



Efficient Peer-to-Peer Keyword Searching

Patrick Reynolds and Amin Vahdat

presented by Volker Kudelko

Overview

1. Introduction
2. Some techniques
3. Simulator design
4. Analysing of the techniques in the simulator
5. Conclusion

1. Introduction

- recent file storage applications built on top of peer-to-peer distributed hash tables do not contain good search capabilities
- but search is a very important part!
- → design and analyze a distributed search engine based on a distributed hash table.

1. Introduction

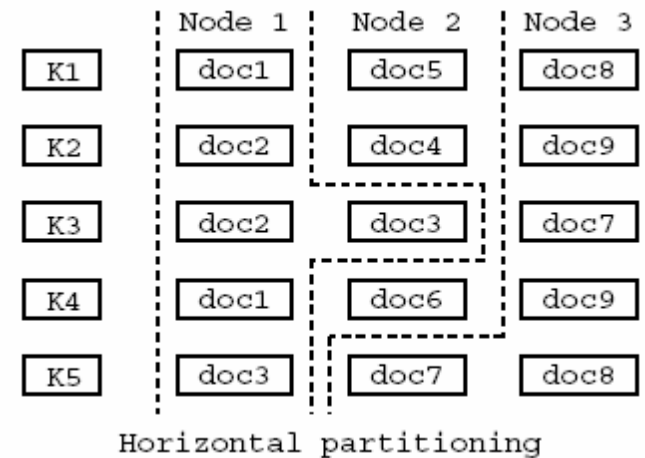
- what's important for a good search system? → end user latency!
- comes mostly from network transfer times
- → try to reduce the number of bytes sent
- most queries contain several keywords
→ minimizing the network traffic for multiple-keyword-search

2. Some techniques

- partitioning
- centralized or distributed design
- bloom filters
- caches
- incremental results
- virtual hosts

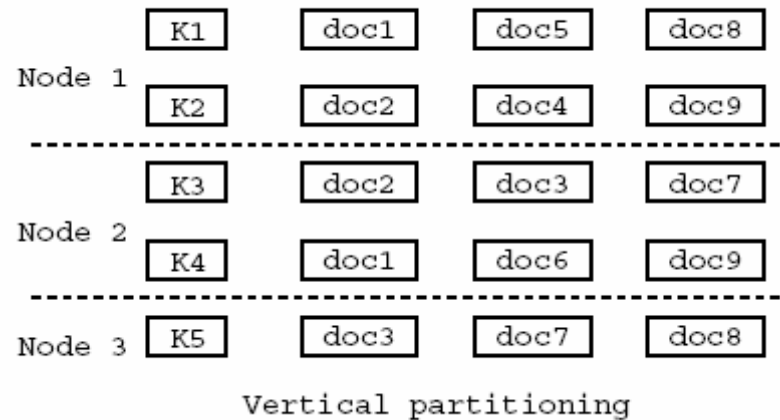
2.1 Partitioning

- horizontal vs. vertical
- **horizontal:** documents are assigned to exactly one node, which stores the dokument's keywords
- **benefit:** to insert or update a dokument with n keywords requires to contact just one single node (vertical needs to contact all servers)



2.1 Partitioning

- **vertical:** one or more keywords are assigned to exactly one node which stores a match-list of the documents containing these keywords
- **benefit:** a search containing k keywords ensures that not more than k servers must be contacted (horizontal needs to contact all servers)



2.1 Partitioning

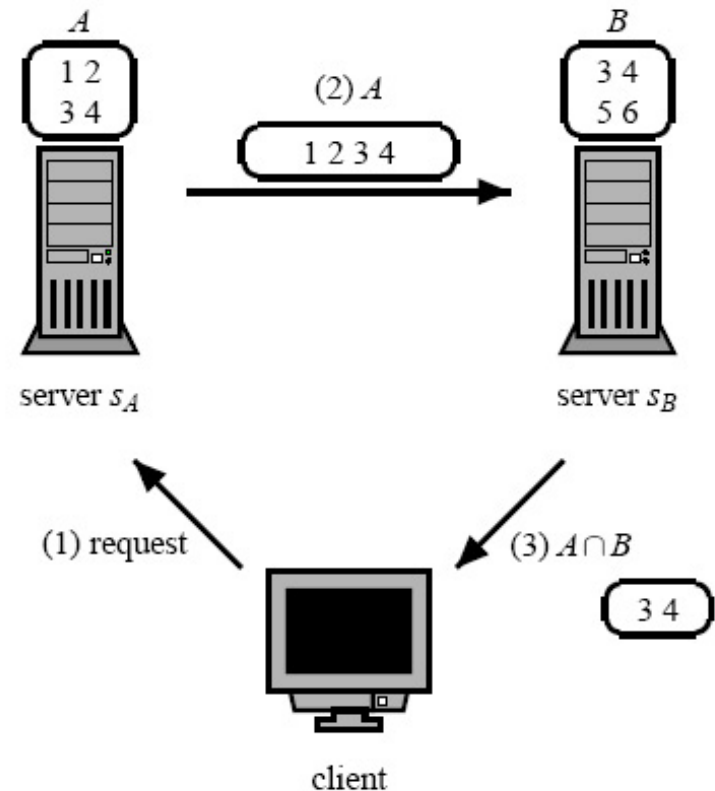
- so far so good... what to use?
- because of the design of an archival storage system, we assume that most files change rarely and those which change do this very often
- so, we believe that queries will outnumber updates
- → we choose the advantages of a **vertically partitioned** system for our search engine

2.2 Centralized or distributed

- distributed search services provide more benefits than a centralized one:
- no single point of failure
- less risk of
 - network outages,
 - denial-of-service attacks and
 - censorship by domestic or foreign authorities
- more replication in distributed systems → better availability

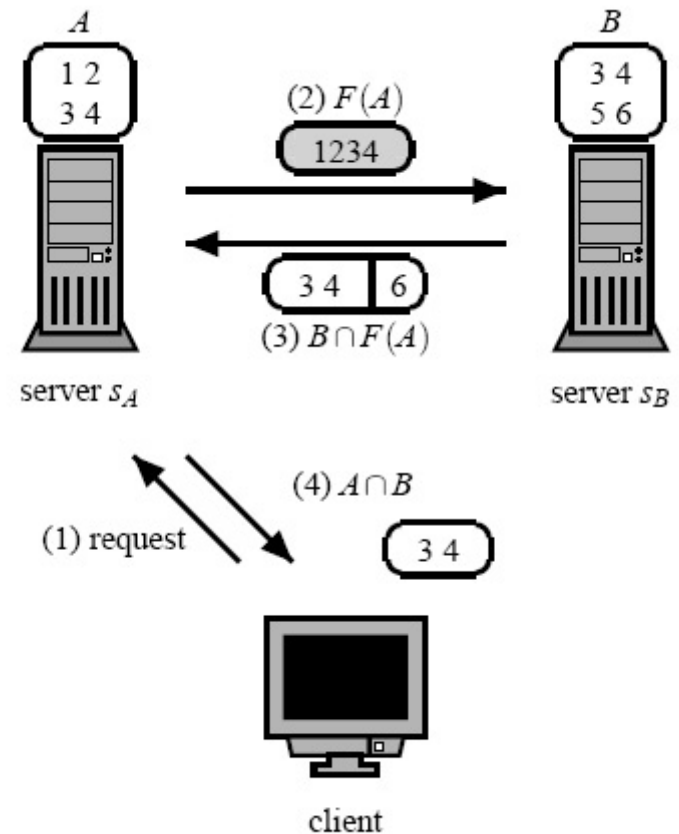
2.3 Bloom filters

- what is the opportunity to use them?
- first take a look at a simple „AND“ query without bloom filters
- s_A sends the list of documents with keyword A to s_B
- s_B sends the intersection $A \cap B$ to the client



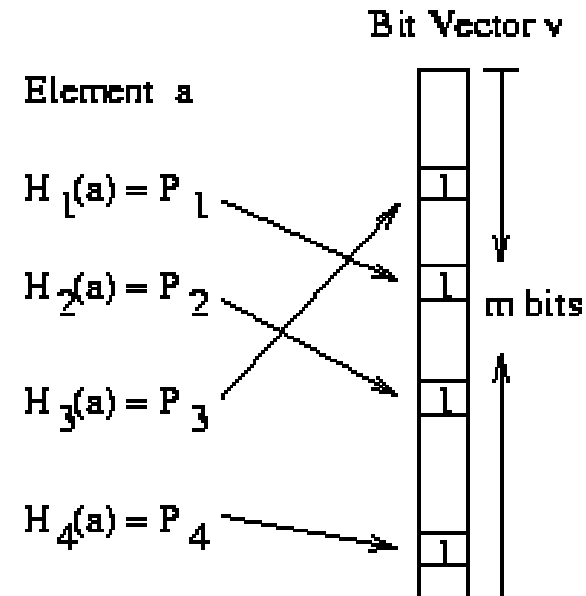
2.3 Bloom filters

- now – what is the situation with bloom filters?
- server s_A sends instead of A a bloom filter $F(A)$
- server s_B calculates $B \cap F(A)$
- server s_A performs then the intersection of A and $B \cap F(A)$ to remove the false positives
- the results are sent to the client
- consequence: decrease in the number of excess bits send
- but additional step needed for removing the false positives



2.3 Bloom filters – a closer look

- a method for representing a set of n elements to support membership queries by using a hash-based data structure
- allocate a vector v of m bits, initially all set to 0
- choose k independent hash functions, with range $1 \dots m$
- bits at positions $h_1(a), \dots, h_k(a)$ in v are set to 1
- given a query for b , check the bits at positions $h_1(b), \dots, h_k(b)$. If any of them are 0, then b is not in the set A .
- otherwise we assume that b is in the set, although this could be wrong (false positive)
- parameters k and m should be chosen so that the probability of a false positive is acceptable.



2.3 Bloom filters

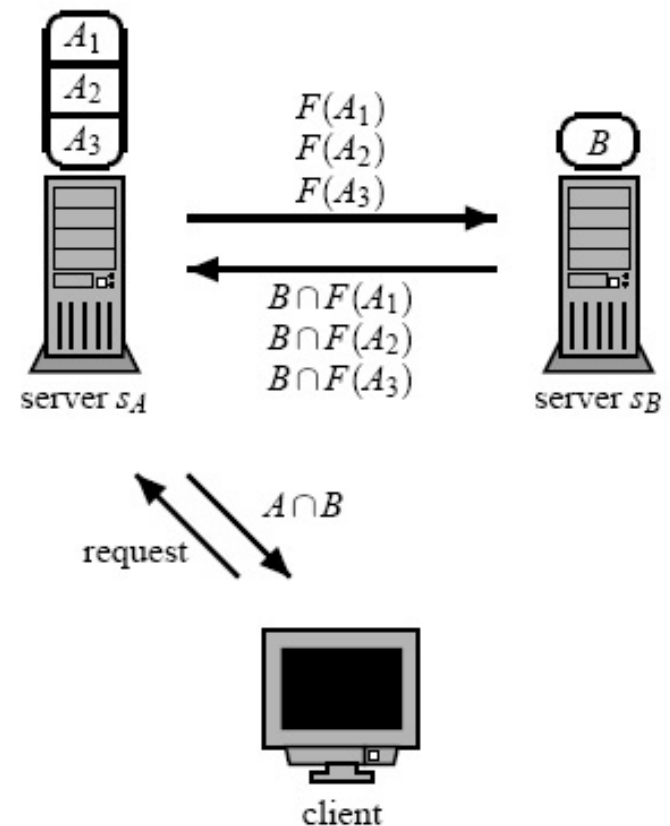
- number of excess bit send example:
 - A and B contain 10.000 documents in their list
 - each document identifier has 128 bits
- **without** bloom filters: $10,000 \cdot 128 = 1,280,000$ bits
- **with** bloom filters
 - probability of false positives: $p_{fp} = 0.6185^{m/n}$
m = # bits in the boom filter, n = # elements in the set
 - → # of excess bits sent: $m + p_{fp} |B| j = m + 0.6185^{m/|A|} |B| j$
 - → computing optimal bloom filter size m (derivation set to 0 and solve for m) gives:
$$m = |A| \log_{.6185} \left(2.081 \frac{|A|}{|B| j} \right)$$
 - → # of excess bits send with bloom filters = 106,544
 - → represents a compression of 12.01 : 1 !

2.4 Caches

- server locally store the data they receive like A or $F(A)$
- better caching bloom filters
 - → they are smaller
 - → possible to store data for more keywords
- improvements in the hit rate
 - → reduce the minimum of excess bits

2.5 Incremental results

- the number of results is proportional to the number of documents in the network
 - also the bandwidth costs grow linearly with the network size
 - bloom filters and caches provide a substantial cost improvement but cannot eliminate the linear growth in cost
- solution: truncate the results
 - divide the set A in chunks and send several bloom filters until a desired number of results is achieved



2.5 Incremental results

■ advantage:

- caches can store several fractional bloom filters
- servers can retain or discard partial entries in the cache
- storing only a fraction reduces the amount of space

■ disadvantage:

- Higher CPU load for processing the search
(costs for $B \cap F(A_i)$ are equal to $B \cap F(A)$)


2.6 Virtual hosts

- a problem of peer-to-peer systems is the heterogeneity, i.e. there are several different powerful machines
- can happen that an individual machine is assigned to an inappropriate number of keywords
- idea: using of virtual hosts
- each machine gets a „power“ index proportional to its processing capacity and referring to the baseline host
- e.g. machine A gets index 10, i.e. it is ten times more powerful than the baseline host and gets 10 virtual IDs
- power index refers to the CPU power, network bandwidth and memory

3. Simulator Design

- Java application
- 1.85 GB HTML data with 1.17 million unique words in 105,953 documents
- 95,409 searches performed with 43,344 unique keywords
- testing 3 distributions
 - Modem
 - Backbone links
 - Gnutella based network (heterogeneous)
- storage range from 1 to 100 MB
- hosts are placed at random in a 2,500 miles square-grid

3. Simulator Design

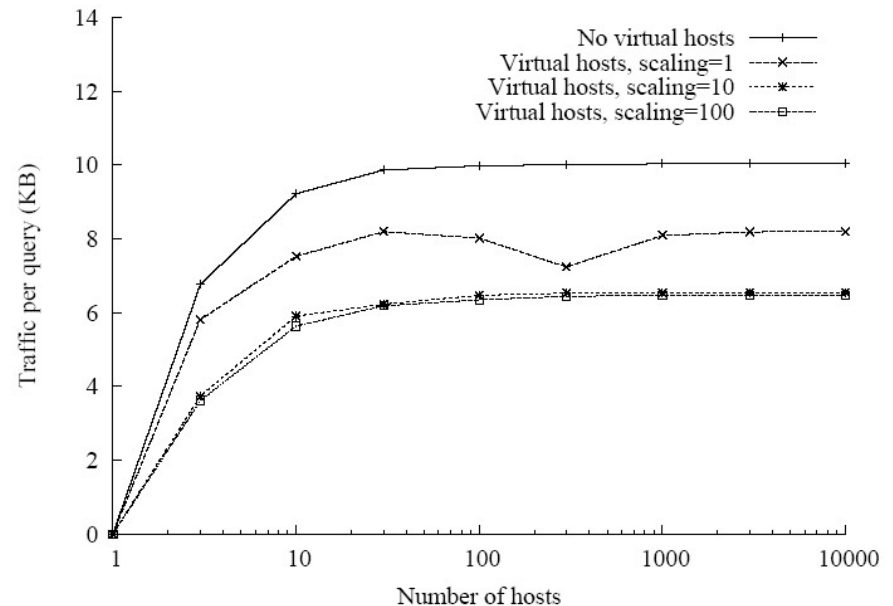
- performing a query:
 - obtaining up to M results for each keyword
 - each host intersects its set with the data from the previous one
 - forward to the subsequent host
 - order given by the number of documents found (few  many)
 - use of a bloom filter depends on the size of the set (threshold)
 - each node that sent a bloom filter is a potentially candidate to remove „false positives“
 - if the resulting documents are less than desired, m is increased adaptively

4. Analysing of the techniques in the simulator

- scalability and Virtual Hosts
- bloom filters
- caching
- putting it all together

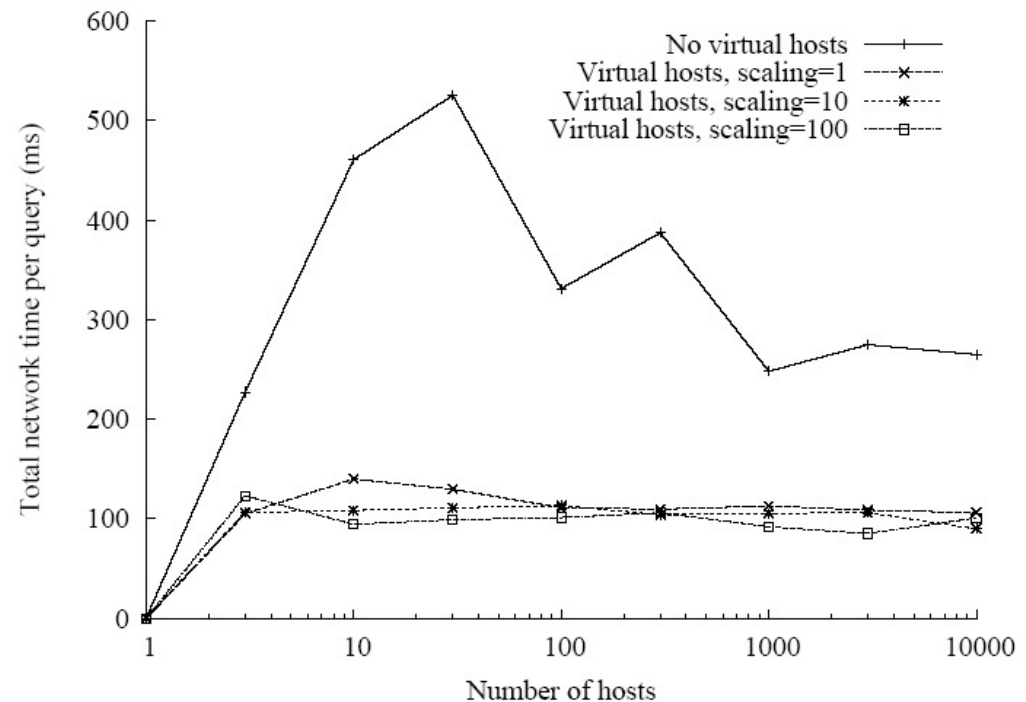
4.1 Scalability and virtual hosts

- goal: reducing the bytes sent per request → minimizing the end-to-end latency
- virtual host concentrate popular data mostly powerful machines
- → higher probability to handle queries entirely locally
- → only a small effect on bytes send per query, but..



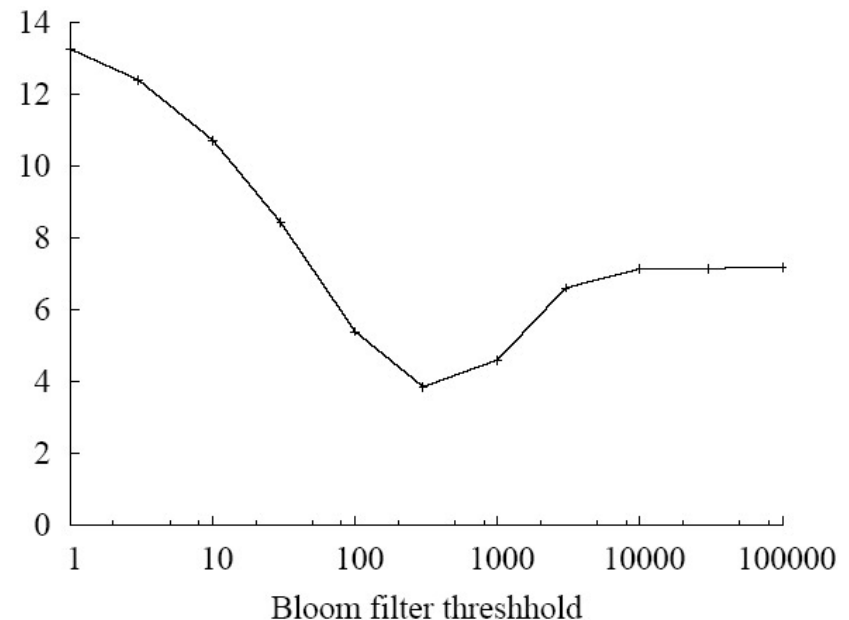
4.1 Scalability and virtual hosts

- ... a much greater effect on network time per query
- no bottleneck with virtual hosts
- better load balancing, faster machines handle a larger fraction of requests



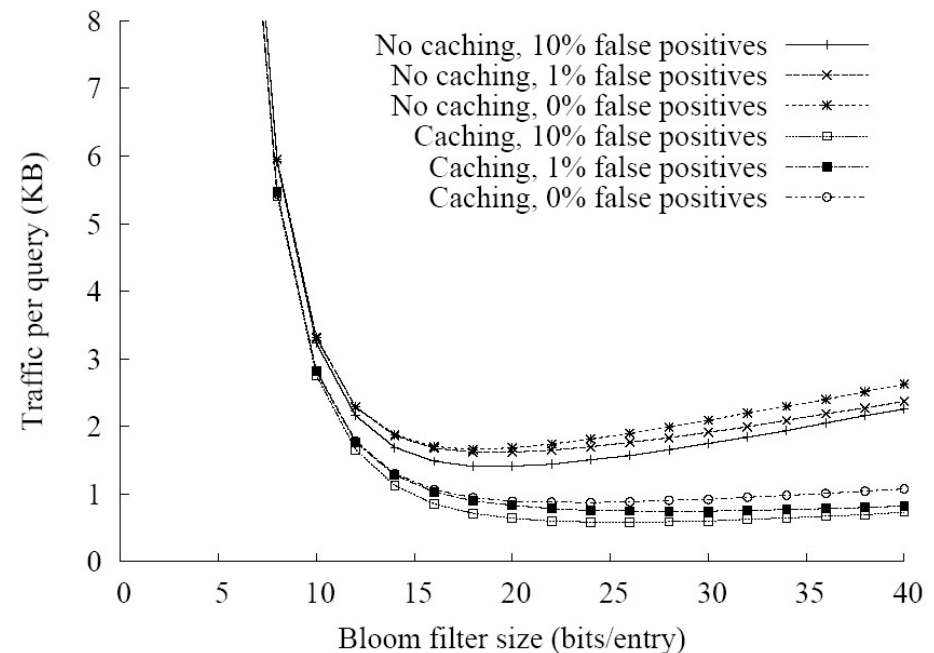
4.2 Bloom filters

- **problem:** using bloom filters always → unnecessary data transfers (eliminate „false positives“)
- **solution:** use them only when the time saved outweighs the time sending clean-up messages
- what is the optimal threshold for bloom filters?
- → threshold approximately 300



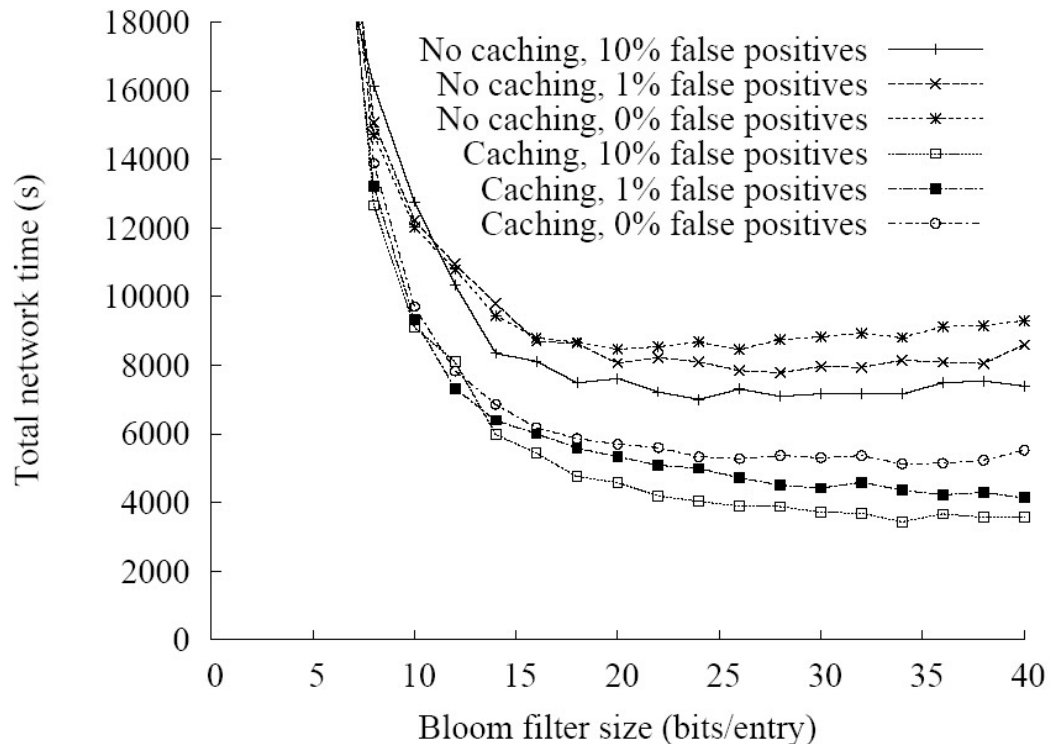
4.3 Caching

- what are the real benefits of caching?
- allowing only „false positives“ rates of 0%, 1% and 10 % in the returning documents
- the lower the false positives rates (additional removing steps needed), the higher the network traffic
- → only a small effect on bytes send per query, but..



4.3 Caching

- ... a much greater effect on network time per query
- allowing more „false positives“ causes eliminating a number of required message transmissions



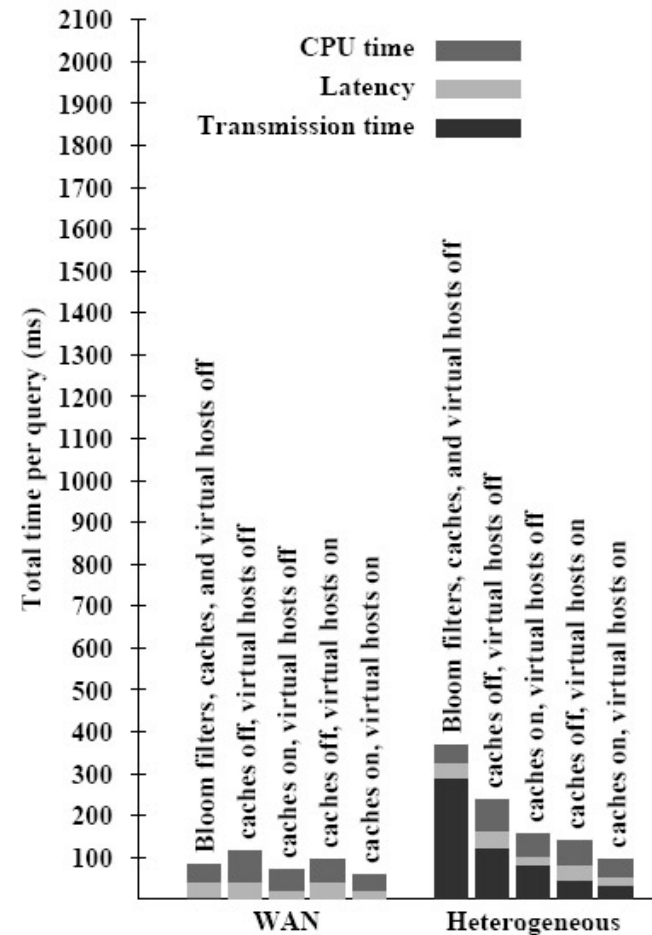
4.4 Putting it all together

splitting the end-to-end time in 3 components:

- CPU processing time
 - network transmission time (bytes transferred divided by the network connection speed of the slowest host)
 - latency (time required for a bit to travel through the line)
- analysing under 3 network conditions
- WAN
 - heterogeneous
 - modem

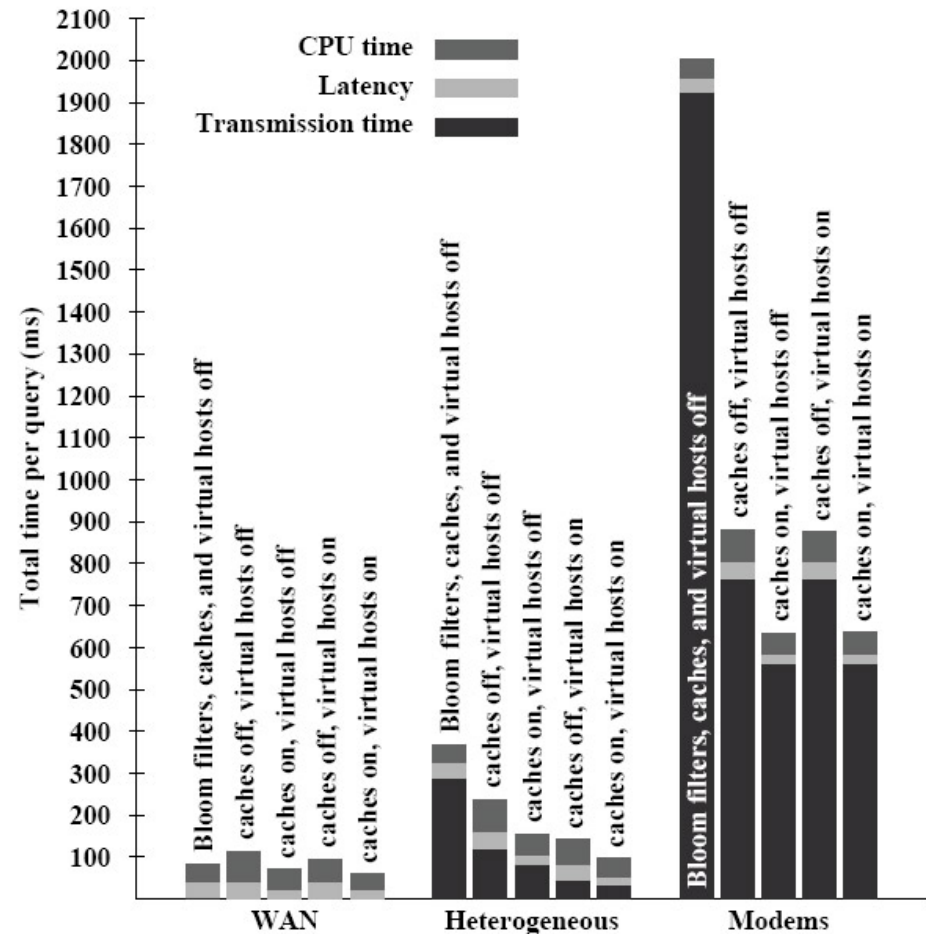
4.4 Putting it all together

- heterogeneous
 - the use of virtual host and caching together has the most powerful effect
 - using virtual hosts → keywords concentrating on nodes with high network bandwidth
- WAN:
 - transmission time negligible (very high transmission speeds)
 - caching reduces CPU time by avoiding to calculate and to transmit Bloom filters
 - virtual hosts also reduce CPU time, requests are concentrated to more powerful nodes



4.4 Putting it all together

- modem
 - end-to-end query time dominated by transmission time
 - no effect of virtual hosts (homogeneous)



5. Conclusion

- the use of virtual hosts and caching together has the most pronounced effect on the heterogeneous network, → reducing average query response times by 59%
- in particular, the use of virtual hosts reduces the network transmission portion of average query response times by 48% by concentrating keywords on the subset of nodes with more network bandwidth
- caching uniformly reduces all aspects of the average query time, in particular reducing the latency components by 47% in each case by eliminating the need for a significant portion of network communication

■ thank you for your attention!

■ questions?

