CFT: Co-operative File Transfer Algorithm for Multi Network Interface Sessions

Anees Fathima S, Sushma K, Maboobi, Vishwas Narayan, Abhishek Alfred Singh,
Kiran K, P Deepa Shenoy, Venugopal K R
Department of Computer Science and Engineering
University Visvesvaraya College of Engineering
Bangalore,India

***Abstract*—File transfer is one of the important operations on the Internet. Generally files are transferred from one machine to another machine through one interface. File transfer can occur through multiple interface connections also. Protocols such as SCTP, transfers data in multiple data stream within a single connection and LFTP transfers file sourced from multiple servers to a single host. Here, we present the concept of using multiple network interfaces for transferring files from a single server. This would ensure the utilization of combined bandwidth of all the interfaces used, so that the rate of file transfer would increase considerably compared to single bandwidth transfer. In this work, we use two interfaces i.e, IEEE 802.3(Ethernet) and IEEE 802.11(WiFi) to accomplish the above task. We use a non pre-emptive context switching framework *Twisted* where threading is avoided for an effective resource utilization. The required file is downloaded utilizing two interfaces instead of one unlike normal file transfer. We analyze the improvement in performance by observing the time taken to download a file using two different interfaces (Ethernet and WiFi) and comparing that with a single interface download (using either Ethernet or WiFi) in real time scenario. We attempt to deal with the issue of when and how to connect through two interfaces which combines the bandwidths of both these interfaces, aiding in improving the performance of file transfer when compared to file transfer using single interface.**

***Index Terms*—Client-Server model, Ethernet, LFTP, Network Interface, SCTP, Twisted, WiFi**

I. INTRODUCTION

Considering the amount of files that are downloaded over the internet, there is a need to improve the file transfer algorithm to save time. Generally, we transfer file through a single interface. It is common for systems to have more than one network interface present by default. We can utilize this feature of systems to transfer large files with maximum speed. Through a single interface, only one bandwidth is used, but in multiple-interface connection we utilize the bandwidths of all the interfaces present, to transmit the data.

In any machine, we can configure multiple wireless cards, ethernet cards or any other network devices for communication. In a host, Ethernet interfaces are identified as *eth0*, *eth1* and so on whereas wireless interfaces are identified as *wlan0*, *wlan1* and so on for network connections.

Our work describes a co-operative Client, Server communication for file transfer. The main idea is to be able to create single session over multiple interfaces much like SCTP protocol. We provide a much higher level of independence over the connections in terms of creation, management and termination of these sessions. In our work, we attempt to increase the time efficiency by transmitting files through two interfaces, WiFi and Ethernet. We define a *THRESHOLD* value, based on the file size, to decide upon whether to use single interface or both these interfaces. The connection is set-up using port numbers and IP addresses(Socket). Our work makes use of IPv4 representation for identifying the Client and the Server. If the requested file size is greater than the *THRESHOLD*, then the Server signals the Client to connect through another interface. Once the file is sent successfully, the Server sends a disconnect request to the Client to release all its connections with it.

A graph is plotted by testing the working of our algorithm in real time, using WiFi and Ethernet. The comparison between the time taken by these two interface connections with that of the single interface connection to send a file, showed an increase in time efficiency of our multiple-interface algorithm when compared to that of single-interface one.

Section II deals with other works done earlier based on the concept of more than one interface. In Section III, we brief about the problem while giving a solution with respect to creating multiple interface connections. Section IV gives the view of design used in this work. In Section V, we discuss how have we implemented the design using the concept of twisted python, and also we explain the algorithm of the Server and Client for both single and multiple connections. In section VI, we present the performance analysis through graphs. Future enhancement forms a part of conclusion in section VII.

II. RELATED WORK

Many researchers have already discussed about how they use multiple interfaces like WiFi, WiMAX, IP Multimedia Subsystem(IMS)-based network, Universal Mobile Telecommunications System(UMTS) network etc [1], [2], [3], [4], [5] for sophisticated file transfer. In the work by K.Kiran et al, the file/data has been split and sent through WiFi and WiMAX interfaces considering a split co-efficient for splitting. These works act as a backbone, by giving us the statistics of improved performance using two radios. But in our work

978-1-4673-6540-6/15/$31.00

involving multiple interfaces, the Client decides creation of new connections with the Server depending upon number of interfaces that Client can provide for establishing connections. K.R.Venugopal et al have demonstrated how cost can be reduced by wavelength converters in WDM wavelength routed all-optical networks[6], but in our work, we were able to reduce the time of file transfer through bandwidth division.

SCTP(Stream Control Transmission Protocol) sends data in multiple data streams simultaneously within a connection. We use this idea to send the data but through different interfaces[7]. LFTP(command-Line File Transfer Program), also has an option to have segmented file transfer(where the file is sourced from multiple servers to a single destination), allowing more than one connection for the same file, which results in maximum download speed for a file[8]. Ethernet is faster when compared to other wireless networks. This analysis is done in [9], [10]. A work on comparison of wired and wireless LANs[11], says ethernet is very sensitive to the number of users and the load offered. Hence we can avoid the overload on any interface alone, by transfering data along multiple interfaces.

[12] presents the design and implementation of all-IP heterogeneous network where the services provided are public and private wireless broadband access, based on fixed WiMAX, WiFi and High Speed Packet Access(HSPA) .In [13], [14], the authors have discussed about routing algorithms and their challenges.

III. PROBLEM STATEMENT

In our work, we are trying to show the improvement in the performance of our multi-interface protocol algorithm by transferring files through two separate interface connections, WiFi and Ethernet. We have used the same protocol for checking the speed of file transfer over single interface. $Twisted$ provides us the framework(platform) for creating this protocol for file transfer and helps in accomplishing different connections asynchronously.

IV. DESIGN

We have used simple Client-Server model wherein the Server on one side keeps listening to a particular port and IP forming a socket. Whenever the Client sends a connection request to that socket bound to the Server, connection is established between the two processes. The key features that we focus on to carry our work are as follows:

- The connection is established by the Server sharing an $unique$ id with the Client for future identification of the Client instance.
- This Client instance stores and exercises this particular id for future communication with its respective Server handler.
- Request for a file, sent from a Client instance is brilliantly handled by the Server host by checking for the seeked file size. If this file size is large, i.e greater than the *THRESHOLD*(compared to the pre-defined file size) then Server sends an interrupt to the Client machine to fork at another IP making another connection with it.
- Depending on the number of interfaces present for establishing connection, the Client gets interrupted from the Server machine, so that file could be split and sent in every possible interface to the Client.

Concurrent to this process, the Server divides the requested file(if found) into significant number of blocks(maximum data that can be sent through each of the interfaces) and sends it simultaneously through all the established connections. Alongside the Client acknowledges the Server for each block received from each connection, making room for the further blocks to be sent. This continues until the connection is healthy or until the entire file is sent. This is depicted by the flow diagram in Fig. 1.

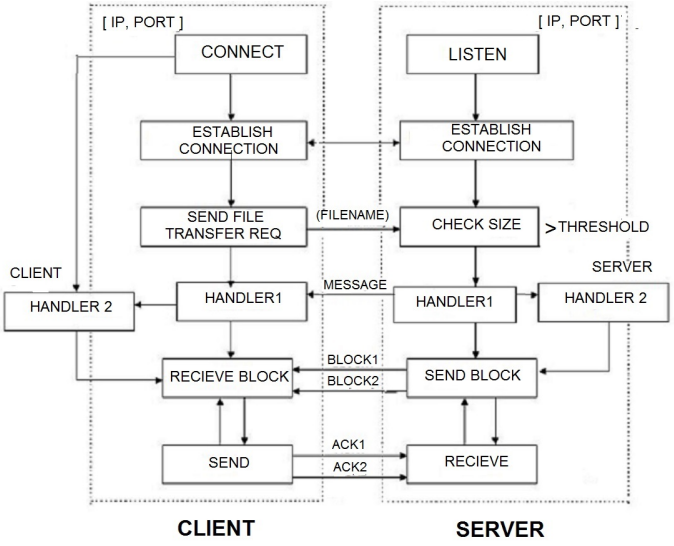

Fig. 1: *Block Diagram for Double Interface Connection.*

V. IMPLEMENTATION

$Twistedpython$ is an important framework on which our work is implemented. It can handle thousands of connection requests in a single thread by its event-driven networking ability. Creating multiple threads for every incoming request, generates an overhead of protecting shared resources being allocated concurrently to each of the multi-thread. But $Twisted$ avoids this overhead by asynchronously handling events in a single thread.

Here, we have made use of $protocol$ and $reactor$ modules from the $internet$ module of $twisted$. The $Protocol$ class of $protocol$ module in turn bestows methods for establishing connections, for sophisticated data transfer and for losing connection between Server and Client. $ClientFactory$ class of $protocol$ module has methods for the Client to lose connection and to know in case there is a connection failure. $Factory$ class of $protocol$ module has a method called $buildProtocol$ which helps in creating an instance of any subclass of $Protocol$ class[15], [16].

ALGORITHM-I and II illustrate file transfer in a Multiple interface mode. Here, availability of an alternate interface and the size of the requested file decides establishment of

another connection through that interface. Only on receiving acknowledgement for each block of data transmitted, further transfers are determined.

ALGORITHM I: *Client Module for Multiple connections*

Send a $connectionRequest$ for a Server's socket;
IF $acceptRequest$ received ($uniqueId$ received)
    request for a file is sent;
ENDIF
IF request for a new connection received
    IF another interface available
        connection is made with the same $uniqueId$;
    ENDIF
ENDIF
receive($dataBlock$);
send($acknowledgementBlock$);
After the file being received, $disconnect$;

ALGORITHM II: *Server Module for Multiple connections*

Listen to a specific port for making connection;
IF connection request received
    For $newConnection$, an $uniqueId$ is generated and sent to the respective peer;
    On receiving a request for a file, the request is validated and checked for the file existence;
    Then $filesize$ is checked;
    IF $filesize >$*THRESHOLD*
        request another interface connection;
        send($dataBlock$);
    ELSE
        send($dataBlock$);
    ENDIF
    IF receive($acknowledgementBlock$);
        send($dataBlock$);
    ELSE
        wait;
    ENDIF
    After the file being sent, $disconnect$;
ENDIF

ALGORITHM-III and IV illustrate file transfer in a single interface mode. Irrespective of the number of interfaces present and the requested file size, the same connection is employed for both sending and acknowledging data.

At the Server, the $listenTCP()$ method of reactor class, blocks the Server by making it listen to a particular port until a new connection is made as briefed in the ALGORITHM -II and ALGORITHM-IV. On establishing new connection, $buildProtocol$ method instantiates the Server to handle that connection on a particular port and IP(on a particular socket). The method $connectTCP()$ of the $reactor$ class in the Client, sends a connection request to a particular IP and port. In the Client, $buildProtocol$ method in-turn instantiates the Client to handle this new connection with the Server.

ALGORITHM III: *Client Module for single connection*

Send a $connectionRequest$ for a Server's socket;
IF $acceptRequest$ received ($uniqueId$ received)
    request for a file is sent;
ENDIF
receive($dataBlock$);
send($acknowledgementBlock$);
After the file being received, $disconnect$;

ALGORITHM IV: *Server Module for single connection*

Listens to a specific port for making connection.
IF connection request received
    For $newConnection$, an $uniqueId$ is generated and sent to the respective peer;
    On receiving a request for a file, the request is validated and checked for the file existence;
    send($dataBlock$);
    IF receive($acknowledgementBlock$);
        send($dataBlock$);
    ELSE
        wait;
    ENDIF
    After the file being sent, $disconnect$;
ENDIF

When the file size of the requested file is greater than the THRESHOLD, Client would be signaled by the Server to make new connection through another interface. Here, we have considered the THRESHOLD value as 6MB, considering the bandwidth of 54Mbps for Wireless 802.11a/g in normal routers. Whenever the Client tries new connection with the Server through another interface(IP address), to facilitate faster download, one of the Client handler calls the $connectTCP()$ method. By then another Client instance gets created to handle this new connection with the Server. At the Server side, $listenTCP()$ method detects this new connection and registers a callback with the $buildProtocol$ method which in turn is responsible for instantiating new Server instance bound to a socket, to handle another connection with the Server. File transfer is carried out with these two separate channels wherein the bandwidth of both the channels are utilized. The state of the protocol is stored in the $Factory$ class of $protocol$ module.

In case of single connection file transfer, only one instance of the Server and Client are created. This single channel is solely responsible for transferring and acknowledging the data blocks sent.

VI. PERFORMANCE ANALYSIS

The implementation is done in real time, by making the Server and Client connection using Ethernet(Wired LAN) and WiFi(Wireless LAN) interfaces for double connection code. Ethernet being fastest interface, it is used for establishing Server-Client connection for implementing single connection. The values recorded for running these two codes are separately tabulated for two trials Run-1 and Run-2 in tables I and II respectively.

TABLE I: *Double Connection File Transfer*

File size(MB)	Run-1 Real time (s)	Run-2 Real time(s)
9	14.7	16
25	39.9	46.16
50	82.4	83.87
75	119.2	127.9
100	162.1	177.8
226	361	396.6

TABLE II: *Single Connection File Transfer*

File size(MB)	Run-1 Real time(s)	Run-2 Real time(s)
9	17.9	17.2
25	49.05	49.5
50	98.7	92
75	147.1	148
100	197	195.8
226	442	475.4

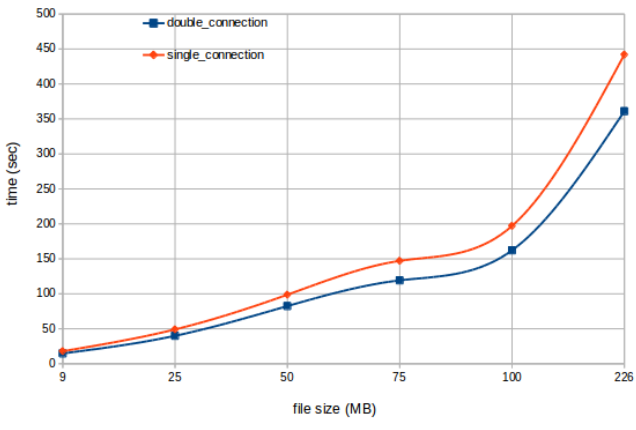

Fig. 2: *File size v/s Time of File Transfer for Run-1.*

User time is the CPU burst time for the user process whereas the system time is the time that the process spends inside the kernel (for calling system functions within kernel). Real time is the actual time taken for completing the entire process which is the time between when the process was invoked and when the process was terminated. Real time includes both user time and system time.

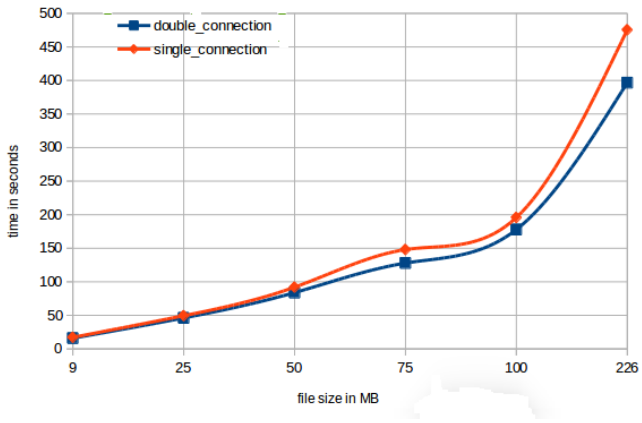

Fig. 3: *File size v/s Time of File Transfer for Run-2.*

From Fig. 2 and Fig. 3, it is observed that the real time taken by our algorithm for transferring files is always greater with two interfaces(double connections) when compared to single connection file transfer irrespective of the file size being sent. We could also observe that our algorithm for double connection has maintained a constant rate of 0.619 Mbps and that of single connection has maintained a rate of 0.507Mbps for transfering files of various sizes when the bandwidth of Ethernet was 100Mbps and that of WiFi had a range between 1Mbps-50Mbps. We could observe that performance increased by 18% for double connection file transfer over single connection one.

VII. CONCLUSION

In this work, we have discussed an algorithm for transferring files using combined bandwidth of multiple interfaces while showing how this would work upon when implemented on commonly used interfaces like Ethernet and WiFi together. This showed us a positive response by taking relatively less time when compared to that of the algorithm implemented using single connection (using Ethernet alone).

Likewise, we can also try to implement this multiple-interface algorithm using more than two interfaces or interfaces otherthan WiFi and Ethernet to find out which of these scenarios best suit for the implementation.

SCTP supports multihoming which means that a connected end point can have alternate IP address so that the data can route around if there is a network failure. So, even in our work if there is a problem in sending the data through one interface, we can send the data through other interface, thus utilising the bandwidth available[17]. Handling dfferent types of Servers like Mail Server, Application Server, Real-Time Communication Server and various others is a challenging task which need to be accomplished.

As a future work, we can also make use of bandwidths of different Ethernet cables to determine the highest bandwidth that can be achieved using these interface cables. By tracking the amount of data transferred through WiFi and Ethernet, we could decide upon dividing the chunk of data that has to be sent through each of these interfaces. The file download and upload could be prioritized as shown in the work [18], wherein we can try to use appropriate proportion of the available bandwidth without reducing the transmission rate.

REFERENCES

[1] K. Kiran, A. A. Singh, P. D. Shenoy, K. Venugopal, and L. M. Patnaik, "Analysis of traffic splitting over a multi-hop network with hybrid wimax and wifi nodes," in *Parallel Distributed and Grid Computing (PDGC), 2012 2nd IEEE International Conference on*. IEEE, 2012, pp. 609–613.

[2] K. Kiran, A. A. Singh, S. Yadunandan, P. D. Shenoy, K. Venugopal, and L. M. Patnaik, "Throughput enhancement by traffic splitting over an ad-hoc network with hybrid radio devices," in *TENCON Spring Conference, 2013 IEEE*. IEEE, 2013, pp. 371–375.

[3] K. Kiran, T. Shivapriya, A. A. Singh, P. D. Shenoy, K. Venugopal, and L. M. Patnaik, "Traffic splitting in a mobile ad-hoc multi-radio network," in *India Conference (INDICON), 2013 Annual IEEE*. IEEE, 2013, pp. 1–4.

[4] N. Psimogiannos, A. Sgora, and D. D. Vergados, "An ims-based network architecture for wimax-umts and wimax-wlan interworking," *Computer Communications*, vol. 34, no. 9, pp. 1077–1099, 2011.

[5] L. Ma and L. Rui, "An end-to-end qos frame in multimedia provision for tight-coupled interworking of ims and wimax," in *Networking, Sensing and Control, 2008. ICNSC 2008. IEEE International Conference on*. IEEE, 2008, pp. 1153–1157.

[6] K. R. Venugopal, E. E. Rajan, and P. S. Kumar, "Performance analysis of wavelength converters in wdm wavelength routed optical networks," in *High Performance Computing, 1998. HIPC'98. 5th International Conference On*. IEEE, 1998, pp. 239–246.

[7] P. Stalvig, "Introduction to the stream control transmission protocol (sctp)," *Oct2007*, 2007.

[8] C. Lameter *et al.*, "lftpsophisticated ftp program," *Internet Document: LFTP Manpage, http://www. dca. fee. unicamp. br/cgi-bin/man2ntml/n/net/man1/1ftp*, vol. 1, p. 1, 2001.

[9] R. Bansal, V. Gupta, and R. Malhotra, "Performance analysis of wired and wireless lan using soft computing techniques-a review," *Global Journal of Computer Science and Technology*, vol. 10, no. 8, 2010.

[10] G. Gent, C. Downing, and J. Dalton, "Comparative performance of wireless and powerline lans for streaming media," *Website at www. citeseerx. ist. psu. edu*, 2003.

[11] I. Gupta and P. Kaur, "Comparative throughput of wifi & ethernet lans using opnet modeler," *International Journal of Computer Applications*, vol. 8, no. 6, pp. 8–11, 2010.

[12] P. Grønsund, A. Jacobsen, T. O. Breivik, V. Hassel, G. Millstein, and T. Haslestad, "Heterogeneous all-ip wireless broadband with wimax, wifi and hspa," in *Personal, Indoor and Mobile Radio Communications, 2009 IEEE 20th International Symposium on*. IEEE, 2009, pp. 1128–1132.

[13] S. Manjula, C. Abhilash, K. Shaila, K. Venugopal, and L. Patnaik, "Performance of aodv routing protocol using group and entity mobility models in wireless sensor networks," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, vol. 2, 2008.

[14] U. Prathap, D. P. Shenoy, K. Venugopal, and L. Patnaik, "Wireless sensor networks applications and routing protocols: survey and research challenges," in *Cloud and Services Computing (ISCOS), 2012 International Symposium on*. IEEE, 2012, pp. 49–56.

[15] "Twisted documentation," July 02, 2015. [Online]. Available: https://media.readthedocs.org/pdf/twisted/latest/twisted.pdf

[16] A. Fettig, *Twisted network programming essentials.* " O'Reilly Media, Inc.", 2005.

[17] Available at http://searchnetworking.techtarget.com/definition/SCTP.

[18] "Aspera mobile - an open platform for rapid content acquisition and delivery."