Project TiniTorrent: Design Space Exploration

Javed I. Khan
Fall 2013
ST: 6/759995 Foundations of Peer-to-Peer Computing
Department of Computer Science, Kent State University
javed@kent.edu

1 Introduction

One of best way to start the project is to divide and conquer. We will progress step by step. We will consider several simplifications for our task and then gradually build toward a full scale BitTorrent which will be able to communicate in any standard BitTorrent swarm over the open Internet.

2 Few Assumptions and Simplifications

- **Dummy for Torrent & Tracker:** Initially we would avoid implementing or even contacting Torrent Server or Tracker. Rather for the first phase we will use two simple text files to make all torrent and tracker information available locally to the TiniTorrent.
- **Strategies in the Base Model:** In the basic version, peers will use *altruistic strategy* rather than tit-for-tat. Thus, all peers will fulfill all requests as long as it has the piece, and will explicitly unchoke all connection at the start and will never choke it. Peers will not use any intelligent download ordering, i.e. the pieces can be requested one by one in *sequential download order*. Peers will not pipeline requests. Indeed, each peer will fetch only *one piece via one connection* and will close the connection after sending/receiving the piece. Initially we will also assume all peers use *matched message sequence*- where at each step a peer sends and the other side waits to receive a specific message. This generally avoids race condition. We will also start with a simple system without connection maintenance. There will be no keep-alive messages.

3 Architecture

Indeed, there are many ways to design the TiniTorrent. However, I have given one high-level schematics of TiniTorrent (Fig-1). Your first task will be to go over it and try to understand the basic components. The top part is the main driver program- which usually handles interactions with users, torrent server, tracker server and initialization. Since, the TiniTorrent will avoid torrent server or tracker server we supply all the required parameters into two files.

> **Torrent & Tracker Files:** You can write those into two text files *SampleFile.tinitorrent* and *SampleFile.tinitracker*. Write using a human readable way such as "field-name=value" to specify all the usual torrent, and tracker supplied parameters and their values. Put dummy value for the information those you will not actually use in basic version (such as HASH info). In advanced version you only have to populate these files from information received from torrent and trackers.
>
> **Local Configuration File:** You may also keep all the local configuration parameters in another text file called *MyInit.tini*. It should have fields such as local port number, various limits such maximum number of concurrent slaves to run, etc.

TiniTorrent program should start with all three of these configuration filenames as one of its arguments such as:

- TiniTorrent MyInit.tini SampleFile.tinitorrent SampleFile.tinitracker

TiniTorrent then fork into two processes- parent process continues to serve as main process- the child process becomes the seeder section. The main process than becomes the user interface. Normally, it would wait for download orders from the users. Since, in our case we already provided the order in the Torrent and Tracker files it simply reads those and continues. Upon reading the orders it then keeps forks the leecher section.

Both the leecher and seeder subsections however ultimately depend on piece exchange slaves for handling the inter peer communications and messaging. Fig-2 provides a high level schematics of the exchange slave. Fig-3 depicts a probable sequence of messages between two peers when one peer is trying to download a piece from the other.

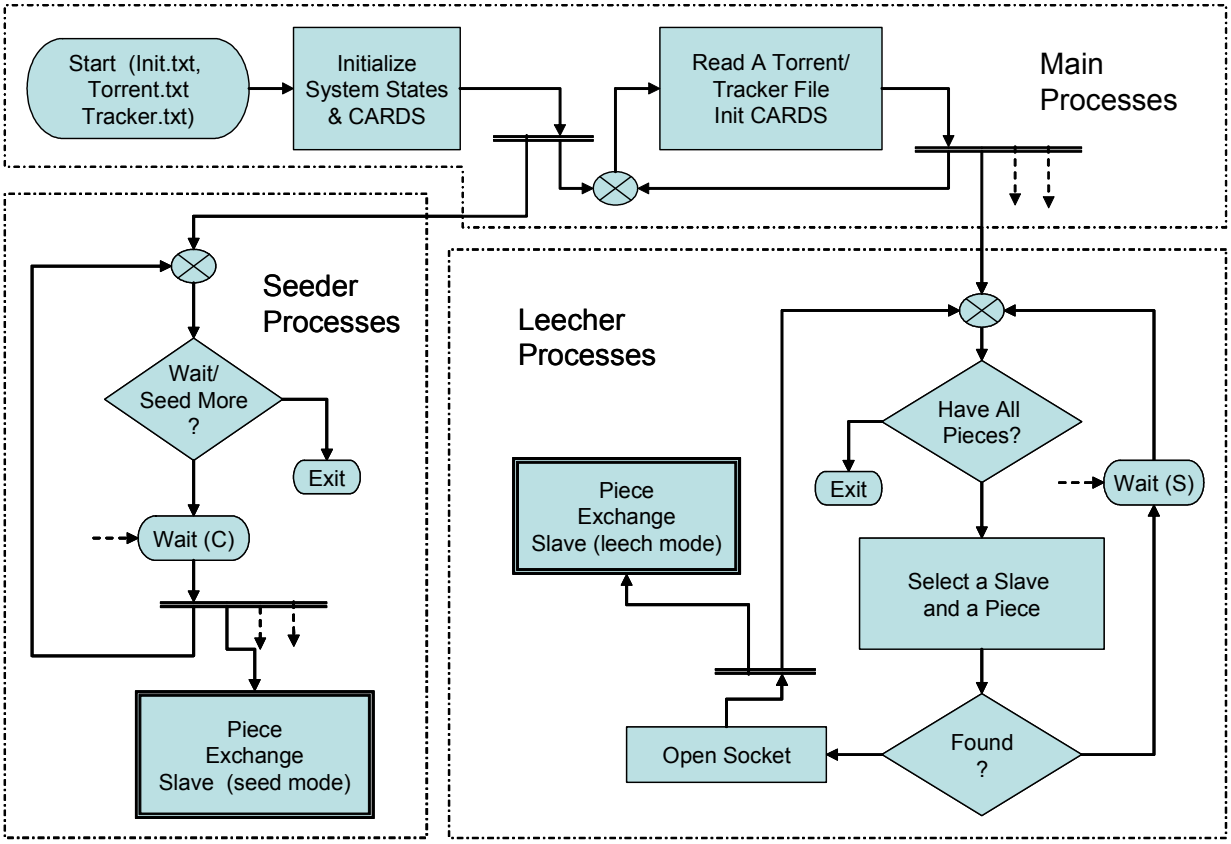

Fig-1 Top Level Architecture of TiniTorrent. It has three major sections. Main processes handles users, trackers, and torrents. The seeder processes handles the communication in seeding mode and the leecher processes handles the communication in other situations. Both seeder and leecher sections however, use piece exchange slaves (next fig) to handle actual messaging with peers.

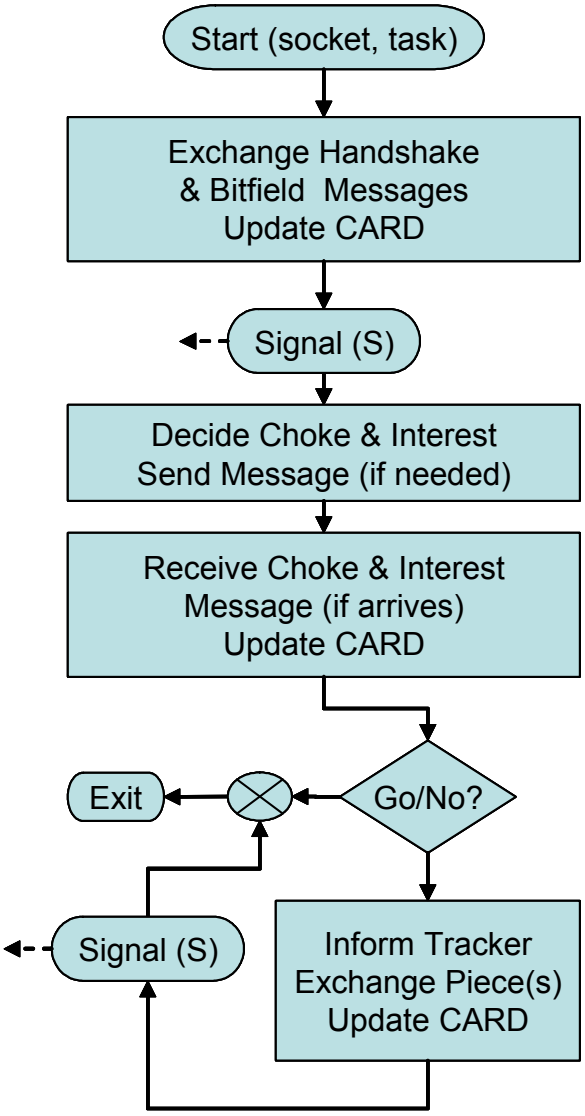

Fig-2 Piece Exchange Slave of TiniTorrent.

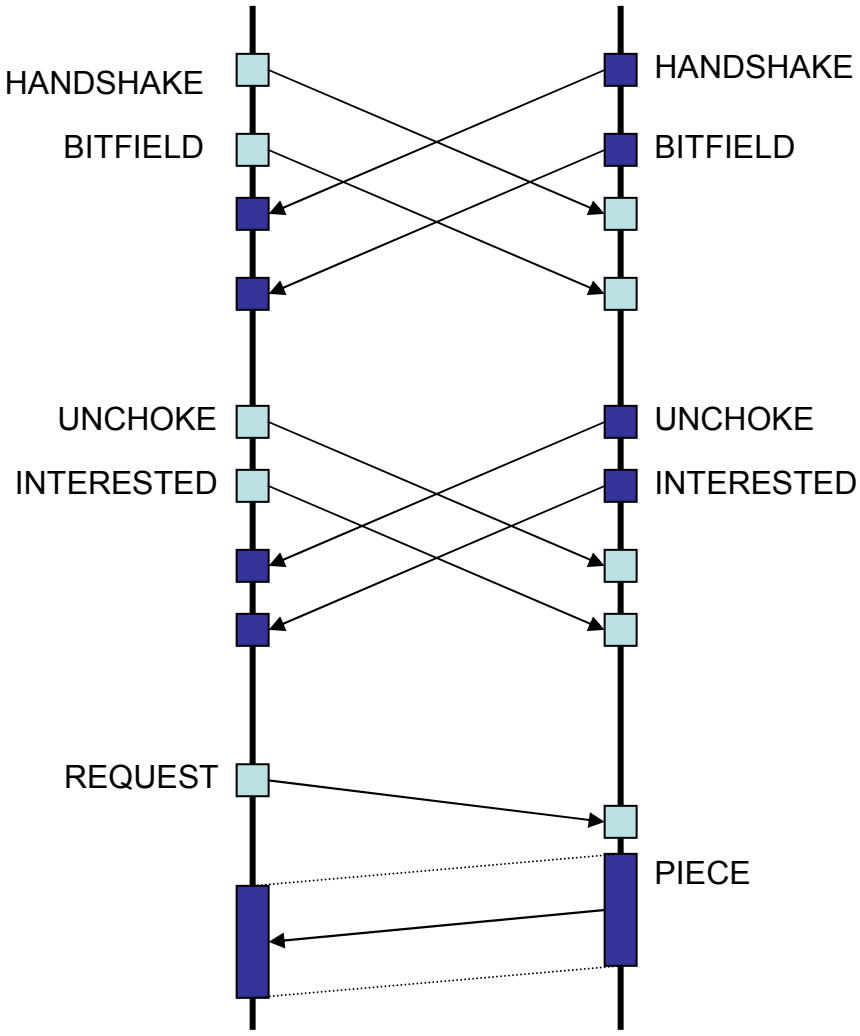

Fig-3 Typical message sequence between two TiniTorrents exchange slaves A & B.

4 TiniTorrent File Structure

Pieces are real or virtual? You really don't need to create a separate file to keep each piece. You may consider creating a fullsized buffer files such as "*SampleFile.buffer*" to store the each active file. At the start create a dummy file of size *filesize* bytes. Then each time you fetch a piece write over the piece bytes at appropriate byte offset in this file. In that way you don't have to manage numerous different pieces separately. Remember to write a small utility routine which can calculate right offsets from piece indices, and handle the piece indexed read write. Once, the file is complete you can simply rename it to make it usable.

5 TiniTorrent Data Structure

One of the crucial steps in the design of a complex system is to design its data structures. I suggest you start with at-least two tables to keep all the parameters, states, and co-ordination variables you will need for each file. I call them CARDs. You can use TORRENT CARD to keep all the information you will receive from the TORRENT file plus any other long term information you will need to seed the file in future. Similarly, you can use the TRACKING CARD to keep all the information about a file which is now being actively shared- whether you are leeching or seeding. Sections below provide some details about suggested structures for these two.

TIPS: Mutual Exclusion & Shared Data Structures: You will see that various processes often will need to read and update same data structure fields in these CARDs. Don't forget to use mutual exclusion where needed.

5.1 Torrent CARD

TORRENT Card stores all the data decoded from a torrent file. It also adds few additional fields which are used by a peer to maintain long time state of a file exchange. There should be one TORRENT CARD for each file which is considered for exchange - whether it is actively exchanged right now or not.

TORRENT CARD	
FILENAME:	
ANNOUNCE:	
FILESIZE:	
PIECESIZE:	
#PIECES:	
……	You can have other fields.. …..
I WANT TO	[SEED|LEECH|NONE]

Fig 5.1

5.2 Tracking CARD & Various States

Each time a file is actively exchanged (pieces are uploaded or downloaded) more detailed piece and connection states are needed to be stored. We will abstract those into TRACKING CARD. TRACKING CARD is a more complex table. It stores the data and states related to a file transfer. It is initialized by data received from the tracker. Then different program components modified other status bits. The later table explains the various enumerations of the fields such slave states, connection states (game), remote peer statistics, status of pieces, etc. There are various ways to model the state transitions. Fig-5.3 to Fig 5.6 explains some of these states.

The shown TRACKING CARD has two main tables. The first part MY PIECES is keeping the status of each of the pieces, namely their hash, whether has the piece (1), or not (0).

The second part is the state of the CONNECTIONs. A peer normally has a limit on the parallel active connections (and a slave assigned to manage the connection). There is one row for each connection or slave. If the slave is

actively serving its process id (pid) is stored. If the slave is actively connected to a peer then remote peer's IP:PORT is also stored. The GAME is a 4 bit status flag (Table 5.4) which stores the current tit-for-tat bits for the connection. The PIECE STATES table then show which pieces has been requested from the remote peer and the state of the request. A connection can serve multiple upload and down load. Table 5.6 shows the code for various states a request might be. The STAT column you can define further. The idea is to keep some statistics about the remote peer so that you can use those for formulating the tit-for-tat strategy and keep a log/record of performances per connection. Once, the piece download is completed in a similar way some log/record of performances can be kept in pSTAT structures in the upper table.

TRACKING CARD									
MY PIECES									
PIECEINDEX→			*1*	*2*	*3*	*……*	*N*		
INFOHASH→			123fd	234fd	1ab2d	23cbb	5333c		
HAVE→			0	0	1	..	1		
pSTAT →									
MY CONNECTIONS									
IP: PORT		*SLVS*	*GAME*	*cSTAT*	***PIECE STATES***				
IP1	PORT1	202	1010		FN				
IP2	PORT2	103	1010			DN			
IP3	PORT3	104	0110				HV		HV
IP4	PORT4	303	0111				UP		
IP5	PORT5	722	0000						UP
…	…	0						…	FN
IPp	PORTp	0				FN	FN		

CODE	SLAVE STATES (SLVS)
BIT1	0=NO ACTIVE SLAVE. xxx = PID OF SLAVE ASSIGNED

Fig 5.3 TiniTorrent's Slave State

GAME BITS	
BIT1	1=AM CHOKING. 0= AM NOT CHOKING
BIT2	1= AM INTERESTED, 0= AM NOT INTERESTED
BIT3	1= CHOCKED, 0= NOT CHECKED
BIT4	1= IS INTERESTED, 0=NOT INTERESTED

Fig 5.4 TiniTorrent's Strategy Bits

SLAVE/CONNECTION/PEER STATISTICS (STAT)	
PID:	SLAVE PID
START:	SLAVE START TIME
UPBYTES:	TOTAL UPLOADED BYTES
DNBYTES:	TOTAL DOWNLOADED BYTES
(others)	Note you can also store other information about the slave, connection, or the remote peer here such as upload, download speed of the remote peers.

Fig 5.5 TiniTorrent's Slave Statistics

NUM	CODE	PIECE STATE(S)
1	FN	I HAVE FOUND THIS PIECE IN THIS PEER
2	IN	I HAVE EXPRESSED 'AM INTERESTED' TO THIS PEER
3	DN	I AM DOWNLOADING (IN PROGRESS) THIS PIECE
4	HV	I GOT IT. I HAVE THE COMPLETE PIECE.
5	OF	I HAVE OFFERED THIS PIECE TO THIS PEER
6	UP	I AM UPLOADING THIS PIECE TO THIS PEER
7	ST	I AM STOPPING UPLOADING
8	GV	I AM DONE UPLOADING THIS PIECE TO THIS PEER

Fig 5.6 TiniTorrent's Piece Exchange States

6 TiniTorrent Messages

All peer to peer communication uses Peer Wire Protocol discuss in the class. In this project you will need to familiarize yourself with byte level details of these communications. Fig-3 provides a typical sequence of such messages. Below I provide byte level description of messages HANDSHAKE, BITFIELD, UNCHOKE, INTERESTED, REQUEST & PIECE massages. Normally, a TiniTorrent peer opens a connection and then sends a continuous stream of characters. Each message is sent in a length, prefix format. The first handshake message has only one byte length field. The sample handshake message has a total of 68 bytes. After sending handshake continuous byte streams are send which are more messages- still in length, fields prefix. One of the first messages is BITFIELD. This is followed by other messages. Nothing other than the length field at the start of each message tells where message begins and where it ends in this endless stream of characters seen over a connection.

HAND SHAKE		
A peer named " MINI001" is sending a handshake message.		
Bytes	Characters	Comment
1	68 (decimal)	Length
19	"BitTorrent Protocol"	A string
8	0 (decimals)	Reserved.
20	1122FF2040..40	Hash-Info
20	MINI001….. 000	Node ID

BITFIELD		
A seeding peer is sending its bitfield message telling it has all the pieces for a sample file which has 32 pieces and each piece has 256 bytes.		
Bytes	Characters	Comment
4	00 00 00 05 H	Length is 5
1	05 H	
4	FF FF FF FF	.

UNCHOKE		
Bytes	Characters	Comment
4	00 00 00 01 H	Length is 1
1	01 H	

INTERESTED		
Bytes	Characters	Comment
4	00 00 00 01 H	Length is 1
1	02 H	

REQUEST		
A peer is sending request to get 0-255 bytes of the 5-th piece of the sample file which has 32 pieces and each piece has 256 bytes.		
Bytes	Characters	Comment
4	00 00 00 0C H	Length is 12
1	06 H	
4	00 00 00 05 H	
4	00 00 00 00 H	Get from 0^{th}
4	00 00 01 00 H	256 bytes

PIECE		
A peer is sending 0-255 bytes of the 5-th piece of the sample file which has 32 pieces and each piece has 256 bytes.		
Bytes	Characters	Comment
4	00 00 01 09 H	Length is 9
1	07 H	
4	00 00 00 05 H	
4	00 00 01 00 H	

7 Road Map Questions:

Here are some roadmap questions to assist you to start the project. There is no grade for them. But if you can answer them you are ready to implement the project.

1. Which system call in *concurrent.c* creates a process?
2. In Top Level Architecture (Fig-1) who signals Wait(S) and who signals wait(C)?
3. Look into message sequences given into Fig-3. What will happen if peer A and peer B both waits for UNCHOKE and INTERESTED before sending them out?
4. How a process can signal another process? (Hint check system calls and functions: signal(), pause(), wait(), sleep()).
5. Now work out a small example in detail. Consider two peers. Initially MINI-EVEN has all the even indexed pieces and MINI-ODD has all the odd indexed pieces of a file which has 4 pieces- each of size 512 Bytes. Determine the sample message streams when MINI-ODD is trying to get pieces 2, 4 and MINI-EVEN is trying to get pieces 1, 3 from the other. Draw the complete message diagram (like Fig-3) and the message streams in both directions. How many bytes will travel in each direction to complete this transaction?
6. What value is returned by fork() call in *concurrentserver.c*? (Hint check system calls getpid(), fork(), excv()).
7. For the case above hand trace the initial PIECE STATES in MINI-EVEN. How it will change with each messages sent when MINI-EVEN is trying to get piece 1 until it is received?
8. Which program blocks (Fig-1) will potentially write into PIECE STATES sub-area of the TRACKING CARD? Which program blocks will potentially read into it?
9. A leecher initiated slave will alter which piece states and to what values?
10. A seeder initiated slave while uploading will alter which piece states and to what values?
11. Design a small program for "Select a Slave & a Piece" that will look into the current TRACKING CARD and return a slave and a piece.
12. How can you measure the execution time in the Slave? (Hint: check on system calls time(), sleep()).
13. How many times the leecher fork (in main process) can occur? What factors should dictate the limit on it? What factors should dictate the limit on the forks to create Piece Exchange Slaves?
14. Complete the TORRENT CARD table so that it will have all the fields of a torrent file. Show the Initial TORRENT CARD for a music file.
15. Design the detail of the pSTAT field's data structure so that exact download start time, end-time, and the id of the remote peer which has successfully supplied the piece all can be logged for each of the pieces and printed out at the end for plotting.
16. Multiple "slaves" will be running concurrently. Which fields of the TRACKING CARD might need to me guarded for mutual exclusion?