Building Content-Based Publish/Subscribe Systems with Distributed Hash Tables

David Tam, Reza Azimi, Hans-Arno Jacobsen

University of Toronto, Canada

September 8, 2003

Introduction: Publish/Subscribe Systems

- Push model (a.k.a. event-notification)
  - subscribe → publish → match
- Applications:
  - stock-market, auction, eBay, news
- 2 types
  - topic-based: ≈ Usenet newsgroup topics
  - content-based: attribute-value pairs
    - e.g. (attr1 = value1) ∧ (attr2 = value2) ∧ (attr3 > value3)

The Problem: Content-Based Publish/Subscribe

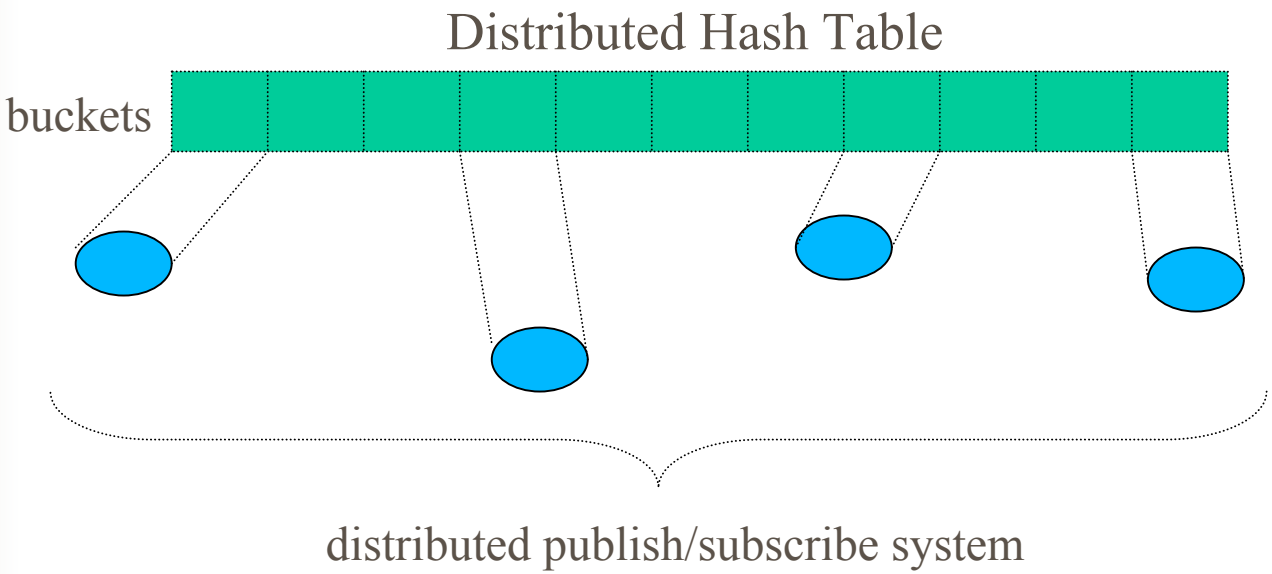
- Traditionally centralized
  - scalability?
- More recently: distributed
  - e.g. SIENA
    - small set of brokers
    - not P2P
- How about fully distributed?
  - exploit P2P
  - 1000s of brokers

Proposed Solution: Use Distributed Hash Tables

- DHTs
  - hash buckets are mapped to P2P nodes
- Why DHTs?
  - scalable, fault-tolerance, load-balancing
- Challenges
  - distributed but co-ordinated and light-weight:
    - subscribing
    - publishing
    - matching is difficult

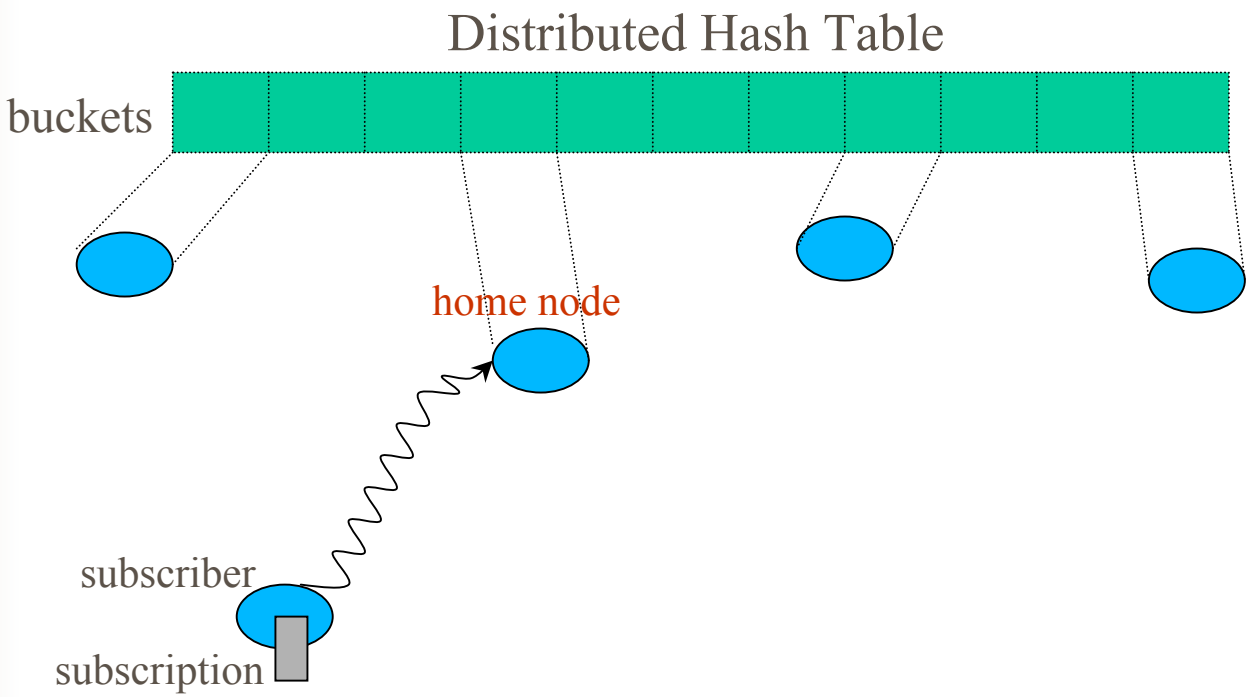
Basic Scheme

- A matching publisher & subscriber must come up with the same hash keys based on the content



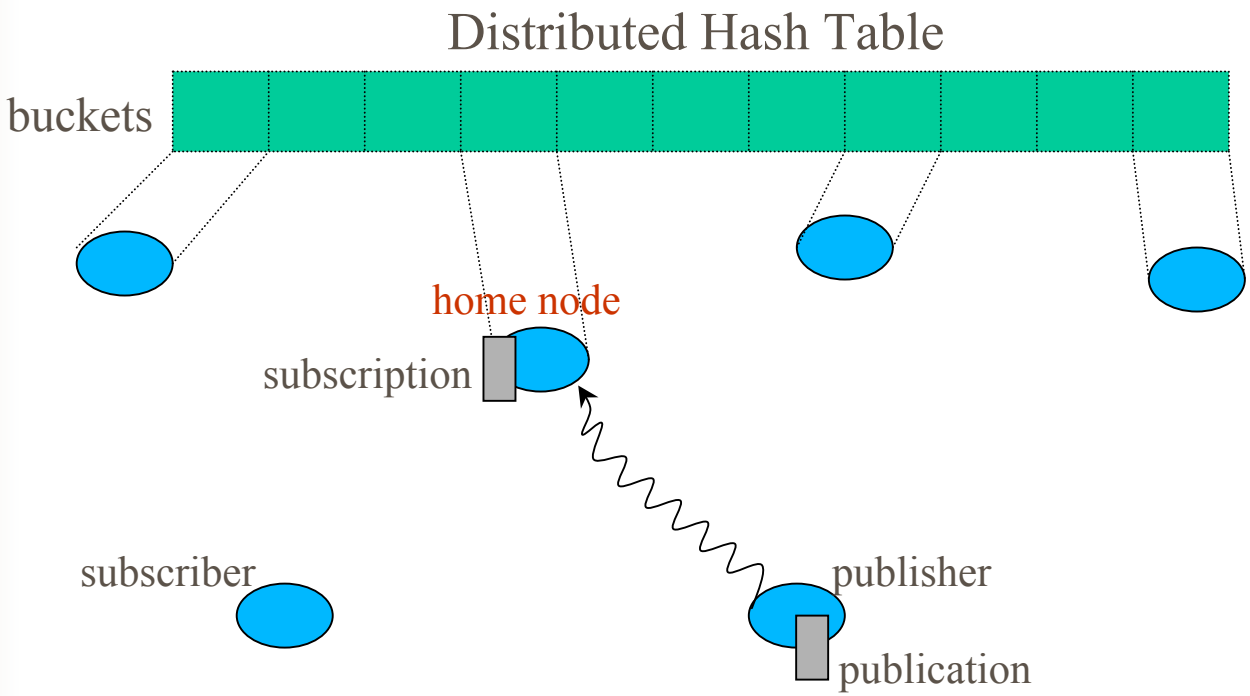
Basic Scheme

- A matching publisher & subscriber must come up with the same hash keys based on the content



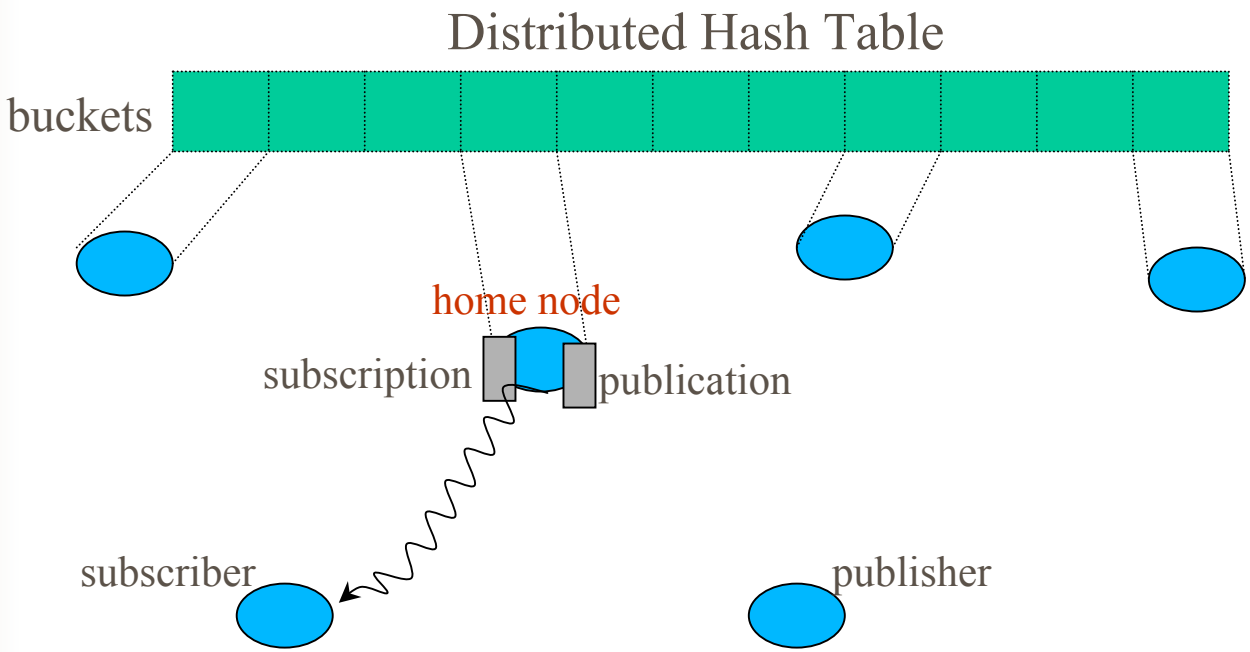
Basic Scheme

- A matching publisher & subscriber must come up with the same hash keys based on the content



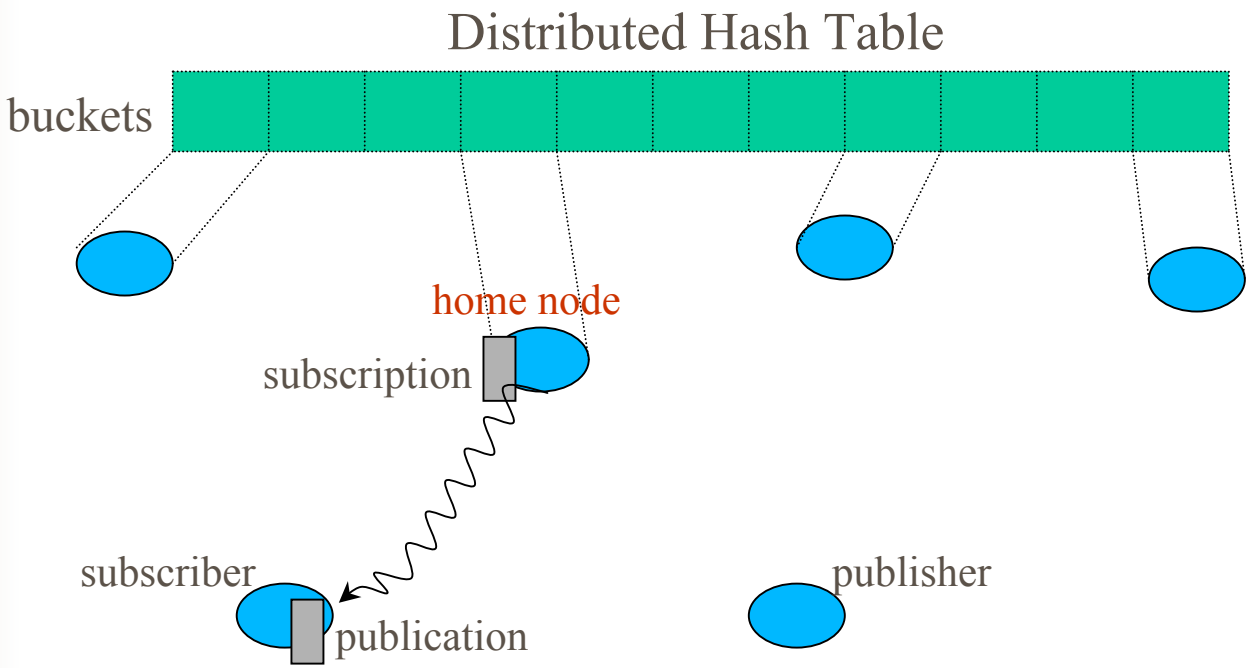
Basic Scheme

- A matching publisher & subscriber must come up with the same hash keys based on the content

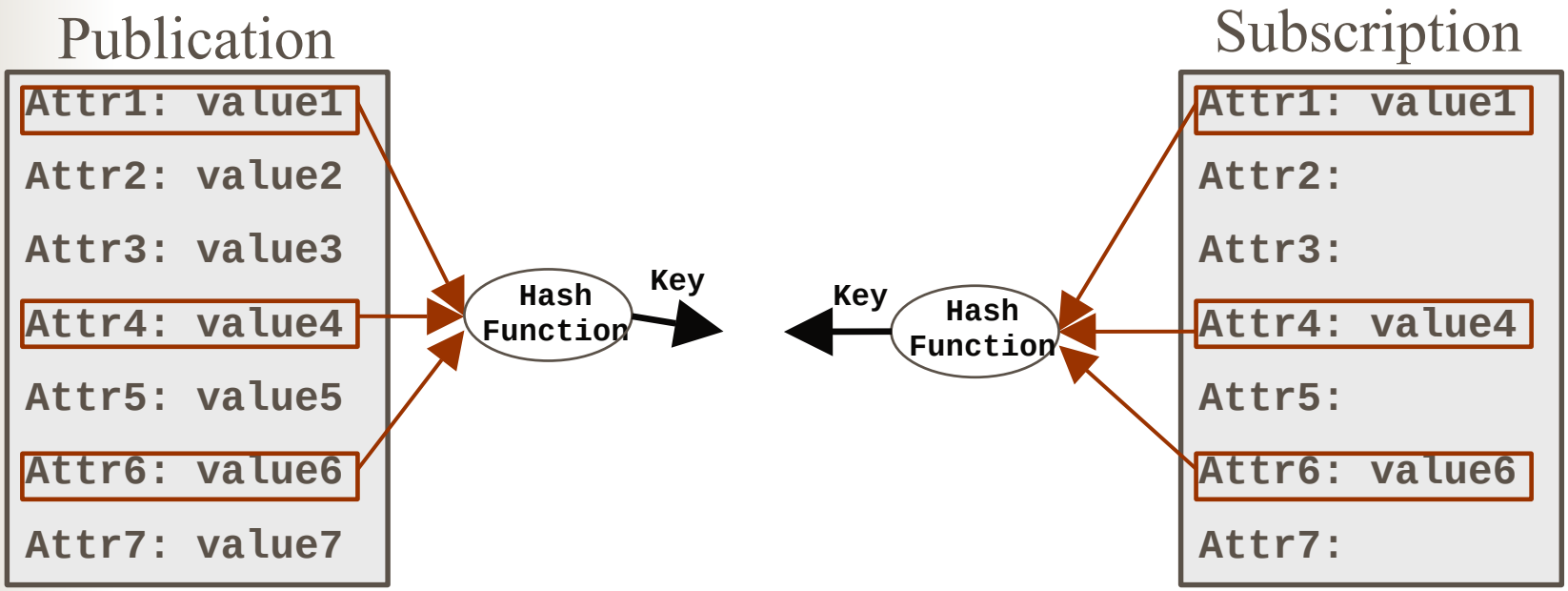


Basic Scheme

- A matching publisher & subscriber must come up with the same hash keys based on the content



Naïve Approach



- Publisher must produce keys for all possible attribute combinations:
  - 2^N keys for each publication
- Bottleneck at hash bucket node
  - subscribing, publishing, matching

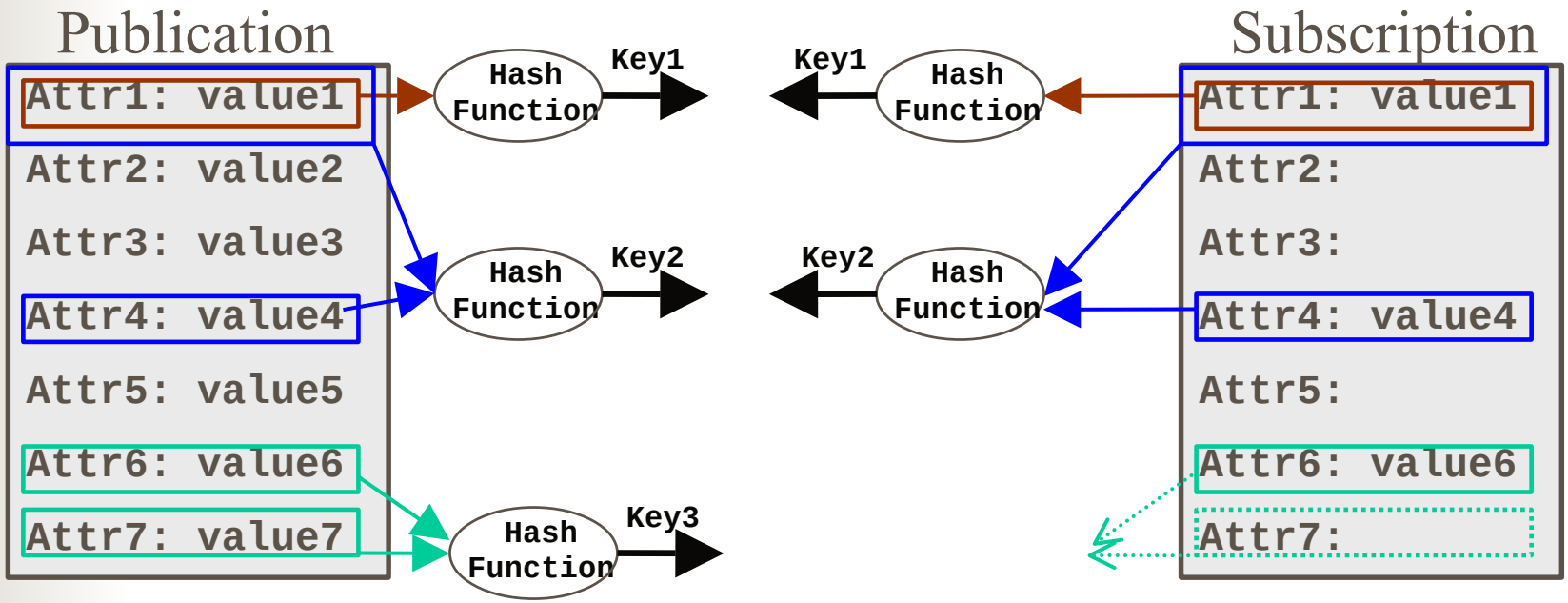
Our Approach

- Domain Schema
  - eliminates 2^N problem
  - similar to RDBMS schema
  - set of attribute names
  - set of value constraints
  - set of indices
    - create hash keys for indices only
    - choose group of attributes that are common but combination of values rare
  - well-known

Order	Type/Unit	Name	Values	Index 1	Index 2	Index 3
1	USD	Price	1000 .. 5000	√		
2	String	CPU	PII, PIII, P4, Celeron		√	√
3	MHz	Clock	200 .. 3000		√	
4	Mbyte	RAM	64 .. 1024		√	
5	Gbyte	HDD	10.. 200			√
6	Inch	Monitor	14 .. 20	√		
7	String	CD	CDROM, CDRW, DVD			√
8	String	Quality	New, Used, Demo	√		

Hash Key Composition

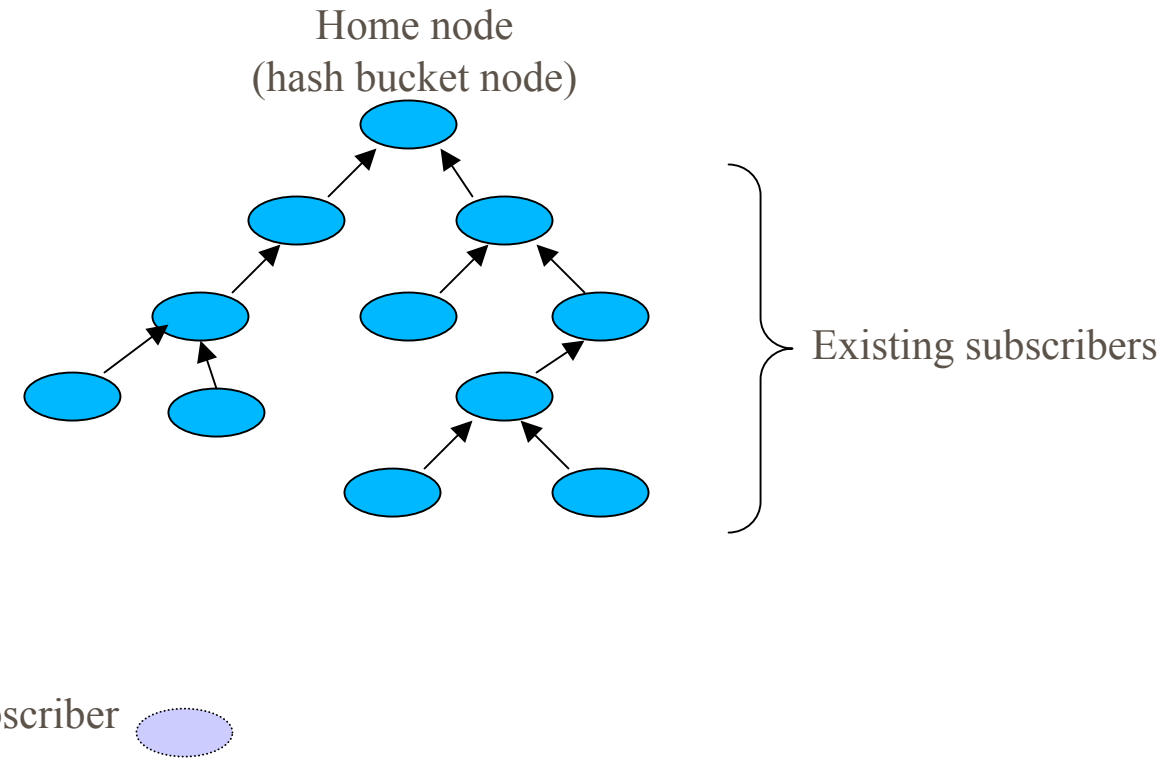
- Indices: {attr1}, {attr1, attr4}, {attr6, attr7}



- Possible false-positives
  - because partial matching
  - filtered by system
- Possible duplicate notifications
  - because multiple subscription keys

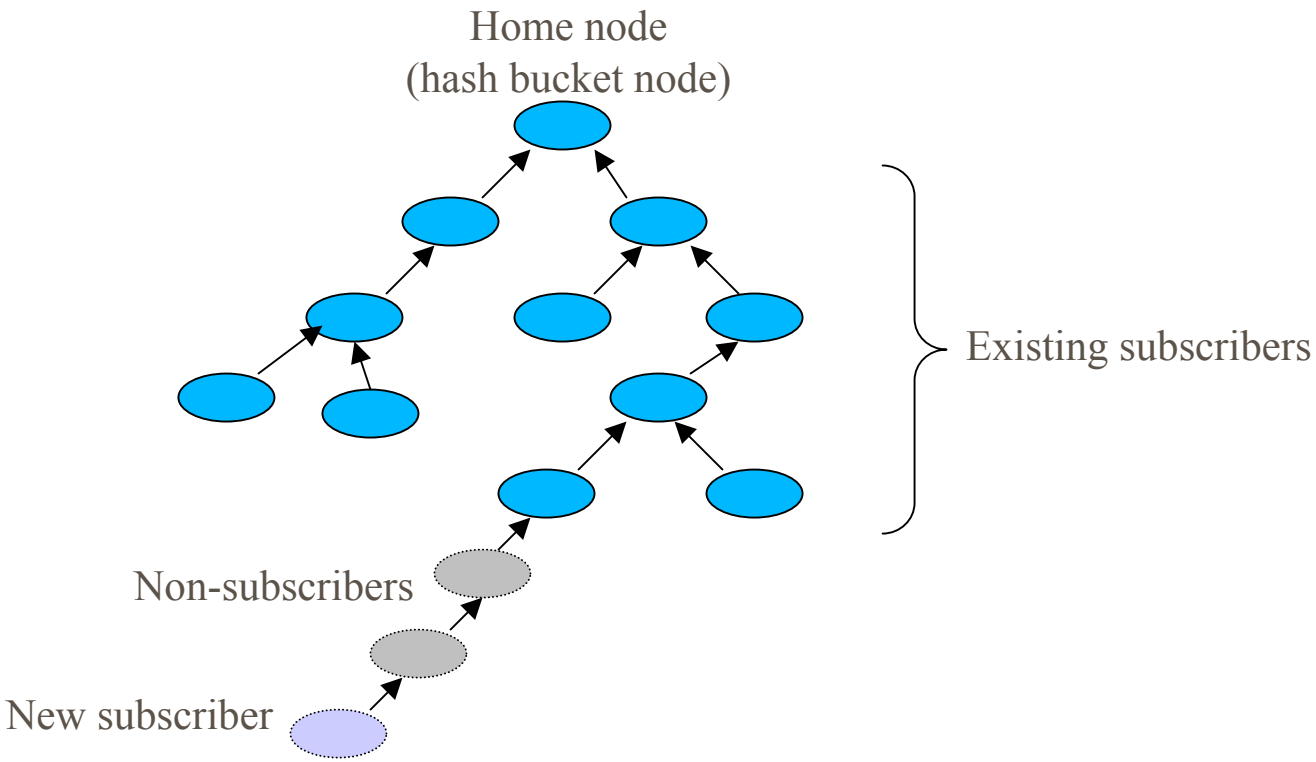
Our Approach (cont'd)

- Multicast Trees
  - eliminates bottleneck at hash bucket nodes
  - distributed subscribing, publishing, matching



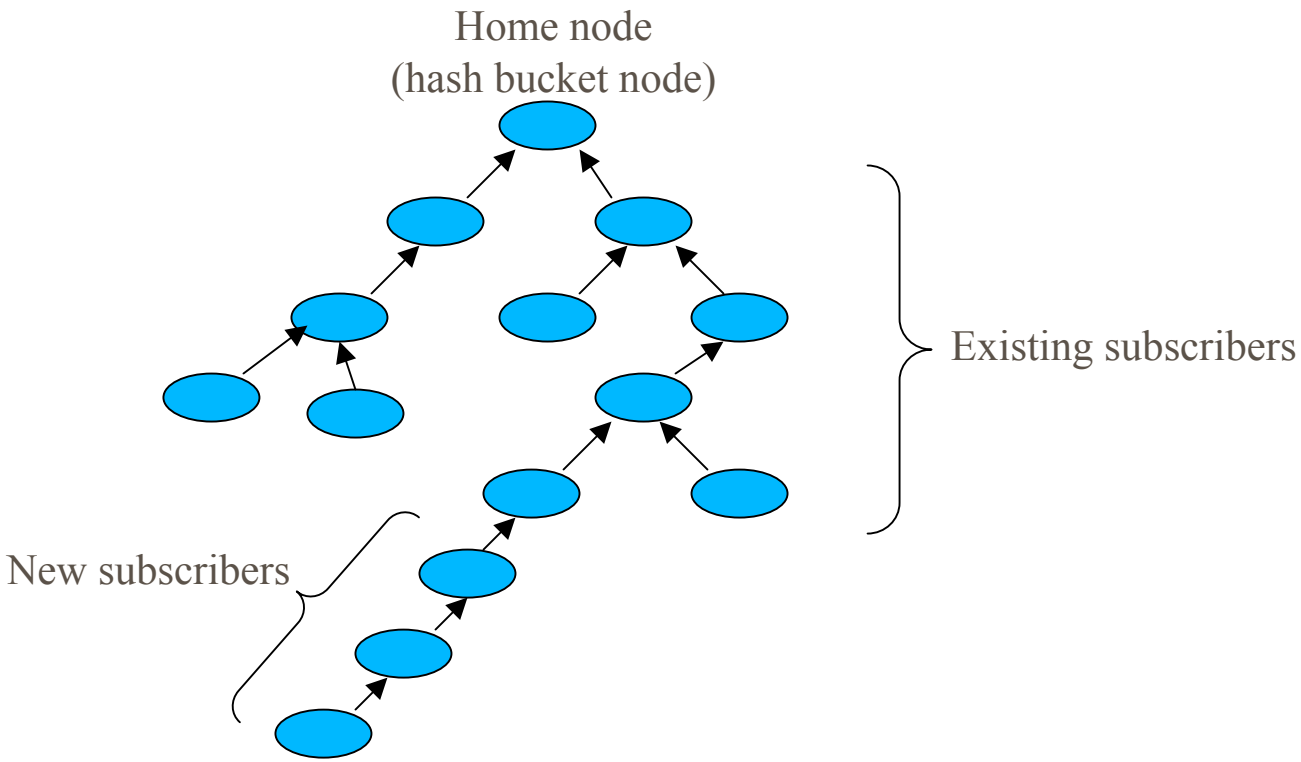
Our Approach (cont'd)

- Multicast Trees
  - eliminates bottleneck at hash bucket nodes
  - distributed subscribing, publishing, matching



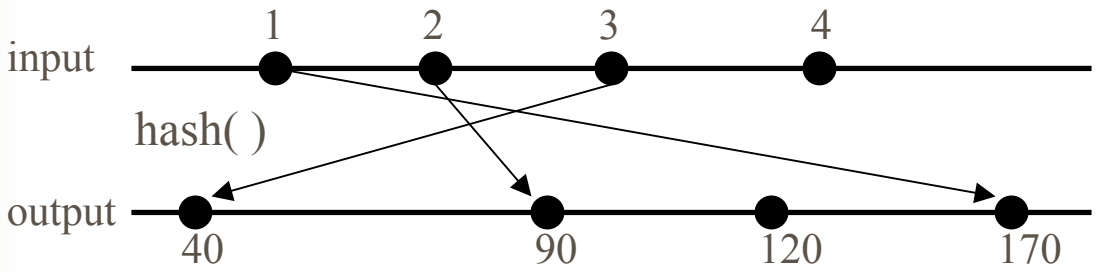
Our Approach (cont'd)

- Multicast Trees
  - eliminates bottleneck at hash bucket nodes
  - distributed subscribing, publishing, matching

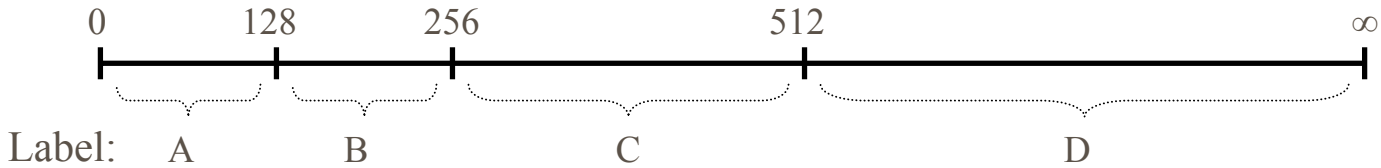


Handling Range Queries

- Hash function ruins locality



- Divide range of values into intervals
  - hash on interval labels
    - e.g. RAM attribute



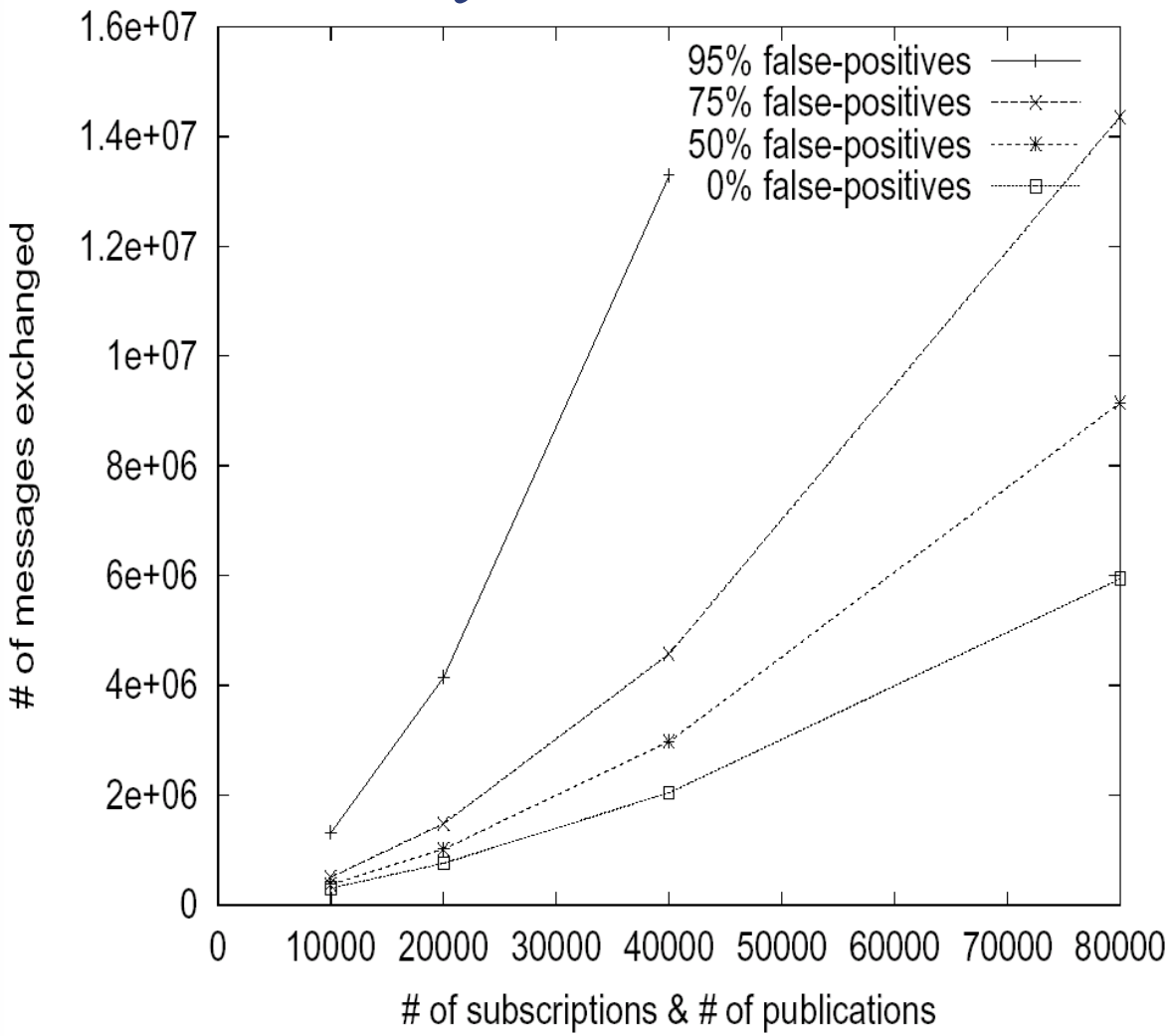
    - For RAM > 384, submit hash keys:
      - RAM = C
      - RAM = D
  - intervals can be sized according to probability distribution

Implementation & Evaluation

- Main Goal: scalability
- Metric: message traffic
- Built using:
  - **Pastry** DHT
  - **Scribe** multicast trees
- Workload Generator: uniformly random distributions

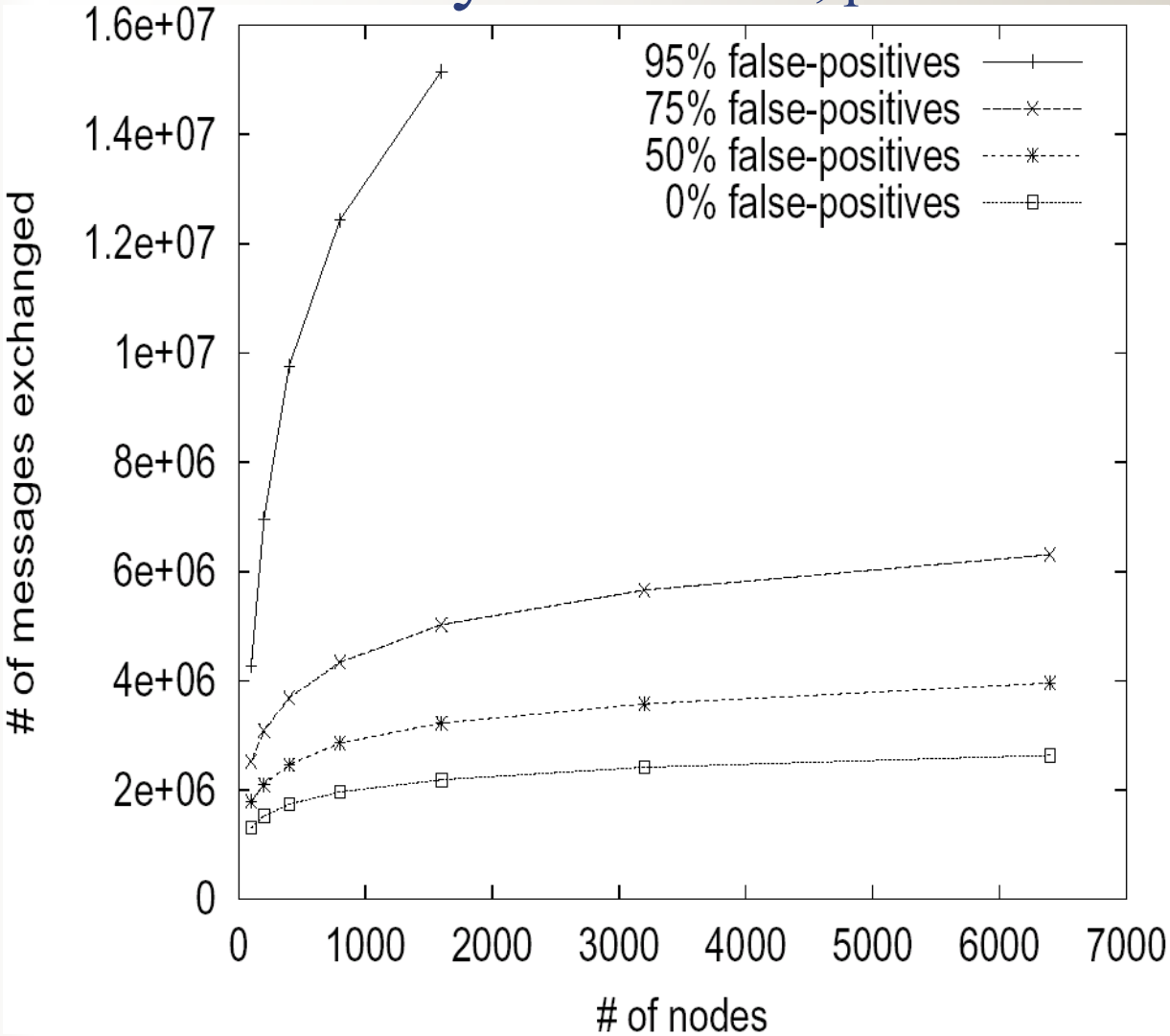
Order	Type	Name	Values	Index 1	Index 2	Index 3	Index 4
1	Integer	Price	1 to 5	√	√	√	√
2	Integer	Volume	1 to 5	√	√	√	√
3	Integer	Color	1 to 5	√	√	√	√
4	Integer	Size	1 to 5	√	√	√	
5	Integer	Temperature	1 to 2	√	√		
6	Integer	Circumference	1 to 2	√			

Event Scalability: 1000 nodes

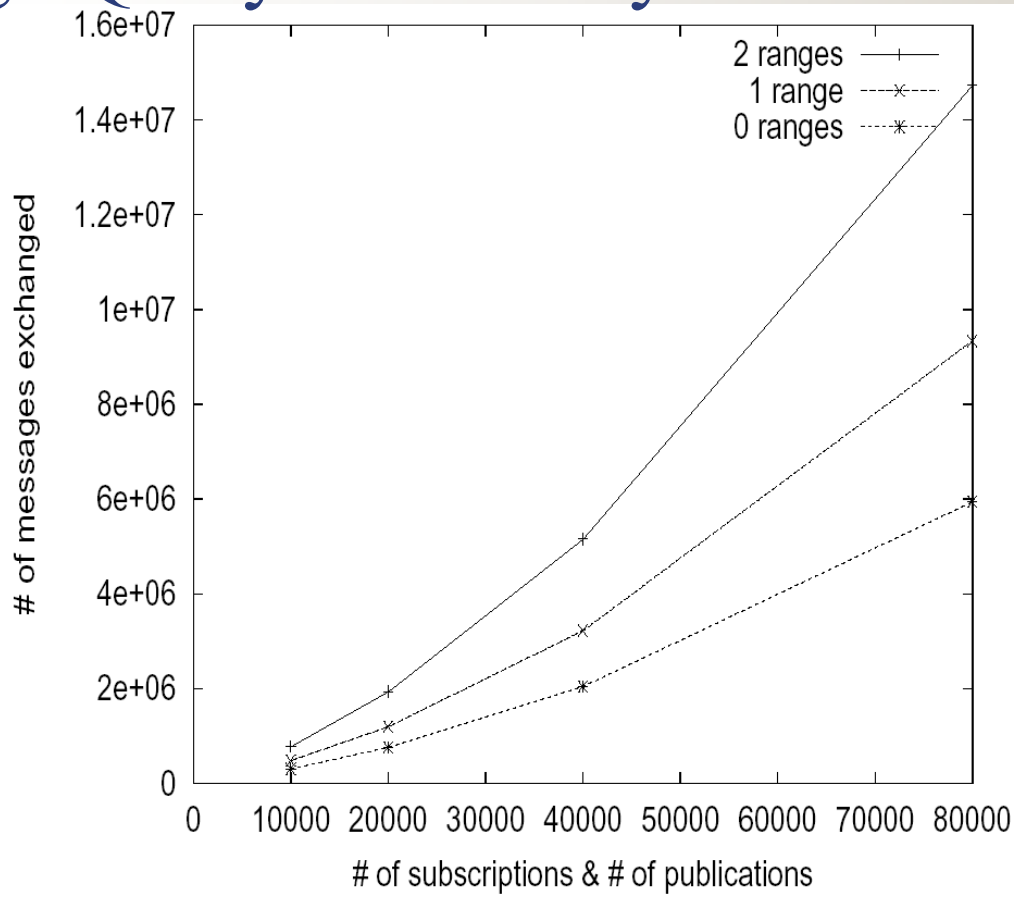


- Need well-designed schema with low false-positives

Node Scalability: 40000 subs, pubs

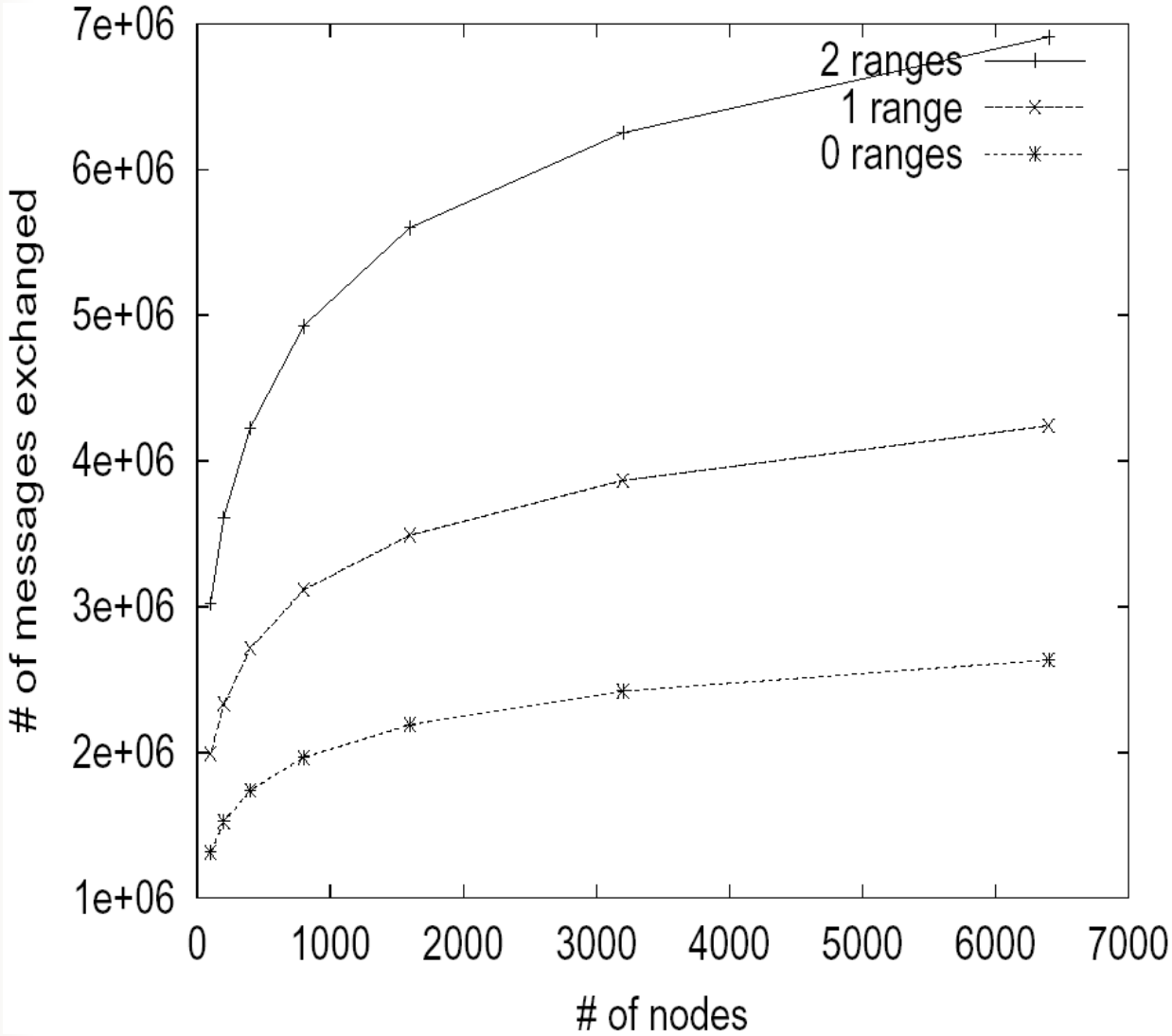


Range Query Scalability: 1000 nodes



- Multicast tree benefits
  - e.g. 1 range vs 0 range, at 40000 subs, pubs
    - Expected 2.33 × msgs, but got 1.6
  - subscription costs decrease

Range Query Scalability: 40000 subs, pubs



Conclusion

- Method: DHT + domain schema
- Scales to 1000s of nodes
- Multicast trees are important
- Interesting point in design space
  - some restrictions on expression of content
    - must adhere to domain schema

Future Work

- range query techniques
- examine multicast tree in detail
- locality-sensitive workload distributions
- real-world workloads
- detailed modelling of P2P network
- fault-tolerance