

Input Sensitive Profiling on FileZilla, FTP Solution



By-
Kiran Sheena Marakala
Pratibha Jagadeesh Dixith
Ritwij Nadagouda

Table of Contents

1	Abstract	3
2	Introduction to FileZilla	4
2.1	Features of FileZilla	4
2.2	FileZilla Client Functions	4
3	A Note on FTP	6
4	A Note on AProf	7
4.1	Read Memory Size	7
4.2	Cost	7
4.3	Calls	7
4.4	Building and using aprof plot:	7
5	Performance Metrics for High Cost Methods	9
5.1	Analysis of Performance Metrics on Methods	9
6	Performance Metrics of Multithreaded Methods	32
6.1	Multithreaded Methods	32
7	Challenges Faced	39
8	References	39

1 Abstract

Input-sensitive profiling is a recent performance analysis technique that makes it possible to estimate the empirical cost function of individual routines of a program, helping developers understand how performance scales to larger inputs and pinpoint asymptotic bottlenecks in the code. A current limitation of input-sensitive profilers is that they specifically target sequential computations, ignoring any communication between threads. We develop new metrics for automatically estimating the size of the input given to each routine activation, addressing input produced by non-deterministic memory stores performed by other threads as well as by the OS kernel (e.g., in response to I/O or network operations). We provide real case studies, showing that our extension allows it to characterize the behavior of complex applications more precisely than previous approaches. An extensive experimental investigation on a variety of benchmark suites (including the SPEC OMP2012 and the PARSEC benchmarks) shows that our Valgrind-based input-sensitive profiler incurs an overhead comparable to other prominent heavyweight analysis tools, while collecting significantly more performance points like self-cost, cumulative cost, number of calls, average cost, minimum cost, RMS(Read memory size) from each profiling session and correctly characterizing both thread-induced and external input.

2 Introduction to FileZilla

FileZilla Client is a fast and reliable cross-platform FTP, FTPS and SFTP client with lots of useful features and an intuitive graphical user interface.

2.1 Features of FileZilla

- Easy to use
- Supports FTP, FTP over SSL/TLS (FTPS) and SSH File Transfer Protocol (SFTP)
- Cross-platform. Runs on Windows, Linux, *BSD, Mac OS X and more
- IPv6 support
- Available in many languages
- Supports resume and transfer of large files >4GB
- Tabbed user interface
- Powerful Site Manager and transfer queue
- Bookmarks
- Drag & drop support
- Configurable transfer speed limits
- Filename filters
- Directory comparison
- Network configuration wizard
- Remote file editing
- Keep-alive
- HTTP/1.1, SOCKS5 and FTP-Proxy support
- Logging to file
- Synchronized directory browsing

2.2 FileZilla Client Functions

FileZilla has the below main functionalities.

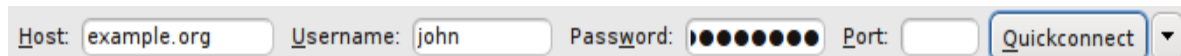
1. Connect to an FTP server

The first thing to do is connecting to a server.

This is our (fictional) login data - please use your own data instead if you want to actively follow the tutorial.

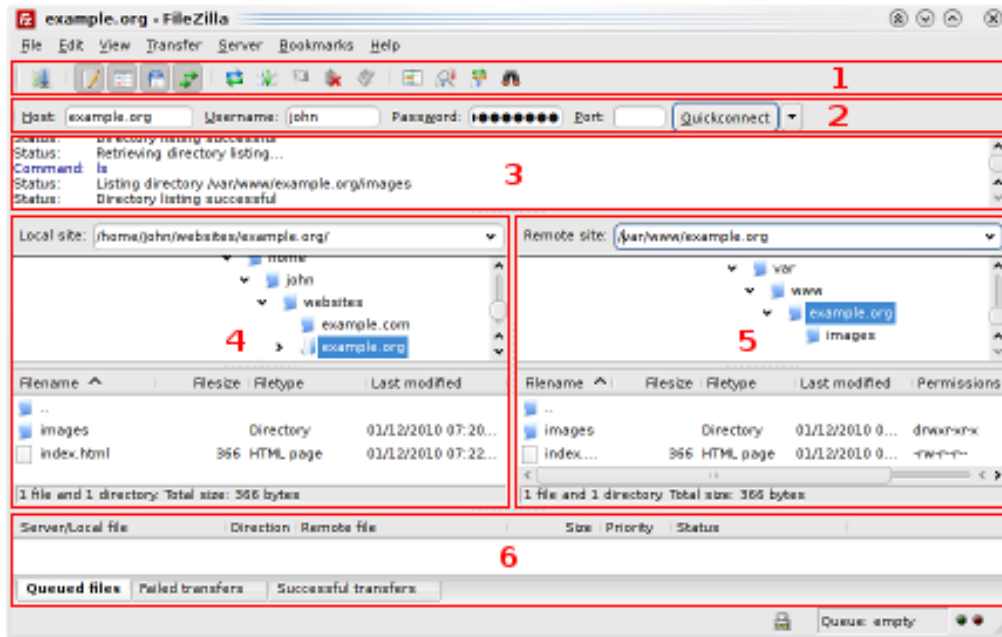
```
Hostname: example.org
Username: john
Password: 7PjU#.J3
```

We will use the Quickconnect bar for establishing the connection:

The image shows the Quickconnect bar from the FileZilla client. It contains four input fields: 'Host' with 'example.org', 'Username' with 'john', 'Password' with masked characters (dots), and 'Port' which is empty. To the right of these fields is a 'Quickconnect' button with a dropdown arrow.

Enter the hostname into the Quickconnect bar's Host: field, the username into the Username: field as well as the password into the Password: field. You may leave the Port: field empty unless your login information specifies a certain port to use. Now click on Quickconnect.

2. Navigating and window layout

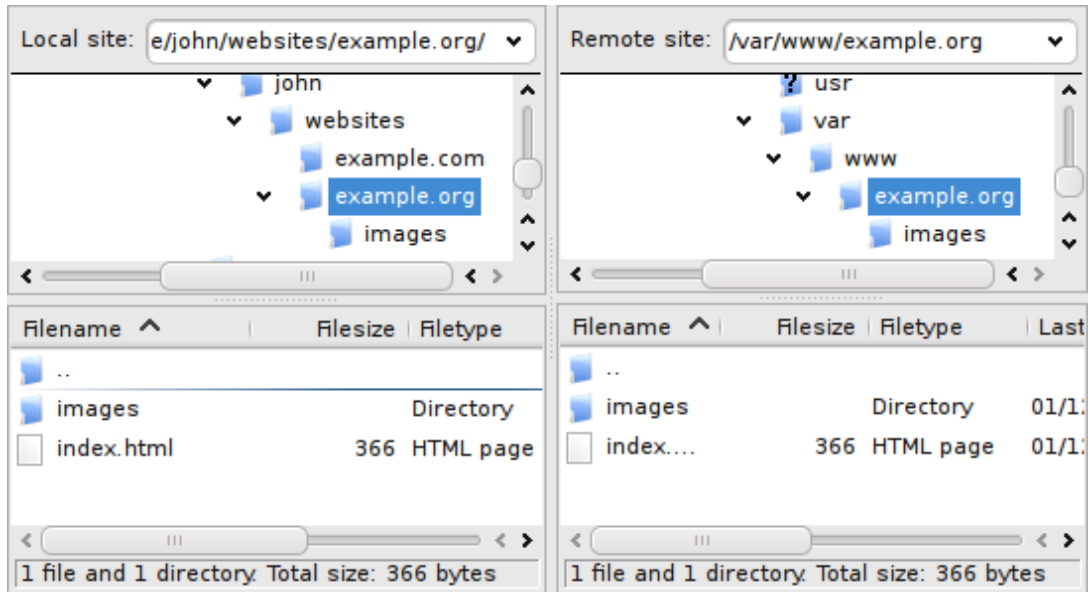


Below the *toolbar* (1) and *quick connect bar* (2), the *message log* (3) displays transfer and connection related messages. Below, you can find the file listings. The left column (*local pane*, 4) displays the local files and directories, i.e. the stuff on the PC you're using FileZilla on. The right column (*server pane*, 5) displays the files and directories on the server you are connected to. Both columns have a directory tree at the top and a detailed listing of the currently selected directory's contents at the bottom. You can easily navigate either of the trees and lists by clicking around like in any other file manager. At the bottom of the window, the *transfer queue* (6) lists the to-be-transferred and already transferred files.

3. Transferring files

Uploading:

First - in the local pane - bring the directory into view which contains data to be uploaded (e.g. index.html and images/). Now, navigate to the desired target directory on the server (using the server pane's file listings). To upload the data, select the respective files/directories and drag them from the local to the remote pane. You will notice that the files will be added to the transfer queue at the bottom of the window and soon thereafter get removed again - since they were (hopefully, if nothing went wrong) just uploaded to the server. The uploaded files and directories should now be displayed in the server content listing at the right side of the window.



Downloading:

Downloading files, or complete directories, works essentially the same way as uploading - you just drag the files/directories from the remote pane to the local pane this time, instead of the other way round.

4. Using Site Manager

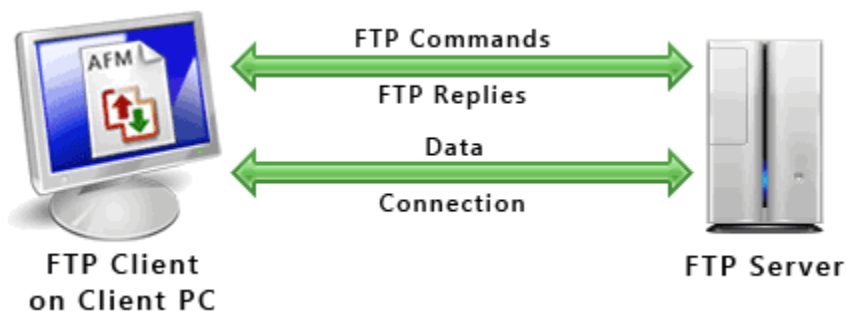
We can also add the server information to the site manager to make it easy to reconnect to this server. To do this, select *Copy current connection to Site Manager...* in the *File* menu. The site manager will be opened and a new entry will be created with all the important information already filled in. You will notice that the entry's name is selected and highlighted - you can enter some descriptive name so you will later on find your server again (enter something like *domain.com FTP server* for example - you can rename it later if you wish). Now close the dialog by clicking on *OK*.

The next time you want to connect to this server, you can simply select it in the site manager and click *Connect*.

3 A Note on FTP

File Transfer Protocol (FTP) is a standard Internet protocol for transmitting files between computers on the Internet over TCP/IP connections.

FTP is a client-server protocol that relies on two communications channels between client and server: a command channel for controlling the conversation and a data channel for transmitting file content. Clients initiate conversations with servers by requesting to download a file. Using FTP, a client can upload, download, delete, and rename, move and copy files on a server. A user typically needs to log on to the FTP server, although some servers make some or all of their content available without login, also known as anonymous FTP.



4 A Note on AProf

Aprof is a Valgrind tool which from one or more runs of a program automatically measures how the performance of individual routines scales as a function of the input size.

Below are the few Performance metrics that we have considered.

4.1 Read Memory Size

Definition: The read memory size (RMS) of the execution of a routine f is the number of distinct memory cells first accessed by f , or by a descendant of f in the call tree, with a read operation.

The main idea is that cells that are accessed by a function for the first time with a read operation contain the input values of the routine. Conversely, if a cell is first written and then read by the routine, the read value is not part of the input as it was determined by the routine itself. We notice that the RMS definition, which is based on tracing low-level memory accesses made by the program, supports memory dereferencing and pointers in a natural way.

4.2 Cost

We use the generic term cost to refer to any performance metric, e.g., time, number of executed basic blocks and so on.

4.3 Calls

It gives the number of calls of that function while executing an operation.

4.4 Building and using aprof plot:

Aprofplot is a Java based GUI to analyze the .aprof files generated by the aprof input sensitive profiler. So, we need JRE, JDK and the build tool Apache ANT to be installed before using aprofplot.

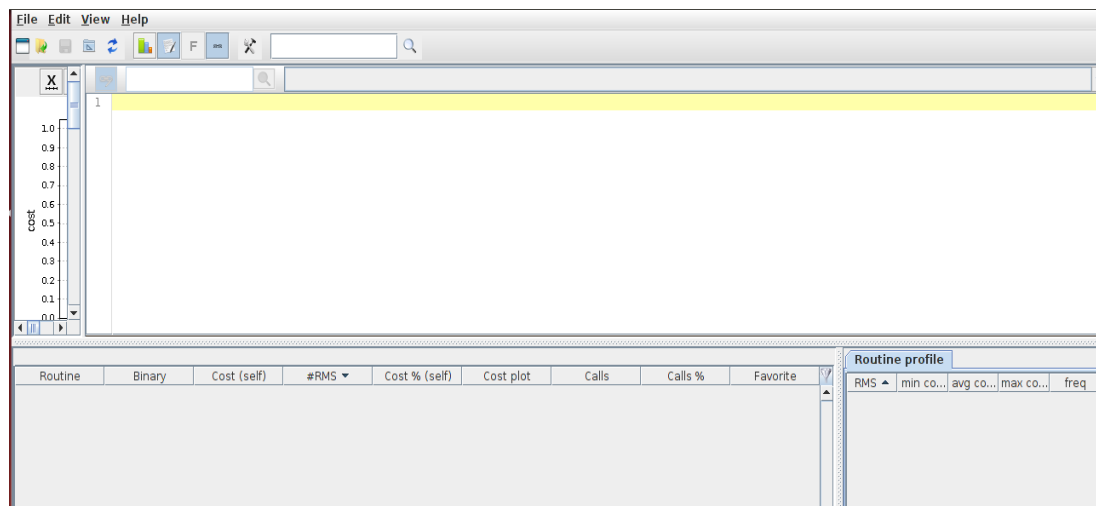
Follow the below steps to install aprofplot.

- `sudo apt-get install openjdk-7-jre`
- `sudo apt-get install openjdk-7-jdk`
- `sudo apt-get install ant`
- Download the aprof source code from <https://github.com/ercoppa/aprof>.
- Change directory to aprofplot which is inside the root directory aprof via Terminal and type the command 'ant' to build aprofplot. If this gives any error then check the build-impl.xml present in /aprof/aprofplot/nbproject folder for the issues with classpath. We had to hardcode the classpath in this build xml to successfully build aprofplot.
- Once the aprofplot is built successfully, it will create a folder directory dist inside aprofplot which will have a jar aprof-plot.jar which was built using ant. Now to run aprofplot, change directory to dist and run the below command:

```
java -cp "aprof-plot.jar:/home/kiran/Desktop/aprof/aprof-master/aprofplot/lib/rsyntaxtextarea.jar:/home/kiran/Desktop/aprof/aprof-master/aprofplot/lib/jcommon-1.0.21.jar:/home/kiran/Desktop/aprof/aprof-master/aprofplot/lib/jfreechart-1.0.17.jar:/home/kiran/Desktop/aprof/aprof-master/aprofplot/lib/jama.jar" aprofplot.Main
```

The actual path of the jars above will change according to corresponding directory structure.

This will open the aprofplot as shown below:



- To install the tool aprof which generates the .aprof files, change directory to valgrind directory in the terminal and run the shell script build.sh using the below command. This will install all the required dependencies and create the binary.

```
./build.sh
```

Now, open Filezilla using the below command


```
...../valgrind/inst/bin/valgrind --tool=aprof /usr/local/bin/filezilla
```

The actual paths might change according to the directory structure.

5 Performance Metrics for High Cost Methods

The top ten methods that are responsible for high value in Performance measurements like Read Memory Size (RMS), cost and calls.

Method Name
1) void CStatusLineCtrl::OnPaint(wxPaintEvent&)
2) wxString CSizeFormatBase::Format(COptionsBase* pOptions, int64_t size, bool add_bytes_suffix /*=false*/)
3) void CStatusLineCtrl::DrawProgressBar(wxDC& dc, int x, int y, int height, int bar_split, int permill)
4) wxString CSizeFormatBase::FormatNumber(COptionsBase* pOptions, int64_t size, bool* thousands_separator /*=0*/)
5) void CStatusLineCtrl::OnTimer(wxTimerEvent&)
6) void CLed::OnTimer(wxTimerEvent& event)
7) wxString CSizeFormat::Format(int64_t size, bool add_bytes_suffix /*=false*/)
8) void CStatusLineCtrl::SetTransferStatus(CTransferStatus const& status)
9) bool CFileZillaApp::FileExists(const wxString& file) const
10) bool CWrapEngine::WrapText(wxWindow* parent, int id, unsigned long maxLength)

5.1 Analysis of Performance Metrics on Methods

1. void CStatusLineCtrl::OnPaint(wxPaintEvent&)

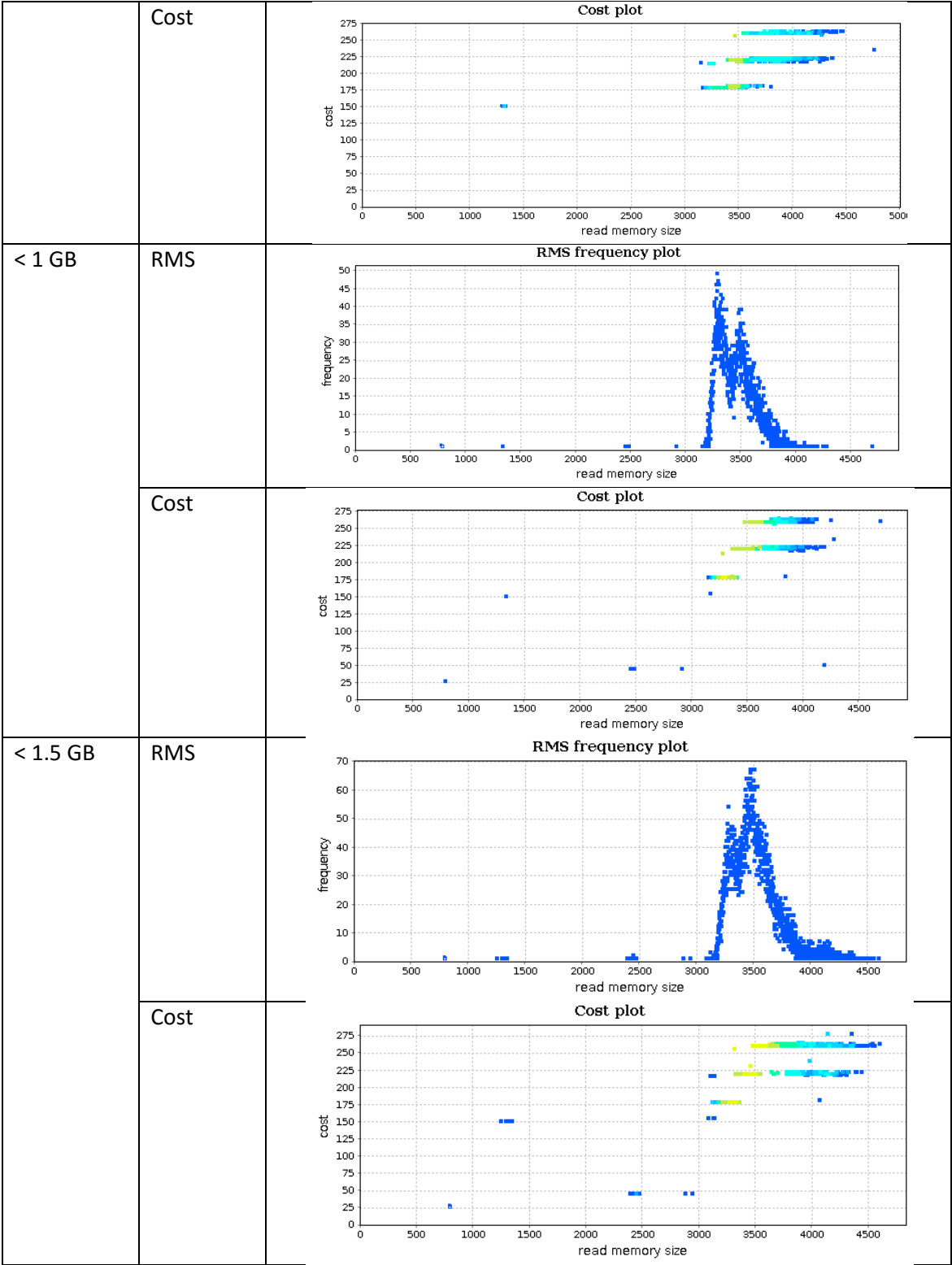
This is an event handler for the paint event. This method is responsible for the update of progress bar which shows the progress of the file being transferred. As this status has to be updated constantly, the number of calls to this method increases as the size of the file being transferred increases and the number of files being transferred increases.

The method first checks if there is any update to be done to the status line and it updates if there is change by getting the following details: the speed at which the files are being transferred, the elapsed time and other required information to display the progress bar.

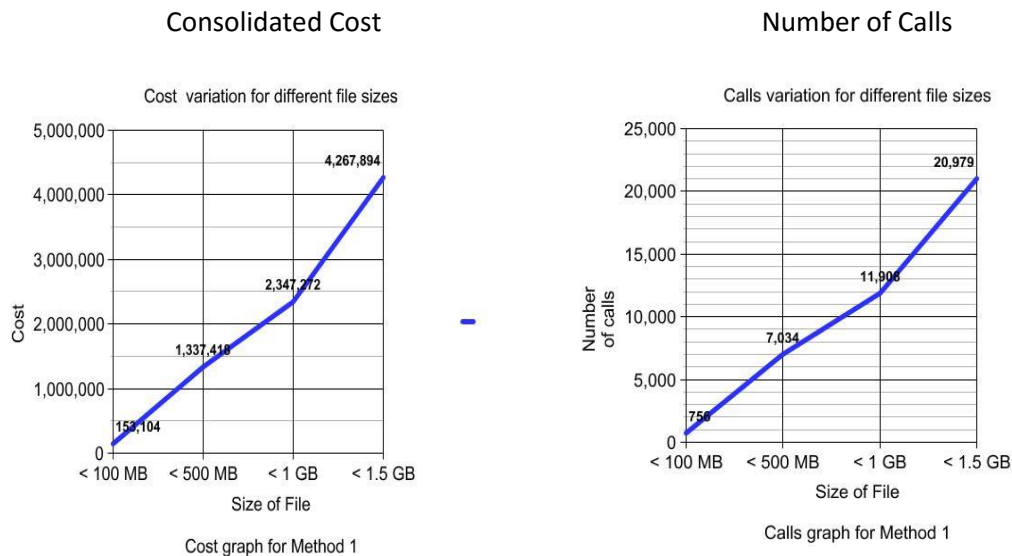
The x-axis of the plots are mapped to Read Memory Size (RMS) of the routine and the y-axis is mapped to the frequency distribution of input size for the various calls to this routine. We can observe in these plots that the Graph almost remains same with respect to the Read Memory Size for the files with size greater than 500MB since this does not depend on the size or number of files being transferred. The memory cells first read by a routine does not change for a particular invocation. But we can see that the frequency is distributed and varies as the size and number of files being transferred increases, since the file transfer time increases which means more number of calls to this routine.

The table also shows the Cost Plot associated with the invocation of this routine with different inputs. The cost here can refer to any performance metric like time, number of executed basic blocks. The Cost is also measured against the Read Memory Size. We observe that the Cost increases as the RMS increases which in turn increases with the increase in the size and the number of files being transferred.

Size of File	Metric	Plot
< 100 MB	RMS	
	Cost	
< 500 MB	RMS	



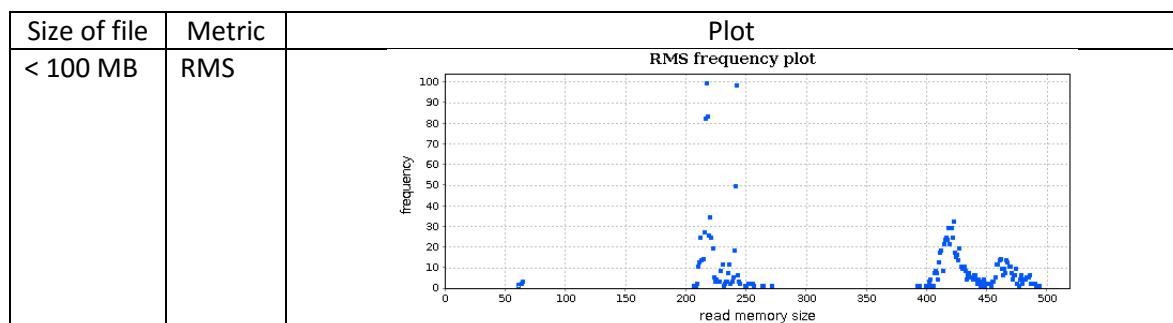
The below graph shows the variation of performance metric for files of different size.

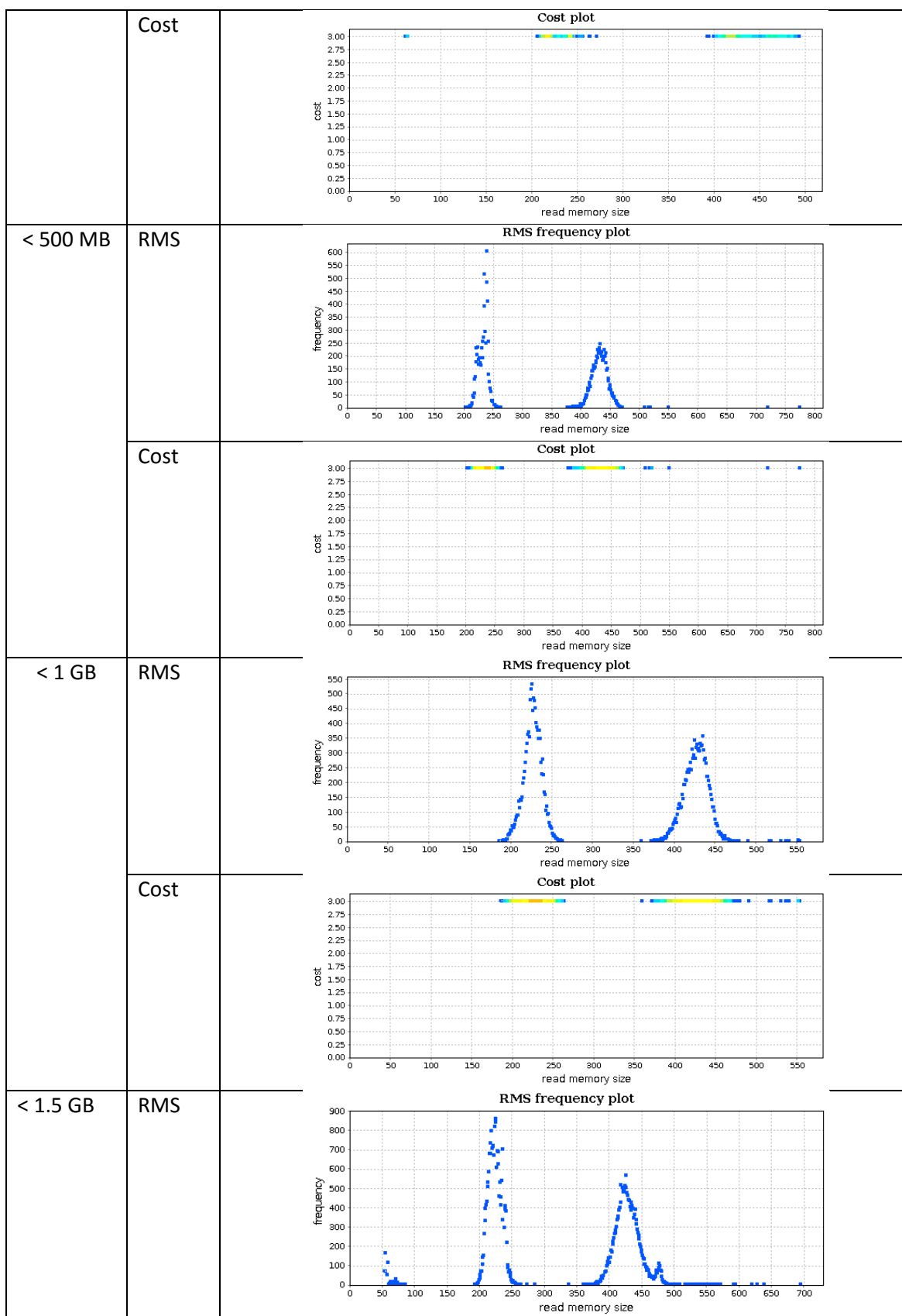


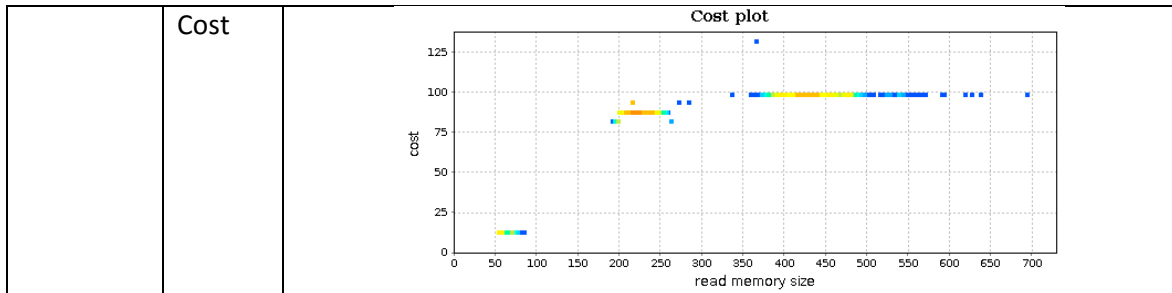
2. wxString CSizeFormatBase::Format(COptionsBase* pOptions, int64_t size, bool add_bytes_suffix /*=false*/)

This method is called from the above onPaint method to format the text (like number of bytes) displayed in the progress bar. This method is called only after a time elapse of 1000ms and a change in the rate of download/upload speed. Hence there is a variation in the frequency distribution with the increase in the file size. We observe that the cost involved with this method does not change with the change in the size since there is no change in the flow within the method.

The aprof plot graphs based on different input sizes on operations like downloading from FTP server are,

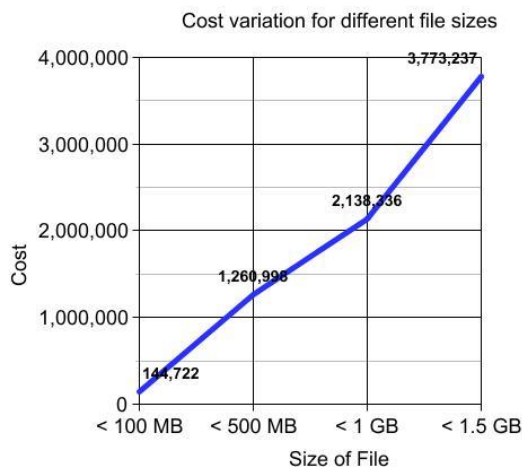






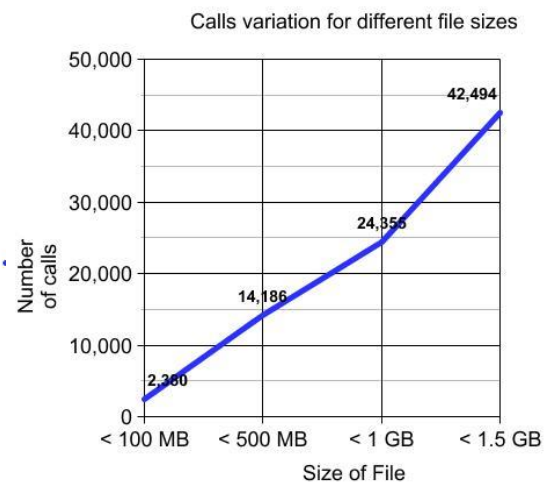
The below two plots shows a consolidated view of Cost and the number of calls. We see that both the parameters grow linearly as the size of the files increases.

Consolidated Cost



Cost graph for Method 2

Number of Calls



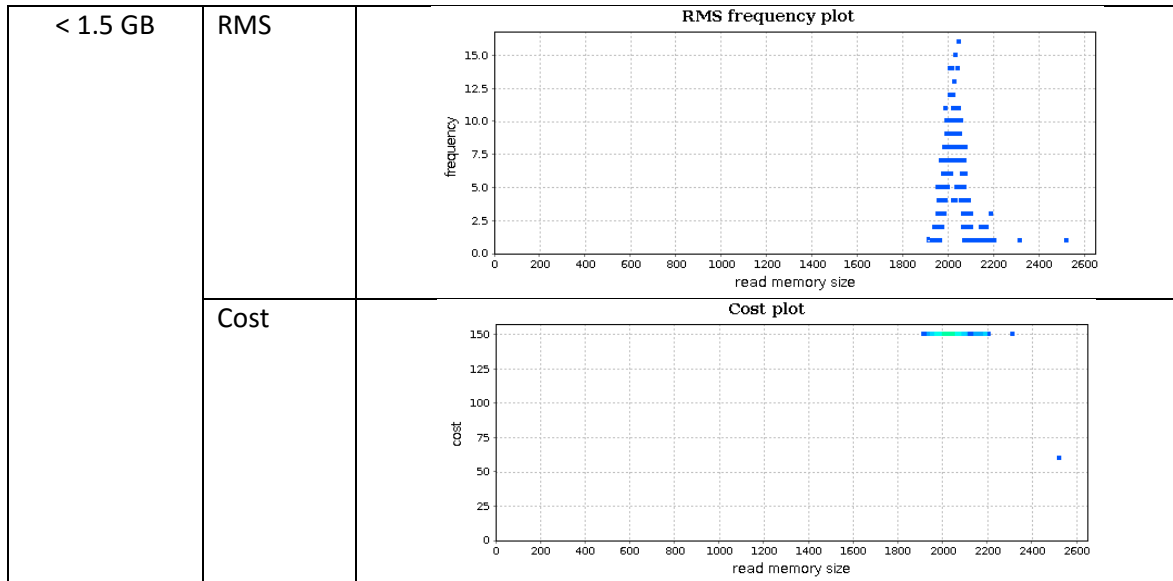
Calls graph for Method 2

3. void CStatusLineCtrl::DrawProgressBar(wxDC& dc, int x, int y, int height, int bar_split, int permill)

This method is responsible for drawing the progress bar shown in the Filezilla Client while transferring the files between the FTP server and the local system. This method accepts a wxWdiget direct context and draws the progress bar on it. It also accepts the height and the coordinates where this progress bar has to be drawn.

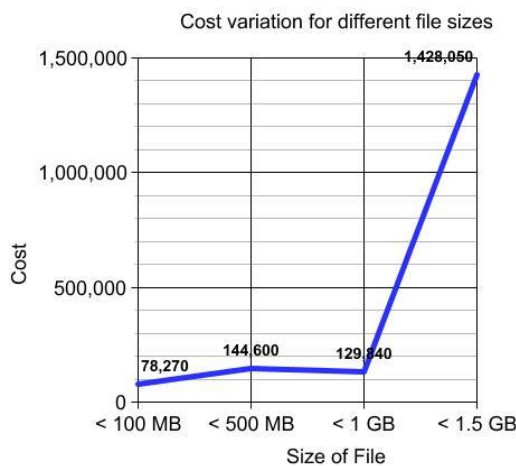
The below table shows the RMS and Cost plots for various sized files and different number of files. The RMS plot shows the frequency distribution against the various input size. We can see the increase (see the values on y axis of the plot) in this distribution as the file size increases. The cost remains almost the same for the various different inputs.

Size of file	Metric	Plot
< 100 MB	RMS	<p>RMS frequency plot</p>
	Cost	<p>Cost plot</p>
< 500 MB	RMS	<p>RMS frequency plot</p>
	Cost	<p>Cost plot</p>
< 1 GB	RMS	<p>RMS frequency plot</p>
	Cost	<p>Cost plot</p>



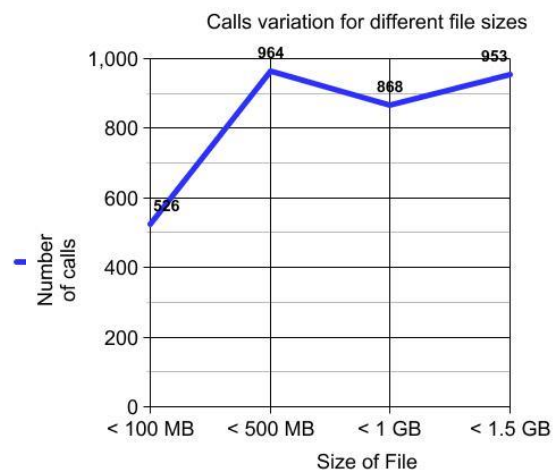
The below two plots show a consolidated view of the cost and number of calls associated with this method for the different inputs.

Consolidated Cost



Cost graph for Method 3

Number of Calls

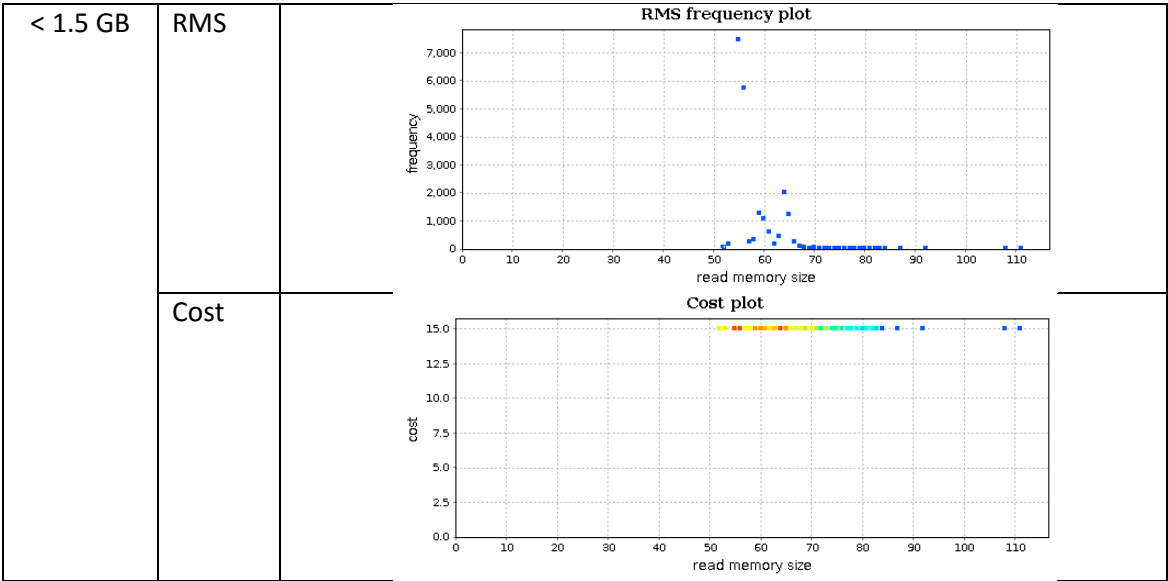


Calls graph for Method 3

4. `wxString CSizeFormatBase::FormatNumber(COptionsBase* pOptions, int64_t size, bool* thousands_separator /*=0*/)`

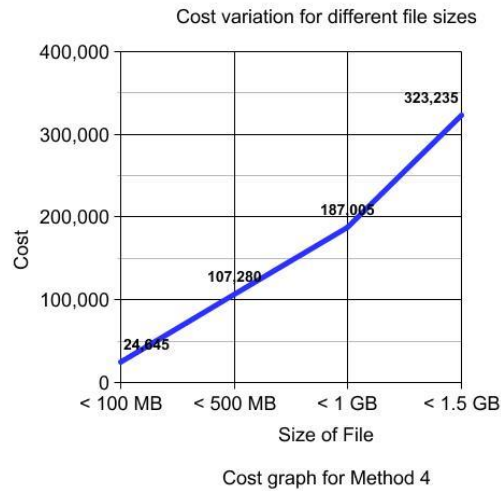
This method is responsible for formatting the number displayed on the progress bar which represents the number of bytes already transferred between the FTP server and the local. As this method does not involve any complex operations the RMS frequency plot and the cost plot remains consistent throughout different inputs.

Size of File	Metric	Plot
< 100 MB	RMS	
	Cost	
< 500 MB	RMS	
	Cost	
< 1 GB	RMS	
	Cost	

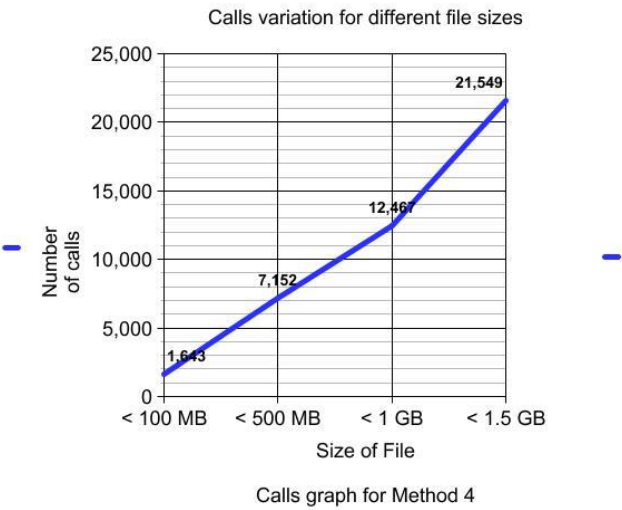


The below two plots show a consolidated view of the cost and the number of calls of this methods see that both the parameters grow linearly for different kind of inputs.

Consolidated Costs



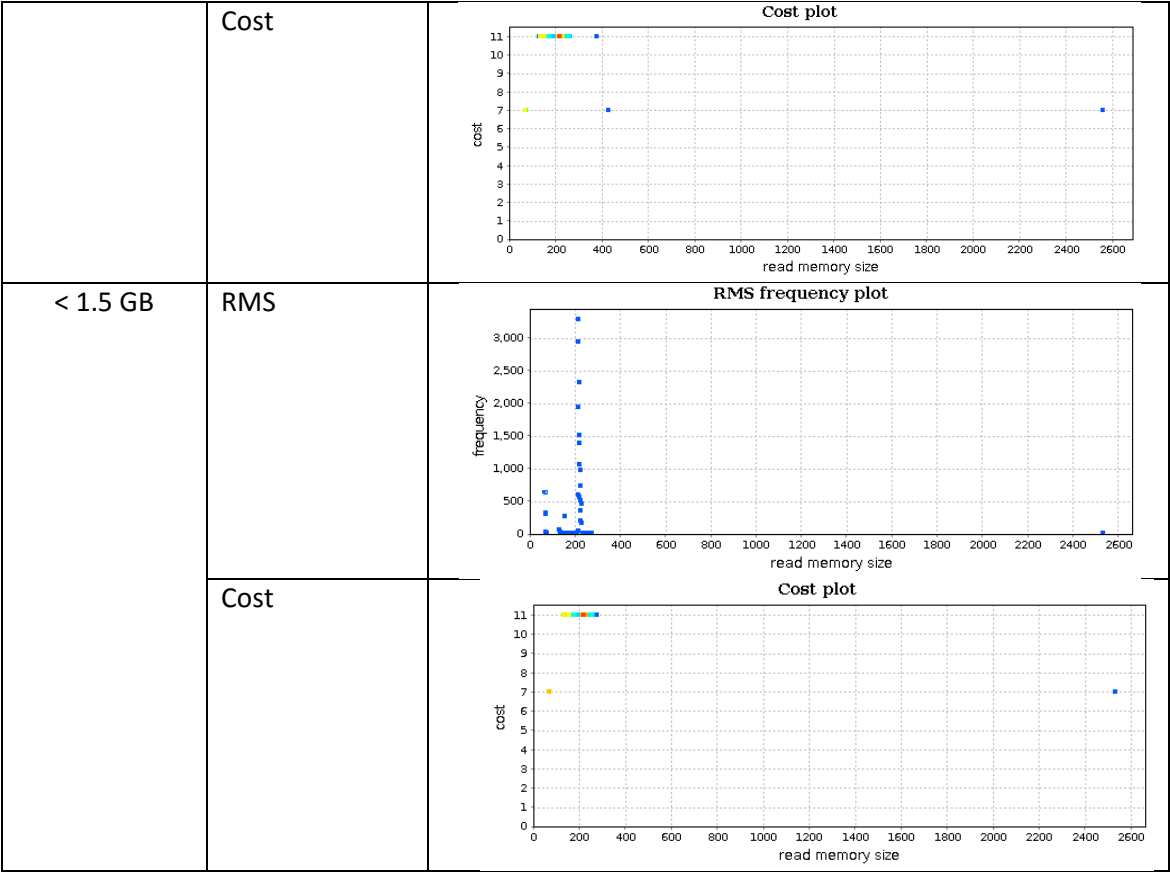
Number of Calls



5. void CStatusLineCtrl::OnTimer(wxTimerEvent&)

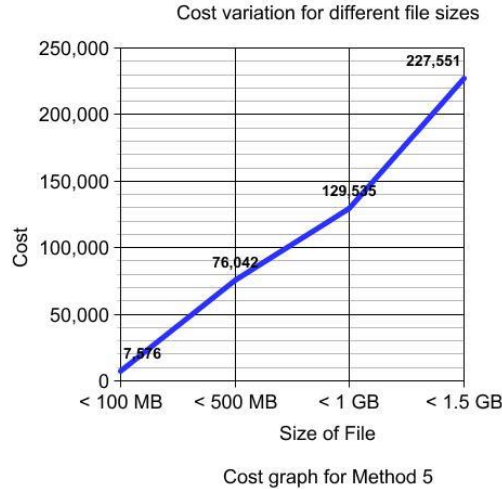
This method is an event handler for the Timer Event which checks the transfer status of an item and see if it has made any progress.

Size of file	Metric	Plot
< 100 MB	RMS	
	Cost	
< 500 MB	RMS	
	Cost	
< 1 GB	RMS	

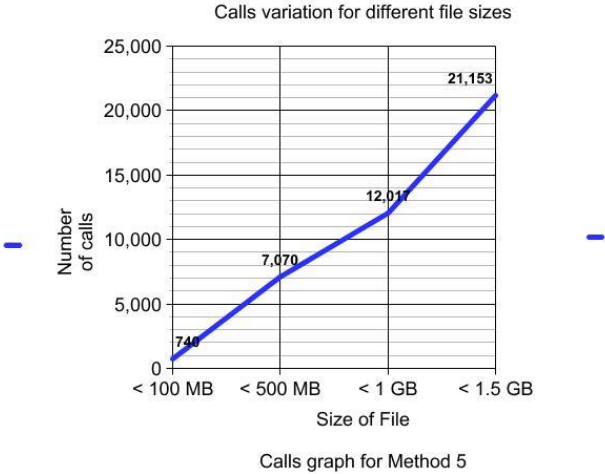


The below two plots show the Consolidated view of total costs and number of calls. We can see that the cost and number of calls grow linearly as the size of the file increases.

Consolidated Costs



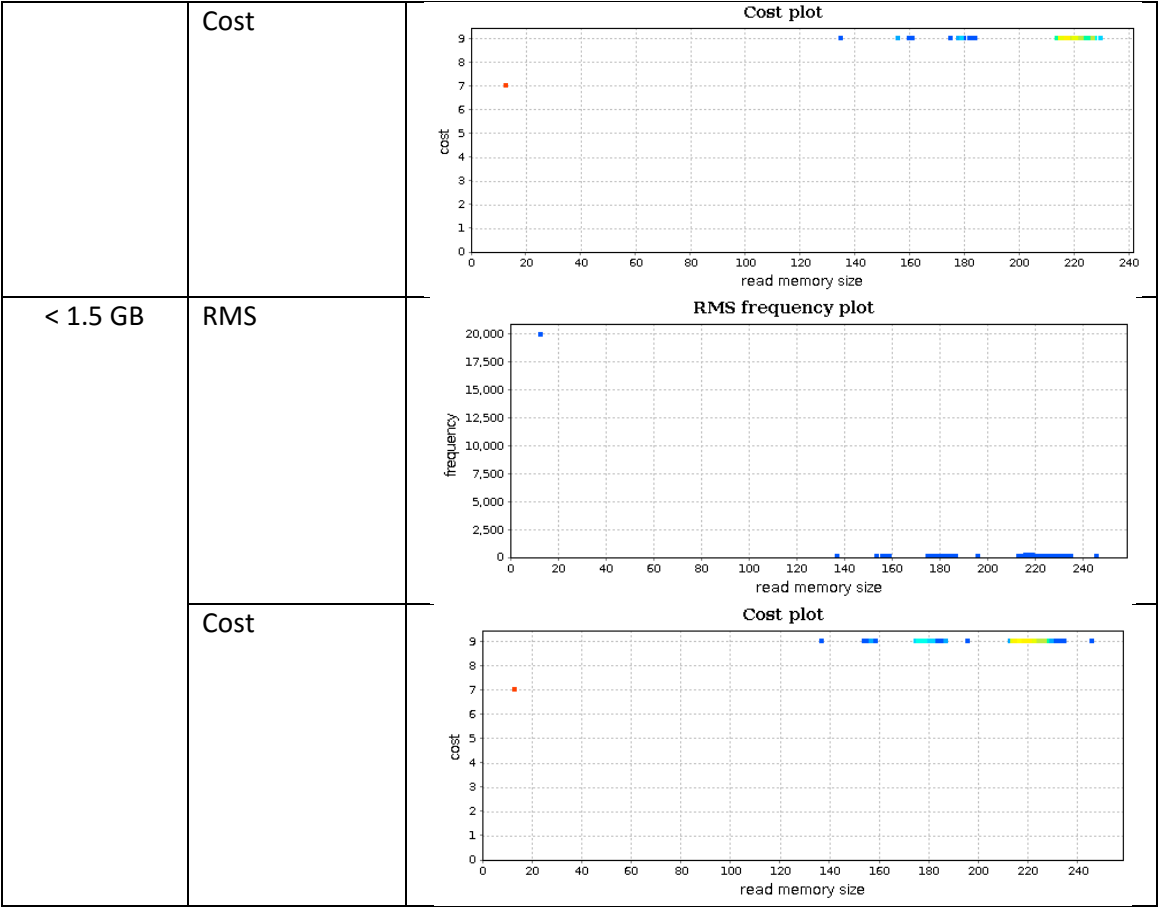
Number of Calls



6. void Cled::OnTimer(wxTimerEvent& event)

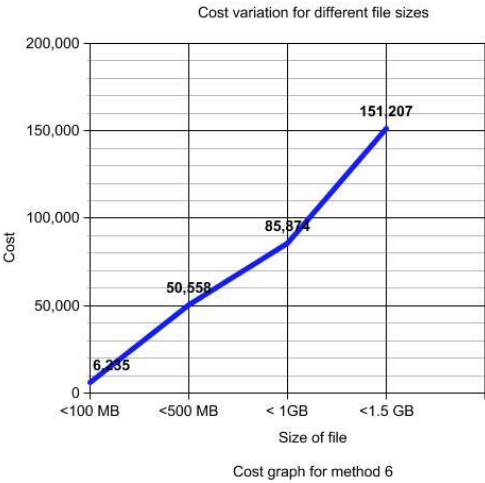
This method is an event handler for the Timer Event which checks if the FileZillaEngine is Active for the timer to be displayed.

Size of file	Metric	Plot
< 100 MB	RMS	
	Cost	
< 500 MB	RMS	
	Cost	
< 1 GB	RMS	

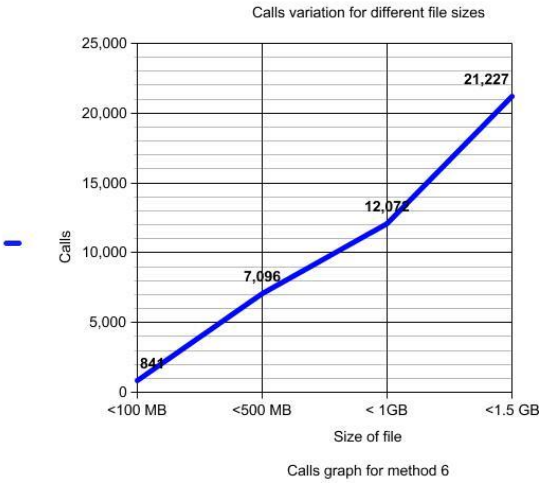


The below two plots show the Consolidated view of total costs and number of calls. We can see that the cost and number of calls grow linearly as the size of the file increases.

Consolidated Costs

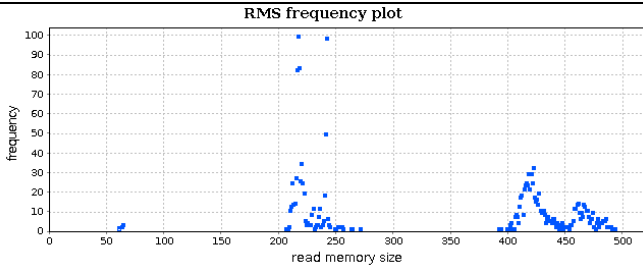
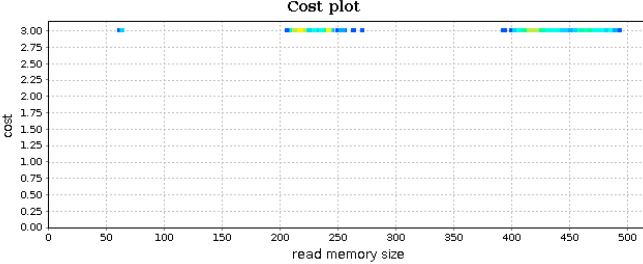
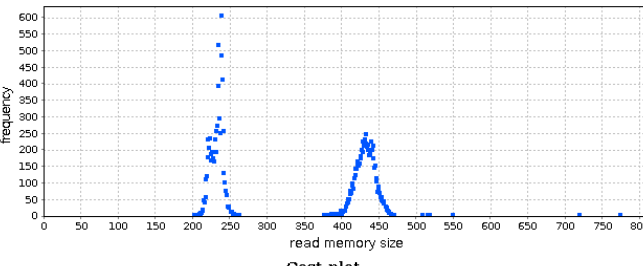
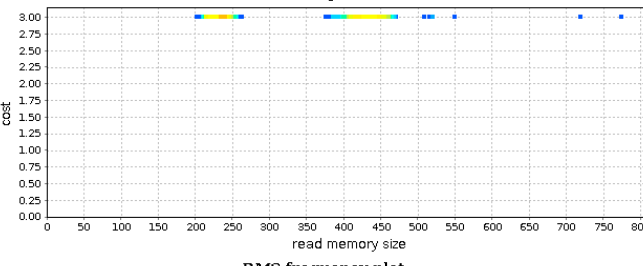
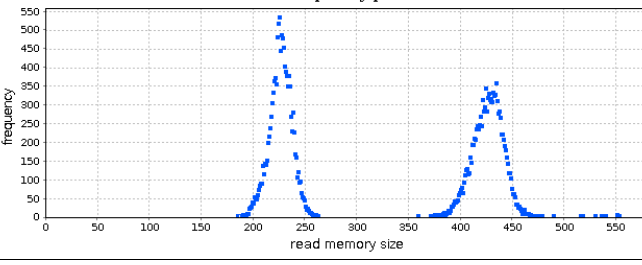


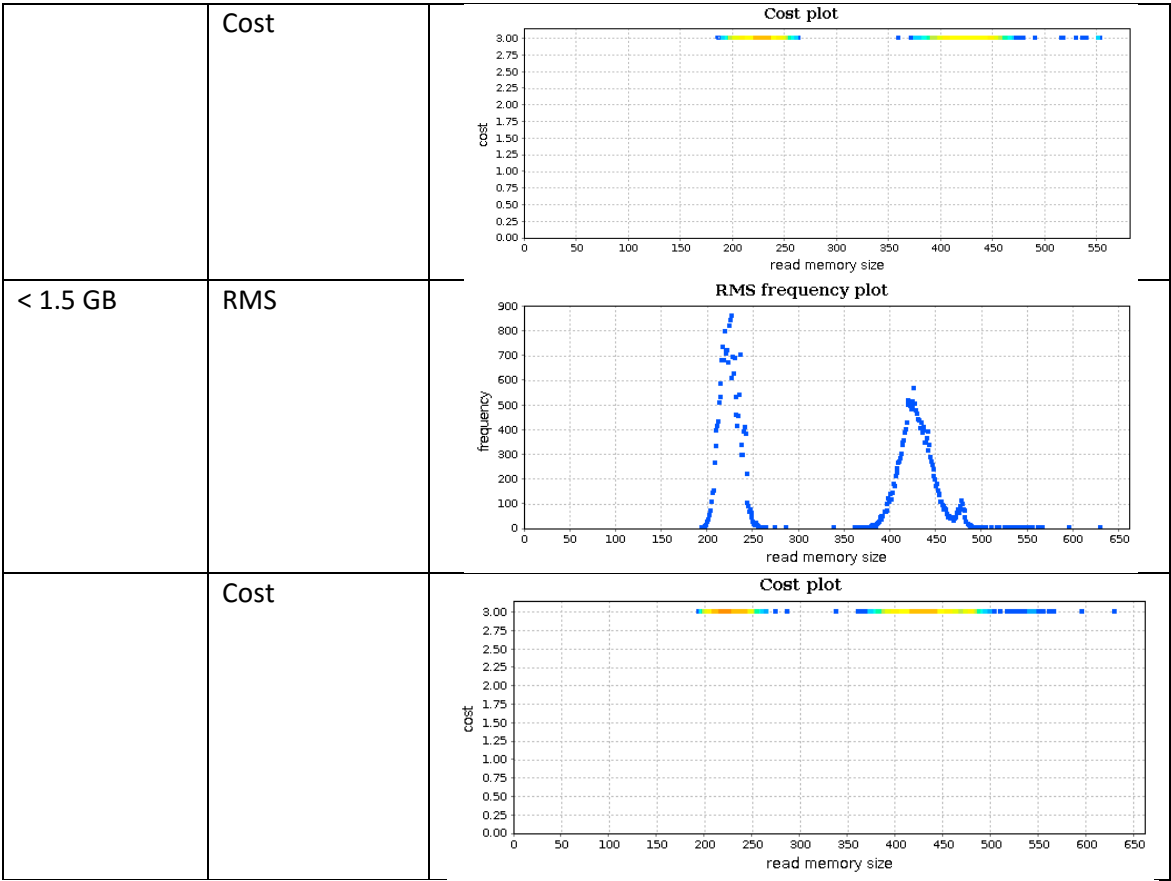
Number of calls



7. wxString CSizeFormat::Format(int64_t size, bool add_bytes_suffix /*=false*/)

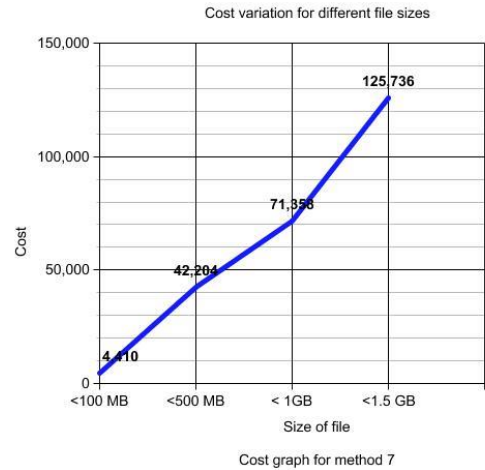
This method is responsible for the graphics of the progress bar which represents the number of bytes already transferred between the FTP server and the local. As this method does not involve any complex operations the RMS frequency plot and the cost plot remains consistent throughout different inputs.

Size of file	Metric	Plot
< 100 MB	RMS	
	Cost	
< 500 MB	RMS	
	Cost	
< 1 GB	RMS	

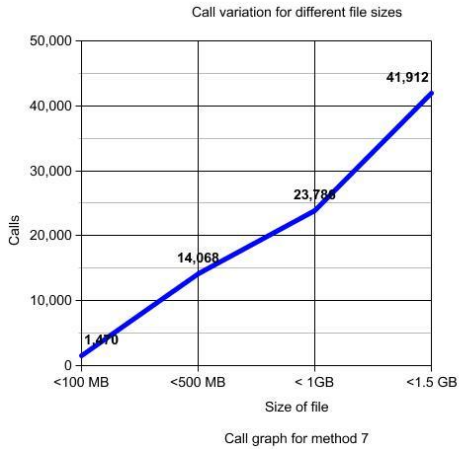


The below two plots show the Consolidated view of total costs and number of calls. We can see that the cost and number of calls grow linearly as the size of the file increases.

Consolidated Costs



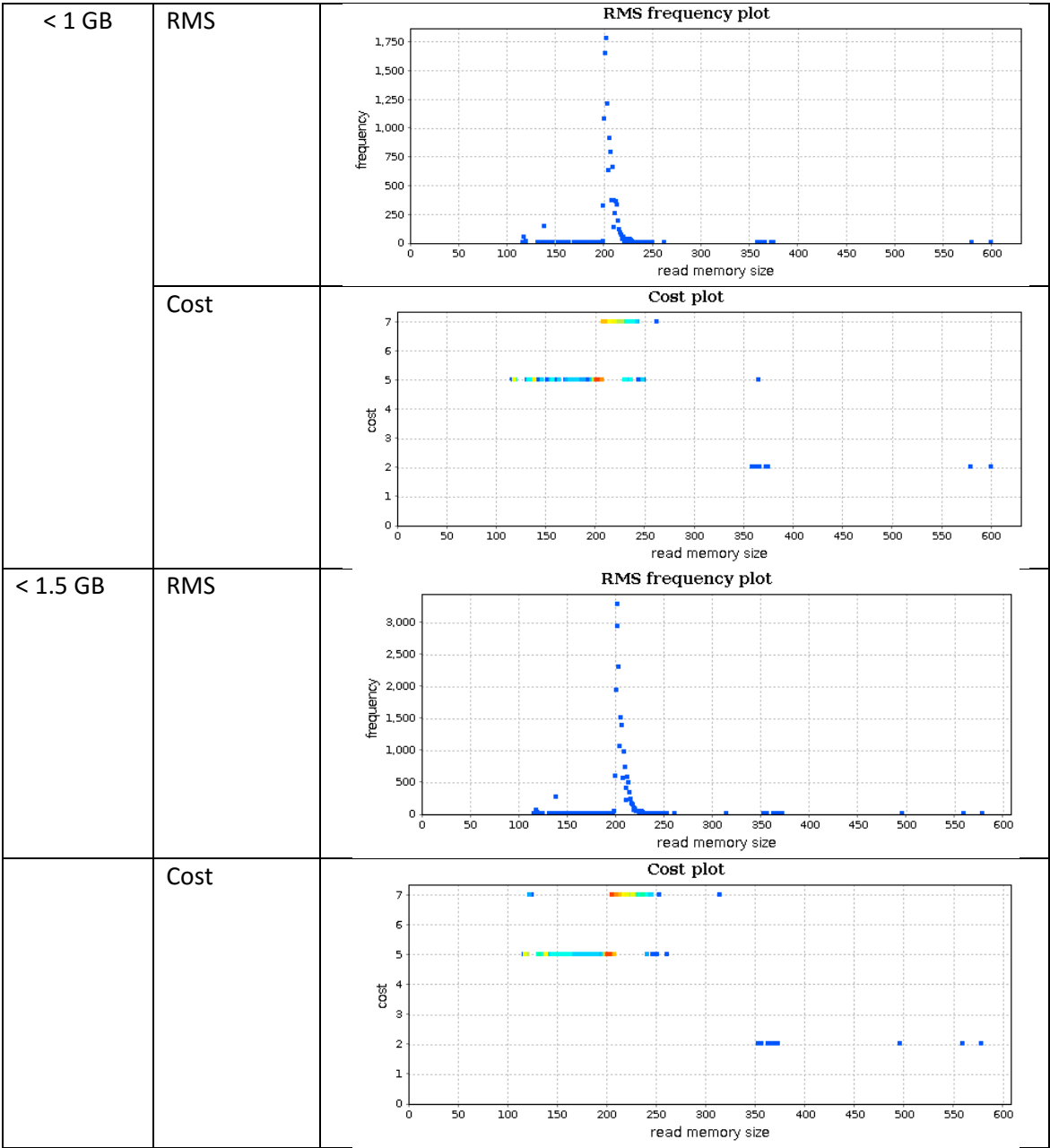
Number of Calls



8. void CStatusLineCtrl::SetTransferStatus(CTransferStatus const& status)

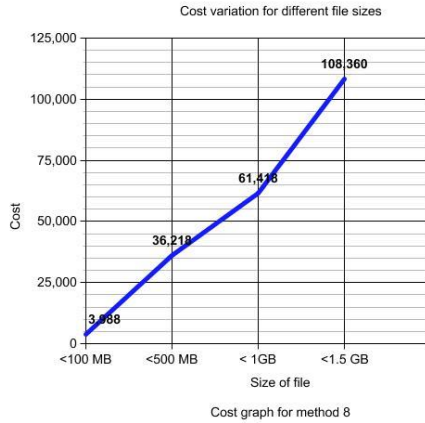
This method is for setting the different status of the transfer like transferring the data, disconnecting, cancelling, connecting. Every time there is a new connection or there is a change in the status of the connection this method is called.

Size of file	Metric	Plot
< 100 MB	RMS	
	Cost	
< 500 MB	RMS	
	Cost	

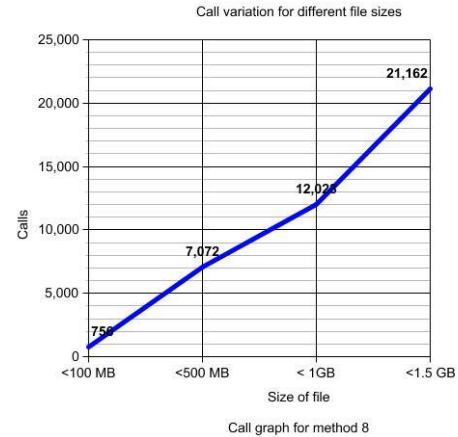


The below two plots show the Consolidated view of total costs and number of calls. We can see that the cost and number of calls grow linearly as the size of the file increases.

Consolidated Costs



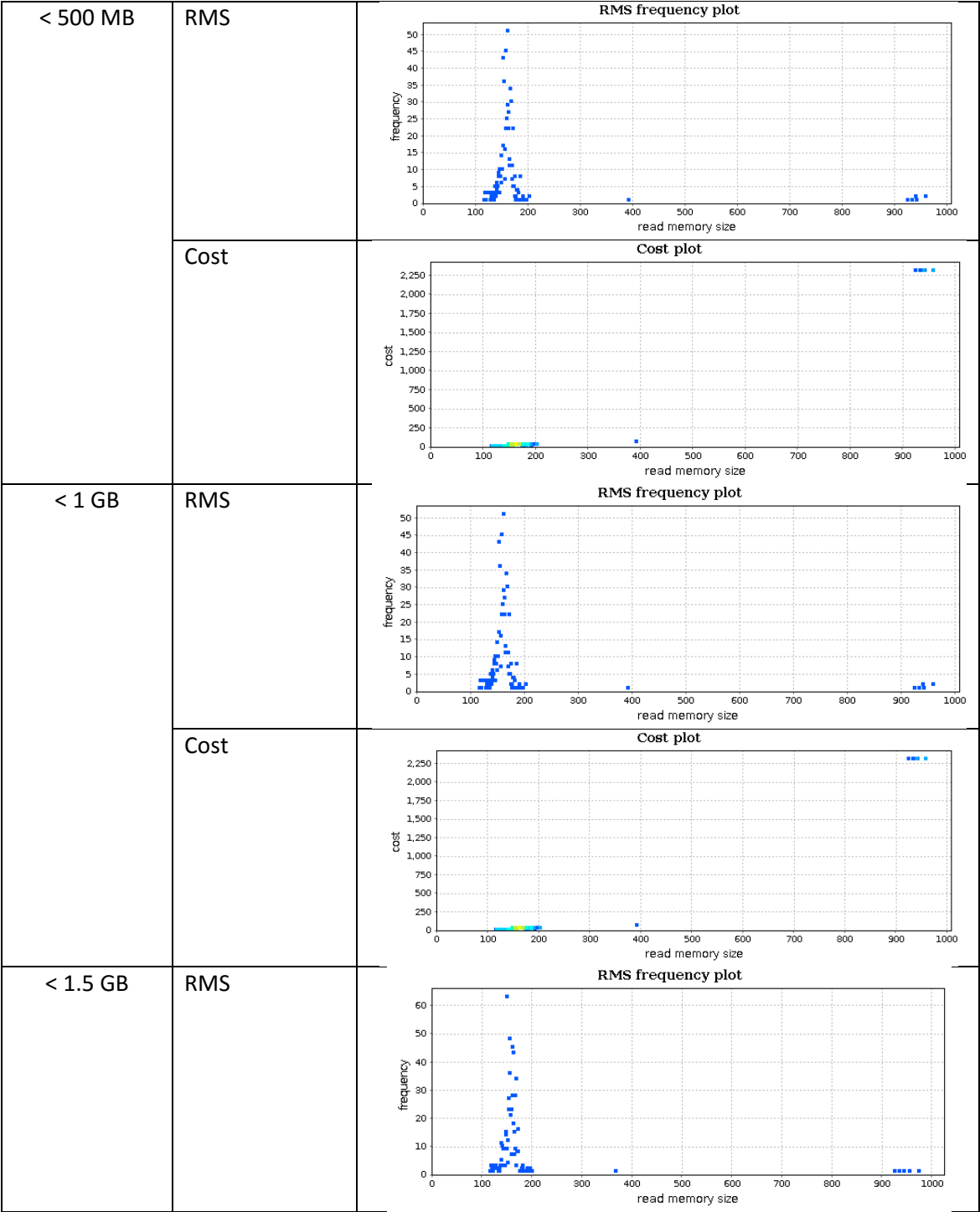
Number of Calls

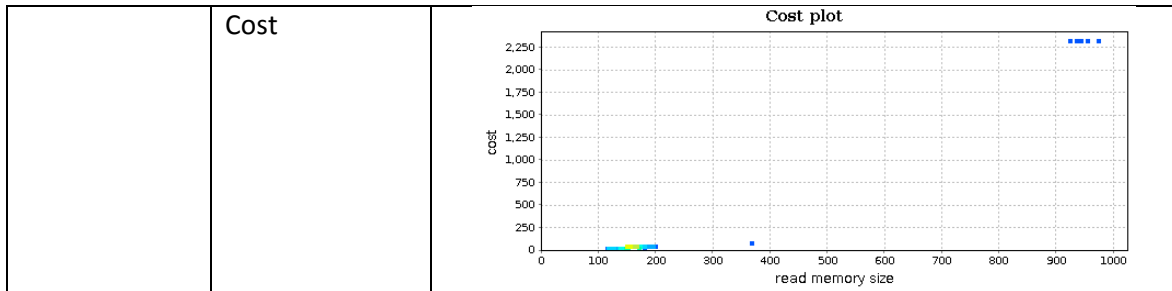


9. `bool CFileZillaApp::FileExists(const wxString& file) const`

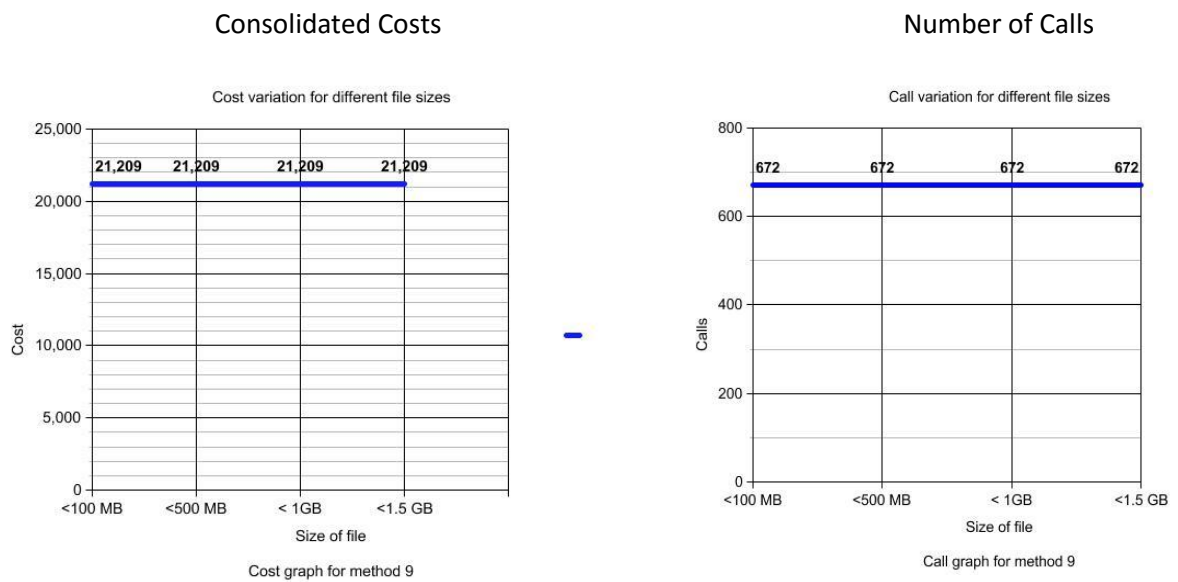
This method accepts full paths as parameter, with the addition that path segments may be omitted with a wildcard (*). If a matching directory is found it will return the path else it will return false.

Size of file	Metric	Plot
< 100 MB	RMS	<p>RMS frequency plot</p>
	Cost	<p>Cost plot</p>



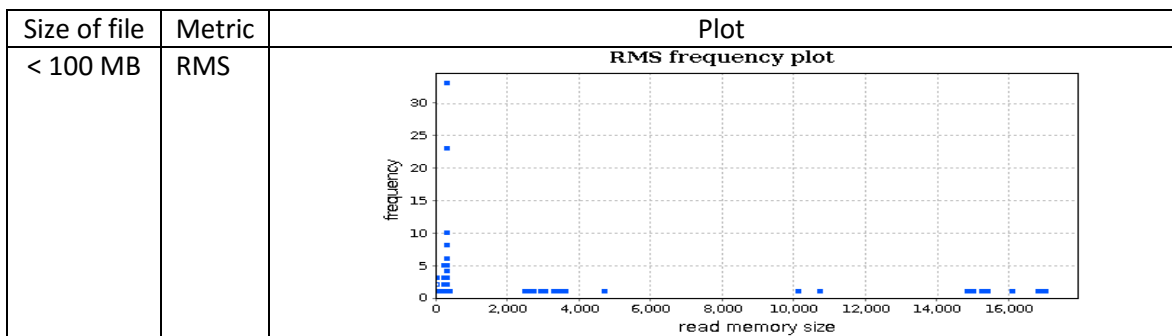


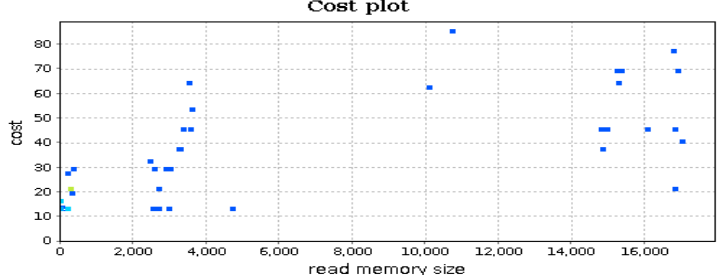
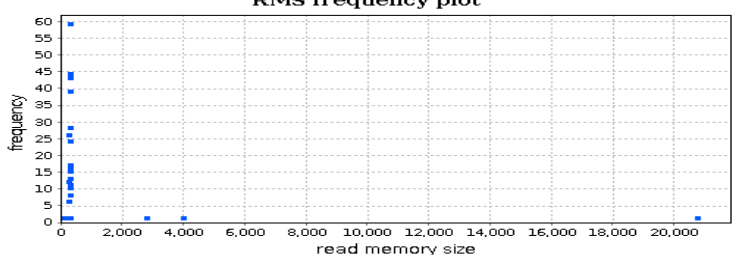
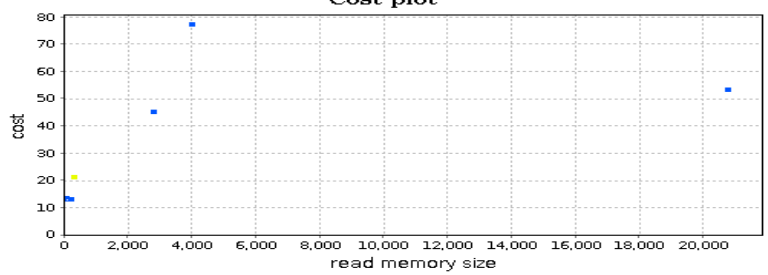
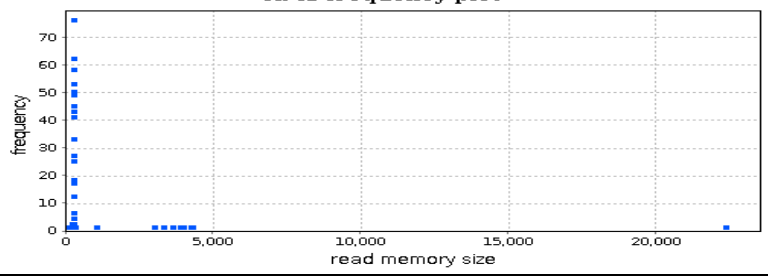
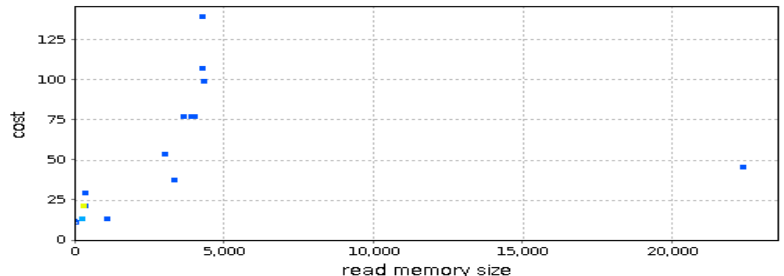
The below two plots show the Consolidated view of total costs and number of calls. We can see that the cost and number of calls are same for files of different files because checking for the path of the file will not in any way depend on the size of the file.

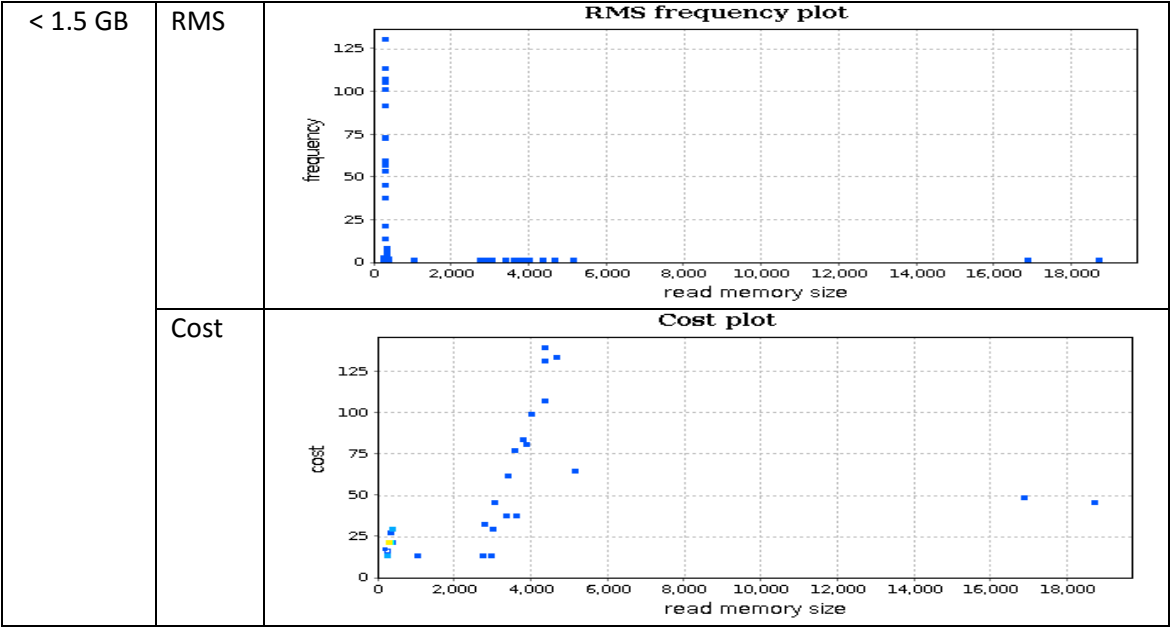


10. void CQueueView::OnEngineEvent(wxFzEvent &event)

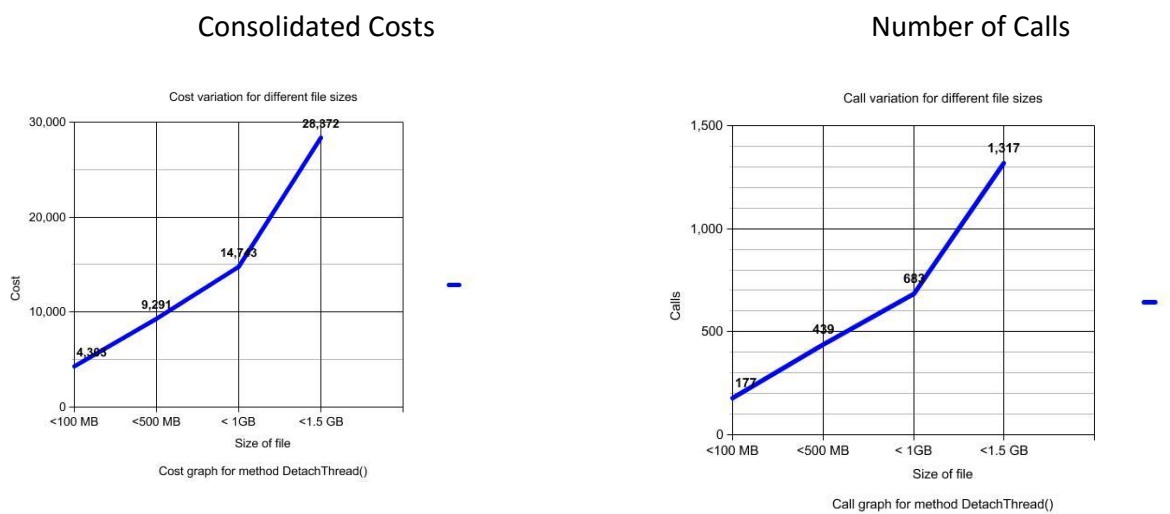
When multiple requests for file upload/downloads are triggered, all these requests are put in a queue. Whenever there is a change in the state of these queued requests, this method provides the notifications.



	Cost	 <p>Cost plot</p> <p>Y-axis: cost (0 to 80). X-axis: read memory size (0 to 18,000). The plot shows a scatter of blue data points with a general upward trend, reaching a cost of approximately 80 at a read memory size of 11,000.</p>
< 500 MB	RMS	 <p>RMS frequency plot</p> <p>Y-axis: frequency (0 to 60). X-axis: read memory size (0 to 20,000). The plot shows a dense cluster of blue data points at low read memory sizes (below 2,000) with frequencies up to 60. A few points are scattered at higher memory sizes.</p>
	Cost	 <p>Cost plot</p> <p>Y-axis: cost (0 to 80). X-axis: read memory size (0 to 20,000). The plot shows a sparse distribution of blue data points, with a notable peak in cost around a read memory size of 4,000.</p>
< 1 GB	RMS	 <p>RMS frequency plot</p> <p>Y-axis: frequency (0 to 70). X-axis: read memory size (0 to 20,000). The plot shows a very dense cluster of blue data points at low read memory sizes (below 2,000) with frequencies up to 70. A few points are scattered at higher memory sizes.</p>
	Cost	 <p>Cost plot</p> <p>Y-axis: cost (0 to 125). X-axis: read memory size (0 to 20,000). The plot shows a sparse distribution of blue data points, with a notable peak in cost around a read memory size of 4,000.</p>



The below two plots show the Consolidated view of total costs and number of calls. We can see that the cost and number of calls grow linearly as the size of the file increases.



6 Performance Metrics of Multithreaded Methods

FileZilla uses a Socket connection to connect to a FTP server to transfer files. It has its own IPv6 capable, non-blocking socket class for use with wxWidgets.

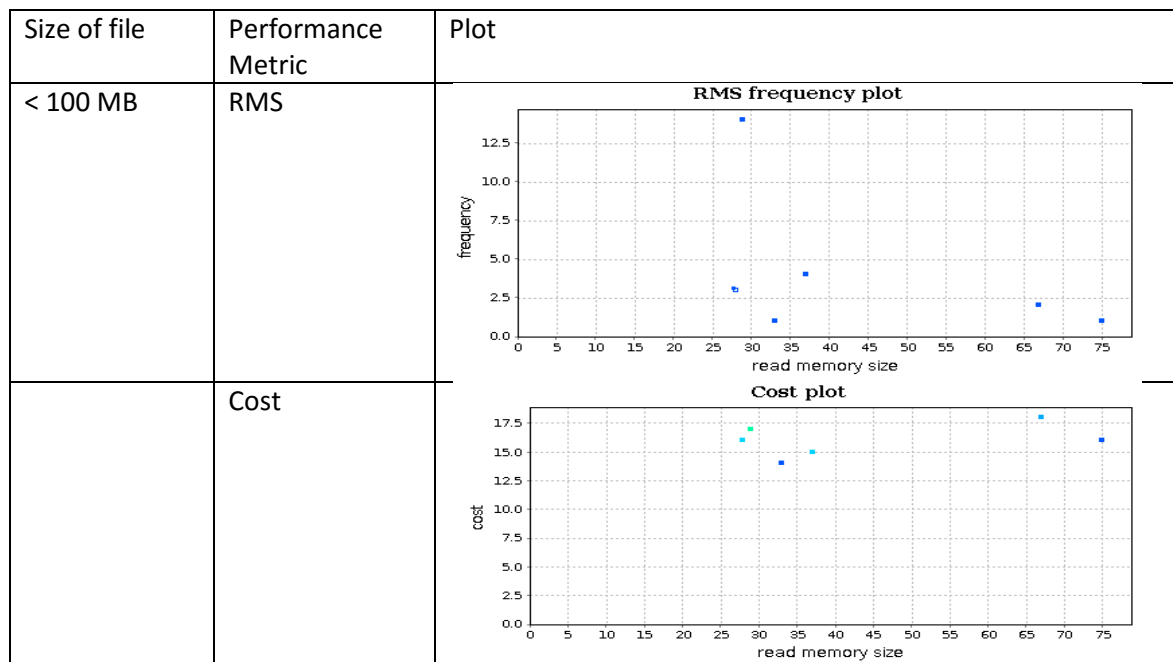
Each socket has the following states in its life cycle.

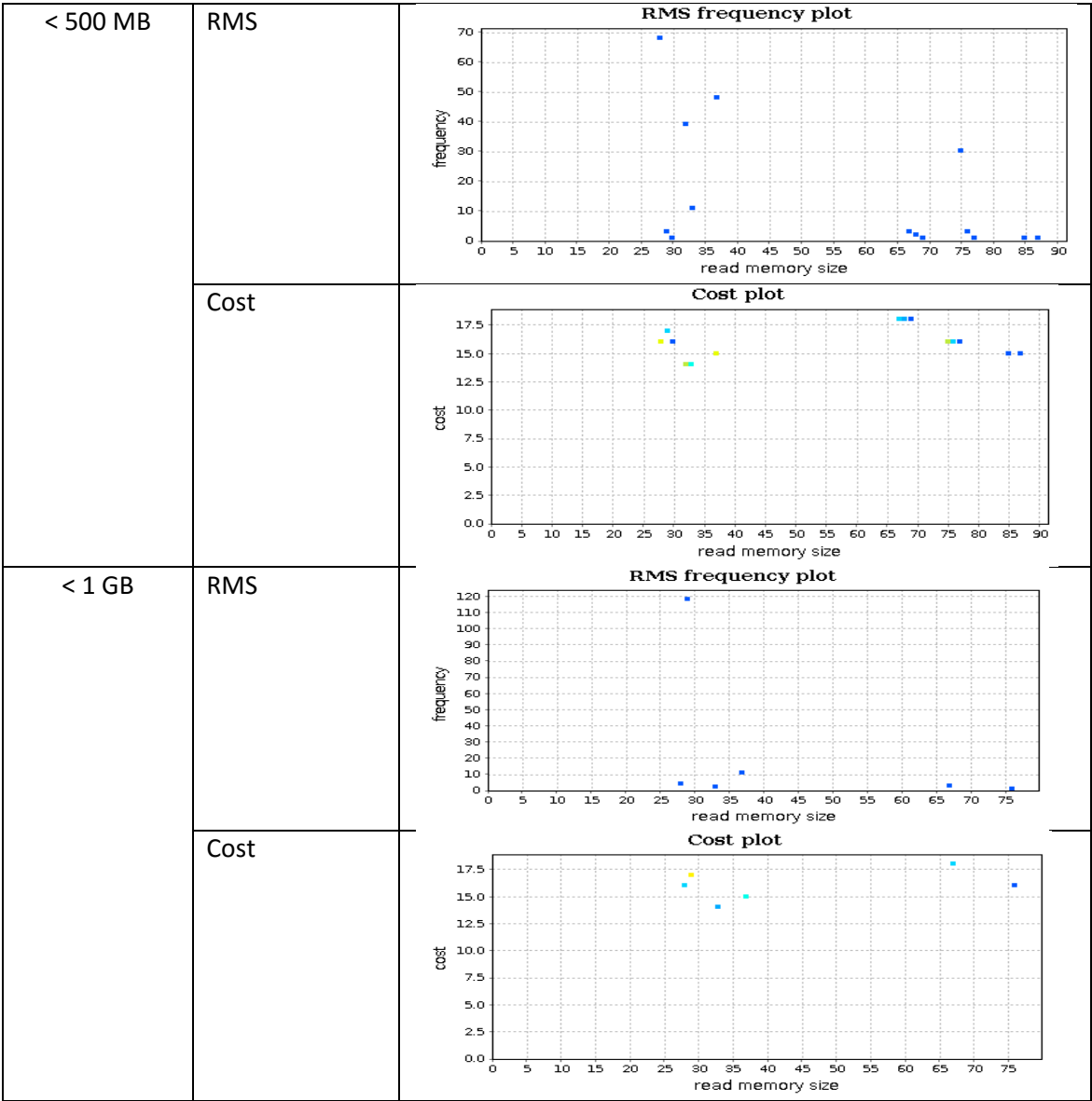
- None – Every socket will be in this state initially.
- Listening & Connecting – A socket can get a connection event only when it is in this state.
- Connected – After sending an event, a socket will be in connected state. It can only send or receive events only when it is in this state.
- Closing & Closed - Once a socket is shutdown gracefully, it will be in 'closing' and then in -'closed' state.

6.1 Multithreaded Methods

1. `int CSocket::Close()`

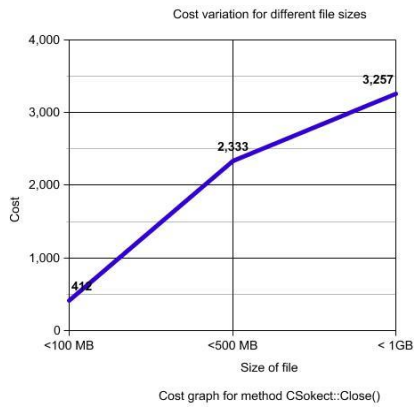
Socket class has a 'Connect' method which is used to connect to a given host, name, IPv4 or IPv6 address. This method returns 0 on success and an error code on failure. If the given host name can be resolved then a host address socket event is sent. Once connections got established, a close event gets sent. If a connection could not be established, a connection close event is sent.



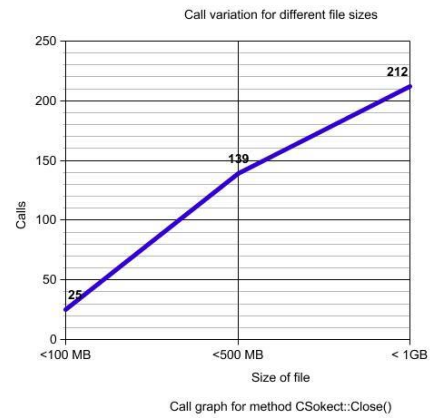


The consolidated cost and Call graphs for different sized files are shown below.

Consolidated Cost



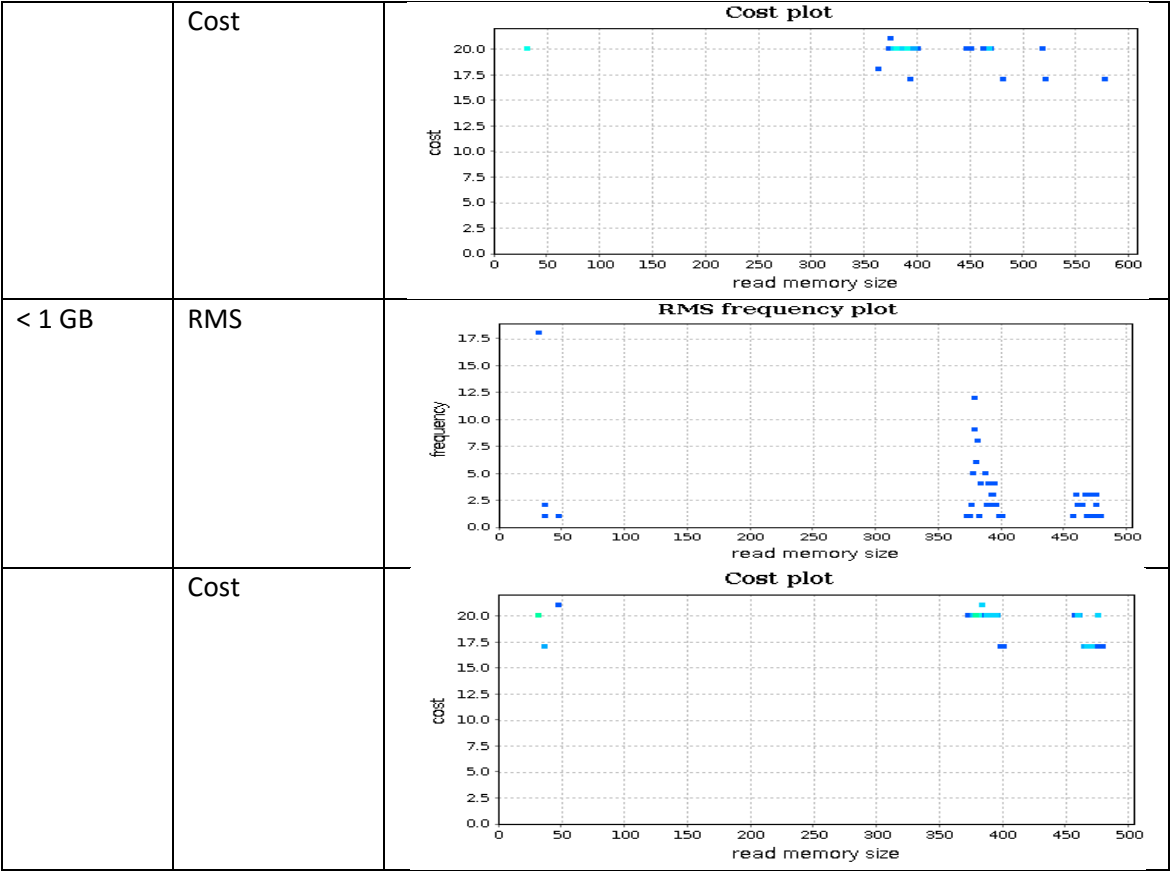
Number of Calls



2. void CSocket::DetachThread()

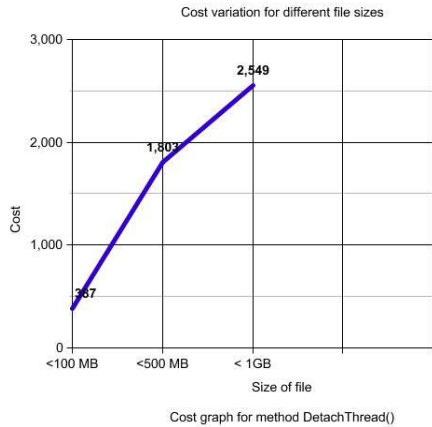
Once the transfers of files are complete, the socket connections are closed and the corresponding threads are detached.

Size of file	Metric	Plot
< 100 MB	RMS	<p>RMS frequency plot</p>
	Cost	<p>Cost plot</p>
< 500 MB	RMS	<p>RMS frequency plot</p>

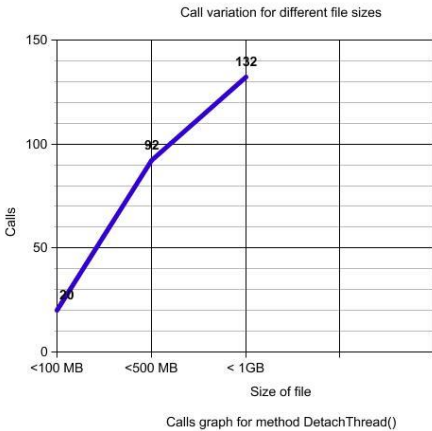


The consolidated cost and Call graphs for different sized files are increasing linearly as the size of the file increases.

Consolidated Cost



Number of Calls



3. `CInterProcessMutex::CInterProcessMutex(enum t_ipcMutexType mutexType, bool initialLock /*=true*/)`

Mutexes guarantee that only one thread can lock a given mutex. If a code section is surrounded by a mutex locking and unlocking, it's guaranteed that only a thread at a time executes that section of code. When that thread **unlocks** the mutex, other threads can enter to that code region.

Mutex has the following operations:

- **void lock()**

The calling thread tries to obtain ownership of the mutex, and if another thread has ownership of the mutex, it waits until it can obtain the ownership. If a thread takes ownership of the mutex the mutex must be unlocked by the same thread. If the mutex supports recursive locking, the mutex must be unlocked the same number of times it is locked.

- **bool try_lock()**

The calling thread tries to obtain ownership of the mutex, and if another thread has ownership of the mutex returns immediately. If the mutex supports recursive locking, the mutex must be unlocked the same number of times it is locked.

- **void unlock()**

Precondition: The thread must have exclusive ownership of the mutex.

Effects: The calling thread releases the exclusive ownership of the mutex. If the mutex supports recursive locking, the mutex must be unlocked the same number of times it is locked.

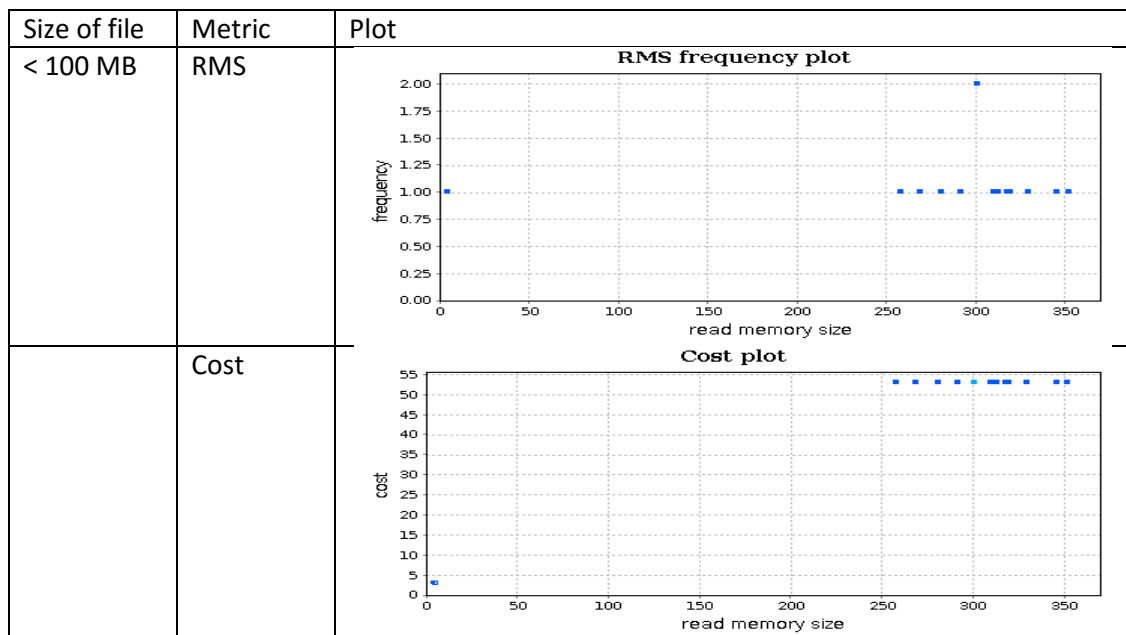
- **Scoped lock**

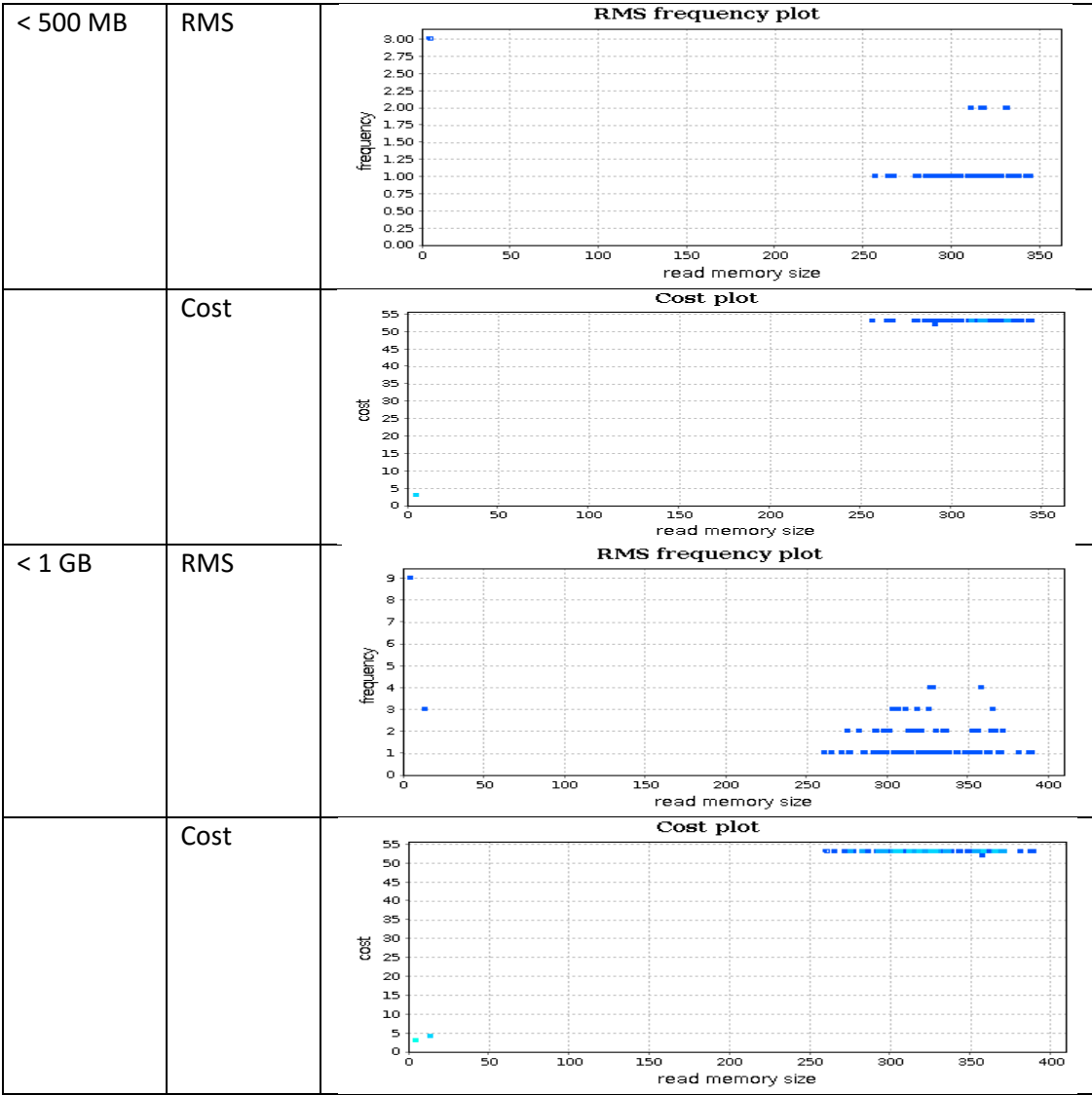
It's very important to unlock a mutex after the process has read or written the data. This can be difficult when dealing with exceptions, so usually mutexes are used with a scoped lock, a class that can guarantee that a mutex will always be unlocked even when an exception occurs. To use a scoped lock just include:

Basically, a scoped lock calls **unlock()** in its destructor, and a mutex is always unlocked when an exception occurs. Scoped lock has many constructors to lock, try_lock, timed_lock a mutex or not to lock it at all.

CInterProcessMutex represents an interprocess mutex. These are created and locked in the Constructor and released in the Destructor.

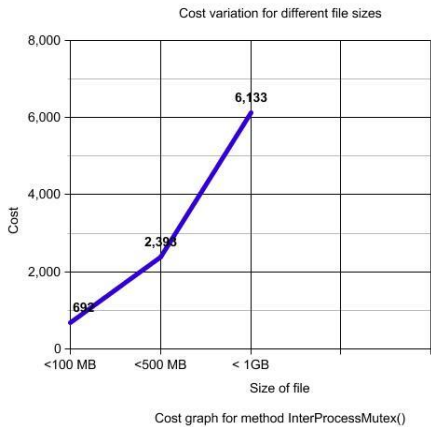
- This Mutex has all the Mutex operations like lock, unlock, try_lock and scoped_lock.
- Here, Mutex is used to lock a code section to restrict its execution by only one thread at a time.
- It also uses Scoped Locks to guarantee that a mutex will always be unlocked even when the exception occurs.
- Filezilla has different Mutex Types to handle synchronization at different parts of the project. The different mutex types used in FileZilla are
 - MUTEX_OPTIONS = 1
 - MUTEX_SITEMANAGER = 2,
 - MUTEX_SITEMANAGERGLOBAL = 3,
 - MUTEX_QUEUE = 4,
 - MUTEX_FILTERS = 5,
 - MUTEX_LAYOUT = 6,
 - MUTEX_MOSTRECENTSERVERS = 7,
 - MUTEX_TRUSTEDCERTS = 8,
 - MUTEX_GLOBALBOOKMARKS = 9,
 - MUTEX_SEARCHCONDITIONS = 10,
 - MUTEX_LASTFREE = 11
- These mutex types are used to prevent simultaneous updates to the FileZilla configuration files like filezilla.xml, recentserver.xml.
- Also for performing operations on Sitemanager like adding/removing/rename a FTP site.
- The local FTP folder can be bookmarked using the bookmarks menu in the toolbar. The
- Globalbookmarks mutex type is used to handle synchronization of updates to the Bookmarks config file



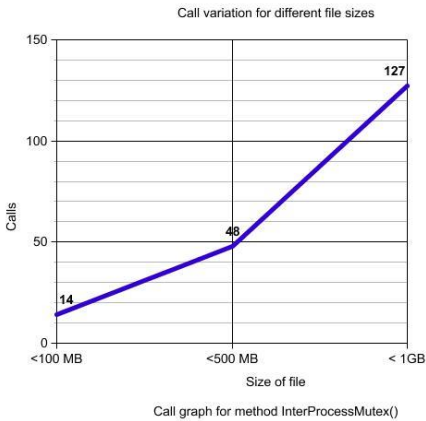


The consolidated cost and Call graphs for different sized files are,

Consolidated Cost



Number of Calls



When filezilla is connected to more than one server and had active transfers in progress from all of them, the .aprof generated involved multithreaded methods. When more connections are established, more number of sockets and threads are created which involves more overhead when compared to normal transfers. The performance metrics for the various multi threaded methods varied on the size of the file being transferred. As the size of the file increases the cost and other performance metrics show an increase.

7 Challenges Faced

- Not able to build Filezilla on Windows.
- Gnutls library installation error on Ubuntu 14.04 version.
- Most of the routines displayed in aprof plot belonged to a 3rd party software component and Linux libraries
- Not able to find source code of routines which belonged to Filezilla displayed in aprof plot.
 - i. `sudo apt-get install ctags`
 - ii. Steps taken to resolve : **Ctags** is a tool that makes it easy to navigate large source code projects. It provides some of the features that you may be used to using in Eclipse or other IDEs, such as the ability to jump from the current source file to definitions of functions and structures in other files.
- Set the path of ctags to Filezilla binary
- Since Filezilla a huge application, analyzing the flow was difficult and consumed most of the time.

8 References

<https://filezilla-project.org/>
[https://wiki.filezilla-project.org/Compiling FileZilla 3 and Getting Dependencies on Linux](https://wiki.filezilla-project.org/Compiling_FileZilla_3_and_Getting_Dependencies_on_Linux)
<https://github.com/ercoppa/aprof/wiki/aprof-plot-manual>
<http://wwwusers.di.uniroma1.it/~finocchi/papers/pldi055-coppa.pdf>
<http://ctags.sourceforge.net/>