15-441 Networks - Project 2
TCP Analyzer

Project 2 Lead TA: Xavier Appe (xja@andrew.cmu.edu)

Assigned: Wednesday, September 25, 2002
Due Date: Monday, October 14, 2002

1 Abstract

In this assignment, you are going to analyze different TCP connections to understand the TCP state machine, the TCP congestion control mechanism and the TCP error handling system. This project is divided into three parts. In Part 1, you will learn how to decipher TCP headers. In Part 2, you will program a TCP state analyzer that must be able to analyze any TCP connection and compute the next state of that connection. In Part 3, you will study the behavior of the TCP error handling system and the congestion control system by measuring transfer times and used bandwidth with different TCP connections in various environments.

As part of this project, you will implement a TCP parser and a TCP state analyzer. You will also verify some mathematical relationships between transfer time and file size and between bandwidth and error rate.

You will work in pairs for this assignment.

1.1 Suggested Checkpoints and Deadlines

Below is a table of *suggested* checkpoints. While it is not required that you meet all of the requirements by the specified dates, you should keep these deadlines in mind to gauge your progress. This assignment is due on **Monday, October 14, by 11:59 PM**. The late policy is explained on the course web site.

Date	Description
September 25	Assignment handed out. **PLEASE START EARLY!**
October 2	Part 1 completed
October 11	Part 2 completed
October 14	Part 3 completed and Assignment due. See Section 8.1 for handin requirement

2 Where to get help

If you have a question, please do not hesitate to ask us for help, that's why we're here! General questions should be posted to the class bulletin board, **academic.cs.15-441**. If you have more specific questions (especially ones that require us to look at your code), please drop by our office hours. The TA's office hours are listed below:

TA	Office	Hours
Julio Lopez	Wean 8205	Monday 2:30 - 3:30
Justin Weisz	Wean 3604	Tuesday 5:00 - 6:00
Umair Shah	Wean 3710	Wednesday 2:00 - 3:00
Xavier Appe	Wean 3108	Friday 2:00 - 3:00

3 Introduction

During the course of this project, you will do the following:

- Parse real Internet packets and learn the significance of every field in the TCP header
- Become familiar with the TCP state machine
- Become familiar with the use of sequence numbers and acknowledgement numbers in the sliding window protocol
- Improve your skills in debugging and testing network programs
- Become familiar with the TCP congestion control mechanism

3.1 Project specification

3.1.1 Background

Throughout the project, you will need a good knowledge of how TCP works.

The following resources may prove to be useful in the design and implementation of your project:

1. **TCP Header Format**
   `http://www.freesoft.org/CIE/Course/Section4/8.htm`
   This document details every field of the TCP header, it may turn out to be useful to write the parser.

2. **TCP State Diagram**
   `http://www.inf.ethz.ch/vs/edu/WS0102/VS/TCP-State-Diagram.html`
   This diagram shows all the possible states of a TCP connection as well as all the possible transitions between states.

3. **RFC 793 — Transmission Control Protocol**
   `http://www-2.cs.cmu.edu/afs/cs/archive/internet-rfc/rfc793.txt`
   The ultimate reference to understand TCP. A very good place to look at if you want detailed information on TCP.

4. **RFC 1071 — Computing the Internet Checksum**
   `http://www.cs.cmu.edu/afs/cs/archive/internet-rfc/rfc1071.txt`
   May help to verify the checksum in part 2.

5. **RFC 2581 — TCP Congestion Control**
   `http://www.rfc-editor.org/rfc/rfc2581.txt`
   If you want to know more about TCP congestion control in part 3.

6. **Lecture Notes**
   `http://www-2.cs.cmu.edu/~srini/15-441/F02/syllabus.html#Schedule`
   These are the notes from the lectures on TCP.

7. **Google Groups**
   `http://groups.google.com/`
   This is Google's archive of USENET postings. If you have a programming question (i.e. how do I do _____ in C?), or if you can't figure out what a certain function does, this is a great place to find answers.

8. **The Class Bulletin Board**
   `academic.cs.15-441`
   We fully encourage you to read and post questions to the class bboard. The TAs will be reading the bboard daily, and this is the best place to get quick answers to short questions. If your question involves a lot of code, or if it will take more one minute to answer, please come to our office hours instead.

3.2 Programming Guidelines

Your programs must be written in the C programming language. C++ submissions will not be accepted. You can use any development environment you like, but please note that we will be compiling and testing your programs on Andrew Linux machines (i.e. linux.andrew.cmu.edu), so you are responsible for making sure your program compiles and runs correctly on these machines. We recommend using `gcc` to compile your program and `gdb` to debug it. You should also use the `-Wall` flag when compiling to generate full warnings and to help debug.

For this project, you will also be responsible for turning in a GNU Make (`gmake`) compatible Makefile. Everything you need to know (and much much more) about `gmake` can be found at `http://www.gnu.org/manual/make/html_mono/make.html`. The Makefile for this assignment should be very simple.

4 Part 1: Parsing a TCP header and understanding the main TCP concepts

In Part 1, we provide a program, called `tcptrace`, that reads an input file containing the packets of a TCP connection and parses the IP layer of each packet. Your job is to add a TCP parser to the program.

4.1 The program tcptrace

The program `tcptrace` is composed by three files: tcptrace.c, ip.c and header.h

- **tcptrace.c** contains the function read_pkts. It opens the file given as an argument on the command line, parses the packets contained in the file and stores them in the array 'pkt'. Then it calls the function handle_IP for each packet.

- **ip.c** is the IP parser. It contains the function handle_IP that takes a packet as an argument, parses the IP header and displays every field of the IP header.

- **header.h** contains the definition of constants, variables and functions you may want to use for Part 1 and Part 2.

`tcptrace` does NOT parse packets directly coming from the network, but coming from an input file that contains all the packets of a given connection.

To compile the code, type `gcc -o tcptrace tcptrace.c ip.c`
Usage: `tcptrace <inputfile>`
The source files can be found in `http://www-2.cs.cmu.edu/~srini/15-441/F02/Projects/lab02/src/`

4.2 The format of the input files

Each file contains a series of packets that are represented as follows:
`pktnb tsec tusec b1 b2 b3 b4 ... bn`
where `pktnb` is the packet number in the file, `b1` is the first byte of that packet in hexadecimal, ..., `bn` is the last byte of that packet, n being the number of bytes of the packet, and `(tsec, tusec)` represents the timestamp of the packet, where tsec is the seconds field and tusec is the microseconds field.

Here is an example of a packet :

```
3 165892455 953265 45 00 00 49 f2 7e 40 00 40 06 9a 51 80 02 dc 8a 80
02 d1 4f 6a b0 00 17 43 97 0e d7 b5 b0 86 cc 50 18 7d 78 e9 2f 00 00 ff fd
26 ff fb 26 ff fd 03 ff fb 18 ff fb 1f ff fb 20 ff fb 21 ff fb 22 ff fb 27
ff fd 05 ff fb 23
```

The packets in the file just have an IP header, a TCP header and a certain amount of data. That means that the first byte of a packet in the file is the first byte of the IP header of that packet.

The beginning of the TCP header (i.e. the part you have to parse) starts `hlen*4` bytes into the packet (hlen is defined in the function handle_IP).

4.3 What you have to do

You first need to implement the TCP parser function handle_TCP and include it in the `tcptrace` program. You must call your TCP parser just after the IP parser. The TCP parser must take an array representing a packet as input argument and extract the value of every field of the TCP header. After processing all the packets, your program must display information about the connection

The format of the output of your program must be:

```
IP address of the host that initiated the connection (host1):  1.2.3.4
IP address of the host that responded to that initialization (host2):  5.6.7.8
IP address of the host that first closed the connection:  5.6.7.8
Total number of data bytes sent by host1:  5826
Total number of data bytes sent by host2:  135
TCP Port used by host1:  4523
TCP Port used by host2:  110
Initial sequence number of host1:  12356858
Initial sequence number of host2:  26534532
```

`Application layer protocol involved:  POP3`

The application layer protocol that you must recognize are {FTP, SSH, TELNET, SMTP, HTTP, POP3, NNTP, HTTPS}. You may use `getservbyport()` to get the string.

We provide some test files called `fpart1_1`, `fpart1_2` and `fpart1_3`. They can be found in `http://www-2.cs.cmu.edu/~srini/15-441/F02/Projects/lab02/testfile`

5 Part 2: The TCP state analyzer

Now that your TCP parser is working, and that you are familiar with the important notions of a TCP connection, you can analyze the packets in more detail.

In Part 2, the goal is to analyze the state of a TCP connection and predict the next possible packets. The possible states of a TCP connection are described in the RFC 793. There will be two main steps in this part:

1. Checking the correctness of the packet and keeping track of the TCP state of the hosts involved in the connection.
2. Predicting the next packet

You should start with the program you've made in Part 1. The TCP connections you will have to analyze will be given in a file of the same format as Part 1.

5.1 Assumptions you should use and things that you do not need to consider

1. You will assume that there will be no loss, no duplication and no reordering.
2. You do not need to support/check the TCP option field
3. You do not need to support/check the URG, RST and PSH flags
4. You do not need to support/check the urgent pointer field in the TCP header
5. You do not need to support/check the information contained in the IP header except the IP address
6. You do not need to consider simultaneous open/close
7. You will not need to use the state TIMEWAIT.

5.2 Checking correctness and updating states

For each packet, you need to check:

- if the TCP checksum is correct. Refer to the RFC 793 and 1071 for more details.
- if the packet belongs to the current connection. The first packet in the array (`pkt[0]`) will be a SYN packet. It will be used to identify the TCP connection. All the following packets must belong to the same connection. Two packets belong to the same connection if they have the same {src IP, dst IP, src port, dst port} pair.

- if the flags are correct regarding the current state of the connection. You should assume that no retransmissions occur so if, for example, you see a SYN packet when the connection is established, you must return an error
- if the sequence, acknowledgment and window are correct according to the sliding window protocol. Refer to the lecture notes or the RFC 793 for more details

When your program detect an error, it should print an error message according to the output format described in section 5.4

we provide a prototype of the structure that will be used to store connection information of the client and the server:

```
struct connec_info_per_host  {
        char constate[30];        //State of the connection
        struct in_addr srcIP;     //IP address of the client
        struct in_addr dstIP;     //IP address of the server
        unsigned int nbpkt;       //Number of sent packets during the connection
        unsigned long sentbytes;  //Amount of sent bytes
        unsigned short srcport;   //TCP port of the client
        unsigned short dstport;   //TCP port of the server
        unsigned int iss;         //initial sent sequence number
        unsigned int irs;         //initial received sequence number
        unsigned int snd_una;     //send unacknowledged
        unsigned int snd_nxt;     //send next
        unsigned int rcv_nxt;     //receive next
        unsigned int rcv_wnd;     //receive window

    }
```

The state of the connection will be updated according to the state diagram provided in the RFC 793. For example if current state of the client is CLOSED and the current state of the server is CLOSED, after receiving a SYN packet from the client to the server, the current state of the client becomes SYN SENT and the current state of the server becomes SYN RECEIVED

When your program has processed all the packets, it should display information about the connection. In other terms, it should display the value of every field of the structure connec_info_per_host for each host as given in section 5.4.

5.3 Predicting the next packet

After processing all packets according to section 5.2, you must predict the next packet that is going to be sent by each host. In addition to the assumptions you made in the previous section, you must also assume that

- only the client sends data.
- the client first closes the connection
- the amount of data to transfer is very large. So, when the connection is established, you should assume that only packets related to data transfer will be sent.
- The maximum segment size is 1448.
- no data is sent during the initialization phase and the termination phase.

- the sequence of packets during the termination phase will be : (1)a FIN packet sent by the client, (2) an ACK packet sent by the server to acknowledge the FIN, (3) a FIN packet from the server, (4) an ACK packet from the client to acknowledge the FIN.
- if the connection is CLOSED, you do not need to predict the next packet.

Your program should display for each host the flags, the sequence number and the acknowledgment number of the predicted packets according to the output format specified in section 5.4.

So for example, if host1 initiates the connection to host2 and if the current state of the connection is ESTAB for both hosts, the next packet that host1 is going to send is a data packet and the next packet that host2 is going to send is an ACK packet

If host1 initiates the connection to host2 and if the current state of the connection is SYN SENT for host1 and SYN RECEIVED for host2, the next packet that host2 is going to send is a SYN,ACK packet and the next packet that host1 is going to send is an ACK packet

5.4 Output Format

You program must output on stdout:

1. An error message if an error occurred.
2. Connection information of the client
3. Connection information of the server
4. The next packet of the client
5. The next packet of the server

5.4.1 Error messages

The possible error messages that are expected in your final version (debugging turned off) are the following strings:
-"ERROR: Incorrect checksum"
-"ERROR: Packet does not belong to the current connection"
-"ERROR: First packet is not a SYN packet"
-"ERROR: Incorrect flag(s)"
-"ERROR: Incorrect sequence number"
-"ERROR: Incorrect acknowledgement number"

Your program must check the state of the connection, the acknowledgment number and the sequence number in that order. The order is important if the packet contains several errors. For example, if a packet has an incorrect flag and an invalid acknowledgment number, the displayed error should be "Incorrect flag(s)" and not "Incorrect acknowledgment number"

5.4.2 Connection information format

Here is the expected format:

```
IP address of the client:  1.2.3.4
TCP port of the client:  1256
State of the client:  ESTAB
Number of packet sent by the client :20
Total number of data bytes sent by the client:  20826
Initial sent sequence number of the client:  12356858
Initial received sequence number of the client:  26534532
snd_una:  1524654565 snd_nxt:  1524656065
rcv_nxt:  1253595584 rcv_wnd:  17568

IP address of the server:  5.6.7.8
TCP port of the server:  23
State of the server:  ESTAB
Number of packet sent by the server :  20
Total number of data bytes sent by the server:  0
Initial sent sequence number of the server:  26534532
Initial received sequence number of the server:  12356858
snd_una:  1253593584 snd_nxt:  1253595584
rcv_nxt:  1524656065 rcv_wnd:  46568
```

5.4.3 Next packet format

Here is the expected format:

```
Client's next packet:
   Flags:  data
   Sequence number:  153415465
   Acknowledgement number:  122222356

Server's next packet:
   Flags:  ACK
   Sequence number:  122222356
   Acknowledgement number:  153456456
```

The possible values for the flags are a combination of {SYN, FIN, ACK, data}.

When you don't know the sequence number or the acknowledgment number, your program should display "unknown".

5.5 What your high-level algorithm should look like

1_ Read the input file, this is provided in `tcptrace.c`
2_ Parse the input file, this is provided in `tcptrace.c`
3_ For each packet

- Parse the packet, that is what you did in Part 1.

- Check if the packet has a valid checksum. If the checksum is not valid, display the appropriate error and exit
- If this is the first packet, initialize the characteristics of the connection; else check if the packet belongs to the current connection
- If the packet does not belong to the current connection, display the appropriate error and exit
- Check if the flags, the acknowledgement number and the sequence number are correct given the current state. If the flags/seq num/ack num is incorrect, display the appropriate error and exit.
- Determine the new state of the connection and update variables in the structures connec_info_per_host

4_ Display information on the current connection for each host.
5_ Predict the next client packet.
6_ Predict the next server packet.

6 Part 3: Throughput Analysis

In Part 3, you will have to measure the transfer time of different files and the bandwidth used during transfer in different environments. You will draw two graphs showing your result. We provide a sample of answer sheet at the end of this handout. The goal of this part is to experimentally verify the formulas you learned in class and to get familiar with the TCP congestion control mechanism.

All the measurements will be done from the machines in the 15441 cluster. There are ten machines: {netclass-1.intro.cs.cmu.edu}, {netclass-2.intro.cs.cmu.edu},...,{netclass-10.intro.cs.cmu.edu}

Some of them have been set up to purposely generate packet losses (machines with an even netclass number), so be careful when you use them.

6.1 Measuring the time to transfer a file

First, you need to connect to one machine in the 15441 cluster.

Here is how you have to proceed :

1. Create a directory in your Andrew home directory called 15-441
2. Now you need to give access to *campusnet* so type the following:
   `fs sa /afs/andrew.cmu.edu/usr/<your_userame> system:campusnet l`
3. and `fs sa /afs/andrew.cmu.edu/usr/<your_userame>/15-441 system:campusnet rl`
4. Create a file called *.klogin* inside the 15-441 folder which contains:
   your_username@ANDREW.CMU.EDU (not your_username@andrew.cmu.edu)

5. You can not use unsecure telnet to connect to these machines.
   Therefore, either use Nifty `telnet` or `ssh`.
   The username you will use will be your_username@andrew.cmu.edu not your Andrew userid.
   Therefore, if you use `ssh`, don't forget to use "-l" flag with the correct username as otherwise, `ssh` will try to use just your Andrew userid and therefore you won't be able to login. For example `ssh -l xja@andrew.cmu.edu netclass-2.intro.cs.cmu.edu`

6. If you have any problems with logging in, send email to Xavier Appe: xja@andrew.cmu.edu

You must choose a pair of machines (one server, one client). The possibilities are :

- server: netclass-2.intro.cs.cmu.edu; and client: netclass-3.intro.cs.cmu.edu
- server: netclass-4.intro.cs.cmu.edu; and client: netclass-5.intro.cs.cmu.edu
- server: netclass-6.intro.cs.cmu.edu; and client: netclass-7.intro.cs.cmu.edu
- server: netclass-8.intro.cs.cmu.edu; and client: netclass-9.intro.cs.cmu.edu

Don't use netclass-1.intro.cs.cmu.edu or netclass-10.intro.cs.cmu.edu. For the testing, you will only need to connect to the client machine of the pair you've chosen.

To simulate the transfer of a file, you will use the program texttttttcp. This program takes as argument the number of buffers you want to send (option -n) (for this assignment, there will be only one buffer), the size of a buffer (option -l), the listening port of the server (option -p) (the default is 2000) and transmission indicators (option -t -s). For example, if you want to simulate the transfer of a 25000-byte file, you will use the following command:

```
ttcp -t -s -n1 -l25000 -p2000 <ip address of the server>
```

For more details, you can look at the man page by typing 'man ttcp'.

Use the following table to get the IP address of a machine

DNS name	IP address
netclass-2.intro.cs.cmu.edu	10.0.1.6
netclass-3.intro.cs.cmu.edu	10.0.1.10
netclass-4.intro.cs.cmu.edu	10.0.1.14
netclass-5.intro.cs.cmu.edu	10.0.1.18
netclass-6.intro.cs.cmu.edu	10.0.2.2
netclass-7.intro.cs.cmu.edu	10.0.2.6
netclass-8.intro.cs.cmu.edu	10.0.2.10
netclass-9.intro.cs.cmu.edu	10.0.2.14

You will use the interface connected to the private network, eth1. On each machine, eth0 is connected to the CMU network whereas eth1 is connected to the "private" network i.e. a network for experimentation.

The output of ttcp will give you the time to transfer the data as well as the used bandwidth.

You will test file sizes ranging from 2000 bytes to 120000 bytes (ie from 1 to 83 in terms of number of packets, a packet can carry up to 1448 bytes). You should increment the file size by 2000 bytes up to a file size of 40000 bytes and then use an increment of 5000 bytes up to 120000

Once you have done the testing, you need to draw the function $T = f(S)$ where T is the transfer time and S the number of packets in the file.

The theoretical formula of f during slow start is : $T = (log_2(S) + 1) \times RTT$. Show in your graph where you think the slow start phase is, compare your function with the theoretical one and comment your result.

6.2 The influence of packet loss on throughput

For this section, you also need to connect to a client machine in the 15441 cluster. Just use the machine you chose in the previous section. Here, the goal is to observe the evolution of the bandwidth as the error rate grows.
The server machines are configured so that:

Listening port of the ttcp server	Error rate
2100	0%
2101	0.05%
2102	0.1%
2103	0.15%
2104	0.2%
2105	0.25%
2106	0.3%
2107	0.35%
2108	0.4%
2109	0.5%
2110	1%
2111	2%
2112	3%

You will use a file size of 30000000 bytes. For example to test what bandwidth is used when transferring a file with an error rate of 0.25%, type:

```
ttcp -t -s -n1 -l30000000 -p2105 <ip address of the server>
```

Now draw the function $log(B) = f(\log(p))$ where B is the bandwidth and p is the error rate.
The theoretical formula is : $B(p) \approx \frac{1}{RTT \times \sqrt{\frac{2}{3} \times b \times p}}$.
So $\log(B) \approx -\frac{1}{2} \times \log(p) + K$
Compare your function with the theoretical one (especially compare the slope) and comment on your result.

7 Testing

7.1 How you should test your programs

The goal of this section is to give you a better idea of how your project will be evaluated, and what you should be on the lookout for while writing your code. The following list is not guaranteed to be exhaustive:

1. Make sure that your TCP parser works perfectly and that it displays the correct information before beginning Part 2.

2. Do not hesitate to use `tracefilecreator` (see the section 7.2 for more information)to create new kinds of input files. For example, you can use it to capture the packets of the connections on different servers on the Internet(FTP, SSH, POP, SMTP, TELNET, HTTP, etc...). You will probably obtain connections that close using different ways. Do not hesitate to test every capture file with the program you made in Part 1. But remember, we assume no retransmission in Part 2 so the file you get from `tracefilecreator` could return an error even if they contain the packets of a TCP connection.

3. You can directly modify the capture file with a classic editor and test your checking procedures. Make sure that every check works.

4. You can create a file containing the trace of a full TCP connection and remove one packet at a time from the end. Then you can check if the predicted possible packets include the packet you just removed.

5. By generating and capturing a connection in which little data is exchanged, you can make sure that the number of transmitted bytes computed by your program is correct.

7.2 The program `tracefilecreator`

The program `tracefilecreator` is basically a sniffer that reads packets from the network and stores them in a file. It uses the file format described in Part 1. So this is the tool you will use if you want to create new input files for the program `tcptrace`

Here is its usage:
`tracefilecreator <filter> <outputfile>`

- `<filter>` is a filter you want to apply to the capture process. The syntax is the one described in the man page of tcpdump for the argument `<expression>`. See `http://www.tcpdump.org/tcpdump_man.html` for more details. Do not forget to surround this argument with quotes.

- `<outputfile>` is the name of a file in which all the packets are going to be stored.

Example : `tracefilecreator "port 21" outfile.pkt`

To compile it : `gcc -o tracefilecreator tracefilecreator.c -lpcap`. You need to have libpcap installed on your computer.

This program needs root permission to sniff the packets.
You can use this program to test your TCP parser and your TCP state analyzer.
The source code will be available at
`http://www-2.cs.cmu.edu/~srini/15-441/F02/Projects/lab02/src`
You can also use the compiled version available on the netclass machines in /usr/local/bin (see Part 3 for directions to connect to these machines).

7.3 The tests.txt file

As part of this assignment, you are responsible for testing your submission and making sure everything works. You should document all of your testing strategies in a file named `tests.txt` and submit it along with the rest of your assignment. Also, if you have any outstanding issues you know about, but are not able to fix, you should let us know about them. Or, if you have any comments about this assignment, felt that it was too hard or too easy, please let us know.

8 Grading Criteria

When we grade your submission, we will run `gmake` to compile your web server, and then run a combination of tests and grading scripts to evaluate your submission. If your project generates compiler warnings during compilation, you will lose credit; if your server dumps core during our testing, you will lose substantial credit. Remember to compile with the `-Wall` flag, otherwise you will lose points if this is not in your `Makefile`.

Poor design, documentation, or code structure will probably reduce your grade by making it hard for you to produce a working program, and making it hard for us to understand it. Remember the 15-212 slogan: *Think before you hack!*. Egregious failures in these areas will cause your grade to be lowered even if your implementation performs adequately. Putting all of your code in one module counts as an egregious design failure.

It is better to have partial functionality working solidly than lots of code that doesn't actually do anything correctly.

The following is a checklist of all of the features your web server must possess. As you write your programs, you should cross off each feature from the list once you have implemented and tested it. All of these features will be tested during grading.

TCP analysis		100%
	(Part 1)TCP Parser	15%
	(Part 2)Checksum, pkt's connection check	15%
	(Part 2)Seq/Ack Number Operations	15%
	(Part 2)State Determination	15%
	(Part 2)Next Packet Prediction	15%
	(Part 3)File Transfer Time Measurements	10%
	(Part 3)Bandwidth Measurements with errors	15%

8.1 What to turn in

Below is a listing of all of the files you should submit. The handin directory is `/afs/cs.cmu.edu/academic/class/15441-f02-users/group-XX`. Use the lower group number between you and your teammate. Please note that we will be checking timestamps in order to determine late submissions. Check the late policy on the course web site and notify the TAs and professors if you will be using late days.

- GNU compatible `Makefile`
- All of your source code for Part 1 in `<handin directory>/project2/part1` (files ending with `.c` or `.h` only, no `.o` files)

- All of your source code for Part 2 in `<handin directory>/project2/part2` (files ending with `.c` or `.h` only, no `.o` files)
- The two graphs you drew for Part 3 as well as a table of your measurements. You can return them in class or directly to me
- Documentation of your test cases and any known issues you have, named `tests.txt`

9 How to succeed in this assignment

Above all, please **start early**. We will be able to help you more if you **start early**. You will feel better if you **start early**. You will feel even **better** if you finish the assignment *before* it is due. Remember, lots of bad things can happen at 11:50pm the night the assignment is due.

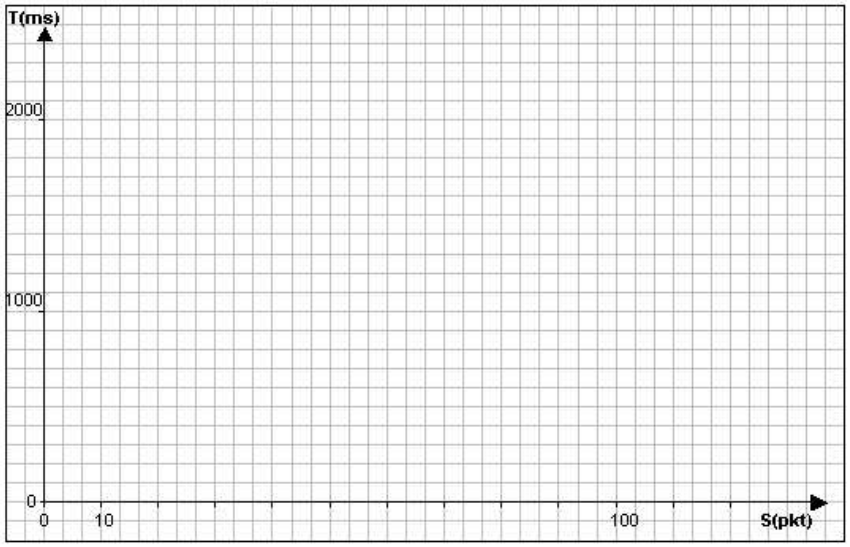
Now that you are resolved to starting this assignment early, read over this handout several times. If anything is unclear, please send email to the bboard (**academic.cs.15-441**), or come to office hours. We are here to help!

Note that you can not begin part2 before part1 is finished. However part3 can be done independently from the other parts. So, if you are stuck in part2, you can try to do part3 before and come back to part2 later. By the way, part1 and part3 are short compared to part2 so arrange your time accordingly.

10 Sample Answer sheet

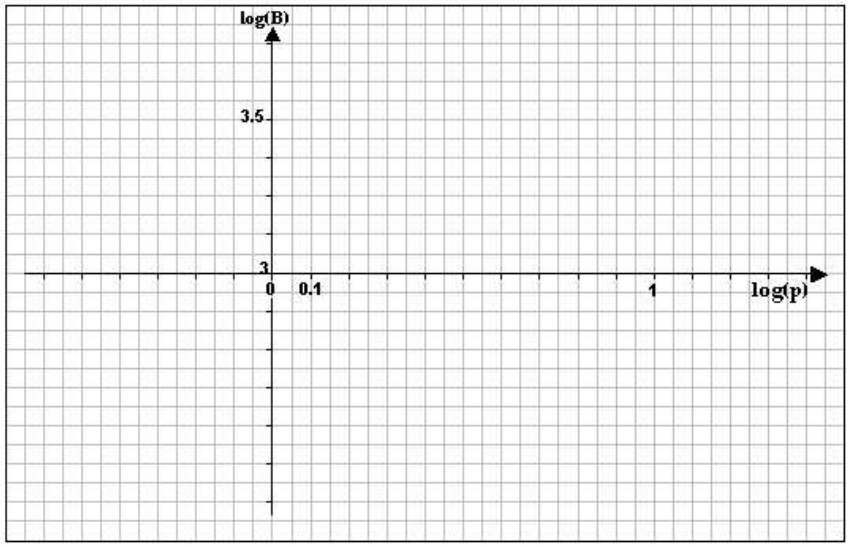
Answer Sheet

Graph: Total transfer time T=f(S)



Comments:

Graph: Evolution of Bandwith log(B) = f(log(p))



Comments: