

# CSLU Challenge - Code Fountain

Carl Ellis

`carl.ellis@comp.lancs.ac.uk`

January 19, 2012

## 1 Summary

Your task is to retrieve a file from a server hidden somewhere in the university. It is being served from an open UDP port but encoded using a forward error correction algorithm called a fountain code. First person to show me the file gets a prize from the author.

## 2 Fountain Codes

Fountain codes allow a file to be encoded into a stream of encoded data. The order of the encoded data is irrelevant and so long as a decoder listens to enough of the data, the file will be decoded successfully. Optimum fountain codes can decode a file of  $k$  blocks by listening to exactly  $k$  chunks of encoded data, but this is unlikely due to the random nature of the encoding - and more to the point, the author never does anything optimally.

Encoded data is created by taking a random number of blocks, XOR'ing them together and then transmitting this data with the block ids used in the encoding process. However as the number of blocks used can be variable, and transmitting variable numbers across the network in static packets is near impossible, the block numbers will also be encoded. The block id encoding method is simple: A common pseudo-random number generator (PRNG) algorithm is shared between client and server, the PRNG on the server chooses which blocks to encode, and then the PRNG seed is sent with the data.

The client receives a seed and encoded data in each network packet. Then using the PRNG the client reproduces the number of blocks and the block ids used to encode the data. If the client has already encoded any of the blocks used in the encoded data, it XORs to remove the already known blocks from the data. When only one block is left in the encoded data, it has been successfully decoded; if there is more than 1 block encoded in the data more information is needed so the data is stored until more blocks have been received.

Key to the success of this algorithm is the distribution of the number of blocks encoded in the stream. Single blocks allow for decoding larger blocks, but do not cover a lot of the original file; multiple blocks allow for decoding large portions of the original file but require decoded blocks to fully unlock. Ideally, a good distribution is one which has a probability peak at 1 and another at some significant portion of the number of blocks. A distribution exists, called the [Robust Soliton](http://en.wikipedia.org/wiki/Soliton_distribution)([http://en.wikipedia.org/wiki/Soliton\\_distribution](http://en.wikipedia.org/wiki/Soliton_distribution)).

## 3 Encoding Procedure

In order for challengers to have a fair chance, the PRNG and an implementation of a robust soliton distribution can be found at the author's [github page](http://github.com/carl-ellis/)(<http://github.com/carl-ellis/>).

The PRNG libraries were used for the server and it is highly recommended that the challenger uses them - unless they want to reimplement them in a language of their choice, but this will reduce their chance of gaining a prize due to the time factor!

The encoding procedure for a block on the server is as follows:

1. Create a random seed
2. Set the PRNG Seed
3. Set the Soliton seed with the PRNG (a seed  $s$  is  $\{0 \leq s < 2^{32} : s \in \mathbb{Z}\}$ )
4. Get the number of blocks  $i$  from the Soliton distribution
5. Create a chunk of empty encoded data
6. Then  $i$  times:
  - (a) Pick a random block using the PRNG
  - (b) XOR the block data with the encoded data
7. Transmit the block

## 4 Protocol & Packet Format

The server is a UDP server which will respond to a simple protocol. If the string size is sent, then the number of blocks is returned by the server. If any other string is sent, then the server will reply with an encoded block with a format described in Figure 1.

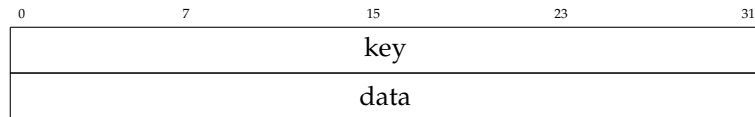


Figure 1: Data packet format

## 5 Target computer

Server: 148.88.226.231

Port: 48155

Protocol: UDP

Line quality: Poor

**Good luck!**