Approximate Algorithms for Distributed Systems

Marie Hoffmann

Institut für Informatik
Freie Universität Berlin

March 8, 2013

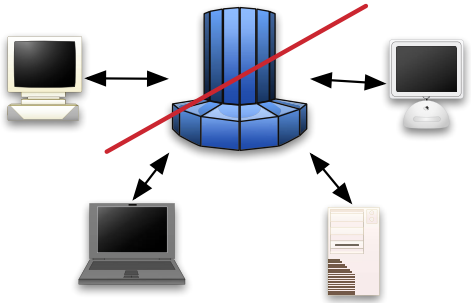
Content

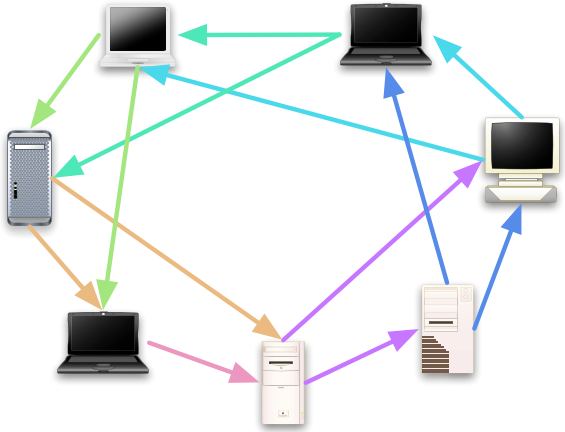
Motivation

Peer-to-Peer networks (P2P) – a special class of distributed systems

- no central infrastructure managing network, routing, resource allocation
- nodes are equals, act as clients and servers with additional privileges
- typically one routing layer on top of the physical one
- nodes might enter or leave at any time

- a lot of traffic for maintenance

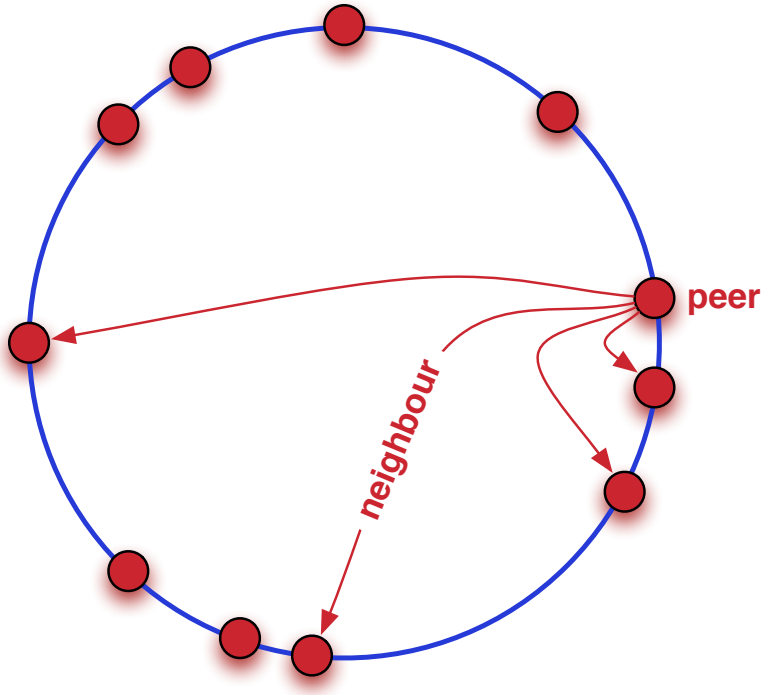




Motivation

A protocol for P2P systems: Scalaris

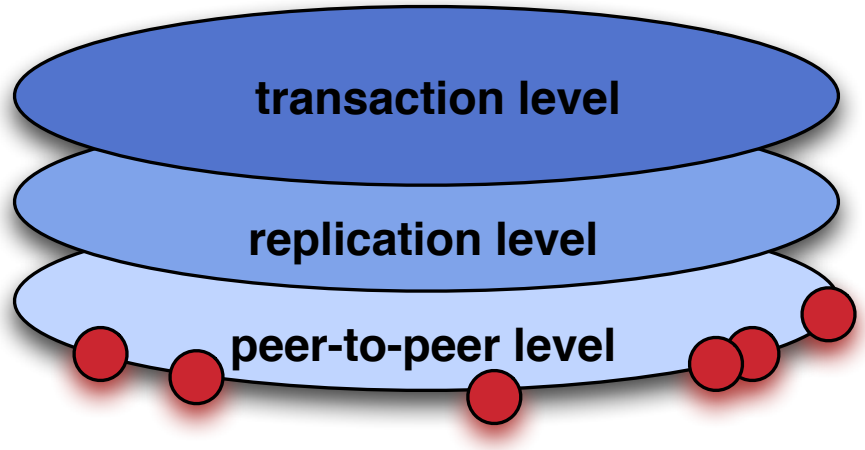
- Scalaris: a scalable, transactional, distributed key-value store
- Project initiated by members of Zuse Institute (ZIB)
- for building web services (e.g. distributed data storage, database, computing)
- Participating nodes are arranged in a ring-like overlay network



Motivation

A protocol for P2P systems: Scalaris

- **scalable** efficient when applied to large situations
- **distributed** storage or computation distributed over the network
- **replicas** e.g. file copied k times, copies (replicas) stored distributed over the network
- **transactional** information processing divided into indivisible operations
- **key-value store** data access via key



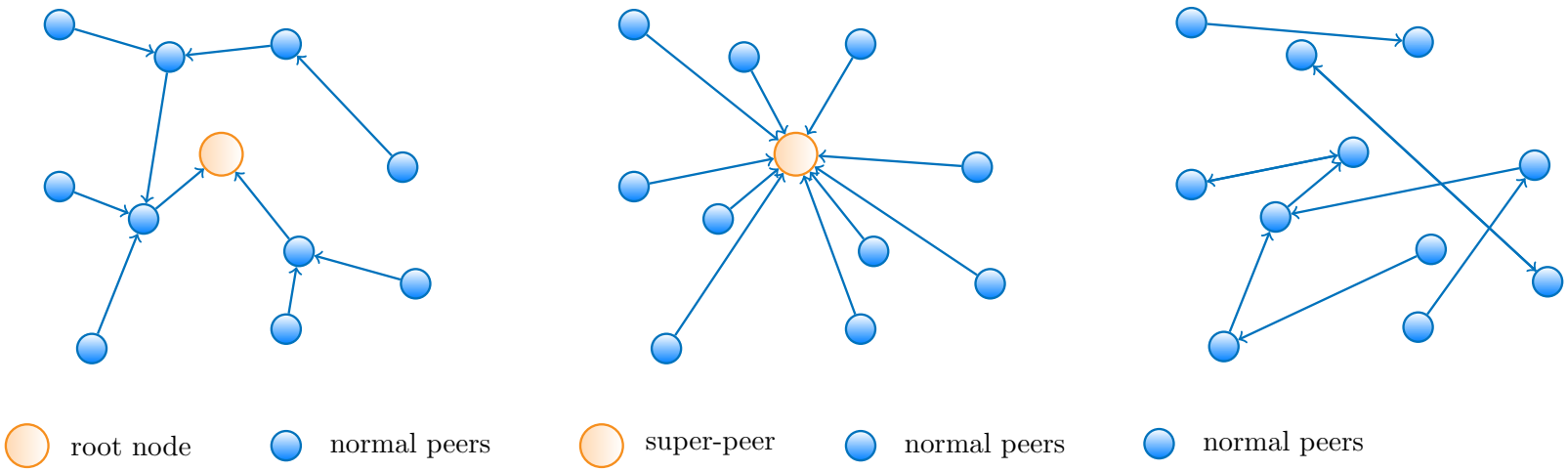
Motivation
Main Questions

We focus on three questions that appear in P2P systems:

1. **Protocol:** How to identify ideal storage locations in a distributed key-value store?
2. **Protocol:** How to reduce the traffic for maintenance messages?
3. **Statistics:** How to compute summaries over distributed data streams?

Communication Patterns
in P2P Systems

Solving a problem contiguously



(c) Routing tree. (d) Super-Peer. (e) Gossiping.

General Thoughts

Properties of P2P systems and possible circumventions for **algorithms** and **protocols** running in P2P systems:

- Unavailability of peers
  ⇒ do not communicate with a fixed set of neighbors, but introduce randomness
  ⇒ gossip protocol – randomized assignment of communicating parties
- Stored data is changing or unavailable
  ⇒ give up demand of exactness
  ⇒ opens a variety of new (approximate) approaches for the same problem

Gossiping

Communication between peers via the gossip protocol

- inspired by gossiping in social networks
- *randomized peer sampling* that runs periodically, either
  - by a node itself from its list of known neighbors
  - or a node-independent routine connecting two nodes
- as soon as connection has been established – nodes exchange their local data

Gossiping

Communication between peers via the gossip protocol

- inspired by gossiping in social networks
- *randomized peer sampling* that runs periodically, either
  - by a node itself from its list of known neighbors
  - or a node-independent routine connecting two nodes
- as soon as connection has been established – nodes exchange their local data

Advantages:

1. Epidemic-like spread of information
2. Simplicity: no synchronization, recovery, or storage of neighborhoods
3. Robustness for unsteady networks: toleration of lost messages, since local data is communicated to many nodes
4. Scalability: no storage of neighborhood sets that scale with the net size, assignment to any node from the whole network

Gossiping

3 types of information exchange: pull (receive), push (send) or push-pull (send and receive)

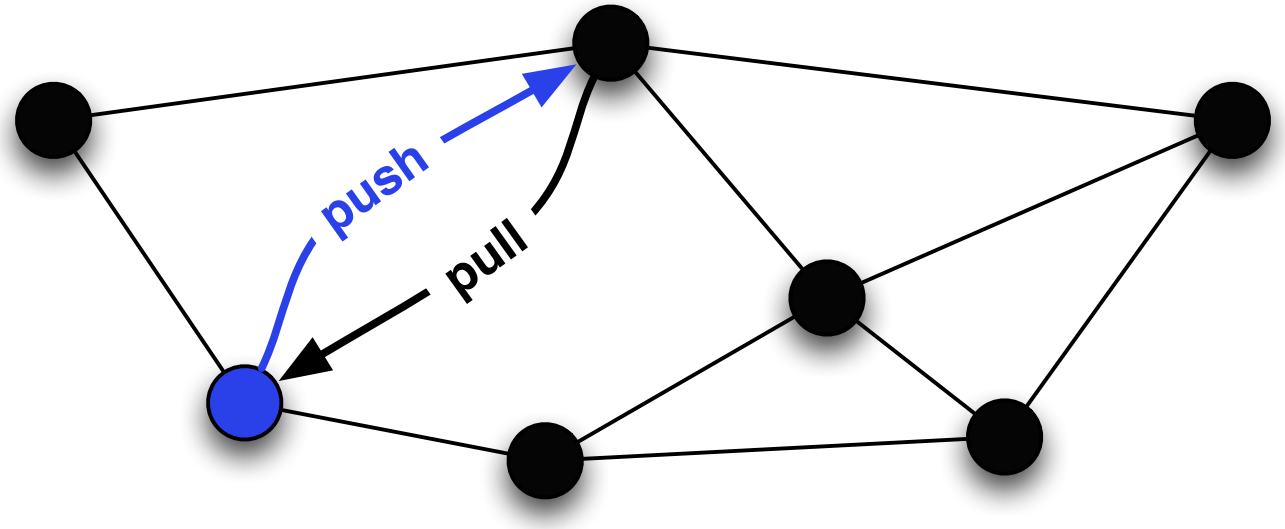
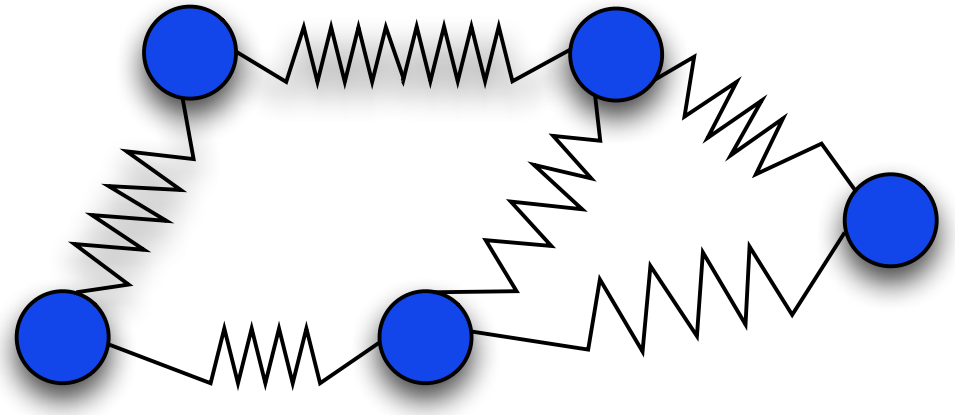


Figure: Peer (blue) to whom another peer (black) is assigned.

Network Coordinates

How to assign 2D network coordinates to peers?
⇒ Frank Dabek et al. 2004: *Vivaldi: A Decentralized Network Coordinate System*

- assigns synthetic coordinates to peers s.t. their distances correspond to the average round-trip times between them
- works for pure P2P networks
- might be piggy-backed or use the gossip protocol
- Vivaldi computes the solution of a spring-relaxation problem

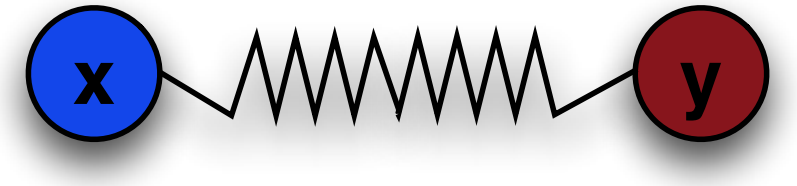


Network Coordinates

What each node does:

1. initially, assigns itself random 2D-coordinates, e.g. $x \in \mathbb{R}^2$ and an error $e = 1.f$
2. on input $y \in \mathbb{R}^2$, round-trip time rrt_{xy}, e_y, a node relaxes the difference between rrt_{xy} and $\|x - y\|$ by moving its own coordinates towards or away from y
3. repeat for several gossiping rounds

$$\big(rtt - \|x - y\|\big) \times u(x - y)$$



Network Coordinates

proc vivaldi($\mathbf{float}\ rtt, \mathbf{float}[\,]\ y, \mathbf{float}\ e_y$) $\equiv$

comment: weight balances local and remote error

$w := e_x/(e_x + e_y)$

comment: relative error of incoming sample

$e_s := \big|\|x - y\| - rtt\big|/rtt$

comment: update wma of local error

$e_x := e_s \times c_e \times w + e_x \times (1 - c_e \times w)$

comment: update local coordinates

$\delta := c_c \times w$

$x := x + \delta \times \big(rtt - \|x - y\|\big) \times u(x - y)$

end

Part 1: How to identify ideal storage locations in a distributed key-value store?

Task

Identification of Storage Locations

1. identification of data centers, given that each *node knows its spatial coordinates*
   - can be seen as clusters of nodes in a P2P network

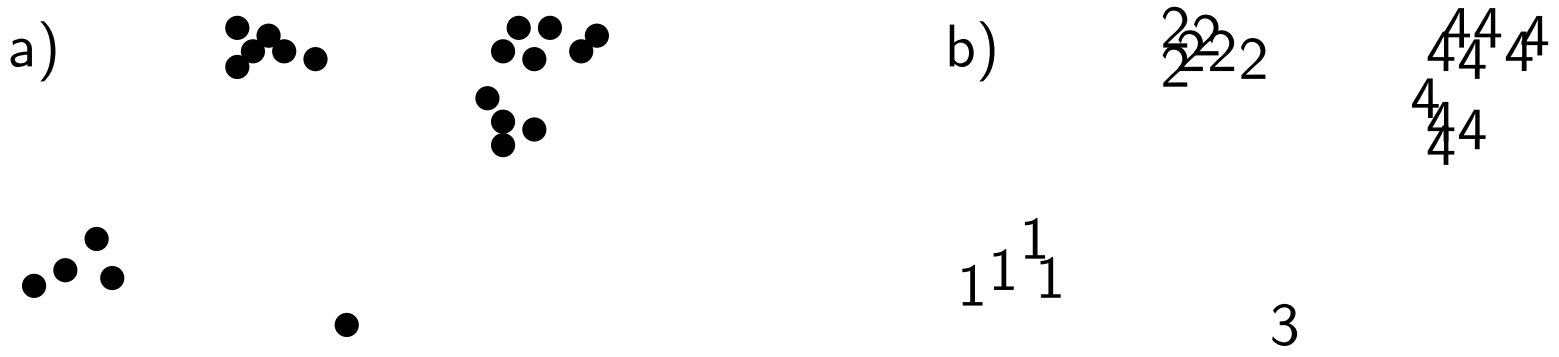
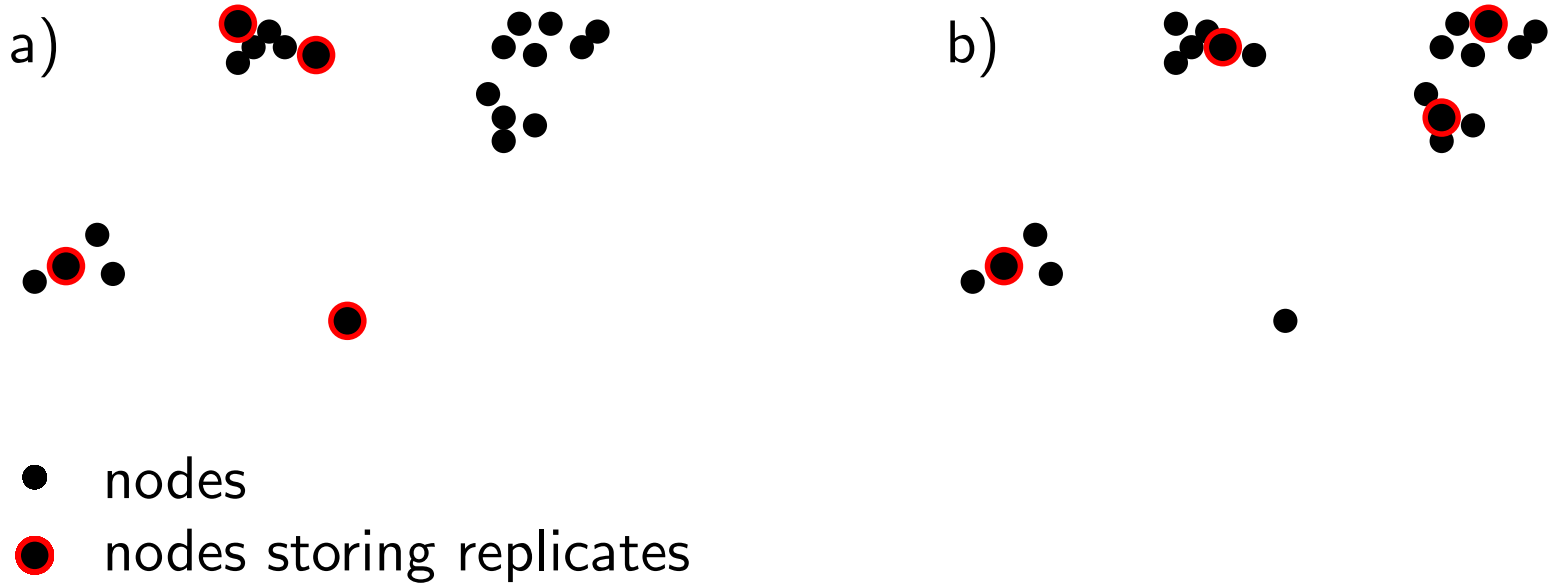


Figure: nodes in a network and their cluster assignments

Task

Identification of Storage Locations

1. identification of data centers, given that each *node knows its spatial coordinates*
   - can be seen as clusters of nodes in a P2P network
2. storage of k replicas, ideally
   - in k different clusters $\Leftarrow$ correlation of node failures
   - with maximum distances between each other
   - not on singletons, but in dense regions representing $1/k$th



Clustering

Data analysis tool

Given a data set $X \in \mathbb{R}^{n \times d}$, goal:

- group items according to their reported features
- items with high similarities should end up in same class
- data (not model) driven
- control quality by giving a distance dimension or prescribe final number of clusters

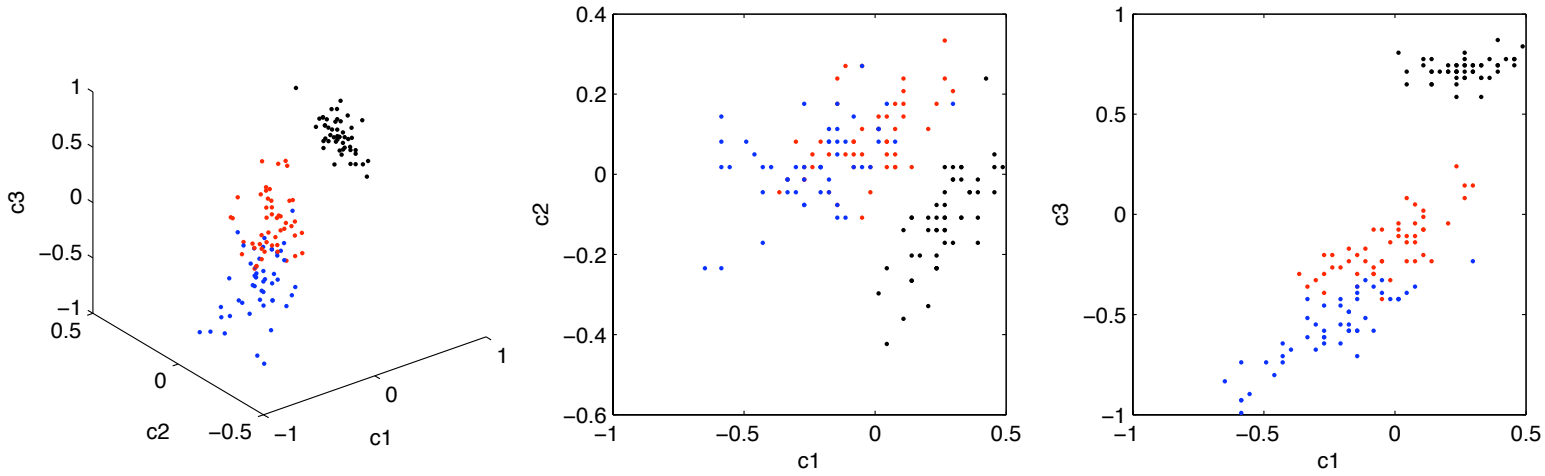
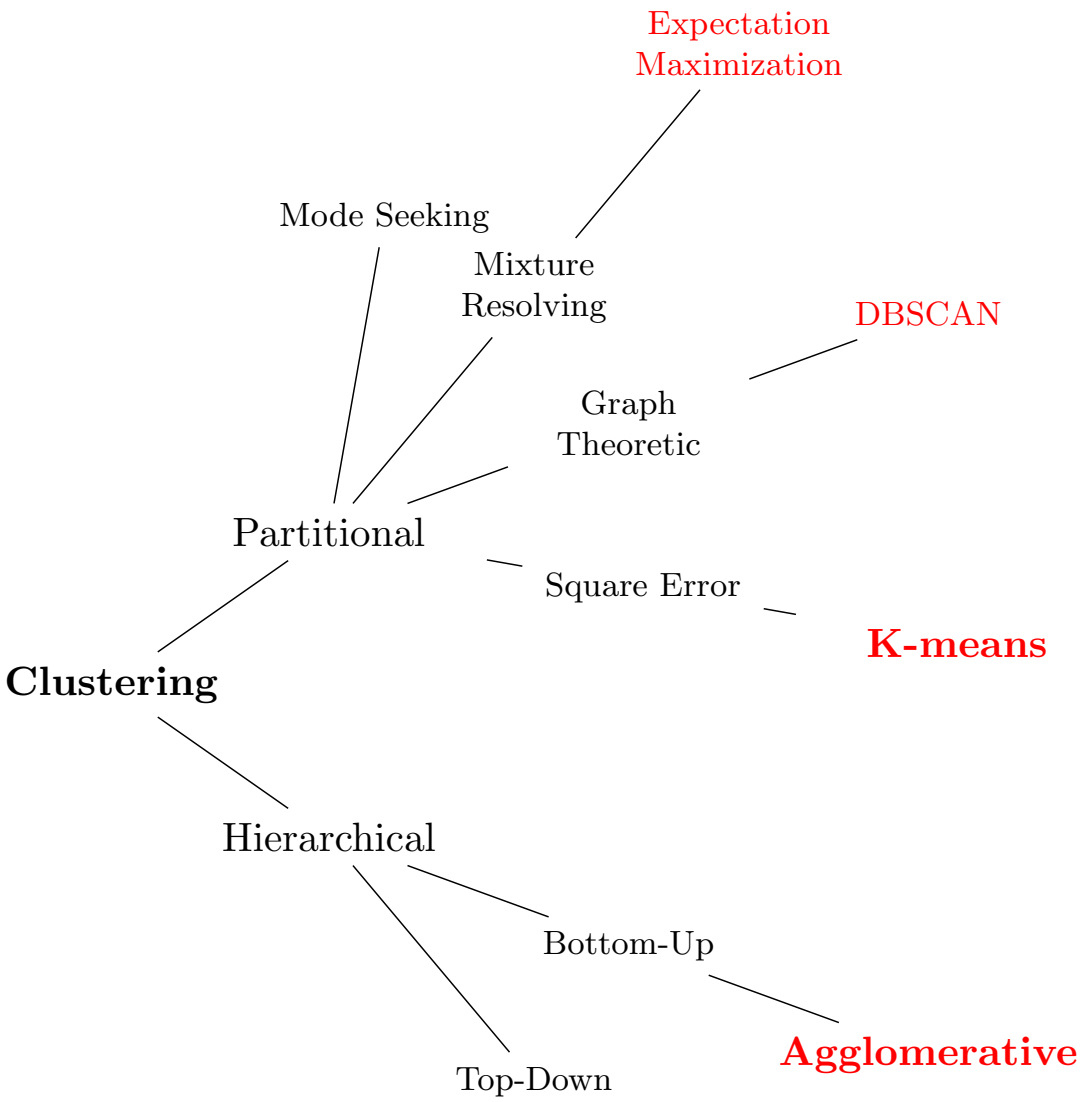


Figure: Matlab's Iris data set

Clustering

Taxonomy of clustering approaches



Clustering

K-means – Global

proc KMeans($\mathbf{float}[\,][\,]\ X, \mathbf{int}\ k, \mathbf{float}\ \gamma$) :

$n := |X|$

$c_l \xleftarrow{\$} X \quad \forall l \in [k]$

$label(i) := \arg\min_{l \in [k]} \|X(i) - c_l\| \quad \forall i \in [n]$

$c^{old} := c$

comment: M-step, re-estimate centroids

$c_l := \frac{1}{|\{i | label(i) == l\}|} \sum_{label(i)==l} X_i \quad \forall l \in [k]$

while $\max_j \{\|c_j^{old} - c_j\|\} \geq \gamma$

$c^{old} := c$

comment: E-step, compute expected label

$label(i) := \arg\min_{l \in [k]} \|X_i - c_l\| \quad \forall i \in [n]$

comment: M-step, re-estimate centroids

$c_l := \frac{1}{|\{i : label(i) == l\}|} \sum_{label(i)==l} X_i \quad \forall l \in [k]$

end

end

Method: iteratively **improve centroids' positions** s.t. the squared error is minimized

1. choose k centroids randomly
2. E-step: assign each data point to its closest centroid
3. M-step: re-estimate centroids
4. iterate E-step and M-step as long as centroids change significantly

Clustering
K-means – Global

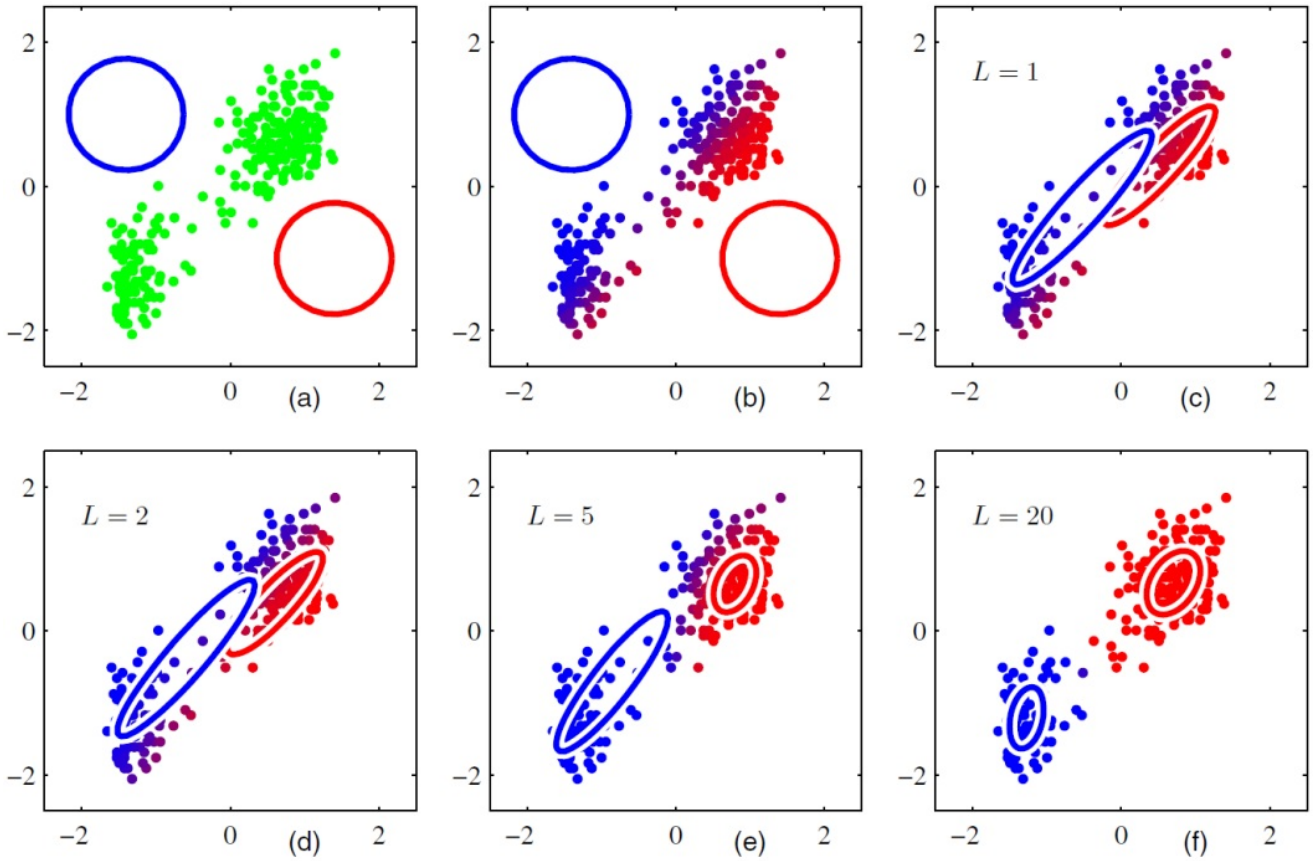
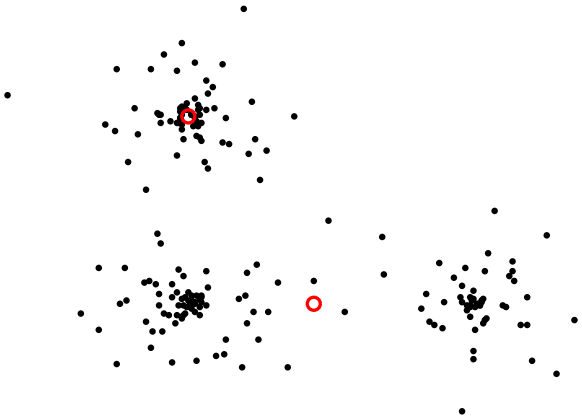


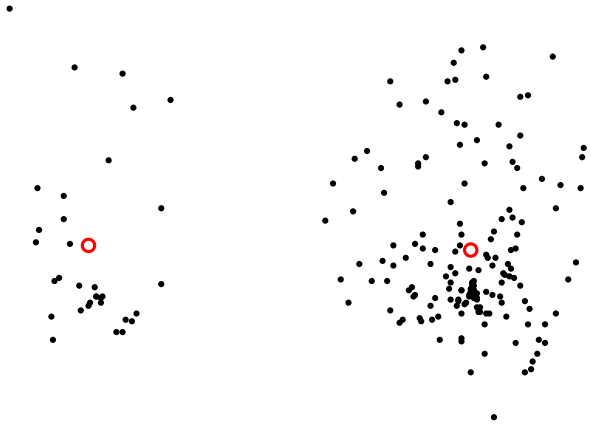
Figure: from Bishop: *Pattern Recognition and Machine Learning*

Clustering

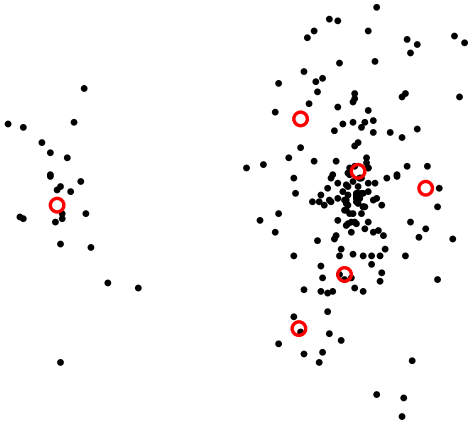
K-means – choice of k_{algo}



(a) $k_{data} > k_{algo}$



(b) $k_{data} = k_{algo}$



(c) $k_{data} < k_{algo}$

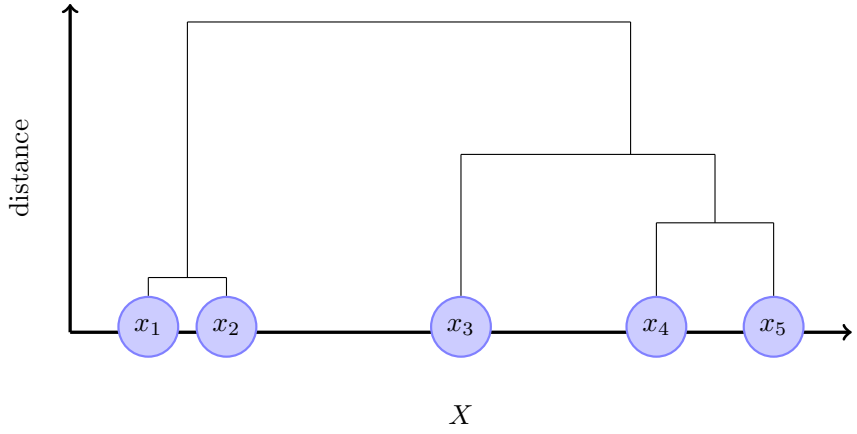
Clustering

Agglomerative Clustering – Global

proc θ-AggloClustering(**float**[ ][ ] $data$, **float** θ)
  $n := |data|$
  $C := data$
  $w_i := 1/|data| \quad \forall i \in [n]$
  $(i,j) := \arg\min_{\substack{i,j \\ i \neq j}} \|c_i - c_j\|$
  while $\|c_i - c_j\| < \theta$
    $(C, w) :=$ Merge(C, w, i, j)
    $(i,j) := \arg\min_{\substack{i,j \\ i \neq j}} \|c_i - c_j\|$
  end
  return (C, w)
end

proc Merge(**float**[ ][ ] C, **float**[ ] w, **int** i, **int** j)
  $C := C \mathbin{+\!\!+} [(c_i \cdot w_i + c_j \cdot w_j)/(w_i + w_j)]$
  $w := w \mathbin{+\!\!+} [w_i + w_j]$
  comment: delete original entries
  delete$(C, [i,j])$
  delete$(w, [i,j])$
  return (C, w)
end

1. start with the whole data, $C = data$ and relative sizes $w = \left(\frac{1}{|C|}\right)_{\{1:|C|\}}$
2. as long as there are two centroids being 'close enough', merge them into one centroid



Clustering

Agglomerative Clustering – Global

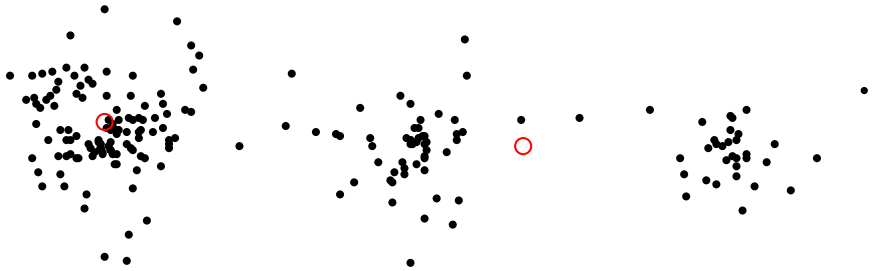
Naive: To receive k cluster for replica storage – stop when $|C| == k_{algo}$

```
proc KAggloClustering(float[ ][ ] data, int k)
    n := |data|
    C := data
    w_i := 1/|data|   ∀i ∈ [n]
    (i, j) := arg min_{i,j, i≠j} ||c_i − c_j||
    while |C| > k
        (C, w) := Merge(C, w, i, j)
        (i, j) := arg min_{i,j, i≠j} ||c_i − c_j||
    end
    return (C, w)
end
```

Clustering

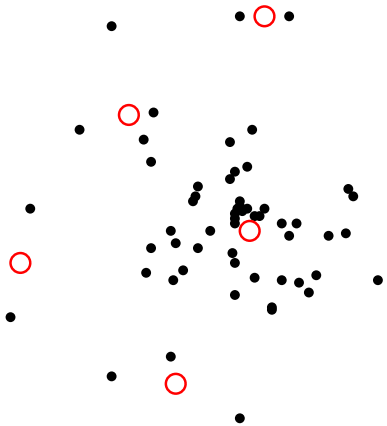
Agglomerative clustering – choice of k_{algo}

Result: for $k_{algo} \neq k_{data}$ we loose ability of cluster detection



(f) $k_{data} > k_{algo}$

(g) $k_{data} = k_{algo}$

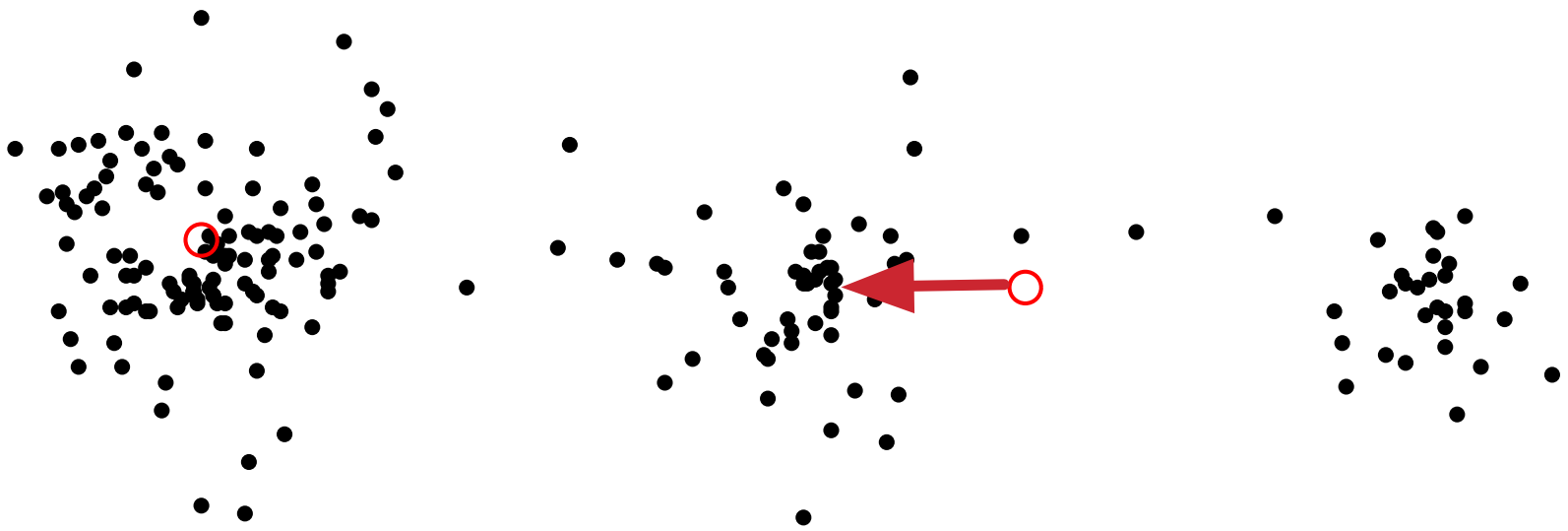


(h) $k_{data} < k_{algo}$

Challenges

We can use agglomerative clustering to detect the latent k_{data}

(i) if $k_{data} < k_{algo}$ place centroid into real cluster, e.g. the k_{algo} biggest ones (easy to solve)

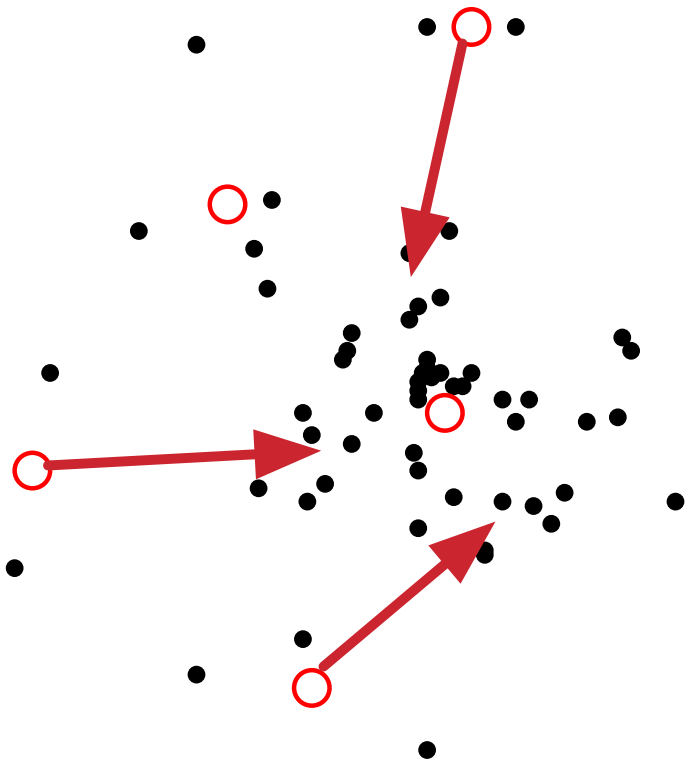


Challenges

We can use agglomerative clustering to detect the latent k_{data}

(i) if $k_{data} < k_{algo}$ place centroid into real cluster, e.g. the k_{algo} biggest ones (easy to solve)

(ii) if $k_{data} > k_{algo}$ place centroids s.t. they represent equal fractions of the cluster



Challenge #2

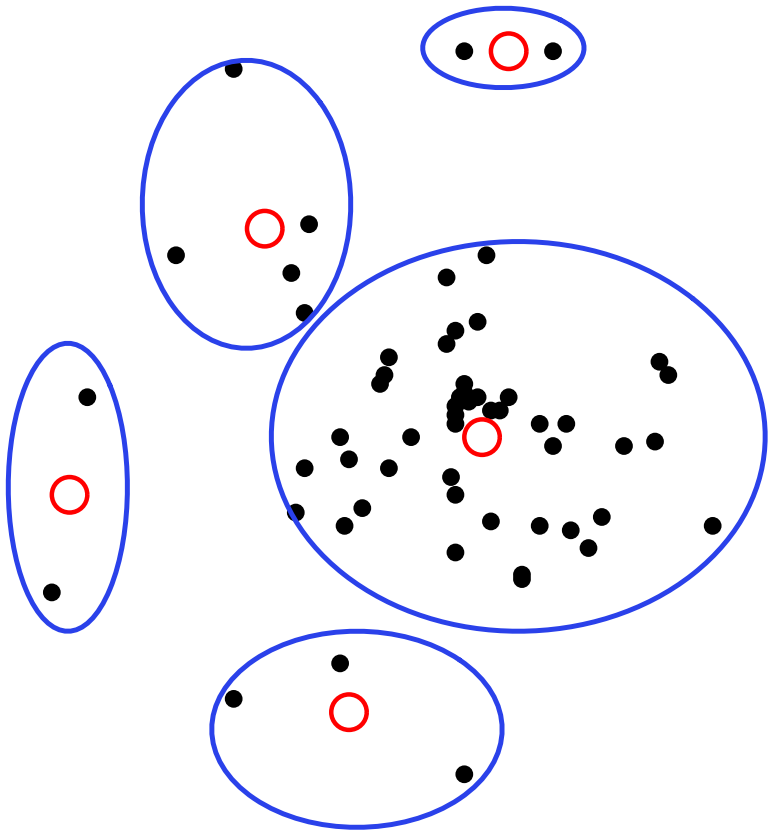
Closer look

Observation

- K-agglomerative clustering ends up with many centroids in the outskirts
- outlying centroids result from very few merges

Reason

- dense regions: high probability of finding close neighbors ⇒ many merges
- non-dense regions: low probability of finding neighbors nearby ⇒ no further agglomeration



Challenge #2

Closer look

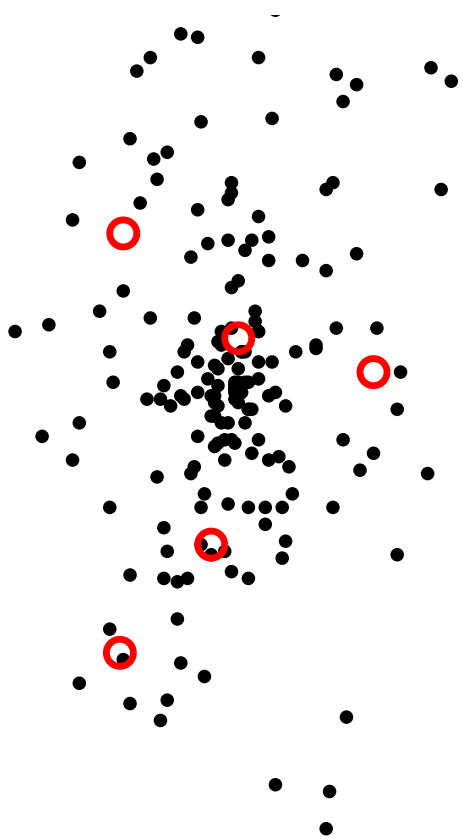
Observation

- K-agglomerative clustering ends up with many centroids in the outskirts
- outlying centroids result from very few merges

Reason

- dense regions: high probability of finding close neighbors ⇒ many merges
- non-dense regions: low probability of finding neighbors nearby ⇒ no further agglomeration

compared to K-means:



Challenge #2
Solution

Boost agglomeration below $1/k$ by incorporating the **relative sizes** into the selection

old selection criterion

$$(\hat{i}, \hat{j}) = \arg\min_{\substack{i,j \\ i<j}} \{\|c_i - c_j\|\}$$

new selection criterion

$$(\hat{i}, \hat{j}) = \arg\min_{\substack{i,j \\ i<j}} \left\{\|c_i - c_j\|/D + \delta \cdot \sigma(w_i + w_j - 1/k, z)\right\}$$

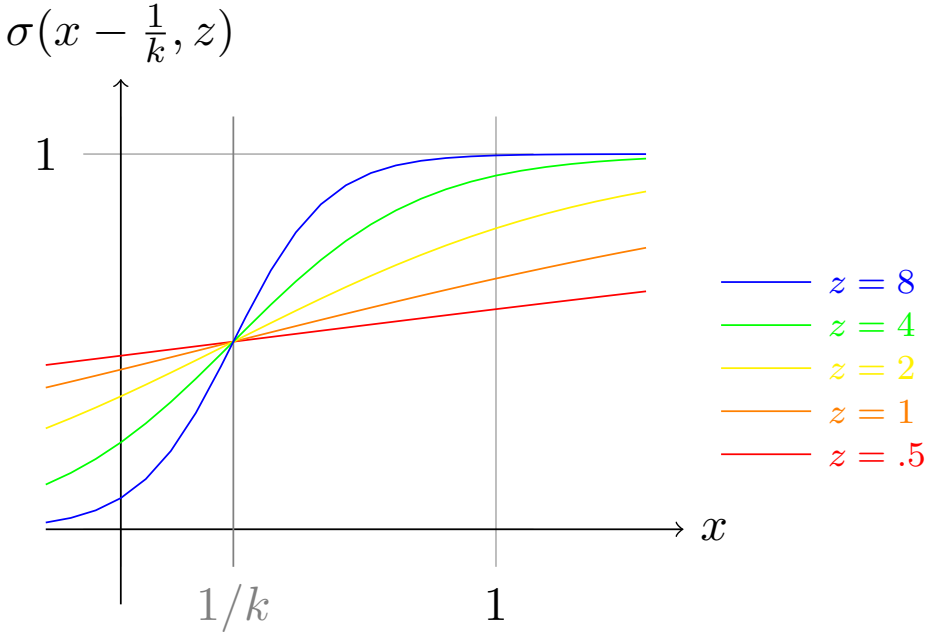
with $\sigma(x, z) = \frac{1}{1+e^{-z \cdot x}}$.

Challenge #2
Solution

Boost agglomeration below $1/k$ by incorporating the **relative sizes** into the selection

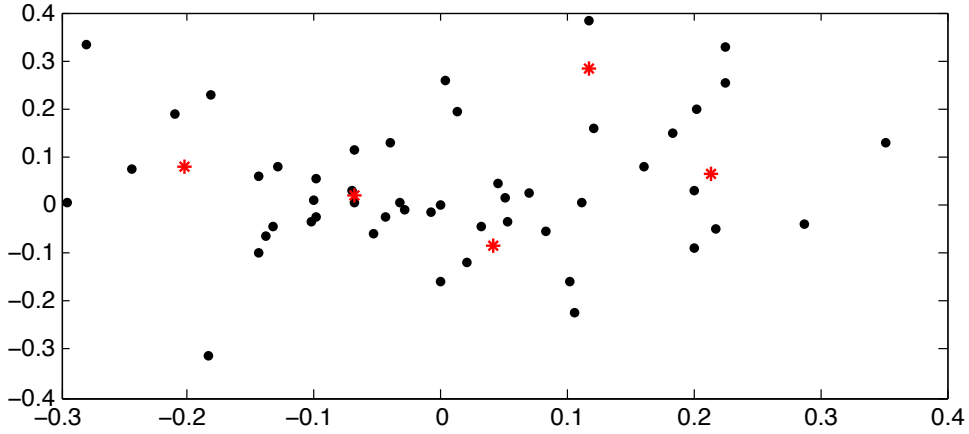
new selection criterion

$$(\hat{i}, \hat{j}) = \arg\min_{\substack{i,j \\ i<j}} \left\{ \|c_i - c_j\|/D + \delta \cdot \sigma(w_i + w_j - 1/k, z) \right\}$$

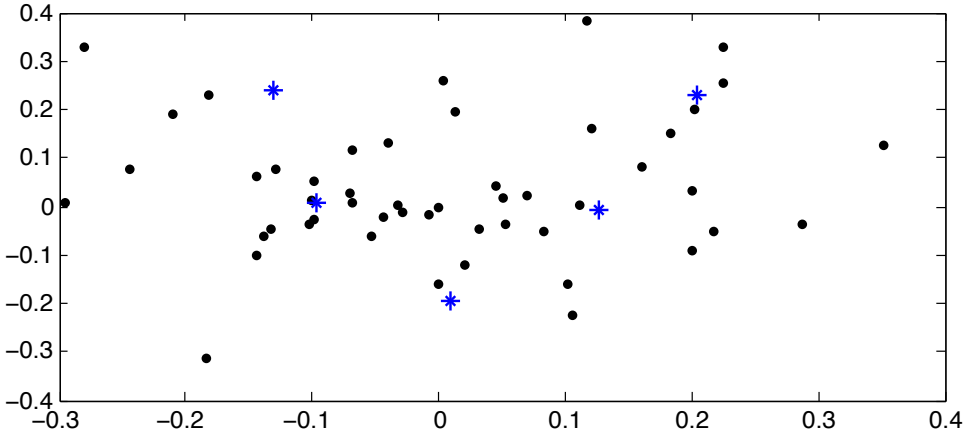


KAggloPlusClustering

Example $k_{data} < k_{algo}$



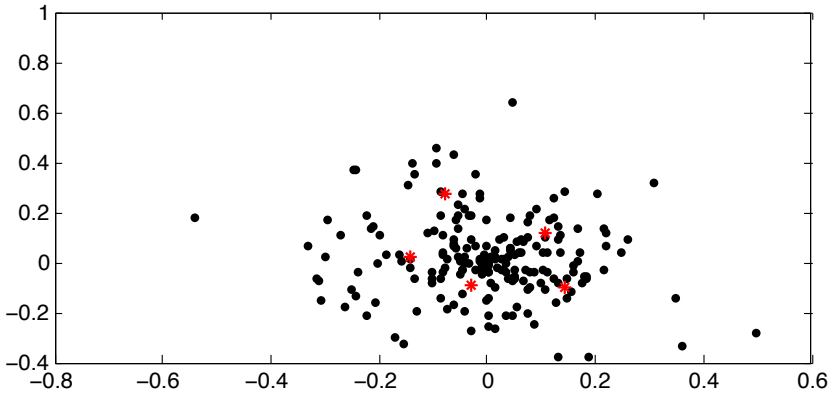
(i) k-AggloPlusClustering



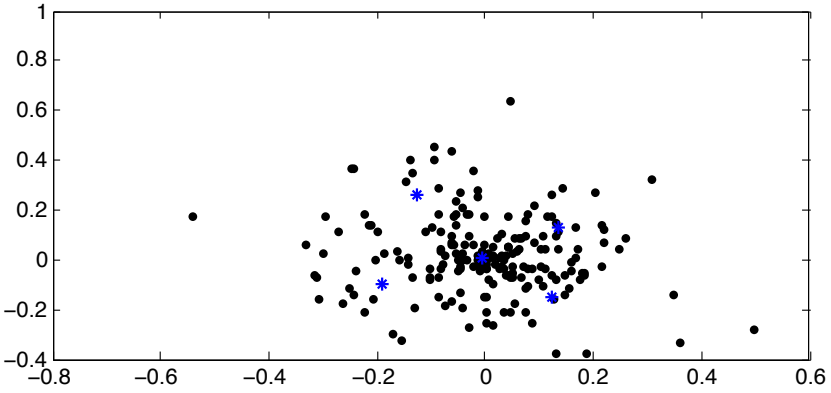
(j) k-Means

KAggloPlusClustering

Example $k_{data} < k_{algo}$



(k) k-AggloPlusClustering



(l) k-Means

Put together

Global algorithm

precondition: input C is the result of standard agglomerative clustering stopped very early ($|C| \geq k \cdot m$)

proc 2SKSub2(**float**[ ][ ] C, **float**[ ] w, **int** k, **float** θ) $\equiv$
  $(C_{latent}, w_{latent}, S)$:= ThetaAggloClustering(C, w, θ)
  comment: if $k_{data} \geq k$ return k largest centroids
  do if $|C_{latent}| \geq k$
    $C := C_{latent}[\text{findKLargest}(w_{latent}, k)]$
    $w := w_{latent}[\text{findKLargest}(w_{latent}, k)]$
    return (C, w, S)
  fi
  comment: else agglomerate with new merge criterion
  (C, w, S) := KAggloPlusClustering(C, w, k)
  return (C, w, S)
  end

k ≤ k_data

k > k_data

Error

Measuring the quality of clusters

correctness of centroid positions $c =$ real and $\widehat{c} =$ estimated centroids

$$err_{pos} = \frac{1}{kD} \sum_i \underbrace{\arg\min_j \{ \|\widehat{c_i} - c_j\| |\widehat{w_i} - w_j| \}}_{\text{closest centroids with respect to position and size}}$$

equally sized centroids

$$err_{eq} = \underbrace{\frac{1}{k} \sum_i (\widehat{w_i} - 1/k)^2}_{\text{deviation from equally sized clusters}}$$

well-separation of centroids

$$err_{sep} = \underbrace{\frac{2}{k(k-1)} \sum_{\substack{ij \\ i<j}} \left(1 - \|c_i - c_j\|/D\right)}_{\text{penalty for low distances between centroids}}$$

Global Clustering

Experimental Setup

- k_{data}, k_{algo} sampled from $[2; 8]$
- 2D data points sampled from k_{data} normal distributions
- relative cluster sizes sampled from $[.2; .7]$
- data set size at least 100
- errors averaged over at least 100 rounds

Global Clustering

Results

error	K-agglomerative (naive)	3S K-agglomerative			K-means
		$\delta = .75$ $z = .5$	$\delta = 1$ $z = 4$	$\delta = 2$ $z = 8$	
pos	1.0000	0.1359	0.3953	0.4790	0.8672
eq	1.0000	0.5949	0.2653	0.1887	0.3129
sep	0.4663	0.8220	0.9515	1.0000	0.8878

Table: position, equality and separation errors for 1S/3S K-agglomerative clustering with three parameter sets and K-means, rows are divided by their maxima

z sharpness, δ weight of sigmoid part

Distributed, Approximate Agglomerative Clustering *2SP2P*

Goal: each node receives a global view of clusters in the network

- **initially** each node knows only its local data

proc Initialize $\equiv$
$(C, w) := ([self.data], \frac{1}{|self.data|}[1.0]_{|self.data|})$
end

Distributed, Approximate Agglomerative Clustering *2SP2P*

Goal: each node receives a global view of clusters in the network

- **initially** each node knows only its local data
- **periodically** a node sends its locally estimated centroids using a gossip protocol

proc Timer $\equiv$
  $peer :=$ SelectRandomPeer()
  $\underline{\textbf{sendTo}}$ $peer$: Shuffle(C, w)
end

Distributed, Approximate Agglomerative Clustering *2SP2P*

Goal: each node receives a global view of clusters in the network

- **initially** each node knows only its local data
- **periodically** a node sends its locally estimated centroids using a gossip protocol
- **upon Shuffle message** (passive node) – update local centroids

$$
\begin{array}{l}
\text{proc Shuffle}(C_{rmt}, w_{rmt})\ \underline{\textbf{from}}\ p\ \equiv \\
\quad \underline{\textbf{sendTo}}\ p : \text{ShuffleResponse}(C, w) \\
\quad (C, w) := \text{Update}(C \mathbin{+\!\!+} C_{rmt}, w \mathbin{+\!\!+} w_{rmt}) \\
\text{end}
\end{array}
$$

Distributed, Approximate Agglomerative Clustering *2SP2P*

Goal: each node receives a global view of clusters in the network

- **initially** each node knows only its local data
- **periodically** a node sends its locally estimated centroids using a gossip protocol
- **upon Shuffle message** (passive node) – update local centroids
- **upon Shuffle message** (active node) – update local centroids, approximate part: cluster on two sets of estimated centroids!

$$
\begin{aligned}
&\text{proc ShuffleResp}(C_{rmt}, w_{rmt})\ \underline{\textbf{from}}\ p\ \equiv \\
&\quad (C, w) := \text{Update}(C \mathbin{+\!\!+} C_{rmt}, w \mathbin{+\!\!+} w_{rmt}) \\
&\text{end}
\end{aligned}
$$

Distributed, Approximate Agglomerative Clustering *2SP2P*

Goal: each node receives a global view of clusters in the network

- **initially** each node knows only its local data
- **periodically** a node sends its locally estimated centroids using a gossip protocol
- **upon Shuffle message** (passive node) – update local centroids
- **upon Shuffle message** (active node) – update local centroids, approximate part: cluster on two sets of estimated centroids!
- **Update** – mk-agglomerative clustering

proc Update $\equiv$
  $(C, w) :=$ KAggloClustering$(C, w, m \cdot k)$
  $(C, w) :=$ Normalize(C, w)
end

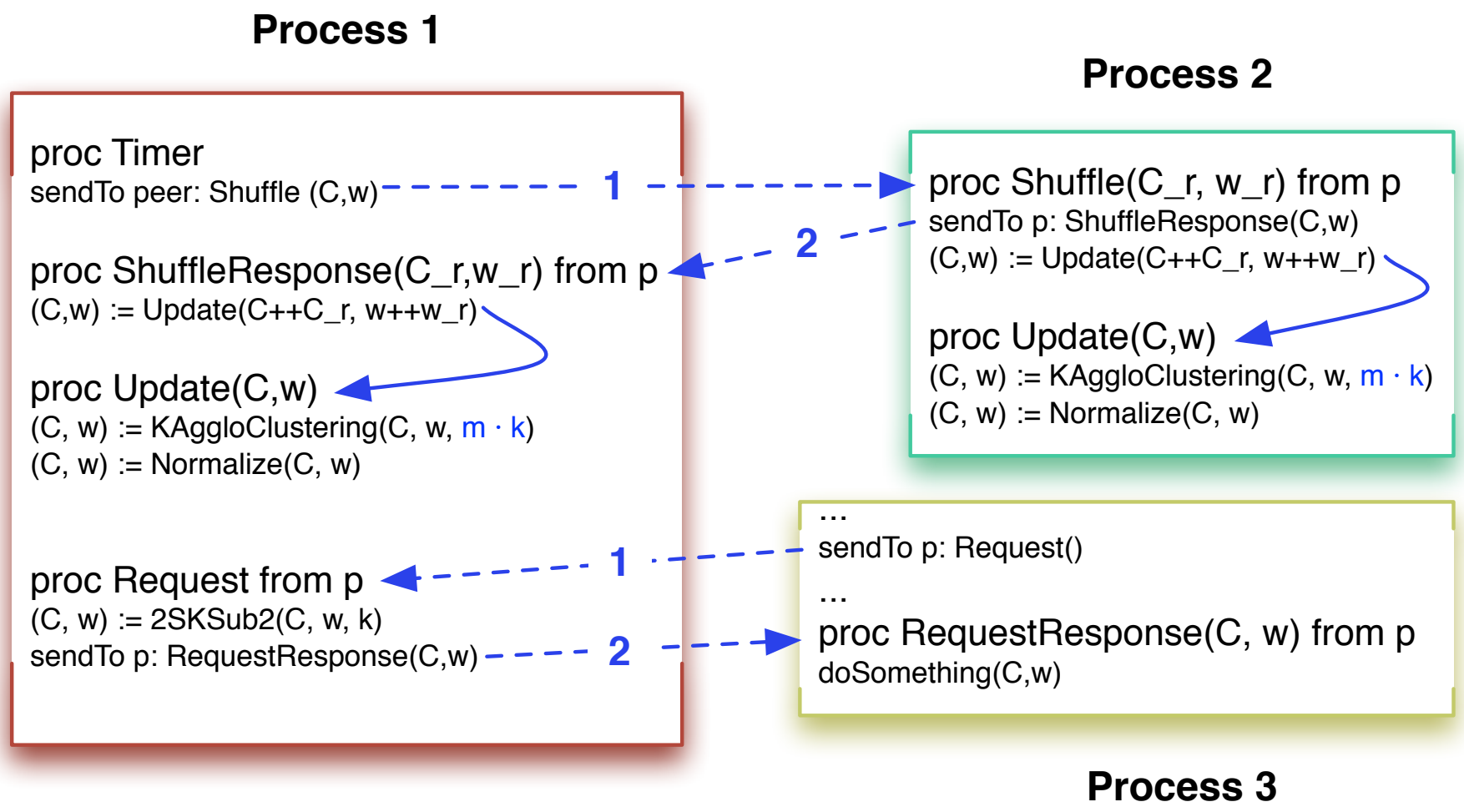
Distributed, Approximate Agglomerative Clustering *2SP2P*

Goal: each node receives a global view of clusters in the network

- **initially** each node knows only its local data
- **periodically** a node sends its locally estimated centroids using a gossip protocol
- **upon Shuffle message** (passive node) – update local centroids
- **upon Shuffle message** (active node) – update local centroids, approximate part: cluster on two sets of estimated centroids!
- **Update** – mk-agglomerative clustering
- **upon Request message** – reduce set of centroids

  proc Request $\underline{\textbf{from}}$ p $\equiv$
  $(C_{req}, w_{req}, S) :=$ 2SKSub2(C, w, k, θ)
  $\underline{\textbf{sendTo}}$ p : RequestResponse(C_{req}, w_{req})
  end

Distributed, Approximate Agglomerative Clustering *2SP2P*



Distributed, Approximate K-Means *LSP2P*

In 2009 Datta et al. presented an **approximate distributed K-means clustering algorithm**

- similar to 2SP2P K-agglomerative clustering: exchange local estimates
- two kinds of centroids: V_l (E-step) and C_l (result of M-step)
- partial synchronization: node enters new iteration together with its neighbors

Distributed, Approximate K-Means *LSP2P*

Algorithm

- **initially** all nodes start with the same set (C_1, w_1)

Distributed, Approximate K-Means *LSP2P*

Algorithm

- **initially** all nodes start with the same set (C_1, w_1)
- **periodically** a node requests (C_l^γ, w_l^γ) from its neighbors Γ

Distributed, Approximate K-Means *LSP2P*

Algorithm

- **initially** all nodes start with the same set (C_1, w_1)
- **periodically** a node requests (C_l^γ, w_l^γ) from its neighbors Γ
- **upon reply** of all neighbors, node N^i computes

$$v_{j,l+1}^i = \frac{\sum_{N_k \in Wait^i} c_{j,l}^k w_{j,l}^k}{\sum_{N_k \in Wait^i} w_{j,l}^k}$$

Distributed, Approximate K-Means *LSP2P*

Algorithm

- **initially** all nodes start with the same set (C_1, w_1)
- **periodically** a node requests (C_l^γ, w_l^γ) from its neighbors Γ
- **upon reply** of all neighbors, node N^i computes

$$v_{j,l+1}^i = \frac{\sum_{N_k \in Wait^i} c_{j,l}^k w_{j,l}^k}{\sum_{N_k \in Wait^i} w_{j,l}^k}$$

- enter a new K-means iteration $l = l + 1$
  - E-step: assignment of local data to its closest centroid $v_{j,l}$
  - M-step: reccompute $C_{j,l}$ with respect to local data set

Distributed, Approximate K-Means *LSP2P*

Algorithm

- **initially** all nodes start with the same set (C_1, w_1)
- **periodically** a node requests (C_l^γ, w_l^γ) from its neighbors Γ
- **upon reply** of all neighbors, node N^i computes

$$v_{j,l+1}^i = \frac{\sum_{N_k \in Wait^i} c_{j,l}^k w_{j,l}^k}{\sum_{N_k \in Wait^i} w_{j,l}^k}$$

- enter a new K-means iteration $l = l + 1$
  - E-step: assignment of local data to its closest centroid $v_{j,l}$
  - M-step: reccompute $C_{j,l}$ with respect to local data set

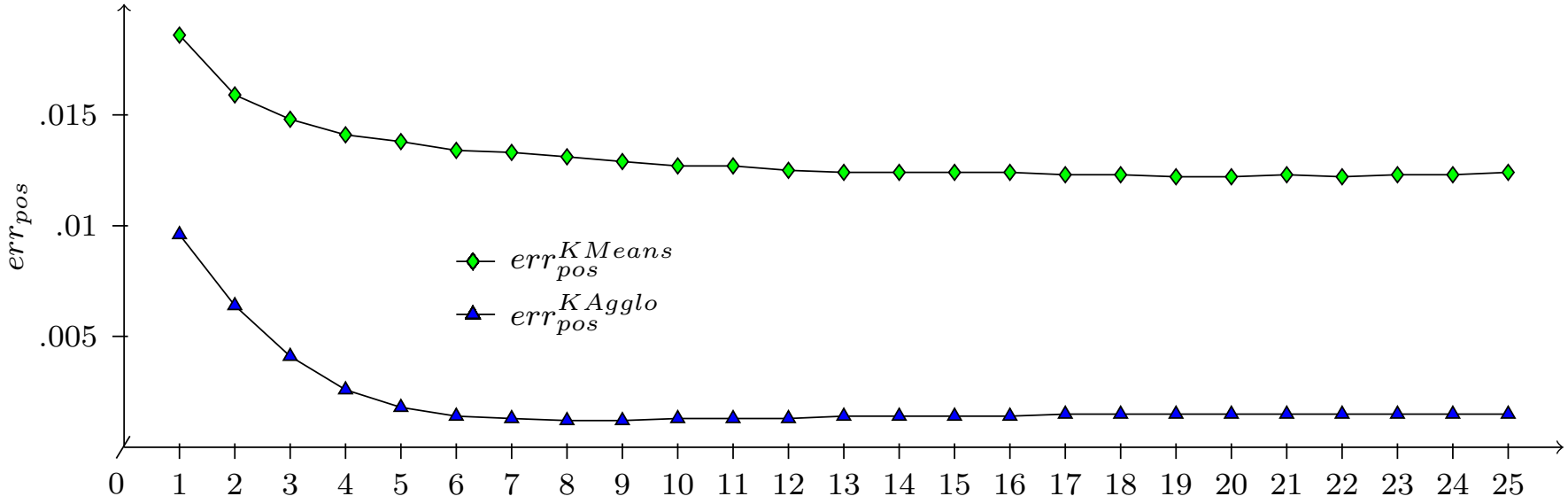
Disadvantage of synchronization:

- N^i can only process centroids from nodes being in the same iteration $\Rightarrow$ wait until all neighbors completed round l
- nodes might request older centroids $\Rightarrow$ store $C_{j,t} \forall t \in [0; l]$

Results

- position error per node between centroids from **global clustering** and **local centroids**
- convergence of err_{eq}, err_{sep} follows from convergence of err_{pos}

$$err_{pos}^{P2P}(t) = \frac{1}{R}\sum_{r=1}^{R}\frac{1}{N}\sum_{i=1}^{N}\frac{1}{kD}\sum_{l=1}^{k}\arg\min_{j}\{\|c_l^i(t)-c_j(t)\|\cdot|w_l^i(t)-w_j(t)|\}$$

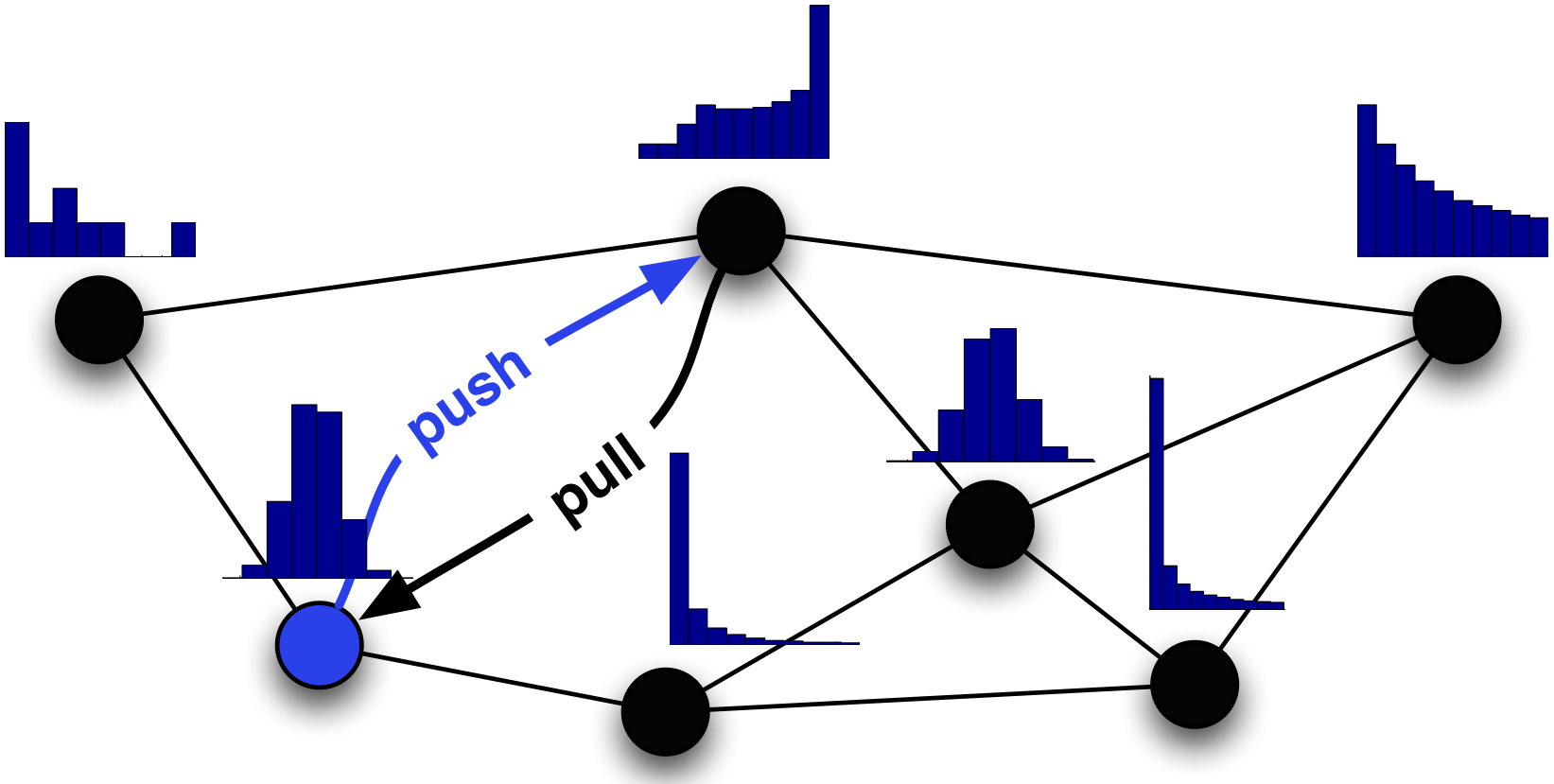


2SP2P K-Agglomerative Clustering vs LSP2P K-Means

Properties	**2SP2P K-agglomerative**	**LSP2P K-means**
synchronization of iterations required	no	yes
k must be fixed during computing phase	no	yes
order of centroids and counts must be kept fixed	no	yes
robustness for outliers	yes	no
bandwidth costs	$\mathcal{O}(mkIL)$	$\mathcal{O}(kIL)$
computational costs	$\mathcal{O}(Im^2k^2 + D^2)$	$\mathcal{O}(IkD)$
memory costs	during exchange phase: $\Omega(mk)$	history and poll table: $\mathcal{O}(I(k+L))$

$I = \#$ number of iterations, $L = \#$ of neighbors contacted per round, $k = \#$ final centroids, $m =$ scale parameter, $D =$ size of local data set

Part2: How to compute quantiles over distributed data streams?



Motivation

Given a set of distributed data streams, e.g.

- servers storing response times
- sensors storing temperature measurements hourly

Naive approach – computing quantiles on single sources (and merge them) does not work!

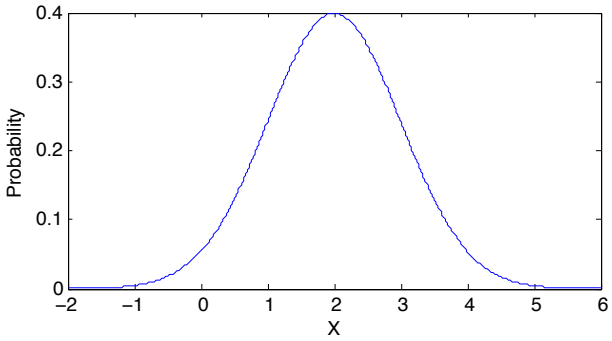
Better:

1. compute a robust summary of local data set – an equi-probable histogram
2. gossip and merge data summaries

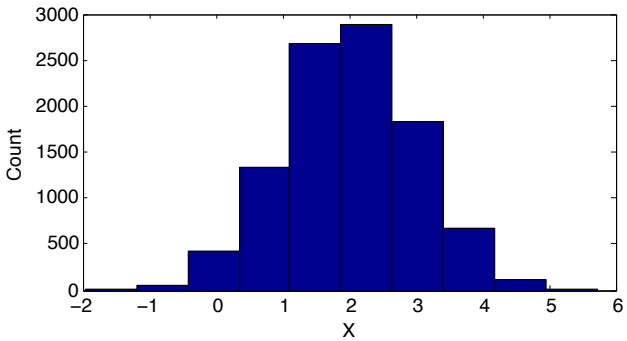
Question: Is randomized merging with no upper bounds on the number of merging operations still robust?

Histogram

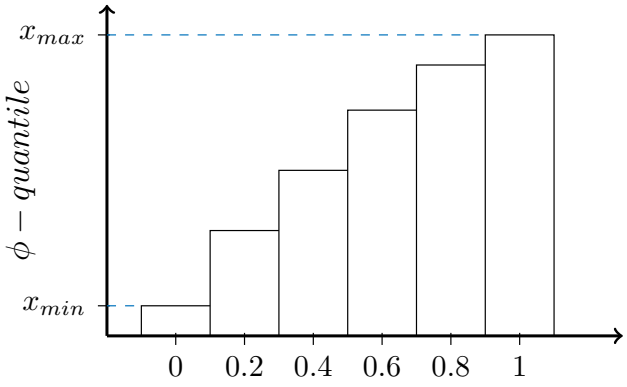
equi-width and equi-probable



(n) data distribution



(o) equidistant histogram



(p) equiprobable histogram

Quantile

Given $\phi \in [0;1]$ and a sorted data set X with $|X| = n$, the ϕ-quantile is the value $x \in X$ at position (int)ϕn.

E.g.

- 0-quantile is $\min(X)$
- 1-quantile is $\max(X)$
- 0.5-quantile is the median of X

1. ϕ-quantile is the inverse of the *cumulative distribution function* (cdf) $P(X \leq x)$
2. an equiprobable histogram can be composed by a series of ϕ-quantiles

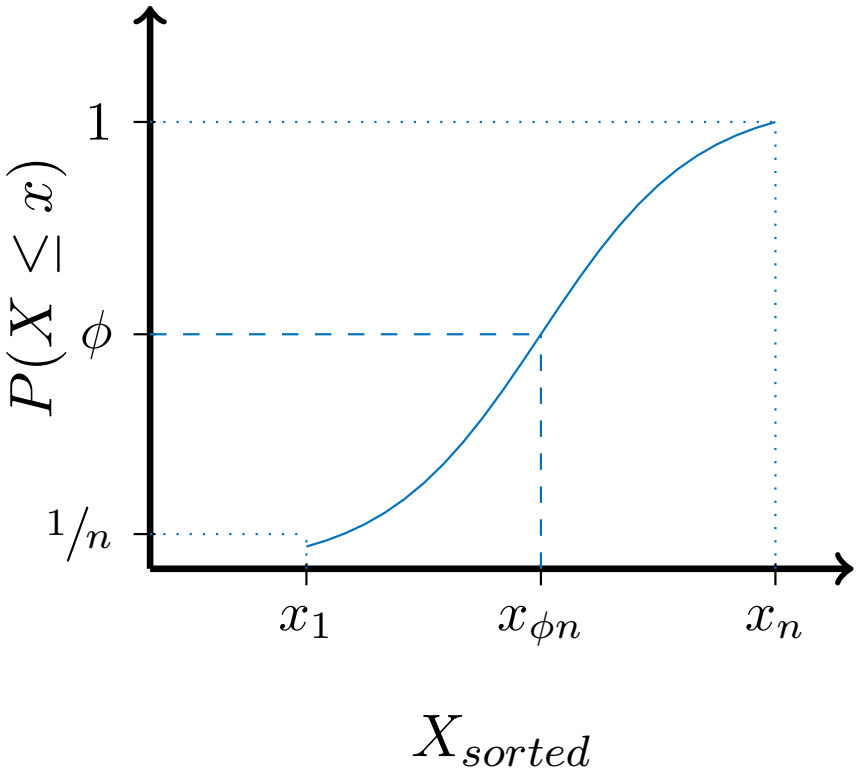


Figure: ϕ-quantile and cdf, $\phi \in [0;1]$

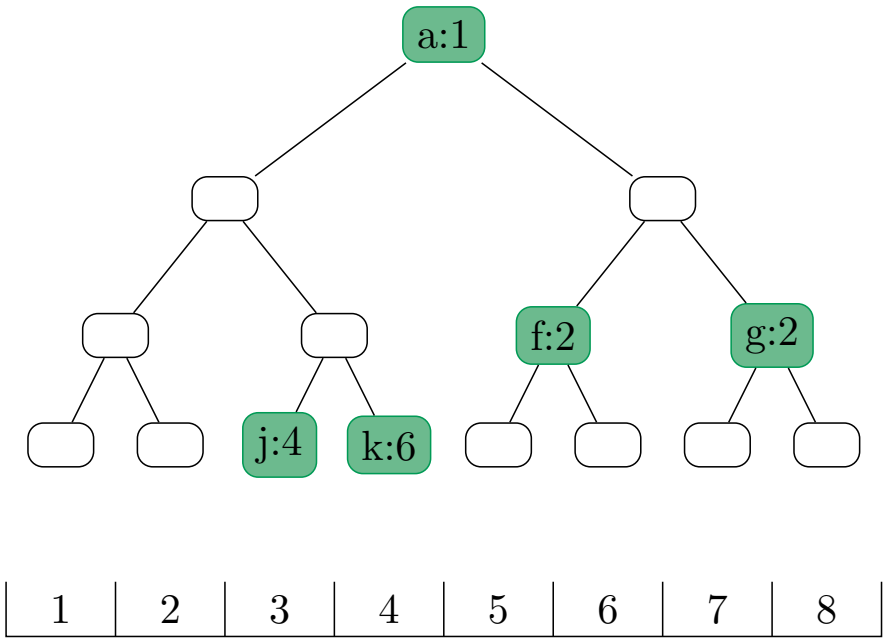
Q-Digest

- incomplete binary tree, whose structure corresponds to an **equiprobable histogram**, except that the bins are overlapping
- each node v fulfills the **q-digest property**:

$$v.count \leq \lfloor n/k \rfloor \quad (1)$$

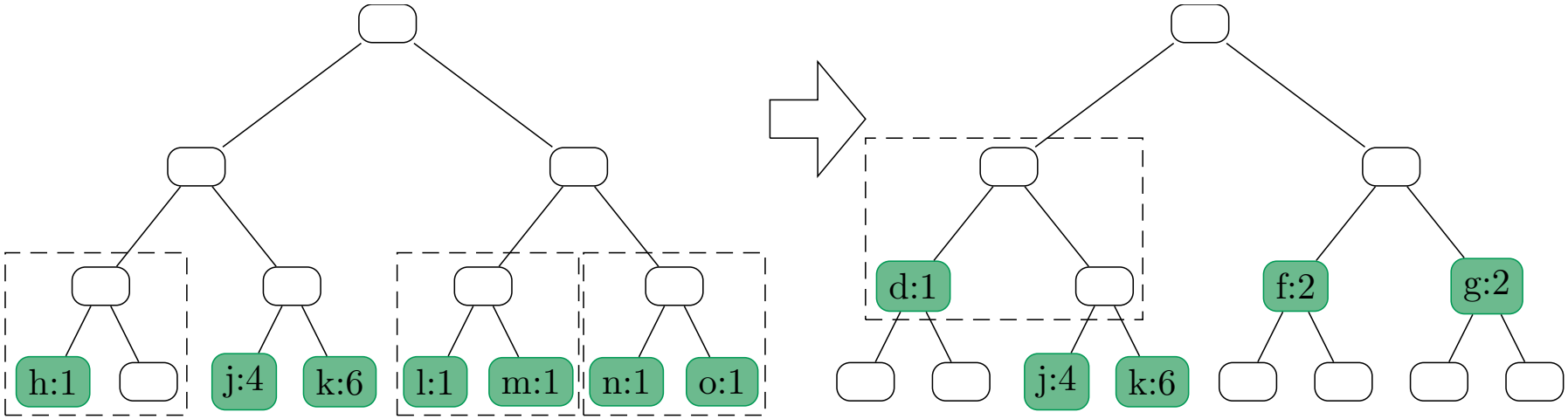
$$v.count + v_p.count + v_s.count > \lfloor n/k \rfloor \quad (2)$$

- data range $X \in [1; \sigma = 2^m]$
- decompression k



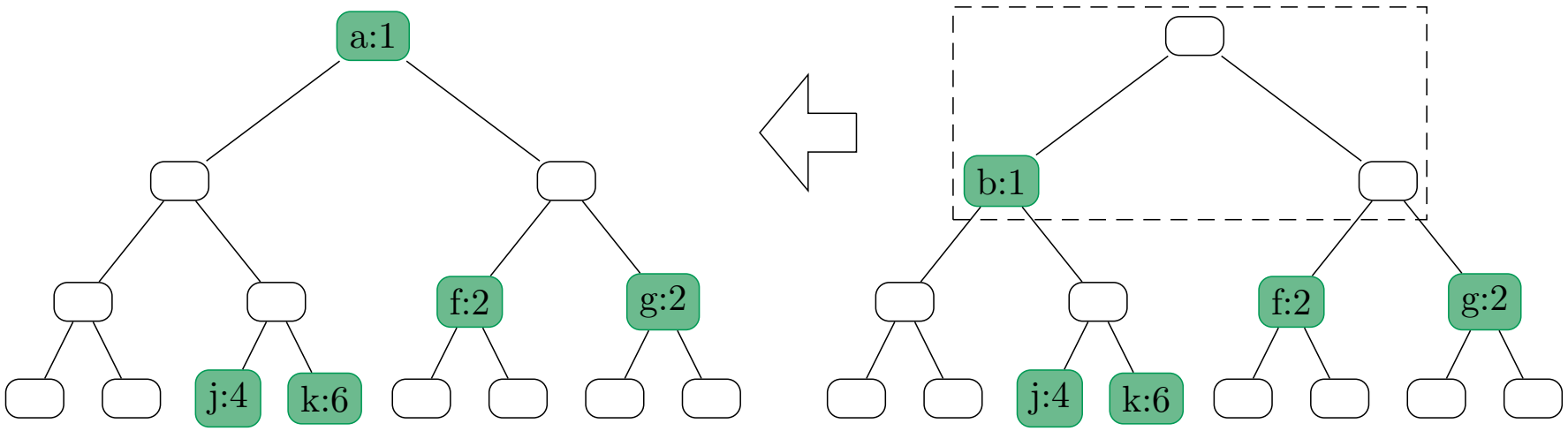
Q-Digest
Compression

- compression function ensures *q-digest property*
- example: $k = 5, n = 15, \sigma = 8,\ X = \{1, 3_4, 4_6, 5, 6, 7, 8\}$
- procedure: compress nodes bottom-up that violate $v.count \leq \lfloor n/k \rfloor$ or $v.count + v_p.count + v_s.count > \lfloor n/k \rfloor$ by accumulating child counts into parent node



Q-Digest
Compression

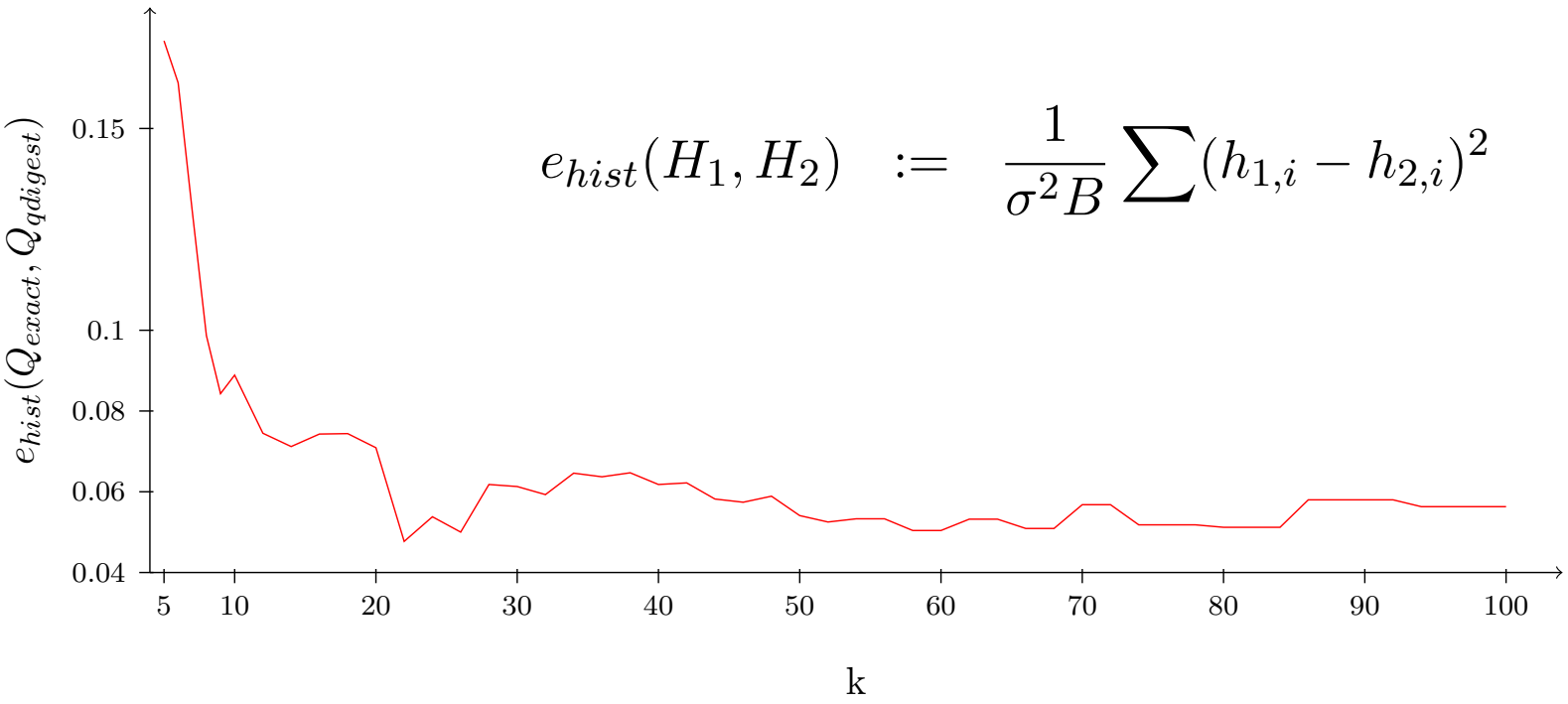
- compression function ensures *q-digest property*
- example: $k = 5, n = 15, \sigma = 8,\ X = \{1, 3_4, 4_6, 5, 6, 7, 8\}$
- procedure: compress nodes bottom-up that violate $v.count \leq \lfloor n/k \rfloor$ or $v.count + v_p.count + v_s.count > \lfloor n/k \rfloor$ by accumulating child counts into parent node



Compression

Choice of Decompression Factor k

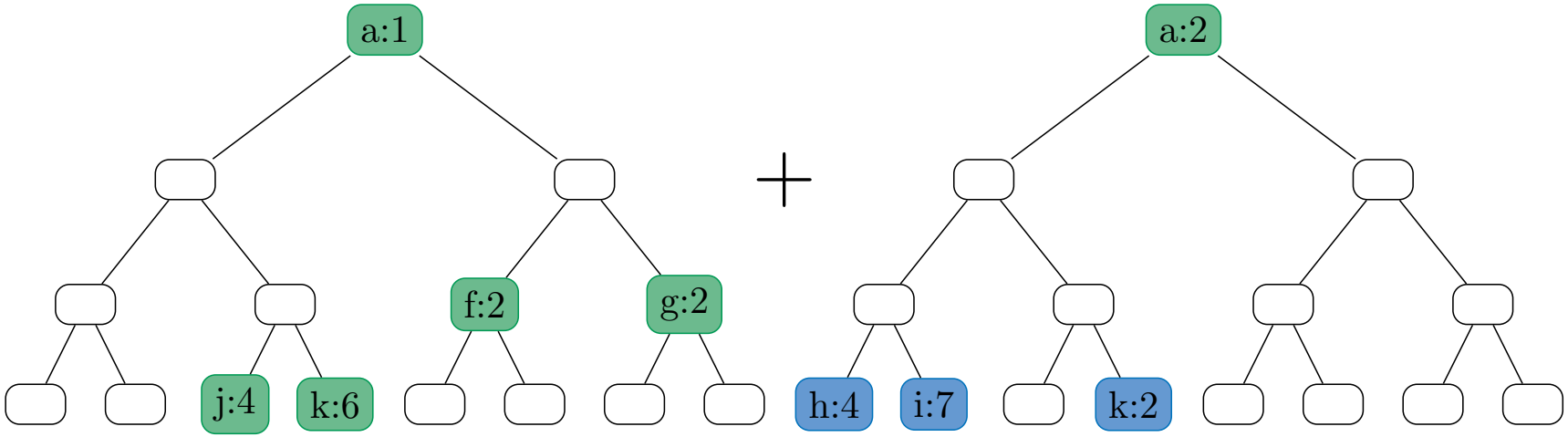
- q-digest stores at most $3k$ nodes
- relation of decompression and error:
  - setting: $X_n \sim \mathcal{N}(\mu, \sigma_{\mathcal{N}}), \phi \in [0 : .1 : 1]$
  - error between equiprobable histogram and histogram computed on a single q-digest



$$e_{hist}(H_1, H_2) \quad := \quad \frac{1}{\sigma^2 B} \sum (h_{1,i} - h_{2,i})^2$$

Q-Digest
Merging

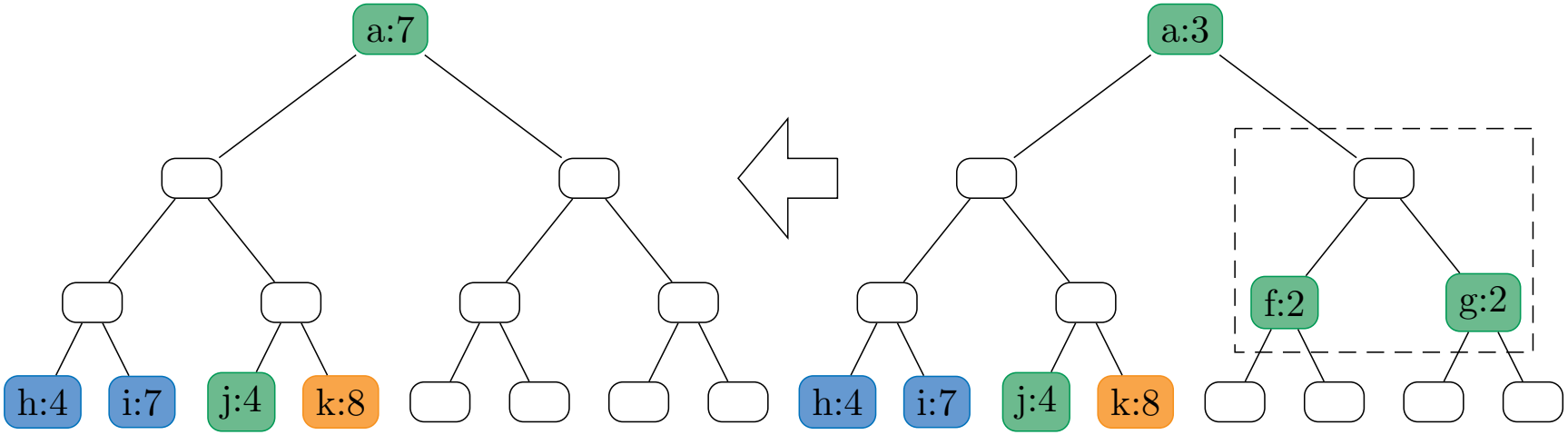
proc Merge ($\mathbf{qdigest}\ Q_1, \mathbf{int}\ n_1, \mathbf{qdigest}\ Q_2, \mathbf{int}\ n_2, \mathbf{int}\ k$) $\equiv$
  $Q := Q_1 \cup Q_2$
  Compress($Q, n_1 + n_2, k$)
  return Q
end



n = 10, k = 5, σ = 8 n = 15, k = 5, σ = 8

Q-Digest
Merging

proc Merge $(\textbf{qdigest } Q_1, \textbf{int } n_1, \textbf{qdigest } Q_2, \textbf{int } n_2, \textbf{int } k) \equiv$
  $Q := Q_1 \cup Q_2$
  Compress$(Q, n_1 + n_2, k)$
  return Q
end



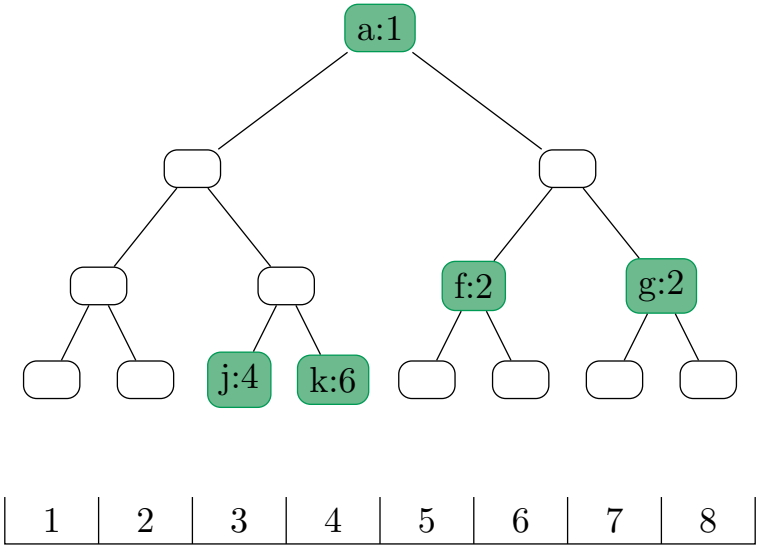
$n = 25$, $k = 5$, $\sigma = 8$ $n = 25$, $k = 5$, $\sigma = 8$

Q-Digest

Quantile and Histogram Computation

```
proc quantile (qdigest Q, float φ) ≡
  L := postorder(Q.tree)
  s := 0
  for v ∈ L do
      s := s + v.count
      if s >= φ · Q.n
        return rightLeaf(v, σ)
      fi
  end
  return rightLeaf(L.end, σ)
end
```

```
proc histogramEquiProb (qdigest Q, ...
  int σ, float τ) ≡
  φ := [0 : τ : 1]
  comment: determine 1/τ + 1 quantiles
  q_i := quantile(Q, φ_i)  ∀i ∈ [1..|φ|]
  return q
end
```



Evaluation

Gossiping of Q-digests

proc Initialize$(data, k, \sigma)$ $\equiv$
  $leaves :=$ buildtree$(data, \sigma)$;
  $n := |data|$
  $Q :=$ Compress$(leaves, n, k)$
end
proc Timer $\equiv$
  $peer :=$ SelectRandomPeer()
  sendTo $peer$: Shuffle(Q, n)
end

proc Shuffle(Q_{rmt}, n_{rmt}) **from** p $\equiv$
  sendTo p : ShuffleResponse(Q)
  $Q :=$ Merge(Q, n, Q_{rmt}, n_{rmt})
end
proc ShuffleResp(Q_{rmt}, n_{rmt}) **from** p $\equiv$
  $(Q, n) :=$ Merge(Q, n, Q_{rmt}, n_{rmt})
end

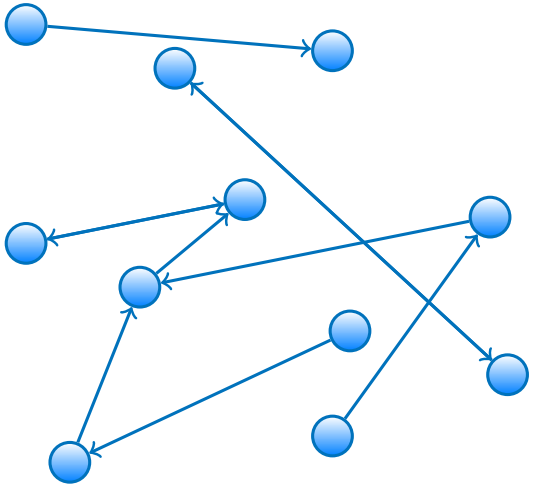
proc Request(τ) **from** p $\equiv$
  $H :=$ histogramEquiProb(Q, σ, τ)
  sendTo p : RequestResponse(H)
end

Gossiping of Q-digests

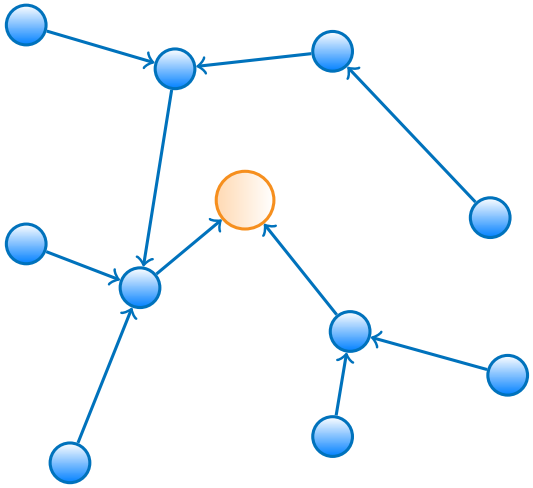
Error

Error between randomly merged q-digest (gossiped) or deterministic merging along a routing tree (routed)

$$e_{hist}(H_1, H_2) \quad := \quad \frac{1}{\sigma^2 B} \sum (h_{1,i} - h_{2,i})^2$$



(c) Gossiping.



(d) Routing Tree.

Gossiping of Q-digests

Error

Error between randomly merged q-digest (gossiped) or deterministic merging along a routing tree (routed)

$$e_{hist}(H_1, H_2) \quad := \quad \frac{1}{\sigma^2 B} \sum (h_{1,i} - h_{2,i})^2$$

Setup:

- 2^{10} sensors/nodes storing 2^{10} items sampled randomly either from
  1. one global normal distribution $\mathcal{N}(\mu, \sigma_{\mathcal{N}})$ with $\mu \in [\sigma/4; 3\sigma/4]$ and $\sigma_{\mathcal{N}}$
  2. 2^{10} different normal distributions $\mathcal{N}^i(\mu^i, \sigma^i_{\mathcal{N}})$ with $\mu^i \in [\sigma/4; 3\sigma/4]$, $\sigma^i_{\mathcal{N}} \in [1; \sigma/4]$, and $i \in [1; 2^{10}]$
- initially each node computes a q-digest on its local data set
- in each round q-digests are exchanged, merged, and replace the local ones

Gossiping of Q-digests

Result 1

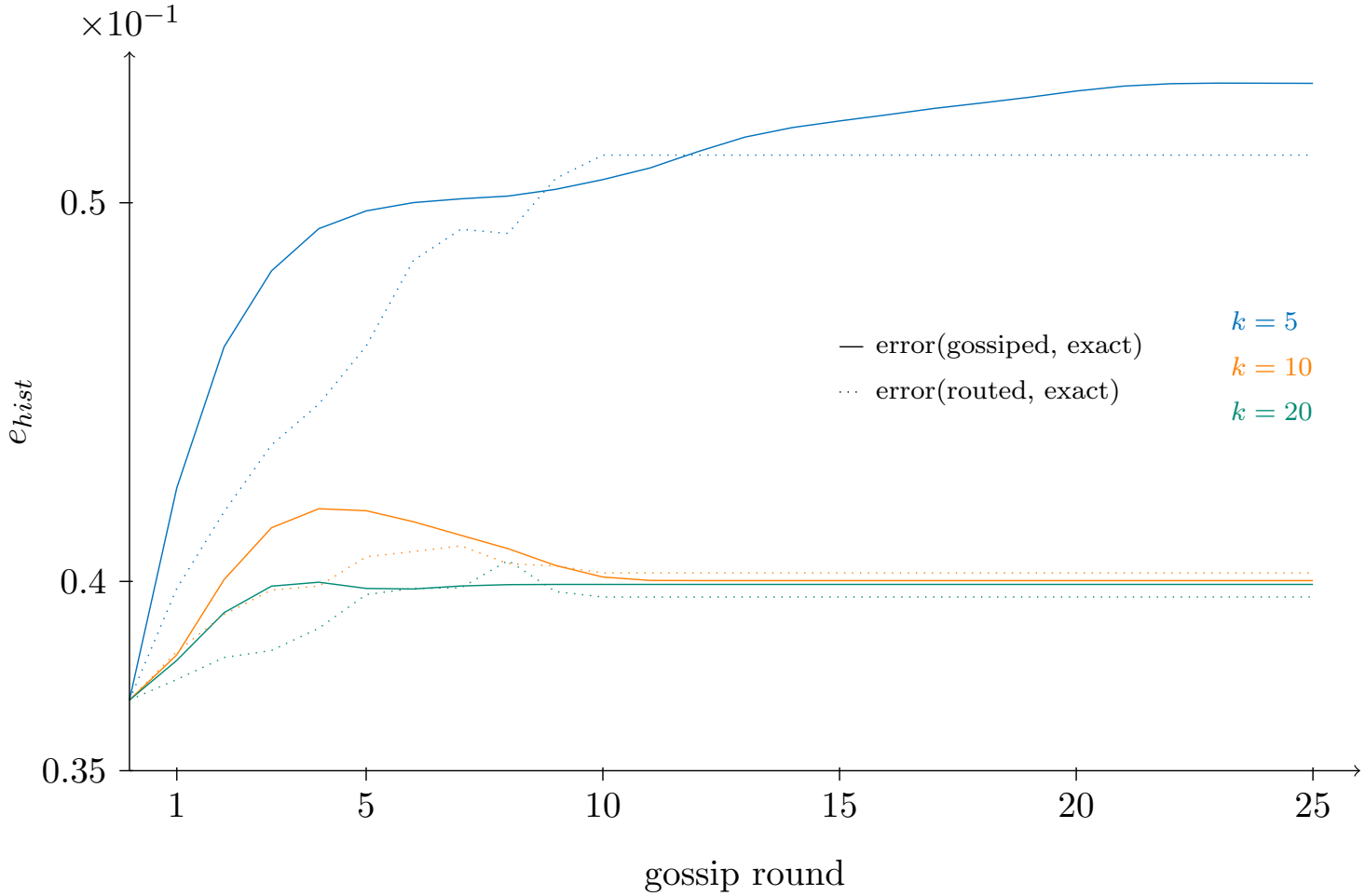


Figure: Data distributed according to one common normal distribution.

Gossiping of Q-digests

Result 2

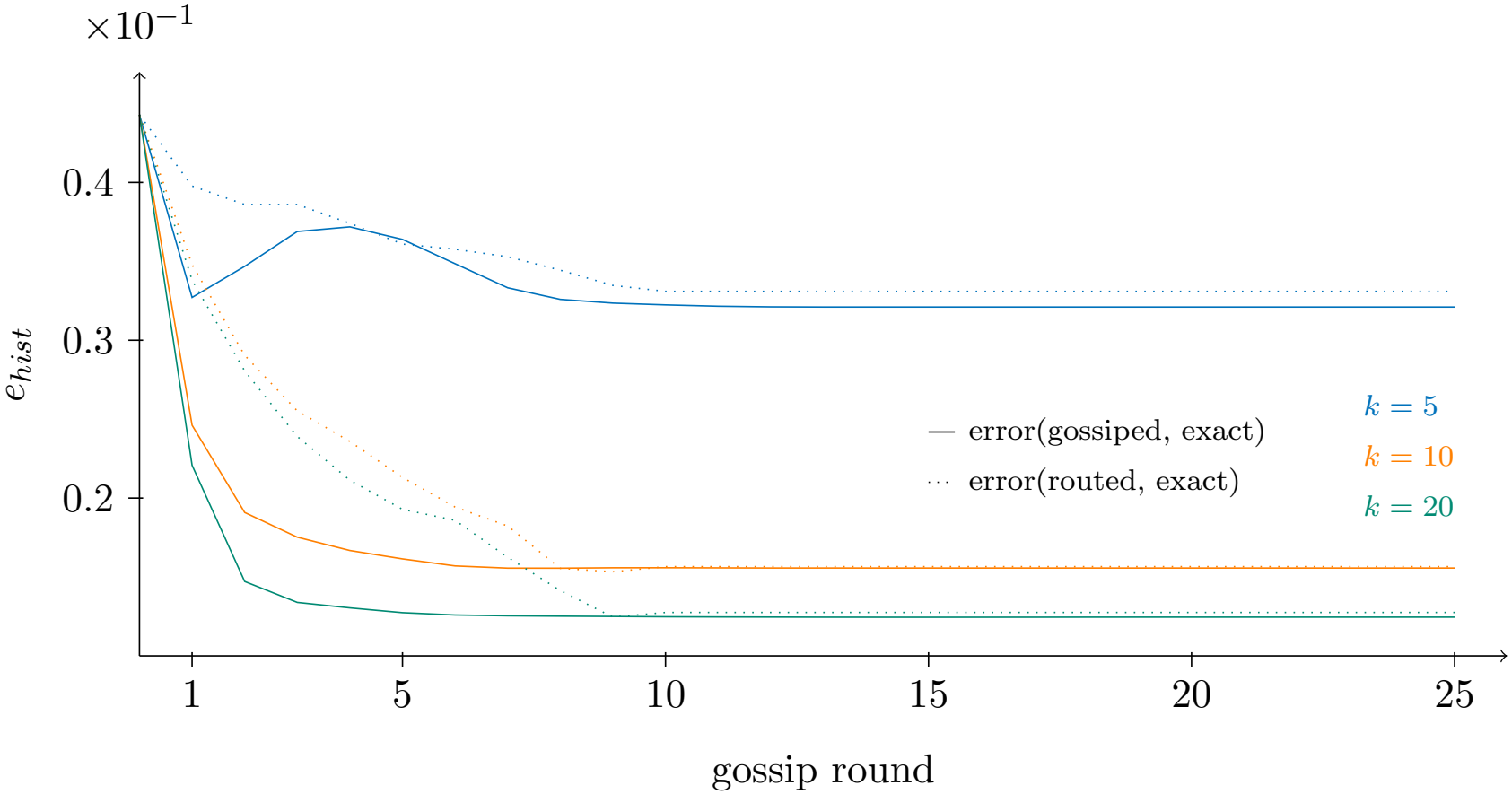
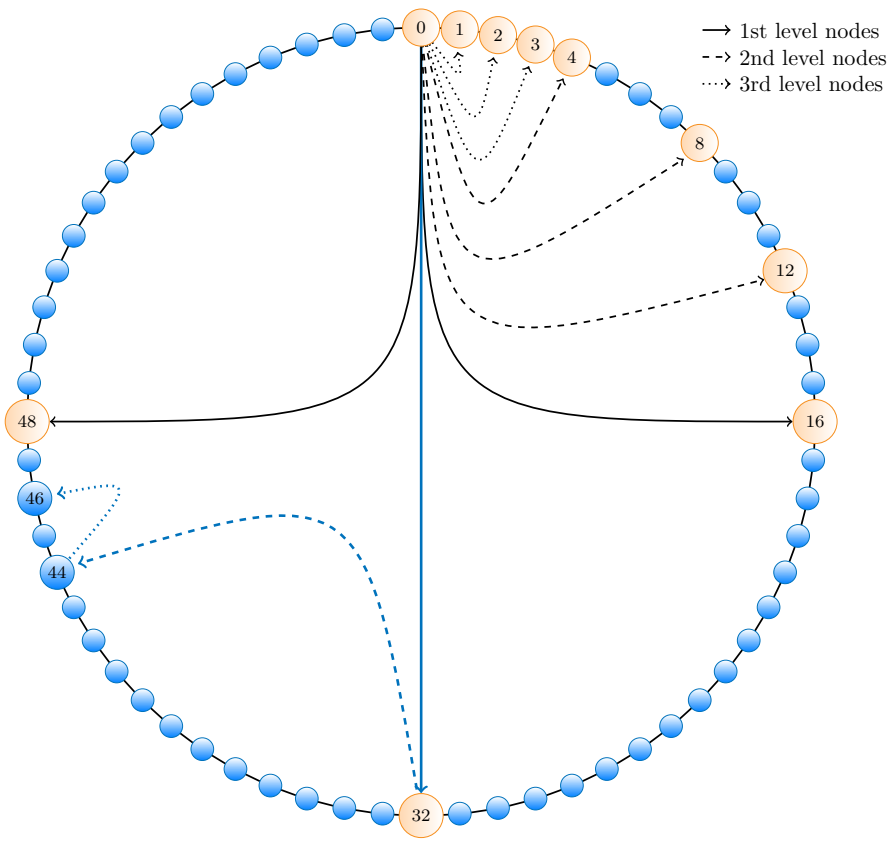


Figure: Data distributed according to s different normal distribution.

Conclusions

- generally: error for gossiping converges towards some k-dependent constant
- one global distibution: merging q-digests randomly increases the error negligible
- different distributions for each sensor:
  - same error as finite, deterministic merging along a routing tree
  - faster convergence for gossiping
- consequence for streams: one could proceed as follows
  1. collect data until buffer is filled
  2. compute q-digest on buffer and merge it with local one
  3. clear buffer, goto step 1
  4. meanwhile gossip and merge local with remote q-digests

Part 3: How to reduce the traffic for maintenance messages?



Traffic in P2P Networks

Overlays: peers are transient

- nodes join and have to be linked
- nodes leave and have to be deregistered, replaced, etc.

$\Rightarrow$ messages for checking presence/abscence of nodes have to be sent permanently

- heuristics could help to reduce their frequency, e.g. adjust frequency to **average lifetime** of nodes

Weibull Distribution

- first identified by *Maurice Fréchet* in 1927, described in detail by *Waloddi Weibull*
- wide range application: breaking strength of material, size distribution of particles, failure probability of electronic devices
- depending on input parameters $k > 0$ (shape) and $\lambda > 0$ (scale) the Weibull distribution may assume the shape of an exponential, normal or Rayleigh distribution

The 3-parameter cdf F with θ (location) for the Weibull function is

$$F(x; k, \lambda, \theta) := \begin{cases} 1 - e^{-(\frac{x-\theta}{\lambda})^k} & , x \geq \theta \\ 0 & , x < \theta \end{cases}$$

Weibull Distribution

pdf and mean

Differentiating F with respect to x results in the *frequency* or *probability density function* f

$$f(x;k,\lambda,\theta) = \frac{dF}{dx} = \begin{cases} \frac{k}{\lambda}\left(\frac{x-\theta}{\lambda}\right)^{k-1} e^{-\left(\frac{x-\theta}{\lambda}\right)^k} & , x \geq \theta \\ 0 & , x < \theta \end{cases}$$

The mean of the Weibull function is

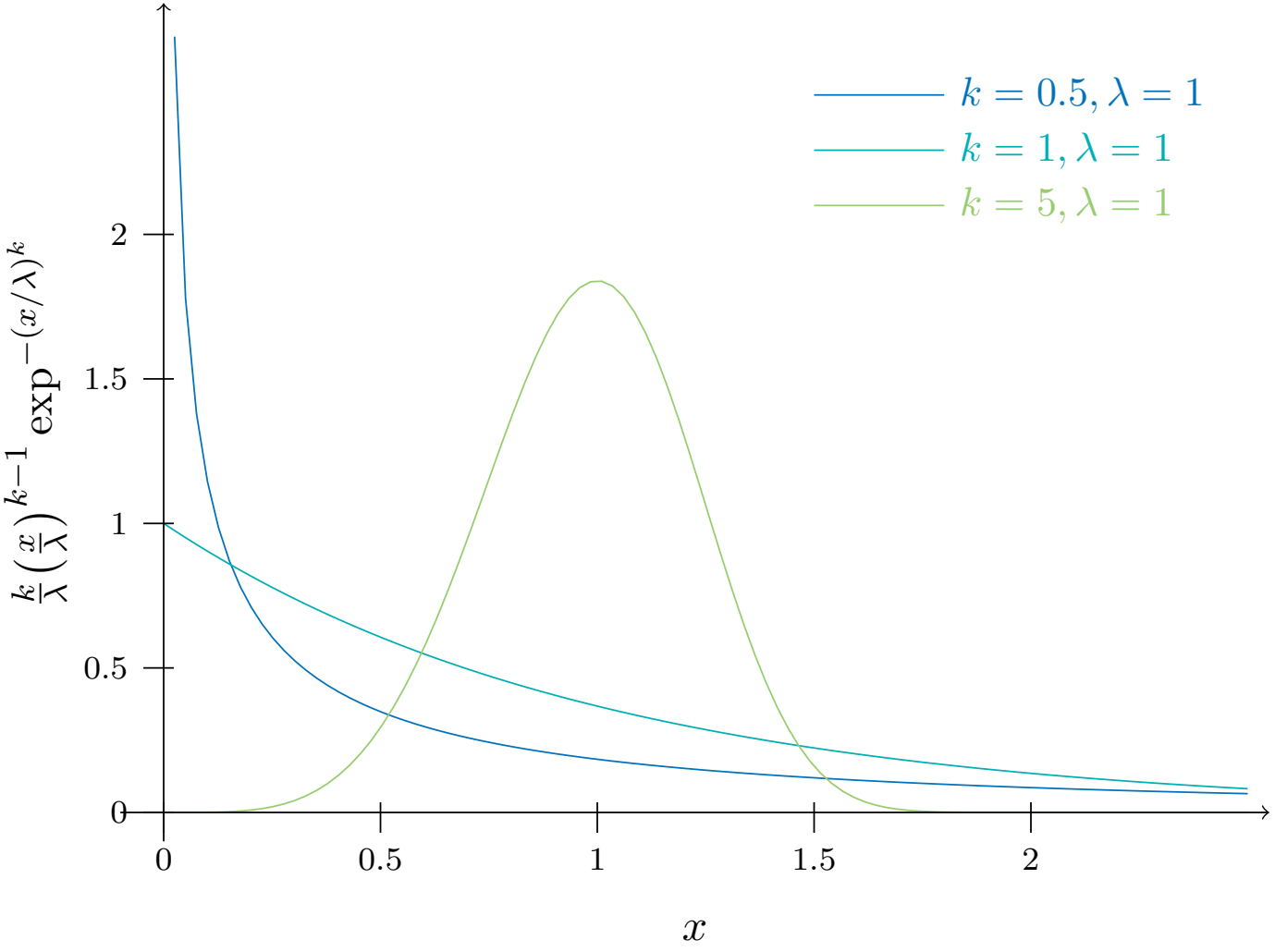
$$\mu = \lambda\Gamma(1 + 1/k)$$

with $\Gamma(x) := \int_0^\infty t^{x-1} \exp^{-t} dt$

Weibull Distribution

Pdf

The Weibull frequency distribution for $\theta = 0$



Linear Regression

Given a set of sampled lifetimes, how to determine the missing parameters?

Let $X \in \mathbb{R}^{n \times m}$, $Y \in \mathbb{R}^{n \times 1}$

- with linear relationship
  $y_i = \alpha_0 + \alpha_1 x_{i,1} + \alpha_2 x_{i,2} + \cdots + \alpha_m x_{i,m}, \forall i \in [1..n]$
- goal: determine coefficient vector α s.t. squared error E between samples and regression line is minimal
- prepend column vector 1_n to X s.t. we can write $X\alpha = Y$

$$E = \frac{1}{2}(Y - X\alpha)^T(Y - X\alpha) \qquad \rightarrow \min!$$

Linear Regression

Determine root of E's derivation for α:

$$
\begin{aligned}
\frac{\partial E}{\partial \alpha} &= 0 \\
\partial\Big[\frac{1}{2}(Y - X\alpha)^T(Y - X\alpha)\Big]/\partial\alpha &= 0 \\
\frac{1}{2}(-X)^T(Y - X\alpha) + \frac{1}{2}(Y - X\alpha)^T(-X) &= 0 \\
-X^T(Y - X\alpha) &= 0 \\
-X^TY + X^TX\alpha &= 0 \\
\alpha &= (X^TX)^{-1}(X^TY)
\end{aligned}
$$

Linear Regression

Applied to Weibull Cdf

Now linearize the Weibull cumulative distribution function F

$$
\begin{aligned}
F(x;k,\lambda) &= 1-\exp^{-(x/\lambda)^k} \\
-\ln(1-F(x;k,\lambda)) &= (x/\lambda)^k \\
\underbrace{\ln\big(-\ln(1-F(x;k,\lambda))\big)}_{y} &= \underbrace{k\ln x}_{mx} - \underbrace{k\ln\lambda}_{c}
\end{aligned}
$$

The relationship between the double logarithm of F and the logarithm of x is linear! $\Rightarrow$ apply solution of linear regression

Linear Regression

Applied to Weibull Cdf

Given a sample x_i, we have the linear relationship

$$\ln(-\ln(1-F(x_i))) = \begin{pmatrix} \ln x_i & 1 \end{pmatrix} \cdot \begin{pmatrix} k & c \end{pmatrix}^T$$

$$\underbrace{\begin{pmatrix} \ln(-\ln(1-F(x_1)) \\ \ln(-\ln(1-F(x_2)) \\ \vdots \\ \ln(-\ln(1-F(x_n)) \end{pmatrix}}_{\tilde{Y}} = \underbrace{\begin{pmatrix} \ln x_1 & 1 \\ \ln x_2 & 1 \\ \vdots & \vdots \\ \ln x_n & 1 \end{pmatrix}}_{\tilde{X}} \begin{pmatrix} k \\ c \end{pmatrix}$$

Plugged into linear regression formula:

$$\begin{pmatrix} k \\ c \end{pmatrix} = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T \tilde{Y}$$

Use solution of k, c to compute λ

$$\lambda = \exp^{-c/k}$$

Mean Lifetime Estimation

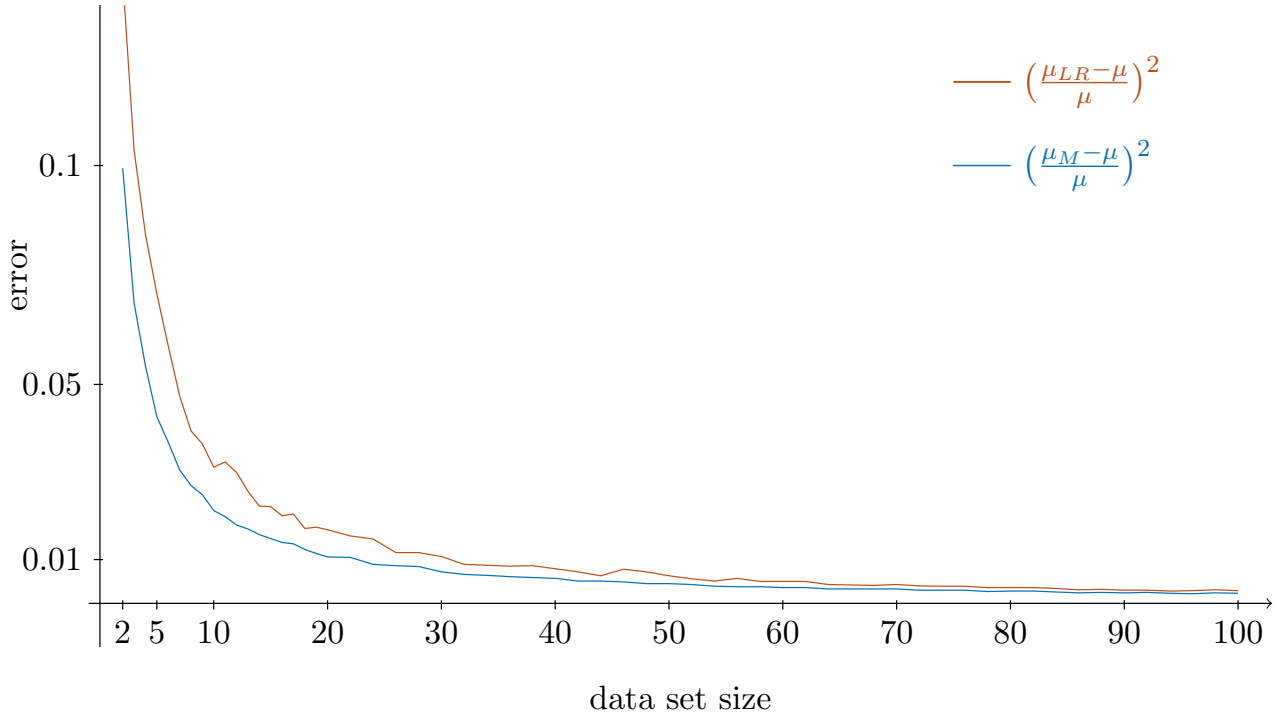
Put Together

Usage in a P2P system running e.g. the Chord protocol

1. Collect some lifetimes X from unavailable peers
2. Compute $\tilde{X}$, $\tilde{Y}$ from X
3. Compute $(k \quad c)^T$ from linear regression formula
4. Compute mean lifetime $\mu = \lambda\Gamma(1 + 1/k)$ from $k, \lambda = \exp^{-c/k}$
5. Use μ for adjusting frequency of maintenance messages of protocol

Evaluation – Linear Regression versus Latent Solution

- scale λ fixed, shape k taken with equal probability from $[1; 5]$
- $data = \texttt{wblrnd}(\lambda, k)$ with $|data| \in [2; 100]$
- μ was computed from parameter estimation using linear regression (LR), Matlab's (M) built-in `wblfit`, and exact, but latent parameters

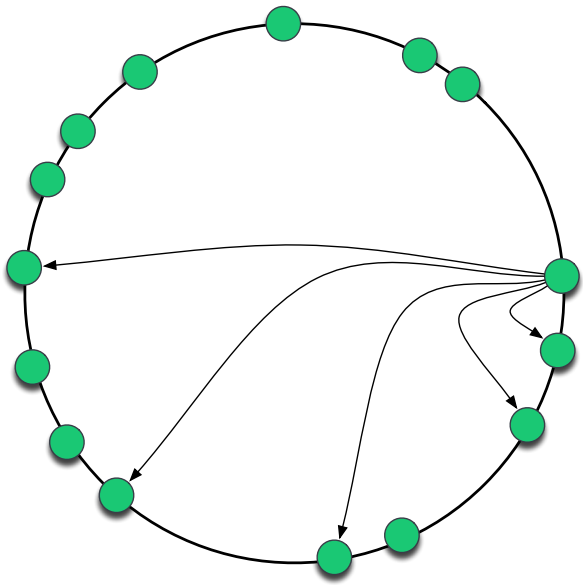


Evaluation

Frequency Reduction in Chord

Chord – structured P2P protocol

- nodes arranged in a ring-like structure with $I = [0; N]$ being the identifier space
- for routing each node stores a **finger table** with pointers to successors
- with high probability the look-up for a node is performed with $\log N$ steps



Evaluation

Fix Finger Messages in Chord

Periodically for each entry in the finger table a look-up is performed

- a finger table has $\log N$ fingers
- a look-up has costs of $\log N$ steps

$\Rightarrow$ to refresh a whole finger table costs $O(\log^2 N)$

- necessity for a refresh depends on average lifetime of peers

$\Rightarrow$ make frequency for a refresh dependent on estimated lifetime

Evaluation
Chord Simulation

Oversim – a simulation framework which contains an implementation of Chord

- added function `churnRateEstimator` to sample lifetimes (from non-responding nodes) and to compute the mean lifetime like described before
- function `handleFixFingersTimerExpired` is called periodically if the timer for fix finger messages expired, it does
  - call `churnRateEstimator` and returns μ
  - computes a new `frequency` from μ which triggers an internal switch for the sending of `fix_finger` messages

$$\texttt{frequency} = \log^2 \mu$$

Evaluation
Chord Simulation

Oversim provides an interface for collecting statistics for traffic, among them are counters for

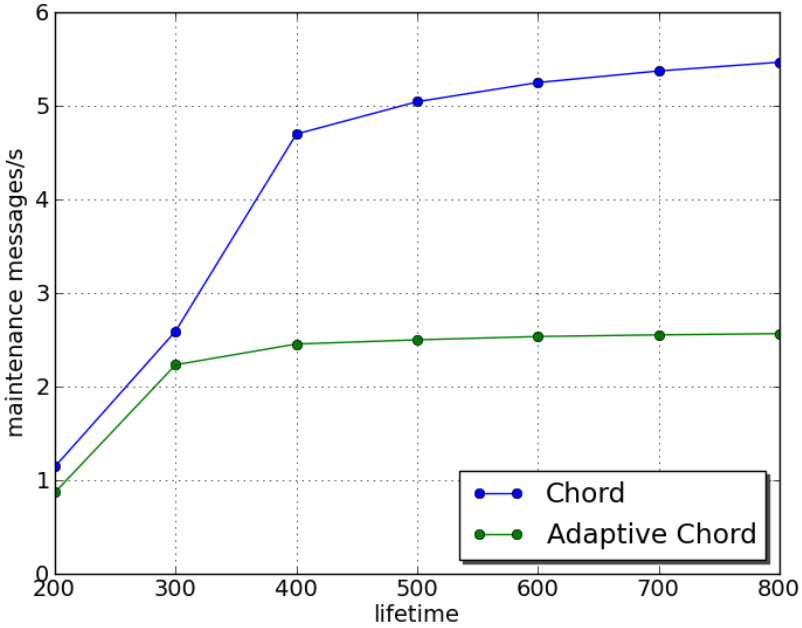
- *maintenance messages*
- *fixfinger messages*
- *ping and pingResponse messages*
- *packets dropped*
- *one-way hop count*

Setup

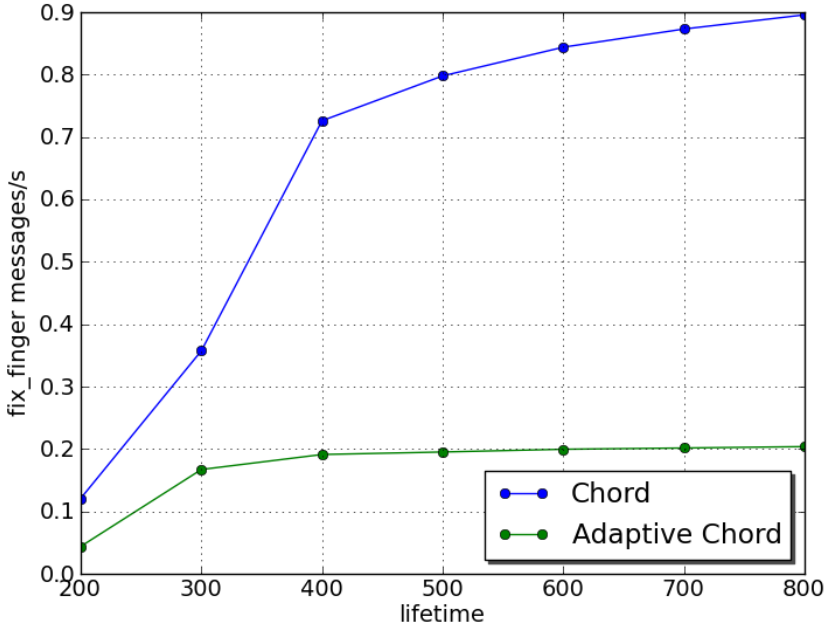
- steady-state net size: 1024
- measuring time: 5000s
- mean lifetimes in $[200; 800]$

Evaluation

Chord Simulation Results 1



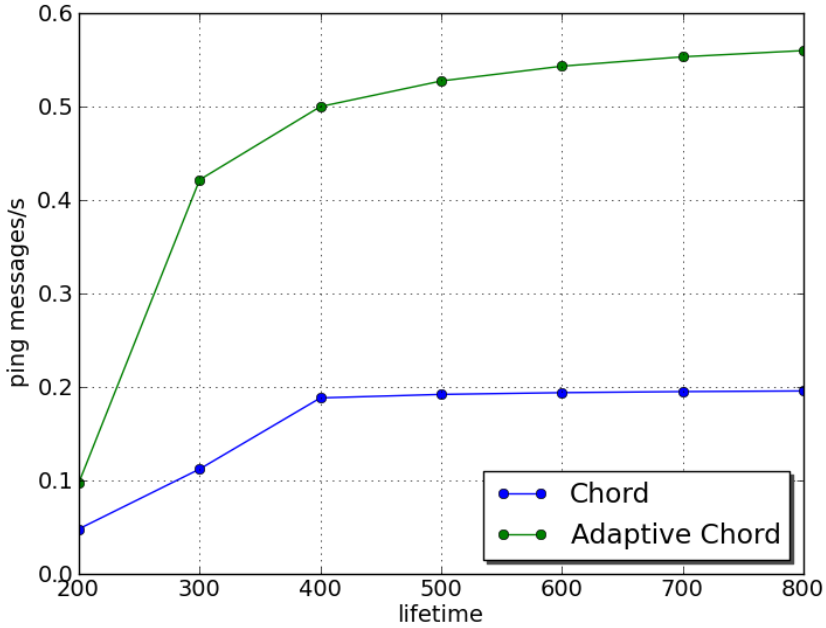
(a) Number of maintenance messages.



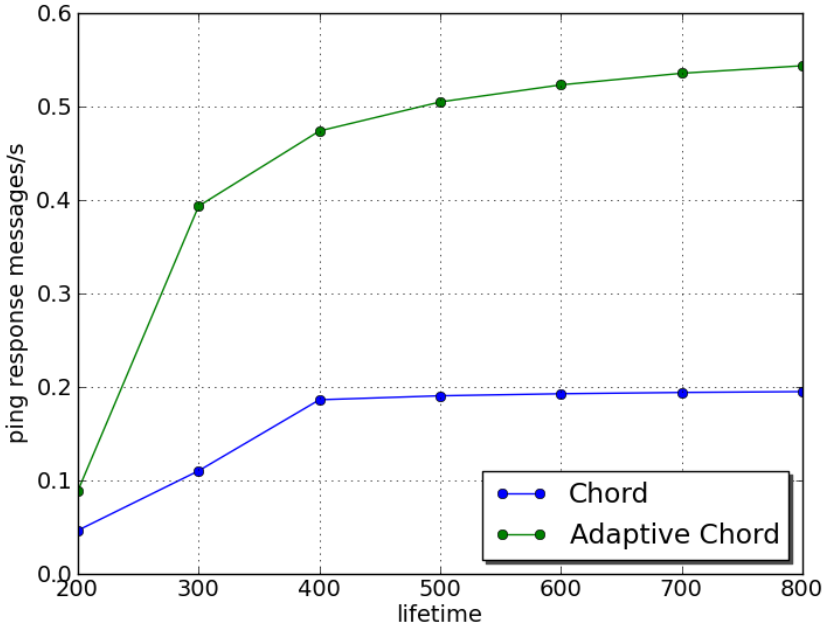
(b) Number of fixfinger messages.

Evaluation

Chord Simulation Results 2



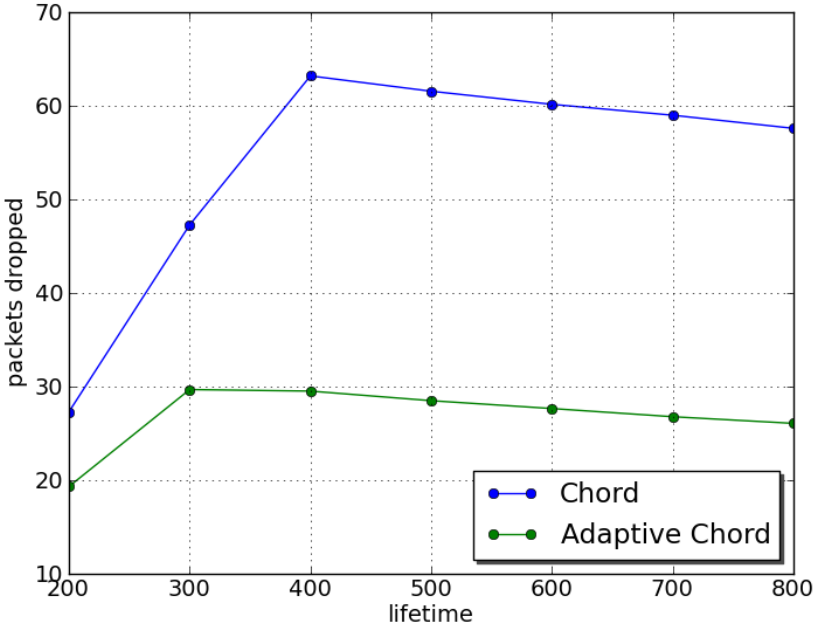
(c) Number of ping messages.



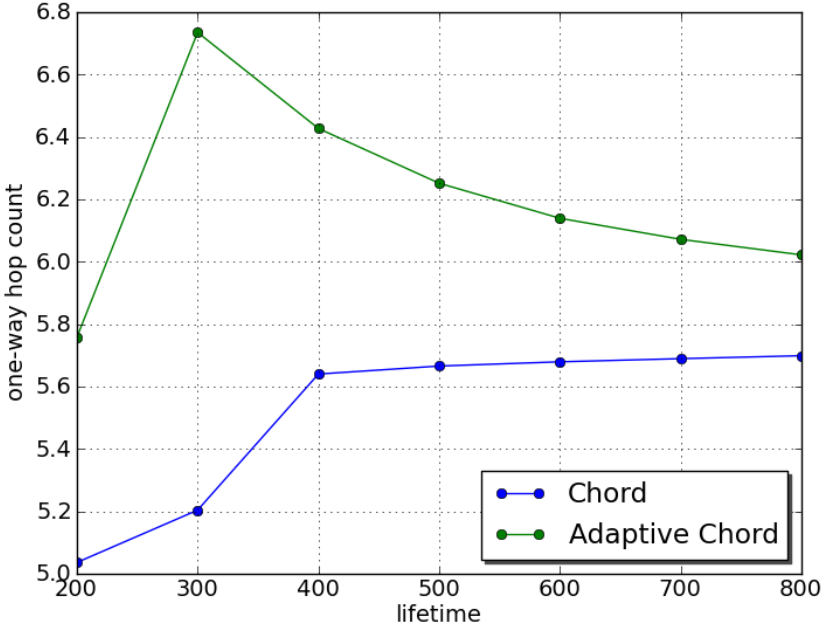
(d) Number of ping response messages.

Evaluation

Chord Simulation Results 3



(e) Packets dropped due to unavailable destination.



(f) One-way hop count.

Thank you, for your attention!