Communications and Network, 2014, 6, 175-185
Published Online August 2014 in SciRes. http://www.scirp.org/journal/cn
http://dx.doi.org/10.4236/cn.2014.63019

Scientific Research

Implementation and Evaluation of Transport Layer Protocol Executing Error Correction (ECP)

Tomofumi Matsuzawa[1], Keisuke Shimazu[2]

[1]Department of Information Sciences, Tokyo University of Science, Tokyo, Japan
[2]Service Operation Division, Internet Initiative Japan Inc., Tokyo, Japan
Email: t-matsu@is.noda.tus.ac.jp

Received 23 April 2014; revised 23 May 2014; accepted 17 June 2014

Abstract

Technologies for retransmission control and error correction are available for communications over the Internet to improve reliability of data. For communications that require the data reliability be ensured, TCP, which performs retransmission control, is often employed. However, for environments and services where response confirmation and retransmission are difficult, error correction technologies are employed. Error correction is generally implemented on UDP, but the existing framework implemented on UDP frequently does not consider the maximum frame size of the data link layer and relegates data division to the IP module. The IP module divides data according to the maximum size for the data link, and the receiving IP module reconstructs the divided data. For a data link layer typified by the current Ethernet with an error detection function, the frame is often destroyed upon error detection. At the IP module, the specification allows destruction of the entire dataset whenever divided data necessary for reconstruction is incomplete. Consequently, an error in a single bit results in a total loss of data handed to the IP module, and thus error correction performance declines with the increase in data size handed to the IP module. The present study considers the MTU of the data link layer and proposes error correction protocol (ECP) over IP, which decreases the transfer data volume flowing to the data link layer by dividing data into blocks of appropriate size based on designated error correction code and its parameters (thus improving error correction performance) and assesses the performance of ECP. Experimental results demonstrate that performance is comparable or better than existing error correction frameworks. Results also show that when a specification not ensuring the reliability of the data link layer was employed, error correction was superior to existing frameworks on UDP.

Keywords

Component, FEC, Transport Protocol, MTU

How to cite this paper: Matsuzawa, T. and Shimazu, K. (2014) Implementation and Evaluation of Transport Layer Protocol Executing Error Correction (ECP). *Communications and Network*, **6**, 175-185. http://dx.doi.org/10.4236/cn.2014.63019

1. Introduction

The spread of the Internet in recent years has come with increased communications in which real-time properties are emphasized, such as video teleconferences and voice communication. The transport layer for Web viewing and email services frequently employs the Transmission Control Protocol (TCP), which is embedded with functionality that the sending and receiving sides mutually handshake, conduct retransmission control for data errors or losses that occur in the communication pathway, and thus secure data reliability. TCP, however, is unsuited to communications that emphasize real-time properties. Instead, the User Datagram Protocol (UDP) is generally used. UDP is a connection-less protocol designed to be lightweight and high-speed. Unlike TCP, no handshaking occurs at the start and the end for communications and no mechanism is implemented to improve reliability, such as retransmission or congestion control. Techniques to improve reliability on UDP include retransmission control such as Automatic Repeat reQuest (ARQ) at the application and recovery of errors and losses with Forward Error Correction (FEC). Applications that emphasize real-time use, however, frequently employ FEC to avoid decline in real-time performance. UDP can handle data in lengths of at most 65,535 bytes, but when the data are handed to the IP module, the latter subdivides the received data according to the maximum transmission unit (MTU) size of the data link layer. The receiving IP module reconstructs the subdivided data to the size designated by the sending side and hands the data off to the UDP module. If even one frame of the subdivided data is destroyed in this process, then the IP module cannot reconstruct the data, and so the remaining subdivided data are all destroyed. Where the data size handled differs greatly between the MTU of the data link layer and UDP, an error of a single bit means loss of 1 UDP datagram. Thus, for technology such as FEC Framework [1] that performs error correction on UDP, in some cases the error correction performance declines. In recent years, technology that increases the MTU size of the data link layer has appeared, such as Ethernet Jumbo Frame [2], with the aim of attaining high speeds at the data link layer. Increasing the MTU size of the data link layer, however, tends to lower the error correction performance.

In response, this paper proposes a transport layer protocol with error correction capacity that considers the data size at the data link layer. The proposed protocol is implemented and its performance assessed.

2. Related Technologies

2.1. Ethernet Jumbo Frame

The standard specification for Ethernet defines the data size that can be transmitted in 1 frame as a minimum of 46 bytes and a maximum of 1500 bytes; however, Ethernet Jumbo Frame [2] expands the maximum size to between 9000 and 16,000 bytes while retaining the minimum size at 46 bytes. Since Ethernet Jumbo Frame has no standard specification, the maximum size differs by vendor or product. The specification simply expands the data size, which makes system-based support easy, improves network performance by several tens of percent, and achieves load reductions for network equipment.

2.2. Forward Error Correction (FEC)

FEC refers to technology that corrects errors and losses arising during communications. The sending side adds repair data (repair symbols) computed by specific calculations based on original data desired for sending, and the receiving side can restore the original data by specific calculations made on original data containing errors or repair data. The sum of the lengths for the original data and repair data is called the code length, and the ratio of original data length to code length is called the code rate. Generally, transmission efficiency is better for higher code rates but correction performance declines.

Algorithms existing for FEC include Reed-Solomon coding [3], LDPC coding [4], and BCH coding [5] [6].

For Reed-Solomon coding, multiple bits are considered one block (symbol), and code words are described by collections of symbols. Error correction is performed for each symbol. On the computer, 1 symbol is frequently implemented as 8 bits (1 byte), and N symbols constitute 1 code word. Within the word, K symbols represent the original data, and $N-K$ symbols represent the repair data. Correction of up to $(N-K)/2$ symbols is allowed.

2.3. FEC Framework

FEC Framework [1] is defined in RFC6363. It is defined between the transport layer protocol and the applica-

tion layer protocol with the aim of making the application and FEC independent. The encoding algorithm actually used and the generation procedure for the decoder that performs decoding are defined under the assumption that sender and receiver agree in advance. These agreements must be made separately using SDP as defined in RFC6364 [7].

3. FEC Framework Issues

Error correction on UDP using FEC Framework suffers a decline in error correction performance when the data size handled by the UDP module and MTU of the data link layer differ greatly.

The maximum size in the specifications for UDP is 65,535, which includes that for the UDP header. Thus, when IPv6 [8] is used, the effective maximum size is 65,527 bytes after subtracting 8 bytes for the UDP header (65,507 bytes for IPv4, since the IPv4 header size is included in the data size that the IP module can handle). For example, in order to stream multiple UDP datagrams in 8192-byte units over Ethernet, since subdivision is performed in terms of MTU of the data link layer by the IP module, frames streamed over Ethernet are sent in blocks of 1500 bytes. If an error in 1 bit occurs, then Ethernet has an error detection mechanism employing CRC coding for the header and data [9], and any frame with a detected error is destroyed. The receiving IP module attempts to reconfigure the UDP datagram of the size designated by the sender UDP module by using a frame other than the destroyed frame; however, unless all of the subdivided frames are complete, the datagram is destroyed. In this case, data lost in 8192-byte units must be restored. For sending the UDP datagram at a size less than the MTU of the data link layer, for example at 1024 bytes, the loss from the error of 1 bit is 1024 bytes, and error correction performance is improved. For too small a size, however, the ratio of repair data to original data increases, and thus the sending data volume also increases. FEC Framework has no data subdividing function based on the MTU of the data link layer. Instead, the application must manage this function. Neither the FEC Framework nor the Reed-Solomon FEC scheme [10] explicitly states specifications that determine data subdivision size based on the MTU. The present paper, therefore, proposes Error Correction Protocol (ECP) as a transport layer protocol, which includes functions of maximum size management, attachment of repair data necessary for error correction, a repair data algorithm, and parameter notification, and assesses the performance of ECP.

4. Error Correction Protocol (ECP) Specifications

4.1. Overview

Unlike UDP, ECP provides a mechanism that conducts error correction, not error detection. Unlike TCP, moreover, ECP is envisioned for an environment in which acknowledgements are not obtained, and features no function to control retransmission to improve reliability. The ECP datagram is divided between header and data, and, unlike FEC Framework, has no need for agreement in advance between sending and receiving on subdivided size, error correction encoding to be used, its parameters, etc. Such designations are made by entry to ECP header. Because it is defined as a transport layer protocol, ECP has a port number concept to enable distinguishing between upper layer services.

4.2. ECP Datagram Configuration

ECP header is configured as 28 bytes (octets) and is located immediately after IP header. The ECP datagram configuration is shown in **Figure 1**.

ECP header is stored immediately after IP header, and data or repair data (repair symbols) are stored immediately after ECP header. Original data input from the application (Application Data Unit: ADU) is divided into lengths no greater than the MTU of the data link layer after subtracting IP and ECP header lengths, and ECP header is attached to each data block. For example, if the MTU of Ethernet is 1500 bytes and ADU is 5000 bytes, then subdivision is into four blocks of 1250 bytes each, where 1250 is referred to as the block length. For an ordinary unicast communication, a path MTU discovery [11] is conduced in advance, and its value is calculated as the data link layer MTU. For cases in which the path MTU discovery cannot be conducted in advance, such as an IP multicast, the minimum MTU specified for IP (1280 for IPv6) is used. As shown in **Figure 1**, repair data are computed across multiple blocks. When subdivided block lengths differ, padding (filling with zeroes) is carried out to reach the maximum block length before generating the repair data. When Reed-Solomon code RS(255, 239) is used for error correction encoding, all blocks are first padded to make their lengths as long as

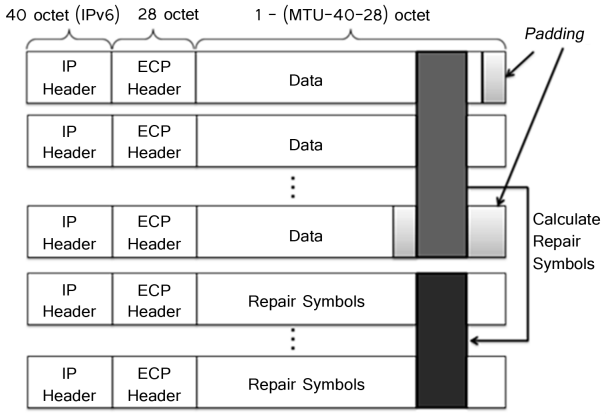

Figure 1. ECP datagram.

the largest of the 239 blocks, a 239 * (maximum block length) matrix is generated, and the respective values of the nth byte are used to calculate the values of the nth byte for 16 blocks of repair data. ADUs for 239 blocks do not need to be ready before data transmission can start. When ADUs are subdivided into blocks, one ECP header is attached to each block and handed to the IP module for sending. Repair data are generated when the 239th block is handed to the IP module, and then 16 blocks of repair data are additionally handed to the IP module. Thus, ECP must store 239 blocks worth of data in a buffer. Sent data do not need to include the padded portions. The receiving ECP module arranges block lengths of 239 + 16 blocks by padding to conduct error correction. If the number of blocks after the subdividing of input data exceeds 239, then repair data are generated and sent after sending the 239th block, the next block is handled as the first block of the next matrix, and this process is repeated. Thus, either storage of multiple ADUs in one matrix or storage of one ADU across two or more matrices by block subdivision may occur, but each block's ECP header, as described below, contains ADU number (Identification) and position within ADU (Data Offset) that enable reconfiguration of ADU at the receiving ECP module. Multiple ADUs are not stored within a single block, however.

4.3. Header Configuration

The ECP header format is shown in **Figure 2**.

Source Port stores the source port number and Destination Port stores the destination port number, both as 16 bits. ECP uses port number, such as TCP and UDP, to distinguish services.

Data Length stores the length of ADU handed from the application as 32 bits. Sequence Number is used to correctly rearrange received blocks when ECP conducts error correction. The first bit of Sequence Number is used as a flag (Repair Flag) to distinguish the data added after the header as either original data (0) or repair data (1) generated as a Repair Symbol. FEC Number stores the value representing the error correction code to be used and its parameters as 16 bits. For Reed-Solomon codes, for example, the RS(255, 239) code and RS(255, 249) code are defined by different numbers. If this field is 0, then error correction encoding is designated as unused. Block Length stores the lengths of each block. Identification stores the number attached per ADU. Data Offset stores the location (in byte units) from the beginning of ADU. Since ECP subdivides input ADUs into blocks and stores this in a matrix, the receiving ECP module must perform reconfiguration of the data, at which point the values stored in Identification and Data Offset are used. In fields other than header Sequence Number, FEC Number, and Header Repair Symbols of ECP header (fields shaded in grey in **Figure 2**), repair data calculated on the basis of the values of the same fields stored in the blocks of subdivided ADU, when used as a repair data header. As shown in **Figure 3**, the generation method of repair data is carried out in the same manner as the computation method for data by using the encoding method designated by FEC Number. In **Figure 3**, Sequence Number, Block Length, Identification, and Data Offset are provided as four examples, but the same generation method for repair data applies to all fields shaded in grey in **Figure 2**.

The example in **Figure 3** shows ADUs handed off from the application to the ECP module in sequences of 2000 bytes, 2800 bytes, and 1200 bytes, when the Ethernet MTU is 1500 bytes. If the three blocks in **Figure 3** whose Sequence Numbers are 2, 3, and 4 were lost, then even though the other blocks can be used to correct

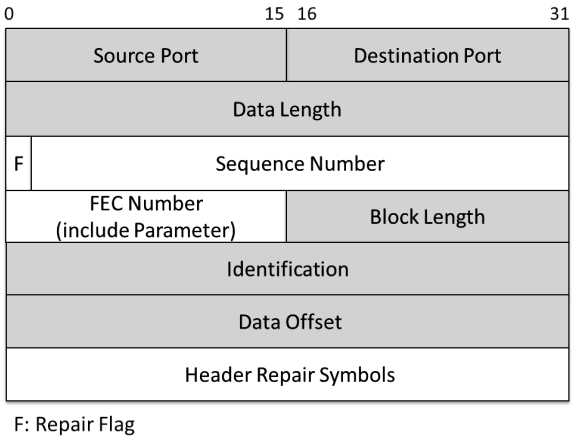

Figure 2. ECP header format.

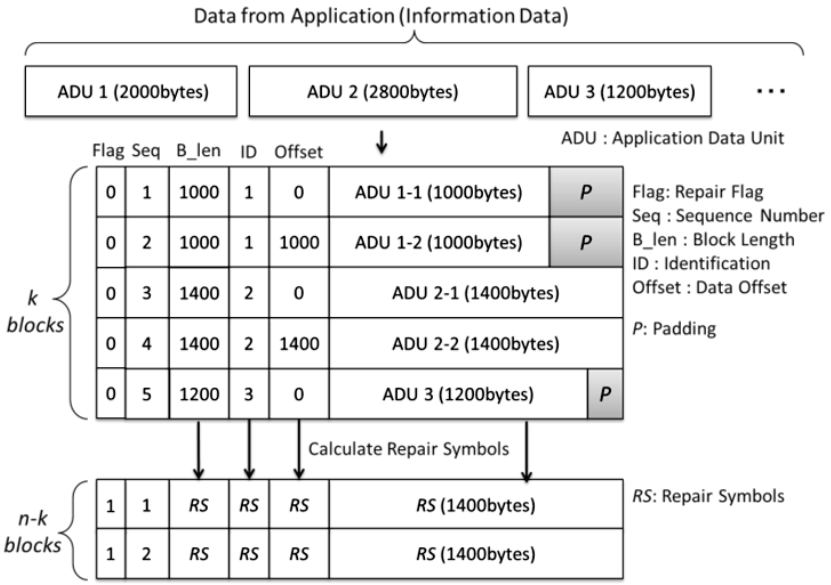

Figure 3. Flow of data division and repair data generation.

these three blocks and recover the data, unless the header information of Data Length, Block Length, Identification, and Data Offset is available, the ADU cannot be reconfigured from the restored blocks. Sequence Number and Repair Flag fields within the repair data header are used to position the blocks storing repair data in the matrix; therefore, flags are stored that indicate Sequence Number for a repair data block and status as a repair data block instead of Header Repair symbols. These two fields do not need to be restored from other blocks when ADU is reconfigured, since Sequence Number of any lost block can be restored from its position in the matrix. The FEC Number field is the value used to decide the matrix size, but this is unnecessary as header information of a restored block. Thus, correction symbols do not need to be stored in the header. In Header Repair Symbols shown in **Figure 2**, only 4 bytes of header repair data generated as RS(28, 24) code from a header of 24 bytes, excluding Header Repair Symbols, are stored. This field is used when header errors in block units are corrected. The type and parameters of error correction encoding used for header error correction and existence or absence of header error correction can be arbitrarily and dynamically changed by FEC Number in the same way as data error correction encoding and its parameters. The current FEC Framework has a strong nuance of “forward erasure correction”, rather than “forward error correction”. Single bit-unit errors remaining are not particularly envisioned. Instead, since data are lost in units of blocks for any error, FEC Framework has a function for restoring lost frames. Envisioned for ECP, however, are cases in which protocols are used that do not ensure reliability of

the data link layer. Since an error of 1 bit at ECP header can cause errors during data reconfiguring, a function that can correct header errors as 1 block is provided.

5. ECP Implementation

In the performance assessment testing, the implementation of this paper employed Reed-Solomon codes RS(255, 239) for error correction encoding of data and RS(28, 24) for header error correction encoding, along with an assignment of "1" as FEC Number. With the RS(255, 239) code, 16 symbols of repair data are added for 239 symbols of data. With a capacity for correcting errors in up to 8 symbols out of 255 symbols, this encoding is also used in digital television signals, as well as ITU-T G.709 and G.975. In the implementation of this paper, ADU subdivision was conducted at the same size to the extent possible in order to avoid differences in block length. For example, the first ADU size input from the application is 2000 bytes and the second data size is 2800 bytes in **Figure 3**, where MTU is 1500 bytes. In this case, the first ADU would be halved to 1000 bytes each {1000, 1000} and the second ADU would be halved to 1400 bytes each {1400, 1400}, and then the blocks would be stored in a matrix. The number of blocks would be unchanged even when subdivided according to the maximum MTU size 1500 − 40 (IPv6 header) − 28 (ECP header) = 1432 bytes, and the sent data count of original data would also be unchanged. However, the largest block length among the blocks of stored original data would become the block length of repair data at the generation of repair data. Subdividing by the MTU size would always make the block length of repair data 1432, which would increase the sent data size of the repair data.

6. Performance Assessment and Discussion

6.1. Testing Environment

In order to assess ECP performance, the present study compared the error rates and data flow in the data link layer for the cases of using UDP, FEC Framework, and ECP. Because FEC Framework assumes that the types and parameters used for error correction encoding all agree in advance, the RS(255, 239) code was used for error correction encoding and the protocol used in Simple Reed-Solomon FEC Scheme for FECFRAME of RFC 6865 [12] was adopted. The data link layer used for testing was executed in a simulator implemented with Ethernet specifications that allowed MTU and bit error rate (BER) to be varied as desired. IPv6 was used for the network layer protocol. As shown in **Figure 4**, identical data were readied for sending and receiving in the test, data were sent from the sender, errors were entered at the data link layer based on BER, error correction was conducted at the receiver, and the ratio by which this differed from the data already on hand in bytes was calculated and compared as the error rate. The data sent from the sender with Ethernet header attached were measured and compared under the different methods in terms of data size.

6.2. Error Rates

The testing compared the various methods with BER set at 10^(−6). BER per hop under wired Ethernet is generally not greater than 10^(−9), versus 10^(−6) to 10^(−8) for wireless LAN. For environments that conduct path MTU discovery in advance, however, multiple hops across different data links frequently occur. Thus, the BER was set at 10^(−6) for testing. For ECP and FEC Framework, data encoding was conducted with RS(255, 239) and ECP header error correction with RS(28, 24). The Ethernet MTU used was 1500 bytes, and 6000 bytes was used for Ethernet Jumbo Frame. Results in terms of the sizes of ADU (1024, 2048, 4096, 8192, 16,384, and 32,768 bytes) handed from the application under each method sent 10,000 times each are shown in **Figure 5** and **Figure 6** for Ethernet MTUs of 1500 and 6000 bytes, respectively.

For both **Figure 5** and **Figure 6**, the errors are close to 0% for ADU sizes up to 2048 bytes for FEC Framework and ECP. Beyond 8192 bytes, however, correction under FEC Framework mostly failed to function and left errors comparable to UDP. Since an error of 1 bit in Ethernet destroys the frame, data loss is proportional to the size of MTU. Since UDP and FEC Framework delegate data division to IP, data reconstruction takes place under IP. Specifications call for destruction, however, unless all subdivided data are ready at reconstruction. Thus, an error of 1 bit becomes a loss of 1 ADU. Since ECP and FEC Framework employed RS(255, 239), a loss of up to 8 blocks out of 255 blocks could be restored. When ADU size increased, however, the probability of even 1 bit of error in 1 block declined for ECP when block length was no greater than MTU, compared to

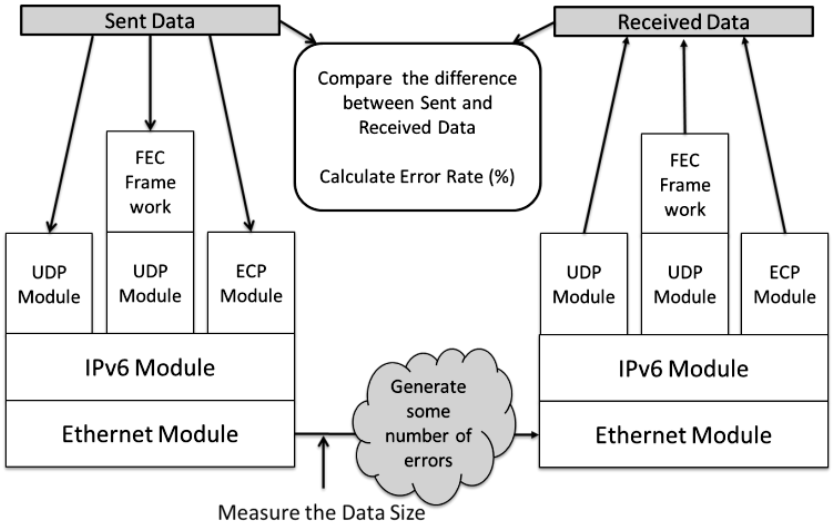

Figure 4. Experimental environment.

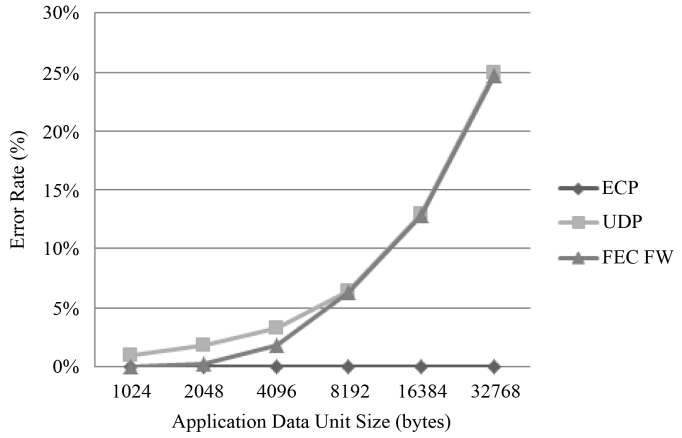

Figure 5. Error rate (Ethernet MTU, 1500 bytes; BER, 1.0 × 10^(−6)).

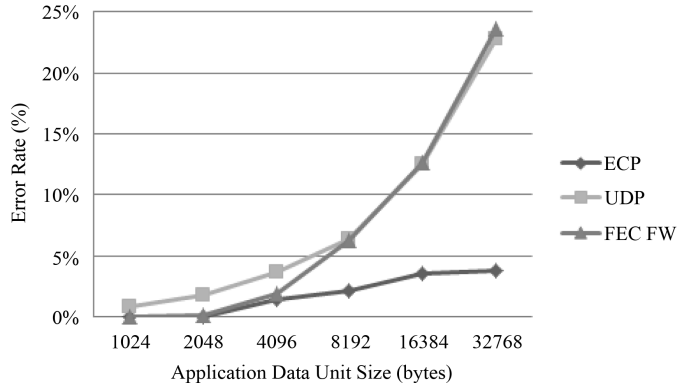

Figure 6. Error rate (Ethernet MTU, 6000 bytes; BER 1.0 × 10^(−6)).

FEC Framework where block length was the ADU size. Consequently, differences emerged in error correction performance. Correction performance between ECP and FEC Framework did not present any difference for ADU sizes no greater than MTU.

6.3. Sent Data Size

The data increase rates, determined by dividing the total sent data size under the respective methods, as measured by data link during testing of error rate measurement, by the total ADU size are shown in **Figure 7** and **Figure 8** for Ethernet MTUs of 1500 and 6000 bytes, respectively.

These test results do not include the sent data by FEC Framework for advance agreement and sent data generated from path MTU discovery conducted in advance by ECP. Since 16 symbols of repair data are generated for 239 symbols under RS(255, 239), the theoretical increase is approximately 7%. However, data measured on Ethernet have Ethernet header and Footer and IP header attached, ECP has one ECP header attached per Ethernet frame unit, FEC Framework has one FEC Framework header and UDP header attached per ADU, and UDP has one UDP header attached per ADU unit. Comparing the frame counts flowing on Ethernet, UDP < ECP—FEC Framework, but ECP header has 28 bytes, which increases the data size by 0.3% to just under 2% more than that with FEC Framework. Increasing the MTU size or the ADU size reduces the difference between ECP and FEC. Since the ADU sizes were fixed in this test, no padding process was carried out at matrix generation under FEC Framework. When ADU sizes differed between input, however, repair data were generated at a size to match the block with the largest block length, which caused data size to increase. For example, a test was conducted in which ADU size was varied from 8192 bytes for an Ethernet MTU of 1500 bytes. In this test, ADU was input 10,000 times, but a variable range x was set as the parameter such that input data between the sizes of $8192 - x$ and $8192 + x$ bytes were handed to ECP and FEC Framework. For example, for $x = 2000$, data took on an input data size randomly between 6192 and 10,191 bytes. The test results are shown in **Figure 9**.

Since sizes were subdivided at no greater than MTU for ECP, the block length for repair data was kept lower than that of FEC Framework. However, since repair data of the same block length as the maximum size among 239 ADUs were generated for FEC Framework, when the variations became large, the data volume flowing through Ethernet increased in comparison to ECP. Error rates in this test did not depend on the variable range and were nearly the same as the case for an ADU size of 8192 bytes in **Figure 4**: 0% for ECP and approximately 6% to 7% for FEC Framework. A similar trend was observed for results with an Ethernet MTU of 6000 bytes.

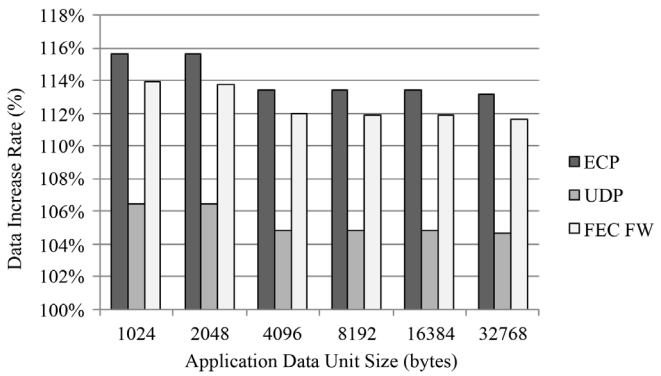

Figure 7. Data increase rate (Ethernet MTU, 1500 bytes).

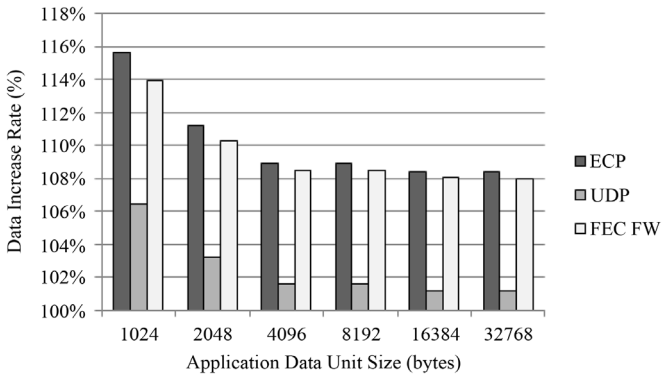

Figure 8. Data increase rate (Ethernet MTU, 6000 bytes).

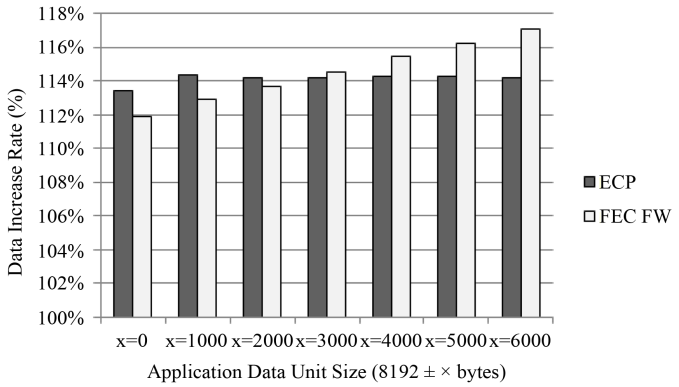

Figure 9. Data increase rate (Ethernet MTU, 1500 bytes; input data size fluctuations).

6.4. Assessment for Data Link Layer with No Error Detection Function

The following three types are defined as models for the data link layer [13].

- Unacknowledged Connectionless Service.
- Acknowledged Connectionless Service.
- Connection-Oriented Service.

Among these, the first type, Unacknowledged Connectionless Service, applies to a high-speed data link layer with relatively few errors, such as a LAN. Special processing is not performed in this type, even if errors occur. Under current Ethernet specifications, however, CRC error detection per frame is not conducted, and entire frames are destroyed when errors are detected. In the future, however, the appearance of Ethernet without an error detection function (error detection performed only on Ethernet frame header) can be envisioned with the emergence of transfer media with fewer errors and the demand for higher speeds to the current data link layers. In actuality, with the addition of the Jumbo Frame function even to today's Ethernet, efforts are underway to expand the MTU upper limit from 1500 bytes to a range of 6000 to 16,000 bytes. The current Ethernet Switch allows the upper limit to be increased beyond 16,000 bytes, but CRC error detection performance drops when around 9000 bytes is exceeded. Thus, the maximum upper limit is currently 16,000 bytes. Taking these conditions into account, it is not unreasonable to assume that application of error detection to just the Ethernet frame header could increase MTU size and advance attempts to improve transfer speeds tremendously. Current TCP and UDP error correction mechanisms, moreover, are simple addition checksums without very high precision, relying as they do on the performance of Ethernet CRC. Thus, disposing of or limiting the error detection function at the data link layer is difficult, but if the current dependence of TCP and UDP on lower layer error detection is inhibiting higher speeds of the data link, then a protocol different from TCP and UDP will be necessary. A test was performed in which Ethernet CRC checking was applied to just the Ethernet header for Ethernet MTUs of 9000 and 16,000 bytes. ADU sizes were varied between 4096, 8192, 16,384, and 32,768 bytes. The results showed no dependence on Ethernet MTU size; therefore, the test results for an Ethernet MTU of 16,000 bytes are shown in **Figure 10**.

Errors can be made to be nearly zero under ECP. For FEC Framework, however, error rates increase with ADU size. This is because, since FEC Framework is implemented on UDP, even when frames contaminated with errors in Ethernet are handed to the UDP, the errors are detected by UDP checksum and then the datagram is destroyed by UDP. Checksum checking can be designated arbitrarily for IPv4, but UDP-based checksum is mandatory for IPv6 in lieu of abandoning the error detection function in IP [8]. Thus, destruction by UDP cannot be avoided. Error correction can be conducted for ECP at a location of an error-contaminated frame in which no errors exist. Thus, error rates can be reduced to lower than when CRC checking is conducted for all data.

7. Future Work

Repair data size and error correction performance of ECP depend greatly on the types and parameters of error correction encoding employed. As such, by providing the function of deciding error correction encoding and pa-

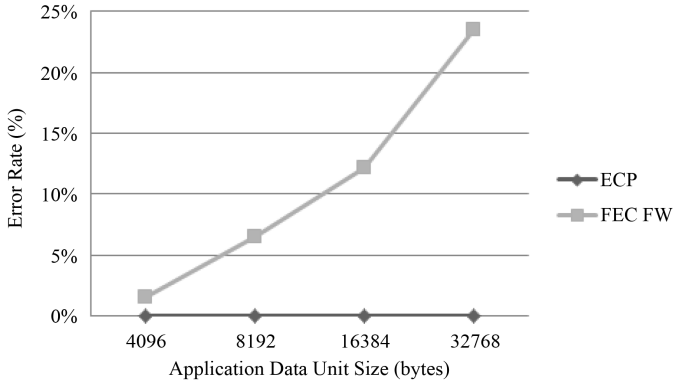

Figure 10. Error rate (Ethernet MTU, 16,000 bytes; BER, 1.0 × 10^(−6); no CRC error detection).

rameters used by ECP on the basis of the input data size and, where possible, transfer error rate, the application developer can engage in development work without considering error correction performance. The specifications presented in this paper did not include the costs of calculation and latency for encoding, decoding, or data division, which must be studied in the final determination of specifications. When cases of error contamination rather than data loss are envisioned for the data link layer, since ECP corrects across multiple blocks, ensuring data reliability requires waiting for the arrival of multiple blocks, which has an adverse impact on real-time communications. Considered as a countermeasure to this was the introduction of a mechanism to enable the detection of the necessity of error correction, by introducing a checksum such as CRC as an error detection mechanism for data contamination of each block; however, the introduction of a checksum causes further enlargement of the header and processing increases for the receiver. In the event of checksum introduction, the 2-byte checksum used for UDP offers low error detection performance. Instead, therefore, something along the lines of CRC32 should be introduced with the institution of a 4-byte field to the header. The necessity of checksum introduction and its algorithm need to be studied further, however. For the testing described here, errors were generated on the basis of BER, but losses in unpredictable bursts have been identified on actual networks. In the future, an ECP implementation involving the introduction of automatic decision-making for error correction encoding and parameters needs to be made on the IPv6 stack for assessment of calculation volume, latency, and performance against error bursts. A separate paper shall describe such assessments.

8. Conclusion

This paper proposed Transport Layer Protocol ECP, which introduces error correction mechanisms, and described performance assessments of the same. ECP holds block sizes for conducting error correction to be no greater than the MTU determined by path MTU discovery, which allows the data volume lost at errors of 1 bit at the data link layer to be suppressed, and attaches appropriately sized repair data. For a data link layer whose error correction function was restricted, ECP revealed error correction performance superior to that of even any error correction technology implemented on UDP.

References

[1] Watson, M., Began, A. and Roca, V. (2011) Forward Error Correction (FEC) Framework. *Request for Comments*, **6363** (Internet Engineering Task Force).

[2] Alteon Networks Extended Frame Sizes for Next Generation Ethernets, a White Paper. http://staff.psc.edu/mathis/MTU/AlteonExtendedFrames_W0601.pdf

[3] Reed, I.S. and Solomon, G. (1960) Polynomial Codes over Certain Finite Fields. *J.SIAM*, **8**, 300-304. http://dx.doi.org/10.1137/0108018

[4] Gallager, R.G. (1962) Low-Density Parity-Check Codes. MIT Press, Cambridge. (Preliminary Version in *IRE Trans on Inf. Theory*, **8**, 21-28.)

[5] Bose, R.C. and Ray-Chaudhuri, D.K. (1960) On a Class of Error-Correcting Binary Group Codes. *Inform. Control*, **3**, 68-79. http://dx.doi.org/10.1016/S0019-9958(60)90287-4

[6] Hocquenghem, A. (1959) Codes correcteursd'erreurs. *Chiffres*, **2**, 147-156.

[7] Begen, A. (2011) Session Description Protocol Elements for the Forward Error Correction (FEC) Framework. *Request for Comments*, **6364** (Internet Engineering Task Force).

[8] Deering, S. and Hinden, R. (1998) Internet Protocol, Version 6 (IPv6) Specification. *Request for Comments*, **2460** (Internet Engineering Task Force).

[9] IEEE 802.3-2002 (1985) IEEE Standard for Information Technology, Telecommunications and Information Exchange between Systems, Local and Metropolitan Area Networks, Specific Requirements Part: 3 Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications Low-Density Parity-Check Codes.

[10] Lacan, J., Roca, V., Peltotalo, J. and Peltotalo, S. (2009) Reed-Solomon Forward Error Correction (FEC) Scheme. *Request for Comments*, **5510** (Internet Engineering Task Force).

[11] McCann, J., Deering, S. and Mogul, J. (1996) Path MTU Discovery for IP Version 6. *Request for Comments*, **1981** (Internet Engineering Task Force).

[12] Roca, V., Cunche, M., Lacan, J., Bouabdallah, A. and Matsuzono, K. (2013) Simple Reed-Solomon Forward Error Correction (FEC) Scheme for FECFRAME. *Request for Comments*, **6865** (Internet Engineering Task Force).

[13] IEEE 802.2. (1998) IEEE Standard for Information Technology, Telecommunications and Information Exchange between Systems, Local and Metropolitan Area Networks, Specific Requirements Part 2: Logical Link Control.

Scientific Research Publishing (SCIRP) is one of the largest Open Access journal publishers. It is currently publishing more than 200 open access, online, peer-reviewed journals covering a wide range of academic disciplines. SCIRP serves the worldwide academic communities and contributes to the progress and application of science with its publication.

Other selected journals from SCIRP are listed as below. Submit your manuscript to us via either submit@scirp.org or Online Submission Portal.