International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 10, No.1 (Apr-2021)

http://dx.doi.org/10.12785/ijcds/100148

A Platform for Porting IPv4 Applications to IPv6

Yasir Mahmood[1] and Ayad Abdulqader[1]

[1]*Department of Computer Science, College of Computer Science and Mathematics, University of Mosul, Mosul, Iraq*

Received 12 May. 2020, Revised 1 Jan. 2021, Accepted 29 Jan. 2021, Published 21 Apr. 2021

Abstract: Developing a new application passes through several stages needing considerable effort from analysts, designers, and programmers, which can be extremely time-consuming and often results in the unnecessary expenditure of large sums of money. It is also known that there is currently a myriad of applications that only support the internet protocol version 4 (IPv4) network. Building a new application from scratch that supports the internet protocol version 6 (IPv6) network is very expensive. Re-engineering these applications that support the IPv4 network to make them support the IPv6 network is the best solution to reduce effort and cost. The aim of this paper is to design and implement a platform used for automatically porting the C++, C#, and visual basic IPv4 network applications to IPv6, by using a partial re-engineering approach. The re-engineered system portion shall be integrated with the current non-re-engineered portion. The main process of porting is conducted by replacing all the IPv4 dependent statements with their corresponding IPv6 dependent statements. Furthermore, all constant values of IPv4 addresses are replaced by suitable IPv6 ones. The proposed porting platform will reduce the duration time for porting network applications to support IPv6 to minutes rather than hours, days, months in manual porting.

Keywords: IPv4; IPv6; Porting; Re-Engineering; Platform

1. Introduction

Every device connected to the internet needs a logical address to be defined on the Internet. This logical address is used in the process of directing information to the concerned device. Given that the number of devices connected to the internet is increasing, this device needs logical addresses for all these devices connected to the internet. The current protocol used is the internet protocol version 4 (IPv4), and it is expected that in the projected future this protocol cannot meet the needs of addresses for new devices. The need for a protocol that meets these needs has emerged, leading to the role of internet protocol version 6 (IPv6) to solve this problem, as this new protocol provides more logical addresses than IPv4.

Because IPv4 cannot meet the needs of the current internet, the use of IPv6 is necessary to solve the problems encountered by IPv4. Nowadays, applications spread over the internet use the IPv4 network to communicate with servers or to communicate with each other. This paper proposes the design and implementation of a porting platform, which port applications (written for the IPv4 network) to applications working on the IPv6 network with the lowest cost and effort, instead of building a new application from scratch that needs great effort and huge cost.

A. literature review

Established scholars in [1] including companies and academic institutions have been inclined to port existing applications to IPv6. In [2],

International Business Machines Corporation (IBM) explores the principles behind a simple IPv6 system; in particular, how IPv6 solves the address space problems and large routing tables. IBM also discusses tunneling, mapped addresses, and porting IPv4 to IPv6 applications. In [3], it outlines the improvements that must be made for an IPv4-compatible application to run within a network environment of IPv6. Then, in [4], it discusses the transition from IPv4 to IPv6 network applications and the way to port the IPv4 broadcast applications to IPv6. This IPv6, however, does not have an implemented broadcasting concept, and several other issues during the transition from the platform to IPv6. Furthermore, in [5], the article describes the author's experiences in porting session initiation protocol (SIP) to IPv6. Next, [6] explains (1) how IPv6 applications can be introduced; (2) how IPv4 applications can be ported to IPv6; and (3) what specific application porting problems should be considered. After that, [7] discusses different issues when porting an IPv4 application to IPv6 with an emphasis on issues facing an application developer. In addition, [8] contains instructions for source code porting with some examples, used to illustrate the required changes in the client

E-mail: yaser.ali@uomosul.edu.iq, ayad_alezzi@uomosul.edu.iq

and server components. Moreover, [9] includes experiences of a case study for porting applications to IPv6. It presents the results of the effort to port OpenH323, an open-source H.323 platform to IPv6, and can serve as guidelines for other projects with similar goals. Additionally, [10] concludes with general recommendations for new IPv6 applications and provides some examples of code porting processes. Besides, [11] explains the application scenarios sensitive to changes to the IP protocol and the solutions that permit applications to work with different IP versions on heterogeneous networks.

B. *Comparing related works to the proposed porting platform*

The research mentioned above discusses how to port an application from IPv4 to IPv6 manually and provides some instructions that help in the porting process. Our work implements porting platform that also provides guidance to the developer in the process of porting application, as well as the proposed porting platform that ports the application automatically and reduces the effort and time to complete the porting process. The proposed platform provides the possibility to compile and build the source code after it is ported and creates an executable file for the ported source code, as well as providing the possibility to test the ported application.

More importantly, the development process using the proposed platform is performed automatically. The proposed platform provides the possibility to update the configuration of the development process and this feature allows the platform to be dynamic with the ability to port applications in languages other than C++, C#, and visual basic, by placing the appropriate instructions in the configuration file. Table 1 summarizes the comparison.

The proposed porting platform is distinguished from the other works with the following two main features:

1. Automatic: the porting process is done automatically.
2. General: can port any IPv4 application in any language by entering the appropriate instructions in the configuration file.

C. *Problem statement*

In the near future, the IPv6 protocol will become widespread and this development will require applications that use the IPv6 protocol. This entails the need for designing and programming new applications that meet the needs of this protocol. Building applications from scratch is not an easy task, needing effort, time, and financial resources. Most applications currently use the IPv4 protocol, however, taking advantage of IPv4 applications and re-engineering them to making them compatible with the IPv6 protocol is an important step in terms of reducing the effort, time, and money needed to create new applications that work on the IPv6 protocol. We propose to implement a platform to porting the IPv4 applications to IPv6 automatically.

TABLE 1. COMPARING RELATED WORKS AND PROPOSED PLATFORM

Reference number	Instruction for Porting source code	Save Effort	Save duration	Make porting platform	Testing	Automatically porting	static or
1	√	×	×	×	×	×	S
2	√	×	×	×	×	×	S
3	√	×	×	×	×	×	S
4	√	×	×	×	×	×	S
5	√	×	×	×	×	×	S
6	√	×	×	×	×	×	S
7	√	×	×	×	×	×	S
8	√	×	×	×	×	×	S
9	√	×	×	×	×	×	S
10	√	×	×	×	×	×	S
11	√	×	×	×	×	×	S
proposed platform	√	√	√	√	√	√	D

2. IPV4 AND IPV6 SOCKET PROGRAMMING

In this section, the structure of the socket in IPv4 and how to port it to IPv6 will be reviewed.

A. *Socket programming*

A socket is an abstraction through which an application may send and receive data, in much the same way as an open-file handle allows an application to read and write data to stable storage. A socket allows an application to plug into the network and communicate with other applications that are plugged into the same network. Network applications based on sockets have a basic procedure for client and server as shown in Figure 1.

1) *Porting IPv4 server application to IPv6*

Create socket:

SOCKET listening = socket(AF_INET, SOCK_STREAM, 0);

Bind IP address and port number to socket and listening

sockaddr_in hint;

hint.sin_family = AF_INET;
hint.sin_port = htons(54000);
hint.sin_addr.S_un.S_addr = INADDR_ANY;
bind(listening, (sockaddr*)&hint, sizeof(hint));
listen(listening, SOMAXCONN);

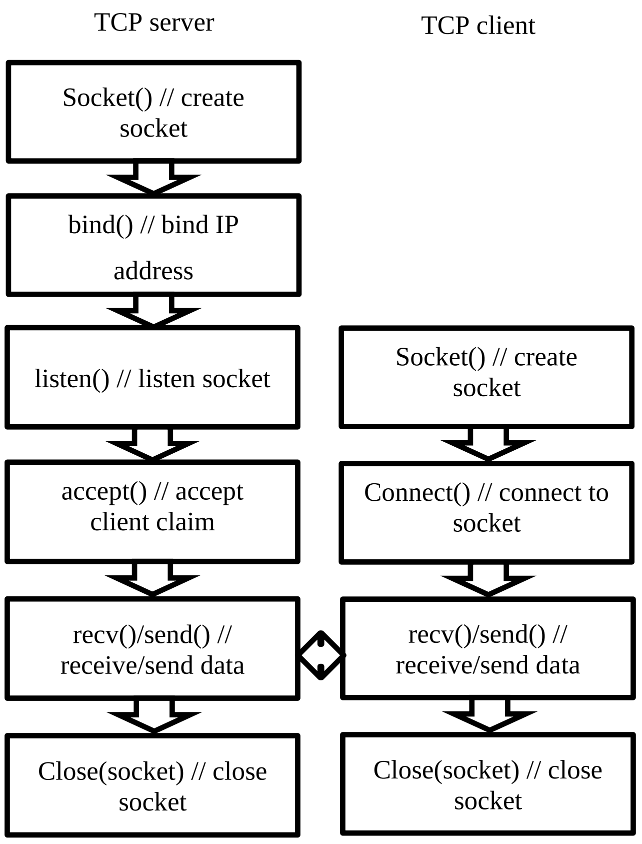
TCP server
TCP client
Socket() // create socket
bind() // bind IP address
listen() // listen socket
accept() // accept client claim
recv()/send() // receive/send data
Close(socket) // close socket
Socket() // create socket
Connect() // connect to socket
recv()/send() // receive/send data
Close(socket) // close socket

Figure. 1. The basic procedure for client-server application

Accept connection
SOCKET clientSocket = accept(listening, (sockaddr*)&client, &clientSize);
For porting the IPv4 server application to IPv6, all the IPv4 dependent statements are replaced with their corresponding IPv6 using Table 2.
SOCKET listening = socket(AF_INET6, SOCK_STREAM, 0);
sockaddr_in6 hint;
hint.sin6_family = AF_INET6;
hint.sin6_port = htons(54000);
hint.sin6_addr = in6addr_any;
bind(listening, (sockaddr*)&hint, sizeof(hint));
listen(listening, SOMAXCONN);
SOCKET clientSocket = accept(listening, (sockaddr*)&client, &clientSize);

2) *Porting IPv4 Client application to IPv6*

Create socket:
SOCKET sock = socket(AF_INET, SOCK_STREAM, 0);
Assign IP address:
sockaddr_in hint;
hint.sin_family = AF_INET;
hint.sin_port = htons(port);
hint.sin_addr.s_addr = in4addr_loopback;
connect to server:
int connResult = connect(sock, (sockaddr*)&hint, sizeof(hint));
For porting the IPv4 client application to IPv6, all the IPv4 dependent statements are replaced with their corresponding IPv6 using Table 2.
Create socket:
SOCKET sock = socket(AF_INET6, SOCK_STREAM, 0);
Assign IP address:
sockaddr_in6 hint;
hint.sin6_family = AF_INET6;
hint.sin6_port = htons(port);
hint.sin6_addr = in6addr_loopback;
connect to server:
int connResult = connect(sock, (sockaddr*)&hint, sizeof(hint));

B. *Domain Name System (DNS)*

If an application requires a name service, a DNS name resolver shall be used to convert it into a list of addresses of the destination. It should be pointed out that the IP version for carrying DNS queries is independent of the protocol version of the addresses in the DNS data records. This means that you can connect to a DNS server by IPv4 to look for IPv6-related information and vice versa. If you want to access your application through IPv6 or to connect with other services using IPv6, you would probably want to add (or ask your Systems Administrator to include) the relevant records in your DNS server using IPv6 data on the name used for accessing your device [21].

C. *Internet control message protocol (ICMP)*

The translator drops ICMP with a single hop and ICMP messages with unknown Type fields silently. The Header Format of ICMPv4 and ICMPv6 are almost the same for the remaining ICMP message. The only exception is the ICMP Parameter Problem message, which has an 8-bit pointer value in ICMPv4 and a 32-bit pointer value in ICMPv6 [21].

Interface parts of DNS and ICMP application programming interface (API) for both IPv4 and IPv6 are the same, so no need for porting. If that has any change in the API it is simply added to the API for the IPv4 and the corresponding API for the IPv6 in the configuration file [22].

3. PROPOSED ALGORITHM

The proposed algorithm relies on the concept of re-engineering in the process of developing applications from IPv4 to IPv6, where a partial re-engineering approach was relied upon. Figure 2 shows the proposed algorithm used to implement the porting platform.

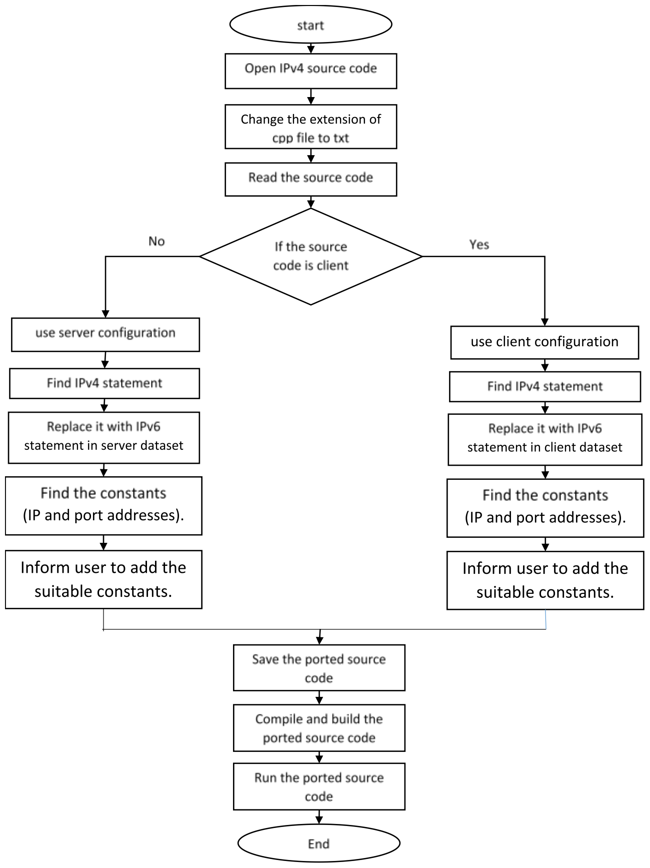
start
Open IPv4 source code
Change the extension of cpp file to txt
Read the source code
If the source code is client
No
Yes
use server configuration
Find IPv4 statement
Replace it with IPv6 statement in server dataset
Find the constants (IP and port addresses).
Inform user to add the suitable constants.
use client configuration
Find IPv4 statement
Replace it with IPv6 statement in client dataset
Find the constants (IP and port addresses).
Inform user to add the suitable constants.
Save the ported source code
Compile and build the ported source code
Run the ported source code
End

Figure. 2. Flow chart of the proposed algorithm

A. Software Re-engineering

Re-engineering is the process of amending the product in some new form. Re-engineering means making fundamental changes to the code. Software re-engineering is concerned with re-implementing legacy systems to make them more maintainable. Re-engineering may involve re-documenting the system, organizing and restructuring the system, translating the system to a more modern programming language, and modifying and updating the structure and values of the system's data. The functionality of the software is not changed and, normally, the system architecture also remains the same.

There are several specific approaches to re-engineering, that involve a particular way of achieving the same goal, i.e., to build the target system (re-engineered system version).

1) Partial re-engineering

In partial re-engineering, the system is only partially re-engineered, where it is separated into two parts: the first part for re-engineering and the second part remains the same. The re-engineered system portion shall be integrated with the current non re-engineered portion; hence this reduces potential system improvements. Firstly, the existing system shall be separated into two parts, i.e., a section which must be re-engineering and another which will not be re-engineering. The re-engineered portion can be part of the system that must be changed or a larger portion of the system that consists of the part that must be changed and another part whose inclusion simplifies the interface to the remainder of the system. Secondly, the re-engineering work must be done. Thirdly, the portions of the system must be merged to produce the target system. Figure 3 shows the three steps of the partial re-engineering approach. In this paper, partial re-engineering is used to re-engineer the applications, because only part of the applications related to IPv4 should be changed to accommodate the applications with the IPv6 network, thus achieving the desired goal of reducing cost and effort.

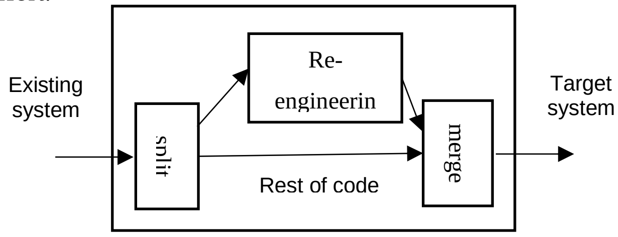

Figure. 3. partial re-engineering

B. Porting process

The process of porting applications is considered exceedingly difficult unless the developer (the person who performs the porting) is aware of the special structure of IPv6 as well as possessing knowledge of IPv6 programming. After much effort, through the use of partial re-engineering of applications, a quick method in the process of applying the porting will be obtained as this method provides a process of separating a portion of the source code for the application of IPv4 that needs to be changed in order for the application to be suitable for work on the IPv6 network. The process of porting the source code needs to be familiar with all aspects of developing IPv6 applications like the details of header files and libraries. The porting process passes through two main phases: analysis and generation. The former is used for finding all IPv4 dependent statements. The latter is used for generating the IPv6 version after replacing all IPv4 dependent statements into their corresponding IPv6 statements as shown in Fig. 4.

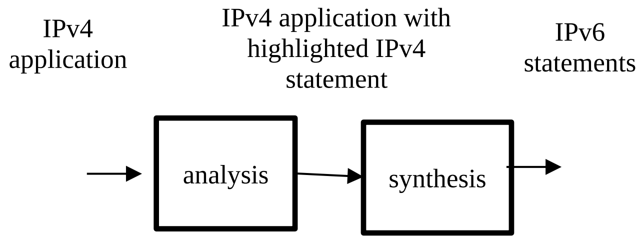

Figure. 4. The main phases of porting process

C. The design of the proposed platform

The porting process passes through several phases. The first phase is the reading, and it works on reading the source code of the application. The second phase is the analysis, where the IPv4 dependent statements are found. The third phase is the exchanging phase where it changes the IPv4 dependent statements to the equivalent IPv6 dependent statements. The last phase is the testing phase where it works to ensure the integrity of the resulting source code after porting and generate the IPv6 application. Figure 5 shows the phases of the proposed porting platform and the input and the output for each phase.

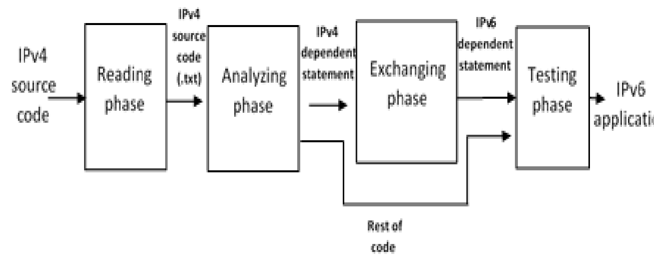

Figure. 5. The phases of the porting platform

1) *Reading phase*

The program works by reading the source code file written in C ++, C#, visual basic and converts the file extension to a text file to be readable and editable by the platform in the remaining phases.

2) *Analyzing phase*

After reading the source code file, the program uses either the client configuration or server configuration as a dataset in the porting process depending on the source code that was read. The dataset contains all the statements that must be ported as shown in Table 2 and Table 3. The program uses them to detect the IPv4 statements that must be ported with the IPv6 statements. There is an executable file checkv4.exe that is included in the Microsoft Windows Software Development Kit (SDK) and intended to furnish you with recognizes potential issues or features code that could profit from IPv6-competent capacities or structures. However, the recognition is limited on the basic items from Table 2 like AF_INET, sockaddr_in, while the other items are not recognized. The main process of this phase is to find the IPv4 dependent statements depending on the dataset and use them in the next phase.

3) *Exchanging phase*

After finding all the IPv4 dependent statements, the next step is to replace all the IPv4 dependent statements with the corresponding IPv6 dependent statements and inform the users to add the IP and port addresses that are used to communicate. The principal procedure of this phase is replacing the IPv4 dependent statements with the corresponding IPv6 dependent statements. That is shown in Table 2 and Table 3.

TABLE 2. IPV4 STATEMENT AND CORRESPONDING IPV6 STATEMENT FOR C++ LANGUAGE

IPV4 dependent statements	IPV6 dependent statements
127.0.0.1	::1
AF_INET	AF_INET6
SOCKADDR_IN	SOCKADDR_IN6
sockaddr_in	sockaddr_in6
sin_family	sin6_family
sin_addr.S_addr sin_addr.S_un.s_addr sin_addr	sin6_addr
sin_port	sin6_port
in4addr_loopback	in6addr_loopback
INADDR_ANY	in6addr_any

TABLE 3. IPV4 STATEMENT AND CORRESPONDING IPV6 STATEMENT FOR C# AND VISUAL BASIC LANGUAGE

IPv4 dependent statements	IPv6 dependent statements
127.0.0.1	::1
.Any	.IPv6Any
.InterNetwork	.InterNetworkV6
.Any	.IPv6Any
.Loopback	.IPv6Loopback

4) *Testing phase*

Save the ported source code and then compile it, and build the ported object code to generate a (.exe) file and test the new IPv6 application by running the ported source code as shown in Figure 6. The compilation and building processes are performed by the Developer Command Prompt for VS 2019 in the windows environment by the command (Cl/EHsc mycode.cpp) to generate the *mycode.exe*.

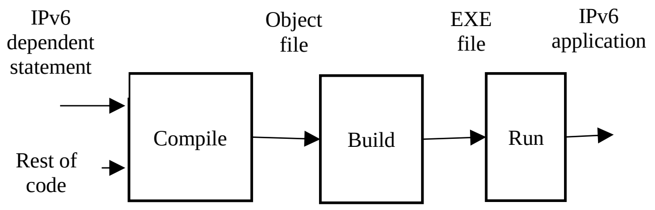

Figure. 6. Steps of the testing phase

4. Reducing cost evaluation

Depending on the basic constructive cost model (COCOMO) equations, the effort and duration are increased if the line of code is increased. The equation of COCOMO is written as

$$E = a * (KLOC)^{b}. \quad (1)$$

$$D = c * (E)^{d}. \quad (2)$$

$$Number\ of\ person = E/D. \quad (3)$$

where E is the effort applied in person-months, D is the development time in chronological months, KLOC is the estimated number of lines of code for the project (expressed in thousands). The coefficients a and c and the exponents b and d are given in Table 4.

TABLE 4. COEFFICIENTS VALUE

Software Project	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

The process of calculating the effort and duration time of the application to be ported takes place in three stages.

The first stage is calculating the effort and time during the reading phase; the process of reading the source code of the application to be ported. After reading the source code, we can easily count the line of code that exists in the source code file. Then, by applying the COCOMO equation we obtain the estimated effort and time needed to develop the source code of the application.

By using the COCOMO equations the Effort is calculated by this equation: $E = 2.4 * (KLOC)^{1.05}$ and the duration time is calculated by this equation: $D = 2.5 * (E)^{0.38}$

The second stage is calculating the effort and duration time during the analyzing phase; the process of finding all the IPv4 dependent statements and highlighted them by counting the line of code of the IPv4 dependent statements. Then, by applying the COCOMO equation we find the effort and time needed for analyzing and find IPv4 dependent statement.

The third stage is calculating the effort and duration time during the exchanging phase. The process of exchanging all the IPv4 dependent statements with corresponding IPv6 dependent statements is also considered extra effort and extra time for general effort and time, and by applying the COCOMO equation we find the effort and time needed for porting IPv4 dependent statements to corresponding IPv6 dependent statements. The proposed porting platform reduces the effort and time for developing the IPv4 application.

Table 5 shows the estimated effort and duration time and number of persons required for a different line of code for developing the source code. After applying the COCOMO equation, it is noted that the effort and duration increase progressively as the number of source code lines increases.

Figure 7 and Figure 8 show the curvature of the effort and duration time in Table 5 required to develop a different size of the line of code. We note that effort and duration increase depending on the size of lines of code.

We use three case studies of client/server applications in three different languages to evaluate the proposed porting platform. The size of these applications is 184 lines of code for C++ application, 151 lines of code for C# application, and 334 lines of code for a visual basic application.

Figures 9, 10, and 11 show the estimated effort, duration, and number of persons needed using COCOMO equations (1), (2), (3) for developing these three cases of client/server applications manually.

TABLE 5. ESTIMATED EFFORT, DURATION, AND NUMBER OF PERSONS FOR A DIFFERENT SIZE OF LINES OF CODE

lines of code	Effort (person-month)	Duration (month)	Number of persons
100	0.21	1.39	0.15
200	0.44	1.83	0.24
300	0.67	2.156	0.31
400	0.91	2.41	0.37
500	1.15	2.64	0.43
1000	2.4	3.48	0.68
1500	3.67	4.09	0.89
2000	4.96	4.59	1.08
2500	6.28	5.02	1.24
3000	7.60	5.40	1.40
3500	8.94	5.74	1.55
4000	10.28	6.06	1.69
5000	13.00	6.62	1.96
5500	14.37	6.88	2.08

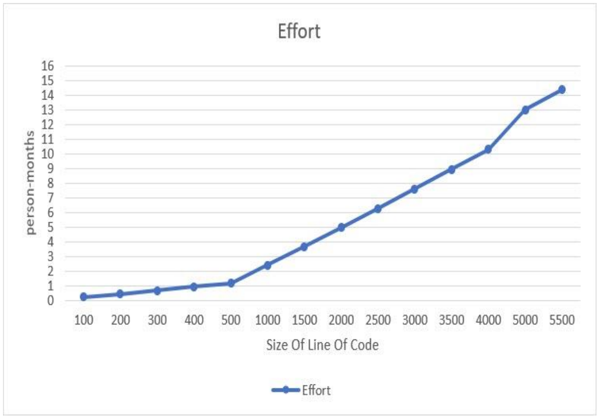

Figure. 7. Effort required to develop source code

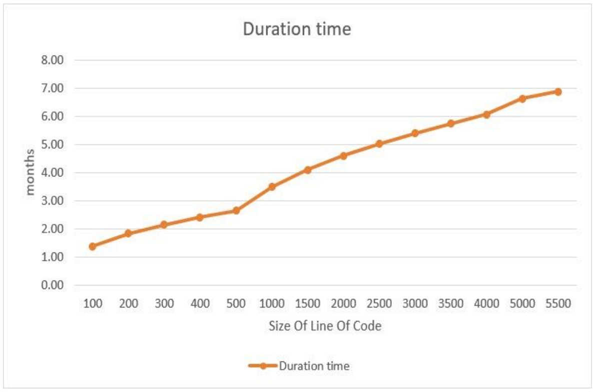

Figure. 8. Duration time required to develop source code

The first application in C++ language and its size is (184) lines of code, so to port this application manually we find it needs 0.41 person-month effort and 3.21 months' duration time and 0.13 person as shown in Figure 9. By using the proposed porting platform to port this application we note that it needs several minutes to complete the porting and building the source code and running the object, so by comparing the proposed porting platform to manually porting, the proposed porting platform reduces the duration time to several minutes while it was 3.21 months.

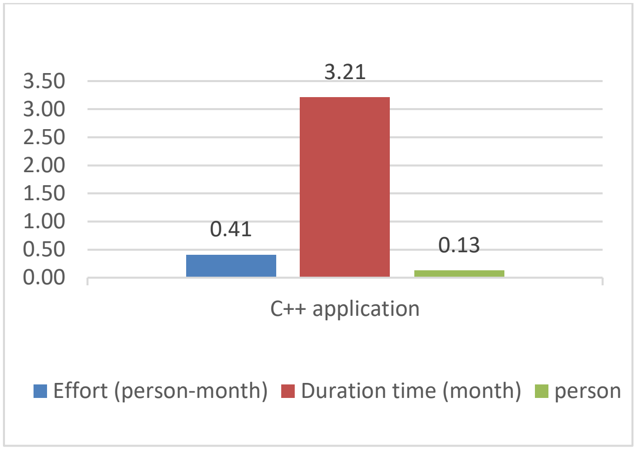

Figure. 9. The effort, duration time, and number of persons for developing applications manually for C++ language

The second application is in C# language and its size is (151) lines of code. To port this application manually we find it needs 0.33 person-month effort and 3.16 months' duration time and 0.10 person as shown in Figure 10. By using the proposed porting platform to port this application we note that it needs several minutes to complete the porting and compiling the ported source code and build the object. So, by comparing the proposed porting platform to manually porting the proposed porting platform the duration time was reduced to several minutes while it was 3.16 months.

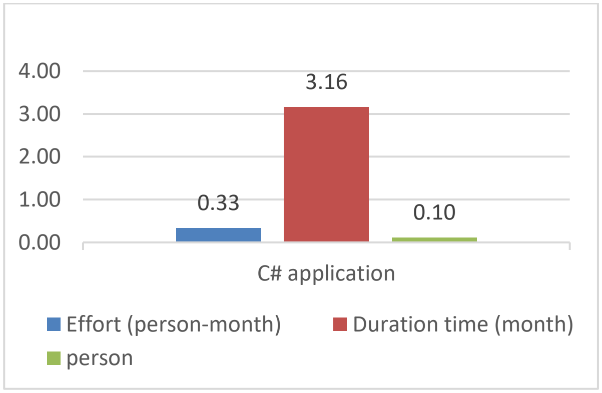

Figure. 10. The effort, duration time, and number of persons for developing applications manually for C# language

The third application in visual basic language and its size is (334) lines of code. So, to port this application manually we find it needs 0.76 person-month effort and 3.40 months' duration time, and 0.22 person as shown in Figure 11. By using the proposed porting platform to port this application we note that it needs several minutes to complete the porting and compiling the source code and build the object. By comparing the proposed porting platform to manually porting, the proposed porting platform reduced the duration time to several minutes while it was 3.40 months.

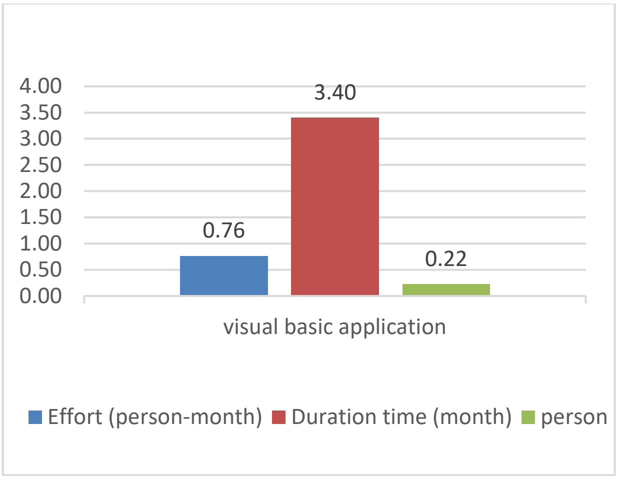

Figure. 11. The effort, duration time, and number of persons for developing applications manually for visual basic language

The proposed porting platform reduces the duration time needed for porting the application and economizes the effort. A significant result was obtained because the analyzing, translating, and testing was done automatically by the proposed porting platform.

5. Conclusion

Re-engineering turns out to be helpful for creating new applications utilizing existing ones. Re-engineering does not totally change the current applications, however, with the assistance of the current applications, it creates new and altered models, acquires a few capacities from the previously running applications, and adds a few capacities to deliver new applications. When we want to develop any network application, we have to look at the transport model in the source code. The transport model (socket procedures) is responsible for the network connection and the data transmission, so in our case of porting IPv4 network applications to IPv6, we must be porting the transport model to make the application run in IPv6 network. By using a partial re-engineering, we can separate parts of the code, which is the transport model, that needs to be ported from the rest of the code. The effort and duration are reduced, using the proposed platform and the result is IPv6 application from IPv4 application with the lowest effort and time. The effort and cost are decreased using this platform if the source code was large in a line of code compared to manual porting.

References

[1] R. Gilligan, S. Thomson, J. Bound, and W. Stevens, "RFC2553: Basic Socket Interface Extensions for IPv6." RFC Editor, 1999.

[2] S. Sundaram, "Writing a simple IPv6 program." 2001, [Online]. Available: https://www.ibm.com/developerworks/library/wa-ipv6.html.

[3] S. S. Johar, "Porting IPv4 Applications to IPv6 By." 2002,[Online].Available: https://www.hpc.mil/images/hpcdocs/ipv6/porting_ipv4tov6_johar_2002.pdf.

[4] H. J. and M. D H, "Transition of IPv4 Network Applications to IPv6 Applications [TIPv4 toTIPv6]," in *IEEE International Conference on emerging trends in computing(ICETiC-2009)*, Jan. 2009.

[5] T. Robles, R. Ortiz, and J. Salvachja, "Porting the session initiation protocol to IPv6," *IEEE Internet Comput.*, vol. 7, no. 3, pp. 43–50, 2003.

[6] "Implementing IPv6 Applications." 2010, [Online]. Available:

[7] K. Ettikan and T. W. Chong, "PORTABILITY ISSUES FOR IPv4 to IPv6 APPLICATIONS," *APRICOT*, 2001.

[8] E. M. Castro, "Porting applications to IPv6 HowTo," Lab. Over Next Gener. Networks.

[9] C. Bouras, A. Gkamas, D. Primpas, and K. Stamos, "Porting and performance aspects from IPv4 to IPv6: The case of OpenH323," *Int. J. Commun. Syst*., vol. 18, no. 9, pp. 847–866, 2005.

[10] T. De Miguel and E. M. Castro, "Programming guidelines on transition to IPv6," Transition, no. January. 2003, doi: 10.1.1.6.7440.

[11] E. M. Castro-Barbero, T. P. de Miguel-Moro, and S. Pavón-Gómez, "Transition of applications to IPv6," IPv6 More than A Protoc., vol. 6, no. 2, p. 15, 2005.

[12] H. Singh, "Software Reengineering: New Approach to Software Development," *International Journal Of Research In Education Methodology www.cirworld.com Council For Innovative Research*, vol. 1, no. 3. 2012.

[13] Stevens WR. Network Programming, vol. 1 (2nd edn). *Prentice-Hall: Englewood Cliffs*, NJ, 1998.

[14] M. Xue and C. Zhu, "The socket programming and software design for communication based on client/server," in *2009 Pacific-Asia Conference on Circuits, Communications and Systems*, 2009, pp. 775–777.

[15] Zeadally S, Raicu I. Evaluating IPv6 on Windows and Solaris. *IEEE Internet Computing 2003*; May–June:51–57.

[16] R. S. Pressman and B. R. Maxin, *Software Engineering_ A Practitioner's Approach-McGraw-Hill Education*, EIGHTH EDI. Raghu Srinivasan, 2014.

[17] L. Kalita, "Socket programming," *Int. J. Comput. Sci. Inf. Technol.*, vol. 5, no. 3, pp. 4802–4807, 2014.

[18] S. G. ZHANG, S. LIANG, and G. X. FU, "Transition of socket application from IPv4 to IPv6," *J. on Communications, Beijing, vol.27(11), pp.24-27*, November 2006.

[19] W. Stevens, M. Thomas, E. Nordmark, and T. Jinmei, "RFC3542: Advanced Sockets Application Program Interface (API) for IPv6." RFC Editor, 2003.

[20] Gilligan R, Thomson S, Bound J, McCann J, Stevens W. Basic socket interface extensions for IPv6. *Internet Engineering Task Force RFC 3493*, February 2003.

[21] M. E. Fiuczynski, V. K. Lam, and B. N. Bershad, "The Design and Implementation of an IPv6/IPv4 Network Address and Protocol Translator.," in *USENIX Annual Technical Conference*, 1998.

[22] J.-ichiro itojun. Hagino, *IPv6 network programming*. Amsterdam: Elsevier-Digital Press, 2005.

Yasir Ali Mahmood
B.Sc. in Computer Science 2009/ Mosul University.
M.Sc. student in Computer Science/ Mosul University2018-2020.
M.Sc. in Computer Science/ University of Mosul 2020

Ayad Hussain Abdulqader
B.Sc. in Computer Science 1989/ Mosul University.
M.Sc. in Computer Science 1993/ Nahrain University.
Ph.D. in Computer Networks 2012/ University Sains Malaysia.