Pastry

R. Himmelmann

Ferienakademie im Sarntal 2008
FAU Erlangen-Nürnberg, TU München, Uni Stuttgart

October 12, 2008

Table of Contents

Table of Contents

Table of Contents

Table of Contents

IDs

- IDs are numbers $\in \mathbb{N}/(2^{128}\mathbb{N}) \equiv \{0, ..., 2^{128} - 1\} := ID$
- $\forall b \in ID. |b| := \min(b, 2^{128} - 1 - b)$
- Every node p in the network has an ID $id(p) \in ID$.
- Every piece of data d has an ID $id(d)$
- ... and is associated with the node p with $|id(p) - id(d)| = min$.

IDs

- IDs are numbers $\in \mathbb{N}/(2^{128}\mathbb{N}) \equiv \{0, ..., 2^{128} - 1\} := ID$
- $\forall b \in ID.|b| := \min(b, 2^{128} - 1 - b)$
- Every node p in the network has an ID $id(p) \in ID$.
- Every piece of data d has an ID $id(d)$
- ... and is associated with the node p with $|id(p) - id(d)| = min$.

IDs

- IDs are numbers $\in \mathbb{N}/(2^{128}\mathbb{N}) \equiv \{0, ..., 2^{128} - 1\} := ID$
- $\forall b \in ID.|b| := \min(b, 2^{128} - 1 - b)$
- Every node p in the network has an ID $id(p) \in ID$.
- Every piece of data d has an ID $id(d)$
- ... and is associated with the node p with $|id(p) - id(d)| = min$.

IDs

- IDs are numbers $\in \mathbb{N}/(2^{128}\mathbb{N}) \equiv \{0, ..., 2^{128} - 1\} := ID$
- $\forall b \in ID. |b| := \min(b, 2^{128} - 1 - b)$
- Every node p in the network has an ID $id(p) \in ID$.
- Every piece of data d has an ID $id(d)$
- ... and is associated with the node p with $|id(p) - id(d)| = min$.

IDs

- IDs are numbers $\in \mathbb{N}/(2^{128}\mathbb{N}) \equiv \{0, ..., 2^{128} - 1\} := ID$
- $\forall b \in ID.|b| := \min(b, 2^{128} - 1 - b)$
- Every node p in the network has an ID $id(p) \in ID$.
- Every piece of data d has an ID $id(d)$
- ... and is associated with the node p with $|id(p) - id(d)| = min$.

IDs

- Interpret IDs as numbers with base $|B| = 2^b$
- Let $m = \log_{|B|} |ID|$
- $pfxl(x, y)$ is the length of the greatest common prefix of x and y.
  - $pfxl(p, q) := pfxl(id(p), id(q))$ for nodes p and q.

IDs

- Interpret IDs as numbers with base $|B| = 2^b$
- Let $m = \log_{|B|} |ID|$
- $pfxl(x, y)$ is the length of the greatest common prefix of x and y.
  - $pfxl(p, q) := pfxl(id(p), id(q))$ for nodes p and q.

IDs

- Interpret IDs as numbers with base $|B| = 2^b$
- Let $m = \log_{|B|} |ID|$
- $pfxl(x, y)$ is the length of the greatest common prefix of x and y.
  - $pfxl(p, q) := pfxl(id(p), id(q))$ for nodes p and q.

IDs

- Interpret IDs as numbers with base $|B| = 2^b$
- Let $m = \log_{|B|} |ID|$
- $pfxl(x, y)$ is the length of the greatest common prefix of x and y.
  - $pfxl(p, q) := pfxl(id(p), id(q))$ for nodes p and q.

Proximity

Assumption: The cost for sending a message from p to q may be measured by a metric d.

- $d(p,q) = d(q,p)$
- $d(p,q) \leq d(p,p') + d(p',q)$

Proximity

Assumption: The cost for sending a message from p to q may be measured by a metric d.

- $d(p,q) = d(q,p)$
- $d(p,q) \leq d(p,p') + d(p',q)$

Proximity

Assumption: The cost for sending a message from p to q may be measured by a metric d.

- $d(p, q) = d(q, p)$
- $d(p, q) \leq d(p, p') + d(p', q)$

What do we want to optimize?

- Every operation should need as few messages as possible.
- The overall distance a message travels should be minimal.
    - A greedy algorithm is used.
    - Fill the routing tables of nodes only with near nodes.

What do we want to optimize?

- Every operation should need as few messages as possible.
- The overall distance a message travels should be minimal.
  - A greedy algorithm is used.
  - Fill the routing tables of nodes only with near nodes.

What do we want to optimize?

- Every operation should need as few messages as possible.
- The overall distance a message travels should be minimal.
  - A greedy algorithm is used.
  - Fill the routing tables of nodes only with near nodes.

What do we want to optimize?

- Every operation should need as few messages as possible.
- The overall distance a message travels should be minimal.
  - A greedy algorithm is used.
  - Fill the routing tables of nodes only with near nodes.

Overview

The Routing Table

- Let p be a node.
- $R_p =: R$ is a matrix $(R[i,j])_{0 \leq i < m, 0 \leq j < |B|}$.
- $R[i,j] =: q$ is a node with
  - $pfxl(p,q) = i$
  - $id(q)[i] = j$
- If there is no such q or if $j = id(p)[i]$ then $R[i,j] = null$.
- If possible choose a q near to p.

The Routing Table

- Let p be a node.
- $R_p =: R$ is a matrix $(R[i,j])_{0 \leq i < m, 0 \leq j < |B|}$.
- $R[i,j] =: q$ is a node with
  - $pfxl(p,q) = i$
  - $id(q)[i] = j$
- If there is no such q or if $j = id(p)[i]$ then $R[i,j] = null$.
- If possible choose a q near to p.

The Routing Table

- Let p be a node.
- $R_p =: R$ is a matrix $(R[i,j])_{0 \leq i < m, 0 \leq j < |B|}$.
- $R[i,j] =: q$ is a node with
  - $pfxl(p,q) = i$
  - $id(q)[i] = j$
- If there is no such q or if $j = id(p)[i]$ then $R[i,j] = null$.
- If possible choose a q near to p.

The Routing Table

- Let p be a node.
- $R_p =: R$ is a matrix $(R[i,j])_{0 \leq i < m, 0 \leq j < |B|}$.
- $R[i,j] =: q$ is a node with
  - $pfxl(p,q) = i$
  - $id(q)[i] = j$
- If there is no such q or if $j = id(p)[i]$ then $R[i,j] = null$.
- If possible choose a q near to p.

The Routing Table

- Let p be a node.
- $R_p =: R$ is a matrix $(R[i,j])_{0 \leq i < m, 0 \leq j < |B|}$.
- $R[i,j] =: q$ is a node with
  - $pfxl(p,q) = i$
  - $id(q)[i] = j$
- If there is no such q or if $j = id(p)[i]$ then $R[i,j] = null$.
- If possible choose a q near to p.

The Routing Table

- Let p be a node.
- $R_p =: R$ is a matrix $(R[i,j])_{0 \leq i < m, 0 \leq j < |B|}$.
- $R[i,j] =: q$ is a node with
  - $pfxl(p,q) = i$
  - $id(q)[i] = j$
- If there is no such q or if $j = id(p)[i]$ then $R[i,j] = null$.
- If possible choose a q near to p.

The Routing Table

- Let p be a node.
- $R_p =: R$ is a matrix $(R[i,j])_{0 \leq i < m, 0 \leq j < |B|}$.
- $R[i,j] =: q$ is a node with
  - $pfxl(p,q) = i$
  - $id(q)[i] = j$
- If there is no such q or if $j = id(p)[i]$ then $R[i,j] = null$.
- If possible choose a q near to p.

Example

	..1..	..2..	..3..	..4..
xxxx	*null*	2xxx	3xxx	4xxx
1xxx	11xx	*null*	13xx	14xx
12xx	121x	122x	*null*	124x
123x	1231	*null*	1233	1234

$$R_p \text{ with } id(p) = 1232$$

$$|B| = 4.$$

Size of R

Theorem

R_p for node p contains usually $\leq \mathcal{O}(\frac{\log n}{b} 2^b)$ entries.

Overview

The Leaf Set

- The leaf set L for a node p is an array with
  - $|L|/2$ nodes with next higher IDs and
  - $|L|/2$ nodes with next lower IDs.
- If $|L|/2 > n$ nodes may be on both sides of the leaf "set".

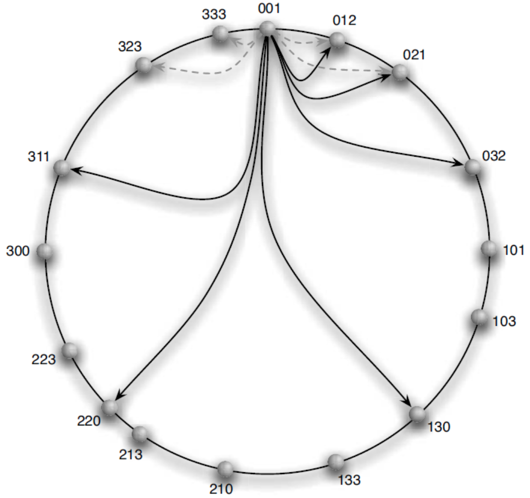
The Leaf Set

- The leaf set L for a node p is an array with
  - $|L|/2$ nodes with next higher IDs and
  - $|L|/2$ nodes with next lower IDs.
- If $|L|/2 > n$ nodes may be on both sides of the leaf "set".

The Leaf Set

- The leaf set L for a node p is an array with
  - $|L|/2$ nodes with next higher IDs and
  - $|L|/2$ nodes with next lower IDs.
- If $|L|/2 > n$ nodes may be on both sides of the leaf "set".

Example



Overview

The Neighbourhood Set

- The Neighbourhood M of a node p contains $|M| = 2^b$ nodes.
- For all $q \in M$ the distance $d(p, q)$ should be small.
- M is used for repairs and insertion of peers.

The Neighbourhood Set

- The Neighbourhood M of a node p contains $|M| = 2^b$ nodes.
- For all $q \in M$ the distance $d(p, q)$ should be small.
- M is used for repairs and insertion of peers.

The Neighbourhood Set

- The Neighbourhood M of a node p contains $|M| = 2^b$ nodes.
- For all $q \in M$ the distance $d(p, q)$ should be small.
- M is used for repairs and insertion of peers.

Example 2

NodeId 10233102

Leaf set

SMALLER		LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing table

-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Neighborhood set

13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

Overview

Algorithm for Routing

- For a given $r \in ID$ we want to find the node n with $|id(n) - r| = \min$.
- We start at a node p.
- If n is in the leaf set we forward the message to it.
- Otherwise let $c = pfxl(id(p), r)$
- If $R[c, r[c]] \neq null$ forward to that node.
- Otherwise route to the "best" node p' known to p with $|r - id(p')| < |r - id(p)|$ and $pfxl(id(p'), r) \geq r$.

Algorithm for Routing

- For a given $r \in ID$ we want to find the node n with $|id(n) - r| = \min$.
- We start at a node p.
- If n is in the leaf set we forward the message to it.
- Otherwise let $c = pfxl(id(p), r)$
- If $R[c, r[c]] \neq null$ forward to that node.
- Otherwise route to the "best" node p' known to p with $|r - id(p')| < |r - id(p)|$ and $pfxl(id(p'), r) \geq r$.

Algorithm for Routing

- For a given $r \in ID$ we want to find the node n with $|id(n) - r| = \min$.
- We start at a node p.

- If n is in the leaf set we forward the message to it.
- Otherwise let $c = pfxl(id(p), r)$
- If $R[c, r[c]] \neq null$ forward to that node.
- Otherwise route to the "best" node p' known to p with $|r - id(p')| < |r - id(p)|$ and $pfxl(id(p'), r) \geq r$.

Algorithm for Routing

- For a given $r \in ID$ we want to find the node n with $|id(n) - r| = \min$.
- We start at a node p.
- If n is in the leaf set we forward the message to it.
- Otherwise let $c = pfxl(id(p), r)$
- If $R[c, r[c]] \neq null$ forward to that node.
- Otherwise route to the "best" node p' known to p with $|r - id(p')| < |r - id(p)|$ and $pfxl(id(p'), r) \geq r$.

Algorithm for Routing

- For a given $r \in ID$ we want to find the node n with $|id(n) - r| = \min$.
- We start at a node p.
- If n is in the leaf set we forward the message to it.
- Otherwise let $c = pfxl(id(p), r)$
- If $R[c, r[c]] \neq null$ forward to that node.
- Otherwise route to the "best" node p' known to p with $|r - id(p')| < |r - id(p)|$ and $pfxl(id(p'), r) \geq r$.

Algorithm for Routing

- For a given $r \in ID$ we want to find the node n with $|id(n) - r| = \min$.
- We start at a node p.
- If n is in the leaf set we forward the message to it.
- Otherwise let $c = pfxl(id(p), r)$
- If $R[c, r[c]] \neq null$ forward to that node.
- Otherwise route to the "best" node p' known to p with $|r - id(p')| < |r - id(p)|$ and $pfxl(id(p'), r) \geq r$.

Correctness

Theorem

Routing takes no more than $\mathcal{O}(n/|L|)$ steps.

- (Assume correct routing tables and leaf sets)

Theorem

The expected value is $\mathcal{O}(\log_{2^b} n) = \mathcal{O}(\frac{\log n}{b})$ messages.

Correctness

Theorem

Routing takes no more than $\mathcal{O}(n/|L|)$ steps.

- (Assume correct routing tables and leaf sets)

Theorem

The expected value is $\mathcal{O}(\log_{2^b} n) = \mathcal{O}(\frac{\log n}{b})$ messages.

Correctness

Theorem

Routing takes no more than $\mathcal{O}(n/|L|)$ steps.

- (Assume correct routing tables and leaf sets)

Theorem

The expected value is $\mathcal{O}(\log_{2^b} n) = \mathcal{O}(\frac{\log n}{b})$ messages.

Notes

- Usually not the node p with $|id(p) - r| = \min$. is needed.
- One of the k nodes nearest to r in the ID-space is sufficient.
- E.g. in Past, a file storage protocol layered atop pastry:
    - At least k copies of a file are stored in the k nodes with the closest IDs to p.

Notes

- Usually not the node p with $|id(p) - r| = \min.$ is needed.
- One of the k nodes nearest to r in the ID-space is sufficient.
- E.g. in Past, a file storage protocol layered atop pastry:
  - At least k copies of a file are stored in the k nodes with the closest IDs to p.

Notes

- Usually not the node p with $|id(p) - r| = \min.$ is needed.
- One of the k nodes nearest to r in the ID-space is sufficient.
- E.g. in Past, a file storage protocol layered atop pastry:
    - At least k copies of a file are stored in the k nodes with the closest IDs to p.

Notes

- Usually not the node p with $|id(p) - r| = \min$. is needed.
- One of the k nodes nearest to r in the ID-space is sufficient.
- E.g. in Past, a file storage protocol layered atop pastry:
  - At least k copies of a file are stored in the k nodes with the closest IDs to p.

Overview

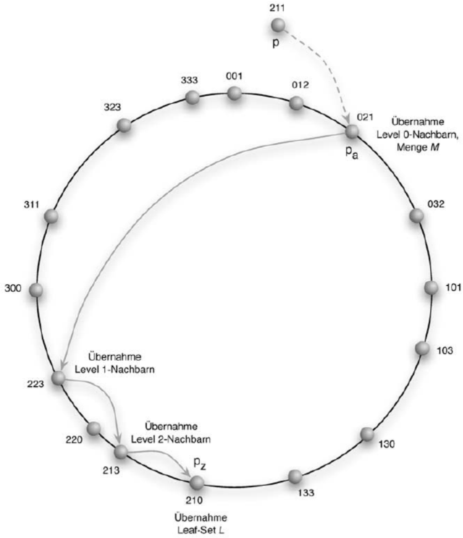
Insertion of a peer

- Let p be a new peer with tables R, L and M.
- Try to choose values for the routing table so, that distances are minimal.

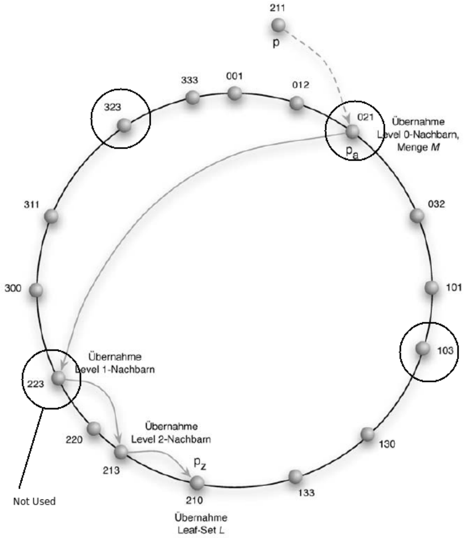
Insertion of a peer

- Let p be a new peer with tables R, L and M.
- Try to choose values for the routing table so, that distances are minimal.

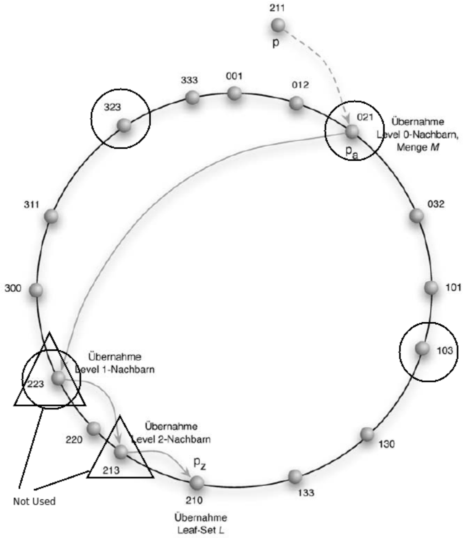
Insertion of a peer



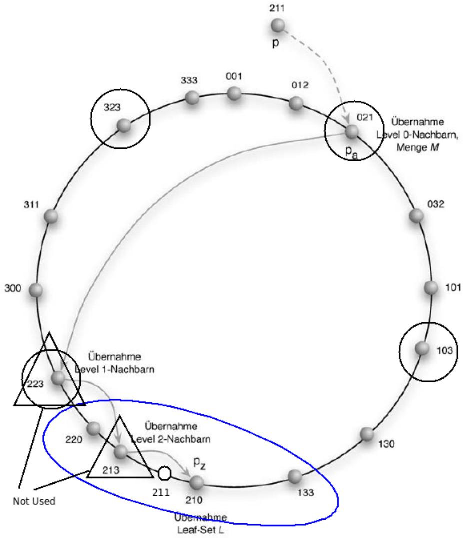
Insertion of a peer



Insertion of a peer



Insertion of a peer



Insertion of a peer

- Request R_q from all $q \in M$ and look for better entries for R.
- Notify every peer in M, L and R about our arrival.

Insertion of a peer

- Request R_q from all $q \in M$ and look for better entries for R.
- Notify every peer in M, L and R about our arrival.

Overview

Locality

- IDs are distributed randomly through the underlying network.
- It is unlikely that $|L|/2$ nodes with consecutive IDs fail.
- Routing is stable.

Locality

- IDs are distributed randomly through the underlying network.
- It is unlikely that $|L|/2$ nodes with consecutive IDs fail.
- Routing is stable.

Locality

- IDs are distributed randomly through the underlying network.
- It is unlikely that $|L|/2$ nodes with consecutive IDs fail.
- Routing is stable.

Locality in insertion

- The cost of routing operations depends on good choices for $R[i,j]$.

Fact

The algorithm for inserting peers generates good $R[i,j]$.

Locality in insertion

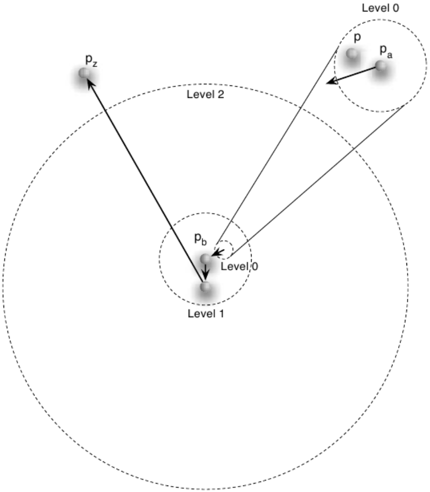
- The cost of routing operations depends on good choices for $R[i,j]$.

Fact

The algorithm for inserting peers generates good $R[i,j]$.

Overview

Locality in Routing



Locality in Routing

- $p_1, \ldots, p_a, p_b, p_c, \ldots, p_z$ with $p_b \in R_a$ and $p_c \in R_b$
- $d(p_a, p_b) < d(p_a, p_c)$
  Otherwise ..., p_a, p_c , ... would be used.
- p_i is taken from a set S with $|S| \approx n/2^{bi}$.
- The most expensive step is usually the last step in the leaf set.
- ... which can often be avoided depending on the protocol.

Locality in Routing

- $p_1, ..., p_a, p_b, p_c, ..., p_z$ with $p_b \in R_a$ and $p_c \in R_b$
- $d(p_a, p_b) < d(p_a, p_c)$
  Otherwise ..., p_a, p_c , ... would be used.
- p_i is taken from a set S with $|S| \approx n/2^{bi}$.
- The most expensive step is usually the last step in the leaf set.
- ... which can often be avoided depending on the protocol.

Locality in Routing

- $p_1, \ldots, p_a, p_b, p_c, \ldots, p_z$ with $p_b \in R_a$ and $p_c \in R_b$
- $d(p_a, p_b) < d(p_a, p_c)$
  Otherwise ..., p_a, p_c , ... would be used.
- p_i is taken from a set S with $|S| \approx n/2^{bi}$.
- The most expensive step is usually the last step in the leaf set.
- ... which can often be avoided depending on the protocol.

Locality in Routing

- $p_1, ..., p_a, p_b, p_c, ..., p_z$ with $p_b \in R_a$ and $p_c \in R_b$
- $d(p_a, p_b) < d(p_a, p_c)$
  Otherwise ..., p_a, p_c , ... would be used.
- p_i is taken from a set S with $|S| \approx n/2^{bi}$.
- The most expensive step is usually the last step in the leaf set.
- ... which can often be avoided depending on the protocol.

Locality in Routing

- $p_1, \ldots, p_a, p_b, p_c, \ldots, p_z$ with $p_b \in R_a$ and $p_c \in R_b$
- $d(p_a, p_b) < d(p_a, p_c)$
  Otherwise ..., p_a, p_c , ... would be used.
- p_i is taken from a set S with $|S| \approx n/2^{bi}$.
- The most expensive step is usually the last step in the leaf set.
- ... which can often be avoided depending on the protocol.

Locality in Routing

- $p_1, ..., p_a, p_b, p_c, ..., p_z$ with $p_b \in R_a$ and $p_c \in R_b$
- $d(p_a, p_b) < d(p_a, p_c)$
  Otherwise ..., p_a, p_c , ... would be used.
- p_i is taken from a set S with $|S| \approx n/2^{bi}$.
- The most expensive step is usually the last step in the leaf set.
- ... which can often be avoided depending on the protocol.

Overview

Dead nodes

- A peer may leave the network without warning.
- The underlying network supports pinging.
- The protocol atop pastry may include keep-alive messages.

Dead nodes

- A peer may leave the network without warning.
- The underlying network supports pinging.
- The protocol atop pastry may include keep-alive messages.

Dead nodes

- A peer may leave the network without warning.
- The underlying network supports pinging.
- The protocol atop pastry may include keep-alive messages.

Repairing the Leaf Set

- A peer p notices that a node $p' \in L_p$ is dead.
- p requests the leaf sets from other nodes in L_p.
- With them p can fill L_p and reconstruct $L_{p'}$.
- All nodes in $L_{p'}$ are notified by p.

Repairing the Leaf Set

- A peer p notices that a node $p' \in L_p$ is dead.
- p requests the leaf sets from other nodes in L_p.
- With them p can fill L_p and reconstruct $L_{p'}$.
- All nodes in $L_{p'}$ are notified by p.

Repairing the Leaf Set

- A peer p notices that a node $p' \in L_p$ is dead.
- p requests the leaf sets from other nodes in L_p.
- With them p can fill L_p and reconstruct $L_{p'}$.
- All nodes in $L_{p'}$ are notified by p.

Repairing the Leaf Set

- A peer p notices that a node $p' \in L_p$ is dead.
- p requests the leaf sets from other nodes in L_p.
- With them p can fill L_p and reconstruct $L_{p'}$.
- All nodes in $L_{p'}$ are notified by p.

Overview

Missing nodes after insertion

- Before a message is routed to a node p' it is checked if p' is alive.
- The algorithm for insertion does not gurantee that all nodes updated.
- Consider a network with no node with an ID with prefix 1.
    - If $id(p)$ starts with 1 all nodes will need to be updated.
    - However correct routing is still guarantied.

Missing nodes after insertion

- Before a message is routed to a node p' it is checked if p' is alive.
- The algorithm for insertion does not gurantee that all nodes updated.
- Consider a network with no node with an ID with prefix 1.
  - If $id(p)$ starts with 1 all nodes will need to be updated.
  - However correct routing is still guarantied.

Missing nodes after insertion

- Before a message is routed to a node p' it is checked if p' is alive.
- The algorithm for insertion does not gurantee that all nodes updated.
- Consider a network with no node with an ID with prefix 1.
  - If $id(p)$ starts with 1 all nodes will need to be updated.
  - However correct routing is still guarantied.

Missing nodes after insertion

- Before a message is routed to a node p' it is checked if p' is alive.
- The algorithm for insertion does not gurantee that all nodes updated.
- Consider a network with no node with an ID with prefix 1.
  - If $id(p)$ starts with 1 all nodes will need to be updated.
  - However correct routing is still guarantied.

Missing nodes after insertion

- Before a message is routed to a node p' it is checked if p' is alive.
- The algorithm for insertion does not gurantee that all nodes updated.
- Consider a network with no node with an ID with prefix 1.
  - If $id(p)$ starts with 1 all nodes will need to be updated.
  - However correct routing is still guarantied.

Repairing the Routing Table II

Method from FreePastry:

- A peer p gets message m routed from peer p'.

```
Let  l ← pflx(id(p), id(m))
if  pfxl(id(p'), id(m)) = l  and
     R[l, id(m)[l]] ≠ null  do
     Send our  R[l, *]  to  p'.
```

Repairing the Routing Table II

Method from FreePastry:

- A peer p gets message m routed from peer p'.

  Let $l \leftarrow pflx(id(p), id(m))$
  if $pfxl(id(p'), id(m)) = l$ and
  $R[l, id(m)[l]] \neq null$ do
  Send our $R[l, *]$ to p'.

Repairing the Routing Table II

Method from FreePastry:

- A peer p gets message m routed from peer p'.

  `Let` $l \leftarrow pflx(id(p), id(m))$
  `if` $pfxl(id(p'), id(m)) = l$ `and`
  $R[l, id(m)[l]] \neq null$ `do`
  `Send our` $R[l, *]$ `to` p'.

Methods against Malicious Nodes

What happens if there are nodes in the network that do not do what they are supposed to do?

- The algorithm discussed earlier is randomized.
- No single node stores all copies of a piece of data.
- Nodes send replies when they store data etc.
- Messages are signed.
  - $id(p)$ is the checksum of p's public key.

Methods against Malicious Nodes

What happens if there are nodes in the network that do not do what they are supposed to do?

- The algorithm discussed earlier is randomized.
- No single node stores all copies of a piece of data.
- Nodes send replies when they store data etc.
- Messages are signed.
    - $id(p)$ is the checksum of p's public key.

Methods against Malicious Nodes

What happens if there are nodes in the network that do not do what they are supposed to do?

- The algorithm discussed earlier is randomized.
- No single node stores all copies of a piece of data.
- Nodes send replies when they store data etc.
- Messages are signed.
    - $id(p)$ is the checksum of p's public key.

Methods against Malicious Nodes

What happens if there are nodes in the network that do not do what they are supposed to do?

- The algorithm discussed earlier is randomized.
- No single node stores all copies of a piece of data.
- Nodes send replies when they store data etc.
- Messages are signed.
    - $id(p)$ is the checksum of p's public key.

Methods against Malicious Nodes

What happens if there are nodes in the network that do not do what they are supposed to do?

- The algorithm discussed earlier is randomized.
- No single node stores all copies of a piece of data.
- Nodes send replies when they store data etc.
- Messages are signed.
  - $id(p)$ is the checksum of p's public key.

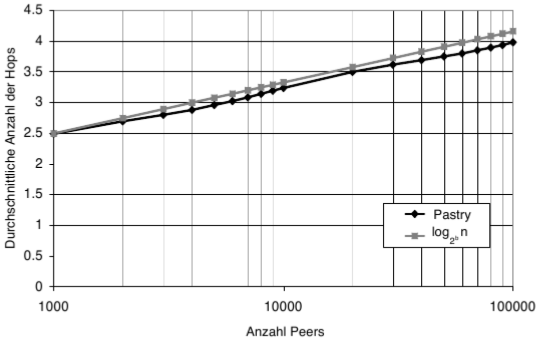
Methods against Malicious Nodes

What happens if there are nodes in the network that do not do what they are supposed to do?

- The algorithm discussed earlier is randomized.
- No single node stores all copies of a piece of data.
- Nodes send replies when they store data etc.
- Messages are signed.
  - $id(p)$ is the checksum of p's public key.

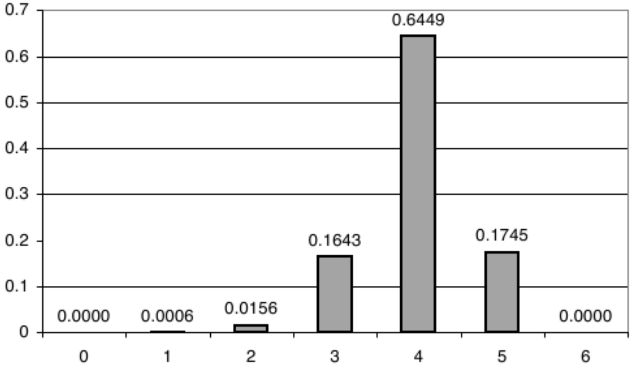
Overview

Number of Hops



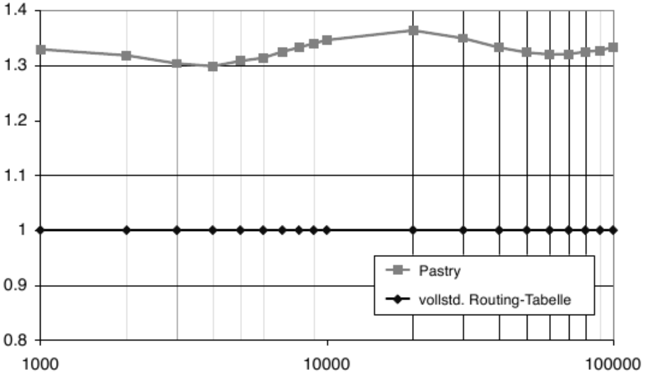
- Average number of hops for $b = 4$, $|L| = 16$ and $|M| = 32$.

Number of Hops



- Probability vs. Number of Hops.
- Again with $b = 4$, $|L| = 16$ and $|M| = 32$ and $n = 10^6$.

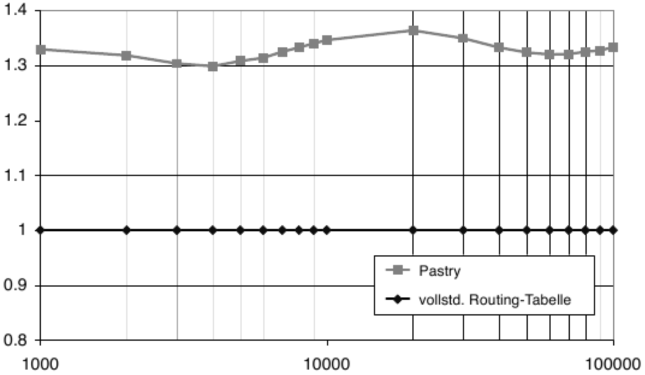
Distances in Routing



Relative Distance vs. Number of Peers

- Pastry is realtively good.
- Finding short routes scales well.

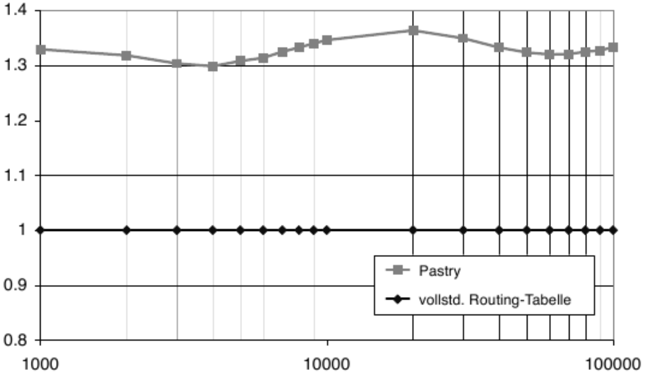
Distances in Routing



Relative Distance vs. Number of Peers

- Pastry is realtively good.
- Finding short routes scales well.

Distances in Routing



Relative Distance vs. Number of Peers

- Pastry is realtively good.
- Finding short routes scales well.

Number of Entries

- $b = 4$, $|L| = 16$ and $|M| = 32$ as before.
- Set n to 10^6
- $\forall p : |R_p| \gtrsim (2^b - 1) \log_{2^b} n > 60$
- $|R_p| + |L_p| + |M_p| \gtrsim 108$

- This is more than most other networks.

Number of Entries

- $b = 4$, $|L| = 16$ and $|M| = 32$ as before.
- Set n to 10^6
- $\forall p : |R_p| \gtrsim (2^b - 1)\log_{2^b} n > 60$
- $|R_p| + |L_p| + |M_p| \gtrsim 108$

- This is more than most other networks.

Overview

FreePastry

- Java-Implementation of Pastry.
- Simultation or TCP/IP.
- Can run Past or Scribe.

FreePastry

- Java-Implementation of Pastry.
- Simultation or TCP/IP.
- Can run Past or Scribe.

FreePastry

- Java-Implementation of Pastry.
- Simultation or TCP/IP.
- Can run Past or Scribe.

Other applications

Other applications build on Pastry include:

- a pubish/subscribe system. (SCRIBE)
- a caching system (SQUIRREL)
- a messaging infrastructure (POST)
- a high-bandwidth contend istribution system (SplitStream)
- ... and many more.

Other applications

Other applications build on Pastry include:

- a pubish/subscribe system. (SCRIBE)
- a caching system (SQUIRREL)
- a messaging infrastructure (POST)
- a high-bandwidth contend istribution system (SplitStream)
- ... and many more.

Other applications

Other applications build on Pastry include:

- a pubish/subscribe system. (SCRIBE)
- a caching system (SQUIRREL)
- a messaging infrastructure (POST)
- a high-bandwidth contend istribution system (SplitStream)
- ... and many more.

Other applications

Other applications build on Pastry include:

- a pubish/subscribe system. (SCRIBE)
- a caching system (SQUIRREL)
- a messaging infrastructure (POST)
- a high-bandwidth contend istribution system (SplitStream)
- ... and many more.

Other applications

Other applications build on Pastry include:

- a pubish/subscribe system. (SCRIBE)
- a caching system (SQUIRREL)
- a messaging infrastructure (POST)
- a high-bandwidth contend istribution system (SplitStream)
- ... and many more.

Other applications

Other applications build on Pastry include:

- a pubish/subscribe system. (SCRIBE)
- a caching system (SQUIRREL)
- a messaging infrastructure (POST)
- a high-bandwidth contend istribution system (SplitStream)
- ... and many more.

Overview

PAST

- A distributed file storage system.
- IDs of files are the checksum of the filename.
- Files are stored in the k nodes with nearest IDs to the ID of a file.
- Mechanisms for maintaining this invariant are added to the protocol.

PAST

- A distributed file storage system.
- IDs of files are the checksum of the filename.
- Files are stored in the k nodes with nearest IDs to the ID of a file.
- Mechanisms for maintaining this invariant are added to the protocol.

PAST

- A distributed file storage system.
- IDs of files are the checksum of the filename.
- Files are stored in the k nodes with nearest IDs to the ID of a file.
- Mechanisms for maintaining this invariant are added to the protocol.

PAST

- A distributed file storage system.
- IDs of files are the checksum of the filename.
- Files are stored in the k nodes with nearest IDs to the ID of a file.
- Mechanisms for maintaining this invariant are added to the protocol.

PAST

- Every node has a capacity.
  - If a node has a relatively high capacity it is split.
  - If a node has a relatively low capacity it is asked to leave the network.
  - Nodes may also join the network as observers.
- For optimisation the following techniques are used:
  - Replica diversion to balance among near nodes.
  - File diversion to avoid a imbalance between regions of the ID-space.
  - Caching

PAST

- Every node has a capacity.
  - If a node has a relatively high capacity it is split.
  - If a node has a relatively low capacity it is asked to leave the network.
  - Nodes may also join the network as observers.
- For optimisation the following techniques are used:
  - Replica diversion to balance among near nodes.
  - File diversion to avoid a imbalance between regions of the ID-space.
  - Caching

PAST

- Every node has a capacity.
  - If a node has a relatively high capacity it is split.
  - If a node has a relatively low capacity it is asked to leave the network.
  - Nodes may also join the network as observers.
- For optimisation the following techniques are used:
  - Replica diversion to balance among near nodes.
  - File diversion to avoid a imbalance between regions of the ID-space.
  - Caching

PAST

- Every node has a capacity.
  - If a node has a relatively high capacity it is split.
  - If a node has a relatively low capacity it is asked to leave the network.
  - Nodes may also join the network as observers.
- For optimisation the following techniques are used:
  - Replica diversion to balance among near nodes.
  - File diversion to avoid a imbalance between regions of the ID-space.
  - Caching

PAST

- Every node has a capacity.
  - If a node has a relatively high capacity it is split.
  - If a node has a relatively low capacity it is asked to leave the network.
  - Nodes may also join the network as observers.
- For optimisation the following techniques are used:
  - Replica diversion to balance among near nodes.
  - File diversion to avoid a imbalance between regions of the ID-space.
  - Caching

PAST

- Every node has a capacity.
  - If a node has a relatively high capacity it is split.
  - If a node has a relatively low capacity it is asked to leave the network.
  - Nodes may also join the network as observers.
- For optimisation the following techniques are used:
  - Replica diversion to balance among near nodes.
  - File diversion to avoid a imbalance between regions of the ID-space.
  - Caching

PAST

- Every node has a capacity.
  - If a node has a relatively high capacity it is split.
  - If a node has a relatively low capacity it is asked to leave the network.
  - Nodes may also join the network as observers.
- For optimisation the following techniques are used:
  - Replica diversion to balance among near nodes.
  - File diversion to avoid a imbalance between regions of the ID-space.
  - Caching

PAST

- Every node has a capacity.
  - If a node has a relatively high capacity it is split.
  - If a node has a relatively low capacity it is asked to leave the network.
  - Nodes may also join the network as observers.
- For optimisation the following techniques are used:
  - Replica diversion to balance among near nodes.
  - File diversion to avoid a imbalance between regions of the ID-space.
  - Caching

Overview

Scribe

- Distribute data to many nodes.
- There are topics.
- Each topic has an ID.
- One node is the root of the topic.
- Build trees using the routing procedure of pastry.
- Distribute messages from the root to the leafs of the tree.

Scribe

- Distribute data to many nodes.
- There are topics.
- Each topic has an ID.
- One node is the root of the topic.
- Build trees using the routing procedure of pastry.
- Distribute messages from the root to the leafs of the tree.

Scribe

- Distribute data to many nodes.
- There are topics.
- Each topic has an ID.
- One node is the root of the topic.
- Build trees using the routing procedure of pastry.
- Distribute messages from the root to the leafs of the tree.

Scribe

- Distribute data to many nodes.
- There are topics.
- Each topic has an ID.
- One node is the root of the topic.
- Build trees using the routing procedure of pastry.
- Distribute messages from the root to the leafs of the tree.

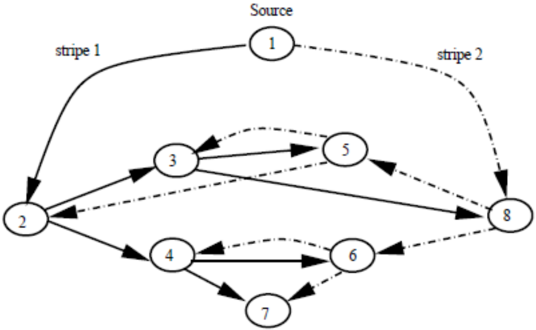
Scribe

- Distribute data to many nodes.
- There are topics.
- Each topic has an ID.
- One node is the root of the topic.
- Build trees using the routing procedure of pastry.
- Distribute messages from the root to the leafs of the tree.

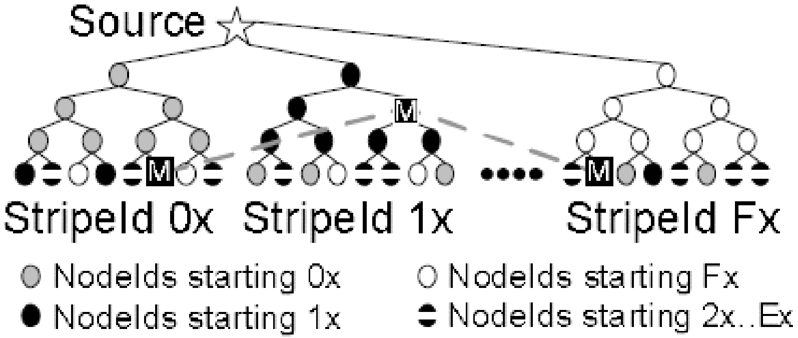
Scribe

- Distribute data to many nodes.
- There are topics.
- Each topic has an ID.
- One node is the root of the topic.
- Build trees using the routing procedure of pastry.
- Distribute messages from the root to the leafs of the tree.

SplitStream



SplitStream



Questions?

Overview

Algorithm for Routing

```
Search(r)
if (id(L[-|L|/2]) ≤ r ≤ id(L[|L|/2]))
    Route to peer p' ∈
    L, so that |r - id(p)'|is minimal.
    return;
c ← pfxl(r, id(p))
if (R[c, r[c]] ≠ null)
    Route to peer R[c, r[c]]
    return
Route to a p' ∈ R ∪ L ∪ M with
    pfxl(r, id(p')) ≥ c and
    |r - id(p')| < |r - id(p)|
```

Algorithm for Routing

```
Search(r)
if (id(L[-|L|/2]) ≤ r ≤ id(L[|L|/2]))
    Route to peer p' ∈
    L, so that |r - id(p)'|is minimal.
    return;
c ← pfxl(r, id(p))
if (R[c, r[c]] ≠ null)
    Route to peer R[c, r[c]]
    return
Route to a p' ∈ R ∪ L ∪ M with
    pfxl(r, id(p')) ≥ c and
    |r - id(p')| < |r - id(p)|
```

Algorithm for Routing

Search(r)
if $(id(L[-|L|/2]) \leq r \leq id(L[|L|/2]))$
    Route to peer $p' \in$
    L, so that $|r - id(p)'|$is minimal.
    return;
$c \leftarrow pfxl(r, id(p))$
if $(R[c, r[c]] \neq null)$
    Route to peer $R[c, r[c]]$
    return
Route to a $p' \in R \cup L \cup M$ with
    $pfxl(r, id(p')) \geq c$ and
    $|r - id(p')| < |r - id(p)|$

Algorithm for Routing

Search(r)
if ($id(L[-|L|/2]) \leq r \leq id(L[|L|/2])$)
    Route to peer $p' \in$
    L, so that $|r - id(p)'|$is minimal.
    return;
$c \leftarrow pfxl(r, id(p))$
if ($R[c, r[c]] \neq null$)
    Route to peer $R[c, r[c]]$
    return
Route to a $p' \in R \cup L \cup M$ with
    $pfxl(r, id(p')) \geq c$ and
    $|r - id(p')| < |r - id(p)|$

Algorithm for Routing

```
Search(r)
if (id(L[-|L|/2]) ≤ r ≤ id(L[|L|/2]))
    Route to peer p' ∈
    L, so that |r - id(p)'|is minimal.
    return;
c ← pfxl(r, id(p))
if (R[c, r[c]] ≠ null)
    Route to peer R[c, r[c]]
    return
Route to a p' ∈ R ∪ L ∪ M with
    pfxl(r, id(p')) ≥ c and
    |r - id(p')| < |r - id(p)|
```

Algorithm for Routing

```
Search(r)
if (id(L[−|L|/2]) ≤ r ≤ id(L[|L|/2]))
    Route to peer p′ ∈
    L, so that |r − id(p)′|is minimal.
    return;
c ← pfxl(r, id(p))
if (R[c, r[c]] ≠ null)
    Route to peer R[c, r[c]]
    return
Route to a p′ ∈ R ∪ L ∪ M with
    pfxl(r, id(p′)) ≥ c and
    |r − id(p′)| < |r − id(p)|
```

Algorithm for Routing

```
Search(r)
if (id(L[-|L|/2]) ≤ r ≤ id(L[|L|/2]))
    Route to peer p' ∈
    L, so that |r - id(p)'|is minimal.
    return;
c ← pfxl(r, id(p))
if (R[c, r[c]] ≠ null)
    Route to peer R[c, r[c]]
    return
Route to a p' ∈ R ∪ L ∪ M with
    pfxl(r, id(p')) ≥ c and
    |r - id(p')| < |r - id(p)|
```

Algorithm for Routing

```
Search(r)
if (id(L[-|L|/2]) ≤ r ≤ id(L[|L|/2]))
    Route to peer p' ∈
    L, so that |r - id(p)'|is minimal.
    return;
c ← pfxl(r, id(p))
if (R[c, r[c]] ≠ null)
    Route to peer R[c, r[c]]
    return
Route to a p' ∈ R ∪ L ∪ M with
    pfxl(r, id(p')) ≥ c and
    |r - id(p')| < |r - id(p)|
```

Size of R

Theorem

R_p for node p contains $\leq \mathcal{O}(\frac{\log n}{b} 2^b)$ entries.

Proof.

- $\forall m.\mathrm{P}(\exists q : pfxl(p,q) \geq m \wedge p \neq q) = (n-1) \cdot (2^{-b})^m$
- Let $m = (c+2)\frac{\log n}{b}$ for constant $c > 0$.
- $P(...) = ... \approx n^{-c-1} \underset{n \to \infty}{\longrightarrow} 0$
- It is probable that $a_{i,j} = null$ for $i > (c+2)\frac{\log n}{b}$.

$\square$

Size of R

Theorem

R_p for node p contains $\leq \mathcal{O}(\frac{\log n}{b} 2^b)$ entries.

Proof.

- $\forall m.\mathrm{P}(\exists q : pfxl(p,q) \geq m \wedge p \neq q) = (n-1) \cdot (2^{-b})^m$
- Let $m = (c+2)\frac{\log n}{b}$ for constant $c > 0$.
- $P(...) = ... \approx n^{-c-1} \xrightarrow[n \to \infty]{} 0$
- It is probable that $a_{i,j} = null$ for $i > (c+2)\frac{\log n}{b}$.

□

Size of R

Theorem

R_p for node p contains $\leq \mathcal{O}(\frac{\log n}{b} 2^b)$ entries.

Proof.

- $\forall m.\mathrm{P}(\exists q : pfxl(p, q) \geq m \wedge p \neq q) = (n-1) \cdot (2^{-b})^m$
- Let $m = (c+2)\frac{\log n}{b}$ for constant $c > 0$.
- $P(...) = ... \approx n^{-c-1} \xrightarrow[n \to \infty]{} 0$
- It is probable that $a_{i,j} = null$ for $i > (c+2)\frac{\log n}{b}$.

$\square$

Size of R

Theorem

R_p for node p contains $\leq \mathcal{O}(\frac{\log n}{b} 2^b)$ entries.

Proof.

- $\forall m.\mathrm{P}(\exists q : pfxl(p,q) \geq m \wedge p \neq q) = (n-1) \cdot (2^{-b})^m$
- Let $m = (c+2)\frac{\log n}{b}$ for constant $c > 0$.
- $P(...) = ... \approx n^{-c-1} \xrightarrow[n \to \infty]{} 0$
- It is probable that $a_{i,j} = null$ for $i > (c+2)\frac{\log n}{b}$.

□

Size of R

Theorem

R_p for node p contains $\leq \mathcal{O}(\frac{\log n}{b} 2^b)$ entries.

Proof.

- $\forall m.\mathrm{P}(\exists q : pfxl(p,q) \geq m \wedge p \neq q) = (n-1) \cdot (2^{-b})^m$
- Let $m = (c+2)\frac{\log n}{b}$ for constant $c > 0$.
- $P(...) = ... \approx n^{-c-1} \xrightarrow[n \to \infty]{} 0$
- It is probable that $a_{i,j} = null$ for $i > (c+2)\frac{\log n}{b}$.

$\square$

Correctness

- (Assume correct routing tables and leaf sets)

Theorem

Routing takes no more than $\mathcal{O}(n/|L|)$ steps.

Proof.

- We only use the leaf set.
- Let $p_0, p_1, ..., p_\nu$ be the nodes along the route.
- $\forall i \in \{0, ..., \nu - 1\} : p_{i+1} = L_{p_i}[\pm|L|/2]$
- $|id(L_{p_i}[\pm|L|/2]) - id(p_i)| \approx \frac{|ID|}{n} * \frac{|L|}{2}$
- $|id(p_0) - id(p_\nu)| \leq \frac{|ID|}{2}$

□

Correctness

- (Assume correct routing tables and leaf sets)

Theorem

Routing takes no more than $\mathcal{O}(n/|L|)$ steps.

Proof.

- We only use the leaf set.
- Let $p_0, p_1, \ldots, p_\nu$ be the nodes along the route.
- $\forall i \in \{0, \ldots, \nu - 1\} : p_{i+1} = L_{p_i}[\pm|L|/2]$
- $|id(L_{p_i}[\pm|L|/2]) - id(p_i)| \approx \frac{|ID|}{n} * \frac{|L|}{2}$
- $|id(p_0) - id(p_\nu)| \leq \frac{|ID|}{2}$

Correctness

- (Assume correct routing tables and leaf sets)

Theorem

Routing takes no more than $\mathcal{O}(n/|L|)$ steps.

Proof.

- We only use the leaf set.
- Let $p_0, p_1, \ldots, p_\nu$ be the nodes along the route.
- $\forall i \in \{0, \ldots, \nu - 1\} : p_{i+1} = L_{p_i}[\pm|L|/2]$
- $|id(L_{p_i}[\pm|L|/2]) - id(p_i)| \approx \frac{|ID|}{n} * \frac{|L|}{2}$
- $|id(p_0) - id(p_\nu)| \leq \frac{|ID|}{2}$

□

Correctness

- (Assume correct routing tables and leaf sets)

Theorem

Routing takes no more than $\mathcal{O}(n/|L|)$ steps.

Proof.

- We only use the leaf set.
- Let $p_0, p_1, \ldots, p_\nu$ be the nodes along the route.
- $\forall i \in \{0, \ldots, \nu - 1\} : p_{i+1} = L_{p_i}[\pm|L|/2]$
- $|id(L_{p_i}[\pm|L|/2]) - id(p_i)| \approx \frac{|ID|}{n} * \frac{|L|}{2}$
- $|id(p_0) - id(p_\nu)| \leq \frac{|ID|}{2}$

$\square$

Correctness

- (Assume correct routing tables and leaf sets)

Theorem

Routing takes no more than $\mathcal{O}(n/|L|)$ steps.

Proof.

- We only use the leaf set.
- Let $p_0, p_1, \ldots, p_\nu$ be the nodes along the route.
- $\forall i \in \{0, \ldots, \nu - 1\} : p_{i+1} = L_{p_i}[\pm|L|/2]$
- $|id(L_{p_i}[\pm|L|/2]) - id(p_i)| \approx \frac{|ID|}{n} * \frac{|L|}{2}$
- $|id(p_0) - id(p_\nu)| \leq \frac{|ID|}{2}$

$\square$

Correctness

- (Assume correct routing tables and leaf sets)

Theorem

Routing takes no more than $\mathcal{O}(n/|L|)$ steps.

Proof.

- We only use the leaf set.
- Let $p_0, p_1, ..., p_\nu$ be the nodes along the route.
- $\forall i \in \{0, ..., \nu - 1\} : p_{i+1} = L_{p_i}[\pm|L|/2]$
- $|id(L_{p_i}[\pm|L|/2]) - id(p_i)| \approx \frac{|ID|}{n} * \frac{|L|}{2}$
- $|id(p_0) - id(p_\nu)| \leq \frac{|ID|}{2}$

$\square$

Correctness

- (Assume correct routing tables and leaf sets)

Theorem

Routing takes no more than $\mathcal{O}(n/|L|)$ steps.

Proof.

- We only use the leaf set.
- Let $p_0, p_1, ..., p_\nu$ be the nodes along the route.
- $\forall i \in \{0, ..., \nu - 1\} : p_{i+1} = L_{p_i}[\pm|L|/2]$
- $|id(L_{p_i}[\pm|L|/2]) - id(p_i)| \approx \frac{|ID|}{n} * \frac{|L|}{2}$
- $|id(p_0) - id(p_\nu)| \leq \frac{|ID|}{2}$

□

Routing is fast.

Theorem

The excpected value is $\mathcal{O}(\log_{2^b} n) = \mathcal{O}(\frac{\log n}{b})$ *messages.*

Proof.

- We use the routing table.
- Each time a p_{i+1} is found in the routing table $pfxl(id(p_{i+1}), r) > pfxl(id(p_i), r)$
- There are only approx. $\mathcal{O}(\frac{\log n}{b})$ rows in each routing table.
- Now we use the leaf set.
- The probability that two (three) hops in L are needed is .02 (.0006).

Routing is fast.

Theorem

The excpected value is $\mathcal{O}(\log_{2^b} n) = \mathcal{O}(\frac{\log n}{b})$ *messages.*

Proof.

- We use the routing table.
- Each time a p_{i+1} is found in the routing table $pfxl(id(p_{i+1}), r) > pfxl(id(p_i), r)$
- There are only approx. $\mathcal{O}(\frac{\log n}{b})$ rows in each routing table.
- Now we use the leaf set.
- The probability that two (three) hops in L are needed is .02 (.0006).

□

Routing is fast.

Theorem

The excpected value is $\mathcal{O}(\log_{2^b} n) = \mathcal{O}(\frac{\log n}{b})$ *messages.*

Proof.

- We use the routing table.
- Each time a p_{i+1} is found in the routing table $pfxl(id(p_{i+1}), r) > pfxl(id(p_i), r)$
- There are only approx. $\mathcal{O}(\frac{\log n}{b})$ rows in each routing table.
- Now we use the leaf set.
- The probability that two (three) hops in L are needed is .02 (.0006).

□

Routing is fast.

Theorem

The excpected value is $\mathcal{O}(\log_{2^b} n) = \mathcal{O}(\frac{\log n}{b})$ *messages.*

Proof.

- We use the routing table.
- Each time a p_{i+1} is found in the routing table $pfxl(id(p_{i+1}), r) > pfxl(id(p_i), r)$
- There are only approx. $\mathcal{O}(\frac{\log n}{b})$ rows in each routing table.
- Now we use the leaf set.
- The probability that two (three) hops in L are needed is .02 (.0006).

□

Routing is fast.

Theorem

The excpected value is $\mathcal{O}(\log_{2^b} n) = \mathcal{O}(\frac{\log n}{b})$ *messages.*

Proof.

- We use the routing table.
- Each time a p_{i+1} is found in the routing table $pfxl(id(p_{i+1}), r) > pfxl(id(p_i), r)$
- There are only approx. $\mathcal{O}(\frac{\log n}{b})$ rows in each routing table.
- Now we use the leaf set.
- The probability that two (three) hops in L are needed is .02 (.0006).

□

Routing is fast.

Theorem

The excpected value is $\mathcal{O}(\log_{2^b} n) = \mathcal{O}(\frac{\log n}{b})$ *messages.*

Proof.

- We use the routing table.
- Each time a p_{i+1} is found in the routing table $pfxl(id(p_{i+1}), r) > pfxl(id(p_i), r)$
- There are only approx. $\mathcal{O}(\frac{\log n}{b})$ rows in each routing table.
- Now we use the leaf set.
- The probability that two (three) hops in L are needed is .02 (.0006).

□

Overview

Insertion of a peer

- Let p be a new peer with tables R, L and M.
- Try to choose values for the routing table so, that distances are minimal.

Algorithm:

- p contacts a peer p_0. Assume that p is near to p_0.
- p sends a *join*-Message with recepient $id(p)$.
- Every peer that gets the message sends its R, L and M.
- Let $p_0, p_1, \ldots, p_z$ be the path of the message.
- Assume $z \geq m$ and $pfxl(p, p_i) \geq i$ for $i < m - 1$ (Otherwise include nodes more than once.)

Insertion of a peer

- Let p be a new peer with tables R, L and M.
- Try to choose values for the routing table so, that distances are minimal.

Algorithm:

- p contacts a peer p_0. Assume that p is near to p_0.
- p sends a *join*-Message with recepient $id(p)$.
- Every peer that gets the message sends its R, L and M.
- Let $p_0, p_1, ..., p_z$ be the path of the message.
- Assume $z \geq m$ and $pfxl(p, p_i) \geq i$ for $i < m - 1$
(Otherwise include nodes more than once.)

Insertion of a peer

- Let p be a new peer with tables R, L and M.
- Try to choose values for the routing table so, that distances are minimal.

Algorithm:

- p contacts a peer p_0. Assume that p is near to p_0.
- p sends a *join*-Message with recepient $id(p)$.
- Every peer that gets the message sends its R, L and M.
- Let $p_0, p_1, ..., p_z$ be the path of the message.
- Assume $z \geq m$ and $pfxl(p, p_i) \geq i$ for $i < m - 1$
  (Otherwise include nodes more than once.)

Insertion of a peer

- Let p be a new peer with tables R, L and M.
- Try to choose values for the routing table so, that distances are minimal.

Algorithm:

- p contacts a peer p_0. Assume that p is near to p_0.
- p sends a *join*-Message with recepient $id(p)$.
- Every peer that gets the message sends its R, L and M.
- Let $p_0, p_1, ..., p_z$ be the path of the message.
- Assume $z \geq m$ and $pfxl(p, p_i) \geq i$ for $i < m - 1$ (Otherwise include nodes more than once.)

Insertion of a peer

- Let p be a new peer with tables R, L and M.
- Try to choose values for the routing table so, that distances are minimal.

Algorithm:

- p contacts a peer p_0. Assume that p is near to p_0.
- p sends a *join*-Message with recepient $id(p)$.
- Every peer that gets the message sends its R, L and M.
- Let $p_0, p_1, \ldots, p_z$ be the path of the message.
- Assume $z \geq m$ and $pfxl(p, p_i) \geq i$ for $i < m - 1$ (Otherwise include nodes more than once.)

Insertion of a peer

- Let p be a new peer with tables R, L and M.
- Try to choose values for the routing table so, that distances are minimal.

Algorithm:

- p contacts a peer p_0. Assume that p is near to p_0.
- p sends a *join*-Message with recepient $id(p)$.
- Every peer that gets the message sends its R, L and M.
- Let $p_0, p_1, ..., p_z$ be the path of the message.
- Assume $z \geq m$ and $pfxl(p, p_i) \geq i$ for $i < m - 1$
  (Otherwise include nodes more than once.)

Insertion of a peer

- Let p be a new peer with tables R, L and M.
- Try to choose values for the routing table so, that distances are minimal.

Algorithm:

- p contacts a peer p_0. Assume that p is near to p_0.
- p sends a *join*-Message with recepient $id(p)$.
- Every peer that gets the message sends its R, L and M.
- Let $p_0, p_1, ..., p_z$ be the path of the message.
- Assume $z \geq m$ and $pfxl(p, p_i) \geq i$ for $i < m - 1$
  (Otherwise include nodes more than once.)

Insertion of a peer (cont.)

- Let $M = M_0$.
- $|id(p) - id(p_z)|$ is minimal.
- Use L_z for L and insert p_z at position 1 or -1.
- Let $R[0,*] = R_0[0,*]$; $R[1,*] = R_1[1,*]$; ... ;
  - $pfxl(p, p_i) \geq i \Rightarrow \forall q \in R_i[i,*].pfxl(q,p) \geq i$
- Request R_q from all $q \in M$ and look for better entries for R.
- Notify every peer in M, L and R about our arrival.

Insertion of a peer (cont.)

- Let $M = M_0$.
- $|id(p) - id(p_z)|$ is minimal.
- Use L_z for L and insert p_z at position 1 or -1.
- Let $R[0, *] = R_0[0, *]$; $R[1, *] = R_1[1, *]$; ... ;
  - $pfxl(p, p_i) \geq i \Rightarrow \forall q \in R_i[i, *].pfxl(q, p) \geq i$
- Request R_q from all $q \in M$ and look for better entries for R.
- Notify every peer in M, L and R about our arrival.

Insertion of a peer (cont.)

- Let $M = M_0$.
- $|id(p) - id(p_z)|$ is minimal.
- Use L_z for L and insert p_z at position 1 or -1.
- Let $R[0,*] = R_0[0,*]$; $R[1,*] = R_1[1,*]$; ... ;
  - $pfxl(p, p_i) \geq i \Rightarrow \forall q \in R_i[i,*].pfxl(q,p) \geq i$
- Request R_q from all $q \in M$ and look for better entries for R.
- Notify every peer in M, L and R about our arrival.

Insertion of a peer (cont.)

- Let $M = M_0$.
- $|id(p) - id(p_z)|$ is minimal.
- Use L_z for L and insert p_z at position 1 or -1.
- Let $R[0,*] = R_0[0,*]$; $R[1,*] = R_1[1,*]$; ... ;
  - $pfxl(p, p_i) \geq i \Rightarrow \forall q \in R_i[i,*].pfxl(q,p) \geq i$
- Request R_q from all $q \in M$ and look for better entries for R.
- Notify every peer in M, L and R about our arrival.

Insertion of a peer (cont.)

- Let $M = M_0$.
- $|id(p) - id(p_z)|$ is minimal.
- Use L_z for L and insert p_z at position 1 or -1.
- Let $R[0,*] = R_0[0,*]$; $R[1,*] = R_1[1,*]$; ... ;
  - $pfxl(p, p_i) \geq i \Rightarrow \forall q \in R_i[i,*].pfxl(q,p) \geq i$
- Request R_q from all $q \in M$ and look for better entries for R.
- Notify every peer in M, L and R about our arrival.

Insertion of a peer (cont.)

- Let $M = M_0$.
- $|id(p) - id(p_z)|$ is minimal.
- Use L_z for L and insert p_z at position 1 or -1.
- Let $R[0,*] = R_0[0,*]$; $R[1,*] = R_1[1,*]$; ... ;
  - $pfxl(p, p_i) \geq i \Rightarrow \forall q \in R_i[i,*].pfxl(q,p) \geq i$
- Request R_q from all $q \in M$ and look for better entries for R.
- Notify every peer in M, L and R about our arrival.

Insertion of a peer (cont.)

- Let $M = M_0$.
- $|id(p) - id(p_z)|$ is minimal.
- Use L_z for L and insert p_z at position 1 or -1.
- Let $R[0,*] = R_0[0,*]$; $R[1,*] = R_1[1,*]$; ... ;
  - $pfxl(p, p_i) \geq i \Rightarrow \forall q \in R_i[i,*].pfxl(q,p) \geq i$
- Request R_q from all $q \in M$ and look for better entries for R.
- Notify every peer in M, L and R about our arrival.

Locality in Insertion

Fact

The algorithm for inserting peers generates good $R[i, j]$.

Locality in Insertion

Proof.

(Outline)

- Assume that the routing tables are optimized.
- $R[0,*]$ is taken from p_0.
- $\forall q \in R[0,*].\ d(p_0,q)$ small $\Longrightarrow d(p,q)$ small.
- $R[1,*]$ is taken from p_1.
- $\forall q \in R[1,*] : d(p_1,q)$ is good, $d(p,p_1)$ is good.
- The following is needed: $q \in \{s | pflx(s,p) \geq 1\}$
- Thus $\forall q \in R[1,*] : d(p,q)$ is relatively small.
- And so on for all $R[i,*]$

$\square$

Locality in Insertion

Proof.

(Outline)

- Assume that the routing tables are optimized.
- $R[0, *]$ is taken from p_0.
- $\forall q \in R[0, *]$. $d(p_0, q)$ small $\implies d(p, q)$ small.
- $R[1, *]$ is taken from p_1.
- $\forall q \in R[1, *] : d(p_1, q)$ is good, $d(p, p_1)$ is good.
- The following is needed: $q \in \{s | pflx(s, p) \geq 1\}$
- Thus $\forall q \in R[1, *] : d(p, q)$ is relatively small.
- And so on for all $R[i, *]$

□

Locality in Insertion

Proof.

(Outline)

- Assume that the routing tables are optimized.
- $R[0, *]$ is taken from p_0.
- $\forall q \in R[0, *].\ d(p_0, q)$ small $\implies d(p, q)$ small.
- $R[1, *]$ is taken from p_1.
- $\forall q \in R[1, *] : d(p_1, q)$ is good, $d(p, p_1)$ is good.
- The following is needed: $q \in \{s | pflx(s, p) \geq 1\}$
- Thus $\forall q \in R[1, *] : d(p, q)$ is relatively small.
- And so on for all $R[i, *]$

□

Locality in Insertion

Proof.

(Outline)

- Assume that the routing tables are optimized.
- $R[0, *]$ is taken from p_0.
- $\forall q \in R[0, *]$. $d(p_0, q)$ small $\Longrightarrow d(p, q)$ small.
- $R[1, *]$ is taken from p_1.
- $\forall q \in R[1, *] : d(p_1, q)$ is good, $d(p, p_1)$ is good.
- The following is needed: $q \in \{s | pflx(s, p) \geq 1\}$
- Thus $\forall q \in R[1, *] : d(p, q)$ is relatively small.
- And so on for all $R[i, *]$

$\square$

Locality in Insertion

Proof.

(Outline)

- Assume that the routing tables are optimized.
- $R[0,*]$ is taken from p_0.
- $\forall q \in R[0,*].\ d(p_0, q)$ small $\implies d(p,q)$ small.
- $R[1,*]$ is taken from p_1.
- $\forall q \in R[1,*] : d(p_1, q)$ is good, $d(p, p_1)$ is good.
- The following is needed: $q \in \{s | pflx(s,p) \geq 1\}$
- Thus $\forall q \in R[1,*] : d(p,q)$ is relatively small.
- And so on for all $R[i,*]$

$\square$

Locality in Insertion

Proof.

(Outline)

- Assume that the routing tables are optimized.
- $R[0,*]$ is taken from p_0.
- $\forall q \in R[0,*].\ d(p_0,q)$ small $\Longrightarrow d(p,q)$ small.
- $R[1,*]$ is taken from p_1.
- $\forall q \in R[1,*] : d(p_1,q)$ is good, $d(p,p_1)$ is good.
- The following is needed: $q \in \{s | pflx(s,p) \geq 1\}$
- Thus $\forall q \in R[1,*] : d(p,q)$ is relatively small.
- And so on for all $R[i,*]$

$\square$

Locality in Insertion

Proof.

(Outline)

- Assume that the routing tables are optimized.
- $R[0,*]$ is taken from p_0.
- $\forall q \in R[0,*].\ d(p_0, q)$ small $\Longrightarrow d(p,q)$ small.
- $R[1,*]$ is taken from p_1.
- $\forall q \in R[1,*] : d(p_1, q)$ is good, $d(p, p_1)$ is good.
- The following is needed: $q \in \{s | pflx(s,p) \geq 1\}$
- Thus $\forall q \in R[1,*] : d(p,q)$ is relatively small.
- And so on for all $R[i,*]$

□

Locality in Insertion

Proof.

(Outline)

- Assume that the routing tables are optimized.
- $R[0,*]$ is taken from p_0.
- $\forall q \in R[0,*].\ d(p_0,q)$ small $\Longrightarrow d(p,q)$ small.
- $R[1,*]$ is taken from p_1.
- $\forall q \in R[1,*] : d(p_1,q)$ is good, $d(p,p_1)$ is good.
- The following is needed: $q \in \{s | pflx(s,p) \geq 1\}$
- Thus $\forall q \in R[1,*] : d(p,q)$ is relatively small.
- And so on for all $R[i,*]$

$\square$

Locality in Insertion

Proof.

(Outline)

- Assume that the routing tables are optimized.
- $R[0, *]$ is taken from p_0.
- $\forall q \in R[0, *]. \ d(p_0, q)$ small $\Longrightarrow d(p, q)$ small.
- $R[1, *]$ is taken from p_1.
- $\forall q \in R[1, *] : d(p_1, q)$ is good, $d(p, p_1)$ is good.
- The following is needed: $q \in \{s | pflx(s, p) \geq 1\}$
- Thus $\forall q \in R[1, *] : d(p, q)$ is relatively small.
- And so on for all $R[i, *]$

$\square$

Overview

Repairing the Routing Table

- How are missing entries in R filled?
- Algorithm:
  - A peer notices that it is missing an entry $R[i, j]$.
  - It asks all other $q \in R[i, *]$ for their entry $R_q[i, j]$
  - If this does not succeed it tries its next row $R[i + 1, *]$, ...

Repairing the Routing Table

- How are missing entries in R filled?
- Algorithm:
  - A peer notices that it is missing an entry $R[i, j]$.
  - It asks all other $q \in R[i, *]$ for their entry $R_q[i, j]$
  - If this does not succeed it tries its next row $R[i + 1, *]$, ...

Repairing the Routing Table

- How are missing entries in R filled?
- Algorithm:
  - A peer notices that it is missing an entry $R[i, j]$.
  - It asks all other $q \in R[i, *]$ for their entry $R_q[i, j]$
  - If this does not succeed it tries its next row $R[i + 1, *]$, ...

Repairing the Routing Table

- How are missing entries in R filled?
- Algorithm:
  - A peer notices that it is missing an entry $R[i, j]$.
  - It asks all other $q \in R[i, *]$ for their entry $R_q[i, j]$
  - If this does not succeed it tries its next row $R[i + 1, *]$, ...

Overview

1. Data Structures
   - Routing Table
   - Leaf Set
   - Neighbourhood Set
2. Operations
   - Routing
   - Insertion of peers
   - Locality
   - Locality in Routing
3. Stability
   - Leaf Set
   - Routing Table
   - Experimental Results
4. Conclusion and Outlook
   - FreePastry
   - PAST
   - Scribe and SplitStream

Malicious Nodes

- There may be faulty implemtations of pastry.
- There may be nodes that try to interfer with the network.

Malicious Nodes

- There may be faulty implemtations of pastry.
- There may be nodes that try to interfer with the network.

Assumtions

- Routing in pastry is deterministic.
- Invariant: In routing through $\dots p_i, p_{i+1}, \dots$ to ID r:
  - $pfxl(id(p_{i+1}), r) > pfxl(id(p_i), r)$ or at least
  - $|id(p_{i+1}) - r| > |id(p_i) - r|$
- Assumption: Most nodes in the network are working properly

Assumtions

- Routing in pastry is deterministic.
- Invariant: In routing through $...p_i, p_{i+1}, ...$ to ID r:
  - $pfxl(id(p_{i+1}), r) > pfxl(id(p_i), r)$ or at least
  - $|id(p_{i+1}) - r| > |id(p_i) - r|$
- Assumption: Most nodes in the network are working properly

Assumtions

- Routing in pastry is deterministic.
- Invariant: In routing through ...$p_i, p_{i+1}, \ldots$ to ID r:
  - $pfxl(id(p_{i+1}), r) > pfxl(id(p_i), r)$ or at least
  - $|id(p_{i+1}) - r| > |id(p_i) - r|$
- Assumption: Most nodes in the network are working properly

Assumtions

- Routing in pastry is deterministic.
- Invariant: In routing through ...p_i, p_{i+1}, ... to ID r:
  - $pfxl(id(p_{i+1}), r) > pfxl(id(p_i), r)$ or at least
  - $|id(p_{i+1}) - r| > |id(p_i) - r|$
- Assumption: Most nodes in the network are working properly

Assumtions

- Routing in pastry is deterministic.
- Invariant: In routing through ...$p_i, p_{i+1}, \ldots$ to ID r:
  - $pfxl(id(p_{i+1}), r) > pfxl(id(p_i), r)$ or at least
  - $|id(p_{i+1}) - r| > |id(p_i) - r|$
- Assumption: Most nodes in the network are working properly

Methods against Malicious Nodes

- The algorithm discussed earlier is randomized.
  - The invariant is maintaied.
  - A strong bias towards using R is applied.
- No single node stores all copies of a piece of data.
- Nodes send replies when they store data etc.
- Messages are signed.
  - $id(p)$ is the checksum of p's public key.

Methods against Malicious Nodes

- The algorithm discussed earlier is randomized.
  - The invariant is maintaied.
  - A strong bias towards using R is applied.
- No single node stores all copies of a piece of data.
- Nodes send replies when they store data etc.
- Messages are signed.
  - $id(p)$ is the checksum of p's public key.

Methods against Malicious Nodes

- The algorithm discussed earlier is randomized.
  - The invariant is maintaied.
  - A strong bias towards using R is applied.
- No single node stores all copies of a piece of data.
- Nodes send replies when they store data etc.
- Messages are signed.
  - $id(p)$ is the checksum of p's public key.

Methods against Malicious Nodes

- The algorithm discussed earlier is randomized.
  - The invariant is maintaied.
  - A strong bias towards using R is applied.
- No single node stores all copies of a piece of data.
- Nodes send replies when they store data etc.
- Messages are signed.
  - $id(p)$ is the checksum of p's public key.

Methods against Malicious Nodes

- The algorithm discussed earlier is randomized.
    - The invariant is maintaied.
    - A strong bias towards using R is applied.
- No single node stores all copies of a piece of data.
- Nodes send replies when they store data etc.
- Messages are signed.
    - $id(p)$ is the checksum of p's public key.

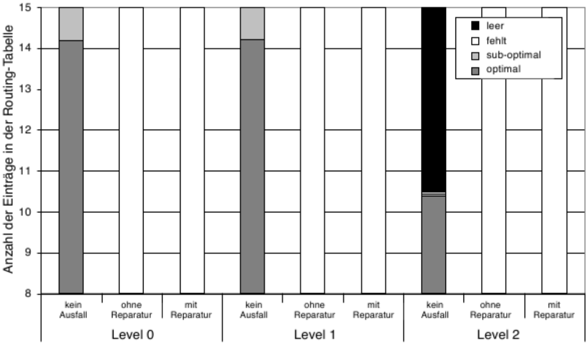
Methods against Malicious Nodes

- The algorithm discussed earlier is randomized.
  - The invariant is maintaied.
  - A strong bias towards using R is applied.
- No single node stores all copies of a piece of data.
- Nodes send replies when they store data etc.
- Messages are signed.
  - $id(p)$ is the checksum of p's public key.

Methods against Malicious Nodes

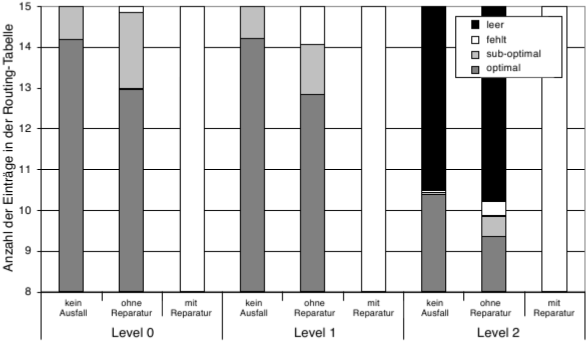
- The algorithm discussed earlier is randomized.
  - The invariant is maintaied.
  - A strong bias towards using R is applied.
- No single node stores all copies of a piece of data.
- Nodes send replies when they store data etc.
- Messages are signed.
  - $id(p)$ is the checksum of p's public key.

Robustness



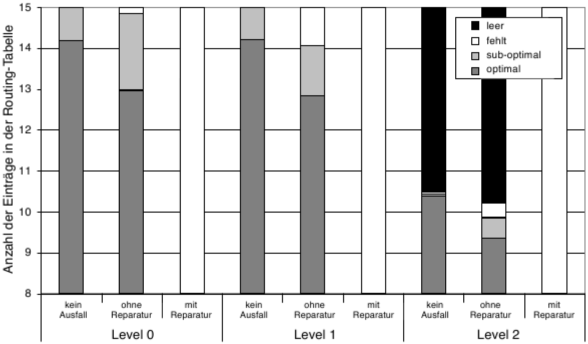
- Remove 10% of all peers from the network.
- Then repair:
  - Randomly choose two peers p and q and an ID d.
  - Send a message from p and from q to d.
  - Repeat 100.000 times.

Robustness



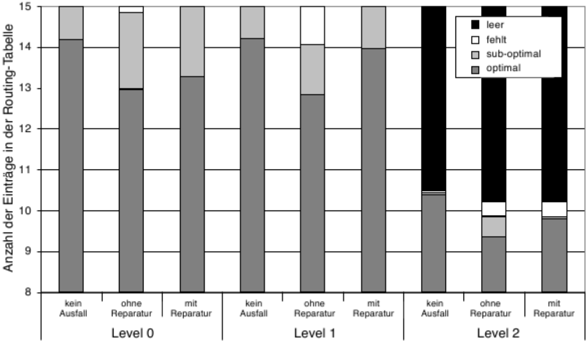
- Remove 10% of all peers from the network.
- Then repair:
  - Randomly choose two peers p and q and an ID d.
  - Send a message from p and from q to d.
  - Repeat 100.000 times.

Robustness



- Remove 10% of all peers from the network.
- Then repair:
  - Randomly choose two peers p and q and an ID d.
  - Send a message from p and from q to d.
  - Repeat 100.000 times.

Robustness



- Remove 10% of all peers from the network.
- Then repair:
  - Randomly choose two peers p and q and an ID d.
  - Send a message from p and from q to d.
  - Repeat 100.000 times.