# AL-FEC Integration

This section describes the AL-FEC operation for the demo setup. At first, the procedure to activate the AL-FEC feature at both sender and receiver side is presented. Secondly, the AL-FEC operation is explained with the help of network captures. In a third place, implementation issues that cause instability issues in the demo setup are identified and alternative solution to simulate the AL-FEC operation is introduced. To finalize the obtained result with the simulated setup are presented.

# 1. Enabling AL-FEC in ATSC 3.0 software

The following are the command line used to start sender and receiver

Sender:

$> ./flute\_sender -S -U -r:20000 -B:/media/Elysium\_1\_0 -f:/media/Elysium\_1\_0/efdt\_Video.xml -m:10.0.2.15 -p:4001 -t:1 -y:/media/Elysium\_1\_0/FluteInput\_Video.txt -Y:1

Receiver

$> ./flute -A -U -B:DASH\_Content1 -m:10.0.2.15 -s:10.0.2.15 -p:4001 -t:1 -Y:1

For enabling AL-FEC, the two following options need to be added to the command line

**-x:int** FEC Encoding [0 = Null, 1 = Simple XOR, 2 = Reed-Solomon

(old I-D) 3 = Reed-Solomon (new I-D), default: 0

**-X:int**  FEC ratio percent, default: 50

Precisely, FEC encoding shall be set to Reed-Solomon (-x: 2) and the FEC ratio percent set to the desire level of protection. For example if desire is to protect the streaming to an average of 10% of packet losses then FEC ratio shall be at least 10

(-X: 10)

In addition, the provided EFDT file, shall include the following FDT-Parameters attributes:

FEC-OTI-FEC-Encoding-ID="129"

FEC-OTI-FEC-Instance-ID="0"

FEC-OTI-Max-Number-of-Encoding-Symbols="1024"

# 2. AL-FEC in operation

When streaming using ROUTE protocol, a packet with **Transmission Object Identifier** **(TOI)** field set to 0 is transmitted at the beginning of each segment. This packet contains the EFDT information in XML format. Such packets are shown as packet #1 and #3 in Figure 1. The AL-FEC setting can be read then by the receiver side, also displayed in the same figure.

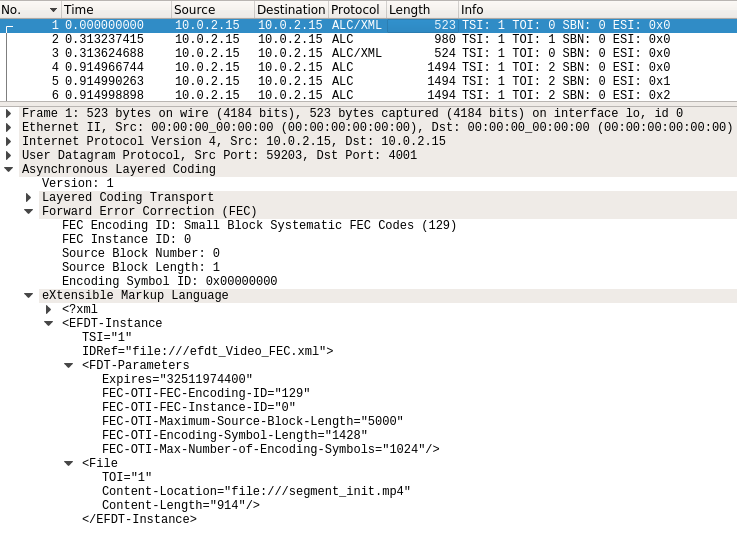


Figure 1. EFDT Information with AL-FEC parameters

With such setting 801 packets are required for transmitting the segment 1 whose size is 1.142.553 bytes. Each symbol transport as maximum 1428 bytes. The calculation is a s follows:

Segment Size / Encoding Symbol Length = 1142553 / 1428 = 800,1

rounding up to 801 Symbols

Looking at packet 4 in the network capture in Figure 2, the **Source Block Length** field tells the receiver side the number of source symbols required to reconstruct the current segment, here 801. The **Encoding Symbol ID (ESI)** field tells the receiver the sequence number of the packet inside a segment. In this particular case, this is the first packet of the segment and so the ESI is set to 0.

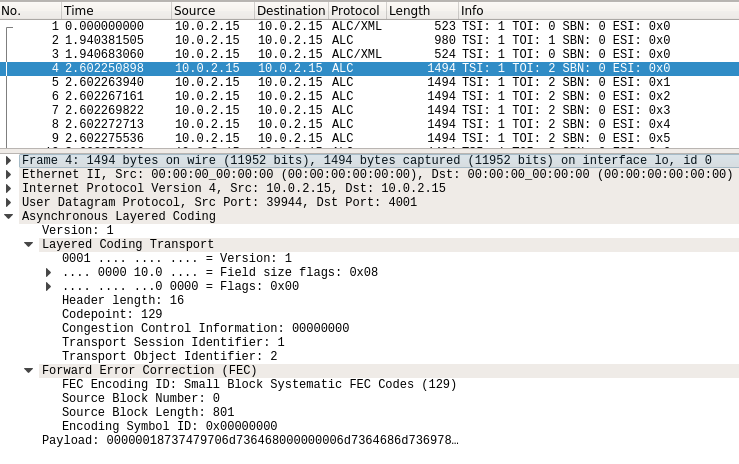


Figure 2. **Source Block Length** and **Encoding Symbol ID (ESI)**

The Segment1.m4s start has started at packet #4 in our capture, see Figure 2. Consequently, the last source symbol shall be at packet #804 with **ESI** set to 0x320 and with the remaining payload of 153 bytes as is confirmed in the Figure 3.

The payload of the last source symbol would be for our example calculated as follows:

* Accumulated size of complete source symbols with a payload of 1428 = 800 x 1428 = 1142400
* Size of the segment - complete source symbols with a payload of 1428 = 1142553 - 1142400
* 1 remaining source symbol with payload of 153 bytes

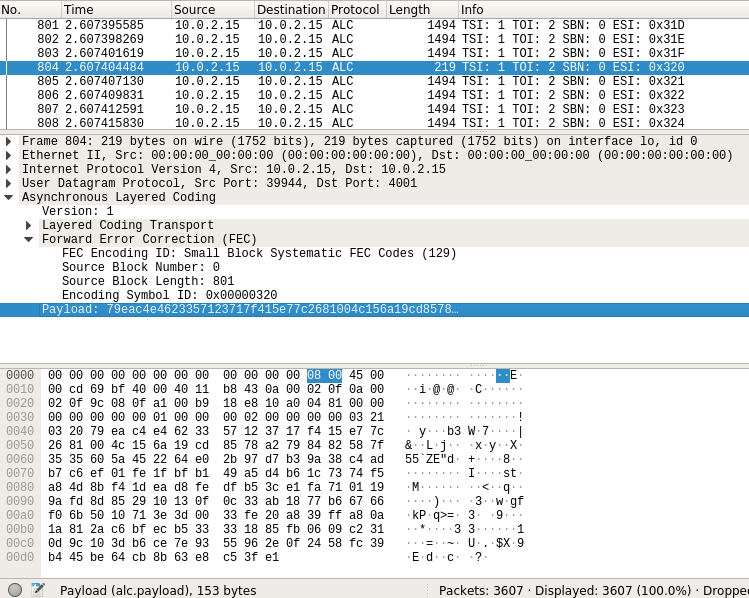


Figure 3. Last Source symbol of a segment

Then, from packet #805 onwards repair symbols shall be transmitted. Taking as example a AL-FEC rate of 10% is expected to have then 80 repair symbols. This mean that the last repair symbol shall be at #884 (**ESI**:0x370), what is confirmed in Figure 4.

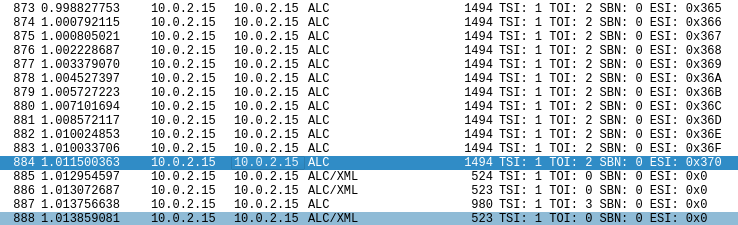


Figure 4. Last repair symbol of a segment and start of a new segment

# 3. Issues with the AL-FEC Implementation

Without AL-FEC protection, any packet loss of a streaming content cause that a complete segment gets lost. Then by setting AL-FEC with a code rate higher that the expected packet loss rate most of the segment shall be able to be recovered. However, the production of repair symbols at the sender side as well as the decoding process at the receiver side with the ATSC 3.0 software being used, creates instabilities in the system.

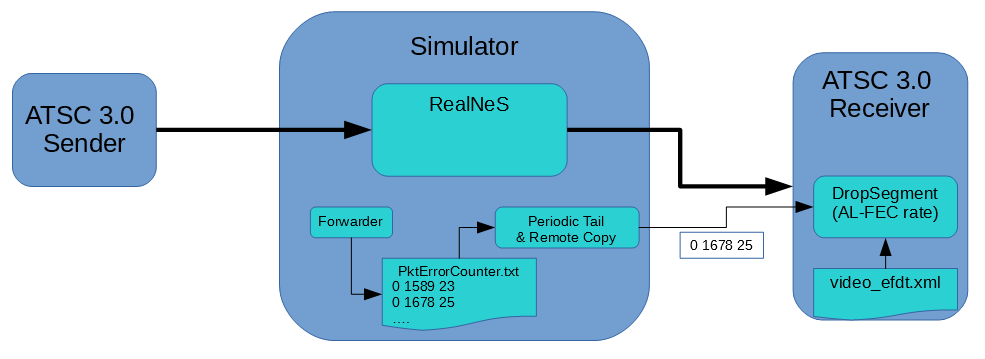
At the sender side, source symbols are simple loaded from disc/memory and packetized to be delivered to the receiver. However, repair symbols need to be computed based on the source symbols. The additional computation for the creation of repair symbols produces time artifacts during the transmission of packets belonging to the same segment. Also, this additional overhead variates from segment to segment as the number of repair symbols to be created depends of the segment size.

Likewise, on the receiver side, the AL-FEC decoder need to wait until a given number of packets has arrived, i.e. the number of source symbols, to reconstruct a segment. Here, two different situations can be distinguished. One case would be that none of the source symbol get lost and consequently no extra computation is required. On the other hand, if any number of source symbols are lost, the receiver initiates an additional procedure to compute the missing source symbols. The extra effort to recover a segment may variate due to one or more of the followings: the segment size itself, the number of symbols being lost and whether those lost symbols correspond to source symbols or repair symbols.

For such reasons, the AL-FEC implementation need to be carefully reviewed and tested. As alternative and to avoid the instabilities introduced by the AL-FEC implementation, the AL-FEC operation has being simulated. The simulated approach is presented in the next section.

# 4. AL-FEC Simulation

The diagram in figure 5 shows how the AL-FEC has being simulated in the demo setup.

Figure 5. AL-FEC simulation

To achieve a stable demo setup. ATSC 3.0 sender and receiver operate without AL-FEC, meaning that extra repair packets are neither produced nor sent. At the simulator machine the **Forwarder** submodule which is attached to the RealNeS simulator keeps track of drop packets by writing the packet error counters to a file (pktErrorCounter.txt). Simultaneously another process (**PeriodicCopy.py**) takes care of tailing periodically this file, about every 1 us, and if any update, transferring the new line with the latest packet counters to the receiver machine. At the receiver machine, the **DropSegment.py** process uses as a input the E-FDT XML file which contains the size of every segment in bytes. With this information on hand, it computes the number of packets that are required per segment and how many packet losses can be tolerated for a targeted AL-FEC protection. Then it checks if the number of packets lost per segment has exceeded the number of repair symbols as for the targeted AL-FEC protection and if so, then remove the segment from the reception folder.

For verification purposes, the following test has been implemented. Eleven (11) intervals were defined. Each of the time intervals has a start and end point determined by the packet number, and also a packet error rate. The values for packet error rate are around the targeted AL-FEC protection (15%), as if the error rate is quite below then the segments can be recovered without problem and in the opposite case if the packet error rate is too high over the AL-FEC protection there will not be chance to recover the segments. In the absence of RealNeS simulator, the **Forwarder** program has being replaced by the additional program **pktCounterGenerator.c** which generate the packet error counter values following then the test definition in table 1.

Table.1 Test definition for verification of drop packet process

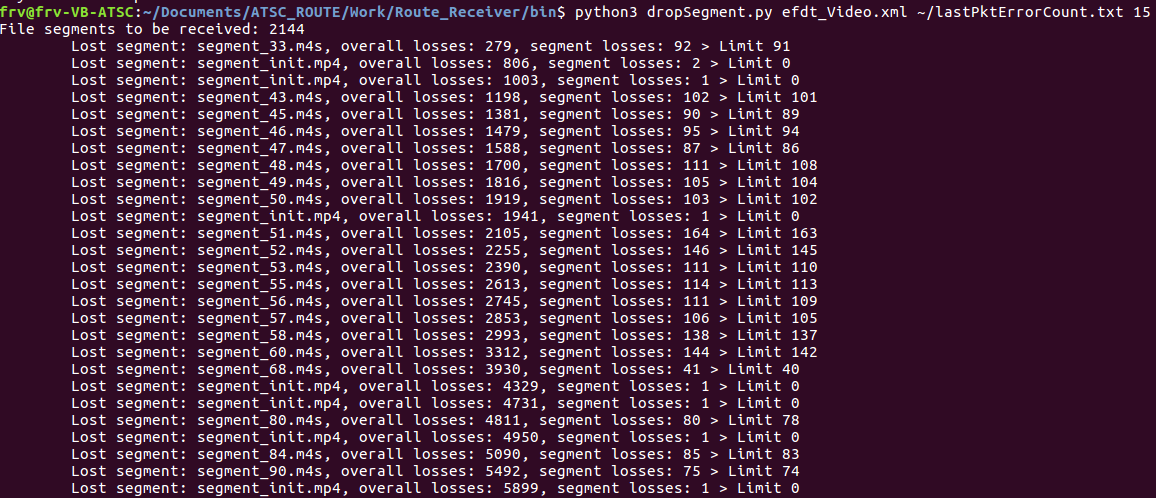
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **AL-FEC (15%)** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| **start** | 0 | 20001 | 25001 | 30001 | 35001 | 40001 | 42001 | 45001 | 55001 | 65001 | 75001 |
| **end** | 20000 | 25000 | 30000 | 35000 | 40000 | 42000 | 45000 | 55000 | 65000 | 75000 | -- |
| **Packet error rate (%)** | 0 | 13 | 14 | 15 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |

Additionally, the information in the EFDT XML file is used at the **DropSegment.py** script, to have a look up table containing the number of packets per segment, the overall number of accumulated packet as well as the number or repair symbols for the targeted AL-FEC. A snapshot of this table is presented in table 2. With the help of the table is easy to identify the segment that shall be drop as per the received counters (total packet counter and packer error counter).

Table.2 Number of packets per segment, accumulated packets and repair symbols



As a result, it is expected to see a transmission with no segment being lost before the packet number 20000 or according to Table 2, before the **segment\_26.m4s**. Then gradually more and more packet shall be lost and eventually segments shall be marked as lost and consequently be removed from the reception folder. For the interval 4 and 5 in Table 1, where the error packet rate is greater or equal 15%, it is expected to see that almost every segment gets lost. This correspond to segments between **segment\_46.m4** and **segment\_54.m4s**

 Figure 6. Segment being marked as dropped

# 5. Open Issues

For certain segment is has been observed more than one EFDT packet per segment, look at packet #805 in Figure 7. After all packets of the **segment\_1.m4s** has being transmitted, again a packet containing the EFDT is present. However, this does not occur for every segment and it seems to have relation with the size of the segment but not yet confirmed. The occurrence of this additional packets breaks the formula how the packets per segment is being calculated to generate the table 2. over time the additional packet might get accumulated and produce that the wrong segment is dropped. Still as segments are in average 1000 packets long and the video being used in the demo setup is form by nearly 1000 segments, the risk of dropping a wrong segment due to these additional packets is virtually zero.

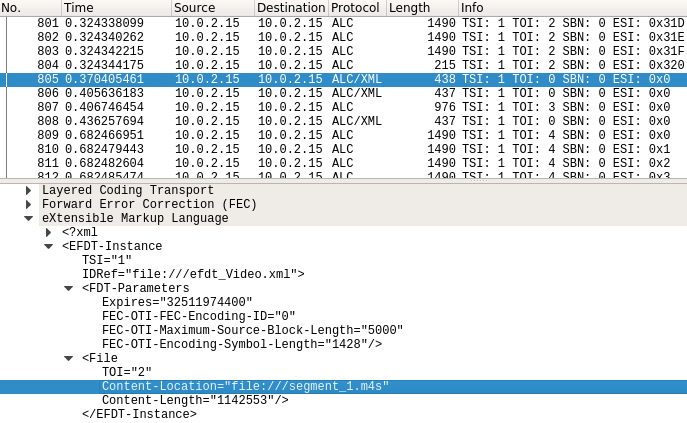


Figure 7. Retransmission of EFDT packet for segment\_1.m4s