Iowa State University
Department of Electrical and Computer Engineering

CprE 489: Computer Networking and Data Communications
Fall 2018

Course Project
<u>Project report</u> due: November 29, 2018 (in class or on Canvas by 11am CST)
<u>Project code</u> due on Canvas: November 30, 2018 (5pm CST)
No late reports or late codes will be accepted

You are allowed to form groups of no more than two students per group

Objective:

To develop a simplified implementation of the Multipath TCP (MPTCP) protocol.

Background:

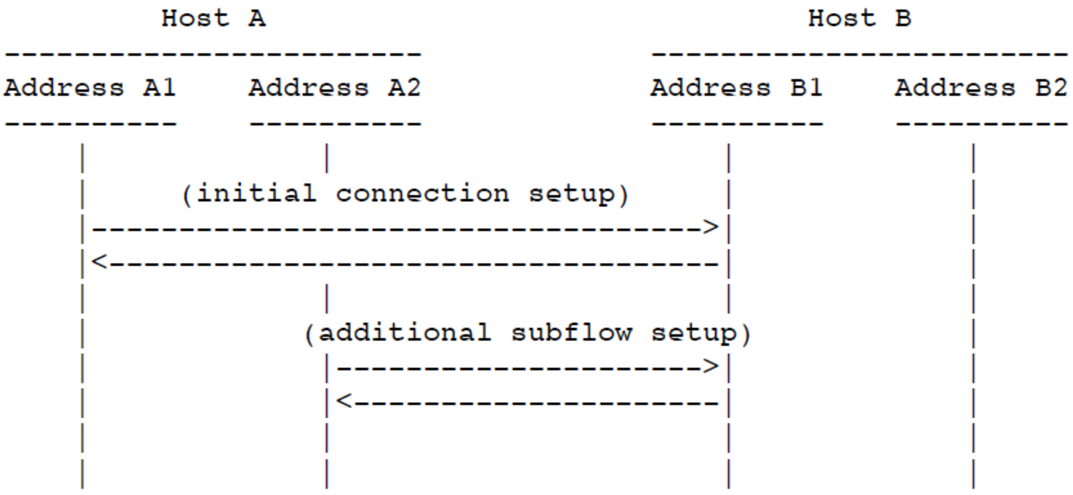
MPTCP:

MPTCP is a new experimental protocol, which is described in:

1. http://tools.ietf.org/html/rfc6824
2. C. Paasch and O. Bonaventure, “Multipath TCP”, Communications of the ACM, April 2014

- The Multipath TCP (MPTCP) is a shim protocol (between Application and the Transport layer) which supports the provisioning of multiple TCP connections to be used by the application, and each such connection is referred to as a subflows:
  - Uses TCP for existing applications, and a single TCP subflow is a special case and can be used by non-MPTCP aware applications
  - Each subflow within the same connection may use a separate network path, and such paths are not necessarily link disjoint. This results in higher throughput, and is tolerant to failures
  - MPTCP is implemented by a number of application clients and servers, including Apple’s iPhone and iPad iOS (starting from iOS 7) to support Siri

- TCP handshaking procedures are used:
  - MPTCP connection is established using the TCP 3-way handshake mechanism
  - Addition, deletion and management of subflows are done using TCP options fields
  - Subflows are terminated using the regular FIN handshake
  - The MPTCP connection is terminated using a connection-level TCP
  - MPTCP adds connection-level (across all subflows) sequence numbers to allow reassembly of segments arriving from multiple subflows



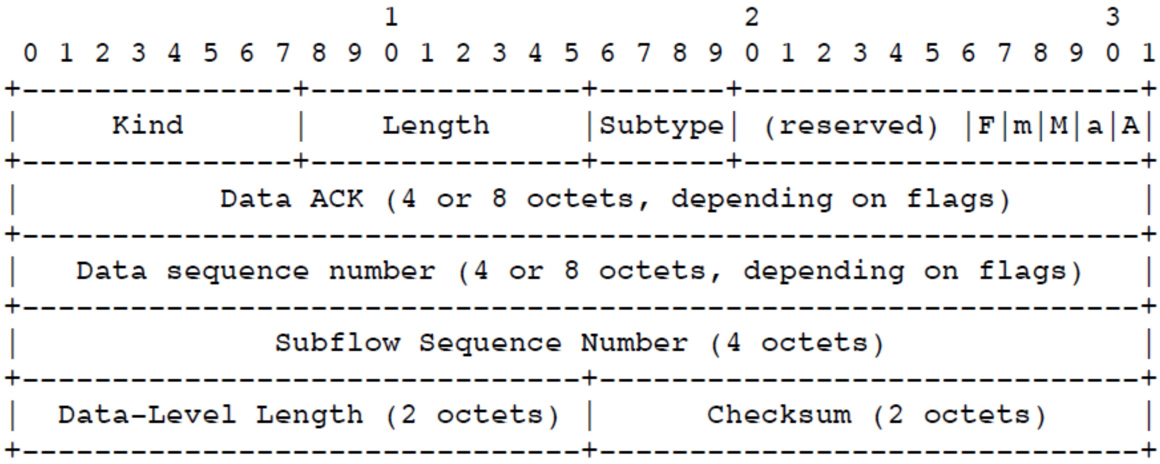
Example of MPTCP usage

MPTCP uses special option fields to add or remove subflows. For this project, **<u>you will not be using the options field</u>**. Instead, each subflow will be established as a new TCP connection.

Transmitting Data Using MPTCP:

- There are two levels of sequence numbers and ACKs:
  - Sequence numbers and ACKs in the subflow TCP header refer to data within the subflow. These are the standard TCP sequence numbers.
  - MPTCP has a 64-bit or a 32-bit data sequence numbers (DSN) to number all bytes sent over **ALL** subflows:
    - Bytes on different subflows are mapped to the DSN so that bytes can be retransmitted on different subflows in the case of data loss
    - **<u>In this project, we will use 32-bit DSNs</u>**

- A host sends a data segment which also includes:
  - DATA_SEQUENCE_SIGNAL (DSS) for a group of packets (as an option)
    - Data Sequence Mapping between subflow data and connection data
    - Data ACK (may or may not be present, depending on flags)
    - Checksum (optional)
- Data Sequence Signal (DSS) option format:



```
                       1                   2                   3
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
  +---------------+---------------+-------+----------------------+
  |     Kind      |    Length     |Subtype| (reserved) |F|m|M|a|A|
  +---------------+---------------+-------+----------------------+
  |           Data ACK (4 or 8 octets, depending on flags)       |
  +--------------------------------------------------------------+
  |   Data sequence number (4 or 8 octets, depending on flags)   |
  +--------------------------------------------------------------+
  |              Subflow Sequence Number (4 octets)              |
  +-------------------------------+------------------------------+
  |  Data-Level Length (2 octets) |      Checksum (2 octets)     |
  +-------------------------------+------------------------------+
```

- A: ACK is present; a: ACK is 8 octets (meaningful only of A is 1)
- M: DSN, SSN, DLL and Checksum are present; m: DSN is 8 octets (meaningful if M is 1)
- F: Connection data FIN
- The sequence number in the TCP header is the subflow sequence number
- DSS usage:
  - Provides mapping from the subflow sequence # to the connection's data sequence #
  - The length of data over which this mapping is valid (data-level length): 0 means infinite mapping, which can be used if there is a single subflow
  - The mapping can be over multiple packets, rather than one packet at a time.
  - Allows the same data to be sent on multiple subflows simultaneously, if links are lossy (i.e., mapping may not be unique)
- Data Sequence Number is absolute (over the entire connection)

- Subflow Sequence Number is relative (first byte in the SYN that established the subflow has sequence number 0)
- Data ACK is a cumulative ACK over the connection

Procedure:

1. In this project, you will implement a simplified version of MPTCP that is mostly focused on using multiple paths, and mapping data to those paths:

- The objective of this project is to establish multiple (3 in this project) TCP **data** subflows between a client and a server
- You will **NOT** use the options field. Rather, you will establish an **additional control** TCP connection on which you will send the **DSS**

2. You will start by implementing a **client program** that will generate 992 bytes, which are numbered 0 through 991. These bytes will contain 16 repetitions of:

- The ASCII digits 0 through 9, followed by
- The ASCII characters a to z, and finally followed by
- The ASCI characters A to Z.

3. The client will establish **4** TCP connections to a server program:

a. First connection will be a control connection on which one DSS will be sent for each data segment sent on a subflow.

b. The other three connections will be data connections, representing 3 TCP subflows to the server. Once each of these 3 connections is established, it will be forked as a **child** process. The parent process will communicate with the child process using either **pipe**() (see the man page for pipe in sections 2 and 7), or by using **Unix sockets**. Make sure that the pipe or the Unix socket is created before the fork.

c. The client will send the data to the 3 TCP subflows child processes 4 bytes at a time, and in a cyclic manner. That is, the first 4 bytes will be sent to the first subflow, the second 4 bytes will be sent to the second subflow, the third 4 bytes will be sent to the third subflow, then the fourth 4 bytes will be sent to the first subflow, and so on.

d. The client will send a DSS on the TCP control connection for each segment sent on a TCP subflow in order to establish data mapping to the server.

The client will write, in a log file, the mapping used from the 992 bytes to the sequence numbers on all 3 connections

4. You will also implement a **server** process that will also accept connections, namely, 4 connections: one is the control connection, and the remaining 3 are TCP data subflows.

a. The server process will also fork child processes for all 3 data TCP subflows. The parent process will also communicate with the child processes using pipes or Unix sockets.

b. The server process will accept data from all TCP subflows, and perform the inverse mapping from the subflow to global sequence numbers.

c. The server process will display the received 992, in order on the display monitor of the machine running the server.

The server will write, in a log file, the mapping used from the sequence numbers received on all 3 connections to the 992 bytes.

Important Notes:

- File descriptors (sockets in our case) are blocking by default. That is, if you want to read from a socket, read() will wait until something is received. If you want to make a file descriptor nonblocking, you can use:

```
#include < fcntl.h>
int flags;
...
flags = fcntl(0, F_GETFL); /* get the file flags */
flags |= O_NONBLOCK; /* set the nonblocking flag */
fcntl(0, F_SETFL, flags); /* set the file flags */
/* the file descriptor is now nonblocking */
```

- You must write your programs in C or C++. If you prefer to use Java, you need to get permission from the instructor first.
- Try to make your program as modular as possible.
- You must submit your program on Canvas by 5pm CDT on the due date. Notice that there are two separate submission links:
    - One for the project report
    - And, the second one is for the project code.

Submit the project code and all other files, e.g., .h files, as one zip file.

- Demonstrate your working programs to the TA (the TA will contact you to set up a demo schedule).
- You may make any reasonable assumptions for any of the missing details. However, in your project report, you must clearly state these assumptions.
- If you introduce any **innovative** idea in the project, you may receive up to **5% bonus** (of the total course grade).