Conference on Information Technology and Data Science

Encryption in a High-Speed Connectionless File Transfer System*

Robert Tornai, Dalma Kiss-Imre, Zoltán Gál

University of Debrecen, Faculty of Informatics
tornai.robert@inf.unideb.hu
imre.dalma99@gmail.com
zgal@unideb.hu

Proceedings of the 1st Conference on Information Technology and Data Science
Debrecen, Hungary, November 6–8, 2020
published at http://ceur-ws.org

Abstract

This paper describes the usage of encryption in FMFT (Fast Manager of File Transfer) program that is based on Xinan Liu's Reliable File Transfer Protocol. The system consists of a server and a client software pair utilizing UDP connection. The aim is to protect big data transfers by utilizing encryption maintaining the high transfer rates. Relying on Xinan Liu's solution a *Java*-based minimum viable product was developed and further enhanced later, which was even more improved by rewriting it in C++. By adding a graphical user interface to the client, it is more user friendly now. Furthermore, by using WebAssembly, the program is available for many platforms now. After presenting the performance hit of the usage of encryption on data packets, it will be discussed that thanks to multithreading, our application can utilize the CPU in a better way.

Keywords: Encryption, high-speed networking, high-performance computing, Internet, parallel communication

AMS Subject Classification: 68M10, 68M12

*This work was supported by the construction EFOP-3.6.3-VEKOP-16-2017-00002. The project was supported by the European Union, co-financed by the European Social Fund. This paper was supported by the FIKP-20428-3/2018/FEKUTSTRAT project of the University of Debrecen, Hungary and by the QoS-HPC-IoT Laboratory.

1. Introduction

We experienced at file transfers forth and back with a server hosted in the Gyires supercomputer's data center [6] that even gigabit connections can slow down to a few megabit range on a busy network in real-life use cases having even packet loss or damage [15]. This paper will focus on the encryption of the transferred packets and the performance hit introduced by the handling algorithm. Furthermore, the introduction of multithreading into the software will be discussed too.

A User Datagram Protocol (UDP) based software handles big data transfers a way better than Transmission Control Protocol (TCP) based solutions. There are UDP based systems as UFTP and UFTPD software pair that mostly accomplish our needed features [16], but they are not available in browsers, which is a basic requirement in our project. Because of this, we decided to write an own implementation designed for WebAssembly from scratch [14]. Furthermore, the Stream Control Transmission Protocol (SCTP [4, 8]) was implemented in our server-client pair [19]. It handles the data transfer and control feedback by utilizing both TCP and UDP protocols.

The structure of the paper is the following: in chapter two the development environment of the fast file transfer manager application is described. In chapter three encryption is presented. The multithreading work is detailed in chapter four. The achievements are enlisted in chapter five. Possible continuation and future research and development aims are described in chapter six.

2. Developing Environment

For the platform independent development the Qt 5.15.1 stable version was chosen [9]. The server software was tested even under the upcoming Qt 6 snapshots, the performance difference was not remarkable. This part of the work was carried out on a Ubuntu 18.04.5 LTS desktop workstation. Qt supports mobile equipment as Android and iOS besides a lot of desktop operating system targets as Windows, Linux and macOS [3]. Through WebAssembly technology almost native speed program running is available in modern web browsers [13]. To enable the WebAssembly target, $C{+}{+}11$ was chosen as the programming language since it is highly portable.

The client is built for the necessary target system natively having a GUI (see Figure 1.). The server is designed to run in headless mode. The client is planned to be made usable from command line also. After testing our SCTP file transfer system for desktop operating systems, we found that the performance of the TCP based FTP transfer and our SCTP and UDP based file transfer implementations need more investigation [11]. Different message sizes will be examined to find an optimal value for later use as default.

For testing purposes we used a gigabit network. The first solution to transfer files was using the standard FTP protocol which runs over TCP. Similar to SABUL [5], the initial approach in our software was to have a UDP data channel with a

TCP control channel. Project SABUL lived on as UDT until it was abandoned in 2013 [20]. Solutions based on UDP, especially by adopting rate-based algorithms, give better performance than other alternatives according to Cosimo Anglano et al. work [1]. Later this led us to the conclusion to refine our software to use a UDP channel for the control messages also based on Xinan Liu's Reliable File Transfer Protocol [10]. His work achieved the first place in CS2105 (Introduction to Computer Networks) Speed Contest AY15/16 Sem1. If the integrity of the packet is damaged, a resend is needed as in the case of lost of either the data packet or the acknowledge packet. It was measured that the CPU utilization of the *C++* code was less than our original *Java* implementation's [7].

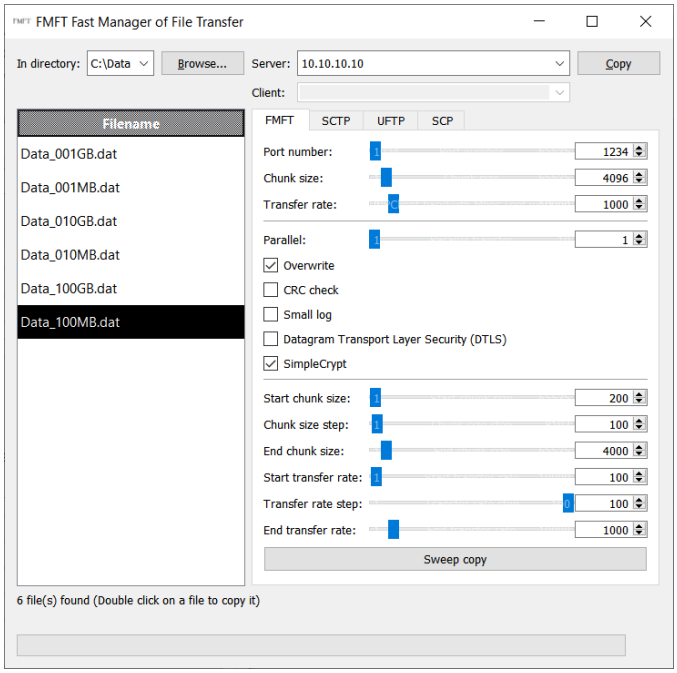

Figure 1. FMFT using SimpleCrypt for encrypting packets.

3. Encryption

For encryption Datagram Transport Layer Security (DTLS) was taken into consideration [2]. Due to its complexity, the first attempt to secure the packages sent over the network a simpler encryption algorithm was chosen [17]. Qt has a sample program having a very simple implementation for encrypting strings or byte arrays called SimpleCrypt. The details of the algorithm is described in their wiki pages [18]. Basically, it applies the exclusive `OR` bitwise operation between the data and the predefined key.

A 100 MB test file was used to carry out the measurements, its data was transferred from the desktop workstation to the test server residing in the Gyires supercomputer's server room. Results of the consecutive transfers starting from 200 bytes to 4000 bytes of chunk sizes can be seen in Figure 2. It can be observed that without encryption of data packets, the transfer rate stabilizes around 800 Mbps from chunk size of 700 bytes up to 4000 bytes.

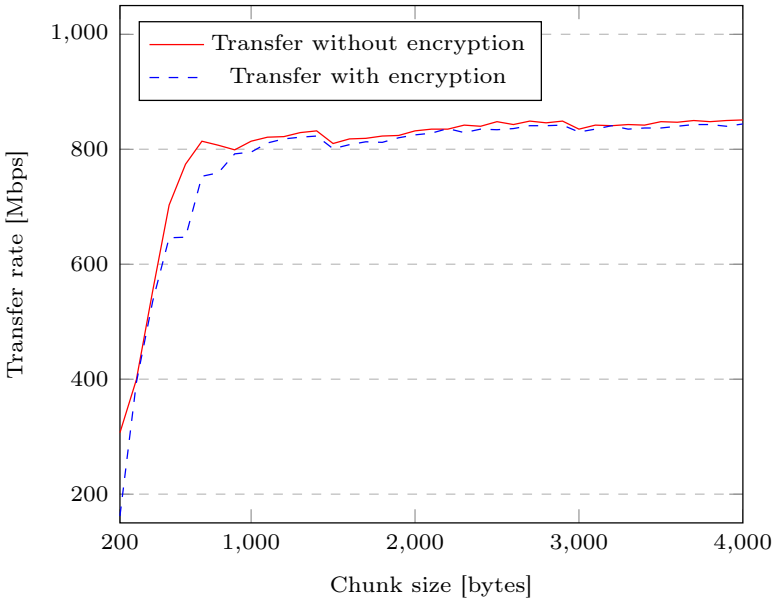

Figure 2. Throughput comparison of the implementations with different chunk sizes.

The test machine has an Intel® Core™ i7 CPU 920 @ 2.67 GHz with 18 GB memory having gigabit Ethernet connection, connected to the academic Internet network running Ubuntu 18.04.5 LTS desktop version. The test server was a Cisco UCS C240 M5 having gigabit Ethernet connection, connected to the Gyires supercomputer. The applied virtual machine runs Ubuntu 18.04.5 LTS server version under VMWare's vSphere 6.7 having 20 GB memory and 8 cores of an Intel® Xeon® Gold 6130 CPU @ 2.10 GHz.

By encrypting the data packets, the transfer rate cross the 800 Mbps boundary with 811 Mbps just at chunk size of 1100 bytes. The curve of results of transfer rates with encryption stays almost entirely under the curve of raw transfer of packages. Exception to this is chunk size of 2200 bytes where encrypted transfer rate of 836 Mbps was better by 1 Mbps over raw transfer rate of 835 Mbps. The difference is mostly under 1.5% at 900 bytes chunk size or higher. This result was consistent over various sized test data. For small chunk sizes as 600 bytes the difference can be as high as almost 20%. At a very small cost SimpleCrypt yields a good protection

against attacks coming from third parties on the same network segment.

SimpleCrypt has a compression option and we tested it also. It was found that the packet size for text files could by reduced up to 50%. However, the typical data for our use case is binary, and for this kind of data usually the gain was generally around 5%. In the worst cases the compressed data was even bigger than original packet. Because of the increased stress on the CPU besides the small gain, this was not tested thoroughly. The next chapter describes our work toward multithreading. When it will be implemented more extensively in our programs, we will move this feature in a distinct thread and will test it deeply.

4. Multithreading

Multithreading was introduced into the client software first, because at long workload the graphical user interface became unresponsible. Separating the GUI from the work thread, the GUI elements as progress bar or buttons started to give appropriate feedback. This could be achieved by using the signal-slot system of Qt. The next essential change was to separate the connection setup from the data transfer. This was done for the server also. Having a distinct thread for the data transfers was a huge boost especially for the server program, because one thread has a practical physical limit for the number of the clients to be served. Modern servers are massively multithreaded nowadays, thus the number of served clients can be raised this way.

To optimize handling of new connections, the connection setup and data transfer was split into two data ports. For each port a distinct thread is started up. Consecutive data transfers from the same client can disturb each other if packets from the first transfer are so late that they arrive during the second data transfer. To minimize the packet mixing, the data transfers got a data port range of 256 elements. This range is used as a circular array, thus giving enough time for wandering packets to arrive. 256 is practical because an unsigned byte can be used to index it without computing modulo values each time. In this way, even one client can initiate multiple data transfers concurrently to the server at the same time. The server is also having a data port range having an own thread assigned for each of them.

5. Results

Relying on Xinan Liu's work a high-speed connectionless file transfer system was developed by using UDP control channel. On busy connections data transfers were sped up over 800 Mbps by utilizing multithreading among other factors. There is always a chance for a man-in-the-middle attack. Even malicious nodes anywhere in the network chain can tamper with packets or the integrity of the data transfers can be damaged by accidental bit alternations. CRC checking fights efficiently against packet modifications, but do not hide the transferred information. Encryption is a

good tool for protecting sensitive data. By using SimpleCrypt to mitigate threats, the performance cost was generally under 1.5% at 900 bytes chunk size or higher. This way we recommend to use it all the time.

6. Future Work

Datagram Transport Layer Security (DTLS) is a transport communication protocol. It ensures security for datagram-based applications by letting them to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery [2]. Qt has a QDtls class for handling encrypted connections through UDP sockets which was introduced in Qt 5.12 [12]. We are planning to use it for securing the whole connection instead of the individual packets in the future. It may introduce an overhead on the data transfers because the two peers first have to successfully complete a TLS handshake. Broken transfers are still a problem, we will make a solution for resuming them. Multithreading shall be even more improved.

References

[1] C. Anglano, M. Canonico: *A Comparative Evaluation of High-Performance File Transfer Systems for Data-intensive Grid Applications*, 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (June 2004), pp. 283–288.

[2] *Datagram Transport Layer Security*, September 9, 2020, URL: https://en.wikipedia.org/wiki/Datagram_Transport_Layer_Security.

[3] L. Z. Eng: *Qt5 C++ GUI Programming Cookbook: Practical recipes for building cross-platform GUI applications, widgets, and animations with Qt 5*, 2nd, Birmingham, England: Packt Publishing Ltd., March 27, 2019.

[4] V. Gn: *Multimedia Streaming in MANETs using SCTP*, Paperback, LAP LAMBERT Academic Publishing, 2019.

[5] Y. Gu, R. Grossman: *SABUL: A Transport Protocol for Grid Computing*, Journal of Grid Computing 1.4 (Dec. 2003), pp. 377–386, ISSN: 1572-9184, DOI: https://doi.org/10.1023/B:GRID.0000037553.18581.3b.

[6] *Gyires supercomputer*, May 8, 2020, URL: https://hpc.unideb.hu/hu/node/219.

[7] *Java performance*, The page was last edited on 7 November 2019, URL: https://en.wikipedia.org/wiki/Java_performance#cite_note-43.

[8] S. Khatri: *SCTP Performance Improvement Based on: Adaptive Retransmission Time-Out Adjustment*, Paperback, LAP LAMBERT Academic Publishing, 2012.

[9] G. Lazar, R. Penea: *Mastering Qt 5: Create stunning cross-platform applications*, Birmingham, England: Packt Publishing Ltd., December 15, 2016.

[10] X. Liu: *ReliableFileTransferProtocol*, October 30, 2015, URL: https://github.com/xinan/ReliableFileTransferProtocol/tree/master/src.

[11] D. Madhuri, P. C. Reddy: *Performance comparison of TCP, UDP and SCTP in a wired network*, in: 2016 International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India, Oct. 2016, pp. 1–6, ISBN: 978-1-5090-1066-0, DOI: https://doi.org/10.1109/CESYS.2016.7889934.

[12] *QDtls Class*, September 9, 2020,
URL: https://doc.qt.io/qt-5/qdtls.html.

[13] *Qt for WebAssembly*, December 16, 2019,
URL: https://wiki.qt.io/Qt_for_WebAssembly.

[14] M. Rourke: *Learn WebAssembly: Build web applications with native performance using Wasm and C/C++*, 1st, Birmingham, England: Packt Publishing Ltd., September 24, 2018.

[15] H. Sawashima, Y. Hori, H. Sunahara: *Characteristics of UDP Packet Loss: Effect of TCP Traffic*, in: Proceeding of the 7th Annual Conference of the Internet Society, Kuala Lumpur, Malaysia, June 1997,
URL: https://web.archive.org/web/20160103125117/https://www.isoc.org/inet97/proceedings/F3/F3_1.HTM.

[16] B. Schuller, T. Pohlmann: *UFTP: High-Performance Data Transfer for UNICORE*, in: 7th UNICORE Summit 2011 Proceedings, ed. by M. Romberg, P. Bała, R. Müller-Pfefferkorn, D. Mallmann, vol. IAS Series 9, Toruń, Poland: Forschungszentrum Jülich GmbH, July 2011, pp. 135–142,
URL: https://core.ac.uk/download/pdf/34995345.pdf#page=144.

[17] *Simple encryption with SimpleCrypt*, September 9, 2020,
URL: https://wiki.qt.io/Simple_encryption_with_SimpleCrypt.

[18] *SimpleCrypt algorithm details*, September 9, 2020,
URL: https://wiki.qt.io/SimpleCrypt_algorithm_details.

[19] R. Tornai, D. Kiss-Imre, P. Fürjes-Benke, Z. Gál: *Developing a High-Speed Connectionless File Transfer System with WASM Based Client*, in: Proceedings of the 11th International Conference on Applied Informatics (ICAI) (Eger, Hungary, Jan. 29–31, 2020), ed. by I. Fazekas, G. Kovásznai, T. Tómács, CEUR Workshop Proceedings 2650, Aachen, 2020, pp. 392–399,
URL: http://ceur-ws.org/Vol-2650/#paper40.

[20] *UDP-based Data Transfer Protocol (UDT)*, September 9, 2020,
URL: https://udt.sourceforge.io/.