# Bloom Filters for Producer List Propagation

Joseph Kearney

February 7, 2020

## 1 Bloom Filters and Their Use in Catalyst

One key issue posed from the consensus mechanism described in [**?**] is the propagation and distribution of information across producers and eventually across the entire network. This is particularly an issue for the distribution of the list of producers that a specific producer considers to have created the correct ledger state update. With larger producer pool which are necessary for security consideration, large lists of information are impractical for distribution. When it it is considered the every PID is 32Bytes long, for a group of 1000 producers lists would reach 32KBytes if the raw lists of producers are distributed. While compression can be used, what is gained in size of the elements is lost in time for compression and decompression of the lists. This is especially true when it is considered that every producer could potentially have to decompress 999 lists from a pool of 1000 producers. Furthermore using a User Datagram Protocol (UDP) as used in Catalyst, the potential for lost information on elements of that size is high. Therefore a more compact and efficient method is required. Bloom filters

Bloom filters are data structures that allow proof of membership of a set. They work is such a way that they can never provide a false negative i.e. showing that an element is in fact in the bloom filter as false. They can however show false positives meaning that an element that is not in the bloom filter can in fact come back true. Bloom filters are simple byte arrays of a specified length. Upon initialisation all bits within the byte array are set to 0. For each element to be added to the bloom filter it is hashed a defined number of times and the corresponding bit for the hash is flipped to a 1. For example for a small bit array `0bx0000` and element `Hello` is hashed to give the digest `04` then the $4^{th}$ bit would be flipped so the bloom filter would become `0bx1000`. This works on a modulo point, so if the digest was `05` then the first bit would be flipped giving the bloom filter `0bx0001`. For each element added to the bloom filter multiple hashes are created and added to the bloom filter, for example if 3 hashing functions are used 3 bits in total will be flipped in the bloom filter and these 3 flipped bits will represent the element.

Bloom filters are made up of four key variables, these are:

- $m$ - The size of the bloom filter in bits.
- $n$ - The number of elements to be added to the bloom filter.
- $k$ - The number of hashing functions that are performed on each element added to the bloom filter.
- $p$ - Probability of a false positive occurring for an element within the bloom filter.

The size of a bloom filter can be defined by:

$$m = -(n \times log(p)) \div (log(2)^2)$$

The optimum hash count of a bloom filter can be defined by:

$$k = (m \div n) \times log(2)$$

The false positive rate for a bloom filter is:

$$p = (1 - e^{-kn \div m})^k$$

The number of elements added to the bloom filter is uniform across all producers as this must be set to the total number of producers for one cycle.

The reason for choosing a bloom filter over directly transferring the lists is for size considerations. Distributing a list containing 100 or a 1000 elements will be extremely cumbersome due to the size. While a bloom filter for 1000 elements would be approximately 1.4kBytes in size. Furthermore, due to the goal of the list creation is to efficiently deduce the members of the worker pool that created the correct delta and that bloom filters are an efficient method for membership validation the use of bloom filters lends itself to the Catalyst consensus algorithm.

## 2 Statistics

Using the script in [**?**] we can determine the probability of failure for any one particular node. Probability of failure is defined as the probability of any one node failing to hold a complete bloom filter of all producers who successfully produced the correct ledger state update. The process for a producer is such that, they produce a bloom filter of all producer they retrieved correct proposed ledger state updates at the beginning of the campaigning phase. This bloom filter is then passed of to all other producers. However, due to natural inefficiencies it is unlikely all producers will receive all of the proposed updates therefore, the correct list must be constructed from all the received bloom filters. It is this created list for each producer that we will test. We analyse two variables, firstly is the message propagation ration, this is the percentage of messages that we assume each producer recieves from the rest of the network, i.e. at a message propagation ration of 0.75, for a producer pool of 100, each producer would recieve randomly 75 other producers bloom filters. The second variable we will be testing is the size of the producer pool.