A Mechanism for New Smart FTP Transfer (NSFTP)

Kazem HASSAN*
* Department of Computer Engineering, Faculty of Engineering
Ittihad University
Ras Al Khimah, RO.Box 2286 United Arab Emirates
(E-mail: hassanishome@gmail.com)

ABSTRACT

All of current technologies that provide File Transfer Protocol re upload whole updated and edited file(s) even if the file changes by a minimum of one byte. This is not sufficient in term of upload time, bandwidth usage and logic. While this can be solved and proposed a solution to the stated scenario. By enabling the FTP client and FTP server to treat only the differences between local files and remote files. Thus for this will amazingly reduce FTP transfer time and speed up the process.

KEY WORDS: FTP, FTP Traffic, Bandwidth NSFTP

INTRODUCTION

The current FTP Clients and FTP servers process the updated files by uploading the whole content even though if the changes are a minimum of one byte. For example, a web developer might neglect a seminoma in his PHP file and thus for the whole PHP File application will not be interpreted and will through and error.

The web developer will fix the error then command the FTP client to re-upload the content while it’s only a minor change. The challenge is to reduce the FTP upload time and bandwidth in a smart mechanism.

CURRENT FTP METHOD

A) Client Side
   1. Update new asci/binary file.
   2. Upload whole file by FTP client.

B) Server Side
   1. Receive file upload request.
   2. Create new stream as New file (overwrite).
   3. End session.

NEW SMART FTP METHOD (NSFTP)

A) Client Side
   1. Update new asci/binary file.
   2. Calculate the difference between new update and previous cached file.
   3. Upload the differences only (an indexed objects).
   4. Cache the latest uploaded file.

B) Server Side
   1. Receive file upload request (an indexed objects).
   2. Update the current file with the received amendments only (sync)
   3. Keep agreed history for case of undo/redo.

4. End session.

Example

A user wants to update one byte in a PHP file, let’s say the user forgot to add seminoma (;) somewhere. The updated file size is 100KB, while the amendment is only 1 byte. In NSFTP the actual content stream is only 34 bytes.

In case of ASCI files, the actual content is similar to: [{"line-number":10, "column-start":30, "column-end":31,"content":';'}] JSON format. Or simply: [{"L":10, "S":30, "E":31,"C":';'}] which is 34 bytes only instead of 100KB, neglecting the other transaction size; such as the command and its parameters.

IMPROVMENTS:

We can try to compress the content before upload and once the content landed in the server side; the server will use the uncompressed to save further bandwidth.