Building and Evaluating a Web Server

Kushal Lakhotia
Department of Computer Science, Purdue University

1. Introduction

The report discusses the system design of the web server, its implementation, the methodology of evaluating it and finally the performance report. The web server supports HTTP 1.0 and 1.1 protocols. The protocol can be specified during server startup along with the port that the server will listen on and the server time-out. The performance evaluation was done using a synthetic client that spawns multiple processes each sending HTTP requests to the server. The connection time (time to establish and tear off the TCP connection) and transmission time (time to send the data) were calculated for files of ten different sizes of and appropriate graphs are presented for representing the results.

2. System Architecture & Design

The system has been designed in two levels. The top layer is an HTTP handler and let us call it the HTTP layer. The HTTP layer manages the implementation of the HTTP protocol. It layer has no information about how the server interacts with clients, that is, it is not concerned with how to retrieve client requests, how to send client requests, how to manage connections for more than client and what to do after the request is served. It takes an HTTP request, processes it and sends back the response. The HTTP layer sit on top of the layer called the Service layer. The Service layer is responsible for listening on a specific port, receive client request, passing it on to the HTTP layer, receive the response the from HTTP layer, send the HTTP response to the client and decide what to do after a client request has been served depending on whether the server supports HTTP 1.0 or 1.1 protocol. It also maintains the list of active clients and updates them as and when the connections are closed. Fig. 1 illustrates the high level handling of the client request and server response.

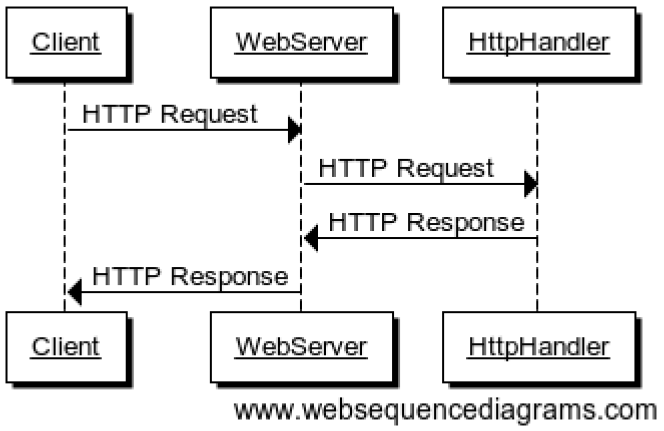


Fig. 1

3. Implementation

The implementation consists of the main function, the WebServer class and the HttpHandler class. The main function accepts the arguments passed (if any) by the user, initializes the server parameters, that is, the port number and the server time-out. It then calls the runWebServer() method of the WebServer class to start the web server. The layers discussed above have been implemented by the classes WebServer and HttpHandler. The WebServer class implements the Service layer and the HttpHandler implements the HTTP layer.

3.1 Pseudo-code of the implementation

```
Main:
  begin
        If user has passed arguments then parse the arguments and initialize server port and time-out
        Else initialize server port and time-out with the default values
        Create an object of WebServer class and call runWebServer() with the values initialized above
        Return the values returned by runWebServer()
  end

Service Layer:
  forever loop
    begin
        Open and bind a socket at the specified port
        Start listening on that port
        If a client is available at a socket to read from then
                If the socket is the listener socket
                        Accept the connection
                        Update the data structures for maintaining client information
                Else
                        Read the client request from the socket
                        Create an object of HttpHandler and call serveConnection() with client request
        Else
                Send the client response stored in the buffer to the client
                If whole response has been sent then
                        Update data structures to remove client information
                        Close the connection
                Else call serveConnection() to send the next chunk of response
    end

HTTP Layer:
  begin
        Receive the client request and parse it for correctness and extract “what to do”
        Check “what is to be done” and call the appropriate method for that
        Compose the response in parts:
                If it is a new request
                        Compose the status line of the response
                        Compose the headers of the response
                Else
                        Compose the message to be sent (if any)
        Return the composed response
  end
```

3.2 Class diagram

The class diagram below of the two classes used in the implementation show the member variables and the methods.

WebServer	HttpHandler
Private:	Private:
unsigned int serverTimeout;	bool clientHttp10;
unsigned int currentTimeout;	bool serverHttp10;
size_t sizeOfResponse[MAX_NUM_OF_CONNS];	bool isNew;
HttpHandler *myHttpHandler[MAX_NUM_OF_CONNS];	int methodType;
FILE *connectionVsFileSize;	int statusCode;
FILE *transmissionVsFileSize;	int serverTimeout;
time_t timer[MAX_NUM_OF_CONNS];	
bool serverHttp10;	size_t sizeOfResponse;
bool done[MAX_NUM_OF_CONNS];	size_t fileSize;
bool timedOut[MAX_NUM_OF_CONNS];	size_t dataLeft;
char *serverPort;	size_t bufferSize;
char *response[MAX_NUM_OF_CONNS];	
int listenerSocket;	FILE *fileToFetch;
int nextActiveConn;	FILE *fileToWrite;
int numOfConns;	
int activeConns[MAX_NUM_OF_CONNS];	char *request;
int connTime[MAX_NUM_OF_CONNS];	char *URL;
int transTime[MAX_NUM_OF_CONNS];	char *version;
	char *statusPhrase;
	char *statusLine;
	char *header;
	char *response;
	char *msgBuffer;
	char *clientMsg;
	Public:
	size_t fileSizeForPublic;
Private:	Private:
int listenAndAccept();	bool parseRequest();
int findPosOfConn(int);	bool myStrCmp(const char *, int, const char *, int);
int findNextActiveConnSlot();	void composeStatusLine();
int findNumOfActiveConns();	void composeHeader();
	void composeResponse();
Public:	void getResponse();
int runWebServer();	void responseGetMethod();
	void responseHeadMethod();
	void responsePutMethod();
	void responseDeleteMethod();
	void responseTraceMethod();
	void responseOptionsMethod();
	void responseBadRequest();
	void responseTimeOut();

```
Pubic:
        bool serveConnection(char *, char **, size_t *, bool);
```

Fig. 2

4. Experiment Setup for Performance Evaluation

4.1 Methodology

The server code is profiled to calculate the time to establish and tear down the TCP connection and the time to transmit data to client. The server was started and the client then sent the same request to the server by spawning twenty-five child processes. The connection time and transmission time are thus calculated for all the twenty-five connections for the file of the same size. This experiment is run five more times by varying the file sizes. For each file size the average connection time and transmission time are calculated and aggregated. The aggregated result is then used to plot the graphs.

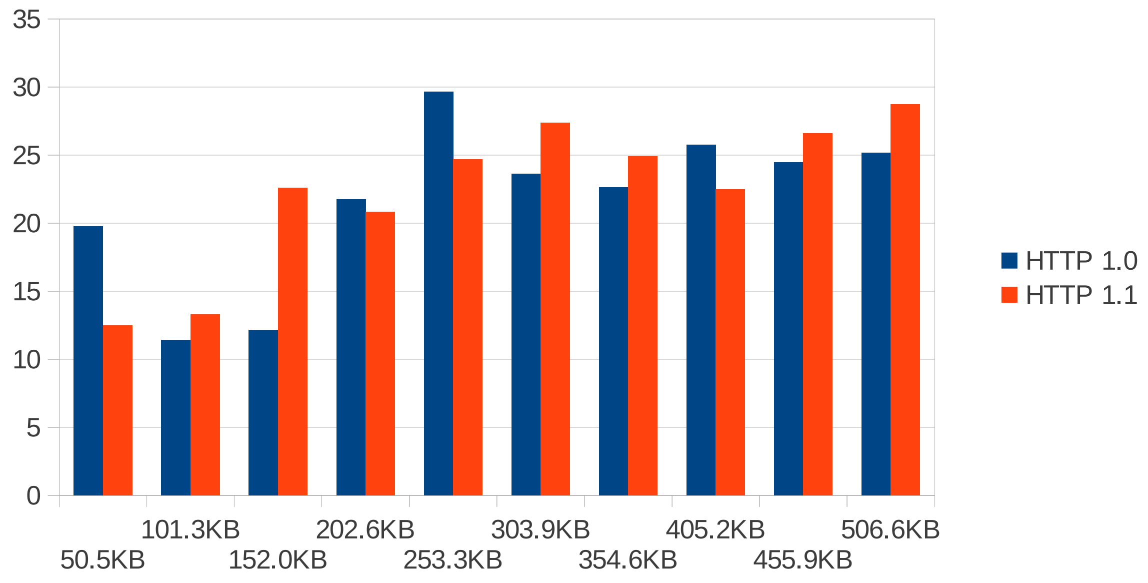
4.2 Evaluation Environment

The HTTP server was run in a Linux machine in the Linux Lab of Department of Computer Science, Purdue University. The system has Intel Pentium 3.00 GHz processor and 3.8GB of RAM. Copies of the client were run on several machines located in the same lab and those machines have the same system configuration as the server machine. The network used for evaluation was the Purdue LAN.

5. Results of Performance Evaluation

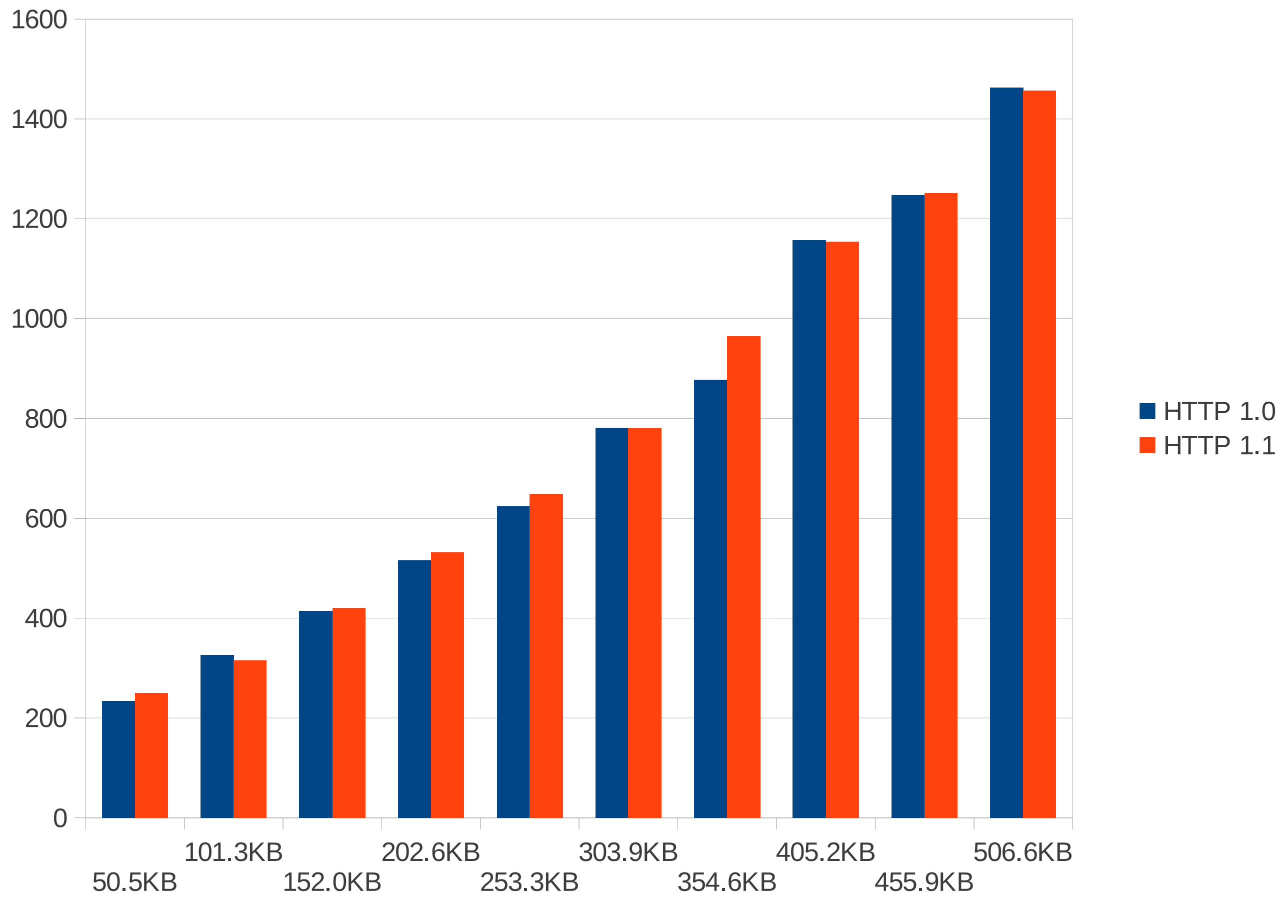
5.1 Connection Time vs File Size

In the chart below, the Y-axis has the connection time in micro-seconds.



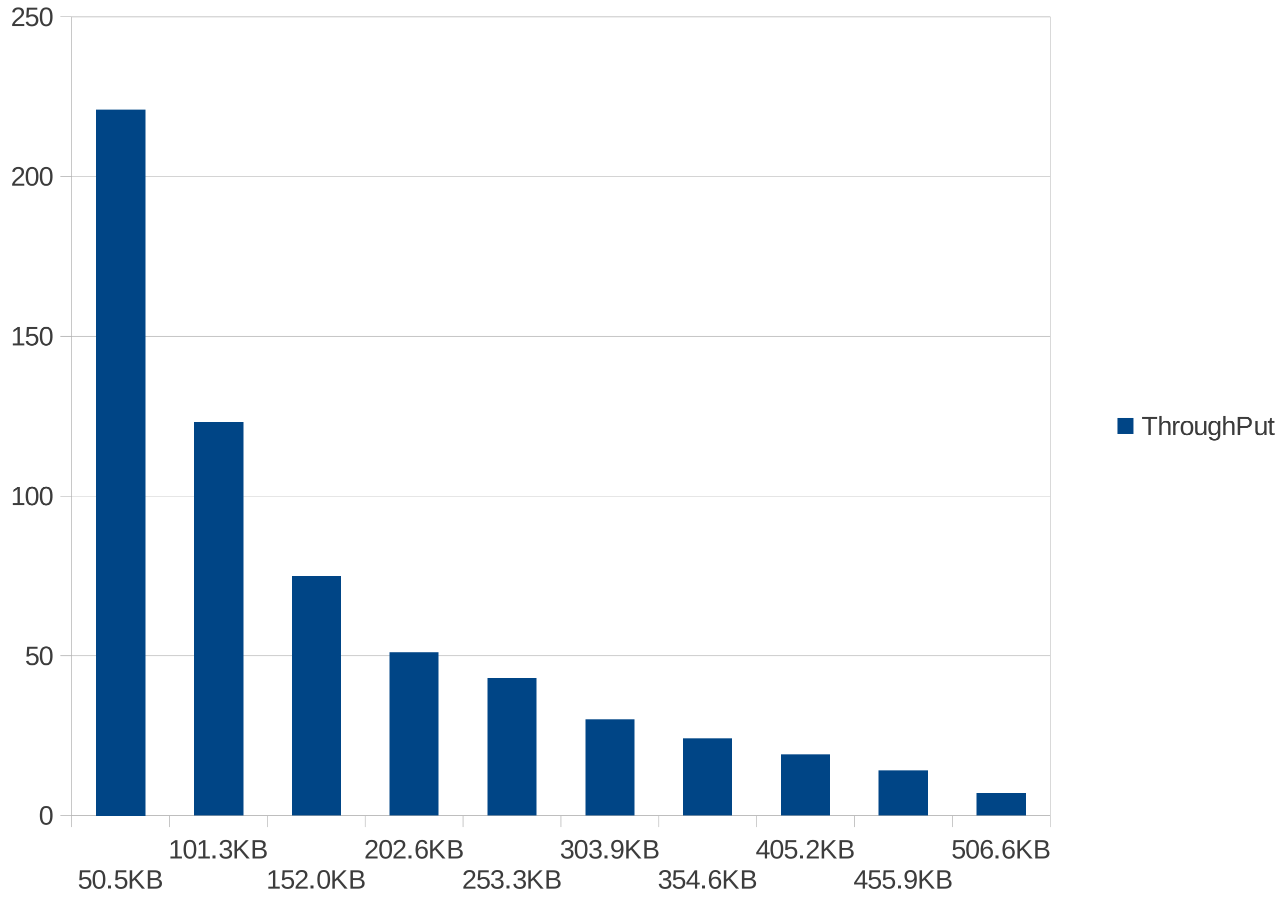
5.2 Transmission Time vs File Size

In the chart below, transmission time is along Y-axis.



5.3 Throughput vs File Size

In the chart below throughput is expressed as number of requests served per second.



250
200
150
100
50
0
50.5KB
101.3KB
152.0KB
202.6KB
253.3KB
303.9KB
354.6KB
405.2KB
455.9KB
506.6KB
ThroughPut