PUB-2-SUB: A Content-Based Publish/Subscribe Framework for Cooperative P2P Networks

Duc A. Tran Cuong Pham

Network Information Systems Lab (NISLab)
Dept. of Computer Science
University of Massachusetts, Boston

Outline

1. Publish/Subscribe Paradigm
2. Pub/Sub in Cooperative P2P Networks
   - Possible Approaches
3. Contribution: PUB-2-SUB
   - Description
   - Evaluation
4. Summary

Publish/Subscribe Paradigm

- An *asynchronous* paradigm of
  - Subscribers: the consumers of the data
    - Subscription: a query submitted by the subscriber to the network to specify a data interest. E.g., "I am very interested in "blah, blah, blah", whenever somebody has it, please send it to me"
  - Publishers: the producers of the data
    - Event: the data information submitted by the publisher to the network to find consumers. E.g., "I have this "blah, blah, blah". Please let those persons interested in it know that I have their data"
- Enable subscribers and publishers, which do not know each other, to find each other quickly

Applications: social networking, e-commerce, event monitoring, anomaly detection, etc.

Pub/Sub vs. Traditional Search

- Traditional search = Request/Reponse
  - Data must exist already
  - Queries expect immediate results
  - Data is stored in the network in advance, not queries
    - Data indexing and storage = an important problem

Pub/Sub ≠ Request/Respose

- Data may currently exist, currently not exist, or never exist
- Queries are sent in advance expecting to be notified
- Queries must be stored in the network in advance
  - Subscription indexing and storage = an important problem

Query indexing in pub/sub is more difficult than data indexing in request/response

Pub/Sub vs. Traditional Search

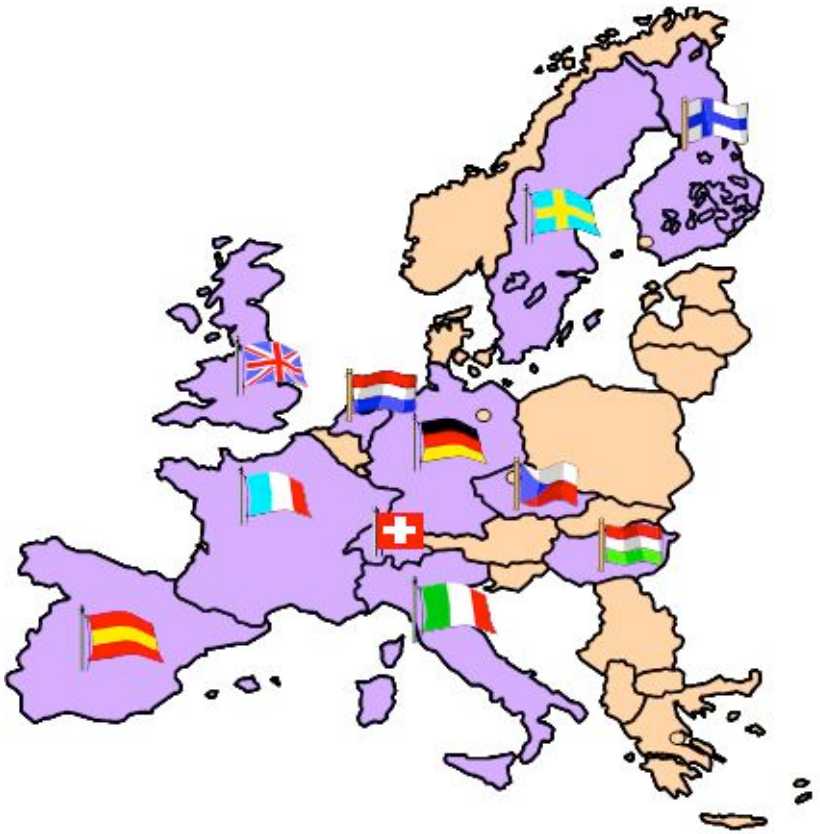
- Traditional search = Request/Reponse
  - Data must exist already
  - Queries expect immediate results
  - Data is stored in the network in advance, not queries
    - Data indexing and storage = an important problem

Pub/Sub ≠ Request/Respose

- Data may currently exist, currently not exist, or never exist
- Queries are sent in advance expecting to be notified
- Queries must be stored in the network in advance
  - Subscription indexing and storage = an important problem

Query indexing in pub/sub is more difficult than data indexing in request/response

Cooperative P2P Networks



- E.g., P2P-based multi-institution collaborative networks, P2P-based cable TV networks (Comcast, GridNetworks), grid data networks
- Possibly (often) unstructured
- Nodes are reliable, and rather static
- Likely a policy: communication must use only provided links

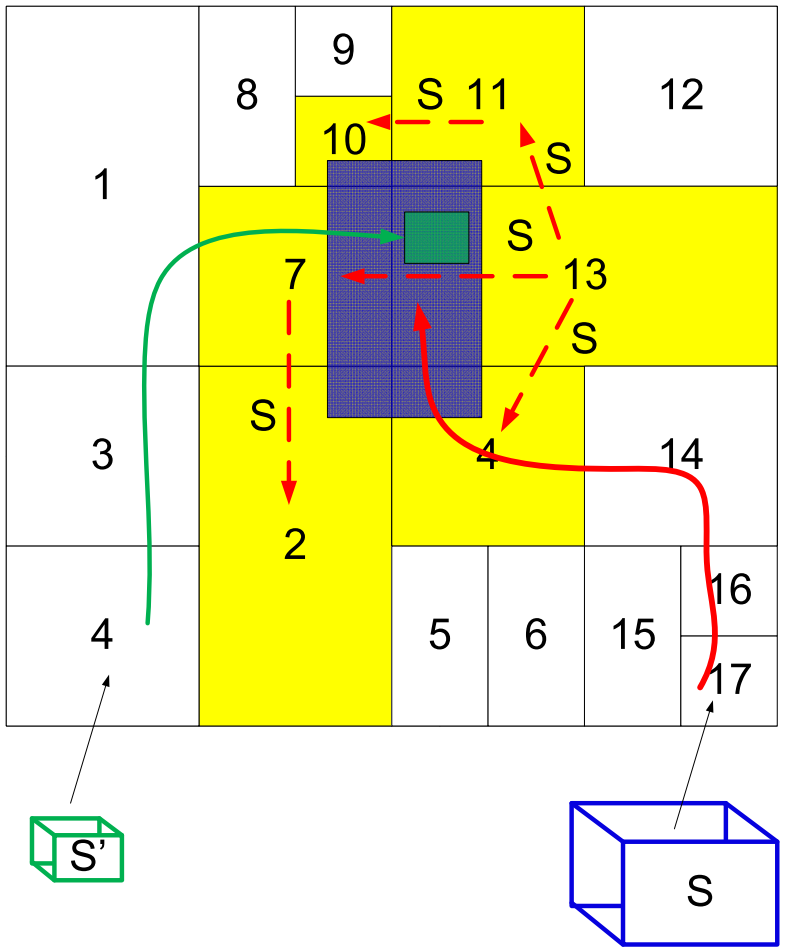
Pub/sub in P2P networks is not trivial:

- Broadcasting a query to all nodes → expensive (traffic, storage)
- Deploy a central server to index everything → not scalable

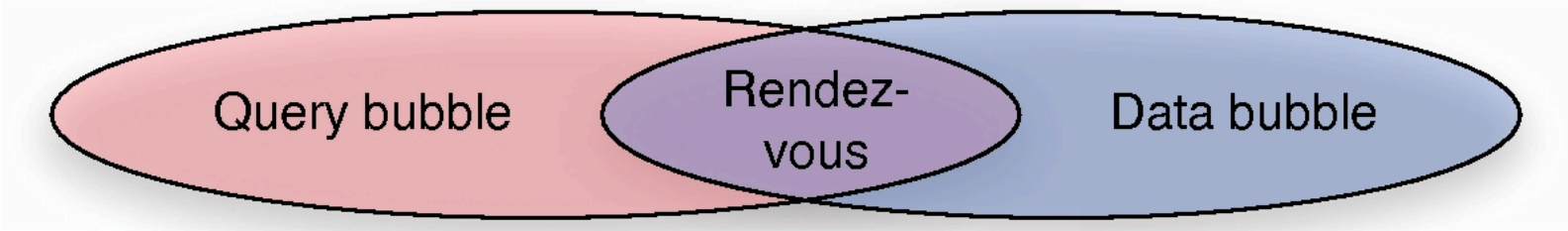
Possible Approaches: Structuralization-based Pub/Sub

Build a structured overlay (e.g., using DHT, Small World, R-tree, Skip Lists)

- E.g., Meghdoot [3], Scribe [1], Hermes [5], Bayeux [7], GosSkip [2]
- Advantage: capable to grow the network and adapt to dynamics
- Additional cost to maintain the overlay
- Different overlays for different pub/sub applications



Possible Approaches: Gossip-based Pub/Sub



Gossip to a number of nodes, not all but large enough so that a rendezvous node exists for any pair of (query, event) highly likely

- E.g., BubbleStorm [6]: Replicate each query at $q = O(\sqrt{n})$ nodes and sends each publication to $c^2 n/q$. Hit probability is $1 - \exp(-c^2)$ (e.g., $c = 3 \rightarrow 99.99\%$ hit)
- Advantage: not require any additional overlay
- Indifferent to query/event content
- Expensive to replicate queries and publish events

Our Contribution

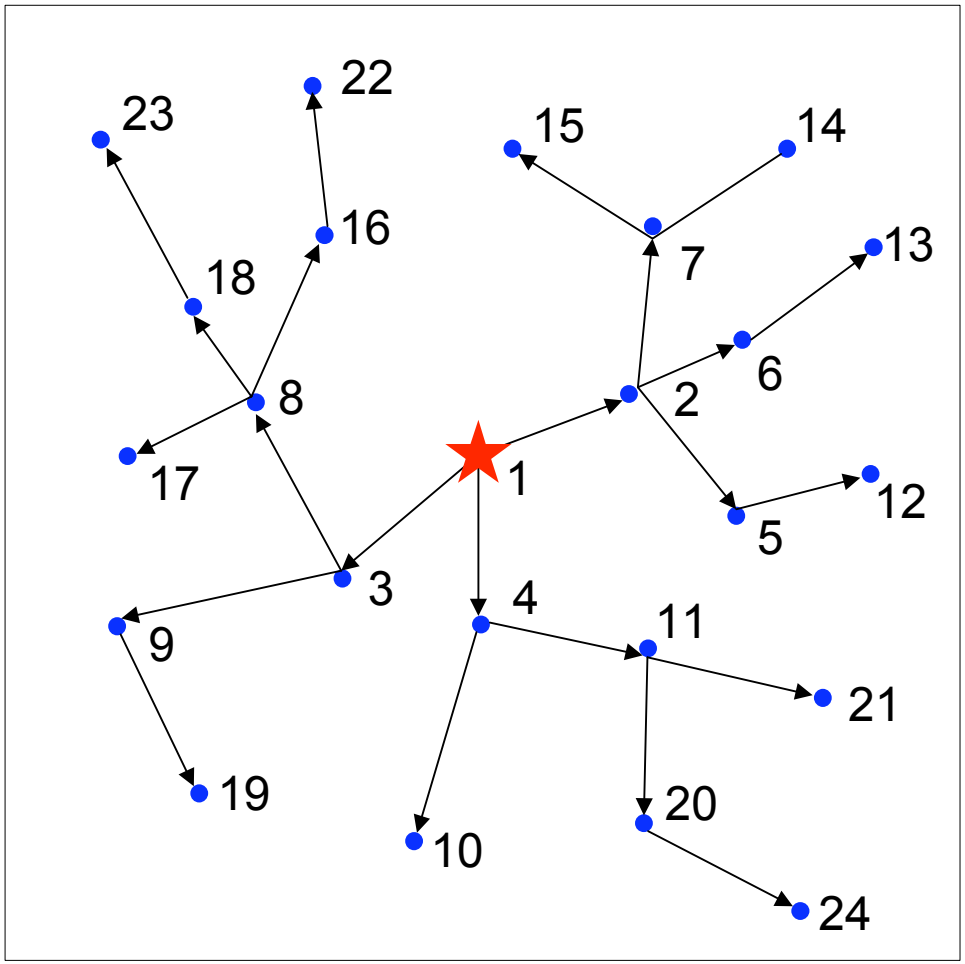
PUB-2-SUB

- Work with any unstructured cooperative P2P network
- Efficiency in terms of both cost and time (aimed to be better than gossiping)
- Allow multiple independent publish/subscribe applications to run simultaneously on a single instance of PUB-2-SUB

How?

- Use directed routing rather than gossiping
- Virtualization: assign to each node a unique virtual address
- Indexing: determine subscription and notification paths whose routing is based on the virtual addresses

Virtualization: Construction

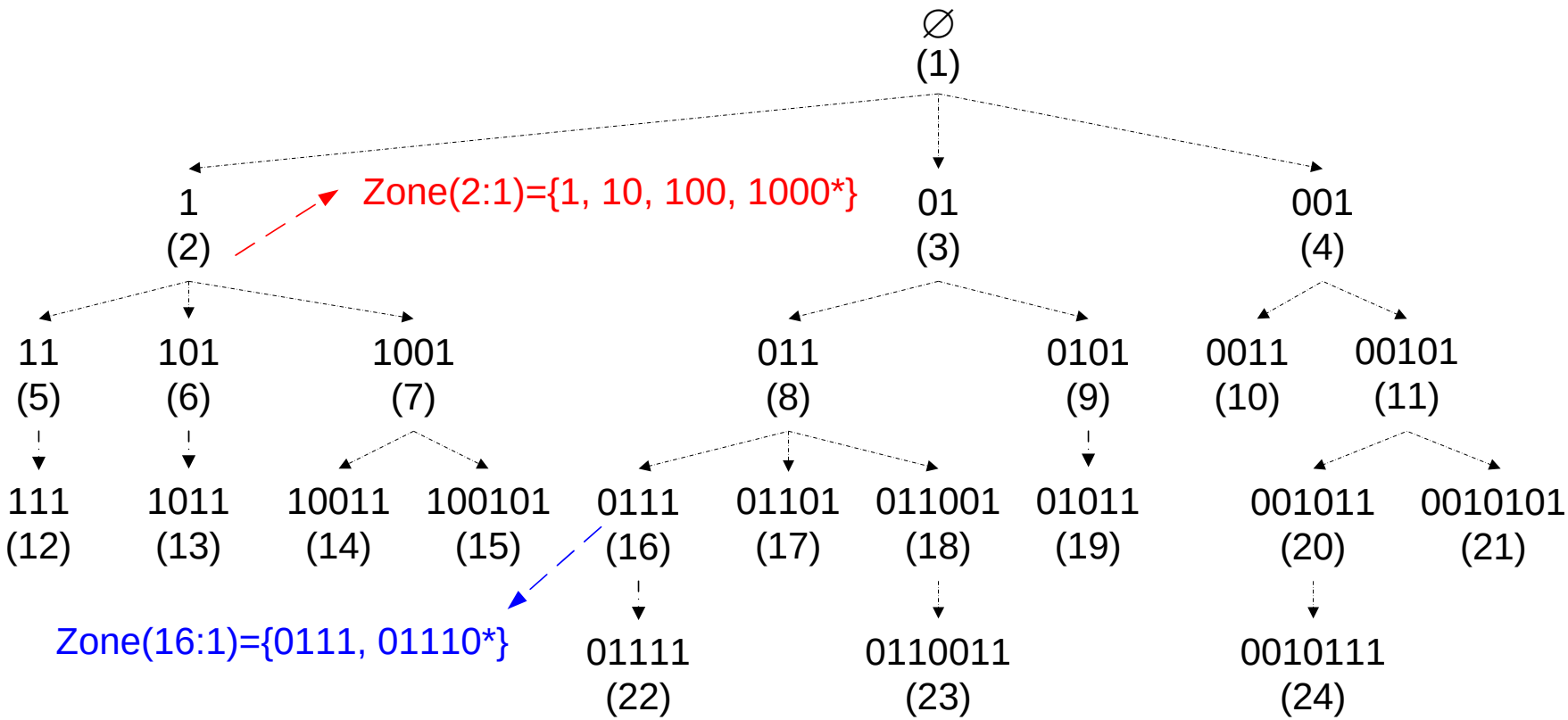


Given a bootstrap node S^*, $VA(S^* : S^*) = \varnothing$:

1. S^* sends JOIN INSTANCE(S^*) message to neighbors
2. At each node S_j that receives JOIN from S_i
   - Accept JOIN if not yet part of INSTANCE(S^*)
   - Assigned a VA by S_i
   - Node S_j follows the same procedure

Virtualization: Virtual Addresses (VA) and Zones

- $ZONE(S_i : S^*)$ = {string str | (1) $VA(S_i : S^*)$ is a prefix of str; (2) No child of S_i has VA a prefix of str}
- $VA(S_i : S^*)$ = (unused) shorted string $VA(parent(S_i) : S^*)$ + '0*1'



Indexing: Query/Event Representation

Event and Subscription

- Event: $x = (x_1, x_2, ..., x_k)$ where $x_i \in \{0, 1\}$
  - k: number of bits
- Subscription $Q = [q_l, q_h]$:
  - q_l, $q_h \in \{0, 1\}^k$
- Matching condition:
  - Q subscribes to all events x belonging to its interval (events are "ordered" lexicographically)

Example: $k = 3$, the events matching a query [‘001’, ‘101’] are {‘001’, ‘010’, ‘011’,‘100’, ‘101’}

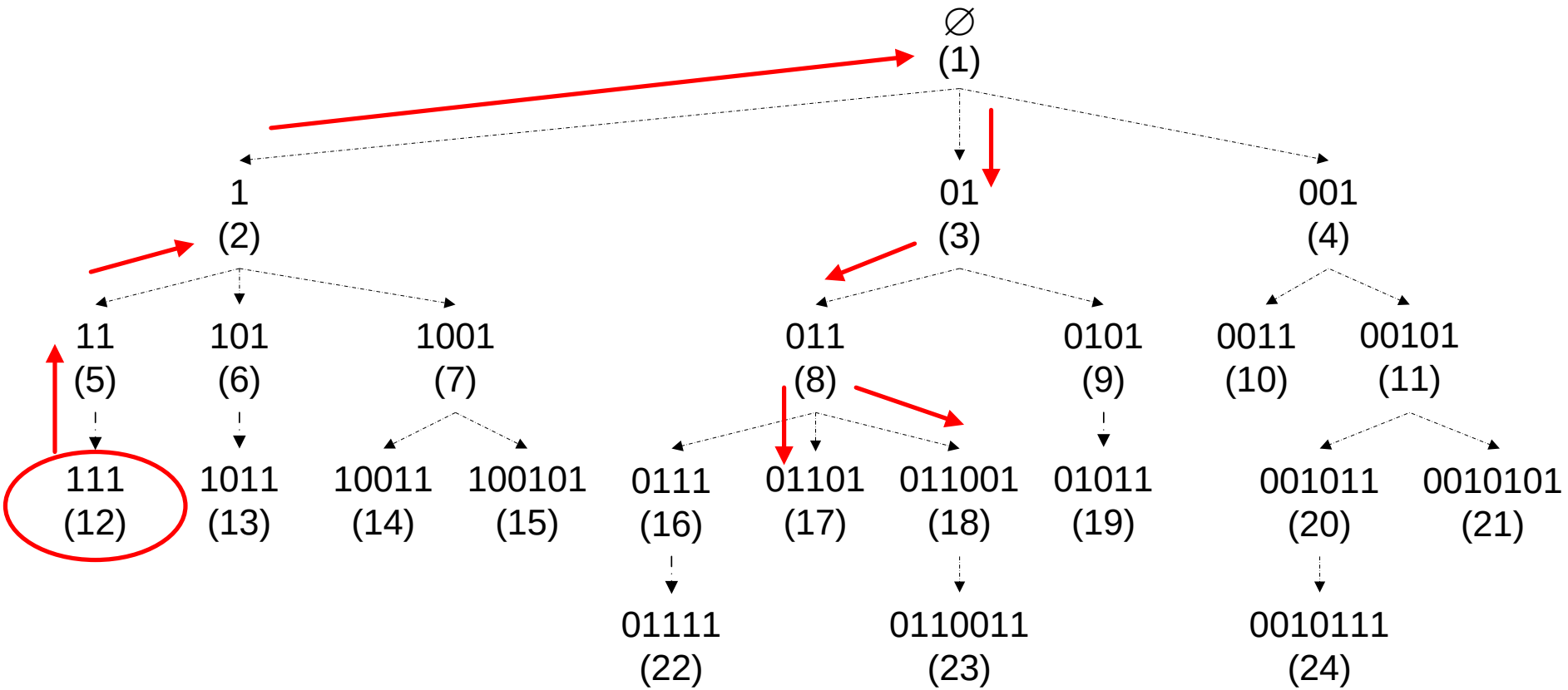
Indexing: Query Subscription

Considering a query Q:

- Initially, the subscription starts at the subscriber node of Q
- At each node S_i that receives Q
  1. Quit if S_i already received this query before
  2. Let $Z = \{str \in \{0,1\}^k \mid VA(S_i : S^*)$ is a prefix of $str\}$
  3. IF (Z does not overlap Q)
     1. Forward Q to the parent node of S_i in $TREE(S^*)$
  4. ELSE
     1. Store Q at S_i if $ZONE(S_i : S^*)$ intersects Q
     2. Forward Q to all children of S_i in $TREE(S^*)$
     3. IF (Z does not contain Q), forward Q to parent of S_i in $TREE(S^*)$

Query Subscription: Example

Subscription path of query Q = ['0110001', '0110101'] from node 12



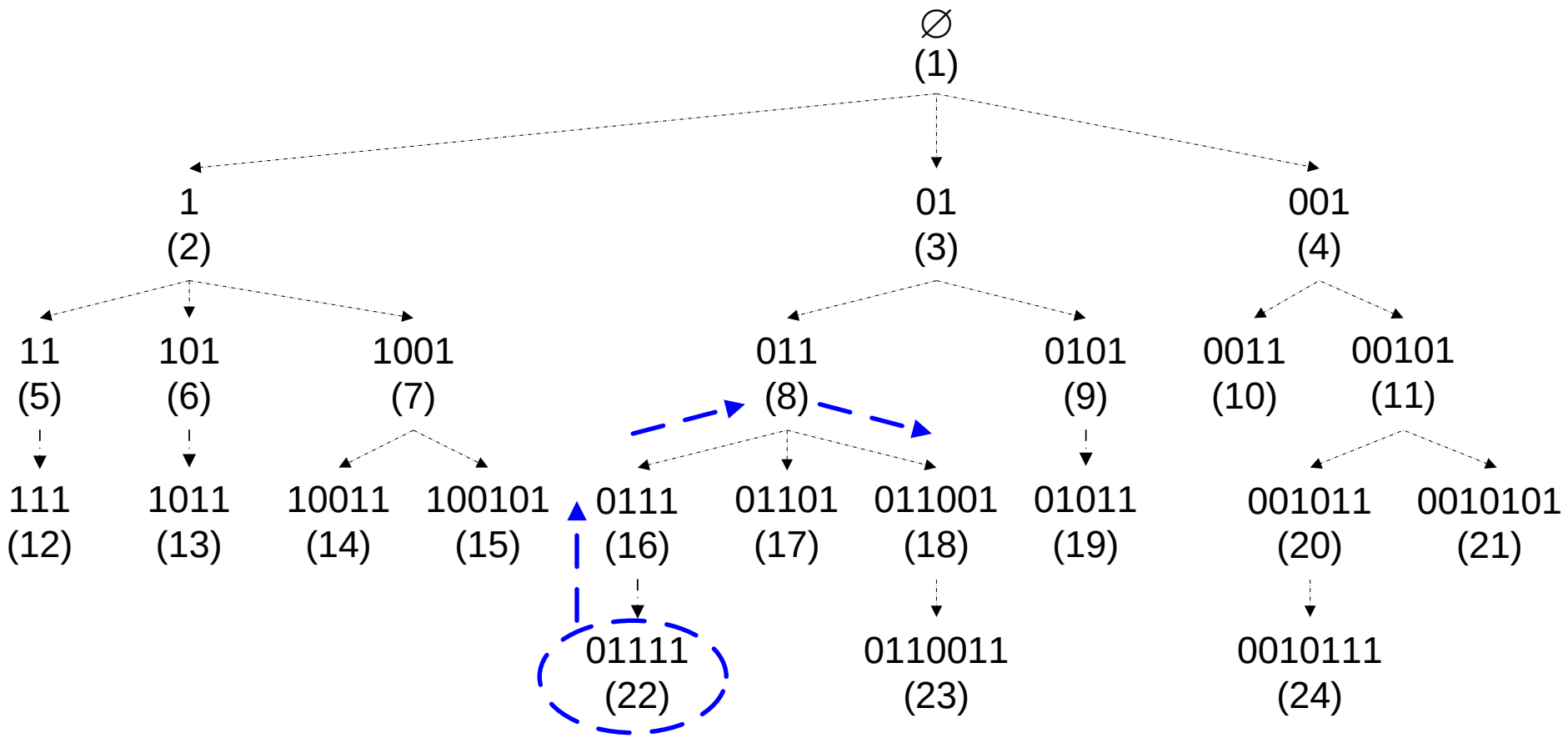
Indexing: Event Notification

Considering an event x:

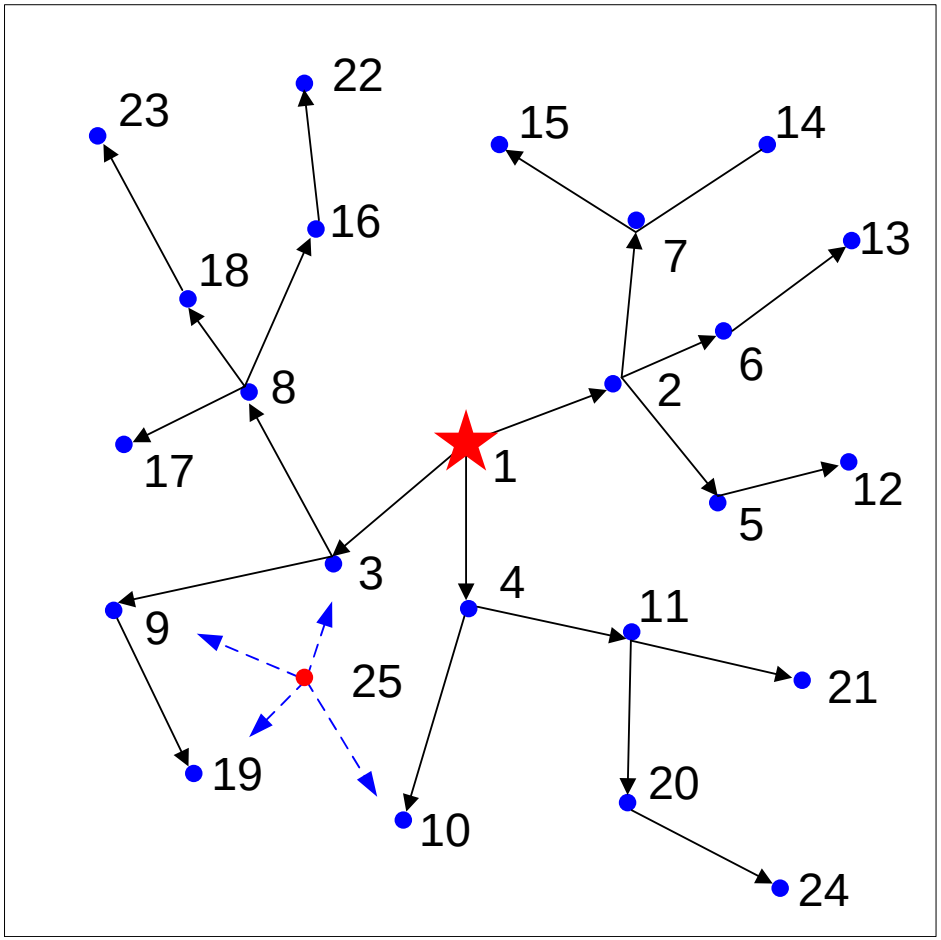
- Initially, the notification starts at the publisher node of x
- At each node S_i that receives x
  1. IF ($VA(S_i : S^*)$ is not a prefix of x) THEN
     1. Forward x to the parent node of S_i in $TREE(S^*)$
  2. ELSE
     1. Find the child node S_j such that $VA(S_j : S^*)$ is a prefix of x
     2. IF (S_j exists) THEN Forward x to S_j
     3. ELSE Search node S_i for those queries matching x

Event Notification: Example

Notification path of event <'0110010'> from node 22



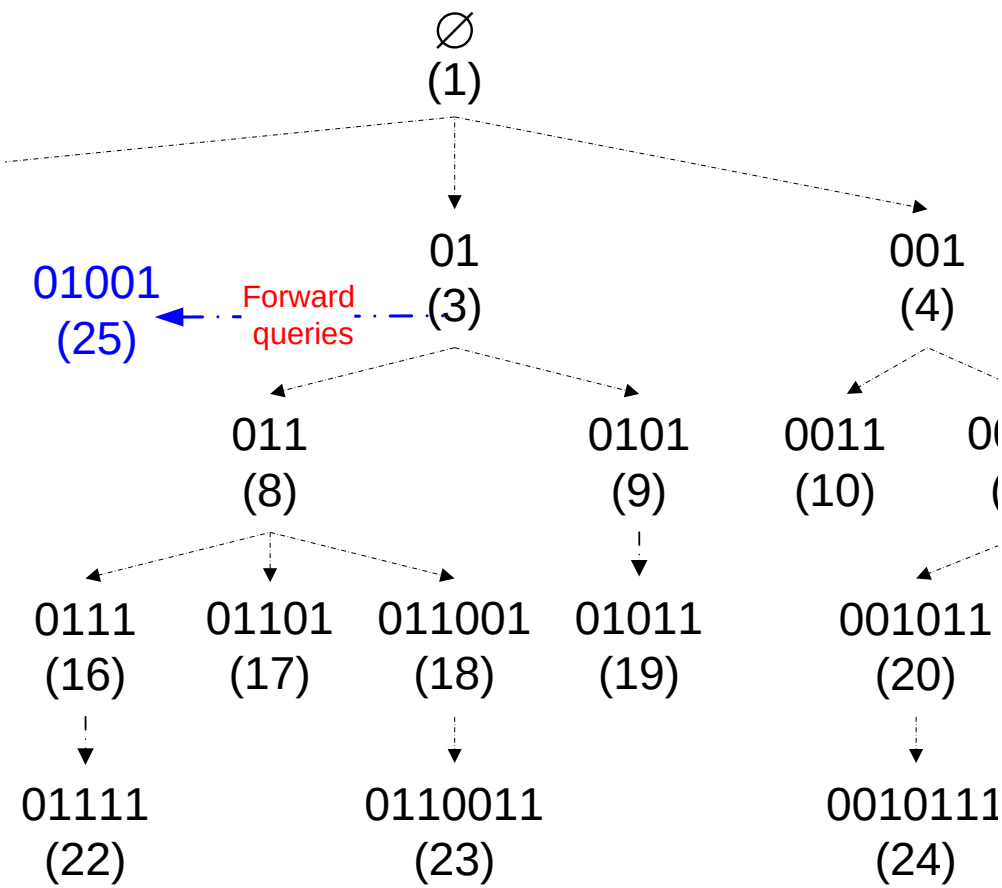
Node Addition (1/2)



Consider a new node S_{new} joined:

1. S_{new} communicates with its neighbors
2. Ask $S_{neighbor}$ with minimum tree depth to be its parent
3. Get a VA from parent

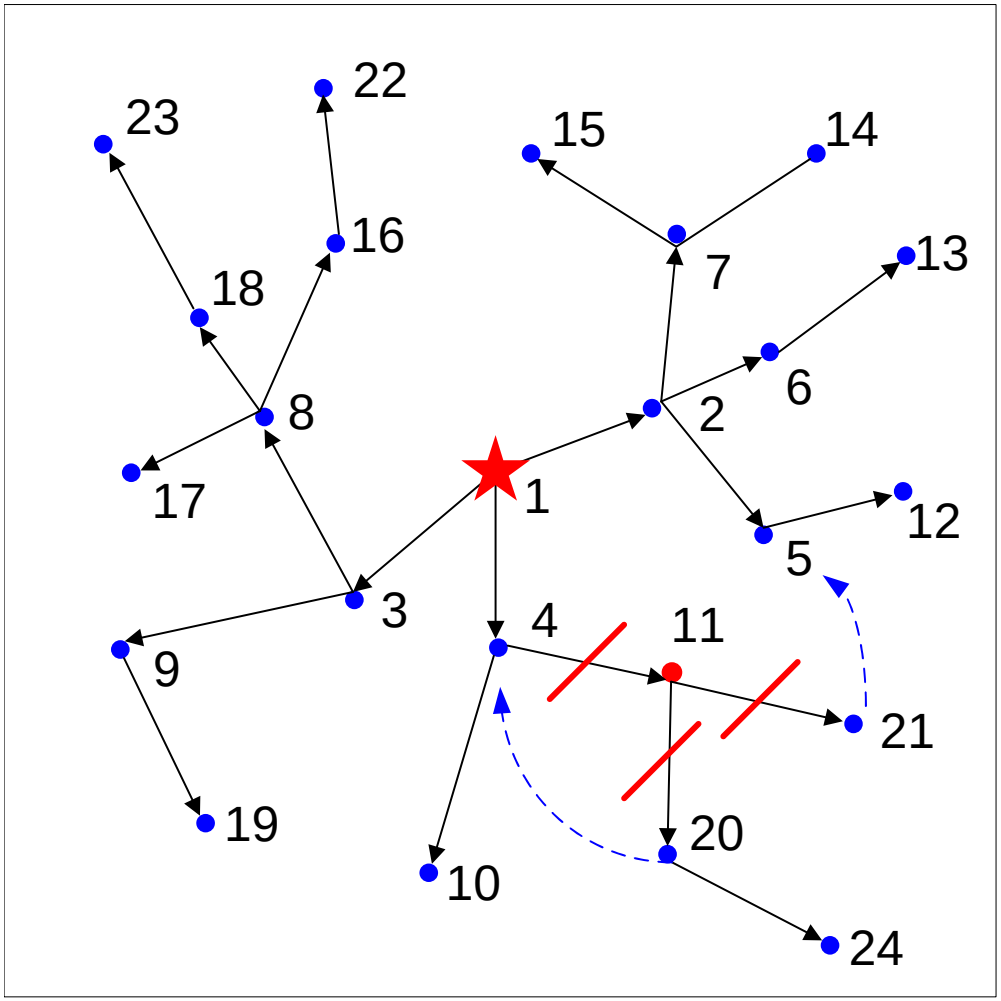
Node Addition (2/2)



Consider a neighbor $S_{neighbor}$ is chosen:

1. Assign S_{new} shortest unused binary string of the form $VA(S_{neighbor} : S^*)$ + '0*1'
2. Delete queries that do not intersect $ZONE(S_{neighbor} : S^*)$
3. Forward to S_{new} the queries that intersect $ZONE(S_{new} : S^*)$

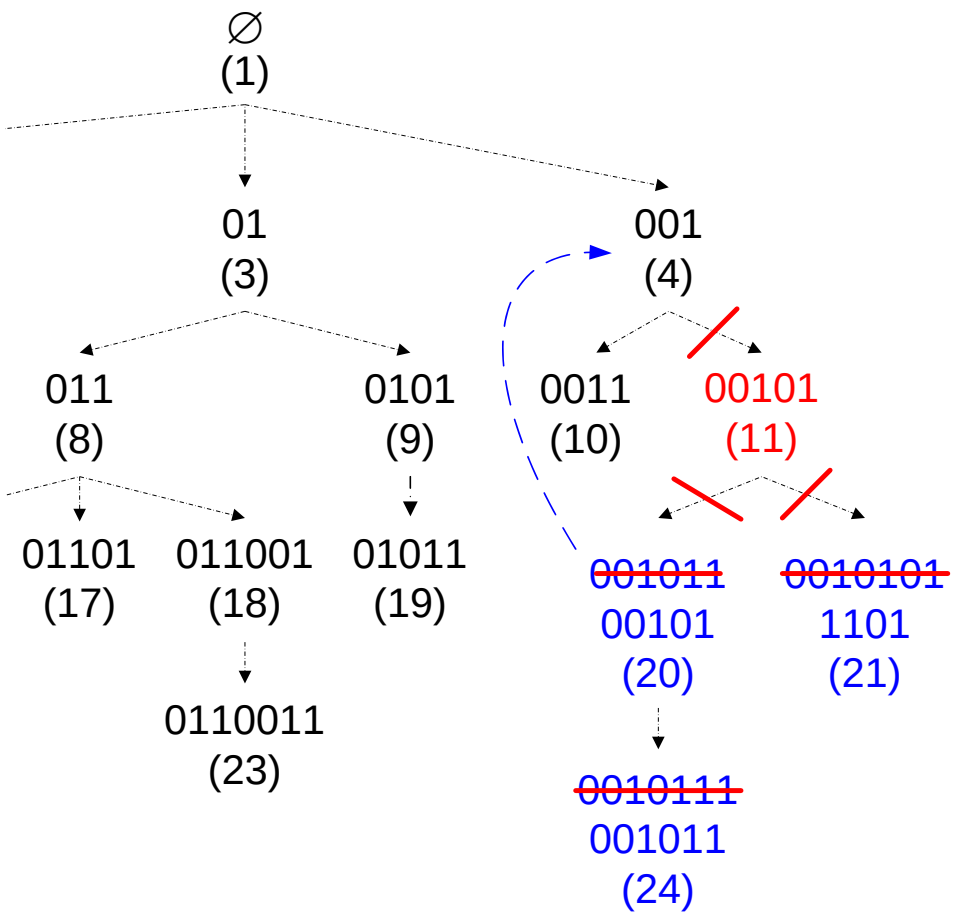
Node Removal (1/2)



When a node fails:

- Child nodes need to find a new parent and get a new VA

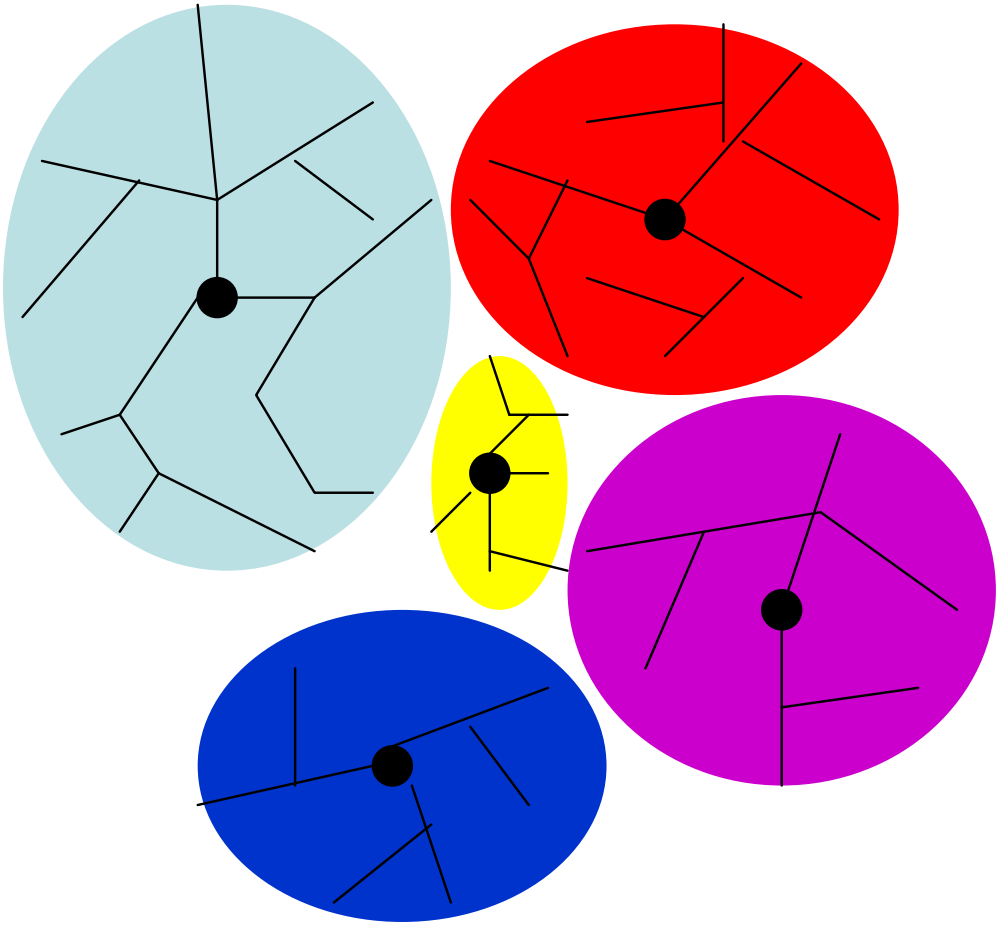
Node Removal (2/2)



Consider a child node S_{child}:

1. Select a new parent S_{parent} from its neighbors and get new VA from this new parent
2. Recompute VAs for its children
3. Transfer/Delete/Forward queries if necessary
4. The same procedure happens with all its descendant nodes downstream

Multiple VA-Instances



Choose *m* VA-instances inititated by random nodes

- Query: subscribed to a random VA-instance
- Event: submitted to every VA-instance
- Advantages:
  - Better load balancing
  - Better reliability
  - Unchanged storage and communication costs per query
  - Linearly increased communication and computation costs per event

One-Dimension vs. Multi-Dimension

- So far:
  - Event: k-bit binary string
  - Query: unidimensional interval
- In practice:
  - Event: d-dimension, where d- number of atributes
  - Query: d-dimensional rectangular range of values

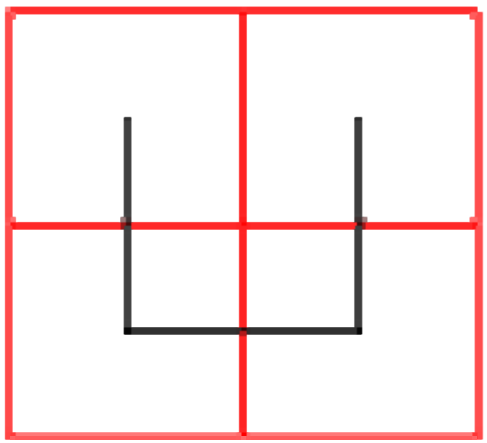
How PUB-2-SUB works in multidimensional domain?

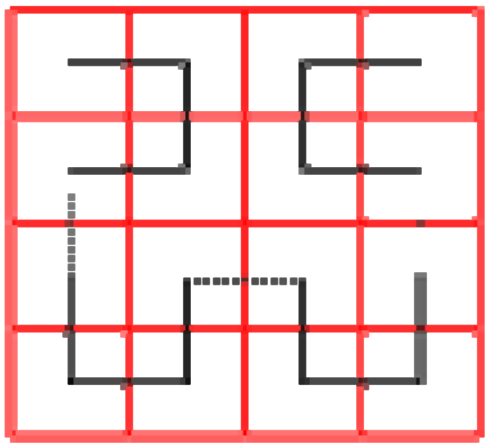
- Hash mechanism f
  1. $x^f = f(x)$, where $f(x)$- a unidimensional k-bit binary string
  2. $Q^f = f(Q)$, where $f(Q)$- an interval of k-bit strings
  3. $x \in Q$ then $x^f \in Q^f$
- Example: $(k/2)$-order Hilbert Curve mapping

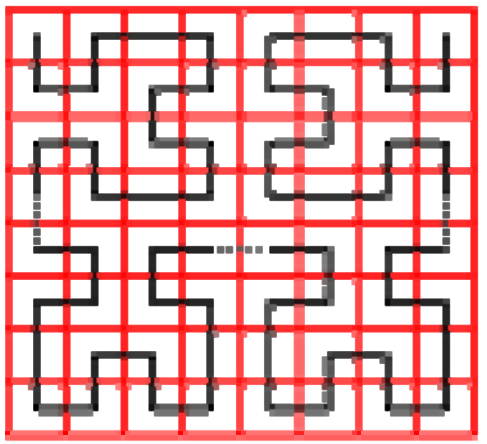
Hilbert Curve

Hilbert (n, d)

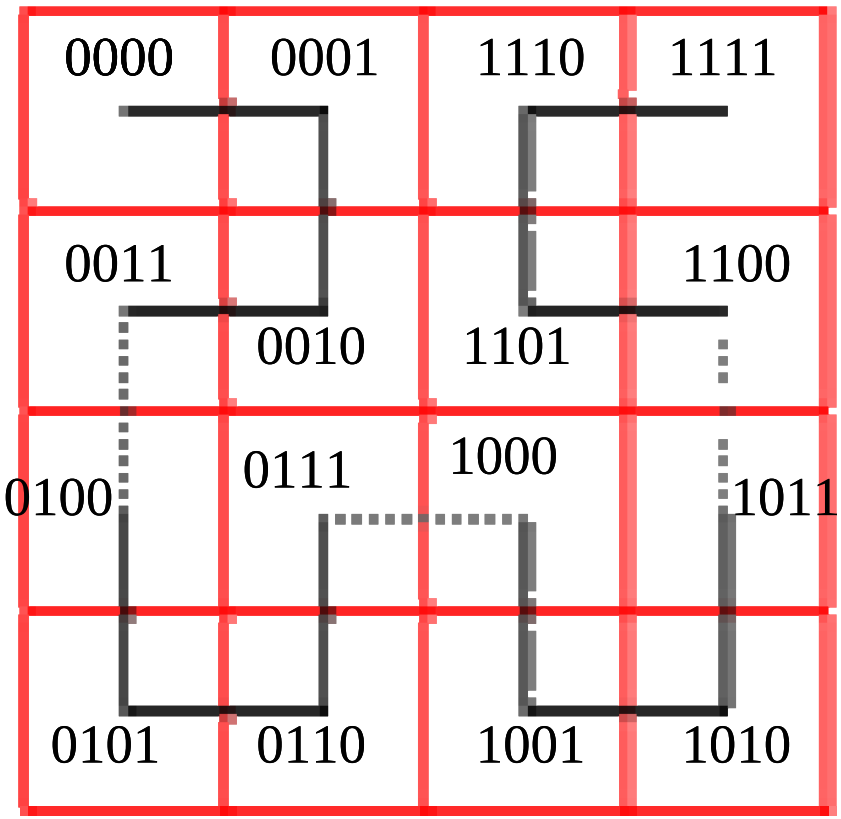
- In each partition, the d-dimensional cube is partitioned into nd equal sub-cubes
- The centers of these sub-cubes are joined with line segments to make a continuous curve
- The same policy applies to each of the sub-cubes







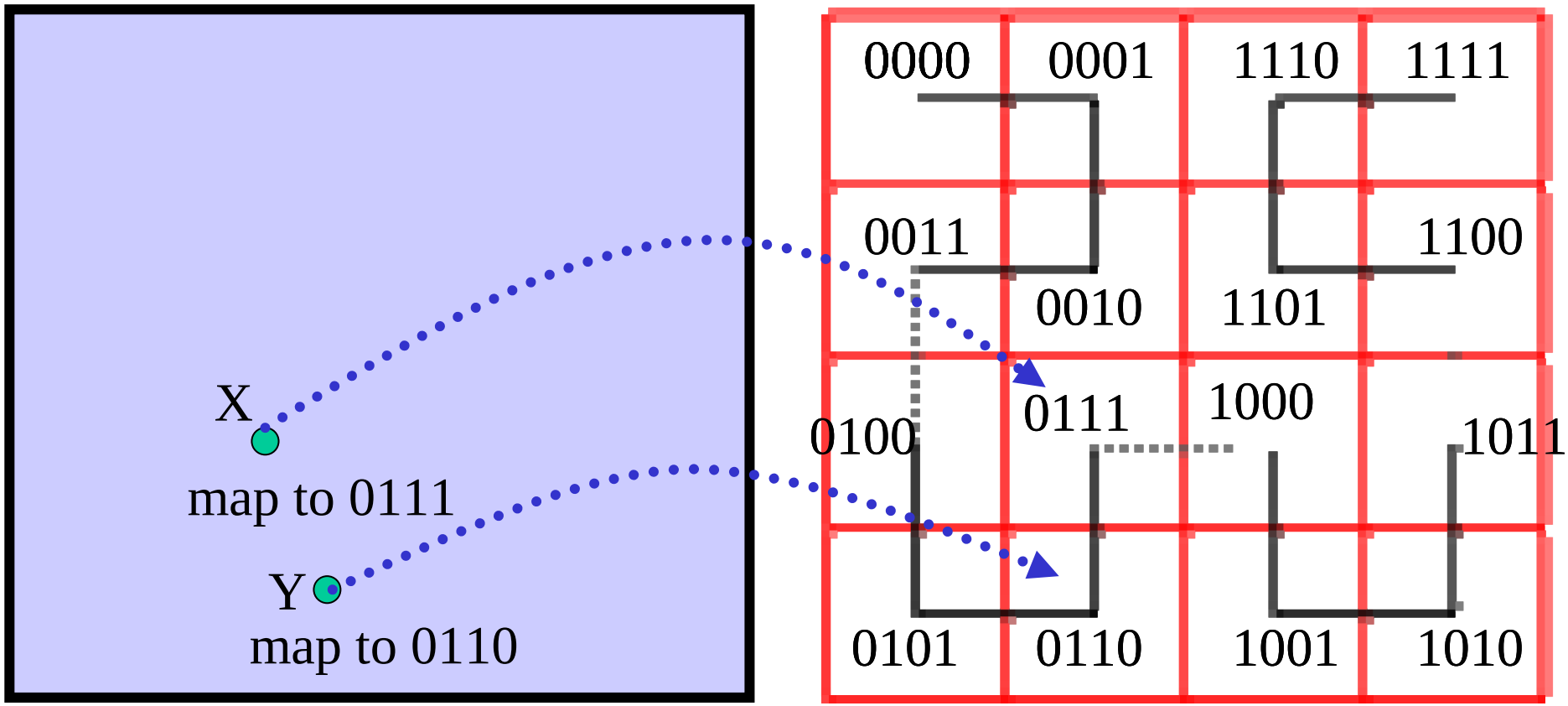
Hilbert Curve: Index Construction



After k^{th} order partition:

- $n^{k.d}$ cells in the d-dimensional cube
- Each cell is given a base-n number of $k.d$ digits
- First cell is 0, incremented by 1 for the next cells

Hilbert Curve: Mapping

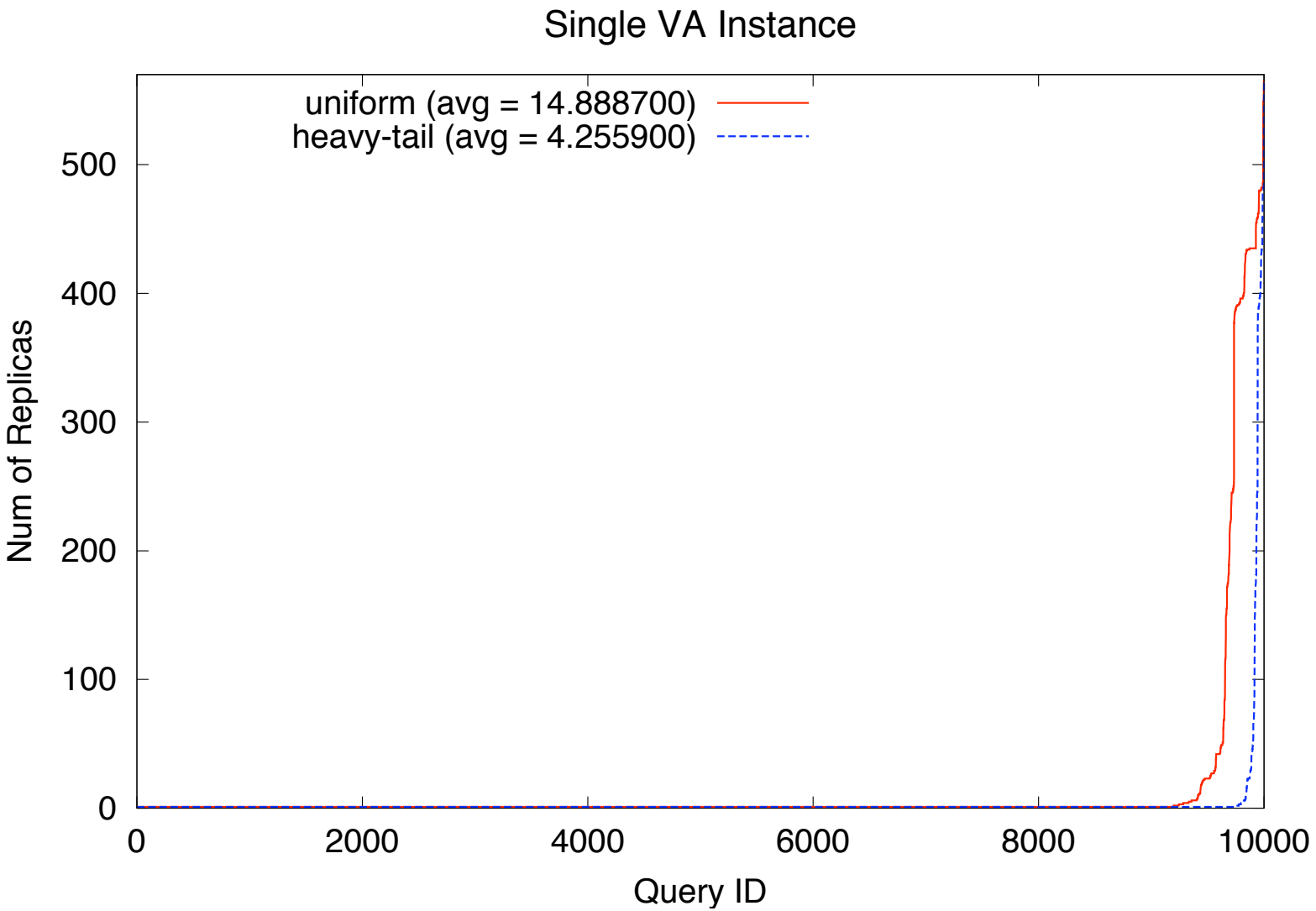


Highly probable that close data elements map to close Hilbert points

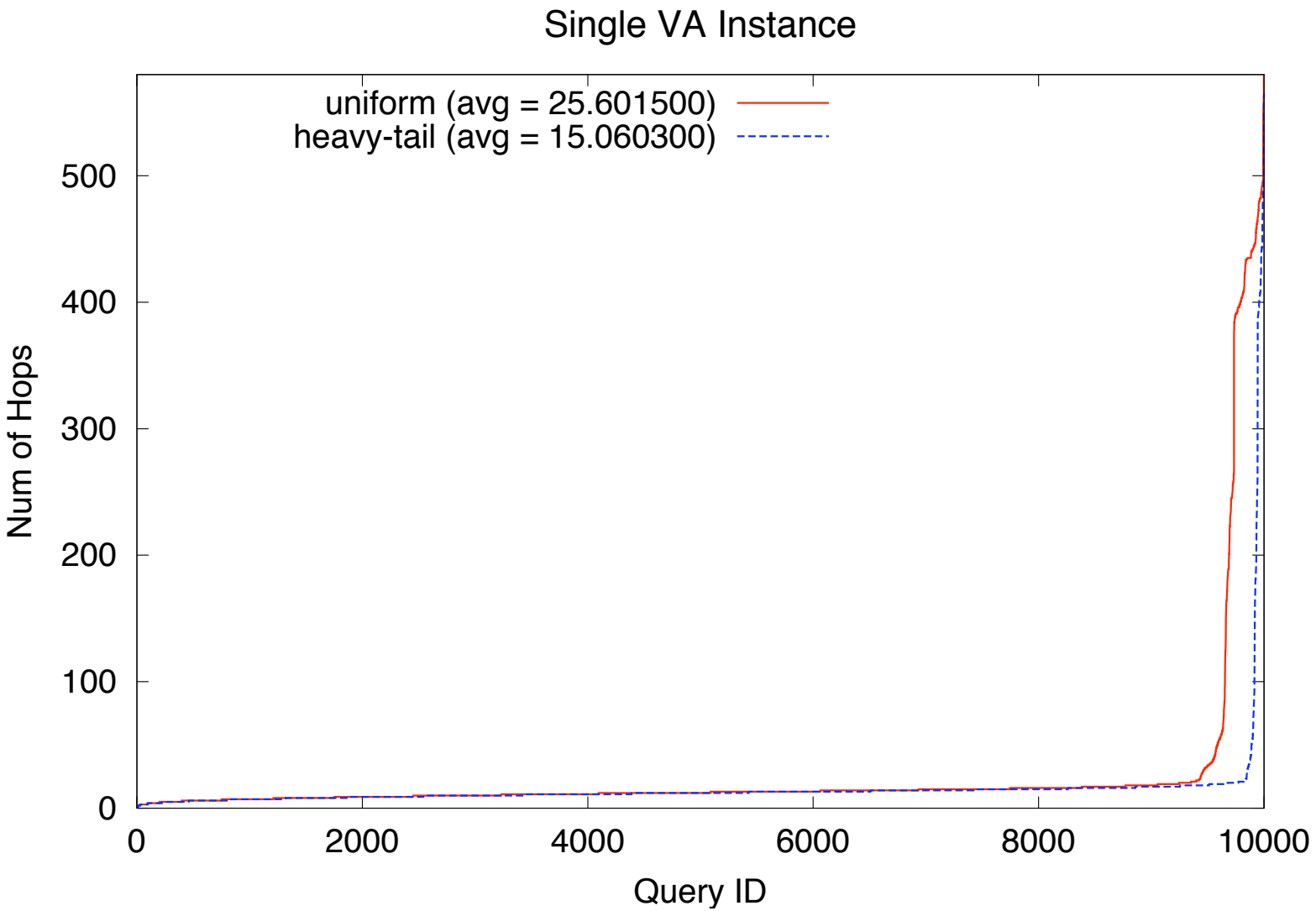
Simulation Study

- Network topology generated by BRITE [4]
  - 1000 nodes, 2766 peer-to-peer links (power-law distributed)
- Inititated by a random node
  - (10,000) Event = k-bit string (k = 50)
  - (10,000) Query = interval of k-bit strings, range chosen by Zipf's law: in $\{2^0, 2^1, ..., 2^{49}\}$, range 2^i is picked with probability $\frac{1/i^\alpha}{\sum_{j=1}^{50}(1/j^\alpha)}$
    - $\alpha = 0$: uniform
    - $\alpha = 0.8$: heavy-tail with most queries being specific
- Metrics: subscription efficiency, notification efficiency, notification delay, failure effect, load balancing, and effect of using multiple VA instances.
- Pub-2-Sub vs. BubbleStorm [6]

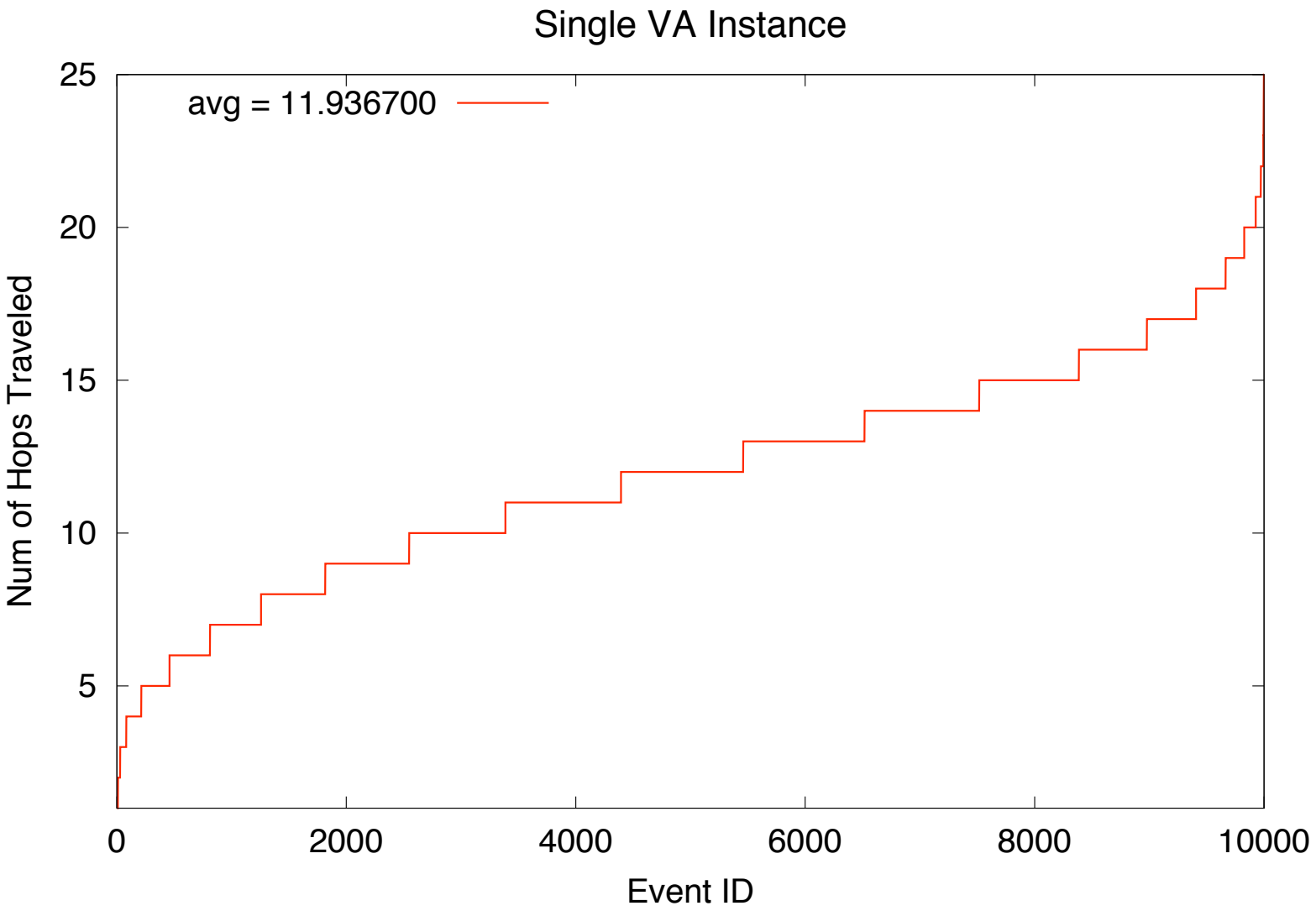
Subscription Efficiency: Storage Cost



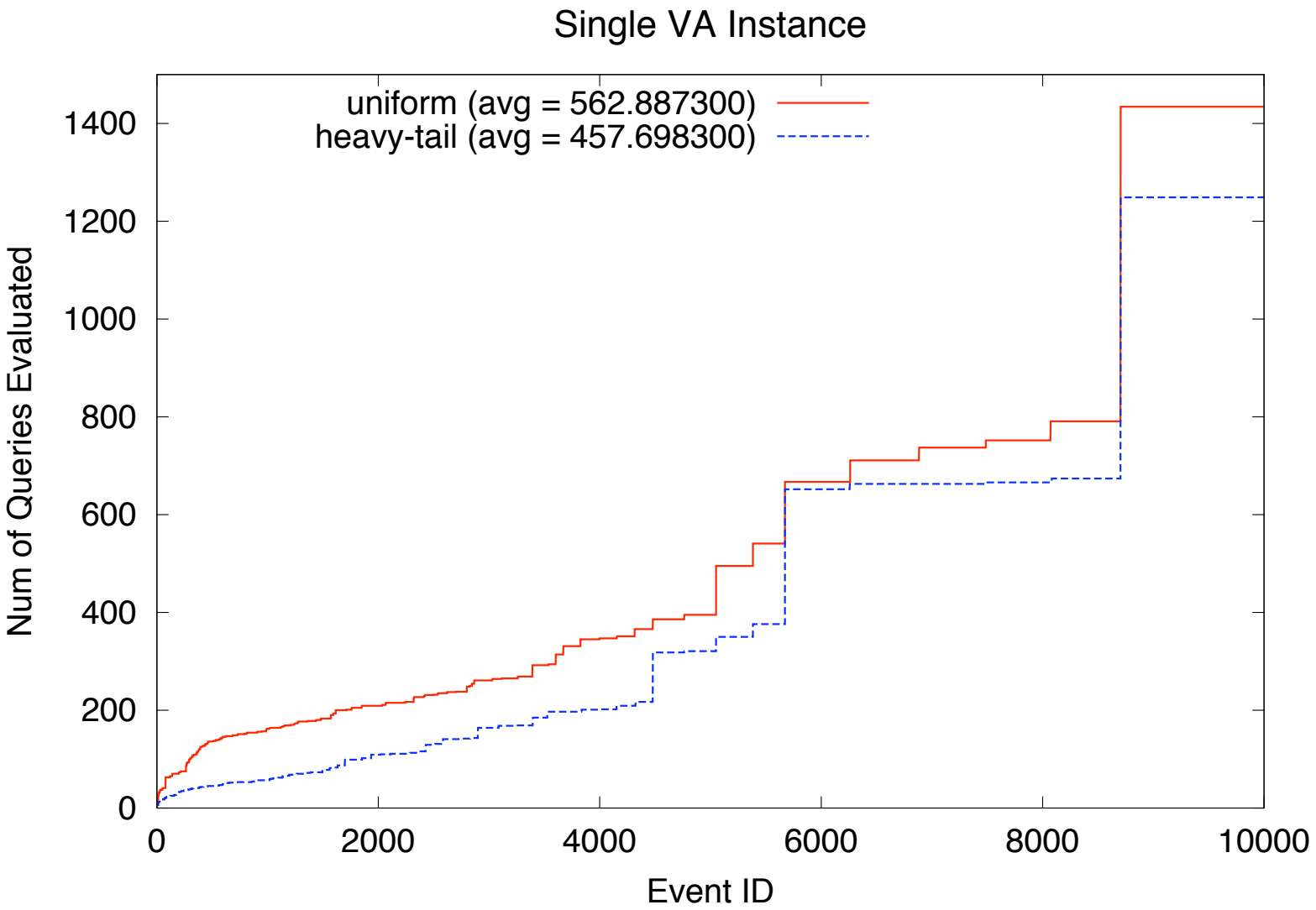
Subscription Efficiency: Communication Cost



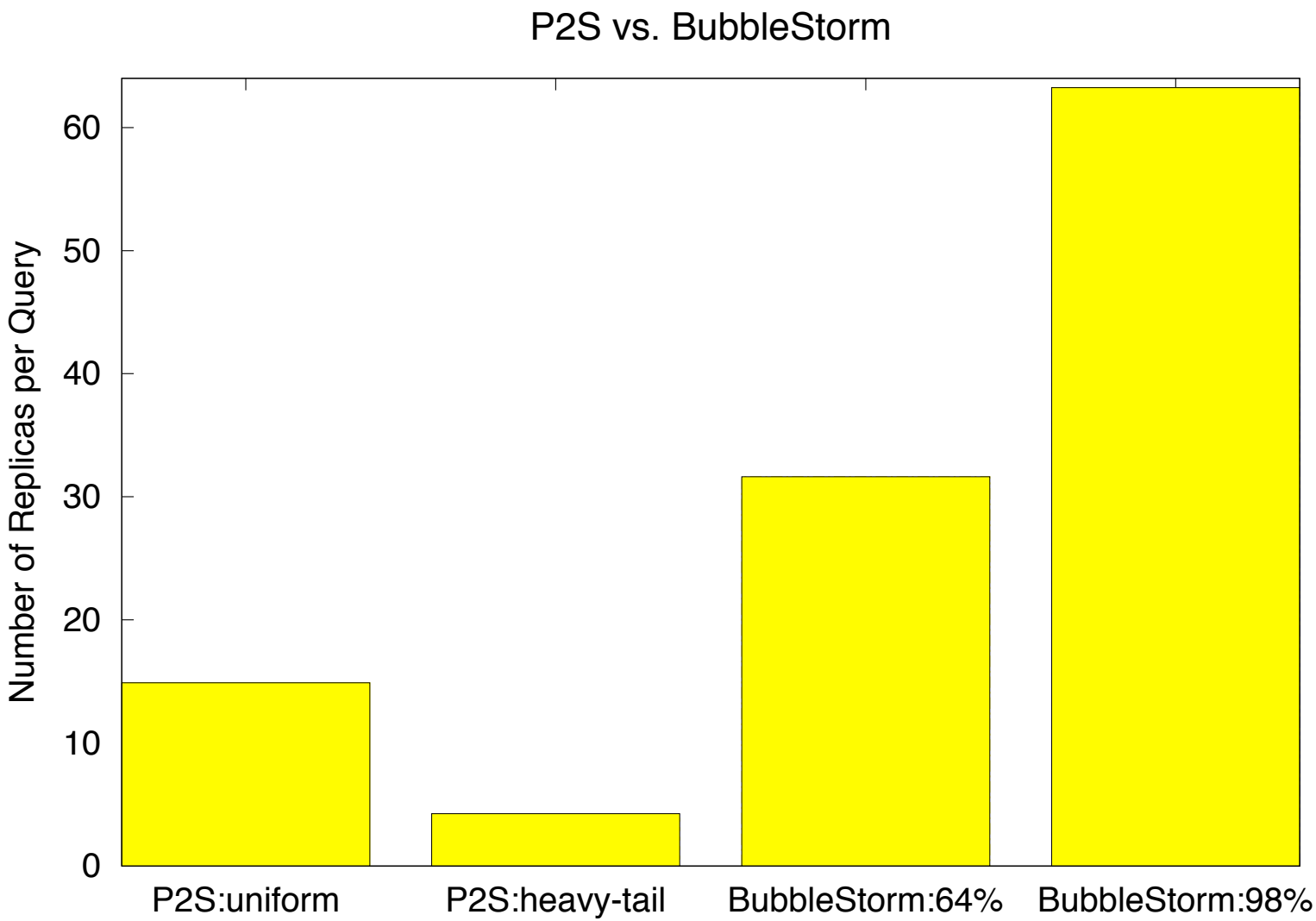
Notification Efficiency: Communication Cost



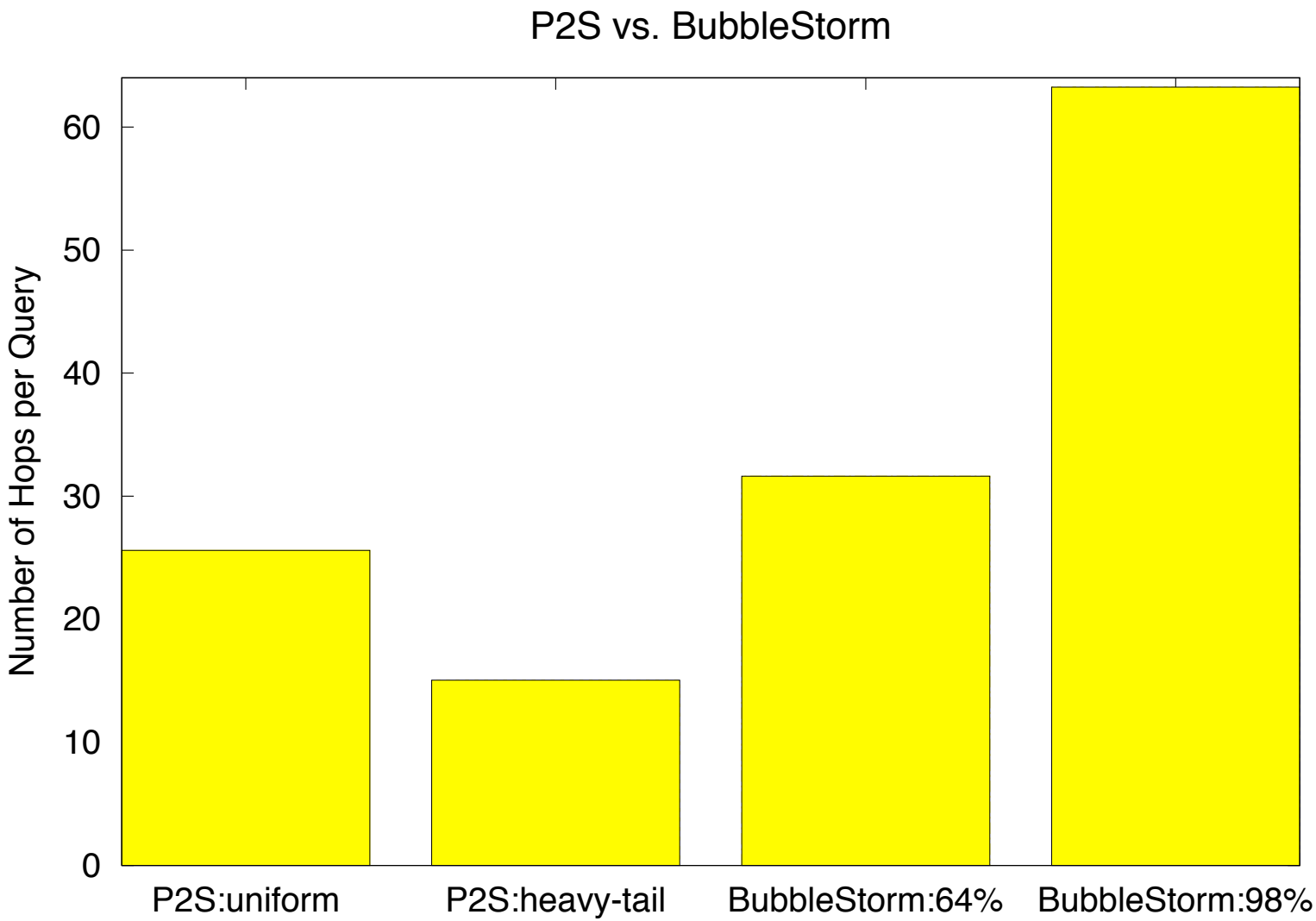
Notification Efficiency: Computation Cost



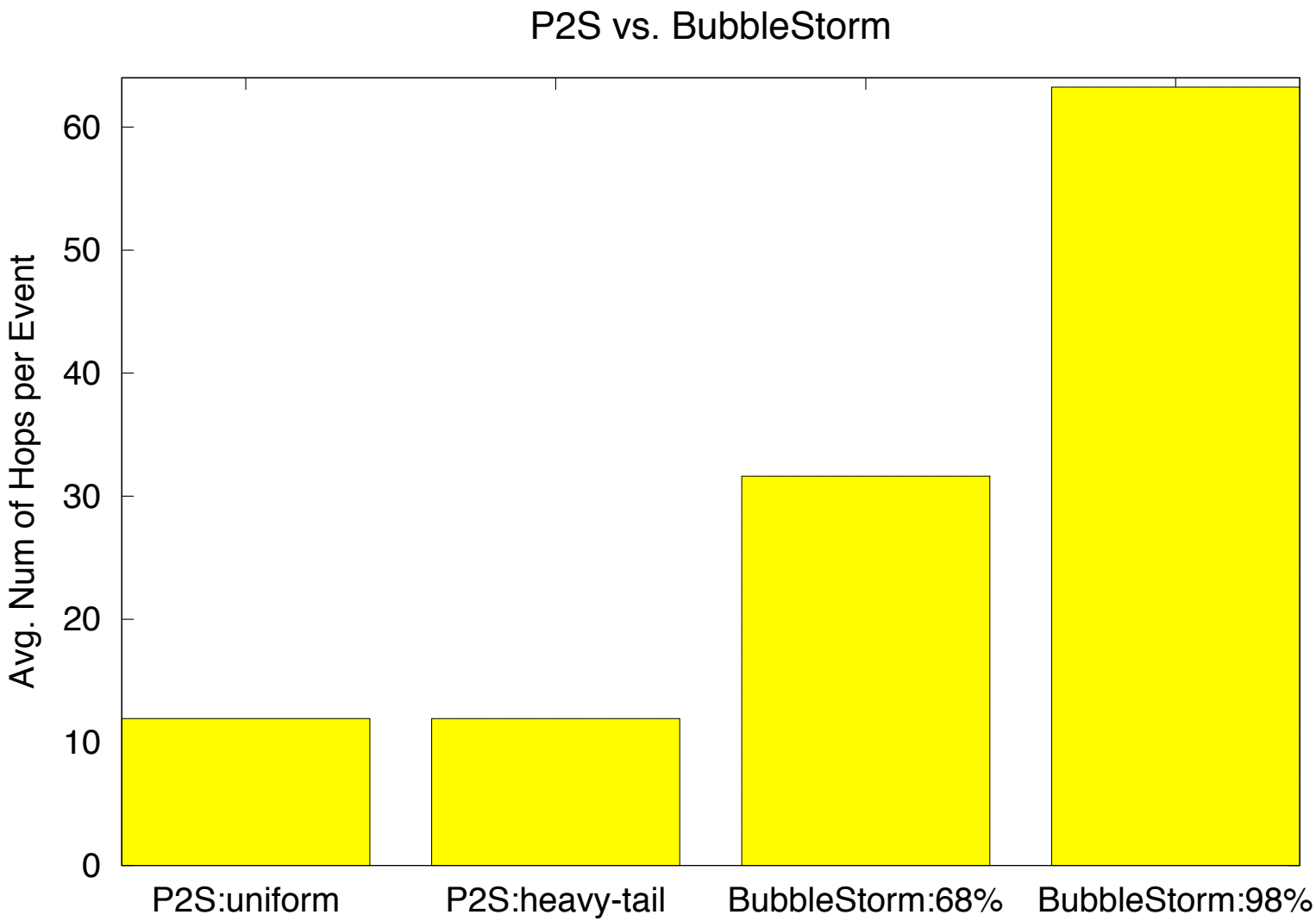
Pub-2-Sub vs. BubbleStorm: Storage Cost



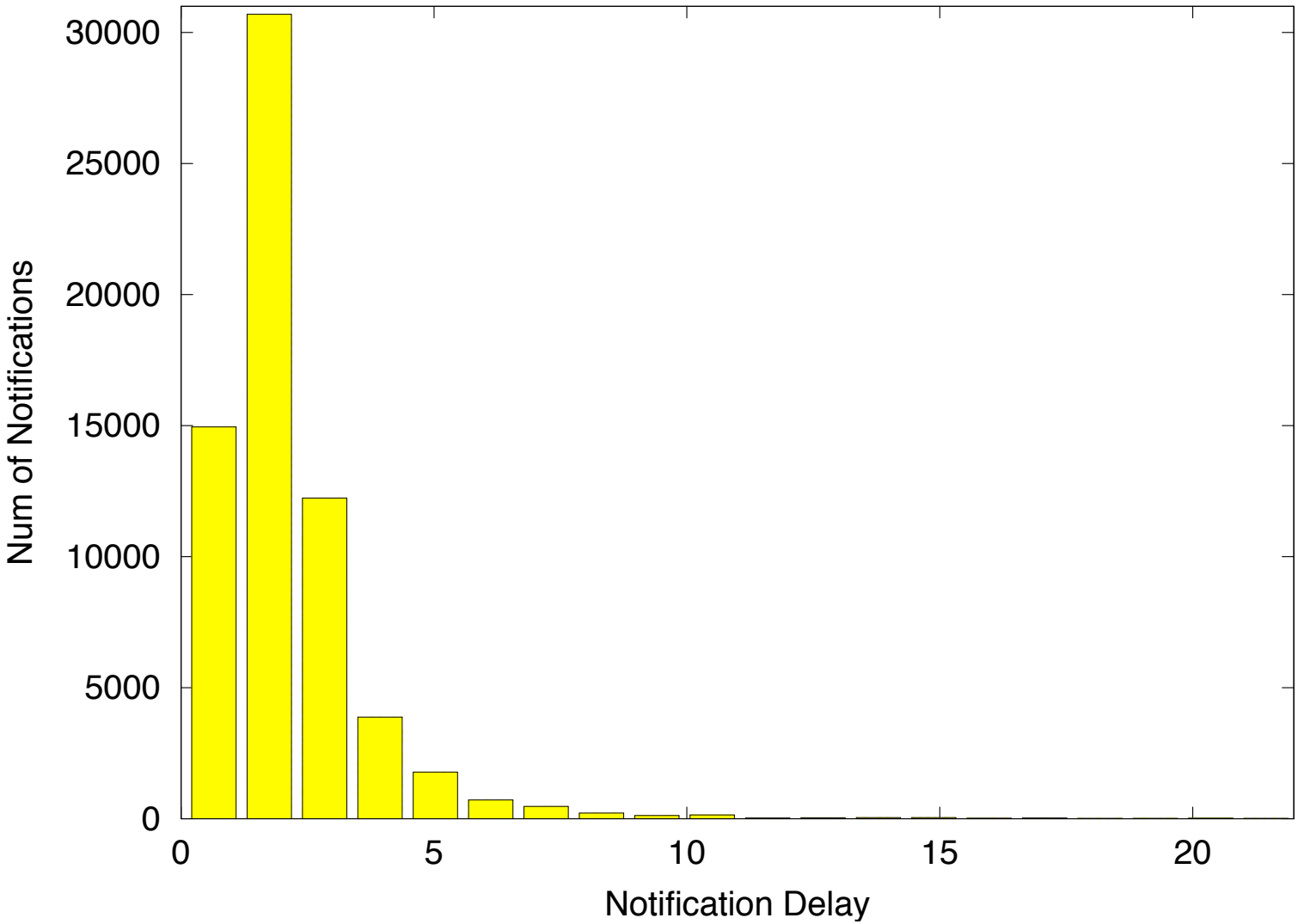
Pub-2-Sub vs. BubbleStorm: Subscription's Communication Cost



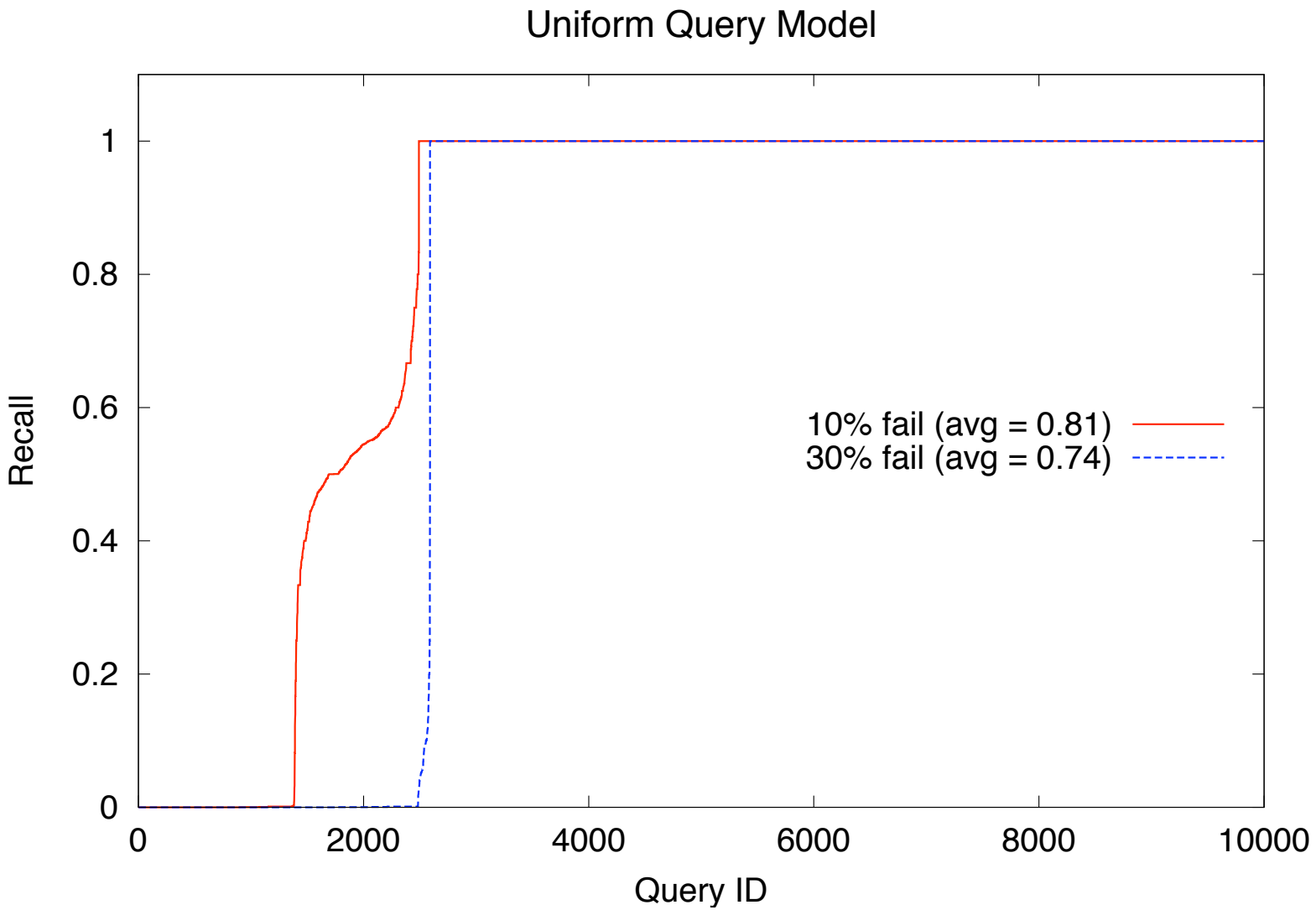
Pub-2-Sub vs. BubbleStorm: Notification's Communication Cost



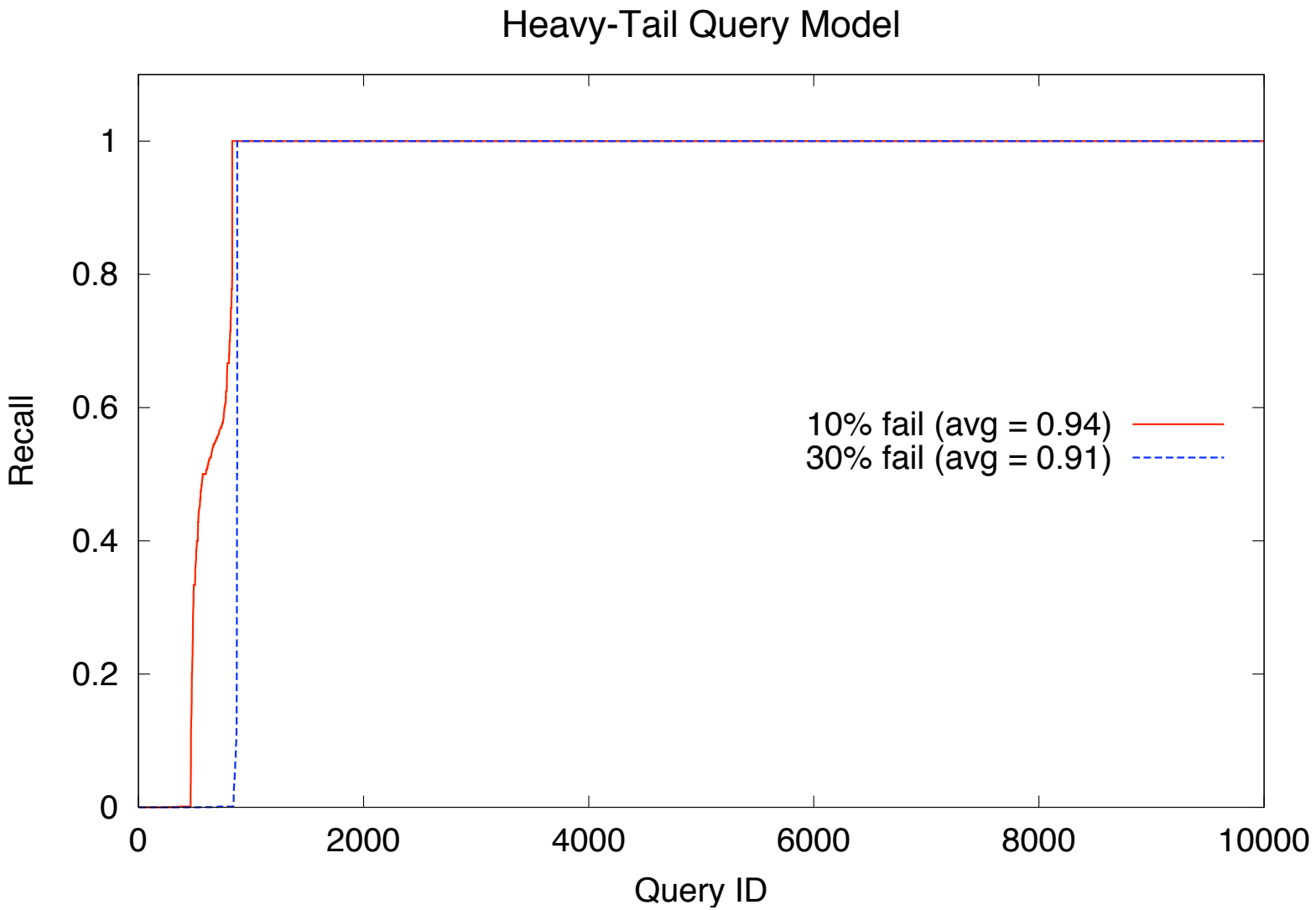
Notification Delay



Failure Effect: Uniform Case

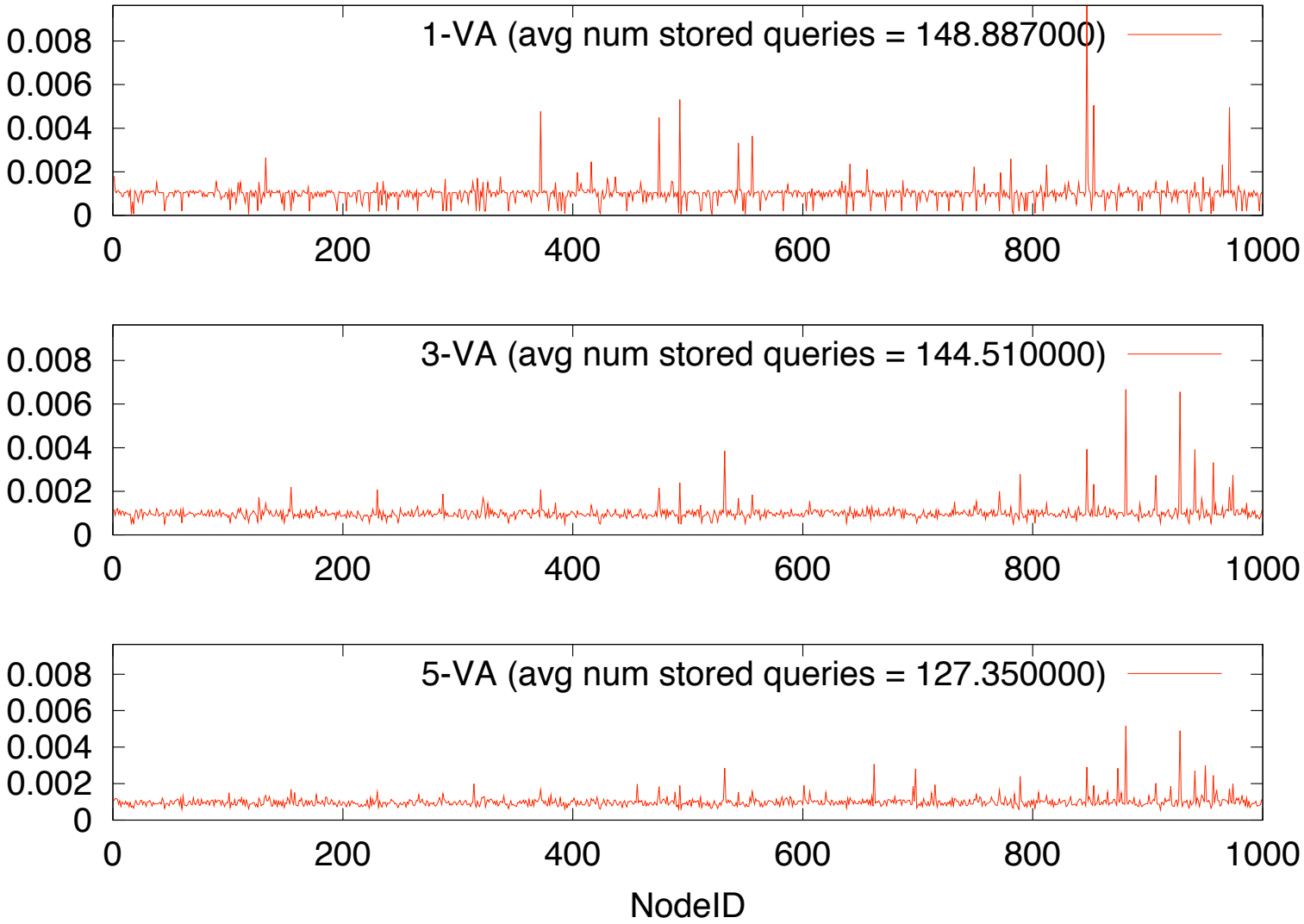


Failure Effect: Heavy-Tail Case



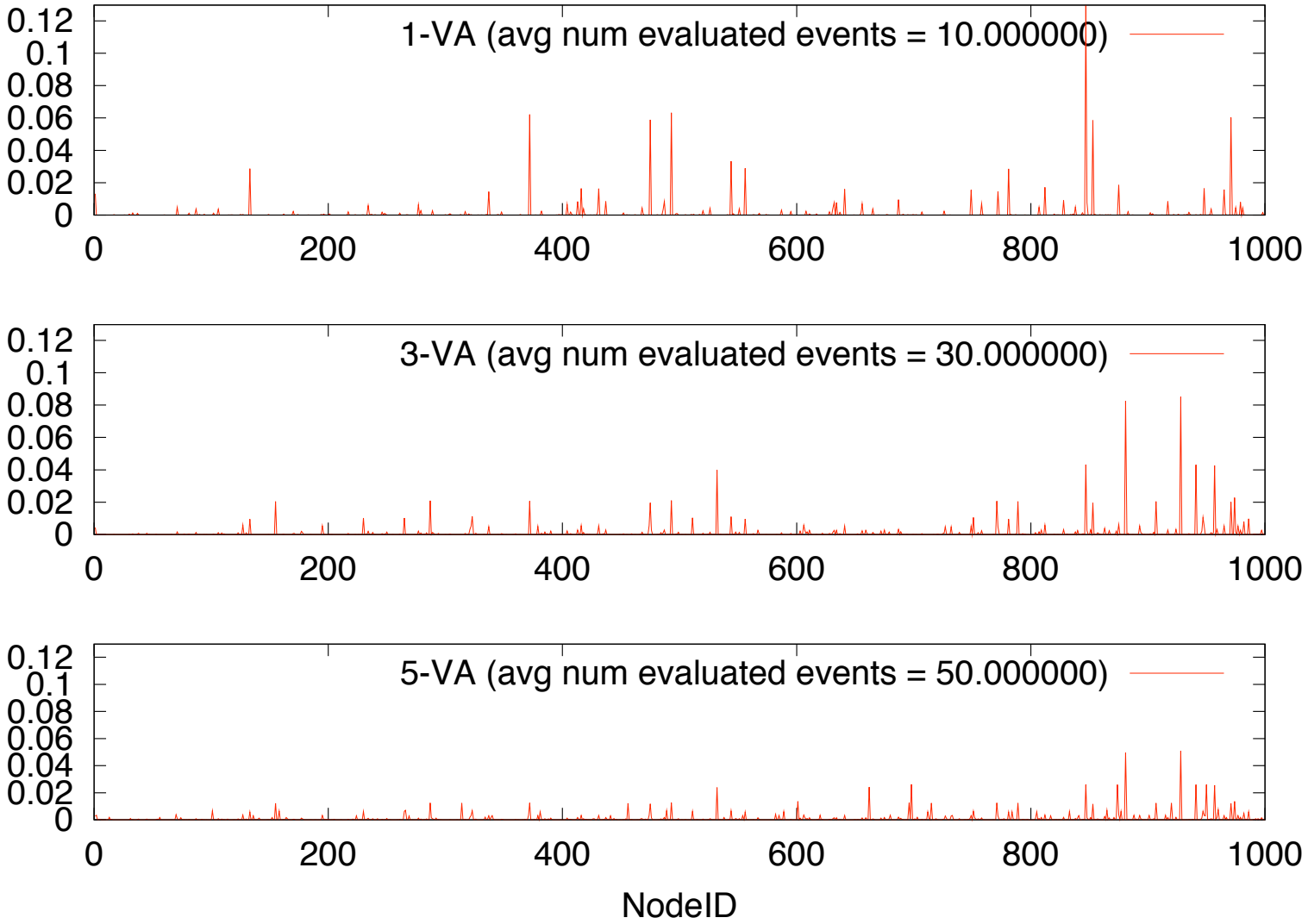
Load Balancing with Multiple VA-Instances

Storage Load



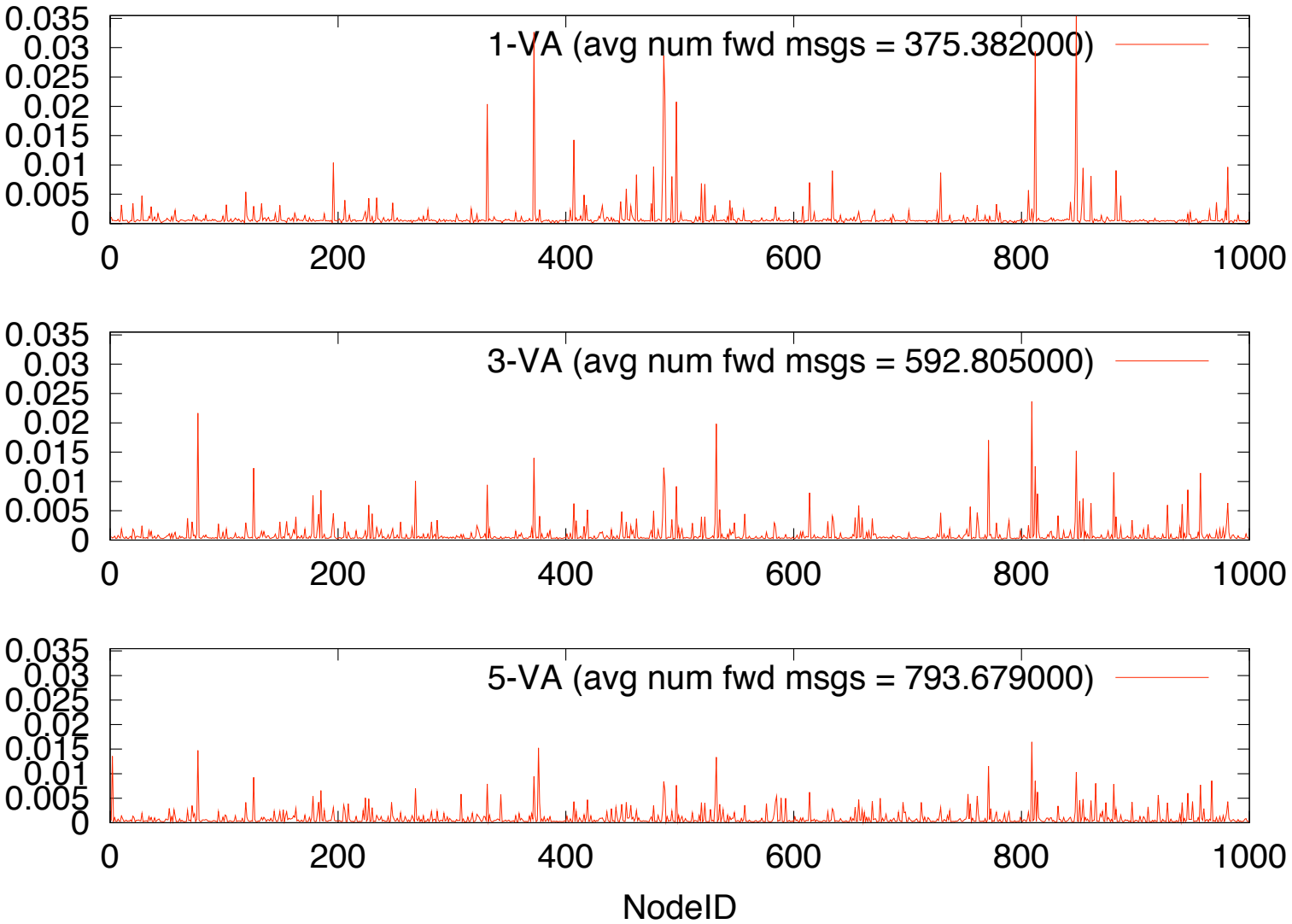
Load Balancing with Multiple VA-Instances

Computation Load



Load Balancing with Multiple VA-Instances

Communication Load



Summary

- P2P: a popular design adopted by organizations, institutions, and content providers → cooperative P2P networks
- Publish/subscribe: an important search model

PUB-2-SUB: A pub/sub framework

- Good for cooperative P2P networks
  - Work with any unstructured network, using only existing peer links
  - Work with any dimensionality
  - Multiple services can run simultaneously
  - Efficient in cost and time, better than the gossip-based approach
- Not recommended for use with highly dynamic P2P networks

- Future work: Better load balancing? Better failure recovery?
- Acknowledgement: NSF Grants CNS-0615055, CNS-0753066, and UMass Boston's Proposal Development Grant

References

M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron.

SCRIBE: A large-scale and decentralized application-level multicast infrastructure.
IEEE Journal on Selected Areas in communications (JSAC), 20(8):1489–1499, 2002.

R. Guerraoui, S. Handurukande, K. Huguenin, A.-M. Kermarrec, F. Le Fessant, and E. Riviere.

GosSkip, an Efficient, Fault-Tolerant and Self Organizing Overlay Using Gossip-based Construction and Skip-Lists principles.
In *IEEE International Conference on Peer-to-Peer Computing*, 2006.

A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi.

Meghdoot: content-based publish/subscribe over p2p networks.
In *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 254–273, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

A. Medina, A. Lakhina, I. Matta, and J. Byers.

Brite: An approach to universal topology generation.
In *MASCOTS '01: Proceedings of the 9th International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, page 346, Washington, DC, USA, 2001. IEEE Computer Society.

P. R. Pietzuch and J. Bacon.

Peer-to-peer overlay broker networks in an event-based middleware.
In *DEBS '03: Proceedings of the 2nd international workshop on Distributed event-based systems*, pages 1–8, New York, NY, USA, 2003. ACM.

W. W. Terpstra, J. Kangasharju, C. Leng, and A. P. Buchmann.

Bubblestorm: resilient, probabilistic, and exhaustive peer-to-peer search.
In *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 49–60, New York, NY, USA, 2007. ACM.

S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz.

Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination.
In *NOSSDAV '01: Proceedings of the 11th international workshop on Network and operating systems support for digital*

THANK YOU!!!

Please visit us: http://nislab.cs.umb.edu