GH-P2P: A Gray-code and Hamming distance based Peer-to-Peer Information System

Nicaise CHOUNGMO, Ali MAKKE and Petar MAKSIMOVIC
nicaiseeric@gmail.com, makkeali@gmail.com
INRIA Sophia Antipolis - LogNet - UBINET Master

May 3, 2010

Abstract

This paper presents GH-P2P, a scalable protocol to build a Gray-code based peer-to-peer overlay using a new metric called the Hamming distance. The properties of the Gray-code and the Hamming distance simplifies the algorithm, the proof of its correctness and others well-suitable properties.

1 Introduction

GH-P2P is a peer-to-peer a DHT-based protocol that helps to store and retrieve the pair ¡key,value¿ on/from a node participating to the network. The next section describes the protocol and the shows the overlay construction. Then the section 3 provides the complete pseudo-code for this overlay. The last section resumes the benefits, the properties and proofs of the overlay.

2 The GH-P2P protocol

2.1 GH-P2P description

The protocol is inspired from [1], according to the author, "the correlation between local structure and long-range connections provides fundamental cues for finding paths through the network". Hence, our protocol is a decentralized algorithm that builds for each node a local structure (or Direct Neighbors aka DN) and some long-range connections (or Far Neighbors aka FN), such that any node has a local and a long-range knowledge about from the network.

As the Fig. 2.1 shows, a node is roughly as likely to form links at distances 1 to 10 as it is at distances 10 to 100, 100 to 1000, and so on. The intuition is to have a uniform probability to form a link at all "distance scales" and add randomness on connections establishment in the network.

This protocol uses the hamming distance to organize the nodes joining the network in a Gray-code-based topology. Each node computes its binary ID by using a consistent hashing function on its IP address. GH-P2P uses the Hamming distance between two identifiers as a metric. This Hamming distance is an integer that represents the numbers of different bit them. The IDs are joining the network such that two adjacent IDs are Gray's codes and, indeed the Hamming distance between them is one.

In the steady state of the network, the one bit changing between a node u and each of its neighbors u_i (DN or FN) in their Gray code represents all the "distance scales" at 2^0, 2^1 till 2^{n-1} where n is the length of the binary ID. Note that the fact that the u_i joins the network randomly with a random ID (due to the hash fucntion) ensure that the connections inside the network are built with randomness like the Kleinberg's model suggested.

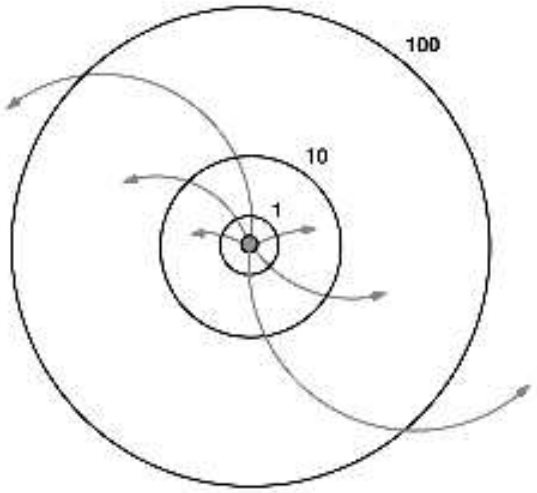


Figure 1: A node with several random shortcuts spanning different distance scales.

2.2 GH-P2P construction

Node representation.
In our system, each node is represented by the triple $\{NodeID, UserIPaddress, MachinePort\}$. To route query messages, each GH-P2P node stores information about its neighbors (the DN and FN) in a routing table called the "Gray table". In a steady state (that happens much more when many nodes join the network), a neighbor's ID differs from the current node by one bit, the Gray table contains at most n nodes (we recall n is the length of the binary ID). If we have N (at most $N = 2^n$) nodes in the network, each node stores at most $O(log_2(N))$ and we will show later that the routing or the path length is also logarithmic.

Node state.
A node u that joins a network tries to fill its Gray table by nodes that are the closest to him. It builds a set of IDs from whose it has the smallest Hamming distance, $ClosestGray = \{u_i, d_H(u, u_i) \leq d_{min}\}$ where $Card(ClosestGray) \leq n$ and d_{min} is the minimum possible distance between the node u and the nodes u_i that already existed in the network. P.S. put a figure to show the neighbors table.

Node processing.
Before joining the overlay, a node v should know one node u inside the network. Then v computes its distance from u and fills its Gray table with the couple $\{u, d_H(v, u)\}$. The action of "fill the Gray table" consists to find the farest neighbors nodes and exchange the oldest one by the new node. The node v sends the join request to node u that consists to find n closest nodes to v. When a node u receives a join request from a node v, its computes the distance $d_H(u, v)$. If u's Gray table is not full, u fills v. Otherwise, for each of its neighbors u_i, u computes their distance $d_H(u_i, v)$ from v; then if $d_H(u, v) < d_H(u_i, v)$, u does not forward the join request to u_i and keeps itself as one of closest node to return to v. If $d_H(u, v) = d_H(u_i, v)$, u forwards the join request to u_i and keeps itself as one of closest node to return to v. If $d_H(u, v) > d_H(u_i, v)$, u forwards the join request to u_i and erase itself as one of closest node to return to v. To resume the join request, u returns itself if it is one of the closest node to v , and forwards the join request to the others closest nodes that it knows for its Gray table. To finish, u

fills v in its routing table. After joining, a node v inside the overlay stays aware of others nodes by using two processes. First, v sends a stabilization request with its ID to all its neighbors v_i. Each neighbor v_i looks for a closest node to v and return this node. Each time v receives a node, it performs the fill action with this node. The second process of stabilization happens when a node receives a request with the ID's sender, its Gray table is updated.

Node interface.
The interface is defined to bear at least the four basic operations are the JOIN, PUT, GET and LEAVE provided by a DHT overlay. The keys are resulting from the hash of the value that we want to stored or put. To put or get any information (IP address, ID or Key), GH-P2P relies on the Hamming distance between two identifiers (the Key and an existing ID). The JOIN operation uses three elementary actions, $hamming(ID)$, $finds\_gray(ID)$ and $fills(ID)$. The PUT operation consists to $finds\_gray(KEY)$ and $stores(KEY)$ on one of the nodes that are the closest to key. The GET operation starts with $finds\_gray(KEY)$ and then calls $retrieves(KEY)$ on the node that is the closest to the key. To LEAVE, a node $stores(KEY)$ (each key) to the closest node that he finds in its neighbors table.

2.3 GH-P2P simulation

To clarify things, let's try an example. We take for this example $n = 4$ (we recall n is the ID length). A key table is represented by:$[< k_1, v_1 >; < k_2, v_2 >; ...; < k_n, v_n >]$ and the Gray table (aka routing table) has two colons. The first colon gives the ID's neighbor and the second gives the distance of the current node from the correspondant neighbor.

A. Menu:
. Five nodes joining, $a, b, c, dande$
. Three keys stored, $k_i, 1 \leq i \leq 3$
. Two nodes leaving with key (fair and failure)
. One key retrieved

B. Execution:
. $a = 1011$ creates the overlay (this happens when the first node joins):

a Gray table	
ID	$d_H(a, ID)$
.	.
.	.
.	.
.	.

. $b = 0010$ joins on a:

b Gray table	
ID	$d_H(b, ID)$
a	2
.	.
.	.
.	.

a Gray table	
ID	$d_H(a, ID)$
b	2
.	.
.	.
.	.

. $c = 0111$ joins on b

c Gray table	
ID	$d_H(c, ID)$
b	2
a	2
.	.
.	.

b Gray table	
ID	$d_H(b, ID)$
a	2
c	2
.	.
.	.

a Gray table	
ID	$d_H(a, ID)$
b	2
c	2
.	.
.	.

. $d = 1000$ joins on a

d Gray table	
ID	$d_H(d, ID)$
a	2
b	2
.	.
.	.

a Gray table	
ID	$d_H(a, ID)$
b	2
c	2
d	2
.	.

b Gray table	
ID	$d_H(b, ID)$
a	2
c	2
d	2
.	.

c Gray table	
ID	$d_H(c, ID)$
b	2
a	2
d	4
.	.

. $e = 0000$ joins on d

e Gray table	
ID	$d_H(e, ID)$
d	1
b	1
.	.
.	.

d Gray table	
ID	$d_H(d, ID)$
e	1
a	2
b	2
.	.

a Gray table	
ID	$d_H(a, ID)$
b	2
c	2
d	2
.	.

b Gray table	
ID	$d_H(b, ID)$
e	1
a	2
c	2
d	2

c Gray table	
ID	$d_H(c, ID)$
b	2
a	2
d	4
.	.

. b stores the key $k_1 = 0100$, hash of the value $v_1 = chou$. The key k_1 is send and then stored on node $e = 0000$, because it has the smallest distance $d_H(k_1, e) = 1$. e's key table is $[< k_1, v_1 >]$.

. e stores the key $k_2 = 1010$, hash of the value $v_2 = poup$. The key k_2 is stored on nodes b and d, since they both have the smallest $d_H(k_2, b) = d_H(k_2, d) = 1$. b and d key table is then $[< k_2, v_2 >]$.

. e leaves properly. e stores the key k_1 it ows on the closest nodes that belong in his Gray table.
So b and d key table become $[< k_2, v_2 >; < k_1, v_1 >]$. The Gray tables then look like:

a Gray table	
ID	$d_H(a, ID)$
b	2
c	2
d	2
.	.

b Gray table	
ID	$d_H(b, ID)$
a	2
c	2
d	2
.	.

c Gray table	
ID	$d_H(c, ID)$
b	2
a	2
d	4
.	.

d Gray table	
ID	$d_H(d, ID)$
a	2
b	2
.	.
.	.

. a fails. Nothing happens because the nodes will discover the failure during the routing or the pinging time. The Gray tables are then:

b Gray table	
ID	$d_H(b, ID)$
c	2
d	2
.	.
.	.

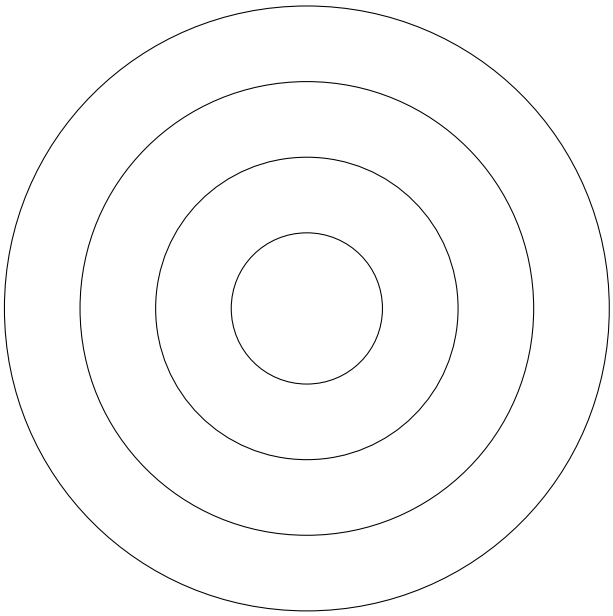
c Gray table	
ID	$d_H(c, ID)$
b	2
d	4
.	.
.	.

d Gray table	
ID	$d_H(d, ID)$
b	2
.	.
.	.
.	.

2.4 GH-P2P visualisation

In this section, we give two representations of our GH-P2P overlay.

With the Hamming distance:



In a Gray topology, the space wraps around. We suppose that the network is full, the overlay looks like

b = 0010	0110	1110	1010
0011	c = 0111	1111	a = 1011
0001	0101	1101	1001
e = 0000	0100	1100	d = 1000

3 GH-P2P pseudo-code

Each node should implement and run the following methods :$hamming(ID)$, $closest\_gray(ID)$, $finds\_gray(ID)$, $fills(ID)$, $stores(KEY)$ and $retrieves(KEY)$.

3.1 $hamming(ID)$

This method computes the Hamming distance d_H. The distance can be evaluated between two IDs (ID and/or key). The implementation depends on the programmer, if he want this method being remote or not, and so we define two prototypes: $ID.hamming(ID')$ or $hamming(ID, ID')$. For both, we have the same pseudo-code:

$ID'.hamming(ID)$:
. $d_H = \sum_{i=1}^{n} ID'(i) \otimes ID(i)$
.
. return d_H
. % $ID(i)$ is the i^{th} bit of the binary ID.

3.2 $closest\_gray(ID)$

This method returns the list of neighbors who are the closest to the ID in the current Gray table. This method can be remote or local, and the code is:

$ID'.closest\_gray(ID)$:
. $d = ID'.hamming(ID)$
. $list = ID'$
. forEach ID'_i
. $d_H = ID'_i.hamming(ID)$
. if $d_H \leq d$,
. then $list = list, ID'_i$
. return $list$
. % ID'_i is the i^{th} neighbor in the ID' Gray table.

3.3 $finds\_gray(ID)$

This method returns the current node if its belongs to list of the closest nodes to ID, if not the method forwards the request to the closest nodes that the current node knows. The code is the following:

$ID'.finds\_gray(ID)$:
. $list = ID'.closest\_gray(ID)$
. if $ID' \in list$,
. then return ID'
. forEach other $ID'_i \in list$,
. $ID'_i.find\_gray(ID)$

. $ID'.fills(ID)$

3.4 $fills(ID)$

To add (resp. remove) a node in (resp. from) the Gray table, we need to check if this node is the closest or the far one.

$ID'.fills(ID)$:
. if $ID'Gray\ table$ is not full,
. then ID'.ADD ID
. else
. ID'.REMOVE the neighbor whose distance is the highest
. ID'.ADD ID

3.5 $stores(KEY)$

Like all the DHTs, we can store information on many nodes.

$ID'.stores(KEY)$:
. $list = ID'.finds\_gray(KEY)$
. forEach $ID \in list$,
. ID.SAVE KEY

3.6 $retrieves(KEY)$

Related to the method $stores(ID)$, this method provides the mean to find an information.

$ID'.retrieves(KEY)$:
. $list = ID'.finds\_gray(KEY)$
. forOne $ID \in list$,
. ID.TAKE KEY

The instructions ADD, REMOVE, SAVE and TAKE are simple and linked to the data structures we use to store the information.

4 GH-P2P properties and proofs

5 Conclusion

References

[1] J. Kleinberg, "The Small-World Phenomenon: An Algorithmic Perspective," Oct 1999.