Communication between multiple processes on same device using TCP/IP suite

Conference Paper · March 2017

DOI: 10.1109/C-CODE.2017.7918919

CITATION
1

READS
465

4 authors, including:

Syed Tauhid Ullah Shah
Huazhong University of Science and Technology
6 PUBLICATIONS **56** CITATIONS
SEE PROFILE

Izaz Ur Rahman
Abdul Wali Khan University Mardan
29 PUBLICATIONS **235** CITATIONS
SEE PROFILE

Communication between multiple processes on same device using TCP/IP Suite

Faizan Badshah, Syed Tauhid Ullah Shah, Syed Roohullah Jan, Izaz Ur Rahman
Department of Computer Science
Abdul Wali Khan University
Mardan, KPK , Pakistan
{faizanb44, tauhidullah116079}@gmail.com, (roohullahsyed, izaz}@awkum.edu.pk

***Abstract*—It is an established fact that the data transmission between two devices or processes is achieved via packets, according to the rules and regulations of TCP/IP protocol suite. The said transmission involves all the layers of TCP/IP protocol suite. The different processes involved in the transmission follow the same rule of the TCP / IP suite either on the same device or on different devices. In this paper, we proposed an algorithm with the help of which the data is being transmitted from one process to another process using the same file if both processes are on the same device. Our proposed algorithm uses only the two top layers, i.e., Application and Transport, of a TCP/IP protocol suite while, the rest of the layers, i.e., Network, Data Link and Physical, are not used. The data is transferred directly from the transport layer of one process to the transport layer of the other process. Our proposed algorithm saves a significant amount of computational power and other system resources that are consumed by the lower layers, i.e., Network, Data Link and Physical.**

Keywords—TCP/IP; UDP; Process-to-Process; Communication

I. INTRODUCTION

TCP/IP protocol suite is a set of protocols used at different layers, for governing communication between multiple processes. Vint Cerf and Bob Kohn developed it in 1978 [1]. According to TCP/IP protocol suite, the data are divided into small fragments, also known as packets. These packets contain the data and addresses of the source and destination for communication among endpoints. The UDP protocol is one of the most frequently used connectionless protocols implemented at transport layer [2]. The IP addresses of communicating devices and port numbers of the communicating processes are used as the source and destination addresses.

During the conventional communication of TCP/IP protocol suite, sending packets from one process to another process within the same device need all the layers [3] [4]. The data are divided into small segments and at each layer, a header is added to the fragmented data. At the receiving side, the segments are assembled and its peer, i.e., the corresponding layer at the receiver, removes the headers encapsulated by each layer. The only exception is the data link layer, which in addition of adding a header, also adds a trailer. The addition and removal of headers and trailer (only at the data link layer) put an excessive burden on each layer. This is due to the fact that significant amount of resources is consumed at each layer, when the communication is performed between two processes either on the same device or on different devices. The resources such as, computational time, memory consumption, transmission time, energy consumption are used excessively.

To solve this problem, we have developed an algorithm that minimizes the consumption of resources, i.e., only a subset of resources is used. To this end, our algorithm uses resources pertaining only to the first two top layers, i.e., application and transport. As a result, a substantial amount of resources is conserved, which can be allocated to other tasks related to a given network. The aim of the algorithm is to skip the extra layers, i.e., Network, Data Link and Physical, of a TCP/IP protocol suite. The IP addresses of the sender and receiver will be compared to check whether both processes are on the same device or not. If both processes are on the same device, then the sender will write the target data in a file and from that file receiver will retrieve the required data. The proposed algorithm provides a better speed and less use of system resources if both processes are on the same device.

The rest of the paper is organized as follows. In Section II, related work from literature is provided. In Section III, we present our proposed scheme. Finally, the paper is concluded and future research directions are provided in Section IV.

II. RELATED WORK

TCP/IP protocol suite is one of the most frequently used protocol suite in the cyber world. TCP/IP protocol suite follows a top-down hierarchical structure consisting of five interconnected layers. The top layer is the application layer, layer 4 is transport, layer three is network, layer 2 is the data link while, and physical is the designated as layer 1.

978-1-5090-4448-1/17/$31.00 ©2017 IEEE

Each one of these layers has its own functionality and characteristics. The TCP/IP protocol is applied not only to computers, but also to routers and switches with some modifications. At the computer end, all the layers are implemented while at the router end, only three layers are implemented, i.e., Network, Data Link and Physical.

The application is the top most layer of TCP/IP protocol suite and work similar to the top three layers, i.e., session, presentation and application of an OSI model. The main task of this layer is the implementation of software and human interaction. For data transfer, this layer depends upon the transport layer protocols to standardize communication for the establishment of a host-to-host-channels. Using this layer for communication, any application does not follow any specific data formats or rules [5].

The transport layer, on the other hand, is responsible for the delivery of data from one process to another process. At the transport layer, headers are added to the data, also known as segments, which contain the details of both the source and destination processes, not of the devices. According to the details present in each header, the packet is delivered to a target process without any underlying issues. The UDP and TCP protocols are the most frequently used protocols implemented at the transport layer. The UDP is a connectionless protocol, whereas, the TCP is a connection-oriented protocol. Connectionless communication, also known as CL-mode, is a method of data transmission. This method is used in those networks, which are based on packet switching. Using this method, the units containing data is addressed individually and based on information carried in each unit the data is routed. Under this method, a message can be sent from a sender to receiver without any prior arrangement. Connection-oriented model is a circuit-switched connection rather than a packet-switched. Before the data are to be processed, a semi connection or communication session must be established.

The network layer is responsible for device-to-device delivery. The IP protocol is the most frequently used protocol at the network layer. At the network layer, headers are added to data packets, which contain details of both devices not of the processes [6]. According to these details, data packets are delivered to a target device by a router or switch. For the successful transmission of packets, sockets are used. A socket is a complete address of a process on the network, which is a combination of device IP address and process port number.

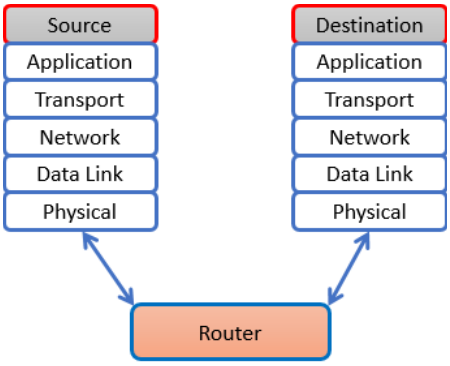
Physical and data link layers, also known as link layer, are the lower layers of a TCP/IP suite, described in [7] [8]. These layers either consist of communication protocols or methods for transmitting and receiving of data packets on a network medium. The data packets, also known as segments, are divided into frames at the data link layer. For error correction, checksum is provided to ensure that the receiver node receives an error-free data, i.e., frames. In addition, acknowledgements are provided to ensure that a reliable data transmission is performed. The physical layer deals with binary digits, i.e., 0 and 1, for the transmission of encrypted data and through a medium.

In the last two decades, most of the process-to-process communication has highlighted the detailed characteristics of small fragments of protocols and networks for process-to-process delivery of data [9] [10]. The existing works are mostly motivated by the influence of transport layer protocol at a specified time. The natural data sources from category 2 Internet service providers [11] to academic connection [12]. In [13], various performance measures such as, latency, bandwidth, jitter, availability of resources, and file sharing patterns, between different processes were discussed. Various process-to-process delivery have focused on the topological characteristics that are based on flow, level analysis [14], retrieval and the availability of content [15] [16]. In [17], the authors developed a signature-based payload technique for the identification of process-to-process traffic. The focus of the author's was on the examination of user's payload with TCP signatures, that is based on file downloading in multiple process-to-process protocols. Almost, all these previous works show that the research conducted either focused on performance measurement and enhancement in these measurements or on the efficiency and accuracy of data delivery.

III. PROPOSED WORK

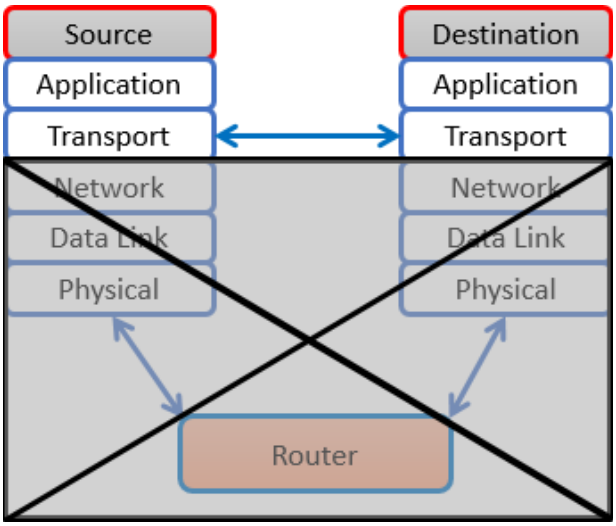
Until now, packets are transmitted by sockets using long route. The destination of packets is either the same device or other devices. This route consists of all layers of TCP/IP protocol suite, the involved medium between computers and router and, all the layers of the router. This route is perfect if both sender and receiver processes are on different devices as shown in Fig. 1.

Fig 1: Conventional communication between processes using TCP/IP



If source and destination are on the same device, then the previously mentioned route is a waste of time and resources. Based on this condition, the optimal path for transferring packets is the direct connection between the transport layer of source process and transport layer of destination process as shown in Fig. 2.

Fig 1: Proposed communication scheme between processes on same device



In this paper, we have developed an algorithm, which transfers packets directly from the transport layer of source to transport layer of destination process, if both processes are on the same device. We used transport layer because transport layer is responsible for process-to-process packet delivery. Through this algorithm, we skipped three layers of TCP/IP suite on computer, medium connecting router and computer and all layers of the router. If source and destination are on different devices, then the packet will follow its default path.

Initially, IP addresses of both sender and receiver are compared to check whether both processes are on the same device or not. If IP addresses are different then default path will be followed for packet transmission otherwise, our newly proposed path will be followed. The short path will be followed through our newly developed algorithm.

This algorithm will take data from each packet at sender side and write it to a file as shown in Algorithm 1. The receiver needs to know the location and name of the file. The receiver will retrieve data from the file created by the sender and copies it to a packet using thread containing our newly proposed algorithm. The file will be deleted automatically as a receiver side's process retrieves the data.

Algorithm 1: SEND PROCESS

```
1.  If ( IPAddress == LocalIPAddress) then
2.          Path ← "C:\tmp\" + LocalPort
3.          SendData ← open( Path , Write )
4.          Write (SendData , Packet.Data , size)
5.          Close ( SendData )
6.  Else
7.          Run default Send Code
8.  End if
```

In Algorithm 1, we used an IP address to check whether the destination program is on the same device or other device. For this purpose, we compared IP address in the packet to the local IP address. If the IP addresses do not match, data will be passed to a target device using default socket rules. If the IP addresses are matched, a new file will be created in C:\tmp\ directory whose name will be port number of current processes. We used port number as file name so that on a receiver side port number of source process can be retrieved. The said directory will also be created if it does not exist. A System Call will be used to open the newly created file. The new file must be opened with writing privileges. Another System Call will be used to take data from packet and write it in the above-mentioned file. The file must be locked during this process so that data should not be manipulated. After doing all these steps, the file will be closed and all its privileges will be released. After this, the receiver side program will retrieve data in the above-mentioned file.

Algorithm 2. RECEIVE PROCESS

```
1.  DiffDeviceThread
2.          Run default receive code
3.  End DiffDeviceThread
4.  SameDeviceThread
5.          FileExist ←False
6.          Path ← "C:\tmp\"
7.          AgainLabel
8.          Files[] ←Path.ListFiles
9.          Foreach FS as Files
10.                 If ( FS.isInteger )
11.                         Path = Path + FS
12.                         FileExist ←True
13.                         Exit Foreach
14.                 End If
15.         End Foreach
16.         If ( !FileExist )
17.                 Goto AgainLabel
18.         End If
19.         While ( !Path.CanWrite )
20.         End While
21.         ReceiveData ← open (Path , Read&Write)
```

```
22.         Read (ReceiveData , Packet.Data , size )
23.         FileName ←Path.getFileName
24.         Packet.Port ← FileName
25.         Packet.IPAddress ← LocalIPAddress
26.         Delete File
27.         Close ReceiveData
28.  End SameDeviceThread
29.  Start SameDeviceThread
30.  Start DiffDeviceThread
31.  If(SameDeviceThread or DiffDeviceThread Completes)
32.         Stop SameDeviceThread
33.         Stop DiffDeviceThread
34.  End If
```

At the receiver side, we split the receiving process into two threads. One thread will wait for data from socket according to the previous rules. The second thread will retrieve data from the file created by the sender process.

In the second thread, we will declare a string with directory address containing the target file. Next, we will retrieve all file names from the directory and save it in an array, named as "Files" in Algorithm 2. Next, we used a foreach loop to check the whole array to find a file whose name is just an integer. We used this statement because the name of the file created by sender process is the port number of the sender process and port number is always an integer. If any file with integer name is found, then the value of Boolean variable FileExist will change to true and the file name will be added to the path string. Next, we check the value of the variable FileExist, if it is true, the program will proceed to next statement otherwise, and it will go back to the statement where directory files are saved in an array. This process will be performed using labels. Now, we will check if the file can be opened using while loop. If the file is opened by another process, the while loop will continue looping until the said file is released. After this, the file will be opened using System Call with full privileges. The data from the file will be retrieved using the System Call and added to packet data. Now, we will retrieve file name from the path and add it as the port number to the receiving packet. Local IP Address will be added to the packet as its IP Address. After this, the file will be deleted permanently which was used for data transferring and the open function variable will be closed.

When a receiving function is called, it will start the two aforementioned threads. These threads will continue to work until their completion. If any one of the two threads complete its function then, the function will close the other thread.

IV. CONCLUSION

The paper focused on the problems associated with process-to-process delivery of data on a single device. Traditionally, process-to-process delivery in TCP/IP protocol suite involves all the layers, which results in extra resource consumptions in terms of bandwidth, computational power, memory and other QoS parameters. In addition, the process-to-process delivery results in higher latency and jitter. We presented a scheme, which bypassed the lower layers of the TCP/IP protocol suite. Our proposed scheme consists of two algorithms, which operates only for the two upper layers. Our proposed scheme used UDP protocol for direct transmission of information between two processes. Currently, we are working towards the implementation of our proposed algorithms using JDK headers. We aim to improve the efficiency and accuracy of data transfer between multiple processes running on a single host.

[1] http://www.historycomputer.com/Internet/Maturing/TCPIP.html

[2] http://ipv6.com/articles/general/User-Datagram-Protocol.htm

[3] http://searchnetworking.techtarget.com/definition/TCP-IP Posted by Margaret Rouse WhatIs.com

[4] The Disadvantage of Layering, , states: layering forms the basis for protocol design.I.S. Sect. 11.10, pp. 192

[5] Kevin R. Fall, W. Richard Stevens , TCP/IP Illustrated, Volume 1: The Protocols , 2nd editionPearson Education, Inc , May 2012

[6] Robert Braden, ed, Requirements for Internet Hosts – Application and Support, Network Working Group of the IETF, (October 1989)

[7] "Requirements for Internet Hosts -- Communication layers," , RFC 1122, IETF, R. Braden (Editor), October 1989

[8] "Requirements for Internet Hosts -- Communication layers," , RFC 1123, IETF, R. Braden (Editor), October 1989

[9] M. Izal, G. Urvoy-Keller, E.W. Biersack, P.A. Felber, A. Al Hamra, and L. Garc'es-Erice. Dissecting BitTorrent: Five Months in a Torrent's Lifetime. In *PAM*, 2004

[10] Robert Braden, ed, Requirements for Internet Hosts – Application and Support, Network Working Group of the IETF, (October 1989)

[11] P. Karbhari, M. Ammar, A. Dhamdhere, H.Raj, G. Riley, and E. Zegura. Bootstrapping in Gnutella: A Measurement Study. In *PAM*, 2004.

[12] N. Leibowitz, A. Bergman, Roy Ben-Shaul, and Aviv Shavit. Are File Swapping Networks Cacheable? Characterizing P2P Traffic. In *7th IWCW*, 2002

[13] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *MMCN*, 2002.

[14] S. Sen and J. Wang. Analyzing Peer-to-Peer Traffic Across Large Networks.In *IMW*, 2002

[15] R. Bhagwan, S. Savage, and G. Voelker. Understanding Availability. In *IPTPS 03*, 2003

[16] C. Gkantsidis, M. Mihail, and A. Saberi. Random Walks in Peer-to-Peer Networks. In INFOCOM, 2004

[17] S. Sen, O. Spatscheck, and D. Wang. Accurate,Scalable In-Network Identification of P2P Traffic Using Application Signatures. In WWW, 2004