BLOOM FILTERS

· invented by Bloom in the 70's · probabalistic (or approximate) dictionaries

stores Fs: symmany/fingerprint of dynamic set S

· regular dictionaries store actual elements

BF\_INSERT(Fs, X): S= SUEX3

BF\_SEARCH(F5, X):

FALSE - x is not in the Set, x & S

TRUE -> x is probably (not certainly) in the set

Bloom filter? Because it is very space efficient.

"Ex: you are a webserver with a large list of blacklisted websites. Cannot store whole list in main memory, then store summary in BF. First check BF, it returns FALSE, know not blacklisted, only check whole list it BF returns TRUE (chance of false positive)."

o use when space constraints & false positives acceptable & when deleting not a big deal

Implementation of BLOOM FILTERS / params: m, t

BF[0, m-1]; an array of m bits, initialized to O

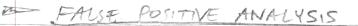
Eh,, ..., he}: set of tem hash functions where hi! U→ 20, ..., m-13,

- recall all hash functions have independent uniform PDF

O(t) · BF\_INSERT(BF, X): for i from 1 to t, BF[hi(x)] = 1;

\* element produces "finger print" from hash functions which is stoned

O(t) • BF-SEARCH(BF,x): for i from 1 to t, if (BF[hi(x)] == 0) return false, else true params t, m affect false positive rate



Assume n elements inserted. Compute probability that BESEARCH(BF, X) returns a false positive, if x has not been inserted.

PROB[BF[h,(x)]== 1 28 BF[ $h_i(x)$ ]==1 28 ... 88 BF[ $h_i(x)$ ]==1)

• hi have uniform independent PDF over m, all equal probs.

PROB[t-  $BF(h_i(x))$ ==1] for any i from 1 to t• PROB[BF(j]==1] for j from 0 to m-1 after n insertions

= 1 - PROB[BF(j)==0] for j from 0 to m-1 after n insertions

=  $1 - (1 - V_m)tn = (1 - e^{-tm})t$  ?!?



