A Variable-Length Network Encoding Protocol for Big Genomic Data

Mohammed Aledhari, Mohamed S. Hefeida, Fahad Saeed

A Variable-Length Network Encoding Protocol for Big Genomic Data

Mohammed Aledhari[1], Mohamed S. Hefeida[2], and Fahad Saeed[1]

[1] Western Michigan University, Kalamazoo MI 49008, USA,
[2] American University of the Middle East, AlEqaila, Kuwait

Abstract. Modern genomic studies utilize high-throughput instruments which can produce data at an astonishing rate. These big genomic datasets produced using next generation sequencing (NGS) machines can easily reach peta-scale level creating storage, analytic and transmission problems for large-scale system biology studies. Traditional networking protocols are oblivious to the data that is being transmitted and are designed for general purpose data transfer. In this paper we present a novel data-aware network transfer protocol to efficiently transfer big genomic data. Our protocol exploits the limited alphabet of DNA nucleotide and is developed over the hypertext transfer protocol (HTTP) framework. Our results show that proposed technique improves transmission up to 84 times when compared to normal HTTP encoding schemes. We also show that the performance of the resultant protocol (called VTTP) using a single machine is comparable to BitTorrent protocol used on 10 machines.

Keywords: Network protocol; Big Data; Genomics; HTTP

1 Introduction

Next generation sequencing (NGS) machines, such as the Illumina HiSeq2500 can generate up to 1TB of data per run and the data grows exponentially for large systems biology studies [21]. More often than not, these large genomic data sets have to be shared with fellow scientists or with cloud services for data analysis. The usual practice is to transfer the data using a networking protocol such as HTTP or FTP. Traditional networking protocols are oblivious to the data that is being transmitted and are designed for general purpose data transfer. Consequently, transfer takes exceedingly long time when large data sets are involved. Previous methods to improve transmission has focused on using FTP/HTTP protocols and multiple machines to increase throughput [14]. However, those solutions are inefficient in terms of hardware and do not exploit the additional data redundancy of DNA sequences for efficient transmission.
This paper introduces a data-aware variable-length text transfer protocol (VTTP) that is able to efficiently handle big genomic datasets. We assert that if the scope of the data is known a priori (such as genomic data) then networking protocols should be able to take advantage of this information for better efficiency. The key idea of VTTP is utilizing variable length codewords for DNA nucleotides in

content-encode of HTTP to maximize network resources usage [8]. Our proposed transfer technique decreases the size of the data that needs to be transmitted via assigning shortest possible codewords for repeated symbols; hence shortening the transfer time. The proposed VTTP does not require any extra hardware resources and is shown to be much faster than other competing techniques and protocols.

1.1 Paper Contribution

Creating the proposed content encoding mechanism relies on assigning variable-length binary codewords for the genomic symbols based on the frequency of the nucleotides in the dataset. The VTTP dynamically switches between the traditional charsets for symbols that do not belong to the genomic charset (A, G, C, T) and to our efficient encoding for DNA nucleotides. Lengths of genomic charset codewords is static variable-length in range of 1-3 bits long. We have implemented our encoding technique on top of HTTP for its universality and flexibility on various platforms. We are not aware of any other data-aware protocols that exploits redundancy in genomic data for efficient transmission. This VTTP is an improvement over our earlier work that used fixed-length codewords for genomic symbols i.e. 2-bit long for each character [3].

1.2 Paper Organization

The goal of this paper is design and implementation of a data-aware transfer protocol called VTTP. We implement VTTP by modifying the HTTP content encoding approach to transfer a big genomic dataset. We compare our results with traditional HTTP, FTP and BitTorrent like transfer protocols to transfer large genomic data sets.
The paper is organized as follows: Section II presents a short background of this work. Section III provides a summary of the related works that are used as a baseline for our implementation. Section IV discusses the overall architecture of the proposed protocol and model description and formulation. Experimental results of the baseline and the proposed content encoding approaches are presented in Section V. HTTP behaviors using the 2 mentioned encoding schemes are discussed in Section VI. Finally, we discuss future work and our conclusions in Section VII.

2 Background

2.1 Networking

There are two conceptual models for network transfer protocols in the network literature called open system interconnection (OSI) model [25] (7 layers) and the transmission control protocol/ Internet protocol (TCP/IP), or defense advanced research projects agency (DARPA) model (4 layers) [22]. A simple scenario to

transfer data between a server and client starts by generating data via an application layer to next (transport) layer using different protocols such as HTTP [8] and FTP [18]. Transport layer establishes a connection between the server and client through 2 main protocols that are transmission control protocol (TCP) [17] and user datagram protocol (UDP) [15]. Transport layer protocols pass data packets to the Internet layer that accomplish many functions. The protocols accomplish these functions such as packet routing using Internet protocol (IP) that put packets on network mediums such as WI-FI in the network interface layer. The normal HTTP is data-oblivious and hence cannot take advantage of redundant data or data with a limited global set. HTTP currently utilizes a fixed length binary encoding that converts each incoming symbol into fixed 8-bits even when the data can be encoded in fewer bits [13]. HTTP transfers data via locating data sources and encoding, after which it compresses and transfer data over network medium. This paper introduces a new content encoding scheme for the HTTP using a variable-length binary encoding that converts genomic symbols into fewer bits and makes it an efficient approach for big genomic data called VTTP.

2.2 HTTP compression algorithms

Data compression converts a certain input file into a smaller file size with compressed (low-redundancy) data. There are two main types of data compression: lossy and lossless. Lossless compression used by HTTP protocol are: compress [19], deflate [6], and gzip [7]. Compress is a file compression program that was originally developed under UNIX environment and uses LZW algorithm [24]. Deflate algorithm combines 2 other algorithms: Huffman coding [11] and Lempel-Ziv (LZ-77) [26] algorithms. GZIP is an open source compression technique that relies on LZ-77 algorithm. LZ-77 algorithm works by replacing repeated occurrences of symbols with their references that indicate length and location of that string which occurred before and can be presented in the tuple (offset, length, symbol). The basic performance metric for compression algorithms is a compression ratio, which refers to the ratio of the original to the compressed data size [20] as shown in the equation 1:

$$Compression\ ratio = \frac{compressed(output)data}{uncompressed(input)data} \tag{1}$$

For example, an 0.3 ratio means that the data occupies 30% of its original size after compression (positive compression). Whereas, a 1.2 ratio means that the output file size (after compression) is larger than the original file (negative compression). There are 2 important factors that impact the compression ratio: symbol *occurrences* and alphabet *scope*. We will use a GZIP as a baseline of compression technique for this implementation because it is a standard, fast and universal and is used by most of today's browsers.

2.3 Binary Encoding Schemes

In general, binary representations can be divided into 2 categories: *fixed* length binary encoding (FLBE) and *variable* length binary encoding (VLBE). The next subsections summarize these binary representations and highlights the pros and cons of each.

Fixed Length Binary Encoding The FLBE scheme, also called singular encoding, converts the alphabet symbols into a fixed number of output bits such as in ASCII code 8 bits long for each codeword [12]. For instance, an alphabet of 3 symbols a, b, c needs 2-bit fixed length codes for each symbol such as c(a) = 00, c(b) = 01, c(c) = 10 codewords, where c refers to coding. Based on the previous example, codeword length can be formatted by LogN-bit for N symbols alphabet. The main advantage of this scheme is both the client and server have prior knowledge of each symbol codeword length and is simple to implement. The disadvantage is that more number of bits are needed than are actually required, wasting precious bandwidth resources.

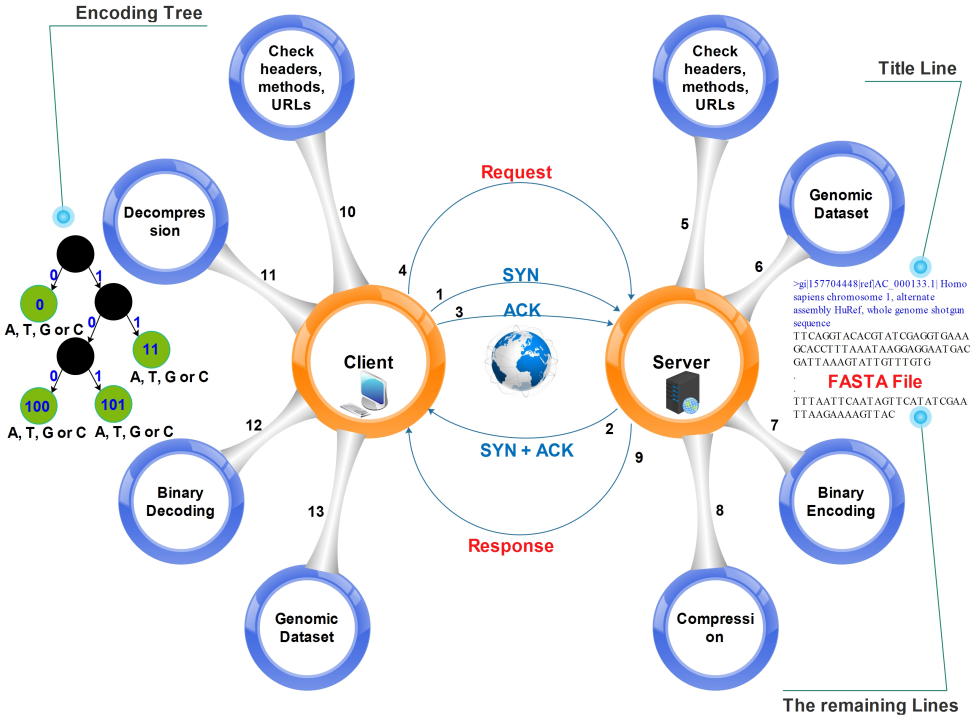


Fig. 1: Client-Server model for the HTTP protocol including our proposed encoding scheme. Also, FASTA file format that consists of 2 main parts: title and data lines.

Variable Length Binary Encoding A VLBE sometimes called uniquely decodable and non-singular code that converts n alphabet symbols λ_n into variable-length codewords, such as $\lambda_i \neq \lambda_j$ for all i and j symbols [10]. For example, the alphabet of 3 symbols i.e. *a, b, c*, can be encoded (E) in 3 unique variable-length codewords, such that $\mathrm{E}(a) = 0$, $\mathrm{E}(b) = 11$, $\mathrm{E}(c) = 100$. The variable length encoding assigns short codes to more symbol repetitions and long codes to less frequent repetitions similar to Huffman coding. Also, VLBE codes the alphabet symbols in a way that assures that each codeword is unique for a given data as an above example. The major advantage of the VLBE is saving space (or bit that needs to be transferred) that positively reflect on the transfer time if employed for data transportation. VLBE has the disadvantage of creating random encoding tables when data needs to be transferred in real-time which makes its implementation rather cumbersome and need extra processing by both the client and the server. However, we show the VTTP, which is based on VLBE, is much more efficient for transmission of genomic datasets and is shown to be 84 times faster than regular HTTP even with extra computational costs associated with VLBE. Binary tree structure is used in this implementation to simplify a binary encoding/decoding table representation [16]. By convention, the left child is labeled 0 and the right child is labeled 1 as shown in figure 1.

3 Related works

Protocols such as HTTP and FTP are techniques for general purpose data transfer and do not modify their behavior with the contents of the data. The HTTP, is a request/response protocol located in the first layer of the TCP/IP model (application), transfers data among web applications i.e. client(s)-server [9] as shown in figure 1. HTTP works via sending a request from the client to the server, followed by a response from the server to the client. Requests and responses are represented in a simple ASCII format (8 bits). The HTTP request contains many elements: a method such as GET, PUT, POST, etc. and a uniform resource locator (URL). Also, HTTP request/response contains information such as message headers, and compression algorithm (content encoding) along with needed data by the client. The server handles the request, then responds according to the specified method. After that, the server sends a response to the client, including the status code indicating whether the request succeeded, along with the reason(s), if any.

FTP is an application layer protocol of the TCP/IP that transfers files between 2 machines only i.e. client-server. FTP works via sending a request from the client to the server along with a valid username and password. FTP needs two connection lines: one for commands called control connection and another one to transfer data itself called data connection. In FTP, data compression occurs during the transfer phase using a deflate compression algorithm via MODE Z command [4]. We implement our encoding method on top of HTTP due to its versatility, friendly interface, its usage in one-to-many/many-to-many modes and security properties of HTTP that are absent in FTP [23].

4 Proposed model of HTTP content encoding

In this section, we illustrate our implementation of VTTP that utilizes VLBE for content encoding. Model formulations are also discussed for different possible scenarios of symbol repetitions to compare our proposed encoding to the current method used in HTTP encoding:

4.1 Model description

This subsection presents our implementation of HTTP that relies on VLBE content encoding to transfer big genomic datasets. This model assigns short variable codewords for genomic symbols. The fact that a genomic alphabet consists of only 4 symbols A, G, C, T makes it an ideal candidate for VLBE encoding and can be represented in less than 8-bits. We can encode the genomic dataset symbols in 4 unique decipherable codewords i.e. [0, 11, 100, 101] or simply [0, 3, 4, 5] as shown in figures 1 and 2. HTTP in most browsers, starts when client searches for specific data, the HTTP client side initiates a connection with the server that contains the required data. The connection between the client and the server establishes a 3-way handshake using the TCP/IP protocol. After establishing the connection, the client sends a request for certain dataset(s) to the server that checks the header(s), the method(s), and the resource address(es). The server retrieves the required data and starts to convert file symbols to binary form using a VLBE and passes it to a compression technique (GZIP in this implementation). FASTA file for a single sequence is described by a title line followed by one or more data lines. The title line begins with a right angle bracket followed by a label. The label ends with the first white space character. The data lines begin right after the title line and contain the sequence characters in order as shown in figure 1. At this point we read the first line of the FASTA file [5] using ASCII character set and the remaining lines are read using VTTP. The server starts encoding using the VLBE character set, compresses via GZIP and transfers the data (response) through network medium. The compressed data is received by the client along with header(s) and method(s) to store it. Client starts decompress the received data using GZIP to obtain the binary form.
We utilize a binary tree as a structure to represent our VLBE because it is faster to search, avoids duplicate values, and easy to decode at a receiver side. Assuming we have a file of 18 symbols with different symbol repetitions: a_1 appears in a frequency rate of 61%, a_2 has a repetition rate of 17%, and 11% for both a_3 and a_4 as shown in figure 2. In this example, the file has redundancy and VLBE works by assigning a variety of bit lengths reaching 1 bit per symbol (bps). There are 3 possible code lengths for the symbol a_1 in this example as appears in figure 2. As can be seen it still produces better results in contrast to the FLBE. Therefore, encoding 18 symbols in 29 bits yields an average of 1.6 bits/symbol in VLBE as compared to 144 bits in current HTTP encoding. The 3 VLBE possibilities of this example show 3 different code lengths 29, 37 and 47-bit long. However, VLBE still assigns short codes for the whole string in contrast to FLBE for the real-time applications i.e. data transfer.

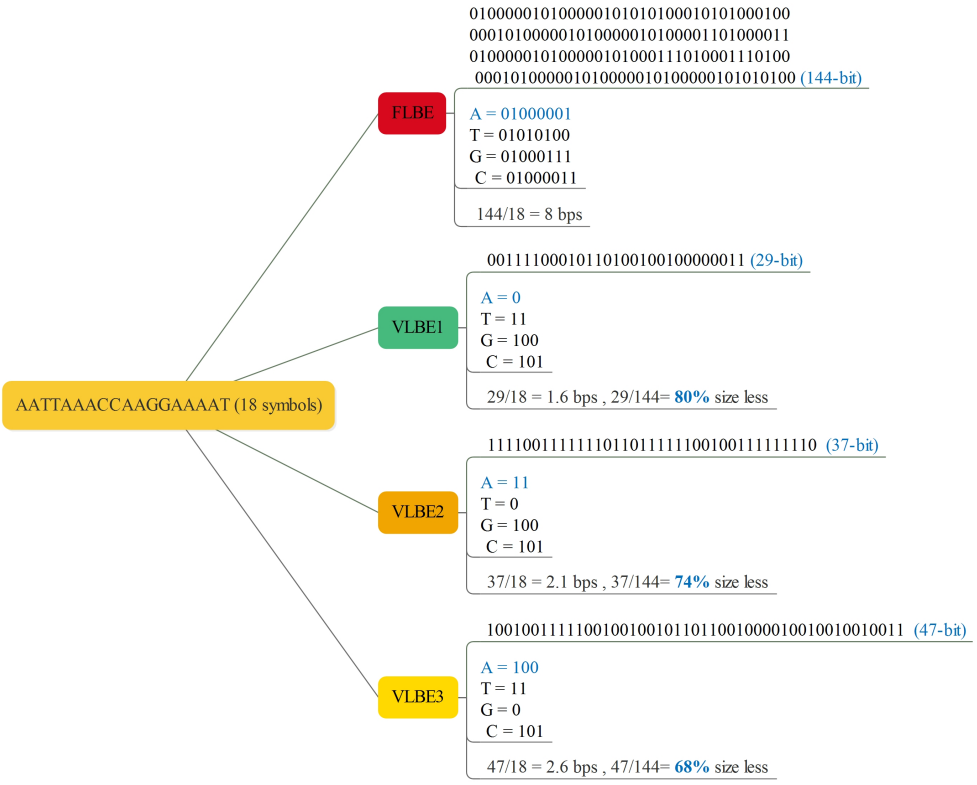


Fig. 2: Example of the HTTP content encoding schemes for a string of 18 symbols with variety of repetitions. 3 possible codes for the genomic symbols [A, T, G and C].

We designed variable codes in figure 2 in a way that makes it easy to decode using a prefix property (unambiguously). This property assigns a unique binary codeword for each single alphabet symbols to make decode operation easy in a client side. The variable length binary encoding has been used for static applications but transferring dynamic decoding trees has not been investigated.
Although the proposed VLBE does not guarantee minimal codewords for the data since the frequency of the data is not calculated (which is a compute intensive process for big data). However, it is expected that the proposed strategy will transfer data much more rapidly as compared to traditional HTTP protocol. For the current implementation, a 1-bit codeword length for a genomic symbol that has the highest occurrence (based on a local sample) and a 2-bit codeword length for a symbol with next largest repetition and 3-bit codeword length for

the remaining two symbols. The pseudocode for our protocol can be seen in algorithm 1.

Algorithm 1 VLBE-based HTTP

```
1: procedure ENCODING
2:     if !inputStream.hasGenomicFileheader then
3:         STOP (IS NOT Genomic File)
4:     else
5:         Encode the whole first line using a traditionalChar
6:     end if
7:     VLBE.writeGenomicSymbolsEncoder(outputStream)
8:     while !inputStream.EOF do
9:         if inputStream.GetChar() ∈ {A, T, G, C} then
10:            genomicChar ← inputStream.GetChar()
11:            code ← VLBE.encode(genomicChar)
12:        else
13:            traditionalChar ← inputStream.GetChar()
14:            code ← VLBE.encode(traditionalChar)
15:        end if
16:        oneByteStore.store(code)
17:        if oneByteStore.ISFull() then
18:            outputStream.write(oneByteStore)
19:            oneByteStore.empty()
20:        end if
21:    end while
22:    if !oneByteStore.ISEmpty() then
23:        outputStream.write(oneByteStore)
24:        outputStream.write(NumOfExtraBits)
25:    end if
26: end procedure
```

4.2 Problem formulation

Our analysis starts from the fact that in practical cases, the transfer time of data fluctuates due to several reasons such as bandwidth and message loss. The data transfer throughput (Th) measured by the minimum time needed (t) to transfer certain data amount (N) from the sender to the receiver. In order to minimize transfer time, we need to either maximize the bandwidth (which costs more) or minimize data size (which will reduce overall resources and time) and is being pursued in this work. The transfer throughput can be formalized as follows:

$$Th = \frac{N}{t} \tag{2}$$

A higher Th means better protocol throughput via transferring large data amount in less time. Consequently, the protocol throughput increases when a bit per symbol (bps) is reduced as much as possible. For example, transferring N string

symbols in B bits indicate efficiency of the encoding scheme as shown in:

$$bps = \frac{B}{N} \tag{3}$$

Here the minimum *bps* is better and hence shows a more time-efficient performance as compared to the original 8-bit transfers. The VLBE scheme utilizes all unused space in each single byte, which reduces the transfer time by decreasing data size in the next phases (compression, transfer, decompression, decoding to plain text). To simplify our model, lets assume we have an A alphabet as $\{a_1, a_2, a_3, ..., a_n\}$ that consists of n symbols, for the genomic dataset, A $\in$ {A, T, G, C}(nucleotides). Codeword can be represented in C(A) $\in$ {0,11,100,101}. The *time* complexity for encoding a string of N symbols using the HTTP fixed encoding is O(N) and the *space* complexity S can be calculated in:

$$S_{http} = \sum_{i=1}^{n} a_i * 8 \tag{4}$$

The space complexity of our proposed encoding scheme can be formulated in:

$$S_{vlbe}(A) = \sum_{i=1}^{n} a_i * codeword_{length} \tag{5}$$

VTTP has a time complexity of O(N/P), where P depends on the connection bandwidth and *bps*. Also, we can divide and formulate our model costs (C) into the following equations:

$$C_{total} = C_{computation} + C_{communication} \tag{6}$$

$$C_{computation} = O\left(C_{header\ check} + C_{encoding} + C_{compression}\right) \tag{7}$$

$$C_{communication} = O\left(C_{3\ way\ handshake} + C_{bandwidth}\right) \tag{8}$$

The equations (6-8) show that the computational of VTTP consumes an extra time to encode symbols since it switches between two charsets during reading the file. However, it takes much less time in next steps shortening the transfer time many times.

5 Experimental Results

In this section we discuss FTP and HTTP behaviors using both the current (fixed) and the proposed (variable) length encoding schemes for a variety of genomic datasets. The examined datasets that are FASTA format files were downloaded through two sources: National Center for Biotechnology Information (NCBI) [1] and University of California Santa Cruz (UCSC) [2] websites as shown in a table 1.

Table 1: Experimental datasets

IDs	Source	Size(KB)	Renamed
pataa	NCBI	563,318	1
refGeneexonNuc	UCSC	639,183	2
envnr	NCBI	1,952,531	3
hg38	UCSC	11,135,899	4
patnt	NCBI	14,807,918	5
gss	NCBI	30,526,525	6
estothers	NCBI	43,632,488	7
humangenomic	NCBI	45,323,884	8
othergenomic	NCBI	346,387,292	9

Table 2: Experimental setup

Specifications	Details
Processor	2.4 GHz Intel Core i7
Memory	8 GB 1600 MHz DDR3
Graphics	Intel HD Graphics 4000 1024 MB
Operating system	Windows 8.1 Pro
Download	87 Mb/s
Upload	40 Mb/s
Programming Language	C# .Net
Protocols	FTP, HTTP, BitTorrent and VTTP
Dataset sizes	550MB - 340GB

5.1 Experimental setup

This paper compares the proposed VTTP with FTP, HTTP and BitTorrent-like transfers. Several datasets size up to 430GB of FASTA files have been fed to these implementations to validate our approach. The experiments were performed on machines that have specifications shown in table 2.

5.2 Results

Our experimental results as shown in figures 3 -4, tables 3 and 4 and compared with FTP and HTTP.

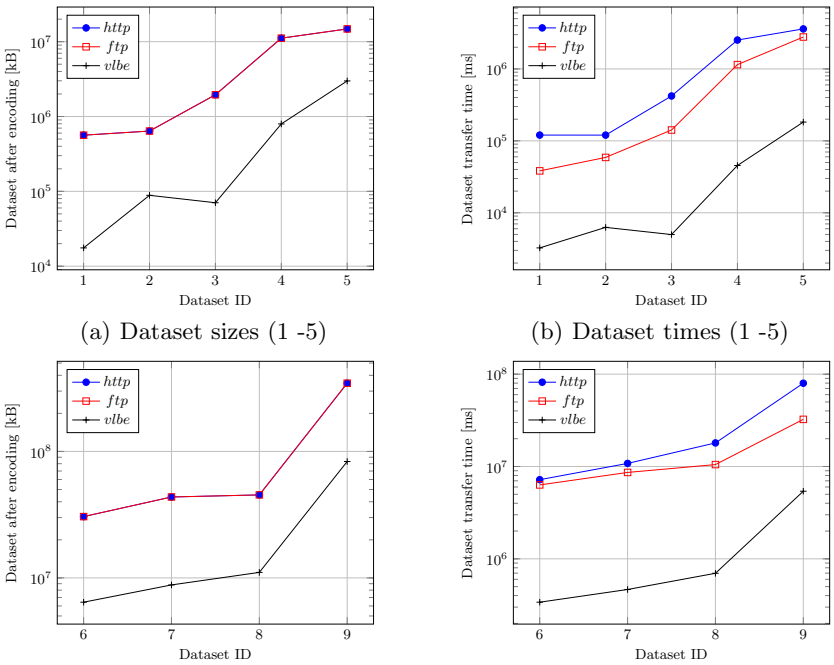
Table 3: Size reduction of VTTP

Dataset	HTTP(KB)	FTP(KB)	VLBE(KB)
1	5.63+05	5.63+05	1.75+04
2	6.39+05	6.39+05	8.85+04
3	1.95+06	1.95+06	7.04+04
4	1.11+07	1.11+07	7.95+05
5	1.48+07	1.48+07	2.98+06
6	3.05+07	3.05+07	6.42+06
7	4.36+07	4.36+07	8.79+06
8	4.53+07	4.53+07	1.10+07
9	3.46+08	3.46+08	8.32+07

Table 4: Time acceleration of VTTP

Dataset	HTTP(ms)	FTP(ms)	VLBE(ms)
1	1.20+05	3.82+04	3.25+03
2	1.22+05	5.88+04	6.26+03
3	4.20+05	1.41+05	4.97+03
4	2.52+06	1.14+06	4.54+04
5	3.60+06	2.77+06	1.82+05
6	7.20+06	6.31+06	3.39+05
7	1.08+07	8.62+06	4.65+05
8	1.80+07	1.04+07	6.97+05
9	7.98+07	3.24+07	5.39+06

As can be seen VLBE decrease the size of the data that needs to be transferred sharply and the corresponding decrease in the transfer time also decreases rapidly. As can be seen in the tables 3 and 4; 1.20 x 10^5 millisecond (ms) are required to transfer 550MB dataset using the traditional HTTP encoding, 3.82 x 10^4 ms via the FTP whereas 3.25 x 10^3 ms to transfer the same file via the HTTP-VLBE. This rate of transfer is about 37 times faster than HTTP-FLBE



(a) Dataset sizes (1 -5)

(b) Dataset times (1 -5)

(c) Dataset sizes (6 -9)

(d) Dataset times (6 -9)

Fig. 3: Dataset transfer time and size comparisons for 9 genomic datasets (1 - 9) over FTP, HTTP using both content-encoding FLBE and VLBE.

and about 12 times faster than FTP. Also, the 30GB dataset was transferred in 7.20 x 10^6 ms using the HTTP, 6.31 x 10^6 ms by FTP whereas it only took 3.39 x 10^5 ms to transfer the file using VLBE. This is about 21 times faster than HTTP and 18 times faster than FTP. We show results for up to 340GB. The average decrease in the size of the data sets as compared to HTTP and FTP is around 15 times. The corresponding decrease in the running time is 33 times faster as compared to HTTP and 16 times faster as compared to FTP over all data sets.

In order to compare the results of the proposed approach with that of existing BitTorrent protocol we implemented the latter approach as well. The results are shown in figure 4 for a 1GB FASTA file that was downloaded from the NCBI website (Homo_sapiens.GRCH38.dna_sm_toplevel). Only 1 machine (server) is used to transfer the same file using VLBE while n machines are used to transfer file utilizing HTTP over BitTorrent protocol. As expected, with increasing number of machines the time to transfer decrease sharply over BitTorrent. It can also be observed that the transfer time required for 1GB of genomic file using our proposed protocol (VTTP) with using only 1 machine is approx. equivalent to 10 machines used in parallel using BitTorrent protocols. This is due to massive reduction in size due to our encoding strategy. The results are presented for 1GB file only and the performance of VTTP is expected to increase with increasing size of the data due to increase in redundancy. Also note that employing VTTP

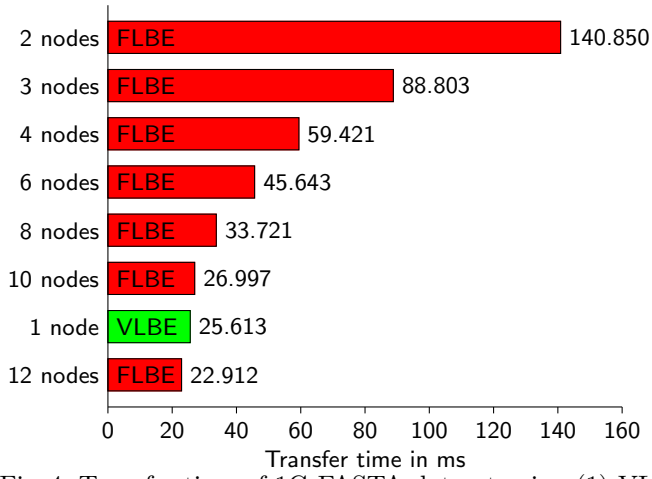


Fig. 4: Transfer time of 1G FASTA dataset using (1) VLBE-based machine and up to (12) FLBE-based machines work in parallel.

for multiple machines will massively decrease the time needed to transfer a file of a give size.

6 Conclusion

This paper presents design and implementation of a data-aware variable-length text transfer protocol (VTTP) that works on top of HTTP. Our protocol exploits the fact that genomic data is limited in its alphabet and is largely redundant. This allow us to design a variable length encoding scheme which decreases the size of the genomic data that needs to transferred over the network significantly. Consequently, enormous reduction in the time is also observed as compared to traditional HTTP and FTP protocols. Our results also show that using the proposed encoding scheme the resulting protocol using a single machine is better than 10 machines that use traditional HTTP protocol to transfer genomic data.

Acknowledgment

This work was supported in part by the grant NSF CCF-1464268.

References

1. (02 2016), `http://www.ncbi.nlm.nih.gov`
2. (02 2016), `http://www.ucsc.edu`
3. Aledhari, M., Saeed, F.: Design and implementation of network transfer protocol for big genomic data. IEEE 4th International Congress on Big Data (BigData Congress 2015) (June 2015)
4. Bhushan, A.: File transfer protocol. The Internet Engineering Task Force (1972)

5. Chenna, R., Sugawara, H., Koike, T., Lopez, R., Gibson, T.J., Higgins, D.G., Thompson, J.D.: Multiple sequence alignment with the clustal series of programs. Nucleic acids research 31(13), 3497–3500 (2003)
6. Deutsch, L.P.: Deflate compressed data format specification version 1.3. The Internet Engineering Task Force (1996)
7. Deutsch, P.: Gzip file format specification version 4.3. The Internet Engineering Task Force (1996)
8. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext transfer protocol – http/1.1 (1999)
9. Forouzan, B.A.: TCP/IP protocol suite. McGraw-Hill, Inc. (2002)
10. Gilbert, E.N., Moore, E.F.: Variable-length binary encodings. Bell System Technical Journal 38(4), 933–967 (1959)
11. Huffman, D.: A method for the construction of minimum-redundancy codes. Proceedings of the IRE 40(9), 1098–1101 (Sept 1952)
12. Krakauer, L.J., Baxter, L.: Method of fixed-length binary encoding and decoding and apparatus for same (1989)
13. Mogul, J.C., Douglis, F., Feldmann, A., Krishnamurthy, B.: Potential benefits of delta encoding and data compression for http. SIGCOMM Comput. Commun. Rev. 27(4), 181–194 (Oct 1997), http://doi.acm.org/10.1145/263109.263162
14. Néron, B., Ménager, H., Maufrais, C., Joly, N., Maupetit, J., Letort, S., Carrere, S., Tuffery, P., Letondal, C.: Mobyle: a new full web bioinformatics framework. Bioinformatics 25(22), 3005–3011 (11 2009), http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2773253/
15. P., J.: User datagram protocol. RFC 768 (August 1980)
16. Pajarola, R.: Fast prefix code processing. In: Information Technology: Coding and Computing [Computers and Communications], 2003. Proceedings. ITCC 2003. International Conference on. pp. 206–211. IEEE (2003)
17. Postel, J.: Transmission control protocol. The Internet Engineering Task Force (1981)
18. Postel, J., Reynolds, J.: File transfer protocol. The Internet Engineering Task Force (1985)
19. Rathore, Y., Ahirwar, M.K., Pandey, R.: A brief study of data compression algorithms. International Journal of Computer Science and Information Security 11(10), 86 (2013)
20. Sayood, K.: Introduction to data compression. Newnes (2012)
21. Singh, E.: Sap hana platform for healthcare: Bringing the world closer to real-time personalized medicine (October 2013)
22. Stevens, W.R.: TCP/IP Illustrated (Vol. 1): The Protocols. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1993)
23. Touch, J., Heidemann, J., Obraczka, K.: Analysis of http performance. ISI Research Report ISI/RR-98-463,(original report dated Aug. 1996), USC/Information Sciences Institute (1998)
24. Welch, T.A.: A technique for high-performance data compression. Computer 6(17), 8–19 (1984)
25. Zimmermann, H.: Innovations in internetworking. chap. OSI Reference Model&Mdash;The ISO Model of Architecture for Open Systems Interconnection, pp. 2–9. Artech House, Inc., Norwood, MA, USA (1988), http://dl.acm.org/citation.cfm?id=59309.59310
26. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Transactions on information theory 23(3), 337–343 (May 1977)