XL Publish Subscribe systems
Pub-sub systems

- Asynchronous (loosely coupled) event notification system
- A set of Subscribers register their interest (subscriptions)
- A set of Publishers issue some events (events)

- Publish-subscribe system
  1. Manage users subscriptions
  2. Match published events against subscriptions
  3. Disseminate events to matching subscribers (and no others)

- Flexible and seamless messaging substrate for applications
Pub/Sub systems

• Prominent way of disseminating information
  – Social networks
  – RSS feeds
  – Recommendations

• Triggers Web navigation

• Importance of scalable systems.

• Centralized solutions on the market (Tibco, Vitria)
  – Scalable but limited expressiveness
  – Highly expressive but limited scalability
  – Can we do better in decentralizing such systems?
A scalable communication paradigm

• Scalability refers to
  – Number of subscriptions/events/class of events

• Decoupling [Eugster et al, ACM CSUR 2003]
  – Time decoupling (rdv point)
  – Space decoupling
  – Synchronization decoupling (no blocking)
Pub-sub systems: expressiveness

• Differences in subscription expressiveness

• System classification
  – Topic-based ~ Application-level multicast
    \[ \text{topic=houses_sales} \]
  – Content-based
    • Attribute-based
      \[ s_1=(\text{city=Rennes}) \land (\text{capacity=2\_Bedrooms}) \]
    • Range queries
      \[ s_1=(\text{city=Rennes} \lor \text{Saint Malo}) \land (\text{capacity=3\_Bedrooms} \lor \text{price < 300,000 EUR}) \]
Topic-Based Pub/Sub over Unstructured Overlays

TERA [Baldoni & al, 2007]
Tera: Overview

• TERA: Topic-based Event Routing Architecture
  – a topic-based publish/subscribe system
  – based on « informed gossip »
  – Intended for ‘unmanaged’ (not centrally controlled) systems

• Supports
  – Dynamic creation and deletion of topics
  – Efficient event routing with:
    • Reasonable number of ‘extra’ messages
    • High event delivery probability
    • Absence of hot spots (i.e. no overloaded nodes)
    • Scalability
Tera: Fundamental Ideas

• **Interest clustering:**
  – Subscribers with similar interests are clustered together.
  – Ideally, given an event, all and only the subscribers interested in that event should be grouped in a single cluster.

• Event delivery consists of two phases:
  – **Outer-cluster routing:**
    • Bring each event from the node where it is published, to at least one interested subscriber node.
  – **Inner-cluster diffusion:**
    • Once a subscriber receives an event it can simply broadcast it in the cluster it is part of.
**Tera: Overlay Structuring**

A two-layer infrastructure:
- One ‘universal’ overlay network connecting all nodes
- Per-topic overlays

Event delivery:
- Access point lookup table for outer-cluster routing (An event is first routed to a single matching subscriber – part of the ‘topic’ overlay – the *access point.*)
- Then, event is ‘flooded’ within the topic overlay.
Tera: Per-node State

- **View**: set of randomly chosen nodes
  - \( \sim \) random-graph overlay (high connectivity + low diameter)
  - Relies on a uniform node sampling service
    - Returns a random sample (view) of the universal overlay
    - Also utilizes a size estimation service
      - Estimates the popularity (i.e. size) of topic overlays

- **Access Point Table**:
  - List of pairs ( \( t: \text{topic}; n: \text{node in cluster for } t \) )
  - \( n \) is an access point for topic \( t \)

- **Subscription Table**:
  - List of pairs
    - \( t: \text{topic the node has subscribed; } i: \text{overlay ID for } t \)
Tera: Subscription Management

- Subscription table (list of \( <t,i> \))
- New subscription for topic \( t \), with overlayID \( i \), at node \( n \):
  - Add \( (t, i) \) to local subscription table
  - Add \( n \) to overlay \( i \) : **Route to a node in overlay \( i \)**:
    - Look-up to the Access Point Lookup (APT) table
    - Using decentralized search with random walks
      » Checking the APTs at intermediate nodes.
  - If no existing overlay, creation

- Periodically, nodes send their subscription tables to a random set of other nodes
  - Along with the size estimation
  - Nodes *build their APTs* from this info.

- Subscription deletion reverses actions of subscription creation.
Tera: Event Management

- An event \textit{e for topic t} is first routed to a single matching subscriber – part of the ‘topic’ overlay – the \textit{access point}.

- To find access point for \textit{t},
  - a decentralized search (using random walks on the APTs) of nodes is conducted.
  - \textit{If the lookup returns an empty list, the event is discarded}.

- Then, event is ‘flooded’ within the topic overlay – \textit{inner-cluster routing}. 

November 2012
Tera: Using APTs for Outer-Cluster Routing

<table>
<thead>
<tr>
<th>topic</th>
<th>AP</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>B1</td>
</tr>
<tr>
<td>y</td>
<td>B6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>topic</th>
<th>AP</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>B1</td>
</tr>
<tr>
<td>a</td>
<td>B5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>topic</th>
<th>AP</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>B4</td>
</tr>
<tr>
<td>h</td>
<td>B4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>topic</th>
<th>AP</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>B5</td>
</tr>
<tr>
<td>f</td>
<td>B6</td>
</tr>
</tbody>
</table>
Access point lookup

• Each APT is a cache
• Continuous updates
  – Upon each subscription advertisement for topic $t$ from $n$,
    • Update of the entry
    • Creation with probability $1/P_t$ where $P_t$ is the popularity of the topic estimated by $n$
    • When an Apt exceeds a predefined size, randomly chosen entries removed
Properties

• Non stale entries (live entries win)
• Inactive topics disappear
• Each access point is a uniform random sample of the population of nodes subscribed to that topic
  – PSS
• Size of APT limited
  – (highly popular topics advertized more)
Tera: Random-walk-based topic search

• Nodes only have a limited knowledge of the active topics
• Nodes perform random walks to search for access points in APTs stored at other nodes
• Given the randomness of both the global overlay topology and the APTs’ content, it is possible to fix the sizes of the walks and of the APT tables so that either
  – An access point will be found
  – Or, the topic can be safely considered as inactive
  – Trade-off APT size, length of random walk
Lookup

Apt=400
Apt=100
Apt=50

5000 subs
1000 topics

Success rate

Random walk’s length

November 2012
Figure 6. The plots show the average number of messages needed by TERA to notify an event when the number of subscriptions (a), of topics (b), the event publication rate (c) and the total number of nodes in the system (d) varies. For each figure, results from a simple event flooding algorithm are reported for comparison.
Figure 6. The plots show the average number of messages needed by TERA to notify an event when the number of subscriptions (a), of topics (b), the event publication rate (c) and the total number of nodes in the system (d) varies. For each figure, results from a simple event flooding algorithm are reported for comparison.
Tera: Partition management

• If $|\text{APT}|$ is too small, an existing access point may not be found, during subscription creation.
  → system may end up containing $>1$ distinct topic overlays for same topic.
• TERA employs a partition recovery mechanism, whose purpose is to detect the presence of partitioned topic overlays and merge them
  – Detection happens when a member of one partition meets a member of another partition (in a global view shuffle)
  – Recovery consists in having both nodes exchange their topic views.
Tera: Summary

• Paid particular attention to outer-cluster routing!
• Replicate enough the topic access points for successful random walk-based lookups
  – Used for event propagation and subscription management
  – Depends on topic popularity
  – Relies on peer sampling
• Performance evaluation
  – Reliable event dissemination
  – Load fairly balanced
Content-based Pub/Sub
Content-based: \((\text{semi-})\text{centralized solutions}\)

- Current systems: One or more centralized servers (brokers)
  - e.g., Tibco \((\text{Web Services Eventing})\)

- Servers become a bottleneck/single point of failure
  - Reverse Path Forwarding
    - Notifications follow reverse paths of subscriptions
  - Brokers deliver events to interested subscribers
    - Brokers end up maintaining the whole set of subscriptions
      - network size increases
      - node churn increases
      - more events are published

- Triggered interest in decentralized P2P solutions
Content-Based Pub/Sub over DHTs

Meghdoot [Gupta et al, 2004]
Meghdoot: An Overview

• Content-based publish-subscribe over P2P networks
• Build over CAN
• Subscriptions/events are expressed over a schema which consists of multiple attributes

\[ S = \{A_1, A_2, \ldots, A_n\} \]

where each \( A_i \) is an attribute

– Each attribute is specified by a name, a type and described as \{name: type, min, max\}
– Data types: integer, floating point, character strings
Meghdoot: Subscriptions & events

- **Subscriptions**: a conjunction of predicates over one or more attributes
  - Operators, =, <, >
  - Exact value, or ranges
  - Example \( S = (A_1 \geq v_1) \land (v_2 \leq A_2 \leq v_3) \)

- **Events**: exact values over (a subset of) the attributes in schema \( e = \{A_1 = c_1, A_2 = c_2, \ldots, A_n = c_n\} \)

- \( E \) matches \( s \) if each predicate of \( s \) is satisfied by \( e \)
Meghdoot: Logical space construction

• For a schema with n attributes, a 2n-dimensional Cartesian space is constructed.

• This space is “indexed” using CAN.

Attribute $A_i$ with domain $[L_i, H_i]$ corresponds to dimensions $2i-1$ and $2i$

• Ranges $[l, h]$ are represented by a point in the logical space
  – Start of the range, $l$, of ith attribute is mapped to dimension $2i-1$
  – End of the range, $h$, is mapped to dimension $2i$
Meghdoot: Subscription installation

\[ S = (l_1 \leq A_1 \leq h_1) \land (l_2 \leq A_2 \leq h_2) \land \ldots \land (l_n \leq A_n \leq h_n) \]

S is mapped to the subscription point \([l_1, h_1, l_2, h_2, \ldots, l_n, h_n]\)

S will be stored at a peer of the zone containing the subscription point

S arrives at Po and is routed to Pt
Meghdoot: Event delivery

\[ e = \{ A_1 = c_1, A_2 = c_2, \ldots, A_n = c_n \} \]

\[ e \text{ is mapped to event point } [c_{11}, c_{12}, c_{21}, c_{22}, \ldots, c_{n1}, c_{n2}] \]

If \( A_i = v \) then \( c_{i1} = v \) and \( c_{i2} = v \)

\[ S = (l_1 \leq A_1 \leq h_1) \land (l_2 \leq A_2 \leq h_2) \land \ldots \land (l_n \leq A_n \leq h_n) \]

is affected by \( e \) if \( \forall i \in \{1, 2, \ldots, n\} \ (l_i \leq c_{i1}) \land (c_{i2} \leq h_i) \)
Meghdoot: Event delivery

- The event space of interest to $S$ (diagonal)
- The subscription space affected by event $e$
Meghdoot: Event delivery

• First, the event is routed to the zone responsible for the event point.
• Then, the following algorithm is executed.

\[ \text{Deliver Event} \ (z, e) \ /* \text{executed at zone } z \text{ for event } e */ \]

1. for all subscriptions \( S \) stored at \( z \)
2. if (matchedSubscriptions\((S, e)\))
3. notifySubscribers\((S, e)\)
4. endfor
5. for all neighbors \( n \) of \( z \)
6. if (eventRegion\((n, e)\)) and
7. upperleftNeighbor\((n, z)\)
8. DeliverEvent\((n, e)\)
9. endfor

• Eventregion\((n, e)\) is true if region of \( n \) intersects with affected region of \( e \)
• UpperleftNeighbor\((n, z)\) is true if \( n \) lies in the upper left region of zone \( e \), avoid propagation in reverse order
Meghdoot: Peer management

• Join: peer A joins, thru known peer C:
  – A peer A can use the simple CAN joining algorithm and contact some existing peer C
  – C locates a *randomly generated point* in the logical space lying in zone managed by, say, V
  – V divides its zone and assigns one half of it to A

• **CAN** enjoys *uniform distribution*: *no longer the case...!*
  – Content-based distribution of data as a *result of data skews*...
Meghdoot: Load balancing

- Differences with storage systems
  - Popular events (values).
  - Skewed subscription-values distribution
  - Event spreading

- Meghdoot uses the content of the subscriptions rather than a “uniform hash” function to place data on peers

- Peers
  - Store subscriptions (and thus serve events to them)
  - Route events (and subscriptions) to their appropriate zones.
**Meghdoot:** Load characteristics

- **Subscription load**
  - Subscription stored at the peer managing the zone the subscription point lies in
  - Load proportional to the number of stored subs
  - New peers can be directed to heavily loaded peers which split their zones given the subscription load so to equally divide the subscription load.

![Uniform splitting](image1)

![Sub-based splitting](image2)
Meghdoot: Load characteristics

• An event generates load because it needs to be propagated.
• Splitting doesn’t help: a peer/zone will continue to be in critical propagation path
• Creating alternate propagation paths, when new nodes join
  – Zone replication for load distribution
  – Increase the state to maintain alternative neighbors
  – Round robin routing events to alternative neighbors

November 2012
Meghdoot: Peer join

• A wishes to join
• A contacts a well-known peer C
• C looks for an overloaded peer
  – Maintains information about neighbors’ load
  – Each peer maintains an estimated list of the $k$ most heavily loaded peers
  – Maintain statistics about the network
  – Splitting or replication depending on whether heavy load is due to:
    • Subscription load (then split zone)
    • Event propagation load (then replicate zone)
Meghdoot: Peer departure or failures

- Peer departure
  - Upon departure, a peer checks if its zone is replicated
  - If the zone is replicated (for load balancing) then it leaves the system
  - Otherwise, find a neighbor willing to take over its load
- Peer failure
  - Failure detection by neighbors
  - When installing a subscription point, install a mirror replica
Experimental evaluation results

- **System scalability**
  - ½ of events delivered by contacting 5% of the peers in a 100 node network
  - 95% of events delivered by contacting 5% of the peers in a 10,000 node network

- **Load balancing**
  - 10,000 peers, 115,000 events, most heavily loaded peers receives 0.123% of the total messages
5.2 System Scalability

In this section we present the evaluation results for the scalability of the system when the number of peers in the system is varied. We also analyzed the effect of varying the number of subscriptions and the number of events on the system.

Figure 9 shows the scalability performance of the system. The x-axis in the plot represents the percentage of peers contacted to deliver an event out of the total number of peers that were present in the system when the event was generated. The buckets on the x-axis have been recalibrated because the values 60-100 on the x-axis had no data points and the range 0-20 has been expanded.

Figure 9(a) shows the scalability of the system for the synthetic event set where the events are distributed uniformly in the domain. In the case of 100 peers, more than half of the events were delivered to all the relevant subscriptions by contacting at most up to 5% of the peers only. For the case of 10,000 peers, 95% of the events were delivered to all affected subscriptions by contacting less than 5% of the peers. In fact, almost all the events were delivered by contacting less than 10% of the peers. Overall, as the number of peers in the system increases, the peers needed to be contacted for delivering events scales very well.

Figure 9(b) shows the results for the simulations with the real event set, which is highly skewed. Because of the skewed distribution of the event data, more zones are created in the vicinity of event points which leads to a small increase in the number of peers contacted to deliver events. Even with skewed events, in the cases of 100 and 1000 peers, 85%-90% are delivered by contacting at most 15% of the peers. For the experiment with 10,000 peers, almost all events are delivered by contacting less than 10% of the peers and 97% of those events contact less than 5% of the peers. This strengthens our conclusion that Meghdoot scales very well as the number of peers in the system increases.

We performed an experiment with 10,000 peers in the system by varying the number of subscriptions installed in the system from 25,000 to 150,000.
Content-based pub/sub over unstructured overlays: Sub-2-Sub

[Voulgaris & al, IPTPS 2006]
Sub-2-Sub

• Peers are subscriptions
  – Rather than physical nodes
  – Each peer manages its own subscription(s)
  – The more it subscribes, the more it contributes

• Self-organizing overlay
  – Eventually cluster similar subscriptions
  – Efficient event dissemination structure
  – Adapted to dynamism

Gossip-based algorithm to cluster peers according to their interests
Sub-2-Sub: definitions

- Assume N attributes (real numbers)
  - $A_1$, $A_2$, ..., $A_N$
  - The N-hyperspace

- Subscriptions are range (trivially exact) predicates on one or more attributes
  - E.g. $A_2 == 3.07$ && $(2.5 < A_4 < 4.7)$
  - A N-hypercube

- Events define exact values for all attributes
  - E.g. $\{A_1, A_2, A_3, A_4\} = \{3, 0, 7, 10.5\}$
  - A point

- The set of all possible events define the event space, It’s an continuous space of dimension N
Sub-2-Sub: Key Concept

“Partition event space in homogeneous subspaces”
(homogeneous subspace: all its events have the same subscribers)
Subscription

\[ S_v \]

\[ S_t \]

\[ S_l \]

\[ S_i \]

\[ S_k \]

\[ S_j \]

Event \( e \) (\( a=10 \))

Value of attribute \( a \)
Sub-2-Sub: Operation

1. Let subscribers of “near” subspaces discover each other
2. Organize subscribers of each subspace in a ring
3. To publish an event, navigate to the right subspace, and hand the event to any one subscriber
   - Event reaches all and only interested subscribers, autonomously!
Sub-2-Sub overlay creation

- Maintain connectivity
- Peer sampling service
- Create clusters of “related” subscribers
- Clustering service
- Organize subscribers within a subspace in a ring
- Ranking service
Maintaining connectivity

• Connectivity = no overlay-network partition

• Peer sampling service: Cyclon

Forming clusters: gossip-based clustering

- Keep a small fixed-sized set of neighbors with similar interests

- Similarity is based on a notion of distance
  - the minimum Euclidean distance between two subscriptions
  (Note: Distance 0 means some overlapping interests)

- peerSelect()
  - Choose a neighbor in the random view provided by peer sampling

- select()
  - Keep neighbors of smallest distance
Organizing clusters in rings: gossip-based structuring

• Each subscription is given a fully random ID upon creation
  – Total order on IDs
  – Defined only to permit ring creation

• peerSelect()
  – Choose a neighbor in the similar interest view

• update()
  – Keep neighbors of smallest distance

• Distance definition :
  – 0 (ZERO), if overlapping and ID is the nearest for part of the subscriptions overlap
  – INFINITE, otherwise

• update() keeps neighbors whose ID is nearer from the subscription ID for any portion of subscription hypercube
  – Size of the neighbor set depends on subscription width
Sub-2-Sub Architecture

- Three-layer architecture
- Each layer gossips to a neighbor’s respective layer

- Ring of subscribers (structured peer sampling)
- Overlapping subscribers (biased peer sampling)
- Random subscribers (peer sampling)
Sub-2-Sub Architecture

- RPS finds random links (needed for BPS)
- and keeps the overlay connected
Sub-2-Sub Architecture

Proximity between 2 subscribers =
- 0 (ZERO), if overlapping
- the Euclidean distance between the 2 hypercubes, otherwise

Attribute value
Sub-2-Sub Architecture

- Proximity between 2 subscribers:
  - 0 (ZERO), if overlapping and “visible”
  - INFINITE, otherwise
- Variable length view

Diagram:

- Ring of subscribers (structured peer sampling)
- Overlapping subscribers (biased peer sampling)
- Random subscribers (peer sampling)
Sub-2-Sub in a nutshell

Random links
- Traditional random-based epidemic algorithms

Interest links
- Peer selection and links kept based on proximity in the attribute space
  \[ d(i,j) = 0 \text{ if no overlap} \]
  \[ S_i = [l^i_1, r^i_1] \times [l^i_n, r^i_n] \]
  \[ d(i, j) = \sqrt{\sum_{k=1}^{n} (\min(r^i_k, r^j_k) - \max(l^i_k, l^j_k))^2} \]

Structured links
- Peer selection: in the ring
- Links kept, sorted according to growing id
- Subscription cover
Dissemination of events

• The event is sent to any of the subscription peer
  – Greedy routing using Euclidean distance along random neighbors and interest proximity links
  – Eventually reaches one of the interested subscriber ⇒ dissemination begins

• A node receiving an event for the first time, forwards it:
  – Along its two ring links
  – To one random subscriber interested in the event (if exists)
Sub-2-Sub simulations

- Experimental evaluation using the *peersim* open source simulation framework
- Simulation settings
  - 10,000 subscriptions
  - 3 attributes
  - Distribution of interests:
    - Uniform
    - Power-law (Zipf) interest distribution:
      (representative of real world behaviors)
  - Distribution of subscription width (for all attributes):
    - Power-law width distribution
  - Evaluated by periodically posting 10,000 random events
Distribution of ring lengths

Number of subscribers in each ring: *powerlaw* is representative of typical real world interest distributions, with some very popular values and a heavy tail of less popular values.
Construction: Subs. Insertion

Represent the distribution of cycles needed for subscriptions to reach their exact position in a ring (powerlaw interests)
Construction: Rings completion

All nodes bootstrap at the same time. Uniform interest distributions

Percentages of: ring links established and completely created rings (in both directions)
Construction: Rings completion

All nodes bootstrap at the same time.

Powerlaw interest distributions

Percentages of ring links established and completely created rings (in both directions)
Dissemination: routing

Number of route hops needed to reach the first interested subscriber prior to dissemination

![Graph showing number of nodes vs. number of cycles for PUBLISHER joins]
Dissemination efficiency

All subscriptions bootstrap at the same time (random links only)

Complete dissemination = % of complete disseminations

Hit ratio = within non-complete disseminations, mean percentage of interested subscribers that get the event
Sub-2-Sub summary

• Content-based pub-sub on unstructured networks
  – Accurate → All and only interested nodes receive event
  – Autonomous → No need for extra device
  – Collaborative
  – Self-organized
  – Very scalable (nodes and attributes)
    • Experiments for 10 attributes present the same results
Range queries over DHTs

Mercury[Bharambe et al, 2004]
(some slides from the authors)
Introduction

• DHTs: exact match interface
  – Uniform load balancing
  – Not flexible

• Range-queries
  – E.g. keyword-based search in file sharing systems

• Challenges
  – Efficient routing
  – Load balancing
DHTs & range queries

• **DHTs**
  – Randomized hash functions
  – The hash of a range is not correlated to the hash of the values within a range.

• **Solutions**
  – Partition the space
  – Range queries = query in the corresponding partition.

× **Drawbacks**
  1. A priori partitioning difficult
  2. Not dynamic
  3. Complex implementation
Data Model

• Data item: list of typed attribute-value pairs (type, attribute, value)
  – Type: int, char, float and string.

• Query: conjunction of predicates i.e. tuples of the form (type, attribute, operator, value)
  – Operators: <, >, ≤, ≥, =.
  – String operators: prefix ("*n"), postfix ("j*")
Examples

Data

float  
   x-coord = 50

float  
   y-coord = 100

string 
   player = "john"

string 
   team = "topgunz"

int 
   score = 76

Query

float  
   x-coord < 53

float  
   x-coord > 34

string 
   player = "j*

int 
   score = "*"
Using DHTs for Range Queries

• No cryptographic hashing for key → identifier

Query: $6 \leq x \leq 13$

key = 6 → 0xab
key = 7 → 0xd3
...

Query: $6 \leq x \leq 13$
Using DHTs for Range Queries

- Nodes in popular regions can be overloaded
- Load imbalance!
Mercury in a nutshell

• Two main components
• Creates a routing hub for each attribute
  – Logical collection of nodes
  – Queries send to hubs corresponding to attributes of the query
• Logical ring
  – Data placed contiguously
  – Potential load balancing issue solved by aggregated system knowledge
DHTs with Load Balancing

• Mercury load balancing strategy
  – Re-adjust responsibilities

• Range ownerships are skewed!
DHTs with Load Balancing

Each routing hop may not reduce node-space by half!

⇒ no log(n) hop guarantee

Finger pointers get skewed!
Ideal Link Structure
Notations

• $A$: set of attributes in the overall schema
• $A_Q$: set of attributes in a query $Q$
• $A_D$: set of attributes in a data-record $D$
• $\pi_\alpha$: value/range of an attribute $\alpha$ in a data-record/query.
• $H_{\alpha}$: hub for attribute $\alpha$
• $r_\alpha$: a contiguous range of attribute values
Routing Data

• One hub, in which nodes are organized in a logical (ring) overlay, per attribute $H_a$
  A physical node can be part of multiple logical hubs

• Data $D$ sent to all hubs $H_b$ where $b$ is in $A_D$ (set of attributes of $A$)

• In each hub, the data is routed to the node responsible for the record’s value
Routing

• A node responsible for a range $r_a$
  – resolves all queries $Q$ for which $\pi_a(Q) \cap r_a \neq \emptyset$
  – stores all data-records $D$ for which $\pi_a(D) \in r_a$
    ! Ranges are assigned to nodes during the join process

• A query $Q$ is passed to exactly one hub $H_a$ where $\alpha$ is any attribute from the set of query attributes

• Within the chosen hub, the query is delivered and processed at all nodes that could have matching values
Replication

• It is not necessary to replicate entire data records across hubs.

• A node within one of the hubs can hold the data record while the other hubs can hold a pointer to the node
  ✓ Reduction of storage requirements
  ✗ One additional hop for query resolution
Routing within a hub

• Within a hub $H_a$, routing is done as follows:
  
  – for routing a data-record $D$, we route to the value $\pi_a(D)$.

  – for a query $Q$, $\pi_a(Q)$ is a range. Queries routed to the first value appearing in the range and then use the contiguity of range values to spread the query along the circle, as needed.
Example

• minimum value=0, maximum value=320 for the x and y attributes
• data-record sent to $H_x$ and $H_y$ and stored at nodes b and f respectively.
• The query enters $H_x$ at node d and is routed and processed at nodes b and c.
Additional requirements

• Each node must have a link to
  – the predecessor and successor nodes within its own hub
  – each of the other hubs (cross-hub links)

• We expect the number of hubs for a particular system to remain low
Design Rationale

• Attributes treated independently

✗ An alternate design would be to route using the values of all attributes present in $D$
  – Since each node in such a design is responsible for a value-range of every attribute, a query that contains a wild-card attribute can get flooded to all nodes

✓ By making the attributes independent, we restrict such flooding to at most one attribute hub.

✓ Furthermore, it is very likely some attribute of the query is more selective. Thus routing the query to that hub, can eliminate flooding.
Routing efficiency
(in a hub)

• Data structures
  – each node stores successor and predecessor links and maintains \( k \) long-distance links
  – This results to each node having a routing table of size \( k+2 \)

• The routing algorithm is simple:
  – let neighbor \( n_i \) be in charge of the range \( [l_i, r_i) \), and
  – \( d \) denotes the clockwise distance or value-distance between two nodes
  – When a node is asked to route a value \( v \), it chooses the neighbor \( n_i \) which minimizes \( d(l_i, v) \).

Small-world topology
Need to establish links based on node-distance

If we had the above information...

For finger $i$
  - Estimate value $v$ for which $2^i$ th node is responsible
Load balancing

• Random sampling to account for non uniformity of ranges
• Sends a sample-request with a TTL set to \( \log(n) \)
• The node at which the TTL expires sends back a sample
• Exchange estimates with a gossip-based protocol
Histogram Maintenance

- Measure node-density locally
- Gossip about it!

Node-density

Values

(Range, density)

Request sample

November 2012
Histgrams

- Used to establish load patterns
  - Average load (data and queries)
  - Nodes load
    - Load accounts for sucessor and predecessor
- Used for query selectivity (* versus others)
  - Estimate number of node/bucket (of values)
  - possible to gather this information from other hubs

Heavy loaded nodes sends requests to lightly loaded nodes
Underloaded nodes rejoined at the heavy loaded portion of the network
Load Balancing

• Basic idea: leave-rejoin
• Steps
  – Find average, check if heavy or light
  – Light nodes perform a leave and rejoin
Evaluation

• Workload
  – Several item insertions
  – Data chosen according to Zipfian distribution
  – Values near 0x00 most popular

• Key questions:
  – Are the histograms accurate?
  – Are the routes efficient?
Sampling Accuracy

- Estimate of total node count by each participant
  - 10000 nodes, Zipf-skewed distribution with load-balancing
Routing Performance

Graph showing the average number of hops required for different number of nodes, comparing Naive DHT, Mercury, and Ideal performance.
Load-balancing

Well-balanced nodes if \( \frac{1}{\delta} < \frac{\text{load}}{\text{average load}} / \delta \)

1 round of LB
- Each node samples
- Each node runs histogram maintenance algorithm
- Check histograms / potential rejoin
Conclusions

• DHTS are not ideal for range queries since the structures do not match

• Correct the load imbalance

• Sampling and histogram maintenance
  – Useful for efficient routing
  – Load balancing
  – Selectivity estimation

• Routing for range queries in P2P networks
  – Efficient in the face of skewed node ranges
  – Explicit load balancing
  – Multiple attributes
VoroNet
A scalable object network based on Voronoï tessellations

[Beaumont et al. IPDPS 07]
Design rationale

• Efficient data location service
  \[
  \text{Efficiency} = \text{expressiveness} + \text{completeness}
  \]

• Expressiveness versus completeness
  – Unstructured overlay/Structured overlays (DHT)

• Overlay structure should reflect the application one
  – Linking objects in an efficient routing overlay
  – Use of Voronoï tessellation of the object space
  – Efficient routing: Kleinberg small world model
Model

• An object is described by a set of attributes
  – Objects with “near” attributes are neighbours in the overlay
  – Multidimensional naming space
  – *For ease of explanation*
    • we limit to the case where dimension is 2

• Native and efficient support for efficient query mechanisms
  – Scalable, polylogarithmic routing
  – No hash mechanisms ⇒ Ordering preserved
  – Generalizes Kleinberg Small-World model

• State per object is O(1)
  – Independently of the object set size and distribution
  – The basic overlay is based on the Voronoï tessellation of the objects
    set in the Euclidean naming space
Application object
A peer in the VoroNet overlay

[0:1]x[0:1] objects space

Computing entities
Node $n_i$ Possess $o_i$ objects $\Rightarrow$ $n_i$ participates $o_i$ times in the overlay
Computing entities

Node $n_i$ Possess $o$ objects $\Rightarrow n_i$ participates $o$ times in the overlay

Voronoï Tessellation
of the set of objects

Application object
A peer in the VoroNet overlay

[0:1]x[0:1] objects space
Node $n_i$ possess $o$ objects $\Rightarrow n_i$ participates $o$ times in the overlay.
Voronoï tesselation

• Definition
  – For each point $p$ among a set
  – $p$’s cell contains all points nearest to $p$ than to any other point

• The dual of the Voronoï diagram is the Delaunay triangulation
  – Mean(#neighbors) $\leq 6$
  – Navigability: greedy Euclidean routing always succeeds (in linear number of steps)

• **Overlay primary links** between objects are adjacency links of objects (virtual) cells
Object insertion

- Each object knows
  - Neighbors coordinates and zones

- A joining peer \( p \) routes a message to its coordinate
  - Peer \( p_i \) is responsible for \( p \) insertion
  - \( p_i \) computes \( p \)'s new zone and modifications to its neighbors’ zones (e.g. \( p_j \) zones)
  - \( p_i \) disseminates changes to its neighbors and notify \( p \) of its new neighborhood
Efficient routing

• Greedy routing
  – Each routing step gets closer to the destination A
  – Delaunay triangulation properties ensure that this succeeds deterministically
  – But.....may be $O(N)$ steps

• Small world routing
  – Additional shortcuts
  – Extension of the Kleinberg’s model
  – Polylogarithmic routing in $N : O(\log^x(N))$
Extending the Kleinberg model

• Each object chooses a shortcut destination point according to a harmonic distribution, and uniform direction

• The topology is not a grid!
  – The destination point is not necessarily an object ...
  – But the destination stands in an object cell

• The object chosen as shortcut neighbour is always the object which has the shortcut destination in its zone

• Greedy routing ensures paths of polylogarithmic size
Management of long links
How many neighbours?

- **Close neighbors**: Voronoï neighbours (Mean $\leq 6$)

- **Shortcuts**
  - Simulations have shown that around 6 shortcuts is a good tradeoff between maintenance cost and performance

- **Back long link neighbours**
  - Dependent on the distribution of objects,
  - Balanced even with sparse distributions due to long link properties (random versus harmonic)

- **Overall neighbour set size is $O(1)$**
  - Independent of the number of objects
  - Independent of objects distribution
Experimental settings

• 300,000 objects (no object leaving)
• 2 object distributions in \([0:1] \times [0:1]\)
  – Uniform
  – Sparse: 5 equally popular regions. Popularity of objects around a region follows a power law with \(\alpha = 5\)
Simulation: object degree
number of Voronoï neighbours

Object out-degree does not depend of the objects distribution in space
Polylogarithmic routes (1)

Route cost evolution has a logarithmic shape. This does not depend on objects distribution.
Using several long links improves routing performance

- Linear improvement: Using $k$ shortcuts provides a routing that is almost $k$ times more efficient
  - At each step, the probability of using a long link that divide the path by $\log(N)$ is $k/\log(N)$
  - A reasonable amount of long links is $\sim 6$ for a 300,000 objects overlay
Voronoï cell computations are an overkill

RayNet: gossip-based approximation of complex structures
Voronoï diagrams, RayNet rationale

• VoroNet
  – Complex structure to compute, to maintain in face of churn, potential unlimited number of neighbours

• What really matters?
  • Neighbours: to ensure correct routing

• Using an approximation of the structure is enough to compute such neighborhoods

• Gossip-based protocols for Voronoï neighbours and shortcuts
Gossip-based construction of RayNet

- Local links: Coverage and closeness
  - Gossip-based construction of approximate Voronoï links
  - Close objects (in the semantic space) in all directions
- Shortcuts: Kleinberg peer sampling

Challenge:
Evolution of local views towards a global routing structure
Coverage and closeness

An object o’s view == Voronoï neighbours

Idea:

- Exchange views & converge towards an approximation of Voronoï neighbours
- No need to compute the Voronoï cells: use the volume as an indication of convergence (the smaller, the better)
Monte Carlo cell size estimation

- **Idea**: sample the boundaries of the zone using “rays”
- **Gossip-based protocol**: evaluate the view as a whole (configuration)
View update operation: naïve approach

- View size is $c=3d+1$ peers
- Exchange entire views: $o.view + o_{\text{partner}}.view$

- For each set $S$ of objects of size $c$, in $o.view + o_{\text{partner}}.view$
  - Estimate the volume of $o$’s cell in the diagram of $S$
  - Keep the set with minimal volume as the new view

- Effective, but there are $O(c!)$ configurations to examine...
View update operation: efficient approach

• Determine the potential contribution of each object to the coverage and closeness (i.e., to the volume of o’s cell)

• For each object o’ in o.view + o_{partner}.view
  – Compute the volume of o’s cell in o.view + o_{partner}.view without o’

  *Ignoring this object results in a bigger zone: High contribution*

  *Ignoring this object does not impact the size of the zone: No contribution*

• Keep the c objects with the greatest contribution
Efficient routing

- Routing in the approximate Voronoi diagram requires $O(N)$ hops
- Small-Worlds models:
  - Small paths + navigability
- Using Kleinberg-biased peer sampling
- $O(\log^d N)$ routing with 1 shortcut
Simulations

• Settings
  – 1.000 to 7.000 objects
  – Emergence from a chaotic state
    • No RayNet links
    • Random graph for the Kleinberg-biased peer sampling service

• Metrics
  – Self-organization speed
    • Cycles needed before full routing success
  – Routing efficiency
    • Mean hops
Self-organization speed

Less than 35 cycles of exchanges are needed for reaching a structure where all routes succeed onto the correct object.
Routing efficiency is achieved by the biased peer sampling layer.
RayNet wrap-up

• RayNet, overlay for exhaustive and expressive queries
  – Self-organizing
  – Routing efficiency

• Approximation of a complex & ‘ideal’ structure while still benefiting from its capacities
  – Expressiveness of the query model preserved
  – Efficient up to 10 dimensions
Conclusions

• Topic-based pub-sub or exact queries

• Content-based pub-sub or range queries
  – inherent structure mismatch with structured P2P overlays
    • Fix the issues as they arise: explicit load-balancing mechanism
    • Adapt the structure to match the one of the pub/sub system
References

- Theoni Pitoura, Nikos Ntarmos, Peter Triantafillou, Replication, Load Balancing and Efficient Range Query Processing in DHTs, 10th International Conference on Extending Database Technology (EDBT06), March 2006.
- Ashwin Bharambe, Mukesh Agrawal, Srinivasan Seshan, Mercury: Supporting Scalable Multi-Attribute Range Queries, SIGCOMM 2004