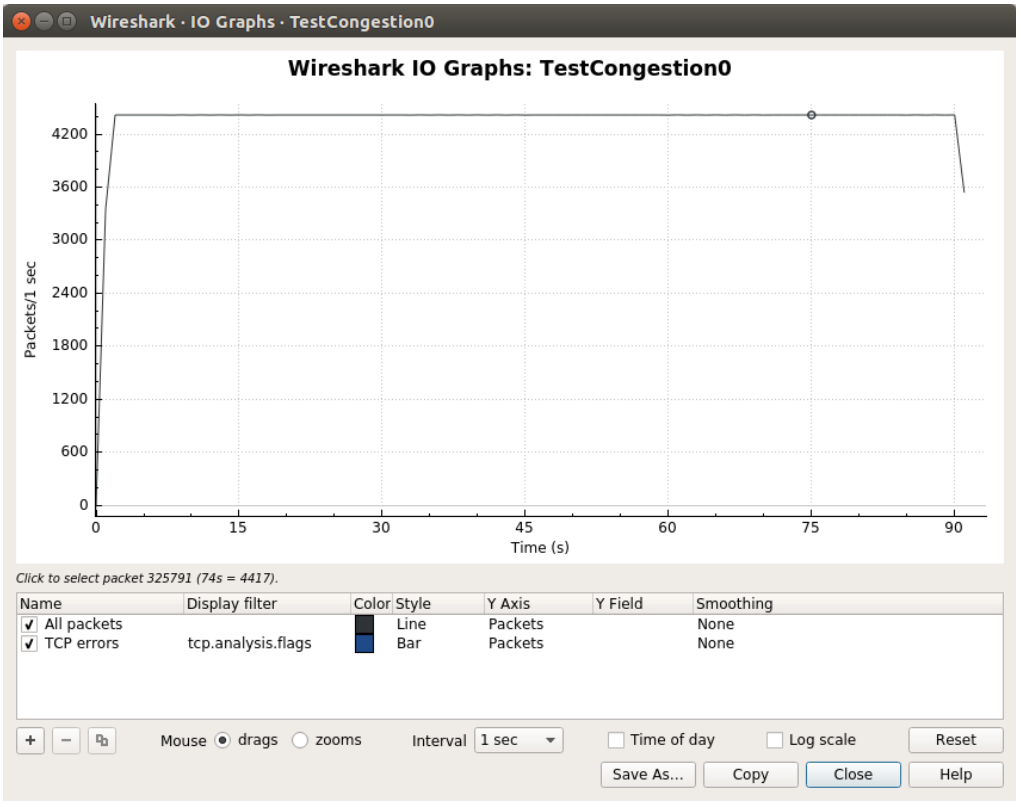
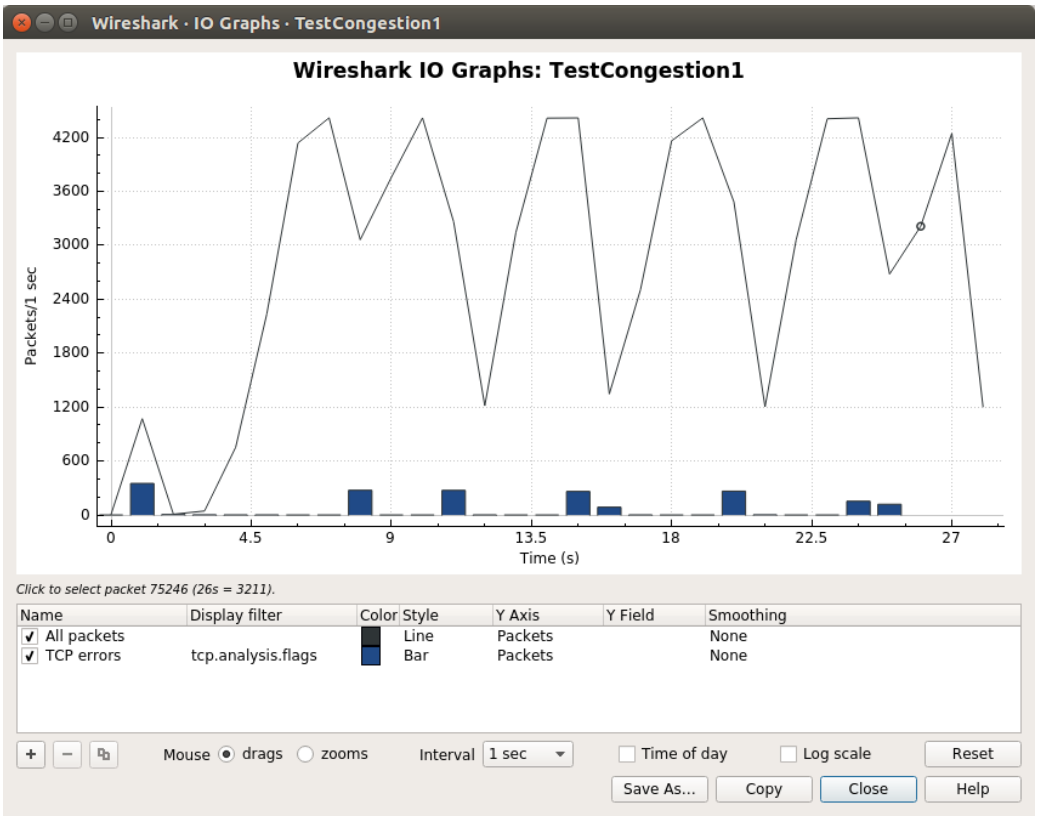
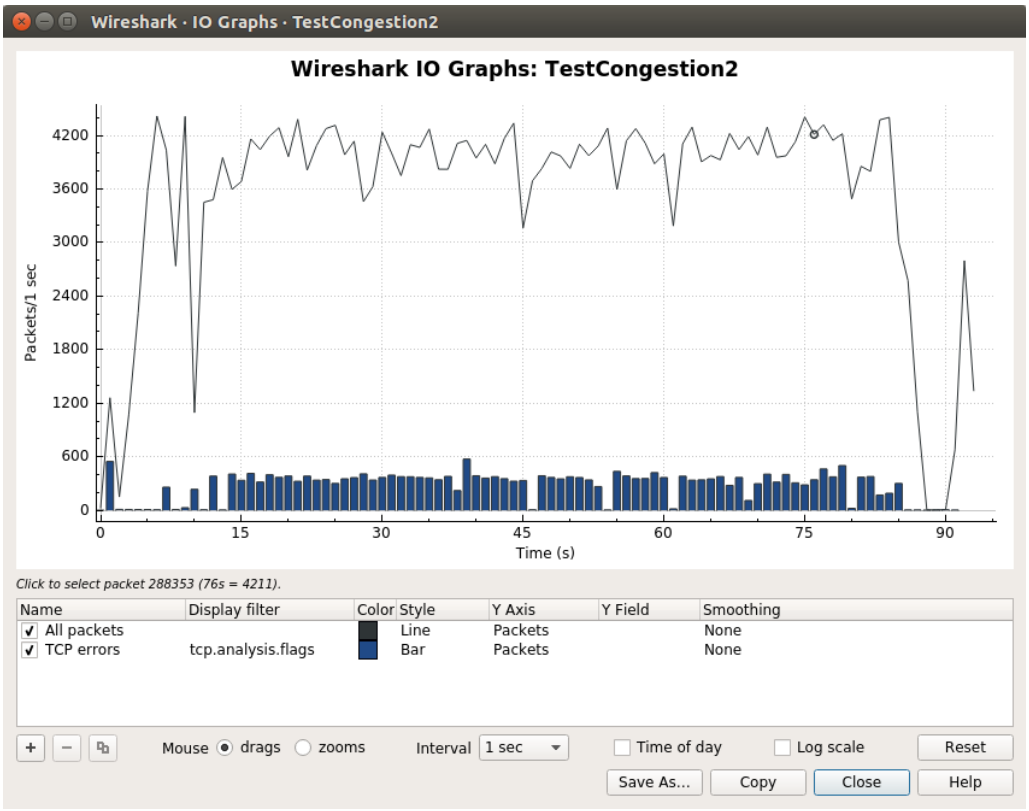
CS341 Project 2 Report

Team 26 - 20150109 Donggyun Kim

Requirement 1







Requirement 2

First, I created a class Socket, which has a necessary information of an socket. It contains a domain, protocol, a boolean value whether it is bounded or not, socket address structure, and size of the structure.
Then, I made three type of tables. First, dtable is a descriptor table which maps a file descriptor to a socket. Since each process must have its own descriptor table, I made another table named ptable which maps a process id to the process's dtable. Last one is aspace, which stores already-in-use socket addresses in the form of (IP address, port number). For dtable and ptable I used standard unordered map class and for aspace I used standard unordered set. When TCPAssignment constructor called, a ptable and an aspace are created with empty tables. These tables are used for the global tables among whole processes using TCPAssignment interface. When TCPAssignment destructor called, all tables are cleared.
Next, I defined four system call helper functions, syscall_socket, syscall_close, syscall_bind, syscall_getsockname.

1) syscall_socket
syscall_socket gets a process id, domain, and protocol. If it's first time of the process to call socket system call, it creates an empty dtable for the process and insert it into the ptable with given pid. Then it allocate a new file descriptor using SystemCallInterface::createFileDescriptor function. Finally, it finds a dtable with given pid, creates a new socket with given domain and protocol, and inserts it into the dtable with the file descriptor.
After all, it returns 0.

2) syscall_close
syscall_close gets a process id and a file descriptor. In general, it removes a socket from the process's dtable and free the descriptor. If the socket is bounded, it also frees the bounded address by removing the socket address from aspace.
If the process hasn't call socket or given file descriptor is wrong, it returns -1. If everything is ok, it returns 0.

3) syscall_bind
syscall_bind gets a process id, a file descriptor, a pointer to sockaddr_in structure, and a length of the structure. Since each applications calls bind system call with sockaddr structure pointer, I casted its type into sockaddr_in pointer before calling syscall_bind, in TCPAssignment::systemCallback.
In general, it assigns a sockaddr_in structure which consists of an IP address, a port number and family. It first finds a socket descripted by the given descriptor in a dtable of a process given by the process id, then stores a sockaddr_in structure in the socket. After storing the sockaddr_in structure, the socket's bound variable which initially was false changes into true. Then a pair of IP address and port number is stored in aspace to mark that the socket address is now using by the process.
There are several error cases for this function. If pid or fd are invalid, or if the socket is already bounded, or if the port number is priviliged, or if the address is already using by other processes, it returns -1. Otherwise, it returns 0.

4) syscall_getsockname
syscall_getsockname gets a process id, a file descriptor, a pointer to sockaddr_in structure, a pointer to socklen_t. It copies a content of a sockaddr_in structure stored in the socket descripted by fd of pid into the given sockaddr_in pointer. Then it writes a length of the structure with given socklen_t pointer.
If pid or fd is invalid or the socket is not bounded, or if the socklen_t pointer initially points a negative number, it returns -1. Otherwise, it returns 0.