Netsukuku topology

http://netsukuku.freaknet.org
AlpT (@freaknet.org)

July 2, 2008

Abstract

In this document, we describe the hierarchical structure of the Netsukuku topology. Moreover, we show how it is possible to use the QSPN v2 on the high levels of the hierarchy.

Contents

1 Preface

We're assuming that you already know the basics of the QSPN. If not, read the QSPN document first: [1].

2 The general idea

The aim of Netsukuku is to be a (physical) scalable mesh network, completely distributed and decentralised, anonymous and autonomous.

The software, which must be executed by every node of the net, has to be unobtrusive. It has to use very few CPU and memory resources, in this way it will be possible to run it inside low-performance computers, like Access Points, embedded devices and old computers.

If this requirements are met, Netsukuku can be easily used to build a worldwide distributed, anonymous and not controlled network, separated from the Internet, without the support of any servers, ISPs or control authorities.

3 Basic definitions

Node We call *node* any computer that is hooked up to the Netsukuku network.

Rnode stands for *in-Range Node*: given a node X, it is any other node directly linked to X, i.e. it's a neighbour of X.

Map A map is a file, kept by each node, which contains all the necessary information about the network, f.e. routes and nodes status.

REM stands route *Route Efficiency Measure*. It is a value that indicates the quality of a route. REM can be calculated in various ways, f.e. by taking in account the total rtt and the bw capacity of the route. We denote the REM of a route r as $REM(r)$. If r is a better route than s, then $rem(r) > rem(s)$.

Example:
A is the rnode of B.

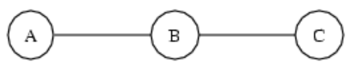

Figure 1: The nodes A,B and C

B is the rnode of A and C.
C is the rnode of B.

4 Network topology

A simple topology, which doesn't impose any structure on the network, can be memorised with a simple map. In this map, all the information regarding the nodes of the network have to be memorised. Surely, this kind of map cannot be utilised by Netsukuku, because it would require too much memory. For example, even if we store just one route to reach one node and even if this route costs one byte, we would need 1Gb of memory for a network composed by 10^9 nodes (the current Internet).

For this reason, it's necessary to structure the network in a convenient topology.

4.1 Hierarchical topology

4.1.1 Level 1

First of all we'll subdivide the network in groups of 256 nodes and we'll use the following definitions:

Gnode means group node. A group node G is a set of connected nodes. Each node of the network belongs to just one gnode. The nodes of G are connected in the following sense:

$$\forall a, b \in G \;\; \exists \text{ a path } a \to b \text{ all contained in } G$$

In the netsukuku implementation, a gnode contains a maximum of 256 nodes.
By writing $|G|$ we indicate the number of nodes contained in G.

Bnode stands for border node. It is a node which belongs to a gnode G, but that is also directly linked to at least one node of another gnode, i.e. some of its rnodes belongs to different gnodes than its.
By writing $b \in G$ we mean that the bnode b belongs to the gnode G.

Example:
$A \in G$, A is a node belonging to the gnode G, its rnode is B.

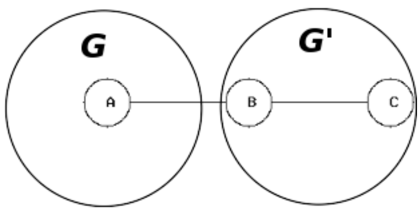

Figure 2: The bnode A and B, belonging respectively to the gnode G and G'

$B \in G'$, B is a node belonging to the gnode G', its rnode is A.
A is a bnode of G, while B is a bnode of G'.

4.1.2 Level n

We further subdivide the network topology in *groups of 256 groups of nodes* and we continue to name them as gnode.

At this point, we repeat recursively this subdivision process until we can group all the nodes of the network into a single gnode.

Doing so, we've structured the network in $n+1$ levels (from 0 to n).
In the base level (level 0), there are 256 single nodes.
In the first level (level 1), there are 256 normal gnodes. Each of them contains 256 single nodes.
In the second (level 2), 256 gnodes of level 1 forms a single *group of groups of nodes*.
In the third (level 3), there are 256 groups of 256 groups of 256 groups of 256 nodes.
Continuing in this way, we arrive at the last level (level n), where there is a single group which contains the whole network.

The QSPN algorithm is able to operate independently on any level, considering each gnode as a single node of level 0. For this reason, we can view the Netsukuku topology as a hierarchy, where each level is composed by single nodes.

Example

Figure 3[1] is an example of the hierarchical topology of Netsukuku.

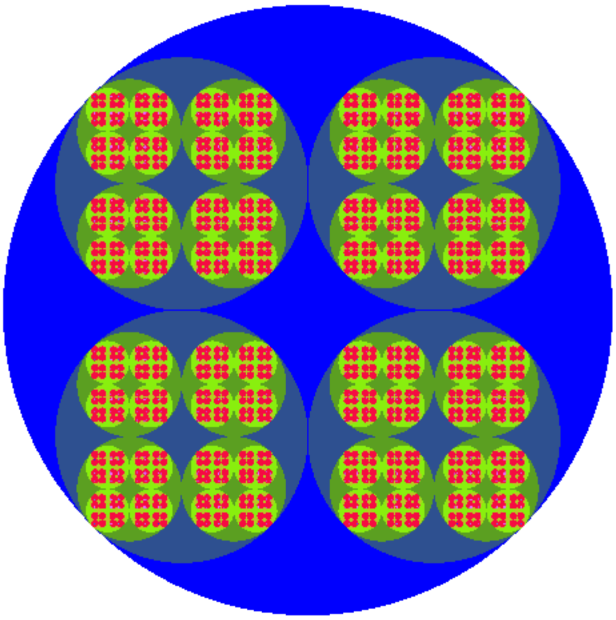

Figure 3: An example of the netsukuku topology structure

In this topology, each gnode contains four nodes, i.e. each group contains four elements. The network is structured in 6 levels.
The red elements, are single nodes (level 0).

[1]this figure has been taken from: http://www.ian.org/FX/Plugins.html

Four nodes forms a single group of nodes (level 1).
A single bright green circle is a group of groups of nodes (level 2).
The dark green circles are groups of groups of groups of nodes (level 3).
The dark blue circle are groups of groups of groups of groups of nodes (level 4).
Finally, the bright blue circle is the gnode which contains the whole network (level 5).

4.1.3 Membership

Let's assign a numeric ID to each (g)gnode, starting from the last level:

1. in the last level (n) there's only one giant gnode, thus we assign to it the ID "0". Our global ID will be:
$$0$$
2. in $n-1$ there are 256 gnodes, which belongs to the gnode 0 of level n, thus we assign them the IDs from 0 to 255. The global ID becomes:
$$0 \cdot i \quad 0 \leq i \leq 255$$
3. we repeat the step 2 recursively gaining an ID of this form:
$$0 \cdot i_{n-1} \cdot i_{n-2} \cdot \cdots \cdot i_0 \quad 0 \leq i_j \leq 255,\ 0 \leq j \leq n-1$$
4. since the last level is always 0, we'll omit it and we'll consider only the first n levels.

In a network with a maximum of 2^{32} nodes (the maximum allowed by the ipv4), there would be five levels ($n = 4$), where each gnode will be composed by 256 nodes. Therefore, the ID will be in the usual IP form:

$$0 \ldots 255 \cdot 0 \ldots 255 \cdot 0 \ldots 255 \cdot 0 \ldots 255$$

For example, a single node of level 0 of the network is:

$$3 \cdot 41 \cdot 5 \cdot 12$$

That said, each gnode of the network belongs to only one combination of gnodes of the various levels. In our previous example we have:

$$g_3 = 3$$
$$g_2 = 41$$
$$g_1 = 5$$
$$g_0 = 12$$

where each g_i corresponds to the gnode ID of the level i. Note that g_0 is the ID attributed to the single node, at level 0.

Finally, we can assign to each gnode G, of any level, a fingerprint $fg(G)$, which identifies G in a unique way, indipendently from its ID: two different, separated gnodes may have the same ID, but not the same fingerprint. A valid fingerprint is $fg(G) = (uptime(G), rid(G))$, where $rid(G)$ is a random ID generated at creation time of G.

4.2 Hierarchical map

The advantages of using a hierarchical topology are clear.
The node N, instead of memorising information about each node of the whole network, will keep only that regarding the gnodes where it belongs to. Suppose the node N had this ID:

$$g_3 \cdot g_2 \cdot g_1 \cdot g_0$$

It will store in memory information regarding:

1. the 256 single nodes which belongs to its same gnode of level 1, or in other words, the 256 nodes of the gnode g_1,
2. the 256 gnodes gnodes which belongs to its same gnode of level 2, of in other words, the 256 gnodes of the gnode g_2,
3. finally, the 256 gnodes which belongs to the gnode g_3.

Note that doing so, the node N will be blind to all the other gnodes. For example, it won't know any information regarding the single nodes which belong to all the other gnodes of level 1 different from g_1.

Even with this lack of knowledge, as we'll see later, the node N is still able to reach all the other nodes of the network. In conclusion, N only needs $256n$ entries in its map, instead of 2^{32}. To clarify the ideas suppose that each entry costs one byte. In the plain topology we needed $4Gb$, while in the hierarchical one we just need $256 \cdot 4\ b = 1Kb$.

4.2.1 IP v4 and v6

Netsukuku is both compatible with ipv4 and ipv6.

In ipv4 there are a maximum of 2^{32} IPs, thus we have five levels $n = 4$.
In ipv6 there are a maximum of 2^{128} IPs, thus $n = 16$.

4.2.2 Internal and external map

For simplicity we divide the map of the node N, in the *internal map* and in the *external* one. The internal map contains information regarding the nodes belonging to g_1. The external map describes all the other levels of the topology.

4.3 CIDR routing

The QSPN, for each level, will build the routes necessary to connect each (g)node to all the other (g)nodes of the same level. The routes will be saved in the maps of each node.

If the node $N = g_3 \cdot g_2 \cdot g_1 \cdot g_0$ wants to reach a node M which belongs to different gnodes, f.e. $M = g_3 \cdot g_2 \cdot h_1 \cdot h_0$, it will add a CIDR[3] route in the routing table of the kernel:
all the packets whose destination is $g_3 \cdot g_2 \cdot h_1 \cdot 0 \ldots 255$ *will be forwarded to the gateway* X.

We'll see later how the gateway X is chosen.

5 The internal map and its myopia

We define a route r_N of the node N as the following tern:

$r_N := (dst, gw, rem)$

where

dst is a node: the destination node of the route

gw is a node: the gateway of the route

rem is a number: the REM value of the route

We'll use $dst(r), gw(r), rem(r)$, to indicate respectively the first, second and third element of the tern r. Let V_N be the set of all neighbours of N.
Let $\mathbf{R}$ be the set of all routes of the node N (note [2]). We can define the following equivalence relation:

$$\forall r, s \in \mathbf{R} \quad r \sim s \Leftrightarrow dst(r) = dst(s)$$

With $\overline{r}$ we indicate the equivalence class of r. Let $B_x(r)$ be the best route of the routes in $\overline{r}$ with gateway x, where $x \in V_N$, i.e.

$$x \in V_N,\ B_x(r) \in \overline{r},\ \ rem(B_x(r)) = \max\{rem(s) \mid s \in \overline{r},\ gw(s) = x\}$$

Consider the following subset of $\mathbf{R}$:

$R = \{r \in \mathbf{R} \mid dst(r) \in G\}$

where G is the gnode of level 1 such that $N \in G$

At this point we can define the internal map of N as the subset $M \subseteq R$, where $M = \{B_x(r) \mid r \in R,\ x = gw(r)\}$. The following property holds:

$$\forall r \in M \ \ \forall s, t \in \overline{r} \text{ with } s \neq t \quad gw(s) \neq gw(t)$$

Algorithmically, when the node N wants to save in M a route s, which has the same destination of a route $r \in M$, the following algorithm will be adopted:

if $\exists t \in \overline{r} :\ gw(t) = gw(s)$
    if $rem(t) < rem(s)$ [s is better than t]
        overwrite t with s;
    else do nothing;
else save s in M;

The reason for the above definitions is that a node N, in order to reach any other node of the network will just need to know to which of its neighbours send the packets.
A route saved in the internal map it's just a tern, not an ordinate sequence of nodes, therefore the node N doesn't know its entire path. For this reason, we can say that the vision of the node N of the entire net is myope, or local. As we'll see later, this local vision can be easily applied to higher levels. Note also that the QSPN v2 is indipendent from the above definitions, i.e. it's way of working doesn't change.

[2]for simplicity we are considering just one metric for all the routes. However, it's easy to see that the following propositions are valid for different metrics (just use a different set $\mathbf{R}_m$ for each metric m)

6 Flat levels

From the point of view of the QSPN v2, the levels are “flattened”, because the propagation of an ETP [3] inside the whole network is exactly as before, briefly: a packet is propagated until it is interesting, the subdvisions of nodes in gnodes are simply ignored . The only added rule serves to economize space:

let T be a generic tracer packet and let $\{a_i \mid i = 1, 2, \ldots, n\}$ be its finite sequence of nodes. Every subsequence

$$\{a_h \in G \mid h = i + 1, i + 2, \ldots, j - 1\}$$

such that $a_i, a_j \notin G$, where G is a gnode of any level, is replaced by the ID of G. The rem associated with the ID of G is the summation of the rem of every term of the subsequence.
This rule is valid for any level, it is called the *group rule*.
Some examples:

1. The tracer packet

   $$11.22.1,\ 11.22.80,\ 11.22.35$$

   cannot be grouped, because $a_i \in 11.22\ \forall i = 1, 2, 3$

2. The tracer packet

   $$11.22.1,\ 11.22.80,\ 11.22.35,\ 11.44.13$$

   can be grouped in the following tracer packet:

   $$11.22.*,\ 11.44.13$$

3. The tracer packet

   $$11.22.1,\ 11.22.80,\ 11.22.35,\ 11.44.13,\ 55.32.20$$

   can be grouped in the following tracer packet:

   $$11.*,\ 55.32.20 \qquad (1)$$

The description of a route in the external map is local, as in the internal map (see 5). For example, when the node 55.32.12 receives the tracer packet (1), it will save the following route:

$$(dst = 11.* = \text{any node of the gnode 11},\ gw = \text{my neighbour 55.32.20},\ rem)$$

Finally, the set M used for the description of the ETP in the QSPN document [1], is changed from the set of all routes of the internal map to the set of all routes (internal and external).

[3] for more info on the ETP, see the QSPN document [1]

6.1 The approximation of the group rule

The group rule implies that a node $c \notin G$ cannot known the intern of G, i.e. it doesn't effectively known what nodes belongs to G and how they are disposed. In fact, the best route r of c to reach any node of G is just a route that reaches a node $d \in G$, that is the nearest node of G to c. However the route r is valid to reach any node $d' \in G$, because a gnode, by definition, is a connected set of nodes (see 4.1.1).
What isn't preserved is the accuracy of the route: r is the best route to reach $d \in G$, but it may not be the best route to reach $d' \in G$, where $d' \neq d$. In every case, r is an approximation of the best route to reach d': since $d \in G$, d knows the best route $d \to d'$, thus the route $c \to d'$ will be $c \to d \to d'$. The approximation can be improved with the use of the causting routing [5] and by imposing that the rem value of a route $b \to g$ contained in G, where b is a bnode, doesn't differ too much from any other route $b \to g'$ contained in G (see 7).

7 Hooking phase

The hooking procedures serve to:

1. Making new nodes join the network while keeping the space of the gnodes uniform.

2. Keep the gnodes internally connected

3. Keep the gnodes compact.

Generally speaking, the hooking procedures serves to resolve the problem of assigning each node of the network to a group. This problem is not simple because the groups must always remain internally connected, they are finite and the network is dynamic.

7.1 Uniform gnodes

Let G be a non empty gnode. Since G must be internally connected, a node n can become a node of G iff it is directly linked to at least one node $g \in G$. This fact brings some problems in the distribution and allocation of free IPs.

We say that G is *allocated* iff it isn't empty. A gnode of level l is said *saturated* iff all its nodes are allocated. An entire gnode $G \in \overline{G}$ of any level l can be kept allocated by a single node n, thus it is possible to quickly saturate the gnode $\overline{G}$ of level $l+1$. A saturated gnode may appear full even if it has free space, f.e. suppose the gnode $\overline{G}$ has been saturated by the gnodes $A, B, C \in \overline{G}$, suppose also that A is full and B, C have some free space. A node n, that is directly linked to a node $a \in A$, won't be able to join in $\overline{G}$. In fact, n can't become a node of A because it is a full gnode and can't become a node of B nor of C because it isn't directly connected to one of their nodes. Therefore, the only option left to n is to create a new gnode in $\overline{G}$. However, if $\overline{G}$ is the highest gnode, i.e. the gnode of all the nodes of the network, n won't be able to, because $\overline{G}$ is saturated by A, B, C. This problem is called *gnodes saturation.*

A strategy to solve the gnodes saturation problem is to uniformly distribute the nodes: all the gnodes of the net will have approximately the same number of nodes, at any time. This is achieved using a system that imitates the communicating vessels:

1. A hooking node n (f.e. a node which is joining the network) will create the set $\overline{S}$ containing the names of all the highest non saturated gnodes, all of level l. n can compose $\overline{S}$ by looking in the maps of its neighbours. From $\overline{S}$ it will choose $\overline{G}$, which is the gnode with the lowest number of nodes, i.e.
$$\overline{G} \in \overline{S} : \quad |\overline{G}| = \min_{G \in \overline{S}} |G|$$
Then, it will become a new (g)node of level $l-1$ of the gnode $\overline{G}$.

2. Let b be a bnode of the gnode G of level 1, and B the set of neighbours of b, such that $\forall c \in B \;\; c \notin G$. Let $G(c)$ be the gnode of level 1 of the node c and
$$\overline{B} = \{G(c) \ : \ |G(c)| + 1 < |G|, \ c \in B\}$$
If $\overline{B} = \emptyset$, stop here, otherwise let $H \in \overline{B}$ be the gnode with the lowest number of nodes, i.e. $|H| = \min_{J \in \overline{B}} |J|$. Then b will become a hooking node, thus it will create a new gnode or it will become a node of H, leaving G. Afterwhile, b will inform its neighbours belonging to G to repeat this same procedure[4]. The adviced neighbour, that cannot fulfill this procedure, will send an ETP in G, to inform the other nodes of the departure of b from G.

Example:

1. Consider the following scenario, where we have only one level, saturated by 3 gnodes:
$$11.11 \leftrightarrow 22.11 \leftrightarrow 33.11$$

2. Two nodes join in 11.∗:
$$11.33 \leftrightarrow 11.22 \leftrightarrow 11.11 \leftrightarrow 22.11 \leftrightarrow 33.11$$

3. The communicating vessels rule is applied:
$$11.33 \leftrightarrow 11.22 \leftrightarrow 22.22 \leftrightarrow 22.11 \leftrightarrow 33.11$$

4. Other two nodes join in 11.∗:
$$11.55 \leftrightarrow 11.44 \leftrightarrow 11.33 \leftrightarrow 11.22 \leftrightarrow 22.22 \leftrightarrow 22.11 \leftrightarrow 33.11$$
The situation evolves to
$$11.55 \leftrightarrow 11.44 \leftrightarrow 11.33 \leftrightarrow 22.33 \leftrightarrow 22.22 \leftrightarrow 22.11 \leftrightarrow 33.11$$
$$11.55 \leftrightarrow 11.44 \leftrightarrow 11.33 \leftrightarrow 22.33 \leftrightarrow 22.22 \leftrightarrow 33.22 \leftrightarrow 33.11$$

[4] note that since these neighbours are linked to b, which is now a node of H, they are also bnodes of G, bordering with H

7.1.1 Coordinator node

The transition of nodes from a gnode A to another B, must be serialized. The nodes can't translate simultaneously, because when a node n migrates, it knows $|A|$ and $|B|$ but it doesn't know how many other nodes are going to B and how many other are going away from A. Thus, without serialization, too many nodes can escape from a gnode or enter in another, without respecting the rules of the communicating vessels.

The solution is to assign a coordinator node $co(G)$ to each gnode of any level. This is essentially achieved using the "P2P over Ntk"[4]: the coordinator P2P service is created, and all the nodes are implicit participant. In this document, it suffices to say that given a gnode G we can reach its coordinator node $co(G)$.

A node n will contact $co(G)$ for any of the following cases:

1. n wants to become a node of G, if $|G|$ satifies a given condition (f.e. $|G| < |N|$, where N is the gnode of n)
2. n wants to leave G, if $|G|$ satifies a given condition (f.e. $|G| > |M|$)
3. n wants to create a new gnode
4. n wants to know the current $|G|$

n will execute the relative action, only if the $co(G)$ replies affirmatively. Obviously, $co(G)$ has to process the requests in a serial manner, i.e. interpreting one request and blocking all the other. $co(G)$ will consider a request as definitive, when it'll receive the relative ETP, otherwise, it will revert the change after a timeout, f.e. n may ask and have granted the request of joining in G, but it may not honor it, thus $co(G)$, will consider as void the request after a timeout of 30 seconds, restoring $|G|$.

7.1.2 Limits of the Communicating Vessels

The communicating vessels system is the main hard part of Netsukuku, it is the most expensive in terms of work, because when a node changes gnode it changes IP too. This is also the main reason for the non-mobility of Netsukuku.
In order to mitigate these problems, it is possible to classify the nodes in Static Nodes and Guest Nodes: the former are nodes connected for more than 10 minutes and are effectively nodes of Netsukuku, instead the latter are connected to the network just behind the NAT of their neighbours.

Contributes to these problems are very welcome.

7.2 Internal connection

A gnode G is internally connected if

$$\forall a, b \in G \ \ \exists \text{ a path } a \to b \text{ all contained in } G$$

A gnode can become broken (not internally connected) if another gnode of the network assumes the same gnode ID, or if one or more link inside the gnode die, splitting the gnode in two separate parts. In practice, the first case may

happen only if two separate networks meet each other for the first time: let $\overline{G}, \overline{G'}$ be two gnodes of level $l+1$, meeting for the first time. Suppose that $\exists G \in \overline{G},\ \exists G' \in \overline{G}'$: $ID(G) = ID(G')$, then the nodes of the smallest gnode or oldest uptime will rehook.
The second case, where G is split into two parts is handled as follow: a bnode $n \in G$ receives the ETP for the death of a link. Updating the maps with this ETP, it notices that it doesn't have any route to reach some nodes of G. Call E the set of the unreachable nodes. (note [5])
There are two possibilities: the nodes of E have changed gnode due the communicating vessels system, or the broken link has effectively split G in two parts. n can distinguish the two cases, because the ETP generated by the communicating vessels system are marked in a particular way. If n is in the second case, it belongs to the set $G \backslash E$. If $|G \backslash E| < |E|$, then n decides to becoming a hooking node, and it informs all the other nodes of $G \backslash E$ to do the same.

Example: see figure 4

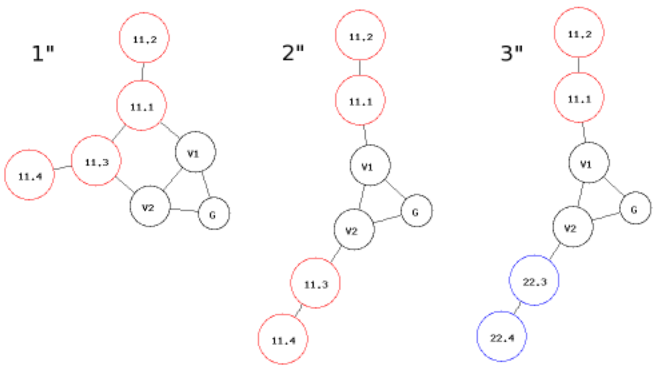

Figure 4: In the first step we have the gnode 11.* (in red). In the second, the gnode becomes broken. In the third, after the procedure described above has been applied, we have two gnodes: 11.* and 22.*

7.3 Compact gnodes

A gnode G is compact if, fixed a rem value $\overline{r}$,

$$\forall a, b \in G \ \ rem(a \xrightarrow{*} b) \leq \overline{r}$$

where $a \xrightarrow{*} b$ is the best path from a to b.
The gnodes don't necessary need to be compact, however the more compact they are, the more the grouping approximation is reduced (see 6.1).

An optimal technique to keep the gnodes compact hasn't been found yet. As an example, consider the following, which has been inspired by the gravitational model: let n be a node and V_n the set of its neighbours. For each neighbours

[5] generally, the fact that n doesn't know any route to reach d doesn't imply that there aren't any routes to reach d, however, due the structure of the internal map, this implication is true

$v \in V_n$, n calculates the following *attraction value*:

$$att(v) = \sum_{r \in R_v} rem(r)$$

where R_v is the set of all the best routes of n passing from v

n is then attracted by the neighbour $\overline{v}$, which has the highest attraction value: n will enter in the gnode of v (if it isn't already in).

The problem with any Compact gnodes system is that the grouping of nodes becomes dependent on the quality of the links, thus the network may be reconfigured more often.

8 ChangeLog

- `July 2008` Changed the term "fractal" with "hierarchical".
- `August 2007`
  1. Hooking procedures redesigned, probably they are near to their definitive form:
     (a) The grouping of nodes is now handled by the communicating vessels system
     (b) The case where a gnode is split in two parts is managed in a less complicated way
     (c) a "gravitational model" for the Compact gnodes has been proposed.
- `July 2007`
  1. Internal and external map structure redisigned: (see 5)
  2. The bnode map is no more necessary.
  3. Flat levels simplified.
  4. The hooking procedure has been fused with the rehooking preocedure.
  5. Removed section "Level 0".
- `March 2007`
  - Description of the Flat levels (sec. 6)
  - Section "Network dynamics - Level 0" expanded.
  - Section "Network dynamics - Level n" updated: the references to the pre-Flatlevels REM metric have been removed.
- `October 2006`
  Initial release.

References

[1] QSPN document: qspn.pdf

[2] Netsukuku website: http://netsukuku.freaknet.org/

[3] CIDR routing: Classless_Inter-Domain_Routing in Wikipedia

[4] P2P over Ntk: P2P over Ntk

[5] Caustic routing: RFC 0013

^_^