Efficient and Scalable Query Routing for Unstructured Peer-to-Peer Networks

Abhishek Kumar  Jun (Jim) Xu  Ellen W. Zegura
Georgia Institute of Technology, 2005
The Problem:

• Searching for content in an unstructured network

Constraints:

• Content and/or structure are highly dynamic
• Any node can originate content (lots of content)
• Limited bandwidth and memory at each node
• No a priori knowledge of the environment
Possible Solutions:

- Flooding
- Random Walk
- Supernodes/Ultrapeers
- One-Hop Replication of Index
- Expanding ring search
- GIA: Optimized Topology Construction, Load Balancing

Problems (trade-offs):

- Speed (low temporal locality in search traffic)
- Scalability (replicating content indices is expensive)
Scalable Query Routing (SQR)
Scalable Query Routing (SQR)

- Maintain probabilistic “routing tables”
- High information about close neighbors
- Information intensity “decays” with distance
- A data-structure at each node to achieve this
- Queries perform a “partially guided” random walk
Scalable Query Routing (SQR)

Information about content on a host decays exponentially with distance.
Bloom Filter

Given a set $S = \{x_1, x_2, x_3, \ldots x_n\}$ on a universe $U$, want to answer queries of the form: 

$\text{does } z \in S$

- Bloom filter answers in “constant” time
- Small amount of space.
- But with some probability of being wrong.
Bloom Filter

Array of $m$ bits all set to 0 initially

$B$  
\[
\begin{array}{cccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

When inserting an element $x$, set $B[h_i(x)] = 1$ for $i = 1$ to $k$

$B$  
\[
\begin{array}{cccccccccccc}
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
\end{array}
\]

To check if $y$ is in $S$, check $B$ at $h_i(y)$. All $k$ values must be 1

$B$  
\[
\begin{array}{cccccccccccc}
0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
\end{array}
\]

May have false positives; all $k$ values are 1, but $y$ is not in $S$

$B$  
\[
\begin{array}{cccccccccccc}
0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
\end{array}
\]
Bloom Filter

Under the assumption:
• Good (pseudo-random) hash functions

Can bound the probability of a false positive and optimize the number $k$ of hash functions to minimize this probability.

Given $n$ objects and a Bloom filter of size $m$:

$$p = \Pr[\text{cell is empty}] = (1 - 1/m)^{kn} \approx e^{-kn/m}$$

$$f = \Pr[\text{false pos}] = (1 - p)^k \approx (1 - e^{-kn/m})^k$$

$k$ that minimizes $f = (\ln 2)m/n$
Exponentially Decaying Bloom Filter (EDBF)

Array of \( m \) bits. Also uses \( k \) hash functions. Insertion is identical to BF.

Testing for membership, returns the number of bits set to 1

\[
\theta(x) = |\{i | A[h_i(x)] = 1, i = 1, 2, \ldots, k\}|
\]

When EDBF is used in the probabilistic query routing in SQR, \( \theta(x)/k \) roughly represents the probability of finding \( x \) along a particular link.
Exponentially Decaying Bloom Filter (EDBF)

- Nodes advertise their EDBF to their neighbors
- Each node keeps separate copies of EDBF received from each of its neighbors
- When advertising to downstream neighbors nodes take the union of local EDBF with EDBFs of the neighbors resetting bits in these with probability $(1/d)$
- Because of the decay, for any object $x$, $\theta(x) = k$ for a node one hop away, $k/d$ two hops away, $k/(d^n)$ $n$ hops away
Exponentially Decaying Bloom Filter (EDBF)

Constructing and updating EDBF:

Create Local EDBF (given local content $X$):
// Populate local EDBF $A$
1. $\forall x \in X$
2. Set bits $A[h_1(x)], \ldots, A[h_n(x)]$ to 1;

Create Update (for neighbor $j$):
// Copy all the bits from the local EDBF $A$ into
// the update $U_j$.
1. $U_j \leftarrow A$
   // Decay the information received from all neighbors
   // other than $j$ by a factor of $d$, and add the
   // surviving bits to $U_j$.
2. $\forall i \in \text{neighbor list}, i \neq j$
3. $\forall r \in \{1, \ldots, m\}$
4. $\text{if}(A_i[r] == 1)$
5. with probability $1/d$, $U_j[r] \leftarrow 1$
6. Return $U_j$

Fig. 2. Algorithms for creating updates in SQR.
Using EDBF for Routing

Local EDBF

0 1 0 0 1 0 0 1 0 0
Using EDBF for Routing

Advertisements Received

Host

0 0 1 0 1 0 0 1 1 0 0
Using EDBF for Routing

Union of received advertisements
Randomly reset half the bits

```
0 0 1 0 1 0 0 0 0 1
```
Using EDBF for Routing

Take union with local EDBF

Send Advertisement to neighbor

0 0 1 0 1 1 0 0 0 0 1
Exponentially Decaying Bloom Filter (EDBF)

Query routing:

• If the query is satisfied locally, it is answered

• If the query has previously been seen, it is forwarded to a random neighbor

• Otherwise the query is forwarded to the neighbor advertising the highest value \( \theta(x) \), the total number of bits set to 1 in locations indexed by \( h_j(x), j \in 1...k \)
Exponentially Decaying Bloom Filter (EDBF)

Query routing:

Forward Query (given query $Y$):

1. if (Seen Query($Y$))
2. Deliver Query($Y, i$);
3. else
4. Forward previously unseen queries to the neighbor
5. with the maximum information about this query
6. Deliver Query($Y, i$);

Lookup (given query $Y$):

1. $\forall i \in \text{neighbor list}$
2. $\forall q \in \{1, \ldots, k\}$
3. $\theta_{i+} = A_i[h_q(y)]$
4. Return $\Theta$; $\Theta = \{\theta_i\}^n$

Fig. 3. Algorithms for forwarding queries in SQR.
Query Routing

Query Host
Exponentially Decaying Bloom Filter (EDBF)

Optimizations:

- Use delta encoding for updates
- Use arithmetic coding for data compression
  - increasing the size of the array while reducing the number of hash functions slightly can improve the efficiency of BF
SQR Performance: Flat Topologies

![Graph showing SQR performance with respect to Hop Limit and % of queries answered.]

- SQR
- OHR
- Random walk
- Flooding
SQR Performance: Flat Topologies
SQR Performance: Hierarchical Topologies
SQR Performance: Impact of Replication

![Graph showing the impact of replication on SQR performance. The x-axis represents the number of copies of each object (replication rate in %), and the y-axis represents the hop limit for 90% query success. The graph compares OHR, GIA, Ultrapeer, and SQR+GIA.](image-url)
SQR Performance: Impact of Replication with Zipf distribution
Conclusions:

• Highly compressed information about content in the neighborhood cab speed up the routing

• Exponential decay of information with distance ensures scalability of the approach

• Probabilistic routing information can be “reliable” and efficient
Problems:

• Deleting content is unsupported in Bloom filters (could be done in EDBF due to probabilistic nature)

• In a large sensor network, random walks may be highly inefficient

• Hashing may be too time/energy expensive for simple nodes
Thank You