The QSPN

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

September 8, 2009

Abstract

This document describes the QSPN, the routing discovery algorithm used by Netsukuku. Through a deductive analysis, the main properties of the QSPN are shown. Moreover, a second version of the algorithm is presented.

Contents

1 The general idea

The aim of Netsukuku is to build 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 in low-performance machines like Access Points, embedded devices and old computers.

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

1.1 The network model

Netsukuku prioritises the stability and scalability of the net: the network has to be able to grow to even 2^{2^7} nodes.

A completely dynamic network would require rapid and frequent updates of the routes and this is in contrast with the stability and the scalability requirements of Netsukuku. For this reason, we restrict Netsukuku to the case where a node won't change its physical location quickly or often.

This assumption is licit because the location of a WiFi node mounted on top of a building won't change and its only dynamic actions would be the joining and the disconnecting to and from the network and the changes of the quality of its links.

However, there are some consequences with this assumption:

1. Mobiles node aren't supported by Netsukuku algorithms. [1]
2. The network isn't updated quickly: several minutes may be required before all the nodes become aware of a change in the network (new nodes have joined, more efficient routes have become available, ...). However, when a node joins the network, it can reach all the other nodes from the first instant using the routes of its neighbours.

1.2 The routing algorithm

One of the most important parts of Netsukuku is the routing discovery algorithm, which is responsible to find all the most efficient routes of the network.

The routing algorithm must be capable to find the routes without overloading the network or the nodes' CPU and memory resources.

[1] It is possible to use other mesh network protocols designed for mobility in conjunction with Netsukuku, in the same way they are used in conjunction with the Internet (i.e. see olsrd).

1.3 The QSPN

Netsukuku implements its own algorithm, the *QSPN*. It is based on the assumptions described in section 1.1.

2 Network topology

The QSPN alone wouldn't be capable of handling the whole network because it would still 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.

2.1 Hierarchical topology

Netsukuku adopts a hierarchical structure: 256 nodes are grouped inside a *group node* (gnode), 256 group nodes are grouped in a single *group of group nodes* (ggnode), 256 group of group nodes are grouped in a gggnode, and so on. (We won't analyse the topology of Netsukuku. You can find more information about it in the proper document: [2]).

Since each gnode acts as a single real node, the QSPN is able to operate independently on each level of the hierarchy.

Since in each level there is a maximum of 256 (g)nodes, the QSPN will always operate on a maximum of 256 (g)nodes. Therefore we would need just to be sure that it works as expected on every cases of a graph composed by ≤ 256 nodes. By the way, we'll directly analyse the general case.

For the sake of simplicity in this paper, we will assume to operate on level 0 (the level formed by 256 single nodes).

3 Tracer Packet

A *TP* (Tracer Packet) is the fundamental concept on which the QSPN is based: it's a packet which stores the ID's of the traversed hops in its body.

3.1 Tracer Packet flood

A TP isn't sent to a specific destination but instead, it's used to flood the network. By saying "the node A sends a TP" we mean that "the node A is starting a TP flood".

A TP flood passes only once through each node of the net: a node which receives a TP will forward it to all its neighbours, except the one from which it received the TP. Once a node has forwarded a TP, it will not forward any other TP's of the same flood.

3.2 Properties of the tracer packet

1. A node D which received a TP, can know the exact route covered by the TP. Therefore, D can know the route to reach the source node S (who sent the TP) and the routes to reach the nodes standing in the middle of the route.

   For example, suppose that the TP received by D is: $\{S, A, B, C, D\}$. By looking at the packet, D will know that the route to reach B is $C \to B$, to reach A is $C \to B \to A$, and finally to reach S is $C \to B \to A \to S$. The same also applies for all the other nodes which received the TP, i.e, B knows that the route to reach S is $A \to S$.

2. The *bouquet of* S is the set of all the TP's which will be forwarded or sent by the node S during the flood. The first TP of this bouquet received by a generic node D, will be the TP which covered the fastest route which connects S to D. The fastest $S \to D$ route is the route with the minimum *rtt* (Round-Trip Time) between S and D. This property is also valid if S is the node which started the TP flood, i.e. the first node which sent the first bouquet of the TP flood.

Example

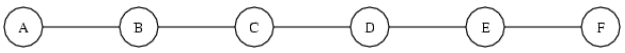

Figure 1: A simple graph

Suppose that D sends a TP. The TP will cover the routes: $D \to E \to F$ and $D \to C \to B \to A$. When the TP reaches the node F and the node A, the flood will stop because either A and F aren't able to forward the TP to any other node.

At the end, A will know the route $A \to B \to C \to D$ and F will know the route $F \to E \to D$.

3.3 Acyclic Tracer Packet flood

The Acyclic flood is not restricted like a normal TP flood: one or more ATP can pass from the same node. The end of the flood is given by this rule: a node will not forward the ATP to any of its neighbours if its node ID is already present in the route contained in the body of the packet. With this rule, an ATP can walk in a cycle only once, hence the name.

Finally, like in the normal TP, a node doesn't forward the ATP to the neighbour from which it has received the packet itself.

If every node of the network sends an ATP flood, then every node will get all the possible routes to reach any other node.

3.4 Continuous Tracer Packet

A Continuous Tracer Packet (CTP) is an extension of the TP flood: a node will always forward a TP to all its neighbours, excepting the one from which it has received the TP.

If a node is the extreme of a segment, i.e. a node with just one link, it will erase the route stored in the body of the TP and will forward back the TP.

In short, a CTP is a TP flood that will never end, thus it will continue to explore all the infinite combination of routes.

3.5 Interesting Information rule

This rule can be described in a single phrase:

A Tracer Packet will continue to roam inside the network until it carries interesting information.

A node considers a received TP interesting when its body contains at least a new route. In other words, if a TP contains routes already known by the node, it's considered uninteresting.

When a node receives an interesting TP, it forwards the packet to all its neighbours, excepting the one from which it has received the TP. If the TP is uninteresting, the packet gets dropped by the node.

Note that if a TP is uninteresting for the node N, then it is also uninteresting for all the other nodes. This is because an uninteresting TP contains routes which have been previously received, memorised and forwarded by the node N. Therefore all the other nodes already know the same routes too.

Example

Consider this graph.
Suppose that A sends a CTP. The two CTP's, after having covered the following

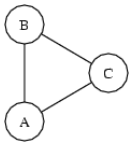

Figure 2: The A-B-C-A cycle

two paths will stop:

$$A \to B \to C \to A \to B \to C$$

$$A \to C \to B \to A \to C \to B$$

Let's analyze the first CTP step by step, considering that before A sent the CTP, none of the nodes knew any route.

A → B At this point B doesn't know any route to reach A, therefore it considers this CTP as interesting and forwards it to C.

A → B → C By looking at this packet, C learns a route to reach B and A.

A → B → C → A The node A learns a route to reach C.

A → B → C → A → B The node B learns a route to reach C.

A → B → C → A → B → C Finally, C drops the packet because it already knows all the routes contained in the CTP.

4 Network dynamics

The QSPN v2 defined until now is not suitable for dynamic networks.

4.1 Extended Tracer Packet

The ETP solves the problem of how the graph should be explored and re-explored to update the maps of the nodes interested in a network change. It's way of working is based on a simple observation:

When a change in the network occurs, only the information stored in the nodes affected by the change must be updated. The unaffected nodes will still have up to date information that they can simply redistribute with the use of the Extended Tracer Packets.

An ETP is an Acyclic Tracer Packet[2] which contains a portion of a map. Since a map is a set of routes and a route can be described by a TP, the ETP can be considered as different TP's packed together. Each TP of the ETP is then subjected to the Interesting Information rule (see 3.5).

In order to give an exact definition of the ETP, we must examine each case of network change.

Changed link Suppose that the quality of the link $A \overset{l}{\leftrightarrow} B$ changes, (it may improve or get worst).
Let's examine the events starting from A, keeping in mind that the situation is symmetric for B.

Since the link l changed, it may be the case for B to discover new optimal routes. For this reason, A will send to B an ETP with all the routes of its map, except those of the form $A \to B \to \ldots,$. If B finds something interesting, it will forwards the ETP. In detail:

1. A updates its map.

[2] An ATP (see paragraph 3.3) is a normal TP with the following rule: a node drops the received ATP if its node ID is already present in the route contained in the body of the packet. Note also, that since it is a normal TP, it is not reflected back, when it reaches the end of a segment.

2. A creates the following set:

   $R = \{r \in \overline{M} \mid \text{gw}(r) \neq B\}$

   where:

   $r \in \overline{M} \;\Leftrightarrow\; r \in M,\;\; rem(r) = \max\{rem(s) \mid s \in M,\; dst(s) = dst(r)\}$

   In other words, $\overline{M}$ is the set of all the best routes of the map

   M is the set of all the routes of the map of B

   $gw(r)$ is the first hop of the route r.

   $dst(r)$ is the destination of the route r.

   $\text{rem}(r)$ is the Route Efficiency Measure of r.

   If $R = \emptyset$ then this algorithm halts.
   Each route $r \in R$ is saved as $(\text{dst}(r), \text{rem}(r))$.

3. A creates the ETP:
   (a) A writes the set R in the ETP.
   (b) A appends its node ID, along with the efficiency value of the link l.
   (c) A sets to 1 the *flag of interest*.
4. A sends the ETP to B.

Suppose the node C has received the ETP from its neighbor N[3]. C will examine the ETP and, if considered interesting, updates its map and forwards the ETP to the other neighbours as follow:

1. If the ID of the node C is already present in the received ETP, then C immediately drops the ETP and skips all the following steps[4].
2. Let R be the set of routes contained in the ETP received by C.
   Let M be the set of all the routes contained in the map of C.
   For each route $r \in R$, the node C looks for a route $m \in M$ such that:

   $$\text{dst}(m) = \text{dst}(r),\;\; \text{gw}(m) = N$$

   If m exists, then C sets $\text{rem}(m) := \text{rem}(r)$ [5]. Otherwise r is copied in the temporary set R'.
   M is sorted, i.e. the routes of the map of C are sorted in order of efficiency.
3. For all $r' \in R'$, let $m' \in M$ be a route such that $\text{dst}(r') = \text{dst}(m')$. If r' is a better alternative[6] to m' and if every hop of r' is reachable by C, then r' is saved in the map of C as the tern $(dst(r'), N, rem(r') + rem(C \to N))$. The map is now ordered.
4. C creates the following set:

   $$S = \{s \in \overline{M} \mid \exists r \in R :\; \text{dst}(s) = \text{dst}(r),\; gw(s) \neq N\}$$

[3] C may be B itself and N may be A
[4] This is the acyclic rule
[5] With this operation, we are actually replacing m with r in the map M
[6] in the current implementation, r' is always a better alternative. In fact, in the previous step we've seen that we don't know any routes to $\text{dst}(r')$ through N. r' is a route to $\text{dst}(r')$ through N, so we memorise it. For more info, see Chapter 5 of topology.pdf[2]

5. If $S \neq \emptyset$
   (a) C creates a new ETP, appending the set S and its ID on it. The *flag of interest* of this ETP is set to 0.
   (b) The ETP is sent to N.
   This new ETP will propagate back in the same fashion of the previous ETP, i.e. until considered interesting. The only difference is when a node considers it uninteresting, it just drops[7] the ETP.

   The reason for the above procedure is simple: C considers uninteresting some of the routes contained in the received ETP, then C sends back its better routes which have the same destination of those belonging to R (hoping that they will be useful to the previous interested nodes).
6. Let
$$\overline{R} = \{r \in R \mid \nexists s \in S : \ dst(r) = dst(s)\}$$
   If $\overline{R} \neq \emptyset$, then the ETP is still interesting: the *flag of interest* remains set to 1. C packs the ETP with the set $\overline{R}$ and adds its ID. The ETP is sent to all its neighbours except N.
   Note [8]

The rnodes of C will use the same procedure. In this way, the ETP will continue to be propagated until is considered uninteresting.

Observation: Suppose the following propositions hold:

$\forall$ node N $\forall$ node D $\ C_D := \{\text{route } r \mid gw(r) \in V_N,\ dst(r) = D\}$
where V_N is the set of all the neighbours of N
$V(C_D) := \{gw(r) \mid r \in C_D\} \subseteq V_N$
Given $r \in C_D$, let $B(r) \in C_D$ such that $rem(B(r)) = \max\{rem(s) \mid s \in C_D,\ gw(s) = gw(r)\}$
$\overline{C_D} = \{B(r) \mid r \in C_D\}$
$\forall x \in V(C_D)\ \exists r \in \overline{C_D} \cap M : \ gw(r) = x \qquad (\gamma)$
where M is the map of N

If the property (γ) holds, then the step 5 can be omitted (in fact, suppose N sends an ETP to C and that C considers the route r contained in it as uninteresting).
By step 5, C sends back to N its known best route s, with $dst(s) = dst(r)$, $rem(s) > rem(r)$. However, s has been already sent by C to N (i.e. when the link $N \leftrightarrow C$ was established), thus by the property (γ), N kept memorised s in its map M. For this reason, the backpropagated ETP doesn't bring any new information. If the map described in the topology document[2] is used, then the property (γ) holds.

New link The case where a new link $A \overset{l}{\leftrightarrow} B$ is established is handled in the same way of the **Changed link** case, because we can consider l as a link

[7] The node will drop the ETP if the *flag of interest* is set to 0 and if it is uninteresting

[8] This step implements the Interesting Information rule: only good routes are kept, the other are discarded. Notice the extension: if the ETP had only one route, it would be almost equal to a CTP (the CTP doesn't have the acyclic rule)

with a new rem. However, there's just one difference with the changed link case: the algorithm does not stop if $R = \emptyset$. This is because, $R = \emptyset$ if and only if A is a node with only one link. Even if A has just one link, it has to send an ETP to advertise its existence to the other nodes of the network.

Broken link

1. B creates the set R:

$$R = \left\{ r \in \overline{M} \mid gw(r) = A \right\}$$

   In other words, R is the set of all the best routes passing through A. If $R = \emptyset$ then this algorithm halts.

   Each route $r \in R$ is saved as the pair $(\mathrm{dst}(r), \mathrm{rem}(r))$.

2. B updates its map:

$$\forall r \in M : \ gw(r) = A \quad \text{it sets } rem(r) := -\infty$$

   The routes are then sorted in order of efficiency.

3. B creates the following set:

$$\overline{R} = \left\{ r \in \overline{M} \mid \exists s \in R : \ dst(s) = dst(r) \right\}$$

4. B fills the ETP:
   (a) B adds in it the set $\overline{R}$.
   (b) B appends its ID.
   (c) B sets to 1 the *flag of interest*.
5. Finally, B sends the ETP to all its neighbours, except A.

At this point, the ETP is propagated in the same way of the **Changed link** case.

A new node joins Suppose the node A is joining the network. It's neighbours are $B_1, B_2, \ldots, B_n$, which are all already hooked, i.e. they aren't joining the net. Then, the **new link** case is applied to each link $A \leftrightarrow B_i, \quad i = 1, 2, \ldots, n$.

A node dies When the node A dies, the **broken link** case is simply applied to each link of A.

5 QSPN optimisations

5.1 Disjoint routes

The routing table of each node should be differentiated, i.e. it should not contain redundant routes.

For example, consider these $P \rightarrow S$ routes:

$$PBCFG_1G_2G_3G_4G_5G_6G_7 \dots G_{19}S \quad (1)$$
$$PRTEG_1G_2G_3G_4G_5G_6G_7 \dots G_{19}S \quad (2)$$
$$PZXMNO_1O_2O_3O_4O_5S \quad (3)$$
$$PQPVY_1Y_2Y_3Y_4S \quad (4)$$

The routes (1) and (2) are almost identical, they only differ only in the first three hops. On the other hand, the last two are totally different from all the others.

Since the first two routes are redundant, the node P should keep in memory only one of them, saving up space for the others non-redundant routes.

Keeping redundant routes in the routing table isn't optimal, because if one of the routes fails, then there's a high probability that all the other redundant routes will fail too. Moreover, when implementing the multipath routing to load balance the traffic, there won't be any significative improvements.

The following distributed mechanism is adopted: suppose that the node N receives a generic TP, suppose also that it contains a route to reach the node S. If N considers the TP interesting, then N will write in the TP the tern $a = (N, S, n+1)$, where n is the number of routes already known by N to reach the node S. In this way, the other nodes will know that the TP is the $(n+1)-$th route of N to reach S.

Now suppose that the node P receives the TP (the TP may or may not have passed through other nodes after having passed from N).

Let $T_S = \{(a, b, c) \in T \mid b = S\}$, where T is the set of all the terns written in the TP. P will calculate the following mean:

$$m = \frac{\sum_{t \in T_S} t_3}{|T_S|}$$

Where t_3 is the third element of the tern t and $|T_S|$ is the number of elements of the set $|T_s|$.

The number m indicates how disjoint is the route contained in the TP from previous routes. If $m \geq 2$, then the route is exactly the copy of a previous route. If $m \approx 2$, then the route is approximately a copy of a previous route. If $m = 1$, the route is definitely original.
Example: see figure 3.
If e_r is the efficiency of the route r, using the value m relative to r, we can obtain a new efficiency value:

$$e'_r = e_r(2 - m)k$$

Where k is an appropriate positive coefficient. The obtained efficiency value e'_r can be used by the Q^2 to decide if the route r is interesting or not.

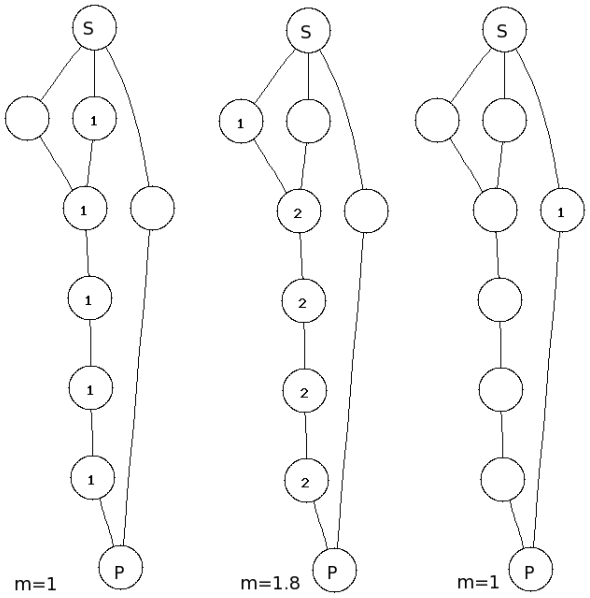

Figure 3: Example of the disjoint route mechanism

5.2 Cryptographic QSPN

A node could easily forge a TP, injecting false routes and information about links in the network. The attack would just create a temporary local damage, thanks to the distributed nature of the QSPN. However the optimal solution is to prevent these attacks.

Whenever joins Netsukuku, it generates a new RSA key pair. The continuous generation of keys prevents the leakage of the nodes' anonymity. The node shares its public key to all the other nodes of its group node.
Each entry appended in a TP is then signed with its private key, doing so the other nodes will be able to prove its authenticity and validate the path covered by the TP.

The size required to store the signatures in the TP can be kept costant using the aggregate signature system [9] [10].

6 TODO

1. Improve, test and implement the Caustic Routing: RFC 0013
2. Well define a "secure QSPN"
3. Research a "mobile QSPN"

7 ChangeLog

- `August 2007`
  1. "Real time issues" removed from the ETP section: it was superfluous.
  2. The whole section regarding the CTP has been removed, because until now the ETP is sufficient to handle all the routing issues.
  3. An observation regarding the step 5 of the **changed link** case of the ETP has been added.
- `July 2007`
  1. The ETP has been simplified: all the cases are now just a minor adaptation of a single case.
  2. Usage of the *tpmask* has been abolished.
  3. Simplified and distributed method to identify non disjoint routes.
- `April 2007`
  - New section: "Network dynamics" (4)
  - Description of the ETP (sec. 4.1)
  - Link ID section remove. With the ETP they are no more necessary.
  - More detailed description of the QSPN v1.
  - Subsection "QSPN v2 - High levels" removed. It was redundant with the topology document[2]
- `October 2006`
  Initial release.

References

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

[2] Netsukuku topology document: topology.pdf

[3] Depth-First Search: http://en.wikipedia.org/wiki/Depth-first_search

[4] Generate Routes in Awk: generate_routes.awk

[5] Simplify Routes in Awk: simplify_routes.awk

[6] Complete graph: http://mathworld.wolfram.com/

[7] Network simulator: http://www-mash.cs.berkeley.edu/ns/

[8] NTK_RFC 002: Bandwidth measurement

[9] A Survey of Two Signature Aggregation Techniques: http://crypto.stanford.edu/ dabo/abstracts/aggsurvey.html

[10] Aggregate and Verifiably Encrypted Signatures from Bilinear Maps: http://crypto.stanford.edu/ dabo/abstracts/aggreg.html

^_^