the information in [1, 5, 4], develop a first prototype for sender and receiver that will transfer a single UDP segment from sender to receiver. The structure of the segment is not specified. It may simply contain a text message read from standard input such as "Hello world". The receiver should output this message to the standard output if no *-f* argument is given. Any errors or debug info should be written to the *error stream*, `std::cerr` in C++. The two applications **MUST** accept the command line arguments as defined above. In case -f argument is used, the packet will contain an appropriate portion of the file as data.

Deliverables (upload on Moodle):

(a) Report describing your setup for an experiment that gives evidence the tasks above were completed. Ideas: you can include log files, links to MS Stream short videos, etc.

(b) Your source code as a zip file with a Makefile or a readme file with instructions for compiling the project.

Due: Sun. March 14, 2021

Marking scheme (total 10 pts):

- The packet is sent and received. Netcat can be used to supplement one of the endpoints in the test. 5 pts
- Port number accepted as command line argument. 2 pts
- Host name or address accepted as command line argument. 1 pts
- File name accepted as command line argument. 2 pts

Checkpoint 2: Extend your program by developing the common functions needed to pack and unpack the protocol headers. These functions or classes will be used by both you sender and receiver application. Use a unit testing framework for the programming language you are coding, like CPPUnit [3]. Examples using Google test library and C++ are available under `/home/lib3780/project/unit-tests` on the CS department network.

Suggestions (you are free to create your own design):

- Create a class with member functions that set and get various protocol fields in the packet.
- The class should have a member function that returns a pointer to the buffer containing the header and payload.
- The class should maintain a buffer with an appropriate size and it should export a pointer to the buffer which can be used in socket receive operations. Recall that multi-byte fields are stored in network byte order in the buffer.
- Write a separate test case for setting and getting each field in the header. Test the field with explicit bit manipulation expressions, not using the corresponding setter or getter class function.

Deliverables (upload on Moodle):

(a) Report providing evidence that your code passes the test cases and that your test cases cover the set and get operations for all fields in the CTP header.
(b) The report should also explain how the binaries (sender/receiver) and the test cases are built.
(c) Source code and Makefile (or an equivalent mechanism) that builds the sender and receiver version from Checkpoint 1 and the test cases.

Due: (see Moodle deadlines)

Marking scheme (total 10 pts):

- There are nine header fields, of which eight are specified and one is left to the interpretation of the coders. Two test cases for each of the eight fields must be passed. 8 pts.
- Clarity of the report (explaining how the targets can be built, etc.) 2 pts

Checkpoint 3: Transfer of a file between sender and receiver assuming no errors occur. Extend the sender and receiver application so that it transfers CTP packets correctly. In case an error does occur, the programs may print an error message and abort the execution.

Deliverables (upload on Moodle):

(a) Report providing evidence that the transfer of a file succeeded (script file, debug messages, diff, etc.)
(b) Source code and Makefile (or an equivalent mechanism for building the programs).

Due: (see Moodle deadlines)

Marking scheme (total 10 pts):

- The project sender and project receiver transfer a file correctly. 5 pts.

– Project sender and the reference implementation receiver transfer a file; Reference sender and project receiver transfer a file 5 pts

Checkpoint 4: Final report: transferring files under the presence of errors.

Deliverables (upload on Moodle):

(a) Project report describing the general architecture of your programs. The report must answer at least the following questions.
   (1) What do you enter in the *Timestamp* field and how youuse the information.
   (2) How do you react to the receipt of `PTYPE_NACK` packets?
   (3) How did you choose the value of the re-transmission timer?
   (4) What is the critical component of your implementation which affects the transfer speed?
   (5) What is the performance (speed) of your protocol? Provide measurements, experiments, etc.

(b) Source code and Makefile (or an equivalent mechanism that builds the programs and test cases).

Due: (see Moodle deadline)

Marking scheme (total 20 pts):

– Project report. 10 pts.
– Evaluation of the transfer under the following conditions: packets may be lost, packets may be corrupted, packets may be truncated, packets may be delayed. You must provide evidence that the programs were tested under these conditions. 10 pts

References

[1] Brian "Beej Jorgensen" Hall. Beej's guide to network programming, 2018. URL: `https://beej.us/guide/bgnet/html/multi/index.html`.

[2] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W Moore, Gianni Antichi, and Marcin Wójcik. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 29–42. ACM, 2017, available at `http://conferences2.sigcomm.org/sigcomm/2017/program.html`.

[3] Arpan Sen. Get to know cppunit, 2010. URL: `https://www.ibm.com/developerworks/aix/library/au-ctools2_cppunit/index.html`.

[4] Graham Shaw. Listen for and receive udp datagrams in c. URL: `http://www.microhowto.info/howto/listen_for_and_receive_udp_datagrams_in_c.html`.

[5] Graham Shaw. Send a udp datagram in c. URL: `http://www.microhowto.info/howto/send_a_udp_datagram_in_c.html`.