Conference on Information Technology and Data Science

CRC Check 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 CRC checking in an application named FMFT (Fast Manager of File Transfer) that is based on Xinan Liu's Reliable File Transfer Protocol. The system consists of a server and a client C++ program utilizing UDP connection. The aim is to transfer big files quickly by using this program on busy network connections. The *Java*-based minimum viable product relying on Xinan Liu's solution was rewritten in C++ and enhanced by adding control over chunk size. To make this program available for as many platforms as possible, WebAssembly programming language was used. This article introduces the performance results of the utilization of CRC checking of sent and received packages. Finally, it will be discussed that thanks to an advanced scheduling, our application can utilize the network more efficiently.

Keywords: CRC checking, 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

Transferring a huge amount of data between workstations and supercomputer nodes for processing introduces new problems: not only the upload process is slow, but the result can be huge also to download. Even gigabit connections can slow down to a few megabit range in a busy network in real-life use cases having even packet loss or damage [7], as we experienced this at file transfers forth and back with a server hosted in the Gyires supercomputer's data center [6]. This paper will focus on the integrity of the transferred packages and the performance hit caused by the handling algorithm. Furthermore, the scheduling of packages will be discussed too.

Transmission Control Protocol (TCP) based solutions have a lot of problems with traffic control. Usually, the transfer rate fluctuates a lot, even having just a small extra data transfer on the network from other nodes. A User Datagram Protocol (UDP) based software handles big data transfers a way better. Albeit, they have a challenge of the feedback of the unsuccessfully transferred packages. By creating a UDP based high-speed file transfer system, the demand to quickly move huge data between supercomputer nodes and workstations could be satisfied.

The UFTP and UFTPD software pair mostly accomplishes our needed features [16]. However, it is not available in browsers, which is a basic requirement in our project. The whole source code of the two applications of UFTP and UFTPD should be modified to be able to compile for WebAssembly. Instead of this, we decided to write an own implementation designed for WASM from scratch, this way it can be run in modern browsers [3].

Xinan Liu's Reliable File Transfer Protocol was chosen for the starting point of the development work, because beside its small code base it achieved the first place in CS2105 (Introduction to Computer Networks) Speed Contest AY15/16 Sem1. Moreover, we implemented the Stream Control Transmission Protocol (SCTP [4, 9]) in our software, which is able to handle the data transfer and control feedback by utilizing both TCP and UDP protocols.

The structure of the paper consists of four main chapters. The development environment of the Fast Manager of File Transfer application is described in chapter two. In chapter three CRC checking is presented. Performance hints related to scheduling of packages are given in chapter four. The achievements, possible continuation, future research and development aims are enlisted in chapter five.

2. Developing Environment

We chose Qt 5.15.1 stable version for the platform-independent development. The work was carried out on a Debian 10.5 workstation. Mobile equipment as Android and iOS has Qt support. Furthermore, it has a lot of desktop operating system targets as Windows, Linux and macOS. In recent years one of the most interesting new platform is WebAssembly [14], through this technology almost native speed program running is available from modern web browsers.

The chosen programming language is the highly portable $C{+}{+}11$, on the contrary of QML. The server was created for command line usage. The client was designed to have a native GUI on the actual operating system (see Figure 1). The WebAssembly client is lacking SCTP support right now. After we had tested 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 comprehensive investigations [11]. Different message sizes shall be examined to find an optimal value for later use as default.

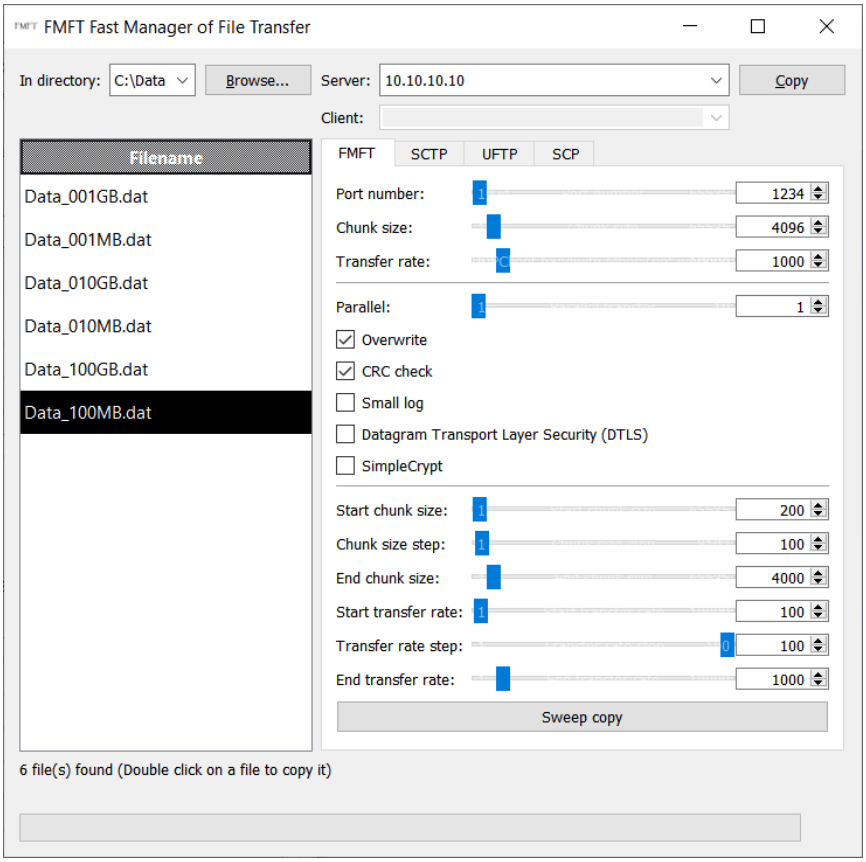

Figure 1. FMFT enhanced with CRC checking.

Tests were carried out on a gigabit network. Firstly, we worked with the TCP-based FTP protocol. Then the initial approach in our software was to have a UDP data channel with a TCP control channel similar to SABUL [5] which lived on as UDT until it was abandoned in 2013 [15]. Solutions based on UDP, especially by adopting rate-based algorithms, give better performance than other alternatives according to the work of Cosimo Anglano and Massimo Canonico [1]. Then we

decided to use a UDP channel in our software for the control messages also [10]. On lost of either the data packet or the acknowledge packet, a resend is needed. It is also true for the case when the integrity of the package is damaged. For the data transfer we used `QNetworkDatagram` and `QUdpSocket` classes. We could observe that the CPU usage of the *C++* version was less than the burden of our original *Java* code [8].

3. CRC Checking

The testing was carried over with a test file of 100 MB, we transferred data between a desktop workstation and the test server which is in the Gyires supercomputer's server room. Figure 2 shows values of the successive transfers starting from 200 bytes to 4000 bytes of chunk sizes. It can be seen that without computing CRC-16 for data packets, the transfer rate maintains over 800 Mbps from chunk size of 1100 bytes up to 4000 bytes.

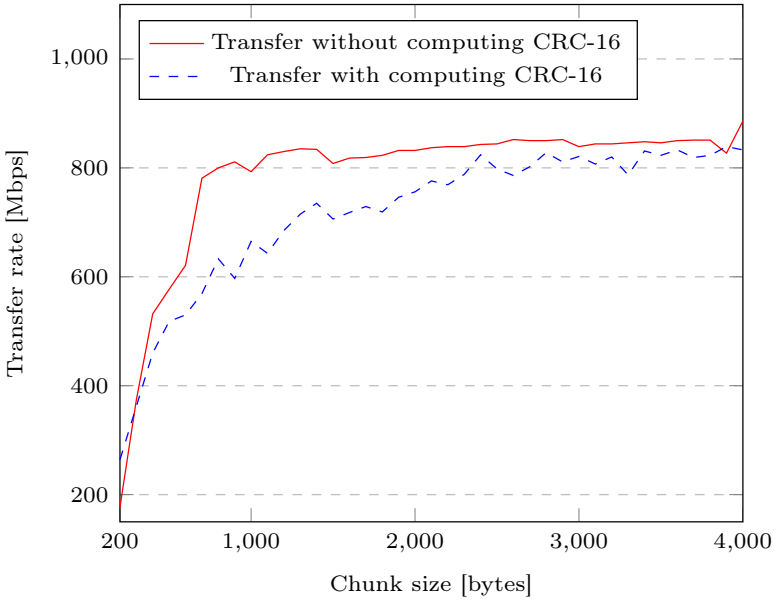

Figure 2. Throughput comparison of data transfers by FMFT without and with computing cyclic redundancy check.

By computing CRC-16 for data packets, the transfer rate cross the 800 Mbps boundary with 824 Mbps just at chunk size of 2400 bytes. The curve of results of transfer rates with computing CRC-16 values stays almost entirely under the curve of raw transfer of packages. This was consistent over various sized test data with the difference being around 5% with higher than 2400 bytes chunk sizes. For small chunk sizes as 900 bytes the difference can be as high as 37%. During our test runs

we have not encountered any package to fail the CRC-16 check. The reason for this was maybe the fact that the packages had to travel a relatively few hops. For longer data transfer paths having more hops it is potential to start having failing packages at CRC-16 checking.

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. 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 Debian 10.5 desktop version.

Qt has a function for calculating a checksum value since version 5.9: `quint16 qChecksum(const char *data, uint len)`, which is applied for the sent and received data packages in our server-client pair of programs. This function is using the CRC-16-CCITT algorithm having a 16-bit cache conserving (16 entry table) implementation [13]. It returns the CRC-16 checksum of the first `len` bytes of array `data`. It is calculated according to the algorithm published in ISO 3309 (Qt::ChecksumIso3309) [2]. Furthermore, the checksum is independent of the byte order (endianness). It is very important in multiplatform projects.

4. Scheduling of Packages

The first solution was to send packages continuously from the client to the server. It meant that a lot of packages got lost, while the CPU and the network stack was at maximum stress. The best packet transmission success rate for tests was 92.6% in the *Java* based minimum viable product. The reimplemented *C++* pair of software decreased the packet transmission success rate to 83.3% what is explained by the twofold increase in efficiency to stress both the CPU and the network stack. By introducing a transfer rate cap at software side the packet transmission success rate was increased over 90.0% again. The scheduling was based on one second length bursts of packages. It means that after sending out enough number of packages of a preset chunk size for the predetermined transfer rate, there was a pause in sending until the next one second time slot. It gives network buffers an unnecessarily heavy load at regular intervals. The next iteration was to fine tune the time interval to one millisecond. The backside of this solution was that the precomputed number of packages for one slot was rather small for higher chunk sizes. This means that the predetermined transfer rate may be achieved by only a relatively big margin, while the success rate improvement was not that great. Although, the CPU and network stress became smaller. The final solution was to use a nanosecond timer for scheduling. At each timer event we check one second history of sent packages. If it shows that the actual transfer rate is under the desired target, the required number of packages of the given chunk size is sent out to achieve the predetermined transfer rate. Under normal circumstances it means only one extra packet to be sent. The predetermined transfer rate is approximated accurately at a very low

margin this way. While it has even a little more stress on the CPU side, it further minimizes the stress on the network stack and buffers. Consequently, it leads to fewer lost and resent packages [12].

5. Conclusions and Future Work

Our first *Java* implementation was enhanced to a high-speed connectionless file transfer system by using UDP control channel relying on Xinan Liu's work. File transfers were increased over 800 Mbps on busy connections. Using WebAssembly, the client even runs in browsers. There is a chance for malicious nodes on the network for tampering with packets using a man in the middle attack. Besides this the integrity of the packets can be damaged by accidental bit alternations anywhere in the network chain.

CRC checking is a good and efficient way for identifying packet modifications. A CRC-16-CCITT algorithm based solution was implemented to mitigate this kind of damage. The transfer ratewith packets using a man-in-the-middle attack hit is around 5% for usable chunk sizes, while the performance hit on CPU is negligible. We have not encountered packets failing the CRC check. However, on longer network paths it will be a more useful tool for reducing packet retransmissions.

The entire data file should have a CRC-32 or CRC-64 test for ensuring the integrity of the process. We have started to work on encryption also. It may be improved by utilizing DTLS for securing the whole connection. The server shall be made fully multithreaded to have a thread for each transfer session to service more clients efficiently. Broken transfers will be enabled to resume later. The client will be modified to be a dual-pane file transfer and manager software.

References

[1] C. Anglano, M. Canonico: *Performance analysis of high-performance file transfer systems for Grid applications*, Special Issue: First International Workshop on Emerging Technologies for Next-generation GRID (ETNGRID 2004) Vol18.Issue8 (July 2006), pp. 807–816, doi: `https://doi.org/10.1002/cpe.976`.

[2] *ChecksumIso3309*, September 9, 2020, url: `https://doc.qt.io/qt-5/qt.html#ChecksumType-enum`.

[3] G. Gallant: *WebAssembly in Action*, 1st, Shelter Island, New York: Manning Publications Co., December 7, 2019.

[4] K. Gettman, R. Stewart, Q. Xie: *Stream Control Transmission Protocol (SCTP): A Reference Guide*, 1st edition, Addison-Wesley Professional, Oct. 2001, isbn: 978-0201721867.

[5] Y. Gu, X. Hong, M. Mazzucco, R. Grossman: *SABUL: A high performance data transfer protocol*, IEEE COMMUNICATIONS LETTERS (2003).

[6] *Gyires supercomputer*, February 19, 2021, url: `https://hpc.unideb.hu/hu/node/219`.

[7] Y. Hori, H. Sawashima, H. Sunahara, Y. Oie: *Performance Evaluation of UDP Traffic Affected by TCP Flows*, in: IEICE TRANSACTIONS on Communications, vol. E81-B, 8, Aug. 1998, pp. 1616–1623.

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

[9] V. C. Leung, E. P. Ribeiro, A. Wagner, J. Iyengar: *Multihomed Communication with SCTP (Stream Control Transmission Protocol)*, 1st, CRC Press Taylor & Francis Group, 2012.

[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] R. Mittal, R. Agarwal, S. Ratnasamy, S. Shenker: *Universal Packet Scheduling*, in:
URL: https://people.eecs.berkeley.edu/~radhika/ups_nsdi16.pdf.

[13] *qChecksum*, September 9, 2020,
URL: https://doc.qt.io/qt-5/qbytearray.html#qChecksum.

[14] *Qt for WebAssembly*, last edited on 4 December 2020,
URL: https://wiki.qt.io/Qt_for_WebAssembly.

[15] *UDP-based Data Transfer Protocol*, October 13, 2020,
URL: https://en.wikipedia.org/wiki/UDP-based_Data_Transfer_Protocol.

[16] *UFTP - Encrypted UDP based FTP with multicast*, April 22, 2020,
URL: http://uftp-multicast.sourceforge.net/.