CS2040S
Data Structures and Algorithms
(e-learning edition)

Fingerprints and Hashing

Today: Fingerprints

Sets

- Hash table sets
- Fingerprint Hash Table
- Bloom Filters

Quick Review

Symbol Table

```
public interface  SymbolTable<Key, Value>
```

void	insert(Key k, Value v)	*insert (k,v) into table*
Value	search(Key k)	*get value paired with k*
void	delete(Key k)	*remove key k (and value)*
boolean	contains(Key k)	*is there a value for k?*
int	size()	*number of (k,v) pairs*

Note: no successor / predecessor queries.

Quick Review

Hash Table

- Implements a symbol table.
- Goal:
  - O(1) insert
  - O(1) lookup
- Idea:
  - Store data in a large array.
  - Hash function maps key to slot in the array.
  - Challenge: choosing a good hash function.

Quick Review

Hash Table with Chaining

- Each array slots stores a linked list.
- All items mapped to the same slot are stored in the linked list.

Open addressing:

- Each array slot stores one element.
- On collision, continue probing.
- Probe sequence specifies order in which cells are examined.

A few examples

Facebook:

- I have a list of (names) of friends:
  - John
  - Mary
  - Bob
- Some are online, some are offline.
- How do I determine which are on-line and which are off-line?

A few examples

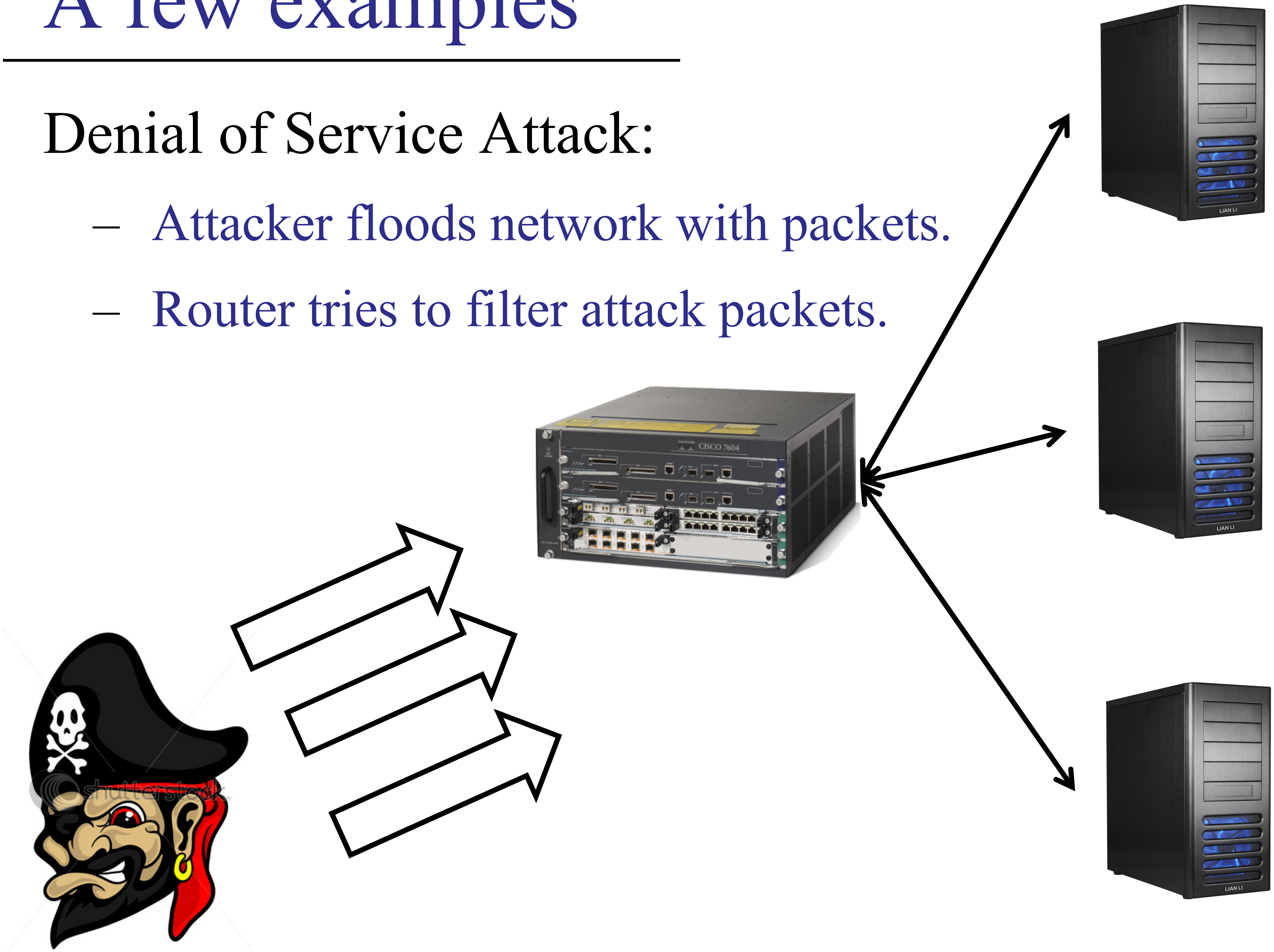
Spam filter:

- I have a list bad e-mail addresses:
  - @ mxkp322ochat.com
  - @ info.dhml212oblackboard.net
  - @ transformationalwellness.com
- I have a list of good e-mail addresses:
  - My mom.
  - *.nus.edu.sg
- How do I quickly check for spam?

A few examples

Denial of Service Attack:

- Attacker floods network with packets.
- Router tries to filter attack packets.



A few examples

Denial of Service Attack:

- Attacker floods network with packets.
- Router tries to filter attack packets.

1. Keep list of bad IP addresses.

2. Only allow 100 packets/second from each IP address.

Abstract Data Type

Set

public class	**Set<Key>**	
void	insert(Key k)	*Insert k into set*
boolean	contains(Key k)	*Is k in the set?*
void	delete(Key k)	*Remove key k from the set*
void	intersect(Set<Key> s)	*Take the intersection.*
void	union(Set<Key> s)	*Take the union.*

Properties:

- No defined ordering.
- Speed is critical.
- Space is critical.

Abstract Data Type

Set

public class	**Set<Key>**	
void	insert(Key k)	*Insert k into set*
boolean	contains(Key k)	*Is k in the set?*
void	delete(Key k)	*Remove key k from the set*
void	intersect(Set<Key> s)	*Take the intersection.*
void	union(Set<Key> s)	*Take the union.*

Java: HashSet<…> implements Set<…>

Abstract Data Type

Set

public class	Set<Key>	
void	insert(Key k)	*Insert k into set*
boolean	contains(Key k)	*Is k in the set?*
void	delete(Key k)	*Remove key k from the set*
void	intersect(Set<Key> s)	*Take the intersection.*
void	union(Set<Key> s)	*Take the union.*

Solution 1: Implement using a Hash Table

Implementing a Set

Use a hash table:

hash(“`www.gmail.com`”)

hash(“`www.apple.com`”)

hash(“`www.microsoft.com`”)

hash(“`www.nytimes.com`”)

0	**0**
1	**0**
2	**www.gmail.com**
3	**www.apple.com**
4	**0**
5	**0**
6	**www.microsoft.com**
7	**0**
8	**www.nytimes.com**
9	**0**

Which problem does a hash table not solve?

1. Fast insertion
2. Fast deletion
3. Fast lookup
4. Small space
5. All of the above
6. None of the above

A hash table takes **more** space than a simple list!

Implementing a Set

Use a hash table:

hash(“www.gmail.com”)

hash(“www.apple.com”)

hash(“www.microsoft.com”)

hash(“www.nytimes.com”)

0	**0**
1	**0**
2	**www.gmail.com**
3	**www.apple.com**
4	**0**
5	**0**
6	**www.microsoft.com**
7	**0**
8	**www.nytimes.com**
9	**0**

Implementing a Set

Use a hash table:

Why do we store the URL data in the hash table?

hash(“`www.microsoft.com`”)

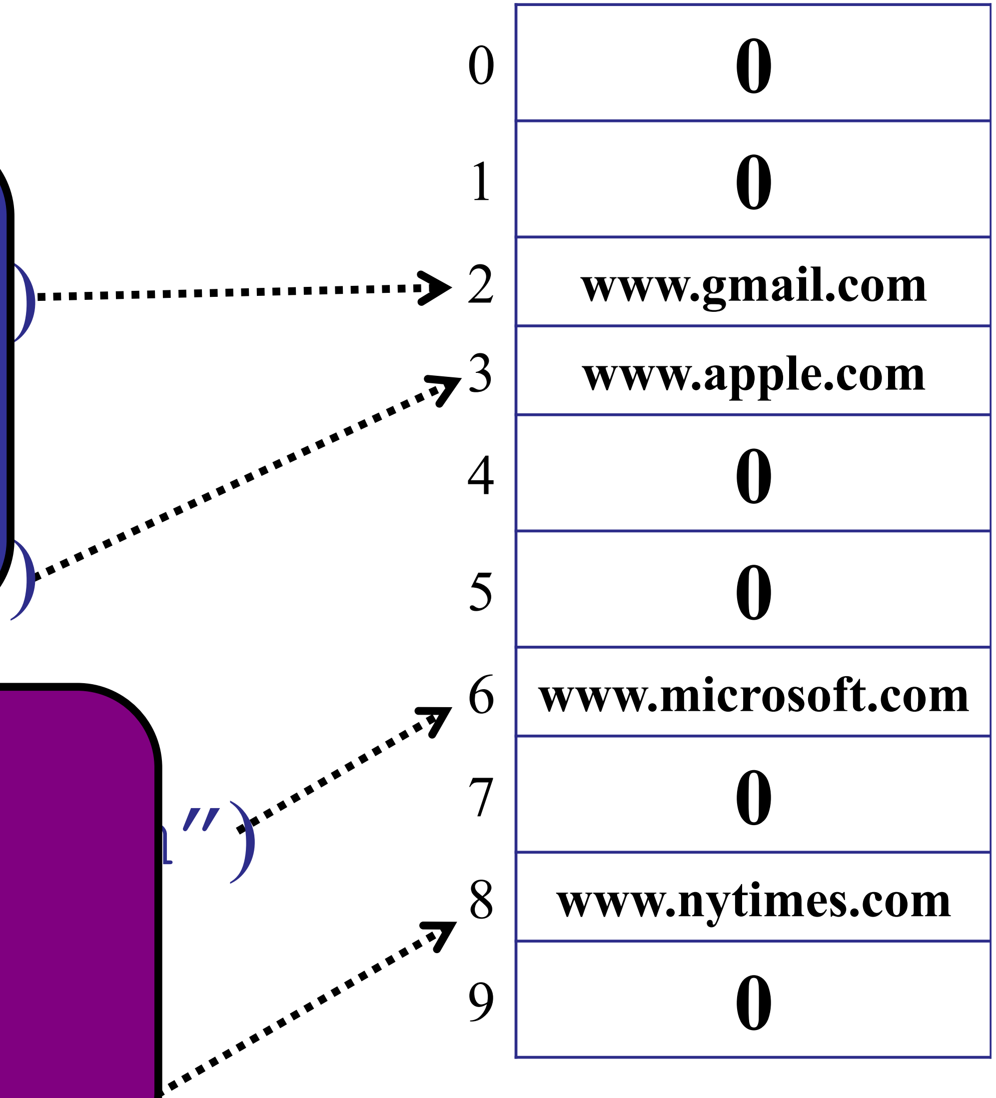
hash(“`www.nytimes.com`”)

0	**0**
1	**0**
2	**www.gmail.com**
3	**www.apple.com**
4	**0**
5	**0**
6	**www.microsoft.com**
7	**0**
8	**www.nytimes.com**
9	**0**

Implementing a Set

Use a hash table:

Why do we store the URL data in the hash table?



So that we can resolve collisions!

Abstract Data Type

Set

public class Set<Key>

void	insert(Key k)	*Insert k into set*
boolean	contains(Key k)	*Is k in the set?*
void	delete(Key k)	*Remove key k from the set*
void	intersect(Set<Key> s)	*Take the intersection.*
void	union(Set<Key> s)	*Take the union.*

Solution 2: Implement using a Fingerprint Hash Table

Implementing a Set

Use a fingerprint:

- Only store/send *m* bits!

hash(“`www.gmail.com`”)

hash(“`www.apple.com`”)

hash(“`www.microsoft.com`”)

hash(“`www.nytimes.com`”)

0	**0**
1	**0**
2	**1**
3	**1**
4	**0**
5	**0**
6	**1**
7	**0**
8	**1**
9	**0**

Fingerprints

Set Abstract Data Type

– Maintain a vector of 0/1 bits.

```
insert(key)
1. h = hash(key);
2. m_table[h] = 1;

lookup(key)
1. h = hash(key);
2. return (m_table[h] == 1);
```

The key difference of a Fingerprint Hash Table (FHT) is:

1. A FHT prevents collisions.
2. A FHT does not store the key in the table.
3. A FHT works with simpler hash functions.
4. A FHT saves time calculating hashes.
5. I don’t understand how an FHT is different.

Implementing a Set

Use a fingerprint:

Implementing a Set

What happens on collision?

Implementing a Set

Lookup operation:

hash(“`www.microsoft.com`”)

0	**0**
1	**0**
2	**0**
3	**1**
4	**0**
5	**0**
6	**1**
7	**0**
8	**1**
9	**0**

If the URL is in the web cache, it will always report **true**.
(No false negatives.)

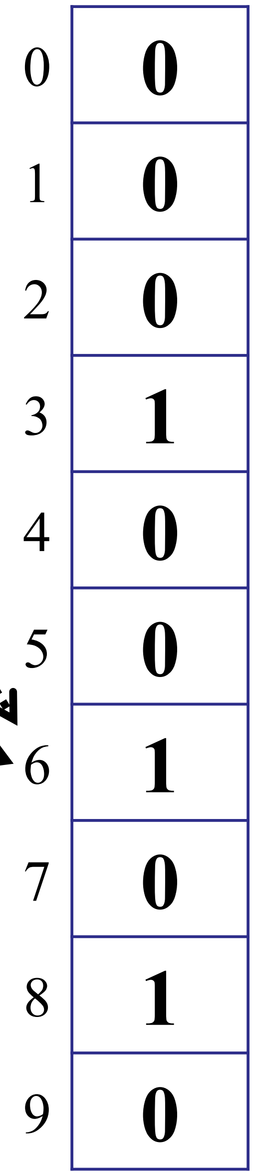
Fingerprint Hash Table

Insert operation:

hash(“`www.microsoft.com`”)

Lookup operation:

hash(“`www.rugby.com`”)



Even if the URL is NOT in the set,
it may *sometimes* report **true**.
(False positives.)

Facebook example: if the FHT stores the set of online users, then you might:

1. Believe Fred is on-line, when he is not.
2. Believe Fred is offline, when is not.
3. Never make any mistakes.

Spam example: it is better to store in the Fingerprint Hash Table:

1. The set of **good** e-mail addresses.
2. The set of **bad** e-mail addresses
3. It does not matter.

I think it is better to mistakenly
accept a few SPAM e-mails
than to accidently reject an e-mail
from my mother!

Fingerprint Analysis

Probability of a false negative: 0

Fingerprint Analysis

Probability of a false positive?

On lookup in a table of size m with n elements,

Probability of **no** false positive:

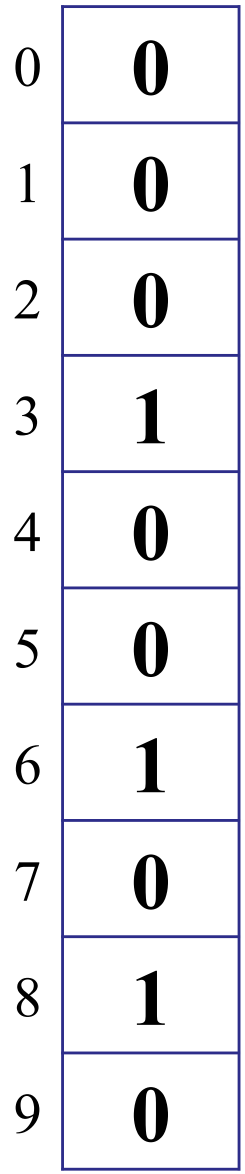
$$\left(1 - \frac{1}{m}\right)^{n} \approx \left(\frac{1}{e}\right)^{n/m}$$

chance of no collision

Fingerprint Analysis

Probability of collision?

hash(“`www.gmail.com`”)



What is the probability that no other URL is in slot 3?

Fingerprint Analysis

Probability of **no** false positive: (simple uniform hashing assumption)

$$\left(1 - \frac{1}{m}\right)^{n} \approx \left(\frac{1}{e}\right)^{n/m}$$

Probability of a false positive, at most:

$$1 - \left(\frac{1}{e}\right)^{n/m}$$

Fingerprint Analysis

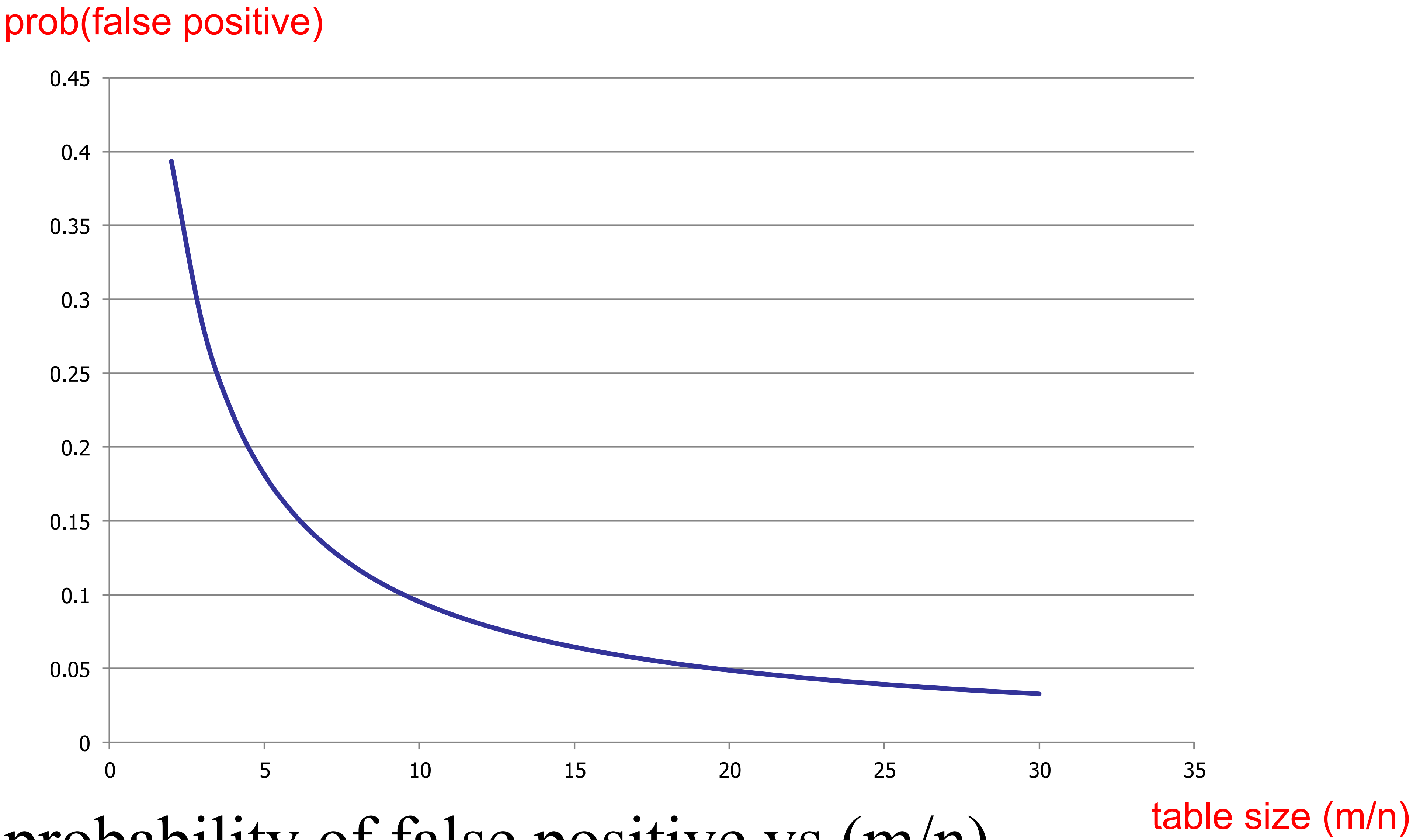
Assume you want:

- Probability of false positives $< p$
  - Example: at most 5% of queries return false positive.

    $p = .05$

- Need: $\frac{n}{m} \leq \log\left(\frac{1}{1-p}\right)$
  - Example: $m >= (13.5)n$

Fingerprint Analysis



probability of false positive vs (m/n)

Summary So Far

Fingerprint Hash Functions

- Don't store the key.
- Only store 0/1 vector.

Summary So Far

Fingerprint Hash Functions

- Don’t store the key.
- Only store 0/1 vector.
- Trade-off:
  - Reduced space: only 1-bit per slot
  - Increase space: bigger table to avoid collisions

Fingerprint Hash Table

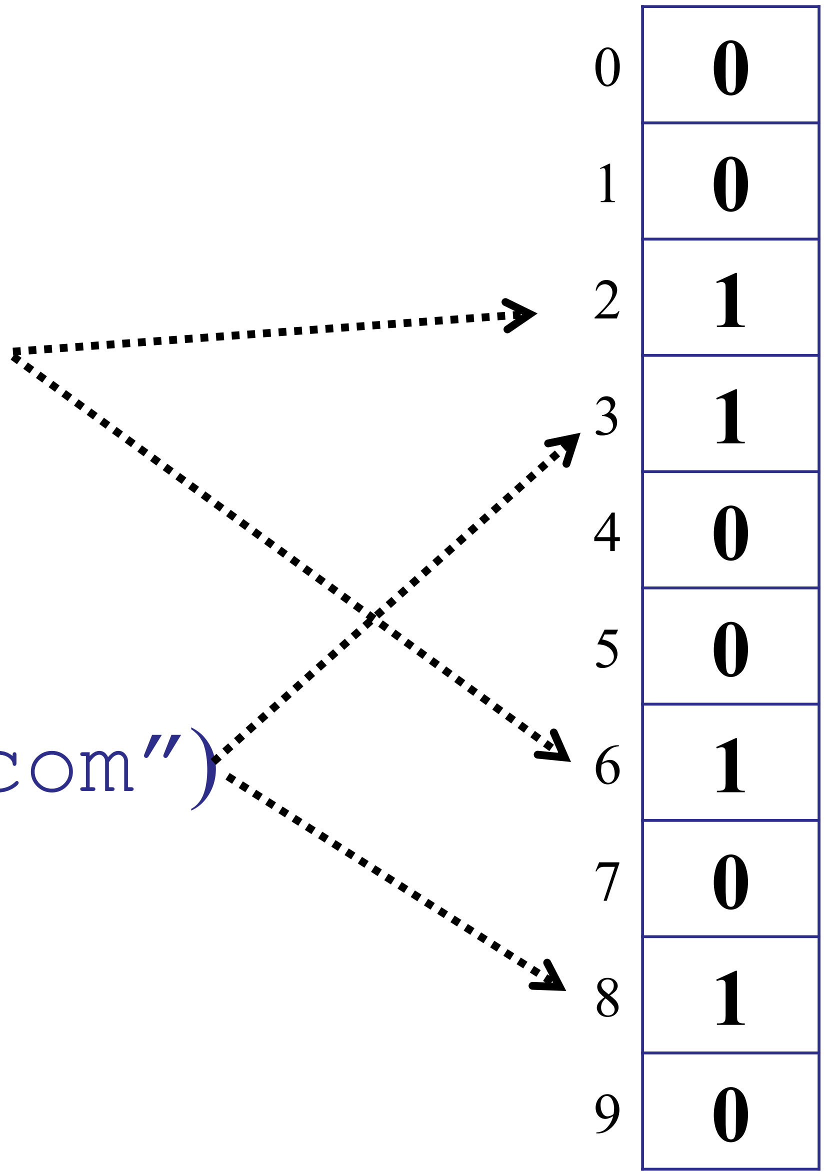
Can we do better?

Bloom Filter

Idea: use 2 hash functions!

hash(“`www.gmail.com`”)

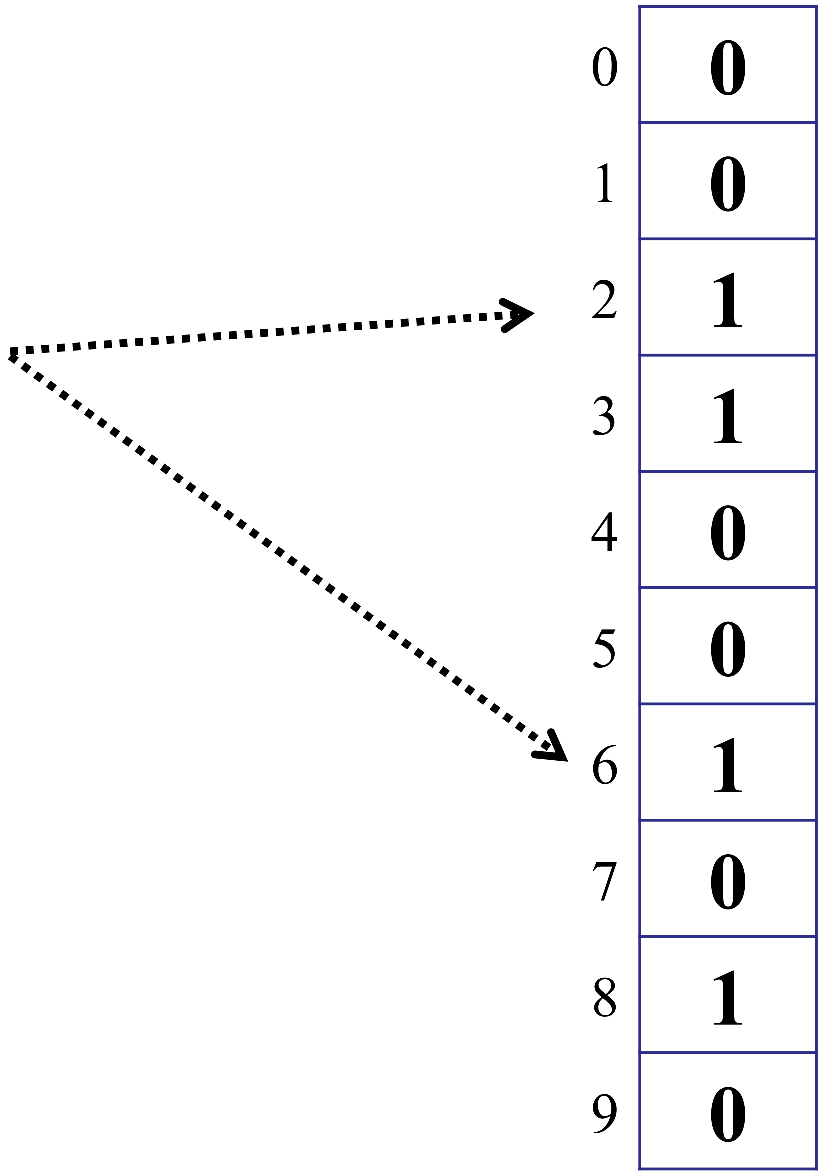
hash(“`www.microsoft.com`”)



Bloom Filter

Idea: use 2 hash functions!

hash(“`www.gmail.com`”)



insert(URL)

$k_1 = \text{hash}_1(\text{URL});$

$k_2 = \text{hash}_2(\text{URL});$

$\text{T}[k_1] = 1;$

$\text{T}[k_2] = 1;$

Bloom Filter

Idea: use 2 hash functions!

query(URL)

k_1 = hash$_1$(URL);

k_2 = hash$_2$(URL);

if (T[k_1] && T[k_2])

return true;

else return false;

0	**0**
1	**0**
2	**1**
3	**1**
4	**0**
5	**0**
6	**1**
7	**0**
8	**1**
9	**0**

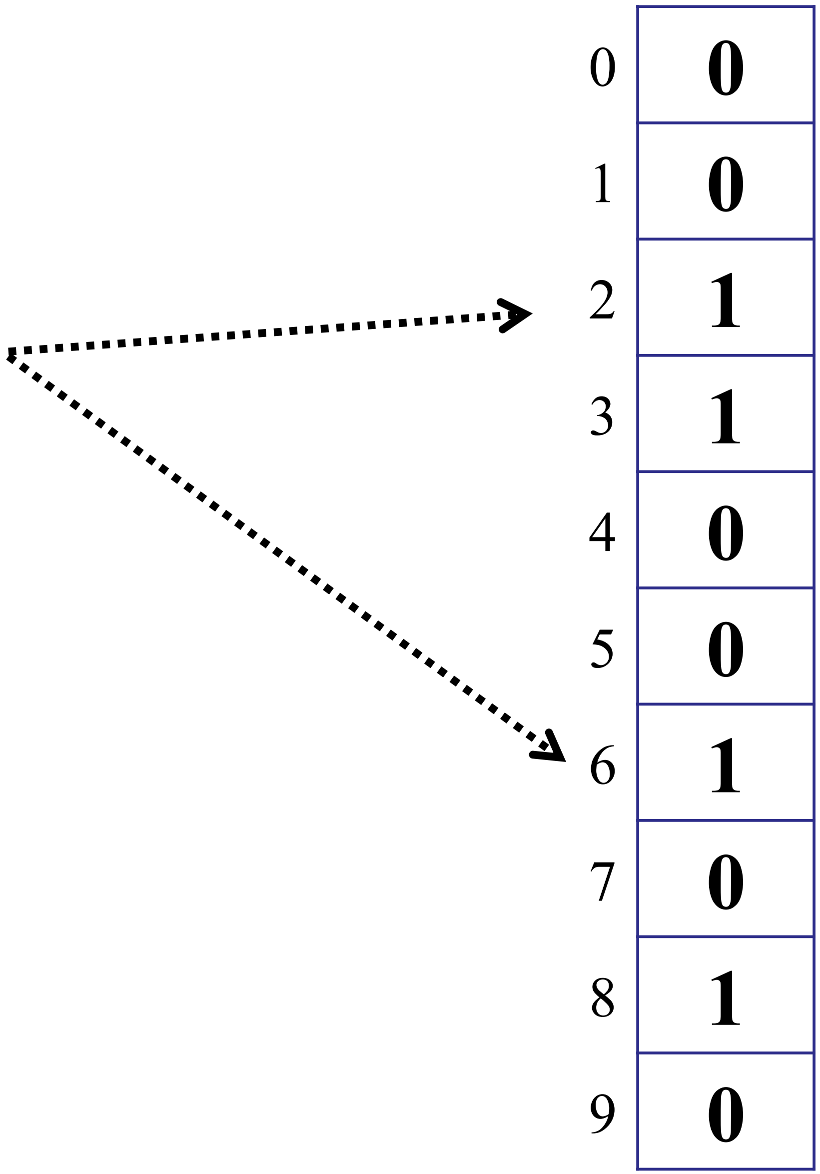
A Bloom Filter can have:

1. Only false positives. ✓
2. Only false negatives.
3. Both false positives and negatives.
4. Wait, which is which again?

Bloom Filter

Idea: use 2 hash functions!

hash(“`www.gmail.com`”)



- No false negatives.
- Possible false positives.

Bloom Filter

Idea: use 2 hash functions!

query(URL)

$k_1 = \text{hash}_1(\text{URL});$

$k_2 = \text{hash}_2(\text{URL});$

if $(T[k_1]$ && $T[k_2])$

return true;

else return false;

0	**0**
1	**0**
2	**1**
3	**1**
4	**0**
5	**0**
6	**1**
7	**0**
8	**1**
9	**0**

Bloom Filter

Idea: use 2 hash functions!

Trade-off:

– Each item takes more “space” in the table.

– Requires <u>two</u> collisions for a false positive.

0	**0**
1	**0**
2	**1**
3	**1**
4	**0**
5	**0**
6	**1**
7	**0**
8	**1**
9	**0**

Bloom Filter Analysis

Probability a given bit is 0:

$$\left(1 - \frac{1}{m}\right)^{2n} \approx \left(\frac{1}{e}\right)^{2n/m}$$

chance hash does not choose this bit

each of n items sets 2 bits in the table

Bloom Filter Analysis

Probability a given bit is 0:

$$\left(1 - \frac{1}{m}\right)^{2n} \approx \left(\frac{1}{e}\right)^{2n/m}$$

Probability of a false positive: (1 set in both slots)

$$\left(1 - \left(\frac{1}{e}\right)^{2n/m}\right)^{2}$$

Bloom Filter Analysis

Probability a given bit is 0:

$$\left(1-\frac{1}{m}\right)^{2n} \approx \left(\frac{1}{e}\right)^{2n/m}$$

Probability of a false positive: (1 set in both slots)

$$\left(1-\left(\frac{1}{e}\right)^{2n/m}\right)^{2}$$

* Assuming BOGUS fact that each table slot is independent…

Bloom Filter Analysis

Assume you want:

- probability of false positives $< p$
  - Example: at most 5% of queries return false positive.

    $$p = .05$$

- Need: $$\frac{n}{m} \leq \frac{1}{2}\log\left(\frac{1}{1-p^{1/2}}\right)$$
  - Example: $m >= (7.9)n$

* Assuming BOGUS fact that each table slot is independent…

Bloom Filter

prob(false positive)

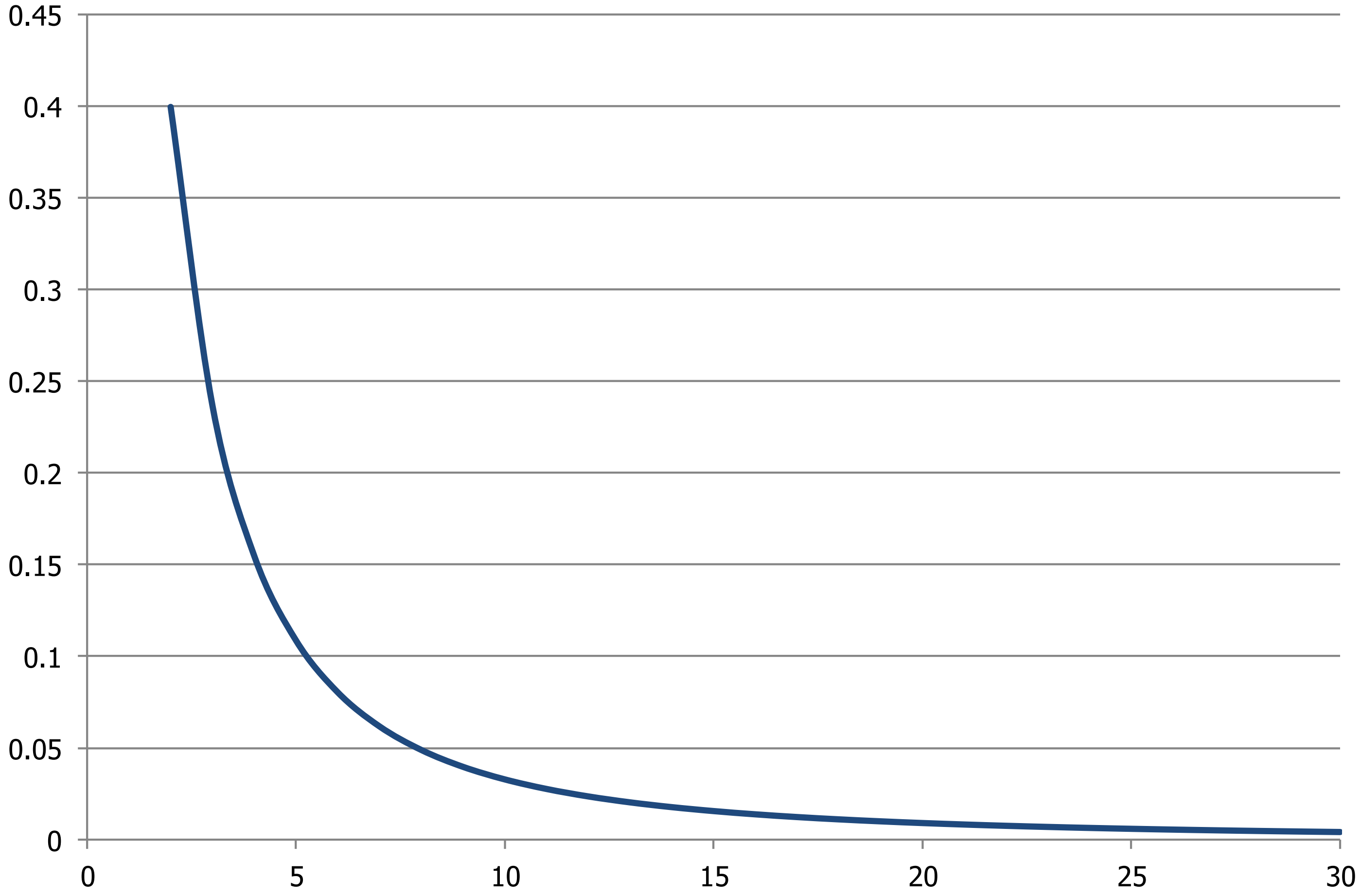


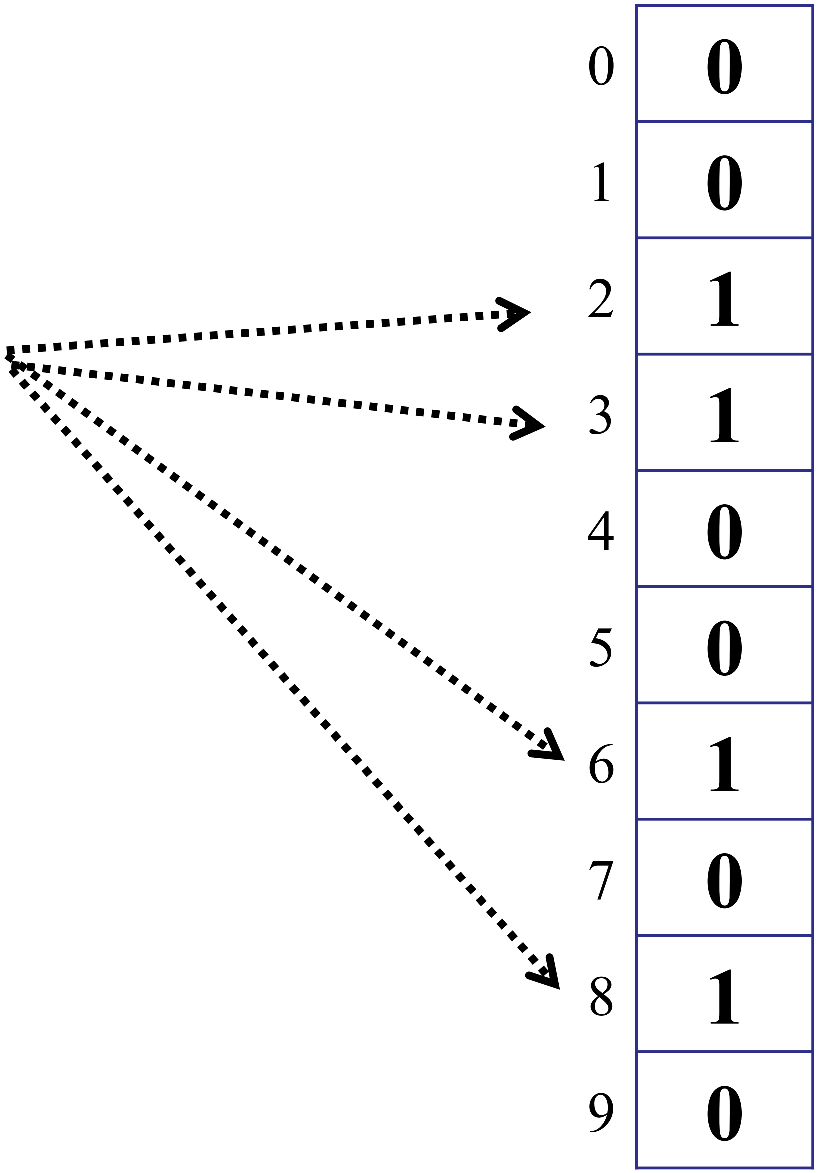
table size (m/n)

False positives rate vs. (m/n)

Bloom Filters

Use *k* hash functions!

hash(“`www.gmail.com`”)



Bloom Filter Analysis

Probability a given bit is 0:

$$\left(1-\frac{1}{m}\right)^{kn} \approx e^{-kn/m}$$

Bloom Filter Analysis

Probability a given bit is 0:

$$\left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m}$$

Probability of a collision at one spot:

$$1 - e^{-kn/m}$$

* Assuming BOGUS fact that each table slot is independent…

Bloom Filter Analysis

Probability of a collision at one spot:

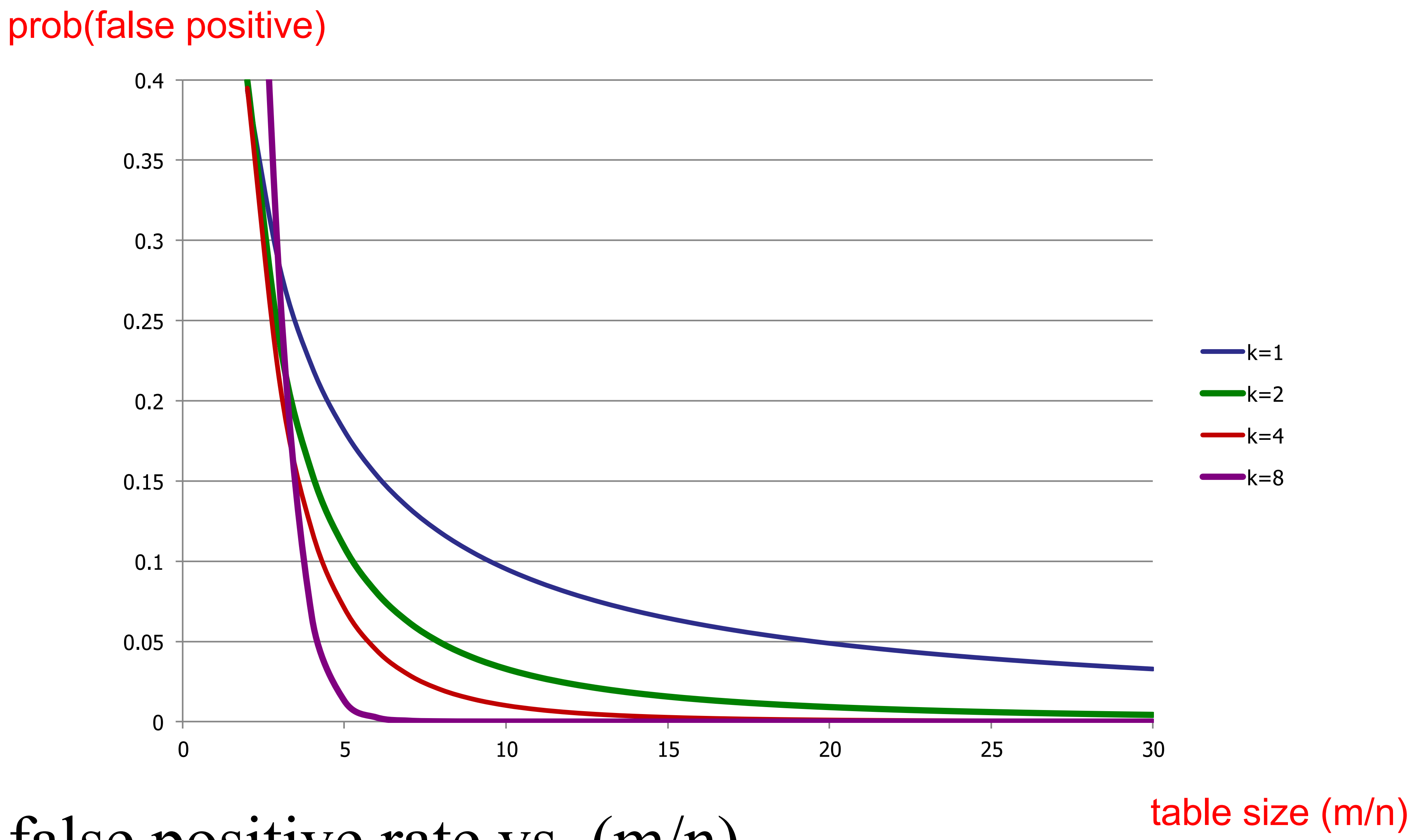
$$1 - e^{-kn/m}$$

Probability of a collision at all k spots:

$$\left(1 - e^{-kn/m}\right)^k$$

* Assuming BOGUS fact that each table slot is independent…

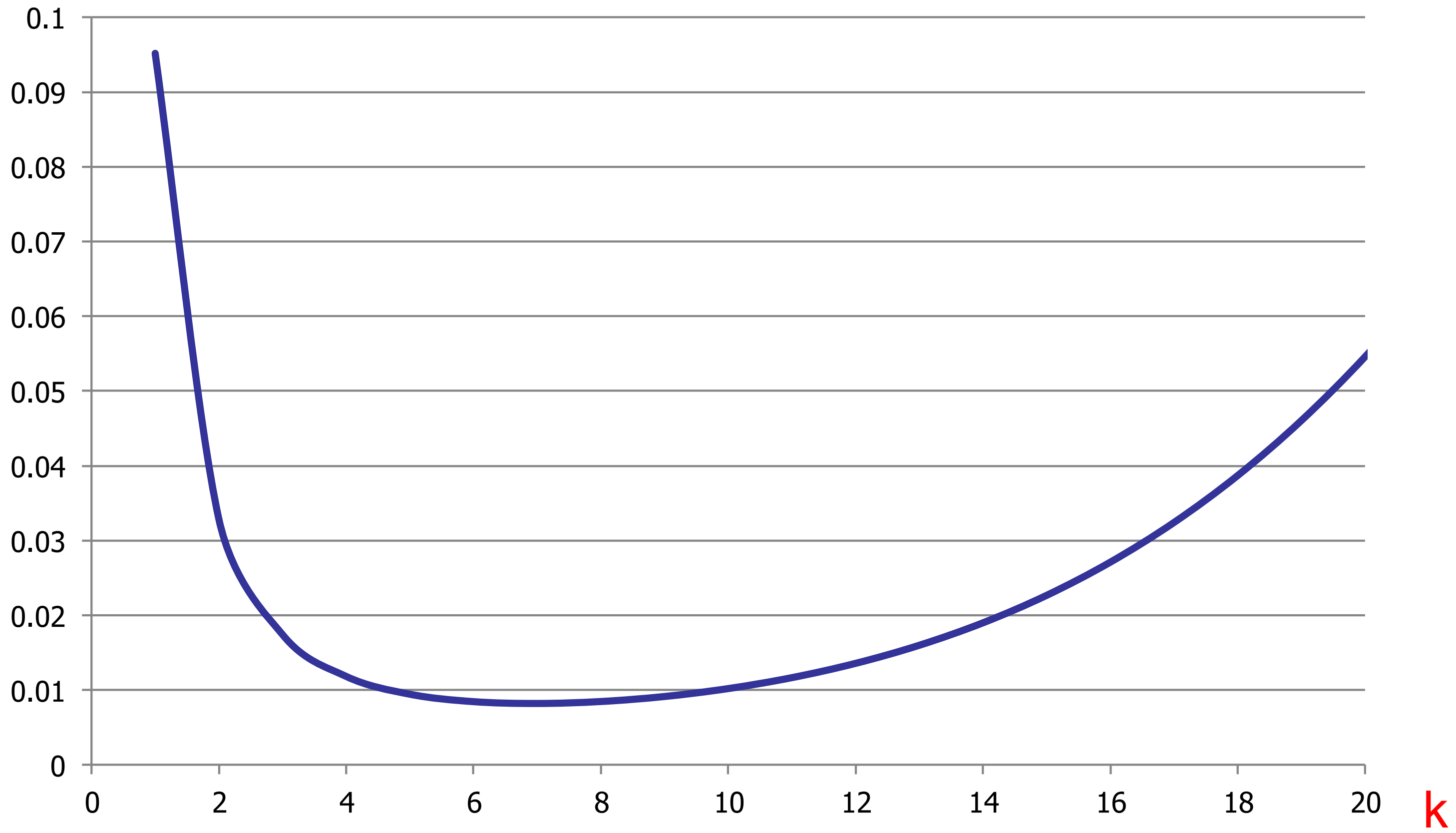
Bloom Filter



false positive rate vs. (m/n)

Bloom Filter

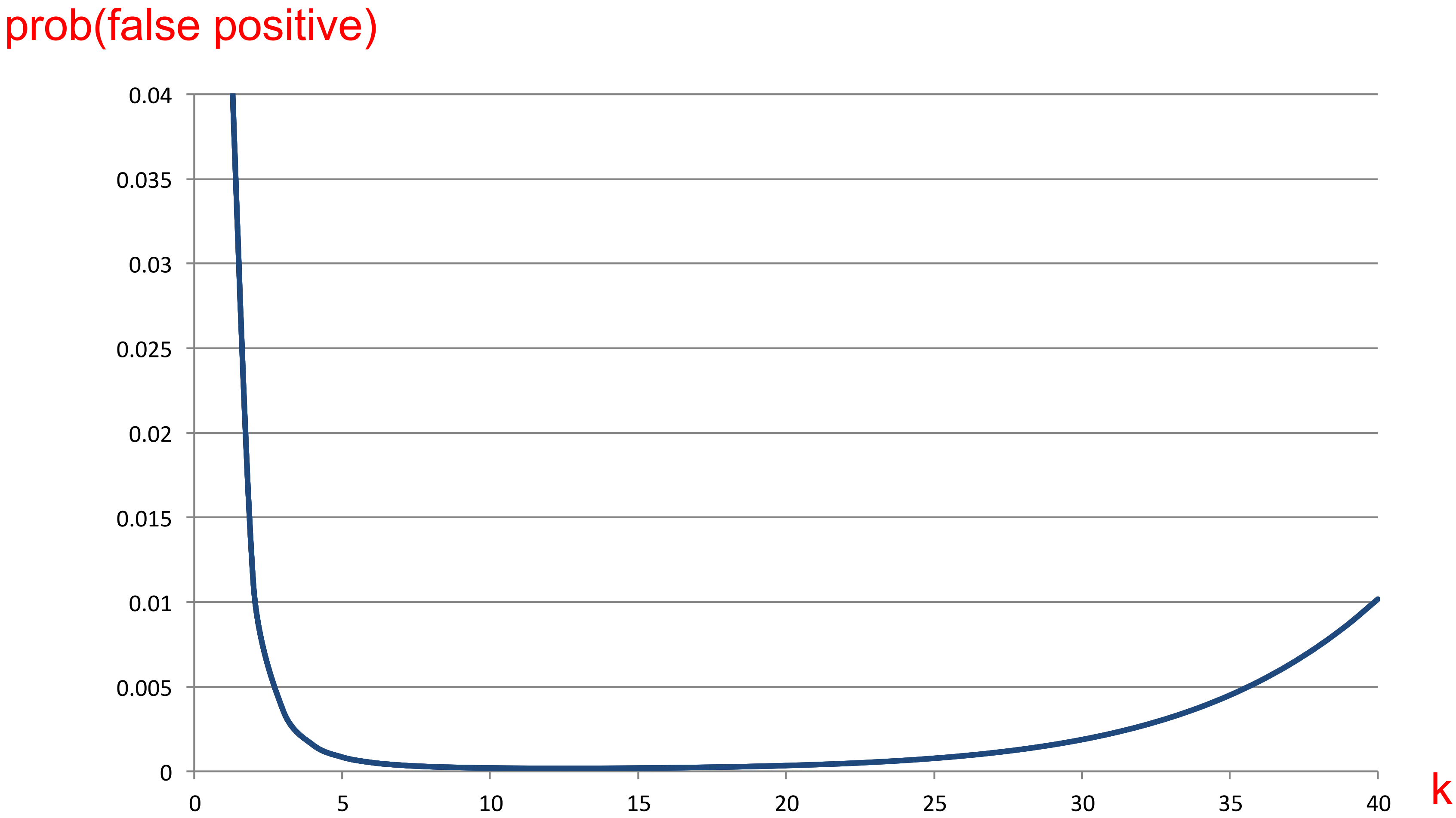
prob(false positive)



false positive rate vs *k*

m = 10n

Bloom Filter



false positive rate vs *k*

m = 18n

Bloom Filter

What is the optimal value of k?

- Probability of false positive:

$$\left(1 - e^{-kn/m}\right)^k$$

- Choose: $k = \frac{m}{n} \ln 2$

- Error probability: 2^{-k}

Summary So Far

- Fingerprint Hash Functions
  - Don’t store the key.
  - Only store 0/1 vector.
- Bloom Filter
  - Use more than one hash function.
  - Redundancy reduces collisions.
- Probability of Error
  - False positives
  - False negatives