CSC791- Advanced Algorithms Krishna Karthik Gadiraju
H1: Bloom Filters Unity ID: kgadira

**Bloom filter principle:**
*Whenever a list or set is used, and space is at a premium, consider using a Bloom filter if the effect of false positives can be mitigated*

**Explanation:**
Bloom filter is an array of m elements, each of which can be 0 or 1. When a set of n elements is available, and n is large, it will **not be possible to store all n values in memory**. In such situations, storing these values in a bloom filter by the following tasks **ensures that we are only storing m values at any point of time.**
1. using a set of hash functions that map a value xi from S into an index of bloom filter (0..m)
2. for each value, for each index generated by hashing the value from the set of hash functions, setting the value at that index on bloom filter to 1.
By doing this, we are **providing an upper bound on amount of memory required for storing information** regarding the sets/lists. However, this will **only help us with indicating the presence/absence of a variable.** Also, since n values are being hashed into m, and n >> m, **there is a possibility that for a value a that is not present in the filter, due to a combination of hashes for different values**, **say b and c , the indices corresponding to hashing(a) will indicate that a is present.** This is called a false positive. This is a limitation to the bloom filter implementation and as a result, only use a bloom filter if this effect can be mitigated.

**Bloom filter variants:**
Main issues: counting, deletion, multi-sets and space-efficiency.

| Filter Name | Issue | How? |
|---|---|---|
| **Counting Bloom Filter**<br>This overcomes some of the issues with standard bloom filters such as counting and deletion. | deletion | Adds a counter for each element in bloom filter. Bloom filter = m bits + m counters. When an item is inserted, its corresponding counters (indices obtained from hashing) are incremented. When it is deleted, corresponding counters are deleted. Counter overflow is avoided by choosing suitably large counters. |
| | counting | Can provide approximate counts of each element inserted into bloom filter by using the values of counters. (Approximate since hashing other values might cause increments in counters as well (overlaps)) |
| **Deletable Bloom Filter**<br>It ensures that element deletions are done by using least memory (minimal memory cost) when compared to other Counting Bloom filters. | deletion | It keeps track of the bits where collision happens. Since an element can be considered to be removed if atleast one of its corresponding entries in the bloom filter is reset. A bit array of size m is divided into r regions. Collision information is represented by considering a bitmap of size r that will code a 0 with a region that is collision free and 1 otherwise. This results in element removal being probabilistic, depending on r. This way it enables probabilistic element deletions |
| | Space Efficiency | using the probabilistic model and keeping track of bit regions instead of bits ensures that least amount of memory is used when dealing with deletions. |
| **Space Code Bloom Filters**<br>This bloom filter can be used to measure approximately per-flow traffic at high speeds. It can be enhanced by parameter tuning to lessen the cost incurred due to the tradeoff between measurement accuracy vs computational and storage complexity. | Multi-sets | Space code bloom filters gives an approximate representation of a multiset. By using Maximum Likelihood Estimation (MLE), we can calculate/measure the multiplicity of any element in the multiset. Since it is built for measuring per-flow traffic, each traffic flow is represented as an element and the multiplicity of the element = number of packets present in the flow. |

**Some others include:**
Decaying Bloom Filters, Stable bloom filter
Issue Addressed: duplicate element detection

CSC791- Advanced Algorithms Krishna Karthik Gadiraju
H1: Bloom Filters Unity ID: kgadira

**Applications:**

| Application | Task | BF | Why? |
|---|---|---|---|
| Packet Routing and Forwarding: Bloom filters can be used for different routing and forwarding tasks such as IP Lookups, Duplicate Detection, Forwarding Engines and Deep Packet Scanning | **IP Lookup:** Since packets are looked up and forwarded based on the address prefixes, next hop destination can be identified by finding the longest prefix match. | The algorithm **Longest Prefix Matching** proposes the use of **Asymmetric Bloom Filters**. | Since Asymmetric Bloom Filters utilize prefix distributions to assign allocate memory resources, they are suitable for this task. |
| | **Duplicate Detection:** where the task is to identify duplicates in a stream of packets | **Stable Bloom Filter** or a **Decaying Bloom Filter** | **Stable Bloom Filter**: Stable Bloom Filter is capable of identifying duplicates. When updating -> randomly select P counters, decrement their value by 1 ->set to max the k counters that correspond to the element being inserted. This results in probabilistic aging of the counters, and ensures that the expected fraction of zeros in the bloom filter remains the same. |
| **Security:** Verfication of password strength, managing security of network by protecting user from malicious internet packets, different kinds of TCP/IP fragmentation evasion attacks, DDOS attacks are a few of the various security aspects that need to be handled by a distributed system. Since bloom filters inherently use hashing, they are very suitable for Security based tasks. | **TCP/IP Fragmentation Evasion Attacks** | **Counting Filters** | 1. split attack signatures into 3-byte strings -> 2. place each of them in a counting bloom filter -> 3. when an incoming fragmented packet data is received, compare(fragmented packet data, counting bloom filters) 4. If substring is detected, remove substring from CBF, enable full string matcher, and then check for false positives. If even full string matcher is matched, then block the flow, thereby restricting fragmentation evasion attack. |

**References:**

[1] Tarkoma, Sasu, Christian Esteve Rothenberg, and Eemil Lagerspetz. "Theory and practice of bloom filters for distributed systems." *Communications Surveys & Tutorials, IEEE* 14, no. 1 (2012): 131-155.
[2] Classroom material