A GLOBALLY CONNECTED OVERLAY FOR VIRTUAL RING ROUTING

REAL

Abstract. We introduce an overlay structure based on the Chord DHT. Given a connected network of nodes, each having a limited local view of the network, we provide a distributed asynchronous algorithm that creates a ring like overlay network. The distributed algorithm ensures that all the nodes end up on a single cycle. In addition, the cycle's structure allows efficient greedy routing of messages between nodes in the network.

1. Intro

Consider a connected network of n computers. Some pairs of computers have a direct connection between them. We want to be able to pass messages between any two computers on the network using the existing direct connections.

Assuming that a message originates from some computer, at each stage the current computer passes the message to one of his neighbors. Eventually the message should arrive at its destination computer. We also assume that every computer on the network has only limited amount of memory, and hence can not comprehend the full structure of the network.

We can think of the network structure as a simple graph, where vertices represent computers in the network. Two vertices in the graph have an edge between them if the two corresponding computers are directly connected.

In this document we propose a distributed algorithm for routing in connected networks which is based on Chord's overlay structure [1]. Our design builds upon the work in [2, 3].

Our main contribution is a new design to the overlay structure, and an analysis of the resulting overlay connectivity. We prove that our new design always results in a single connected overlay ring structure.

2. Initial setup

We assume that every node (computer) in the network has a unique ID that is picked from a large set $S_l = [0, 2^l)$. In the rest of this paper we use the unique IDs of node to refer to them.

Definition 2.1. Given two nodes $x, y \in S_l$, the virtual distance between x and y is:

$$d(x,y) = \begin{cases} y - x & \text{if } y \geq x \\ 2^l + y - x & \text{otherwise} \end{cases}$$

Date: 30.04.2017.

2010 *Mathematics Subject Classification.* Primary 05C30.

Freedomlayer research facility.

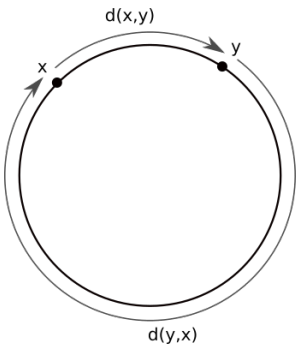

FIGURE 1. The virtual distance from x to y, and from y to x.

In other words, $dist(x, y)$ is the clockwise distance from x to y.

Note that generally $d(x, y) \neq d(y, x)$. $d(x, y) = d(y, x)$ if and only if $x = y$. In addition, $d(x, y) + d(y, x) = 2^l$. Also note that $d(x, y) \equiv y - x \pmod{2^l}$.

Lemma 2.2. $d(x, y) + d(y, z) \equiv d(x, z) \pmod{2^l}$

Proof. $d(x, y) + d(y, z) \equiv (y - x) + (z - y) \equiv z - x \equiv d(x, z) \pmod{2^l}$

□

We will use the network nodes' IDs to create a virtual overlay structure to help us route messages in the network. This overlay structure is computed by all the nodes of the network together, and every node contains a small part of the computation result.

3. IterF-1 Algorithm

We begin with a weak construction called the **IterF-1** algorithm. Analyzing this simpler version of the algorithm will help us to prove a few properties of that apply to the more general version of the algorithm.

During the execution of this algorithm every node x maintains the following information:

(1) x^l: A node y that minimizes $d(y, x-1)$, of all the nodes y that x knows of.
(2) x^r: A node z that minimizes $d(x+1, z)$, of all the nodes z that x knows of.

x will also keep the shortest path he has found to x^l and x^r, where a path between two nodes a, b in the network is a series of nodes $v_1 = a, v_2, \ldots, v_k = b$, And the nodes v_i, v_{i+1} are directly connected for $1 \leq i \leq k$.

A path to a newly discovered node is calculated by concatenation of paths. As an example, if node x knows a path to node y, and y tells x about a node z, x will concatenate the path he has to y: $(x, \ldots, y)$, together with the path y has sent him $(y, \ldots, z)$, which will result in the path $(x, \ldots, y, \ldots, z)$.

For brevity we omit the details of path keeping and concatenation from the pseudo code of the IterF-1 Algorithm 1.

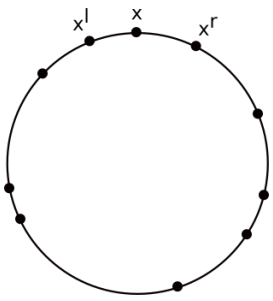

FIGURE 2. x^l and x^r

Algorithm 1 IterF-1 algorithm

```
1: procedure UpdateFingers(candidates)
2:     x^l ← arg min_{y∈candidates} d(y, x − 1)
3:     x^r ← arg min_{z∈candidates} d(x + 1, z)
4: end procedure
5: procedure GetFingers
6:     return {x^l, x^r}
7: end procedure
8: procedure GetKnown
9:     return (Neighbors(x) ∪ GetFingers()) \ {x}
10: end procedure
11: procedure SendUpdateRequests
12:     for each z ∈ GetKnown() do
13:         send UpdateRequest(GetKnown()) to z
14:     end for
15: end procedure
16: procedure ProcessUpdate(z, known_z)
17:     UpdateFingers(known_z ∪ { z, x } ∪ GetKnown())
18: end procedure
---
19: procedure Initialize
20:     UpdateFingers(Neighbors(x) ∪ {x})
21: end procedure
22: procedure onIncomingUpdateRequest(z, known_z)
23:     ProcessUpdate(z, known_z)
24:     send UpdateResponse(GetKnown()) to z
25: end procedure
26: procedure onIncomingUpdateResponse(z, known_z)
27:     ProcessUpdate(z, known_z)
28: end procedure
29: procedure PeriodicTick
30:     SendUpdateRequests()
31: end procedure
```

3.1. **Summary of IterF-1.** IterF-1 works by having each node keep his best candidates for a node from the left and a node from the right. By exchanging their best left and right candidates, nodes are able to improve their candidates over time.

We explore the pseudo code: The procedure Initialize is called on the initialization of the node x. The procedure PeriodicTick is called every period of time, for example, every few seconds[1]. The procedures onIncomingUpdateRequest and onIncomingUpdateResponse are called upon the receipt of UpdateRequest and UpdateResponse messages.

Upon initialization, the node x picks x^l and x^r to be the most suitable nodes from the set of nodes he knows: all of his neighbors.

Upon a periodic tick, the node x sends his set of known nodes: x^l, x^r and his neighbors, to all the nodes that he knows about: x^l, x^r and his neighbors. This is done by sending UpdateRequest messages to all those nodes.

Every node y that receives such an UpdateRequest message first updates his choices of y^l and y^r, in case better options were found. Then y responds to x with an UpdateResponse message that contains all the nodes known to y: y^l, y^r and y's neighbors. x will receive this UpdateResponse message from y, and possibly update his x^l and x^r if better options were found.

3.2. **Analysis of IterF-1.** Assuming that the set of nodes is $V \subseteq S_l$, we define the state of the algorithm to be

$$K = \{(x, x^l, x^r) | x \in V\}$$

This is the set of choices of left and right node for all the nodes in the network. We say that a state K is **stationary** if K will never change for any order of periodic ticks and message arrivals.

Lemma 3.1 (IterF-1 Convergence)**.** *Every run of IterF-1 arrives at a stationary state after a finite amount of changes to K.*

Proof. For a state K of the algorithm we define:

$$e_1(K) := \sum_{x \in V} d(x+1, x^r) + d(x^l, x-1)$$

e_1 is the energy function of the current state of the algorithm. Note that K changes whenever a node y updates y^l or y^r to a better node. In every such case $e_1(K)$ decreases.

As $e_1 \in \mathbb{N}$, it can decrease only a finite amount of times. Hence K can change only a finite amount of times, after which the algorithm will reach a stationary state. □

Lemma 3.2. *Assume a stationary state of IterF-1 over a graph G. Then for every node $x \in V$: $x^{rl} = x$ and $x^{lr} = x$. (Where x^{rl} is a short notation for $(x^r)^l$) Every run of IterF-1 arrives at a stationary state after a finite amount of changes to K.*

Proof. Let x be a node in V. Consider x^r and x^{rl}. If $x = x^r$ this means that x has no neighbors, and therefore $x = x^r = x^l$, and we get $x^{rl} = x$ trivially. Similarly, If $x^r = x^{rl}$ then x^r has no neighbors, which means $x = x^r$. Hence x has no neighbors,

[1]The only property that we require from the periodic ticking is that a tick happens eventually. The algorithm works properly even if different nodes have different tick frequency, and even if a node has a changing ticking frequency

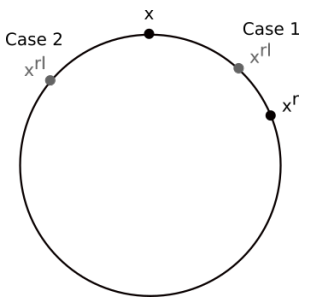

FIGURE 3. The two cases for x^{rl}

and again we obtain trivially that $x^{rl} = x$. Therefore in the rest of this proof we will assume that $x \neq x^r$ and $x^r \neq x^{rl}$.

Suppose that $x \neq x^{rl}$. Then we distinct between two cases (See Figure 3).

Case 1: $x^{rl} \in (x, x^r)$. (This means that x^{rl} is on the arc between x and x^r, clockwise). In the tick of x, x will send x^r an UpdateRequest message. x^r will then send to back to x an UpdateResponse message, which will contain a path to x^{rl}. x will then update x^r to be x^{rl}. This is a contradiction to the given stationary state.

Case 2: $x^{rl} \in (x^r, x)$. In the next tick of x, x will send x^r an UpdateRequest message. Upon receiving this message, x^r will update x^{rl} to be x. This is a contradiction to the given stationary state.

Therefore we conclude that $x = x^{rl}$. In a symmetric way it can be shown that $x^{lr} = x$. □

Lemma 3.3. *Assume a stationary state of IterF-1 over a graph. Then the function $h(x) := x^r$ (Or dually, $t(x) := x^l$) is injective (one to one).*

Proof. Let x, y nodes such that $h(x) = h(y)$. By definition this means that $x^r = y^r$. Therefore $(x^r)^l = (y^r)^l$. By Lemma 3.2 we get that $x = (x^r)^l = (y^r)^l = y$. □

3.3. **Division to h-cycles.** We now begin from a node x, and apply the function h over x multiple times. We get the nodes $x, h(x), h(h(x)), \ldots$. There is a finite amount of nodes, so we expect that eventually some element $h^k(x)$ (h applied k times over x will be equal to some other element $h^q(x), q < k$ that was already on the series. Assume that k is the smallest such integer.

But since h is injective, $h^q(x) = h^k(x)$ means that $x = h^{k-q}(x)$. We assumed earlier that k is the index of the first element in our series that equals to a previously seen element, therefore we conclude that $k - q = k$, or $q = 0$. Hence $h^k(x) = x$, the first element of the series. We get that the series $x, h(x), \ldots, h^k(x) = x$ form a cycle. We can such a cycle an h-cycle.

If we now pick some node z that has not appeared in the cycle of the node x, and start applying h many times over z, we will get another cycle. We can continue in this fashion to divide all the nodes to different cycles.[2]

We have special interest in the amount of resulting h cycles after running our algorithm over a given connected graph. More specifically, our method of routing relies on the requirement that all nodes end up on the same cycle.

[2]This property of division to cycles applies to any injective function that operates on a finite set.

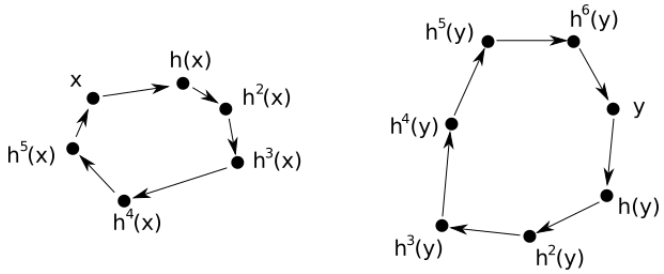

FIGURE 4. The h-cycles formed by an injective function h operating on a finite set

```
7 - 5 - 3     8 -10 -12
|       |     |       |
9 -11 - 1 --- 6 - 4 - 2
```

FIGURE 5. A connected graph that results in multiple h cycles after running IterF-1

Unfortunately, IterF-1 does not supply this guarantee. It is possible to run IterF-1 over a connected graph and end up with multiple disjoint h cycles. Figure 5 shows an example for such a connected graph.

Assume that we run the IterF-1 over the connected graph in Figure 5. After the Initialize procedure, every node will have his two neighbors as his best left and right nodes, except for the nodes 1 and 6 which have three neighbors each. 1 will have 11 as his best left node, and 3 as his best right node. 6 will have 4 as his best left node, and 8 as his best right node.

It can be checked that this state of the algorithm is stationary. As an example, observe node 5. Node 7 will send information about node 9 to node 5, but 5 will prefer his current best left and right nodes (3 and 7). Node 3 will send information about node 1 to 5, but again 5 will prefer his current best left and right nodes.

The stationary state resulted from running IterF-1 over this graph results in two h disjoint cycles: $(1, 3, 5, 7, 9, 11)$ and $(2, 4, 6, 8, 10, 12)$.

3.4. **Multiple rounds.** Given a graph G with $V(G) \subseteq S_l$, we define the following:

Definition 3.4. For every node $x \in V(G)$:

(1) $next(x) := \arg\min_{y \in V(G)} d(x+1, y)$
(2) $prev(x) := \arg\min_{z \in V(G)} d(z, x-1)$

Assume that after running IterF-1 over the graph G only one h-cycle was formed. Is it always true that $x^r = next(x)$ and $x^l = prev(x)$ for every $x \in V(G)$? This property can be useful to create a naive routing algorithm.

Given this property, during a stationary state of IterF-1 every node x keeps a path to $x^r = next(x)$, so we can go from x to $next(x)$ following this path. The

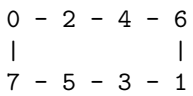

FIGURE 6. A one-cycle graph with multiple rounds

ability to go from x to $next(x)$ allows us to visit all nodes in the graph, eventually arriving to a wanted destination node.[3]

Unfortunately, IterF-1 does not always satisfy $x^r = next(x)$ for all $x \in V(G)$. This is demonstrated by the graph in Figure 6.

In this example, $next(x) = (x+1) \mod 8$ for every $x \in V(G)$. Running IterF-1 algorithm over the graph in Figure 6 will yield a stationary state (Right after running the Initialize procedure for every node), where every node has his best left and right nodes to be his two neighbors in the graph. Hence the resulting h-cycle will be $(0, 2, 4, 6, 1, 3, 5, 7)$, which contains all the nodes of the graph.

For this stationary state we have $next(0) = 1$, but $0^r = 2$, so it is not true that $x^r = next(x)$ for every node $x \in V(G)$.

Informally, we want our cycle to complete only one round, and not two as in the graph of Figure 6. The following definition tries to capture this idea.

Definition 3.5. Given a cycle $C = (c_0, c_1, \ldots, c_{k-1}, c_k = c_0) \subseteq \mathrm{S}_l$, we define the **amount of rounds of** C to be the value $R(C) = \frac{1}{2^l} \sum_{i=0}^{k-1} d(c_i, c_{i+1})$

Lemma 3.6. *Let C be an cycle in S_l. Then $R(C) \in \mathbb{Z}$.*

Proof. For a cycle $C = (c_0, \ldots, c_{k-1}, c_k = c_0) \subseteq S_l$ we denote $d(C) := \sum_{i=0}^{k-1} d(c_i, c_{i+1})$. To prove that lemma it is enough to show that $2^l \mid d(C)$ for every cycle $C \subseteq S_l$.

The proof is by induction on the amount of nodes in the cycle. Base case: For a cycle of one node: (c_0), we obtain $d(C) = d(c_0, c_0) = 0$ which is divisible by 2^l.

Inductive step: Assume that $2^l \mid d(C)$ for every cycle C of at most k nodes. Let C be a cycle of $k+1$ nodes, and denote by C' the reduced cycle $(c_0, \ldots, c_{k-2}, c_{k-1})$. We obtain:

$$d(C) = \sum_{i=0}^{k} d(c_i, c_{i+1}) = \left(\sum_{i=0}^{k-2} d(c_i, c_{i+1}) \right) + d(c_{k-1}, c_k) + d(c_k, c_0) \tag{3.1}$$

$$\equiv \left(\sum_{i=0}^{k-2} d(c_i, c_{i+1}) \right) + d(c_{k-1}, c_0) \tag{3.2}$$

$$\equiv d(C') \pmod{2^l} \tag{3.3}$$

The equality 3.2 is true because of Lemma 2.2.

As C' contains k nodes, we can use the inductive hypothesis to conclude that $2^l \mid d(C')$. But $d(C) \equiv d(C') \pmod{2^l}$, so $2^l \mid d(C)$ We conclude that $2^l \mid d(C)$ for every cycle $C \subseteq S_l$, and therefore $R(C) := d(C)/2^l \in \mathbb{Z}$.

□

[3]This is a very slow routing algorithm. It will visit $O(|V|)$ nodes before arriving at the destination.

See for example the cycle in Figure 6. Assuming $l = 3$, we can calculate the amount of rounds for this cycle to be

$$(d(0,2) + d(2,4) + d(4,6) + d(6,1) + d(1,3) + d(3,5) + d(5,7) + d(7,0))/8 \tag{3.4}$$

$$= (2 + 2 + 2 + 3 + 2 + 2 + 2 + 1)/8 = 16/8 = 2 \tag{3.5}$$

4. IterF-2 Algorithm

Next, we make some modifications to the IterF-1 to ensure all the cycles C that we obtain have only one round ($R(C) = 1$). Recall that in the IterF-1 algorithm every node x maintains a path to the nodes closest to him (With respect to the d metric) from the left and from the right that he knowns of. We called those nodes x^r and x^l. In IterF-2 we plan to let every node maintain paths to more nodes.

Definition 4.1. Given a node x, we denote:

(1) $\overleftarrow{x_k}$ is the node y that minimizes $d(y, x+k)$, of all the nodes y that x knows of.
(2) $\overrightarrow{x_k}$ is the node z that minimizes $d(x+k, z)$, of all the nodes z that x knows of.

The addition $x + k$ is performed modulo 2^l.

Note that by definition, $x^r = \overrightarrow{x_1}$ and $x^l = \overleftarrow{x_{-1}}$.

In IterF-2 every node x maintains paths to x^l and to all nodes of the form $\overrightarrow{x_{\pm 2^i}}$, for $0 \leq i < l$. The rest of the IterF-2 algorithm is identical to IterF-1. We describe in Algorithm 2 the modifications required to obtain IterF-2. Note that only the procedures UpdateFingers and GetFingers were changed.

Algorithm 2 IterF-2 algorithm (Modifications to IterF-1)

```
1: procedure UpdateFingers(candidates)
2:     x^l ← arg min_{y∈candidates} d(y, x − 1)
3:     for i from 0 to l do
4:         x→_{±2^i} ← arg min_{z∈candidates} d(x ± 2^i, z)
5:     end for
6: end procedure
7: procedure GetFingers
8:     return { x^l } ∪ { x→_{±2^i} | 0 ≤ i < l }
9: end procedure
```

4.1. **Analysis of IterF-2.** Given a graph G with $V \subseteq S_l$, and an execution of IterF-2 algorithm over G, we define the **state** of the algorithm to be

$$K = \left\{ (x, x^l, \{\overrightarrow{x_{\pm 2^i}}\}_{i=0}^{l-1}) \;\middle|\; x \in V \right\}$$

We say that a state K of the IterF-2 algorithm is **stationary** if K will never change for any order of periodic ticks and message arrivals.

Lemma 4.2 (IterF-2 Convergence)**.** *Every run of IterF-2 arrives at a stationary state after a finite amount of changes to K.*

Proof. For a state K of the algorithm we define:

$$e_2(K) := \sum_{x \in V} \sum_{0 \le i < l} \left(d(x^l, x-1) + d(x+2^i, \overrightarrow{x_{2^i}}) + d(x-2^i, \overrightarrow{x_{-2^i}}) \right)$$

e_2 is the energy function of the current state of the algorithm. Note that K changes whenever a node y updates one of his maintained nodes to a better node In every such case $e_2(K)$ decreases.

As $e_2 \in \mathbb{N}$, it can decrease only a finite amount of times. Hence K can change only a finite amount of times, after which the algorithm will reach a stationary state. □

For every state K_2 of an IterF-2 execution, we can **reduce** the state to a state K_1 of an IterF-1 execution by leaving only x, x^l and $x^r = \overrightarrow{x_1}$ inside every tuple of the state.

Lemma 4.3 (Stationary Reduction)**.** *Assume an IterF-2 stationary state K_2 over a graph G with $V(G) \subseteq S_l$. Then a reduction of K_2 to an IterF-1 state will yield a state K_1 that is IterF-1 stationary.*

Proof. Suppose that the reduction state K_1 is not IterF-1 stationary. This means that there is some node z that can receive a message from some node y which will result in updating one of his best left or right nodes, z^l or z^r, to a better node.

This could have happened by y sending an UpdateRequest message to z, or by z sending an UpdateRequest message to y, followed by an UpdateResponse message from y to z.

The message from y to z contained $\{ y^l, y^r \} \cup Neighbors(y)$. This means that the set $\{ y, y^l, y^r \} \cup Neighbors(y)$ contains a node that is better than z^l or z^r.

If in the IterF-2 algorithm execution y sends a message to z (Either an UpdateRequest or UpdateResponse message), it will give z information about the nodes $\{ y, y^l, y^r \} \cup Neighbors(y)$, one of which is a better than z^l or z^r. Therefore z will update one of z^l, z^r. This shows that the state K_2 is not IterF-2 stationary, which is a contradiction. □

We use state reductions to be able to apply our previously obtained IterF-1 results for the execution of the IterF-2 algorithm.

Definition 4.4. An **h-path** Z between nodes a and b (clockwise) is a sequence $a = z_1, z_2 \ldots, z_k = b$ where $h(z_j) = z_{j+1}$ The **d-length** of an h-path Z is the sum $d(Z) := \sum_{j=1}^{k-1} d(z_j, z_{j+1})$. A **direct h-path** Z between nodes a and b is an h-path $z_1, \ldots, z_k$ such that $d(Z) = d(a, b)$. Informally, a direct h-path is a path that doesn't go more than one round around the circle.

Lemma 4.5. *Assume a stationary state during execution of IterF-2 over a graph G with $V(G) \subseteq S_l$. Let $a \in G$ be a node, and assume $0 \le t < l$. Then $DPATH[t]$: There exist a direct h-path from the node a to the node $\overrightarrow{a_{2^t}}$, and from the node $\overrightarrow{a_{-2^t}}$ to the node a.*

Proof. By induction on t. We begin with the **base case**: $t = 0$. Let a be some node. For $\overrightarrow{a_1}$: The h-path $(a, \overrightarrow{a_1} = h(a))$ is a direct h-path.

For $\overrightarrow{a_{-1}}$: Note that it must be a node from the set $\{ a-1, a \}$. If $\overrightarrow{a_{-1}} = a$ then (a) is a 1-node direct h-path from $\overrightarrow{a_{-1}}$ to a. If $\overrightarrow{a_{-1}} = a - 1$, then $h(\overrightarrow{a_{-1}}) = a$.

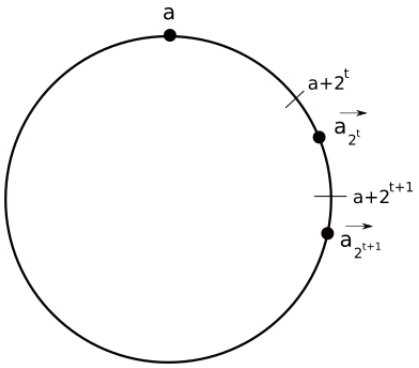

FIGURE 7. $\overrightarrow{a_{2^t}}$ and $\overrightarrow{a_{2^{t+1}}}$

(Otherwise in the next tick of a, a will send $\overrightarrow{a_{-1}}$ an UpdateRequest message and a_{-1} will update $h(\overrightarrow{a_{-1}}) = a$, a contradiction to the assumed stationary state).

The inductive step: Assume that for some $0 \leq t < l-1$ that $DPATH[t]$. We will prove $DPATH[t+1]$.

First, we show that there is a direct h-path **from** a **to** $\overrightarrow{a_{2^{t+1}}}$. We distinct between two cases:

Case 1: $d(a+2^t, \overrightarrow{a_{2^t}}) < d(a+2^{t+1}, \overrightarrow{a_{2^{t+1}}})$. Denote $z := \overrightarrow{\overrightarrow{a_{2^t}}_{2^t}}$. We will show that in this case $z = \overrightarrow{a_{2^{t+1}}}$.

By definition $z \in [\overrightarrow{a_{2^t}}+2^t, \overrightarrow{a_{2^t}}]$. If $z \in [\overrightarrow{a_{2^t}}+2^t, \overrightarrow{a^{2^{t+1}}})$ then z is a better candidate for $\overrightarrow{a_{2^{t+1}}}$. In the next tick of a, a will send an UpdateRequest message to $\overrightarrow{a_{2^t}}$. $\overrightarrow{a_{2^t}}$ will then send a an UpdateResponse message that contains a path to z. a will then update $\overrightarrow{a_{2^{t+1}}}$ to be z, which is a contradiction to the assumed stationary state of the algorithm.

If $z \in (\overrightarrow{a_{2^{t+1}}}, \overrightarrow{a_{2^t}}]$ then $\overrightarrow{a_{2^{t+1}}}$ is a better candidate for $z = \overrightarrow{\overrightarrow{a_{2^t}}_{2^t}}$. In the next tick of a, a will send an UpdateRequest message to $\overrightarrow{a_{2^t}}$. This message will contain a path to $\overrightarrow{a_{2^{t+1}}}$. $\overrightarrow{a_{2^t}}$ will then update $\overrightarrow{\overrightarrow{a_{2^t}}_{2^t}}$ to be $\overrightarrow{a_{2^{t+1}}}$, which is a contradiction to the assumed stationary state.

Hence the only possibility for z is the value $\overrightarrow{a_{2^{t+1}}}$. By the inductive hypothesis we have a direct h-path from a to $\overrightarrow{a_{2^t}}$, and a direct h-path from $\overrightarrow{a_{2^t}}$ to $\overrightarrow{\overrightarrow{a_{2^t}}_{2^t}} = z = \overrightarrow{a_{2^{t+1}}}$. The triple $(a, \overrightarrow{a_{2^t}}, \overrightarrow{a_{2^{t+1}}})$ is ordered clockwise. Therefore the concatenation of the two direct h-paths will result in a direct h-path from a to $\overrightarrow{a_{2^{t+1}}}$.

Case 2: $d(a+2^t, \overrightarrow{a_{2^t}}) \geq d(a+2^{t+1}, \overrightarrow{a_{2^{t+1}}})$. Denote $y := \overrightarrow{\overrightarrow{a_{2^{t+1}}}_{-2^t}}$. We will show that in this case $y = \overrightarrow{a_{2^t}}$.

By definition, $y \in [\overrightarrow{a_{2^{t+1}}} - 2^t, \overrightarrow{a_{2^{t+1}}}]$. If $y \in [\overrightarrow{a_{2^{t+1}}} - 2^t, \overrightarrow{a_{2^t}})$ then y is a better candidate for $\overrightarrow{a_{2^t}}$. In the next tick of a, a will send an UpdateRequest message to $\overrightarrow{a_{2^{t+1}}}$. $\overrightarrow{a_{2^{t+1}}}$ will then send an UpdateResponse message to a that contains a path to y. a will then update $\overrightarrow{a_{2^t}}$ to be y, which is a contradiction to the stationary state of the algorithm.

If $y \in (\overrightarrow{a_{2^t}}, \overrightarrow{a_{2^{t+1}}}]$ then $\overrightarrow{a_{2^t}}$ is a better candidate for $y = \overrightarrow{\overrightarrow{a_{2^{t+1}}}_{-2^t}}$. In the next tick of a, a will send an UpdateRequest message to $\overrightarrow{a_{2^{t+1}}}$. This message will contain a

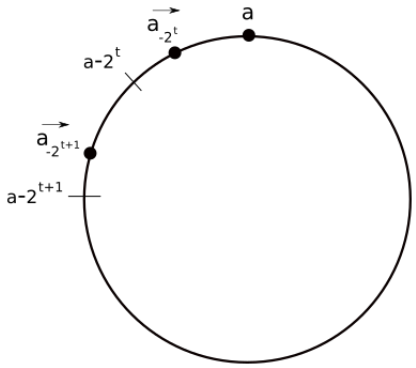

FIGURE 8. $\overrightarrow{a_{-2^t}}$ and $\overrightarrow{a_{-2^{t+1}}}$

path to $\overrightarrow{a_{2^t}}$. $\overrightarrow{a_{2^{t+1}}}$ will then update $\overrightarrow{\overrightarrow{a_{2^{t+1}}}_{-2^t}}$ to be $\overrightarrow{a_{2^t}}$, which is a contradiction to the stationary state of the algorithm.

Therefore the only possibility for y is the value $\overrightarrow{a_{2^t}}$. By the inductive hypothesis we have a direct h-path from a to $\overrightarrow{a_{2^t}}$, and a direct h-path from $\overrightarrow{a_{2^t}} = y = \overrightarrow{\overrightarrow{a_{2^{t+1}}}_{-2^t}}$ to $\overrightarrow{a_{2^{t+1}}}$. The triple $(a, \overrightarrow{a_{2^t}}, \overrightarrow{a_{2^{t+1}}})$ is ordered clockwise. Therefore the concatenation of the two direct h-paths will result in a direct h-path from a to $\overrightarrow{a_{2^{t+1}}}$.

Next, we show that there is a direct h-path **from** $\overrightarrow{a_{-2^{t+1}}}$ **to** a. We distinct between two cases:

Case 1: $d(a - 2^t, \overrightarrow{a_{-2^t}}) < d(a - 2^{t+1}, \overrightarrow{a_{-2^{t+1}}})$. Denote $z := \overrightarrow{\overrightarrow{a_{-2^t}}_{-2^t}}$. We will show that in this case $z = \overrightarrow{a_{-2^{t+1}}}$.

By definition $z \in [\overrightarrow{a_{-2^t}} - 2^t, \overrightarrow{a_{-2^t}}]$. If $z \in [\overrightarrow{a_{-2^t}} - 2^t, \overrightarrow{a_{-2^{t+1}}})$, then z is a better candidate for $\overrightarrow{a_{-2^{t+1}}}$. In the next tick of a, a will send an UpdateRequest message to $\overrightarrow{a_{-2^t}}$. $\overrightarrow{a_{-2^t}}$ will then send an UpdateResponse message back to a that contains a path to z. a will then update $\overrightarrow{a_{-2^{t+1}}}$ to be z, which is a contradiction to the assumed stationary state of the algorithm.

If $z \in (\overrightarrow{a_{-2^{t+1}}}, \overrightarrow{a_{-2^t}}]$ then $\overrightarrow{a_{-2^{t+1}}}$ is a better candidate for $z = \overrightarrow{\overrightarrow{a_{-2^t}}_{-2^t}}$. In the next tick of a, a will send an UpdateRequest message to $\overrightarrow{a_{-2^t}}$ which contains a path to $\overrightarrow{a_{-2^{t+1}}}$. $\overrightarrow{a_{-2^t}}$ will then update $\overrightarrow{\overrightarrow{a_{-2^t}}_{-2^t}}$ to be $\overrightarrow{a_{-2^{t+1}}}$, which is a contradiction to the assumed stationary state of the algorithm.

Therefore the only possibility for z is $\overrightarrow{a_{-2^{t+1}}}$. By the inductive hypothesis we have a direct h-path from $\overrightarrow{\overrightarrow{a_{-2^t}}_{-2^t}} = z = \overrightarrow{a_{-2^{t+1}}}$ to $\overrightarrow{a_{-2^t}}$, and a direct h-path from $\overrightarrow{a_{-2^t}}$ to a. The triple $(\overrightarrow{a_{-2^{t+1}}}, \overrightarrow{a_{-2^t}}, a)$ is ordered clockwise. Therefore the concatenation of the two direct h-paths will result in a direct h-path from $\overrightarrow{a_{-2^{t+1}}}$ to a

Case 2: $d(a - 2^t, \overrightarrow{a_{-2^t}}) \geq d(a - 2^{t+1}, \overrightarrow{a_{-2^{t+1}}})$. Denote $y := \overrightarrow{\overrightarrow{a_{-2^{t+1}}}_{2^t}}$. We will show that $y = \overrightarrow{a_{-2^t}}$.

By definition $y \in [\overrightarrow{a_{-2^{t+1}}} + 2^t, \overrightarrow{a_{-2^{t+1}}}]$. If $y \in [\overrightarrow{a_{-2^{t+1}}} + 2^t, \overrightarrow{a_{-2^t}})$ then y is a better candidate for $\overrightarrow{a_{-2^t}}$. In the next tick of a, a will send an UpdateRequest message to $\overrightarrow{a_{-2^{t+1}}}$. $\overrightarrow{a_{-2^{t+1}}}$ will send back to a an UpdateResponse message with a path to y. a will then update $\overrightarrow{a_{-2^t}}$ to be y, which is a contradiction to the assumed stationary state of the algorithm.

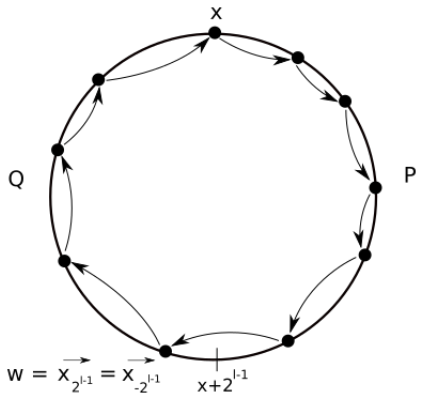

FIGURE 9. Paths from x to $\overrightarrow{x_{2^{l-1}}}$ and from $\overrightarrow{x_{-2^{l-1}}}$ to x.

If $y \in (\overrightarrow{a_{-2^t}}, \overrightarrow{a_{-2^{t+1}}}]$ then $\overrightarrow{a_{-2^t}}$ is a better candidate for $y = \overrightarrow{\overrightarrow{a_{-2^{t+1}}}_{2^t}}$. In the next tick of a, a will send an UpdateRequest message to $\overrightarrow{a_{-2^{t+1}}}$ that contains a path to $\overrightarrow{a_{-2^t}}$. $\overrightarrow{a_{-2^{t+1}}}$ will then update $\overrightarrow{\overrightarrow{a_{-2^{t+1}}}_{2^t}}$ to be $\overrightarrow{a_{-2^t}}$, which is a contradiction to the assumed stationary state of the algorithm.

Hence the only possibility for y is $\overrightarrow{a_{-2^t}}$. By the inductive hypothesis we have a direct h-path from $\overrightarrow{a_{-2^{t+1}}}$ to $\overrightarrow{\overrightarrow{a_{-2^{t+1}}}_{2^t}} = y = \overrightarrow{a_{-2^t}}$, and from $\overrightarrow{a_{-2^t}}$ to a. The triple $(\overrightarrow{a_{-2^{t+1}}}, \overrightarrow{a_{-2^t}}, a)$ is ordered clockwise. Therefore the concatenation of the two direct h-paths will result in a direct h-path from $\overrightarrow{a_{-2^{t+1}}}$ to a. □

Lemma 4.6 (one-round)**.** *Assume a stationary state during execution of IterF-2 over a graph G with $V(G) \subseteq S_l$. Let C be an h-cycle of nodes. Then $R(C) = 1$ (C has only one round).*

Proof. Observe the node $w = \overrightarrow{x_{2^{l-1}}} = \overrightarrow{x_{-2^{l-1}}}$.

On one hand, by Lemma 4.5 there is a direct h-path P from x to $w = \overrightarrow{x_{2^{l-1}}}$. On the other hand, by Lemma 4.5 there is a direct h-path Q from $w = \overrightarrow{x_{-2^{l-1}}}$ to x.

Therefore the cycle C is the concatenation of P and Q. Hence $d(C) = d(P) + d(Q) = d(x, w) + d(w, x) = 2^l$, and $R(C) = d(C)/2^l = 1$.

□

5. IterF-3 Algorithm

5.1. **Same h-cycle.** In IterF-2 we made sure we always arrive at a stationary state that contains possibly a few cycles, each of only one round ($R(C) = 1$). To allow routing between any two nodes, we would like to get a single cycle of only one round.

In this section we will show how achieve a single cycle by adding a few additional nodes to maintain.

We begin by observing the algorithm IterF-$2_{m,(b,b+2^j)}$, ($b \in S_l$ and $0 \leq j < l$, addition is done modulo 2^l) where for some specific node $m \in V$ we add two additional nodes to maintain: $\overrightarrow{m_b}$ and $\overrightarrow{m_{b+2^j}}$. (See Algorithm 3). **The rest of the nodes perform IterF-2, as before.**

Algorithm 3 IterF-$2_{m,(b,b+2^j)}$ algorithm for the node m
(Modifications to IterF-1)

procedure UPDATEFINGERS(candidates)
  $m^l \leftarrow \underset{y \in candidates}{\arg\min} \; d(y, m-1)$
  $\overrightarrow{m_b} \leftarrow \underset{z \in candidates}{\arg\min} \; d(m+b, z)$
  $\overrightarrow{m_{b+2^j}} \leftarrow \underset{z \in candidates}{\arg\min} \; d(m+b+2^j, z)$
  for i from 0 to l **do**
    $\overrightarrow{m_{\pm 2^i}} \leftarrow \underset{z \in candidates}{\arg\min} \; d(m \pm 2^i, z)$
  end for
end procedure
procedure GETFINGERS
  return $\{ m^l, \overrightarrow{m_b}, \overrightarrow{m_{b+2^j}} \} \cup \{ \overrightarrow{m_{\pm 2^i}} \mid 0 \leq i < l \}$
end procedure

Given a graph G with $V \subseteq S_l$, and an execution of IterF-$2_{(b,b+2^j)}$ algorithm over G, we define the state of the algorithm to be

$$K = \left\{ (x, x^l, \{\overrightarrow{x_{\pm 2^i}}\}_{i=0}^{l-1}) \;\middle|\; x \in V \setminus \{ m \} \right\} \cup \left\{ (m, m^l, \overrightarrow{m_b}, \overrightarrow{m_{b+2^j}}, \{\overrightarrow{m_{\pm 2^i}}\}_{i=0}^{l-1}) \right\}$$

We say that a state K is stationary if K will never change for any order of periodic ticks and message arrivals. Like before, we can prove that every execution of the algorithm IterF-$2_{m,(b,b+2^j)}$ eventually arrives at a stationary state after a finite amount of changes to K.

We can reduce a state of IterF-$2_{m,(b,b+2^j)}$ to an IterF-2 state by leaving all the tuples of $x \neq m$ unchanged, and for m's tuple: Omitting the extra maintained nodes $\overrightarrow{m_b}, \overrightarrow{m_{b+2^j}}$. We can prove the properties of state reductions similarly to Lemma 4.3. In other words, if we have a stationary state of IterF-$2_{m,(b,b+2^j)}$, we can reduce it to a stationary state of IterF-2 (And then reduce it even further to a stationary state of IterF-1).

Lemma 5.1 (Same-h-cycle)**.** *Assume a stationary state during execution of IterF-$2_{m,(b,b+2^j)}$ over a graph G with $V(G) \subseteq S_l$. Then in the division of the graph to disjoint one round h-cycles, the nodes $\overrightarrow{m_b}$ and $\overrightarrow{m_{b+2^j}}$ are on the same h-cycle.*

Proof. We distinct between two cases:

Case 1: $d(m+b, \overrightarrow{m_b}) < d(m+b+2^j, \overrightarrow{m_{b+2^j}})$. Then we denote $z := \overrightarrow{\overrightarrow{m_b}_{2^j}}$. We will show that $z = \overrightarrow{m_{b+2^j}}$.

By definition $z \in [\overrightarrow{m_b} + 2^j, \overrightarrow{m_b}]$. If $z \in [\overrightarrow{m_b} + 2^j, \overrightarrow{m_{b+2^j}})$ then z is a better candidate for $\overrightarrow{m_{b+2^j}}$. In the next tick of m, m will send $\overrightarrow{m_b}$ an UpdateRequest message. $\overrightarrow{m_b}$ will then send m an UpdateResponse message that contains a path to z. m will then update $\overrightarrow{m_{b+2^j}}$ to be z, which is a contradiction to the stationary state of the algorithm.

If $z \in (\overrightarrow{m_{b+2^j}}, \overrightarrow{m_b}]$ then $\overrightarrow{m_{b+2^j}}$ is a better candidate for $z = \overrightarrow{\overrightarrow{m_b}_{2^j}}$. In the next tick of m, m will send an UpdateRequest message to $\overrightarrow{m_b}$ that contains a path to $\overrightarrow{m_{b+2^j}}$. $\overrightarrow{m_b}$ will then update $\overrightarrow{\overrightarrow{m_b}_{2^j}}$ to be $\overrightarrow{m_{b+2^j}}$, which is a contradiction to the assumed stationary state of the algorithm.

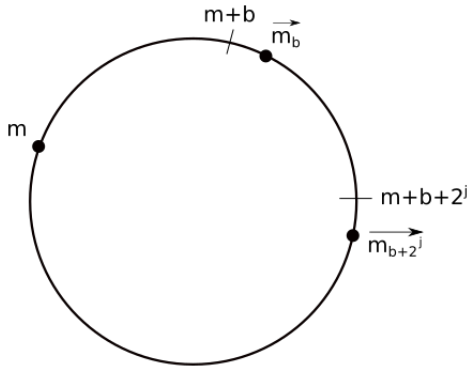

FIGURE 10. m, $\overrightarrow{m_b}$ and $\overrightarrow{m_{b+2^j}}$

Therefore the only possibility left is that $z = \overrightarrow{m_{b+2^j}}$. This means that $\overrightarrow{\overrightarrow{m_{b2^j}}} = \overrightarrow{m_{b+2^j}}$. Given that the current state of the algorithm is stationary, we can reduce it to a stationary state of the IterF-2 algorithm, and by Lemma 4.5 we conclude that there is a direct h-path from $\overrightarrow{m_b}$ to $\overrightarrow{m_{b+2^j}}$. Therefore $\overrightarrow{m_b}$ and $\overrightarrow{m_{b+2^j}}$ must be on the same h-cycle.

Case 2: $d(m+b, \overrightarrow{m_b}) \geq d(m+b+2^j, \overrightarrow{m_{b+2^j}})$. We denote by $y := \overrightarrow{\overrightarrow{m_{b+2^j}}_{-2^j}}$. We will show that $y = \overrightarrow{m_b}$.

By definition, $y \in [\overrightarrow{m_{b+2^j}} - 2^j, \overrightarrow{m_{b+2^j}}]$. If $y \in [\overrightarrow{m_{b+2^j}} - 2^j, \overrightarrow{m_b})$ then y is a better candidate for $\overrightarrow{m_b}$. In the next tick of m, m will send an UpdateRequest message to $\overrightarrow{m_{b+2^j}}$. $\overrightarrow{m_{b+2^j}}$ will then send m an UpdateResponse message that contains a path to y. m will then update $\overrightarrow{m_b}$ to be y, which is a contradiction to the stationary state of the algorithm.

If $y \in (\overrightarrow{m_b}, \overrightarrow{m_{b+2^j}}]$ then $\overrightarrow{m_b}$ is a better candidate for $y = \overrightarrow{\overrightarrow{m_{b+2^j}}_{-2^j}}$. In the next tick of a, a will send an UpdateRequest to $\overrightarrow{m_{b+2^j}}$ that contains a path to $\overrightarrow{m_b}$. $\overrightarrow{m_{b+2^j}}$ will then update $\overrightarrow{\overrightarrow{m_{b+2^j}}_{-2^j}}$ to be $\overrightarrow{m_b}$, which is a contradiction to the stationary state of the the algorithm.

Hence the only possibility left is that $y = \overrightarrow{m_b}$. This means that $\overrightarrow{\overrightarrow{m_{b+2^j}}_{-2^j}} = \overrightarrow{m_b}$. Given that the current state of the algorithm is stationary, we can reduce it to a stationary state of the IterF-2 algorithm, and by Lemma 4.5 we conclude that there is a direct h-path from $\overrightarrow{m_b}$ to $\overrightarrow{m_{b+2^j}}$. Therefore $\overrightarrow{m_b}$ and $\overrightarrow{m_{b+2^j}}$ must be on the same h-cycle.

□

5.2. **One h-cycle.** Lemma 5.1 allows us to make sure that two nodes of the form: $\overrightarrow{m_b}, \overrightarrow{m_{b+2^j}}$ will always end up on the same h-cycle during a stationary state. To achieve this, the node m has to maintain paths to two additional nodes: $\overrightarrow{m_b}$ and $\overrightarrow{m_{b+2^j}}$.

Chaining this method repeatedly, we can make sure that m ends up on the same h-cycle with any neighboring node x (A node that is directly connected to x). We first observe the binary structure of the number $x - m \pmod{2^l}$. Let $j_1, \ldots, j_s$ be the set of index numbers of set bits in this number. In other words, $s \leq l$ and $x - m \equiv \sum_{1 \leq i \leq s} 2^{j_i} \pmod{2^l}$.

The following will be the list of nodes for m to maintain:

- $\overrightarrow{m_{2^{j_1}}}$
- ...
- $\overrightarrow{m_{2^{j_1}+2^{j_2}+\cdots+2^{j_s}}} = \overrightarrow{m_{x-m}} = x$

The last equality $\overrightarrow{m_{x-m}} = x$ is true because m is directly connected to x. Maintaining the list of nodes above, we guarantee that m will be on the same h-cycle as $\overrightarrow{m_{2^{j_1}}}$, which will be on the same h-cycle as $\overrightarrow{m_{2^{j_1}+2^{j_2}}}$, ..., which will be on the same h-cycle as x. This means that m and x will be on the same h-cycle during a stationary state. We denote by $C_{(m,x)}$ the list $\left(2^{j_1}, \ldots, 2^{j_1} + 2^{j_2} + \cdots + 2^{j_s} = x\right)$.

Algorithm 4 IterF-3, algorithm for the node x (Modifications to IterF-1)

```
1: procedure UpdateFingers(candidates)
2:     x^l ← arg min_{y∈candidates} d(y, x − 1)
3:     for i from 0 to l do
4:         x→_{±2^i} ← arg min_{z∈candidates} d(x ± 2^i, z)
5:     end for
6:     for n ∈ Neighbors(x) do
7:         for q ∈ C_{(x,n)} do
8:             x→_q ← arg min_{z∈candidates} d(x + q, z)
9:         end for
10:    end for
11: end procedure
12: procedure GetFingers
13:    return { x^l } ∪ { x→_{±2^i} | 0 ≤ i < l } ∪ { x→_q | q ∈ C_{(x,n)} ∧ n ∈ Neighbors(x) }
14: end procedure
```

Based on the ideas above, we construct the IterF-3 Algorithm 4. Given a graph G with $V \subseteq S_l$ and an execution of IterF-3 algorithm over G we define the state of the algorithm to be:

$$K = \left\{ (x, x^l, \{\overrightarrow{x_{\pm 2^i}}\}_{i=0}^{l-1}, \{\overrightarrow{x_q}\}_{q \in C_{(x,n)} \wedge n \in Neighbors(x)} \;\middle|\; x \in V \right\}$$

We say that a state K is stationary if K will never change for any order of periodic ticks and message arrivals. As before, we can prove that every execution of the algorithm IterF-3 eventually arrives at a stationary state after a finite amount of changes to K.

If in IterF-3 the node m maintains the nodes m_b, m_{b+2^j}, it is possible to reduce the state IterF-3 an IterF-$2_{m,(b,b+2^j)}$ state by omitting all the extra maintained nodes. We can prove the properties of state reductions similarly to Lemma 4.3.

In other words, if we have a stationary state of IterF-3 where the node m maintains the nodes m_b, m_{b+2^j}, we can reduce it to a stationary state of IterF-$2_{m,(b,b+2^j)}$ (And then possibly reduce it further to a stationary state of IterF-2 and IterF-1).

Theorem 5.2 (one-cycle)**.** *Assume a stationary state during execution of IterF-3 over a connected graph G with $V(G) \subseteq S_l$. Then there exists a one-round h-cycle that contains all the nodes.*

Proof. The stationary state K of IterF-3 can be reduced to a stationary state of IterF-2. During a stationary state of IterF-2 there is a division of the nodes to disjoint h-cycles, each of one round.

Let x be some node, and let y be a neighbor of x. (y is directly connected to x). According to IterF-3 Algorithm 4, assuming that $s \leq l$ and $y - x \equiv \sum_{1 \leq i \leq s} 2^{j_i} \pmod{2^l}$, x maintains the nodes

$$\overrightarrow{x_{2^{j_1}}}, \ldots, \overrightarrow{x_{2^{j_1}+2^{j_2}+\cdots+2^{j_s}}} = y$$

Therefore the stationary state K can be reduced to a stationary state of the algorithm IterF-$2_{x,(2^{j_1}+\cdots+2^{j_w},2^{j_{w+1}})}$ for $1 \leq w < s$. By Lemma 4.5 we obtain that x and $\overrightarrow{x_{2^{j_1}}}$ are on the same h-cycle. By Lemma 5.1 we get that $\overrightarrow{x_{2^{j_1}+\cdots+2^{j_w}}}$ is on the same h-cycle as $\overrightarrow{x_{2^{j_1}+\cdots+2^{j_w}+2^{j_{w+1}}}}$ for $1 \leq w < s$. Being on the same h-cycle is transitive, therefore x and y will be on the same h-cycle.

As every two directly connected nodes are on the same one-round h-cycle and the graph G is connected, we conclude that all the nodes in $V(G)$ are on the same one-round h-cycle. □

Note that the addition of nodes to maintain as done in IterF-3 Algorithm 4 is very unlikely to be the most efficient method to ensure a single one-round h-cycle.

A specific way to think about this problem is to consider the graph Q: $V(Q) = S_l = [0, 2^l)$, with edges

$$E(Q) = \left\{ \{u, v\} \;\middle|\; d(u, v) = 2^j \text{ or } d(v, u) = 2^j \textit{ for some } 0 \leq j < l \right\}$$

Given a subset $U \subseteq V(Q)$, what is the minimum amount of edges required for a tree to cover the vertices set U?

Our current solution (Used in IterF-3 Algorithm 4) is to pick one vertex $u \in U$ and find paths from u to all of the other vertices in U. This requires at most $l \cdot (|U| - 1)$, which is $O(l \cdot |Neighbors(u)|)$.

6. Greedy Routing

In the previous sections we have demonstrated how to create a connected ring overlay structure given a connected network. We now show how routing is performed in the resulting overlay structure.

Lemma 6.1 (global-optimality)**.** *Assume a stationary state during execution of IterF-3 over a connected graph G with $V(G) \subseteq S_l$. Let $x \in V(G)$ and some $r \in S_l$ such that $\overrightarrow{x_r}$ is maintained by x. Then $\overrightarrow{x_r} = \arg\min_{y \in V(G)} d(x + r, y)$*

Proof. Assume a stationary state of the IterF-3 Algorithm 4. Denote $g_r := \arg\min_{y \in V(G)} d(x + r, y)$. Suppose that $\overrightarrow{x_r} \neq g_r$. By the definition of g_r, $g_r \in [x+r, \overrightarrow{x_r}]$. By Theorem 5.2 all the nodes $V(G)$ are on the same single one-round h-cycle. Therefore there is a direct h-path from g_r to $\overrightarrow{x_r}$.

Consider the node p in that direct h-path such that $p^r = h(p) = \overrightarrow{x_r}$. $p \in (g_r, \overrightarrow{x_r})$, therefore p is a better candidate for $\overrightarrow{x_r}$. By Lemma 3.2, $\overrightarrow{x_r}^l = p$. Therefore next time x sends an UpdateRequest message to $\overrightarrow{x_r}$, $\overrightarrow{x_r}$ will respond with an UpdateResponse message that contains a path to the node p. x will then update his value of $\overrightarrow{x_r}$ to be p, which is a contradiction to the given stationary state of the algorithm.

We conclude that $\overrightarrow{x_r} = g_r = \arg\min_{y \in V(G)} d(x + r, y)$. □

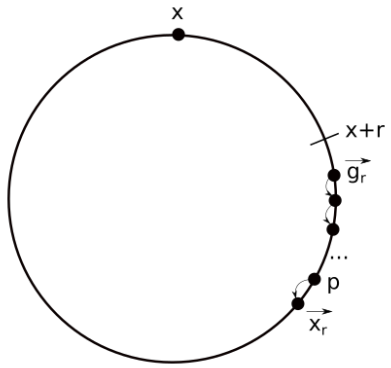

FIGURE 11. A direct path from g_r to $\overrightarrow{x_r}$

Algorithm 5 Greedy Routing, algorithm for node x

```
1: procedure passMessage(m,y)
2:     if y == x then
3:         onIncomingMessage(m)
4:         return Received
5:     end if
6:     closest ← arg min_{z∈GetKnown()∪{x}} d(z,y)
7:     if closest == x then
8:         return Dropped
9:     end if
10:    send PassMessage(m,y) to closest along the shortest maintained path.
11: end procedure
12: procedure onIncomingPassMessage(m,y)
13:    return passMessage(m,y)
14: end procedure
```

Algorithm 5 describes how greedy routing is performed. If a node x wants to send a message to node y, he finds the closest node he knows of to y (With respect to the d distance), and then passes the message to that node along a maintained path. We call this process a hop.[4]

Lemma 6.2 (routing-convergence)**.** *Assume a stationary state during execution of IterF-3 over a connected graph G with $V(G) \subseteq S_l$. Let $x \in V(G)$ and $y \in S_l$. Then the execution of $passMessage(m, y)$ from the node x will terminate after at most $\log_2 d(x, y)$ hops. If $y \in V(G)$, the message will be received by y.*

Proof. During the i-th hop, the node x^i runs $passMessage(m, y)$. If $x^i = y$, the node x^i receives the message. Otherwise, x^i calculates $closest_i$ which is the node z that minimizes $d(z, y)$ of all the nodes x^i knows of.

[4]Note that during a hop a message is usually passed through many nodes in the network, as it is sent along a maintained path

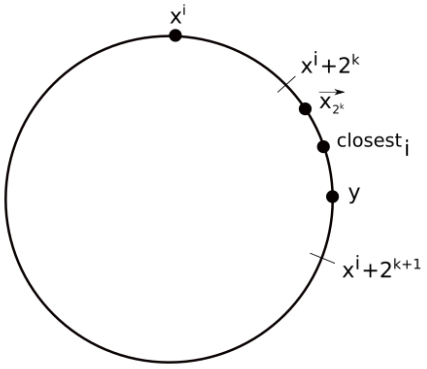

FIGURE 12. The i-th hop: Routing from x^i to y

Consider the distance $d(x^i, y)$. For some $k \in [0, l)$, $2^k \leq d(x^i, y) < 2^{k+1}$. x^i maintains a path to the node $\overrightarrow{x^i}_{2^k}$. By Lemma 6.1 $\overrightarrow{x^i}_{2^k} = \arg\min_{v \in V(G)} d(x^i + 2^k, v)$.

Consider the case of $closest_i = x^i$ (x^i will drop the message). If $y \in V(G)$ then $\overrightarrow{x^i}_{2^k} \in [x^i + 2^k, y]$. Note that by definition of $closest_i$, $closest_i \in [\overrightarrow{x^i}_{2^k}, y]$. If $closest_i = x^i$ then $x^i \in [\overrightarrow{x^i}_{2^k}, y]$, which is a contradiction. Therefore the message can only be dropped in some hop if $y \notin V(G)$.

In the case where the message was not dropped or received by x^i, it will be passed on: x^i sends $passMessage(m, y)$ to the node $closest_i = x^{i+1}$. $\overrightarrow{x^i}_{2^k} \in [x^i + 2^k, y]$, and so also $closest_i = x^{i+1} \in [x^i + 2^k, y]$. Thus $d(x^{i+1}, y) \leq d(x^i, y) - 2^k < d(x^i, y)/2$.

The distance to y: $d(x^i, y)$ decreases by a factor of 2 in every message passing hop. $d(x^i, y)$ is of integral positive value, therefore any message may be passed only a finite amount of times. For the first hop, $d(x, y) < 2^l$, hence there can be at most $\log_2 d(x, y) < l$ message passing hops.

We conclude that after at most l hops, every message will either be received or dropped. □

REFERENCES

1. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan *Chord: A scalable Peer-to-peer Lookup Service for Internet Applications*
2. Thomas Fuhrmann, Pengfei Di, Kendy Kutzner, Curt Cramer *Pushing Chord into the Underlay: Scalable Routing for Hybrid MANETs*
3. Matthew Caesar, Miguel Castro, Edmund B. Nightingale, Greg O'Shea, Antony Rowstron *Virtual Ring Routing: Network Routing Inspired by DHTs*

E-mail address, real: real@freedomlayer.org