Analysis of the FLUTE Data Carousel

Jani Peltotalo, Sami Peltotalo and Jarmo Harju
Tampere University of Technology
Institute of Communications Engineering
P.O.Box 553, FIN-33101 Tampere, Finland
Email: {jani.peltotalo, sami.peltotalo, jarmo.harju}@tut.fi

***Abstract*— This paper presents the results of performance tests done for File Delivery over Unidirectional Transport (FLUTE) protocol. FLUTE is a protocol used to deliver files over the Internet or unidirectional systems from one or more senders to one or more receivers. Because FLUTE uses unreliable transport protocol, packet losses must be handled at higher layers. This paper shows how FLUTE manages to recover from packet losses using data carousel.**

I. Introduction to FLUTE

File Delivery over Unidirectional Transport (FLUTE) [1] is a protocol used to deliver files (e.g. documents, images, video/audio clips) over the Internet or unidirectional systems from one or more senders to one or more receivers. FLUTE can be used with both multicast and unicast User Datagram Protocol (UDP) delivery, but it is particularly suited to multicast networks. Both multicast models, Any-Source Multicast (ASM) and Source-Specific Multicast (SSM), can be used with FLUTE. FLUTE supports also both IP versions (IPv4 and IPv6), because there are no IP version specific parts in the FLUTE header.

FLUTE builds on Asynchronous Layered Coding (ALC) Protocol Instantiation [2] of the Layered Coding Transport (LCT) Building Block [3]. LCT provides transport level support for reliable content delivery and stream delivery protocols. ALC combines the LCT building block, a Congestion Control (CC) building block and a Forward Error Correction (FEC) building block to provide congestion controlled reliable asynchronous delivery.

A FLUTE session (i.e. an ALC/LCT session) consists of one or more ALC/LCT channels defined by the combination of a sender and an address associated with the channel by the sender. A receiver joins a channel to start receiving the data packets sent to the channel by the sender, and the receiver leaves the channel to stop receiving data packets from the channel.

Figure 1 shows how a file is splitted into FLUTE packets. Assume that the user wants to send a file, which is the transport object for the FLUTE protocol. Based on the transport object length, the Encoding Symbol Length and the Maximum Source Block Length a FLUTE sender calculates the source block structure, i.e. the number of source blocks and their lengths. A Maximum Source Block Length is a maximum length of a source block that FLUTE's algorithm for calculating the length of the source blocks gives for a source block. This algorithm generates at most two different lengths for the source blocks, and the lengths are as close to each other as possible. The user of the FLUTE sender configures both the Encoding Symbol Length and the Maximum Source Block Length.

Each source block is then fragmented into source symbols according to the Encoding Symbol Length. If FEC is used, then parity symbol(s) are calculated based on the source symbols. Source symbols and parity symbols together comprise encoding symbols for the FLUTE protocol. Then a FLUTE packet is constructed from a FLUTE header and an encoding symbol. Finally the FLUTE packet is ready for UDP/IP delivery.

The sender communicates the transport object length, the Encoding Symbol Length and the Maximum Source Block Length to the receiver(s) either in the FLUTE header or using a special transport object, named File Delivery Table (FDT). Thus the FLUTE receiver(s) are able to calculate the source block structure in advance of receiving a file.

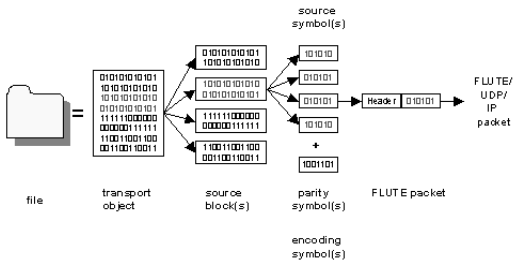

Fig. 1. Building up a FLUTE packet

The use of CC and FEC building blocks with FLUTE is optional. By default no CC is used and the FEC code is Compact No-Code FEC [4], which means that there is no actual FEC encoding or decoding, and encoding symbols contain only the source symbols. But when using for example Reed-Solomon FEC, encoding symbols contain also parity symbols. Reed-Solomon is a typical block (n, k) FEC code, where k is the number of source symbols per block and n is the number of encoding symbols per block. To be able to decode a source block, the decoder needs to have k encoding symbols for the block; so all encoding symbols are equal.

As mentioned earlier, a FLUTE sender calculates the source block structure, i.e. the number of source blocks and their lengths. The Source Block Length is the length in units of source symbols of the source block, i.e. it is also the value of k for FEC codes. The value of n is computed for example using "n-algorithm" specified in [5].

II. Test Setup

Simulations defined in this paper were done by using MAD-FLUTE [7] (version 1.0), which is one of the publicly available implementations of the FLUTE protocol. MAD-FLUTE is available for both Windows and Linux. It is IPv4 and IPv6 capable and it supports both unicast and multicast UDP delivery. The simulations were done in Linux (kernel version 2.6.5) using IPv4 and one ALC/LCT channel (multicast group).

In the simulations the FLUTE sender transmitted data, i.e. File Delivery Table and one file, in a carousel (encoding symbols were sent sequentially). Two types of carousels were used, data carousel and FEC data carousel. Data carousel used FLUTE with Compact No-Code FEC, and FEC data carousel used FLUTE with Reed-Solomon FEC, based on Vandermonde matrices [6].

The size of the file was 5488640B (mp3 file). If not otherwise mentioned, the used Encoding Symbol Length was 1428B with Compact No-Code FEC and 1424B with Reed-Solomon FEC, so the file consists of 3844 and 3855 source symbols respectively. The used Encoding Symbol Length was maximum length for the encoding symbol so that the IP packet length did not exceed the Ethernet link's Maximum Transmission Unit (MTU) 1500B. FLUTE header is four bytes longer with Reed-Solomon FEC compared to Compact No-Code FEC.

The intention was to study how FLUTE manages to recover from packet losses. In this paper we define the performance of the FLUTE as the number of loops the FLUTE sender has to transmit so that the FLUTE receiver gets the whole file. We assume that a single receiver can represent the behaviour of all receivers, which is naturally not the case with the Internet. But the assumption is much closer to true in environments, where there is only one hop between the sender and the receivers, which is the case for example in DVB-H.

Both the FLUTE sender and the FLUTE receiver were running on the same machine to avoid uncontrolled packet loss in the network. Instead, uniformly distributed packet loss was generated using a probability p to drop a packet. If p is quite small (under 0,1) an average packet loss burst size is close to one. In real network, error bursts (i.e. a group of packet losses) occur, and for this purpose error bursts were generated by using two probabilities, p_1 and p_2, for packet losses. p_1 was used to make a packet loss and p_2 to make a packet loss after a packet loss, and p_2 was naturally higher than p_1.

The average packet loss with error bursts can be calculated by using two state Markov chain, where state one is state when earlier packet was not lost and state two is state when earlier packet was lost. Equations $P_1 p_{12} = P_2 p_{21}$ (balance equation) and $P_1 + P_2 = 1$ with following transition probabilities:

$$\begin{pmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{pmatrix} = \begin{pmatrix} 1-p_1 & p_1 \\ 1-p_2 & p_2 \end{pmatrix}$$

gives the probabilities P_1 and P_2 to find the system in state one and in state two respectively. Then the average packet loss is $P_1 p_{12} + P_2 p_{22}$, and the average error burst size is $1/(1-p_2)$. For example $p_1 = 0,01$ and $p_2 = 0,80$, and also $p_1 = 0,02$ and $p_2 = 0,60$, will generate 4,76% average packet loss with average error burst size 5 and 2,5 packets respectively.

III. Data Carousel versus FEC Data Carousel

Because FLUTE uses unreliable transport protocol, packet losses must be handled at higher layers. Use of data carousel is one option for this purpose, i.e. missing packets are tried to be caught in the next loop(s). Better results are received by using FEC data carousel, which includes parity data to recover from packet losses, but depending on the amount of parity data and missing packets next loop(s) might still be needed.

In all data carousel simulations we studied how many loops were needed to receive the whole file with different amount of FEC data. We also did some mathematical analysis of data carousel.

Table I describes symbols, which are used further in other tables.

TABLE I

Definition of symbols

Loss [%]	Average packet loss percentage
Avg	Average number of loops needed to receive the whole file
Min	Minimum number of loops needed to receive the whole file
Max	Maximum number of loops needed to receive the whole file
Exp	Number of experiments
l	Encoding Symbol Length
L	Maximum Source Block Length
RS [%]	Amount of the parity data (FEC) compared to the source data, for example 100% means that there is equal amount of source data and parity data

A. Simulation of Data Carousel under Uniformly Distributed Errors

In this test Compact No-Code FEC (also called Null FEC) was used with different average packet loss percentages. The sender generated artificial packet loss, which did not contain error bursts. With Compact No-Code FEC the Source Block Length does not affect to the performance or to the encoding and decoding times, because there is no actual FEC encoding and decoding. The used Maximum Source Block Length was 10.

The number of loops needed to receive the whole file with data carousel under uniformly distributed errors is shown in Table II. It should be noted that the size of the file (the number of packets forming the file) affects to these values.

TABLE II

Data carousel under uniformly distributed errors

Loss [%]	Avg	Min	Max	Exp
50	13,33	10	22	1000
25	6,89	5	12	1000
10	4,34	3	7	1000
5	3,40	3	5	1000
1	2,33	2	4	1000
0,1	1,99	1	3	1000

From Table II we can notice that performance gets quite poor already with low average packet loss. For example with 10% average packet loss it can take seven loops to receive the whole file.

B. Mathematical Analysis of Data Carousel under Uniformly Distributed Errors

In this test we analysed the data carousel mathematically under uniformly distributed errors. We tried to mathematically solve out how many loops are averagely needed to receive the whole file with different packet loss ratios. The analysis was done with the help of Mathematica.

Equation 1 gives the probability to receive x new packets for each loop, where s is the number of sent packets per loop (constant), l is the number of lost packets per loop (constant), and m is the number of missing packets at the beginning of the loop.

$$P(x,m) = \frac{\binom{m}{x}\binom{s-m}{s-l-x}}{\binom{s}{s-l}} \quad (1)$$

For the expectation value of the number of new packets, which are received at loop no. i we have

$$x(i) = \sum_{\xi=0}^{m} \xi P(\xi, m) \quad (2)$$

The following algorithm utilises Equations 1 and 2 to give the number of loops averagely needed to receive the whole file. To handle the last missing packet in a realistic way, a Monte Carlo simulation part is added to the algorithm.

```
i = 2, m = l
while (1) {
  if (m == 1) {
    if (l/s < random(0,100)/100) {
      result = i, break
    }
    else {i++, continue}
  }
  m -= round(x(i))
  if (m > 0) {i++}
  else {result = i, break}
}
```

The algorithm was run a thousand times for each packet loss ratio, so that the effect of the random function in the algorithm was taken into account. Table III shows the number of loops averagely needed to receive the whole file. In the table "Math" means the mathematical analysis and "Simul" means the simulation done in Section III-A. The algorithm gives quite similar results, for different packet loss ratios, than achieved in the simulation, so it could be used for other packet loss ratios too.

TABLE III
MATHEMATICAL ANALYSIS COMPARED TO SIMULATION OF DATA CAROUSEL UNDER UNIFORMLY DISTRIBUTED ERRORS

Loss [%]	Math	Simul
50	12,94	13,33
25	7,32	6,89
10	4,00	4,34
5	3,00	3,40
1	2,00	2,33
0,1	2,00	1,99

C. Simulation of FEC Data Carousel under Uniformly Distributed Errors

In this test Reed-Solomon FEC was used with different average packet loss percentages. The sender generated artificial packet loss, which did not contain error bursts. With different average packet loss percentages the amount of FEC data was increased step by step (5, 10, 25, 50, 100, 150 and 200 percents) until the whole file was received always at the first loop. The Maximum Source Block Length was set to 85 so that it was possible to generate 200% FEC data, because FEC encoder's and decoder's maximum value for n (the number of encoding symbols per block) was 255.

Table IV shows the number of loops needed to receive the whole file with different average packet loss and FEC data percentages. The effect of adding even a small amount of FEC data into the carousel is remarkable. When considering the performance of the FEC data carousel, it should be noted that the overhead data in one loop increases when the amount of FEC data increases, so the total amount of data averagely needed to transmit describes better the performance of the FEC data carousel.

For example, adding of 10% Reed-Solomon FEC data with 10% average packet loss, more than halved (4,34 $\rightarrow$ 2) the average number of loops needed to receive the whole file compared to Compact No-Code FEC, but the average amount of transmitted data ($Avg\ Data = (1 + RS[\%]/100) * Avg$) does not halve (2,20 with 10% Reed-Solomon FEC data and 4,34 with Compact No-Code FEC).

TABLE IV
FEC DATA CAROUSEL UNDER UNIFORMLY DISTRIBUTED ERRORS

Loss [%]	RS [%]	Avg	Min	Max	Avg Data	Exp
0,1	5	1,00	1	1	1,05	1000
1	5	1,09	1	2	1,14	1000
1	10	1,00	1	1	1,10	1000
5	5	2,00	2	3	2,10	1000
5	10	1,87	1	2	2,06	1000
5	25	1,00	1	1	1,25	1000
10	5	2,09	2	3	2,19	1000
10	10	2,00	2	2	2,20	1000
10	25	1,03	1	2	1,29	1000
10	50	1,00	1	1	1,50	1000
25	5	3,44	3	4	3,61	1000
25	10	3,00	2	4	3,30	1000
25	25	2,00	2	2	2,50	1000
25	50	1,49	1	2	2,24	1000
25	100	1,00	1	1	2,00	1000
50	5	6,45	5	9	6,77	1000
50	10	5,10	4	7	5,61	1000
50	25	3,35	3	4	4,19	1000
50	50	2,48	2	3	3,72	1000
50	100	2,00	2	2	4,00	1000
50	150	1,08	1	2	2,70	1000
50	200	1,00	1	1	3,00	1000

D. Simulation of Data Carousel with Error Bursts

The reason for this test was to study how error bursts affect to the performance of data carousel, and also to figure out the effect of the use of different Encoding Symbol Lengths. In this test Compact No-Code FEC was used. The sender generated artificial packet loss, which contained error bursts. Packet loss probabilities p_1 and p_2 (see Section II) were 1% and 80% respectively. With these values an average packet loss is 4,76% (an average error burst size is 5 packets), so we can compare this test to uniformly distributed errors case with 5% average packet loss. The used Maximum Source Block Length was 10.

The number of loops needed to receive the whole file is shown in Table V. First we can notice that all values are better

or equal compared to uniformly distributed errors case with Encoding Symbol Length 1428B (Avg = 3,4, Min = 3, Max = 5 in Table II). This might be due to the fact that with error bursts the likelihood to loose the same packet at the next loop is smaller, because only the place of the first packet of the error burst is random and the other packet losses come after that packet. With uniformly distributed errors, packet losses are distributed more smoothly. Another observation is that when decreasing the Encoding Symbol Length the performance also gets worse, as expected.

TABLE V
DATA CAROUSEL WITH ERROR BURSTS

l	Avg	Min	Max	Exp
1428	3,17	2	5	1000
714	3,35	2	6	1000
357	3,62	3	6	1000

E. Simulation of FEC Data Carousel with Error Bursts

The reason for this test was to study how error bursts affect to the performance of FEC data carousel. The focus was mainly to figure out how different Source Block Lengths give different protections against error bursts. In this test Reed-Solomon FEC was used. The amount of the FEC data was 10% and the sender generated artificial packet loss, which contained error bursts.

In the first test case, packet loss probabilities p_1 and p_2 were 1% and 80% respectively, and in the second test case p_1 and p_2 were 2% and 60% respectively. With these values an average packet loss is 4,76% (an average error burst size is 5 packets in the first test case and 2,5 packets in the second test case), so we can compare these test cases to uniformly distributed errors case with 5% average packet loss.

The number of loops needed to receive the whole file is shown in Table VI. First we can notice that all values in both test cases are worse or equal compared to uniformly distributed errors case with Maximum Source Block Length 85 (Avg = 1,87, Min = 1, Max = 2 in Table IV). But by increasing the Maximum Source Block Length it is possible to get better protection against error bursts. With the second test case we can notice that when the error burst size is small even Source Block Length 85 has quite good performance.

TABLE VI
FEC DATA CAROUSEL WITH ERROR BURSTS

Avg Burst	L	Avg	Min	Max	Exp
5	85	2,12	2	4	1000
5	170	1,99	1	3	1000
5	230	1,92	1	3	1000
2,5	85	2,00	1	3	1000
2,5	170	1,90	1	2	1000
2,5	230	1,68	1	2	1000

Theoretically if the studied average size error burst happens once for a block, all used Maximum Source Block Lengths should have enough protection against it. With Maximum Source Block Length 85 (with 10% Reed-Solomon FEC data) there are eight redundant symbols per source block, so a maximum acceptable error burst size per source block is thus eight. With Maximum Source Block Length 170 and 230 there are 16 and 22 redundant symbols respectively, so the maximum acceptable error burst sizes are also higher.

According to our results there must have been several error bursts within one block, or the error burst has been longer than the maximum acceptable error burst size, because the whole file was not always received at the first loop.

IV. CONCLUSIONS

FLUTE has good performance when some amount of parity data is added into the data carousel to minimize the number of loops that are needed to successfully receive the file(s). For example, simulations showed that it is possible to protect against 1% average packet loss by adding 10% Reed-Solomon parity data. Two to four loops are needed to recover missing packets in the same case without the parity data. With higher packet loss ratios it is even more beneficial to use parity data. It should be also noted that other FEC techniques might perform even better compared to Reed-Solomon.

The used Encoding Symbol Length should be the maximum length for the encoding symbol carried in the FLUTE packet, so that IP packet length do not exceed link's MTU. Also large Source Block Length gives better protection against packet loss, when FEC is used, but with cost of increased encoding and decoding times.

Because FLUTE uses unidirectional transport the FLUTE sender does not know anything about the receiving status of the FLUTE receiver(s). The results presented in this paper gives some hints how to use the FLUTE sender so that the FLUTE receiver(s) gets the file(s) with optimal amount of data transmitted in a network.

Another option to the carousel (with or without parity data) type of packet loss recovery is to use some kind of point-to-point or point-to-multipoint file repair technique, which is utilized when packets are still missing after the FLUTE sender has stopped sending the file. If some file repair technique is supported, the FLUTE sender could carousel the file for example the average number of loops presented in this paper. In other cases it might be best to use the worst case values to enable reliable delivery.

REFERENCES

[1] Paila, T., Luby, M., Lehtonen, R., Roca, V. and R. Walsh: FLUTE - File Delivery over Unidirectional Transport. RFC 3926, October 2004

[2] Luby, M., Gemmell, J., Vicisano, L., Rizzo, L. and J. Crowcroft: Asynchronous Layered Coding (ALC) Protocol Instantiation. RFC 3450, December 2002

[3] Luby, M., Gemmell, J., Vicisano, L., Rizzo, L., Handley, M. and J. Crowcroft: Layered Coding Transport (LCT) Building Block. RFC 3451, December 2002

[4] Luby, M. and L. Vicisano: Compact Forward Error Correction (FEC) Schemes. RFC 3695, February 2004

[5] Peltotalo, J., Peltotalo, S. and V. Roca: Simple XOR, Reed-Solomon, and Parity Check Matrix-based FEC Schemes. IETF, draft-peltotalo-rmt-bb-fec-supp-xor-pcm-rs-00.txt, June 2004

[6] Rizzo, L.: Effective Erasure Codes for Reliable Computer Communication Protocols. ACM SIGCOMM Computer Communication Review Vol.27, No.2, pp.24-36, April 1997

[7] MAD/TUT project's home page, April 2005 http://www.atm.tut.fi/mad