Group 13
Raptor Codes

project: Introduction to Internet and Security

Departement Mathematik und Informatik

University of basel

Cyrill Imahorn, Ken Walzer

July 18, 2021

Contents

1 Motivation

In BACnet there are a lot of different communication methods. There are communications over an error susceptible medium where packages get lost, therefore there is a huge interest to have methods which can fix that. There is a solution which is called fountain code. In practice fountain codes are erasure codes, which means these are codes which can correct bit (or packet in practice) losses. It produces limitless encoded symbols of a given set of source symbols, which can be decoded if the receiver got enough encoded symbols.
This has the ideal property for our problem. Our project is called Raptor Codes, which is a special case of fountain codes. We want to try to write such code by our self. Therefore we didn't use any libraries except of such as standard ones like numpy but our definitions are oriented to [1].

2 Raptor Codes

In this section, we want to define Raptor codes. Before doing that, we will define Fountain codes and Luby transform codes (LT codes) which we will use in order to define the raptor codes.

2.1 Terminology

As described in motivation, we want to encode a bitstring. Therefore we introduce some convention we will use from now on. Let *data* be a message to send. For every encoding we will consider a sub-bitstring from *data* with length of k, which will be our blocksize. (we used k =32 in source code) Since bits are either 0 or 1, we can identify a bit as an element of the field $\mathbb{F}_2$. Therefore we will write $x = (x_1, \dots x_k) \in \mathbb{F}_2^k$ for our sub-bitstring. We call x a tuple of source symbol. Further, we write $+$ and $\cdot$ for the addition and multiplication of $\mathbb{F}_2$, hence we have for example $1 + 1 = 0$.

Remark 1. *One can see that $+$ correspond to the XOR and $\cdot$ to AND.*

In general x_i can be an element of $\mathbb{F}_2^n$, hence $x \in F_2^{n \cdot k}$. But in this case since we can do every operation in $\mathbb{F}_2^{k \cdot n}$ coordinate wise, we can reduce this to $\mathbb{F}_2^k$.

2.2 Fountain Codes

In coding theory, a binary linear code is defined as image of a homomorphism from $\mathbb{F}_2^k$ to $\mathbb{F}_2^n$, where $k \leq n$. In that sense, one can define the fountain code as following.

Definition 1 (Fountain codes)**.**
Let k be a natural number. We call

$$\mathcal{F} = \{(L, N) \in Hom(\mathbb{F}_2^k, \mathbb{F}_2^N) \times \mathbb{N} \mid rank(L) = k\}$$

the set of fountain codes.

Remark 2. *We defined here the code with the homomorphism instead of its image since we won't need the image. We will also write it as $\{(A, N) \in Mat_{N \times k} \times \mathbb{N} | rank(A) = k\}$ since we can identify a homomorphism by its matrix. Further, one can see that $N \geq k$.*

Compared to other codes the fountain code has no fix homomorphism (Therefore it is defined as set of homomorphisms.). For every transmission this matrix will be generated randomly and is defined over the property such that the rank must be k, which is sufficient for decoding. This

randomness is generated by a probability space. Therefore we need to define a probability space (in particular a probability measure) for each fountain code we are writing.
We now want to show how we use such code in our algorithms. We started to write an general encoding and decoding function, since they don't differ much between different fountain codes. (where the probability space was left as black box)
The tuple $(A, \mathbb{N}) \in \mathcal{F}$ is generated over encoding and decoding. Let $(\mathbb{F}_2^k, \mathcal{P}(\mathbb{F}_2^k), P)$, be a probability space where $\mathcal{P}(\mathbb{F}_2^k)$ is the power set of $\mathbb{F}_2^k$.

2.3 Encoding

We get a tuple $x = (x_1, \ldots, x_k) \in \mathbb{F}_2^k$ of source symbols as defined above. Now for every $i \in \mathbb{N}$ we generate a sample $A_i = (a_1, \ldots a_k) \in \mathbb{F}_2^k$ with respect to P. Then we construct

$$y_i = \langle A_i, x \rangle = \sum_{j=1}^{k} a_j \cdot x_j,$$

where we call y_i an encoded symbol.
Further we define $Enc = \{(y_i, A_i) \in \mathbb{F}_2 \times \mathbb{F}_2^k : y_i = \langle A_i, x \rangle, A_i \text{ generated as above for } i \in \mathbb{N}\}$.
Our encoding algorithm is a generator. Hence every call it will return a tuple $(y_i, A_i) \in Enc$. The application is then responsible to send all these tuples. (or a big enough subset of it.)
In our algorithm we give a tuple a boolean value which can be identified as y_i and a integer value which can be identified as A_i by mapping the vector in $\mathbb{F}_2^k$ to a binary value.

2.4 Decoding

The application has to collect an finite subset of Enc. Let's say $Dec \subseteq Enc$ is a such subset. Up to renaming, we can say that we've collected $Dec = \{(y_1, A_1), \ldots, (y_N, A_N)\}, N \in \mathbb{N}$ (which we can say because of the randomness of it). We set $y = (y_1, \ldots, y_N)^T \in \mathbb{F}_2^N$ and a matrix $A \in Mat_{N \times k}(\mathbb{F}_2)$ where $A_1, \ldots A_N$ are the rows. Then by construction of y we have the following linear equation system.

$$A \cdot x = (y_1, \ldots y_N)^T \tag{1}$$

Since our goal is to compute x, it is sufficient that $rank(A) = k$, otherwise the decoding fails. In particular one can see that (A, N) fulfills the Definition 1, hence we have constructed a fountain code $(A, N) \in \mathcal{F}$.
Further we call $o = N - k$ the overhead of this code. Since it is sufficient that $N \geq k$, we have $o \geq 0$. To have a small overhead is one of the main interest of fountain codes.
Since this overhead it dependent to the distribution of the probability space, the next step is to find a reasonable distribution on $\mathcal{P}(\mathbb{F}_2^k)$.
Now we need to solve the linear equation system 1. For this we used the belief-propagation decoder which we got from [1] (page 230 - 231.)

2.5 The Random Binary Fountain

We call the fountain code on the probability space $(\mathbb{F}_2^k, \mathcal{P}(\mathbb{F}_2^k), P)$ where P has an uniform distribution the Random Binary Fountain.
From [1] and by testing we realised that this code has a big overhead.

3 Luby transform code (LT codes)

For the following we need a definition.

Definition 2 (hamming weight)**.**
For any $a \in \mathbb{F}_2^k$ we call $H(a)$ its hamming weight, where H is a function defined as following.

$$H : \begin{cases} \mathbb{F}_2^k & \to \{1, \dots k\} \subseteq \mathbb{Z} \\ a = (a_1, \dots, a_k) & \mapsto \sum_{i=1}^k a_i, \end{cases}$$

where we use here the standard addition of the group $(\mathbb{Z}, +)$.

The random binary fountain code with $(\mathbb{F}_2^k, \mathcal{P}(\mathbb{F}_2^k), P)$ using the belief-propagation decoder has a problem. If Dec is the set of collected encoded symbols, we need for any $i \in \{1, \dots, k\}$ a tuple $(y, a) \in Dec$ such that the coefficient vector a has the hamming weight i, i.e. $H(a) = i$. Otherwise the decoder fails. We want to calculate the probability of a sample $a \in \mathbb{F}_2^k$ having $H(a) = i$. For this we define a random variable $X = H$. Then one can see that by combinatorial reasons we get

$$P(X = i) = \frac{\binom{k}{i}}{2^k},$$

which means that the probability for a vector $a \in \mathbb{F}_2^k$ with $H(a) = 1$ is a lot smaller than one with $H(a) = \lceil \frac{k}{2} \rceil$, what is a big disadvantage for the belief-propagation decoder. Therefore we want to modify our probability measure.
Let $\tilde{P}$ be a probability measure on $X(\mathbb{F}_2^k) = \{1, \dots, k\}$. For every $i \in \{1, \dots, k\}$ we write $\tilde{P}(\{i\}) = \Omega_i$. This induces a probability mass function on $\mathbb{F}_2^k$.

$$P(X = i) = \tilde{P}(\{i\}) = \Omega_i.$$

Further, we define a conditional probability on $\mathbb{F}_2^k$ with

$$P(\{x\}|X = i) = \frac{1}{\binom{k}{i}}, \text{ for any } i \in \{1, \dots, k\}, x \in \mathbb{F}_2^k.$$

This has a uniform distribution on $X^{-1}(\{i\})$ for each $i \in \{1, \dots k\}$. Since $X^{-1}(\{1, \dots k\}) = \mathbb{F}_2^k$ and $P(\{x\}|X = i) = \frac{P(\{x\}, X=i)}{P(X=i)}$ we can define the following probability measure P_l on $\mathbb{F}_2^k$

$$P_l(\{x\}) = P(\{x\}, X = i) = \frac{\Omega_i}{\binom{k}{i}}, \text{ where } i \in \{1, \dots, k\} \text{ such that } x \in X^{-1}(i)$$

The LT code is a fountain code on $(\mathbb{F}_2^k, \mathcal{P}(\mathbb{F}_2^k), P_l)$. It follows that the Lt code is unique up to the probability measure $\tilde{P}$. Our naive approach was to use the uniform distribution, since we need coefficient vector of all weight, but considering [1] page 235 - 238 we have found a better one. We use the probability measure

$$P_{LT}(\{i\}) = \begin{cases} \frac{1}{k} & \text{if } i = 1 \\ \frac{1}{i \cdot (i-1)} & \text{if otherwise} \end{cases}$$

where $i \in \{1, \dots k\}$. By trying out we could convince our self that this distribution was a lot better.
In our algorithm we realised this probability measure P_{LT} as following. We calculated it in two steps. At first, we sampled over $\{1, \dots, k\}$ by weighting each number such that this sampling

corresponded to P_{LT} on $\{1,\dots k\}$. Assuming we got $i \in \{1,\dots,k\}$ we did a new sampling over $\{a \in \mathbb{F}_2^k : H(a) = i\} \subseteq \mathbb{F}_2^k$ with respect of uniform distribution. This second sampling gave us a random vector $a \in \mathbb{F}_2^k$ with respect to the probability explained above.
The LT Code is a lot better as the random fountain code. We had only a overhead of A little Less than k. While with random fountain code we had a overhead around $2k$.

4 Raptor codes

Now how can we improve the LT code? We consider the linear equation system (1). One can see that in order to have a full rank, (of k) it is not allowed that the matrix A has a column with all zeros. Now a column of zeros means in sense of encoding algorithm that there exist a $i \in \{1,\dots,k\}$ such that for every coefficient vector $a \in \mathbb{F}_2^k$ we sampled, that $a_i = 0$. Obviously the probability of such column decreases for every bigger N, but is still not low for a N in the near of k. (Consider that in ideal case we have $N = k$.) There are further details about this probability in [1] page 238 - 242. Now how do we decrease this probability.
For a $\tilde{k} > k$ precoding is defined as a map

$$L_r : (x_1,\dots,x_k) \in \mathbb{F}_2^k \mapsto (x_1,\dots,x_k,x_{k+1},\dots x_{\tilde{k}}) \in \mathbb{F}_2^{\tilde{k}}$$

where for any $i \in \{k+1,\dots,\tilde{k}\}$ we have $x_i = \sum_{j=1}^{k} c_j x_j$, with $c_j \in \{0,1\}$.
Hence one can see that the matrix of the linear map L_r has a submatrix in the first k rows and columns which is the $k \times k$ unit-matrix, the last two rows are, up to the last two coordinate which are 0, arbitrary and the last two columns are zero vectors. Hence this matrix has rank of k.
The raptor code is a fountain code with such precoding and then applying the LT code on the new tuple of source symbols $(x_1,\dots,x_k,x_{k+1},\dots x_{\tilde{k}})$. the idea of this precoding is to decrease the probability that the matrix A in (1) gets a zero-column what causes that the algorithm needs less encoded symbols.
In our algorithm we applied the following precoding

$$L_r : (x_1,\dots,x_k) \in \mathbb{F}_2^k \mapsto (x_1,\dots,x_k,x_{k+1},\dots x_{\tilde{k}}) \in \mathbb{F}_2^{k+1}$$

where $x_k + 1 = \sum_{j=1}^{k} x_j$. And applied the LT code of before on the new tuple of source symbols. The decoding algorithm needed to be changed a little. Obviously we want only recover $(x_1,\dots x_k)$ since we know how x_{k+1} is construted. With this knowledge we could add the equation $\sum_{j=1}^{k} c_j x_j + x_{k+1} = 0$. And the rest remained as same as in LT codes.

Unfortunately this precoding didn't decrease our overhead compared to LT codes. We assume that the choice of our precoding was not good enough. We tried some other precoding but unfortunately we couldn't decrease our overhead remarkable.

5 Result and possible improvements

As a result our code works. In our demo one can see that we can send a bytestring (in our case a textmessage "Lorem Ipsum..." with the correct blocksize of 32 bits) and decode this on the other side.
But unfortunately for a tuple $(x_1,\dots,x_k) \in \mathbb{F}_2^k$ with k symbols, We need at little less than $2k$ of encoded symbols in order to recover $(x_1,\dots,x_k)$. This is a lot more if we compare this to the

newest Raptor Code, the RaptorQ code which only needs a overhead of 1 or 2 in average. We want to show three possible improvement in order to get a less overhead.
The first possible improvement is to improve the decoding algorithm. Our algorithm uses the belief-propagation decoder where our matrix in the linearly equation system need to have the for every $i \in \{1, \dots k\}$ at least one row $v \in \mathbb{F}_2^k$ of the matrix A such that $H(v) = i$. From linear algebra, we know that this is not necessarily to solve the equation system (Consider the unit-matrix). But it is necessarily that the matrix has rank of k. We could add some operation on this equation system which are done in gauss-algorithm such that the matrix A gets the property for succeeding the belief-propagation decoder earlier. This could decrease the overhead.

A second possible improvement is to use a better precoding. Our precoding doesn't really change the overhead. There might be better precoding maps, with another probability space i combination which will have e less overhead. For example the RaptorQ code which has only 1 or 2 in average.

Finally to the third possible improvement. It would be nice if this code could be also used, such that the receiver can receive one message by recovering the decoded symbols from different senders. Unfortunately at the moment we have a variable in our code which is called $informationID$ and will be produced randomly for every message. Hence if two different sender send one encoded symbol, even if they are from the same message, since the ID will differ, the receiver won't recognize these as symbols from the same message. As $informationID$ we could use the hashvalue of the message instead of a random variable which will cause that the receiver will recognize the recovered symbols as symbols from the same message since the hashvalue must be the same.

6 Integration to BACnet

We consider the linear equation system (1). One can see that our algorithm is lethal to bit-errors. Because a bit-error changes the solution of our equation system, hence we won't be able to recover the correct message. Hence a precondition for a working fountain code is to have a erasure channel.

Definition 3 (erasure channel)**.**
We call a channel a erasure channel if only correct packet are being collected. If the packet is corrupt, the packet must be dropped.

Fortunately BACnet fulfils this precondition. In every packet, there is a hash-value of the data, which will be compared with the new hash-value of the recovered packet. If these are not equal, the packet will be dropped. Since the hash-function is injective restricted in every small enough neighborhood of a point, we can be assured to get no bit-errors, hence to have a erasure channel.

We now want to give a brief overview on how our encode and decode could be used in BACnet.

6.1 Encoding

Raptor Code generates multiple data packages for data which should be encoded. Therefore the decoding should be made for each bundle of information that is getting transmitted (e.g. a log entry) before it's reached down to the transport layer. So that for each generated package there will be it's own "internet package".

6.2 Decoding

The decoding should be in the same position. Our decoding function takes as input such transferred packages. At the point where it has enough packages to decode the data, it will return the decoded data. Until that point it won't return anything. So if the raptor code is used, every received internet package should be given to the decoder and once the decoder returns a value, it can be further processed.
For more details how to use these codes please consider our README.

7 Lessons learned

Since we didn't use a lot libraries and therefore we studied the raptor code by [1] there was a lot to learn. Every step in improving the code was a new algorithmic challenge. Since we both are very interested in theory and algorithms it was also a very nice experience.

References

[1] Amin Shokrollahi, Michael Luby. Raptor Codes. *Foundations and Trends® in Communications and Information Theory*, 6(3-4):213–322, 2009.

A Libraries

In order to write our functions, we used the following libraries.

- math
- numpy
- random