Ninux Roma

The Routing Architecture

September, 2013 - Version 1

Authors

- ZioPRoTo - Saverio Proto
- Nino - Antonino Ciurleo
- Clauz - Claudio Pisa
- claudyus - Claudio Mignanti
- Hispanico - Marco Giuntini
- ordex - Antonio Quartulli
- G10h4ck - Gioacchino Mazzurco
- SCO - Pierluigi Checchi
- AGGIUNGITI QUI SE LAVORI AL DOCUMENTO

Scope of this document

From 2003 to 2011 we run the Ninux Wireless Community network in Roma, in small scale and with private IPv4 addressing. Exchanging traffic with the Internet was done using NAT, and exploiting the user home ADSL lines.
In the very beginning we had a layer 2 network, then RIP routing, and finally since 2005 we started deploying the OLSR routing protocol.
Because we had a very simple network structure and running just a single routing protocol, up to now we never needed a routing architecture.
In 2011 we started to deploy a new Ninux network in Rome. Dual stack IPv4 and IPv6, and with BGP peerings to upstream providers.
Now the time has come to define a routing architecture.

Table of Contents

Requirements

The Ninux network of Rome is designed to be dual stack and to fully support native IPv6 and IPv4. Moreover we try avoid tunneling in the core network to guarantee the delivery to any Ninux site of the network a MTU of 1500 bytes, this way we never have fragmented IP packets in the backbone.
All the destinations are fully routable within our autonomous system. We do not make use of NAT inside the autonomous system.
Managing the IPv6 side is easier because we have to deal only with public addresses and deliver the traffic towards external autonomous systems just to our upstream providers.
The IPv4 requirements are more complex because we have a mixed public and private address space.
We use at any site private or public IPv4 addresses, and in terms of IP reachability we allow any of the following exchanges:

- any private IPv4 to any private IPv4
- any public IPv4 to any public IPv4
- any private IPv4 to any public IPv4
- any public IPv4 to any private IPv4

For what concerns the outgoing traffic towards other ASs we have as a requirement to exploit our upstream providers where we have BGP peerings, and also the bandwidth shared by Ninux members, who provide access to their home Internet connections.

Glossary

- Ninux Node: a Ninux Node is a set of devices on a roof managed by a Ninux member.
- Ninux Super Node: a Super Node is a Ninux Node that provides transit (i.e. is not a leaf of the topology graph)
- Public Ninux IPv4: It is a public IPv4 assigned to the Ninux AS.
- Backbone: In this document we refer to backbone as the sets of links between Ninux Super Nodes. These links are usually Point to Point (PtP) or Point to Multi Point (PtMP), with high bandwidth capacity and directional antennas.
- *In the Version 0 of the document you might find the Glossary is not complete*

Address Space

At the moment, Ninux has been given the autonomous system number 197835, as stated in https://stat.ripe.net/AS197835.
To better exploit the features of this architecture at least a /48 IPv6 network and a /24 IPv4 public network are needed. However the network could be deployed also with fewer resources.
For IPv4 any private IP address defined in the RFC 1918 can be used. However if you plan to build a Ninux network in your city and you want to be able in the future to connect via VPN to our services, we suggest to write to the Ninux mailing list to define a subnetting pattern for your city that avoids collisions with the address space in the rest of the country.

IPv4

In Ninux Roma we currently use a /24 public network and various private networks from RFC 1918, however the architecture presented in this document is suitable for any address space.
The radio interfaces of the backbone have an IPv4 address in the 172.16.0.0/16 private space. The third byte of this IP address is chosen to be close to the postal code of the geographic address of the Ninux node.
So we define this address space: 172.16<.ZIPCODE>.x. Where x is allocated sequentially by checking on our database which is the last IP address already allocated.
The LAN interfaces use the 10.0.0.0/8 address space. Again usually the Ninux Node obtains a /24 trying to follow this pattern: 10.<ZIPCODE>.x.0/24. The rules to follow this pattern are the same for the backbone interfaces.
We have some legacy Ninux nodes that do not follow this scheme, and may use also 192.168.0.0/16 addresses. This is not a problem unless there is a IP address collision.

To compute the <ZIPCODE> value, we start from the Italian zip code postal address, that is a number of 5 digits, and we apply the following formula:

```
if zipcode is XXYYY
then compute (YYY mod 255 + XX ) mod 255
```

The Ninux Public IP addresses are not auto managed with an algorithm as it is for the private address space. If a Ninux member needs a public IPv4 he needs to request it to the community and the IP address will be assigned to him. The public ninux IPv4 address will be assigned considering where the node of the member is locate, to avoid asymmetric traffic with the BGP gateways (more later).

IPv6

The Ninux Network in Rome uses the IPv6 address space 2001:4c00:893b::/48. Every Ninux member allocates for the LAN at his place one or more /64 networks. It is important to delegate an entire /64 to a Ninux Node because most IPv6 software implementations expect the LAN to be a /64 subnet. By delegating a smaller IPv6 space you will run in problems when assigning dynamically IPv6 addresses to your network hosts.

The OLSR protocol has a nice feature: because it was originally devised to support the mobility of the nodes, the signalling works as long as there is a working layer 2 connection between two routers. Because the OLSR packets have a multicast destination address, both the source IP address and the subnet mask of the OLSR router are not important for the correct behavior of the signalling. This means that we could have routers with a completely random 128bit sequence instead of a coherent IP address and the OLSR protocol would be able to make its routing properly. The radio interfaces of the backbone have IPv6 addresses inside this /64 network

2001:4c00:893b:1::/64

By default we use this convention:

2001:4c00:893b:1:<ZIPCODE>::X

Where <ZIPCODE> is the same value explained for IPv4 and X is the same 8 bit integer as the last IPv4 octet. However because the IPv6 address space is bigger than IPv4 and many nodes can be in the same ZIPCODE area, we do not mandate to strictly use ZIPCODE so we have several exceptions to this rule. The mandatory is that radio backbone interfaces MUST be in the 2001:4c00:893b:1::/64 subnet.

In summary the only reserved /64 subnets are 2001:4c00:893b:1::/64 and 2001:4c00:893b:ffff::/64, so any other free /64 network can be choosen by a Ninux member for his home LAN.

Conventions in Addressing

There are some conventions that we use to easily managing numbering point to point links.

VPN address space:

The Ninux VPN as explained later is only IPv4. As a convention we chose 10.0.1.X for the Rome VPN and 10.0.5.x for Ninux Islands VPN connection addressing. These services will be described later in Section X

GRE Tunnels Point to Point Links:

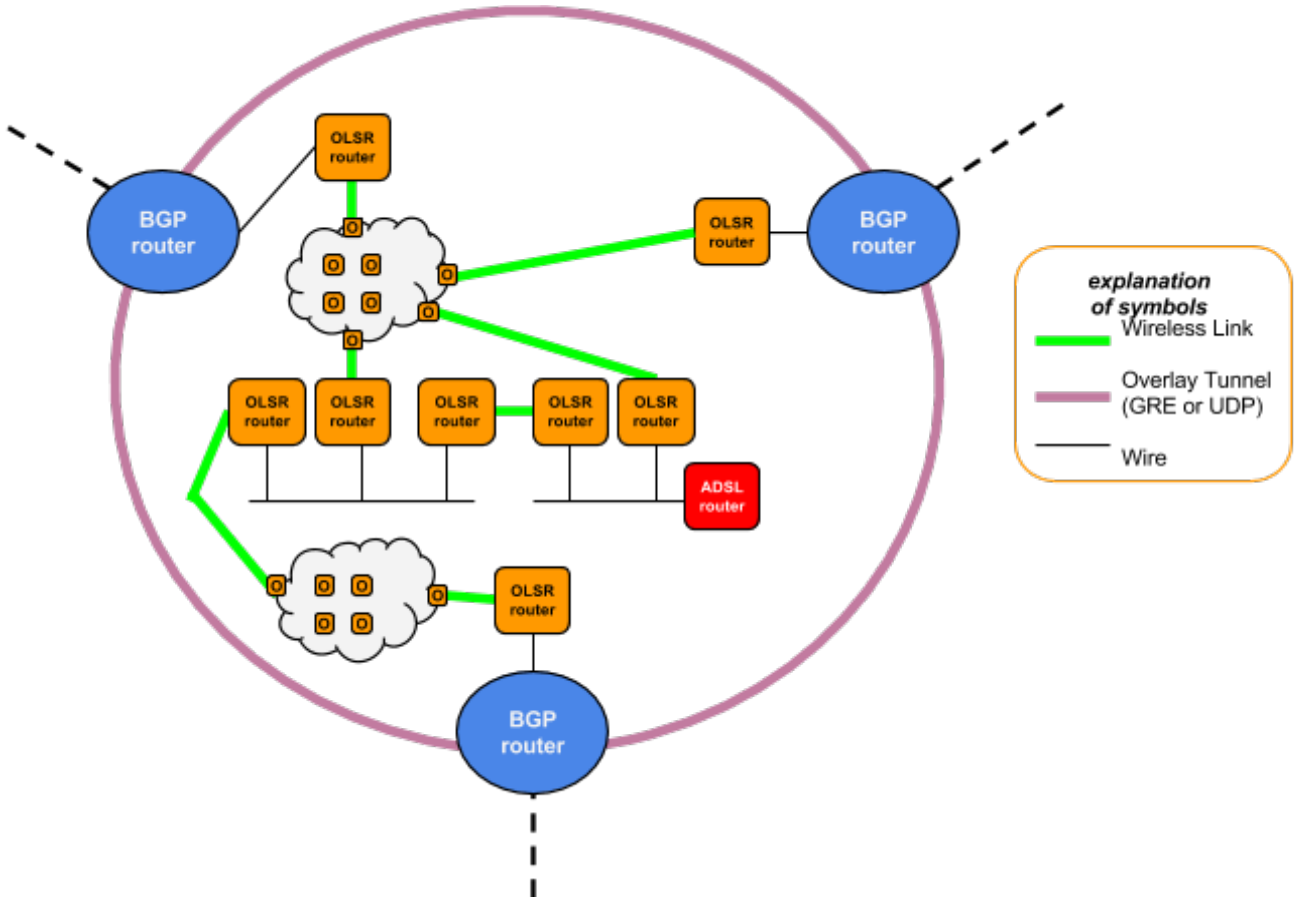
For full mesh iBGP tunnel we use, for IPv4, 10.0.3.0/24 subnetted in /30, one for each tunnel. IPv6 tunnel address are /127 (rfc6164) subnets taken from 2001:4c00:893b:ffff::/64

iBGP Peering VPN

The iBGP peering session can be built on top of a overlay VPN based on tinc-vpn instead of GRE tunnels. In this case we still use for IPv4 the 10.0.3.0/24 network and for IPv6 the 2001:4c00:893b:ffff::/64 but there is no need for subnetting in point-to-point networks because the VPN technology gives the abstraction of a single LAN.

Network Topology

In the following figure we show a simplified topology of the Ninux Network.
For the complete topology of the network of Rome you can check out the web site http://map.ninux.org .
We show three BGP routers, as in the Ninux Network of Rome, however this architecture works with any number of BGP routers.

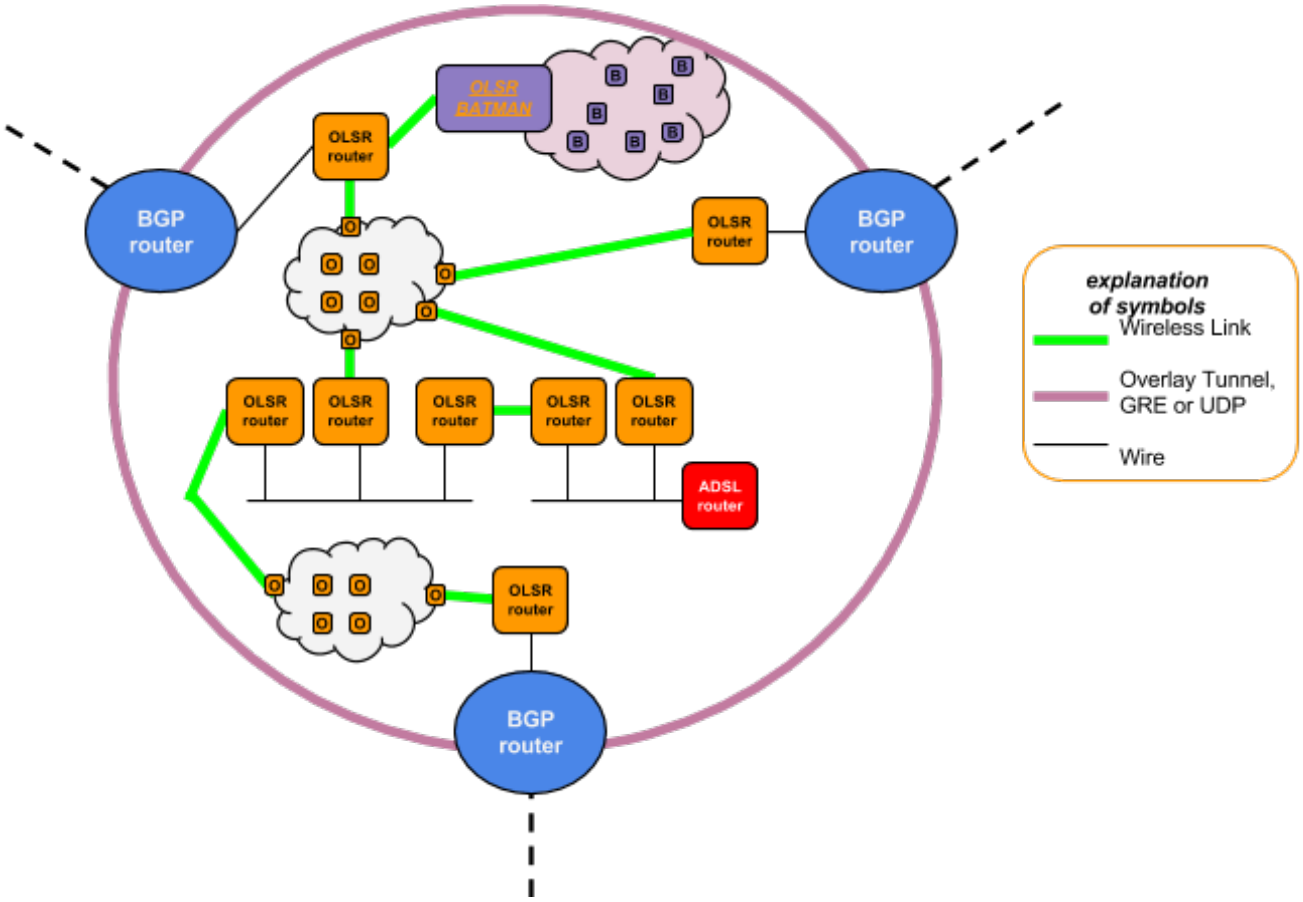


OLSR routers, as we will explain more in deep later, are the devices that the community members have on the roof of their houses. On the same roof there might be more than one OLSR router, but at least one must be present to run a Ninux Node. OLSR routers can be connected to other routers via both wireless and wired links. Links are not mandatory to be point to point. In the case of wired links we have an Ethernet segment, where all the routers that are connected will establish an adjacency to all the others in the segment. In wireless links, even if we prefer to have point to point links to reach better performances, we can have any kind of L2 configuration (AP-STA or ad-hoc for wireless) and the routing protocol will take care of

establishing the router's adjacencies.
In the figure we depict a cloud of routers, it means that these devices route traffic just by being OLSR speakers.
Two Ninux nodes are represented in higher detail. The one in the left is a roof with three routers on the top, and no Internet connection available in that site. The one in the right is a roof with two OLSR routers, where the Ninux member is also sharing her ADSL bandwidth.

Mesh stub areas

The usual OLSR router has an interface connected to the user LAN, in a subnet with some hosts that do not run the OLSR protocol. To announce the IP addresses of the LAN in the OLSR protocol we use the HNA configuration setting in OLSR.
Of course the HNA configuration setting can be exploited to announce a subnetwork that is not a normal LAN, but it is a mesh network where we run a L2 routing protocol, for example B.A.T.M.A.N.-Advanced.

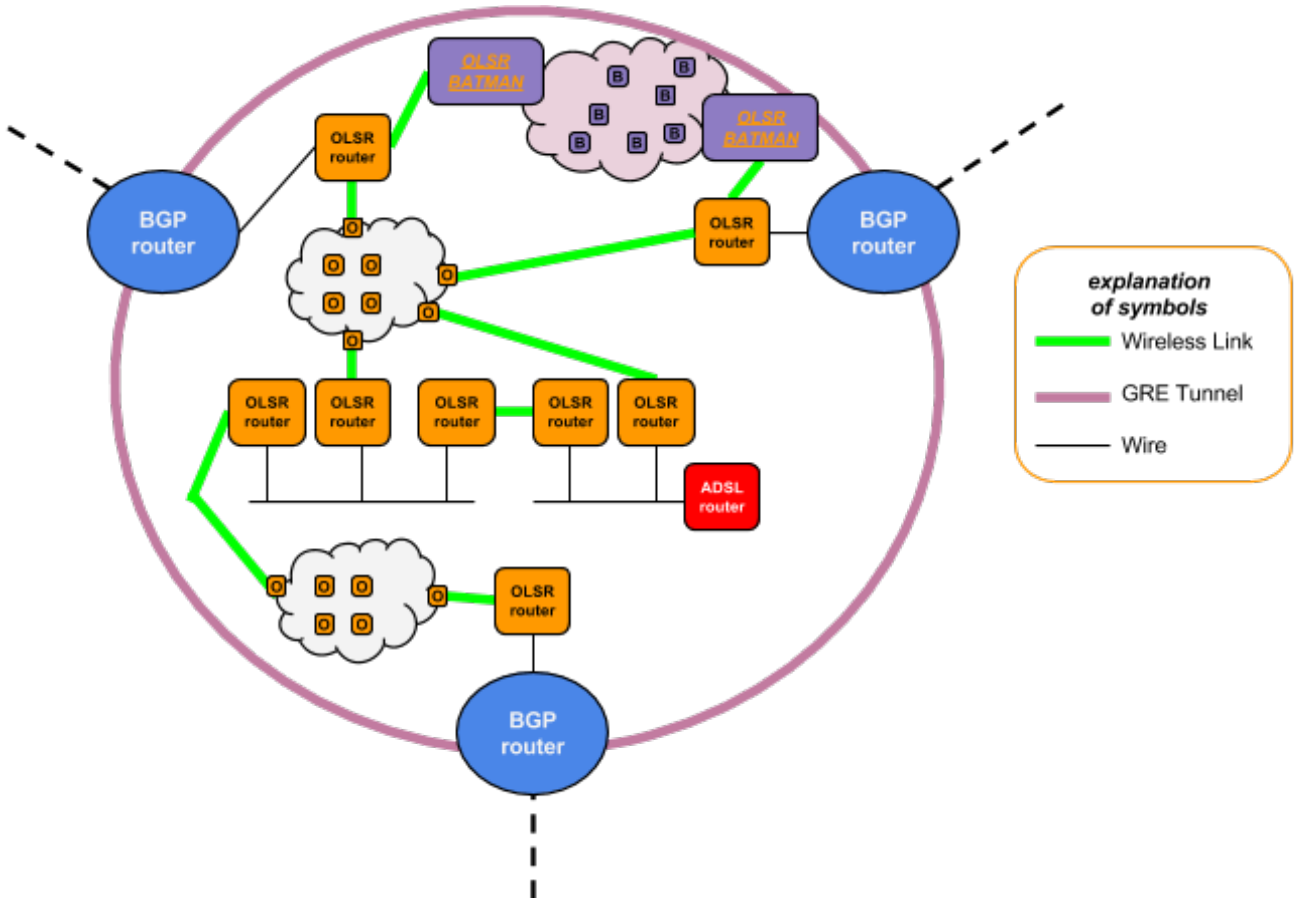


As we see in the figure there will be an edge router between the two domains that speaks both the OLSR and the B.A.T.M.A.N.-Advanced routing protocol. This edge router will announce all the IP addresses used in the batman-adv network with an HNA setting.

Note that is not mandatory to have a single router speaking both protocols. The setup will work also with an additional device connected to the HNA segment of the OLSR router that speaks the B.A.T.M.A.N.-Advanced routing protocol and makes the connection from the HNA domain and the batman cloud.
On the other “side” the edge router will run a DHCP server instance and will use it to distribute addresses to all the batman-adv nodes belonging to the L2 mesh network (the LAN). In this way, the nodes will directly exploit the OLSR edge router as default gateway towards the rest of the network without caring about any L3 routing issue. Here it is easy to understand that the batman-adv mesh network is exactly behaving like a normal LAN connected to a classic OLSR router.

Mesh transit areas

If the mesh network connects to the backbone in more points, then we don’t have a mesh stub area anymore. Note that the mesh address space is announced to the Ninux Backbone via HNA in both the edge routers. Moreover the two edge routers will establish a OLSR adjacency that traverses some hops in the batman-adv mesh network. This happens because OLSR is not aware of the batman-adv mesh topology and about the number of hops it has to traverse, therefore the weight of this “single OLSR-link” should be manually adjusted by the Ninux Members running this part of the network.



For what concerns the batman-adv nodes, as for the stub area, they do not have any knowledge about the L3 routing issue, therefore they will continue to use as default gateway the edge router assigned by the DHCP protocol.

Depending on the Ninux members, all the edge routers could have their own DHCP server up and running, therefore the batman-adv area would end up in having multiple servers assigning DHCP addresses and default routes. To solve this issue, batman-adv implements a convenience feature called “Gateway Selection”: with this mechanism each and every broadcast DHCP request issued by any node in the area is directly redirected to the best gateways in the area. If you enable this not standard feature of batman-adv you have to make sure that every DHCP server is managing an IPv4 range, and these ranges must be not overlapping. The best gateway is chosen using the same metric used by batman-adv for the routing process so ensuring that each node will pick the most reliable one. In this way each node will contact only one of the DHCP servers that are present in the area and will use the default route assigned by it. In a general scenario each GW running a DHCP server will assign itself as default route for its clients.

Routers specification

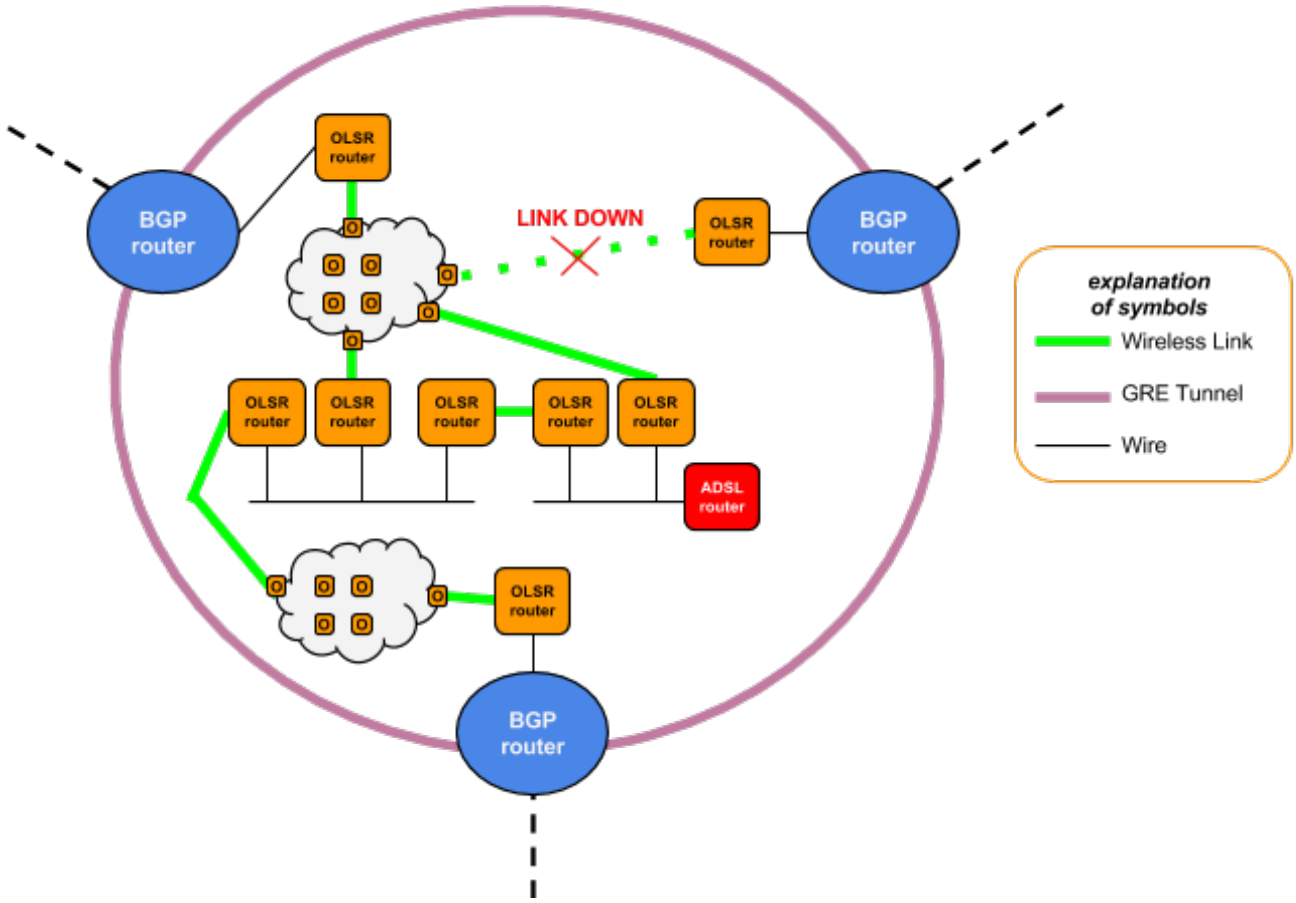
This architecture assumes that all the routers in the Ninux network are Linux based (running AirOS or OpenWRT systems) with a Kernel capable of managing multiple routing tables.
It is by using multiple routing tables that we are able to build up a complex IPv4 routing to meet our requirements.
Note that the olsrd daemon is natively able to use multiple routing tables for its operations. In particular olsrd will write all the hosts routes in a table and will write his best default route into a different table. We'll see later on how to exploit this feature.

BGP routers

The BGP routers in the proposed architecture are located only at the edge of the autonomous system. The routers are connected to each other forming a full mesh made by IP overlay tunnels and iBGP is handled with a full mesh of peering. The IP tunnels can be implemented using GRE tunneling or UDP tunneling with any VPN software. The key is that the tunnels are built over the ninux mesh network using OLSR routed addresses. This configuration guarantees that if there is a working path between two BGP nodes, their tunnel is up.
Note that using GRE could lead to a configuration bottleneck, because when deploying a new BGP router you need to configure a tunnel to each one of the already existing routers, allocating a /30 and /127 network for each tunnel. The BGP routers will result in having a virtual GRE interface of every configured tunnel. If you decide to use tinc-vpn as overlay technology each router will have a single virtual interface that is attached to a virtual LAN were all the other iBGP peers are present.
We do not expect the number of our BGP routers to grow too much, because we place them only at the edge of the autonomous system. However this architecture can be extended with more scalable iBGP schemes (for example a route reflector cluster) if the number of BGP routers gets too big.
The BGP router must be Linux based, because we need to run the olsrd daemon on the router to redistribute our IGP routes to BGP.
In our implementation we use Quagga version **0.99.21** with the "manet" patch and the olsrd daemon version 0.6.5.2 (or later) with the quagga plugin.
Because the Ninux backbone is designed with redundant paths, it should never happen that the Autonomous System becomes partitioned. If this happens unpredictable routing decisions may be taken for what concerns the traffic to the Internet.

Export Policy

The BGP router is however connected to the Ninux backbone with a single link, so there is the necessity to handle the case where the BGP router is disconnected.



As we see in the figure, we want to handle the scenario in which the BGP router loses its connectivity to its OLSR 2-hop neighbor because the wireless link is down. In this case we need the BGP router to stop announcing upstream the public IPv4 and IPv6 routes of Ninux.
To do this we configure two static routes, one for IPv4 and one for IPv6, to route all the public Ninux addresses to a next-hop that is the address of the 2-hop neighbor.
This static route is active and then redistributed in BGP only if there is a OLSR route to correctly resolve the next-hop, that we configured as the OLSR 2-hop neighbor.
With this trick we achieve to have in quagga something similar to the route tracking feature of the cisco routers.
Note that if a BGP router has multiple 2-hop neighbor this setup will still work adding multiple next-hops for the static routes that we are defining.

Import Policy

The minimal requirement is to receive from the upstream a default route via BGP. However because we like to experiment we receive from the upstream a full routing table.

Regarding the redistribution of the default route to the OLSR network, **the BGP routers announce in the OLSRd process (via HNA announces) two special prefixes that are: 0.0.0.0/1 and 128.0.0.0/1**. The sum of these two prefixes is all the IPv4 Internet, so why aren't we just announcing the default route? We announce these two more specific routes together with the default one because we want this information to go into a particular routing table in all the other OLSR routers (more later).
It is important also to distribute in OLSR the classical default route, in addition to the two /1 aggregates, because we will see later that this piece of information will be stored in a specific table of the OLSR routers that we use only for the default.
The BGP routers use a hot potato (or early exit) policy. Once traffic is there we send it on the upstream. It makes no sense to send the traffic to another BGP router within the GRE tunnel, maybe 15 wireless hops away, just because of a better AS path.
However, if for some special destination we have a policy to exit the Ninux network from a specific upstream provider, remember that traffic routed with a route learnt via BGP must always be routed into a GRE tunnel or to a upstream provider. This is mandatory because the next-hop must be a BGP speaker to have a consistent routing.

OLSR routers

This type of router is the most common in the Ninux Network. It runs the OLSR routing protocol and it makes use of Policy Based Routing (PBR) with the following routing tables.

IPv6Table:	IPv6 routing table, learnt via OLSR
RtTable:	olsrd IPv4 routing table
RtTableDefault:	IPv4 default route learnt via OLSR
RtTableLowPrefix:	IPv4 low prefix like 0.0.0.0/1 and 128.0.0.0/1 used for special operations
BlackholeTable:	IPv4 routing table with the blackhole rules
LocalTable:	IPv4 networks directly connected to the router
Main:	IPv4 main Linux routing table. Routes applied via web interface go here

Please refer to this document for better comprehension of naming the tables:
http://olsr.org/git/?p=olsrd.git;a=blob;f=files/olsrd.conf.default.full

Routing operation inside the OLSR router

The OLSR router is the key element of our routing architecture. The rules to choose which will be the routing table where to process the traversing IP packet, form the core part of all the network routing process.

If packet is IPv6: use IPv6Table

If packet is IPv4:
1) First match LocalTable
2) If destination is private: use RtTable

3) If destination is Ninux Public IPv4: use RtTable
4) Use blackhole table
5) If source is Ninux Public IPv4: use RtTableLowPrefix
6) If source is Transit Public IPv4: use RtTableLowPrefix
7) Anything else use Main and then RtTableDefault

TODO: textual description of the algorithm. Ripetere che annunciamo le /1

Avoid ghost traffic to gateways

What will happen if a host starts to generate packets to a destination that doesn't exist, that is in private address space, or in the Ninux public address space? This packets will follow the default route and will die into an Internet Gateway.
To avoid this useless traffic we insert some special static routes in the Blackhole Table. This routes will match the aggregate of the private IPv4 space (RFC 1918) and will match the aggregate of all the public IPv4 Ninux space. As a matter of fact if the host or subnet do exist, there is always a more specific route that is matched in the RtTable, this special routes that are inserted before the default route in the main table have the effect of stopping traffic towards not existent destinations that was traveling in the direction of the default gateways. Practically speaking we are talking about adding these routes:

```
ip route add blackhole 10.0.0.0/8 table 114
ip route add blackhole 172.16.0.0/12 table 114
ip route add blackhole 192.168.0.0/16 table 114
ip route add blackhole 176.62.53.0/24 table 114
```

Please note that we are blackholing aggregates! In the *LocalTable* you will always have a subnet before the blackhole route of the aggregate because the local subnet have a longer prefix. For example if your home subnet is 10.40.0.0/24 you will have this directly connected route always before the blackhole route to 10.0.0.0/8 because the /24 has a longer prefix. You should not blackhole a route of a network that you are using completely. If you get a new subnet, for example 176.62.54.0/24, and you use this network in a single site, then on this router you should not blackhole with route in the main table, because you cannot think of this network as an aggregate anymore.

Avoid ADSL Blackholes

If the OLSR router sends traffic to a default gateway defined in RtTableManual, but the ADSL connection is a blackhole, the traffic gets lost.
To prevent this situation the router announces the default via OLSR only if the connection through the ADSL modem is working.
To be able to detect this situation we define a *detection_address* address and a routing table *detection_table* so that:

- *detection_table* has always higher priority than the other tables
- *detection_address* is a public IPv4 address

- *detection_address* belongs to a host that has high availability
- *detection_address* belongs to a host that is ICMP enabled
- *detection_table* contains an entry for detection_address so that it is always routed through the ADSL line

Then we use olsrd dynamic Internet Gateway plugin and our bash script (*adsl_check*):

- the olsrd dynamic Internet Gateway plugin is configured to announce the default route only if *detection_address* is available
- *adsl_check* inserts the static route with gateway the ADSL modem as default route in RtTableDefault if *detection_address* is available and deletes the same static route if *detection_address* is not available

Note that more than one *detection_address* can be configured, as long as the above characteristics apply.

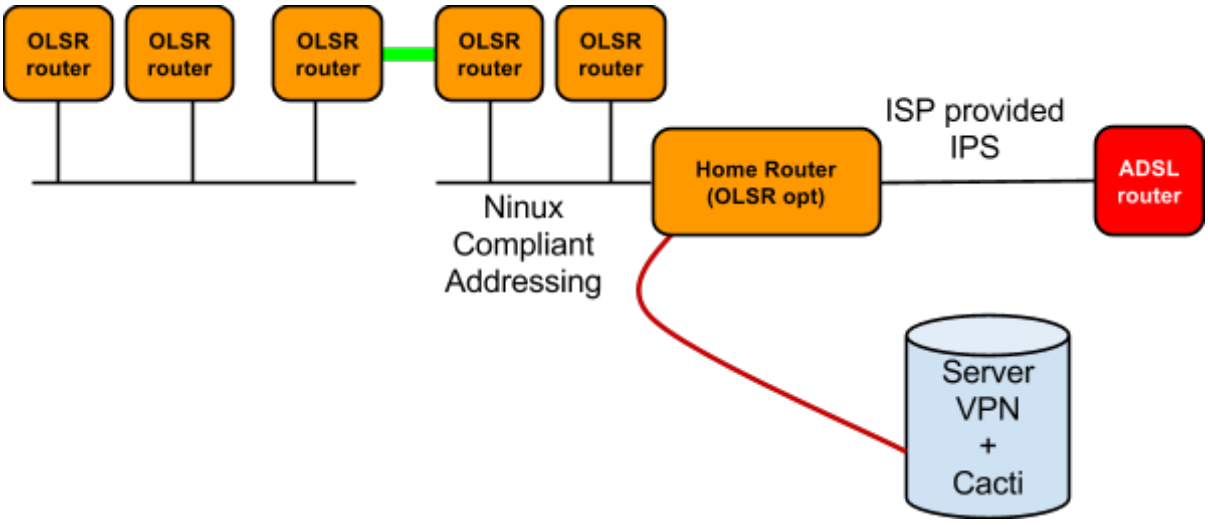
As an alternative, the OLSR protocol injection plug-in can be used. TBD

ADSL routers

This is the home Internet Gateway of a Ninux member, if any.
There are a few requirements that ADSL routers in our specification should match:

1. Possibility to specify the IP address
2. Possibility to add static routes to a gateway connected on the LAN interface

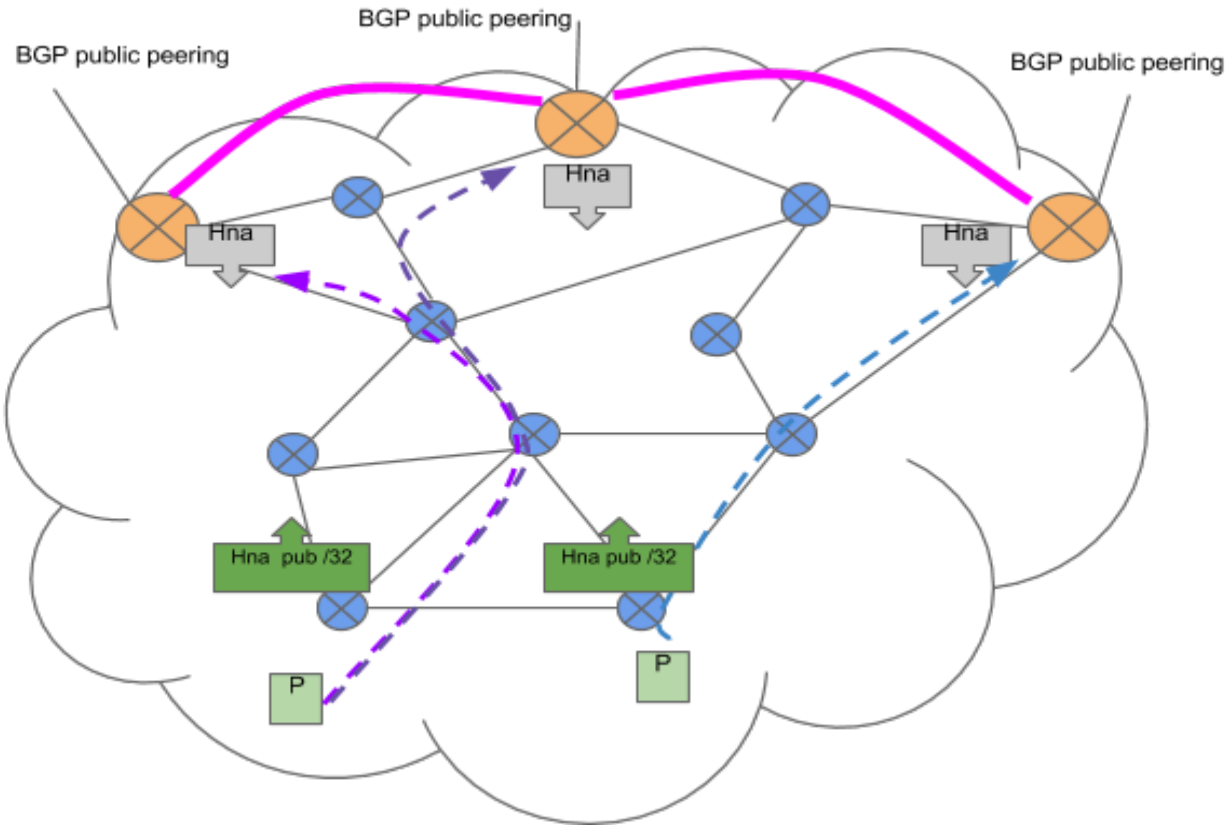
Note that the router that is provided by some Internet Service Provider might not be of use, i.e. the routers by Alice Telecom Italia have a fixed IP address 192.168.1.1 that cannot be changed. Also the Fastweb ISP architecture may not be compliant with the Ninux ADSL router requirements: addresses in the private IP address range have been provided to customers (and the range itself is not configurable by the users).
In both these cases an intermediate device, the “home router”, which must have an “ISP” and “Ninux” interfaces, implements a “double NAT” system were both ISP and Ninux address space are masqueraded.
Alternatively, we can configure an alias interface on the ethernet (say, 192.168.1.254 on the eth0:1) of the roof OLSR router and use the directly connected IP to route to the fixed unchangeable default gateway.
The traffic on the “home router” can be also monitored using compliant protocols such as SNMP and can be used as VPN host, in conjunction with an olsrd instance (see below).



Another further approach is to use the smart gateway feature of OLSR, that is capable of “sensing” an ip address on internet in order to inject the default route on HNA4 and that permit to choose “smartly” the default gateway to use for internet traffic, based on bandwidth/nat/ preference requirements. This is therefore a very cool approach, being able to tolerate for a few more hops the hotness of the potato. We are still experimenting. Issues are mainly the necessity to tunnel the traffic via IPIP, the MTU clamping hysteria, the stability of the route announcements, the tcp session disruptions in case of gateway change... and so on.

Anycast GRE tunnels

The legacy OLSR routers of the Ninux network do not have Policy Based Routing configured. This means that before all routers are upgraded we cannot rely on the network for proper routing of the Ninux public addresses. The problem comes when a packet with source address a public IP address of Ninux, and as destination address a general Internet ip destination, is captured by a ADSL gateway because matching a default route.
Moreover starting to announce the /1 prefixes via HNA has a magnetic effect of traffic to the BGP routers, if many routers are not yet upgraded.
As a transition mechanism we introduce anycast GRE tunnels to transport public IP addresses. The anycast



Server side
In the olsr.conf file make sure you are announcing the anycast address

```
Hna 10.0.162.1 255.255.255.0
```

Server site bash script:

```
#!/bin/bash
greifname=$1
ip addr add 10.0.162.1/32 dev lo
ip tunnel add $greifname  mode gre local 10.0.162.1 ttl 255
ip link set $greifname up
```

tunnel script: example: ./bgpsidescript.sh ninuxpub

Client side
olsr.conf (on nearest olsr node):

... Hna <assigned pub ip> 255.255.255.255 ...

client script:

```
#!/bin/bash

greifname=$1
local_ip=$2
pub_ip=$3
remote_ip=10.0.162.1
ip tunnel add $greifname  mode gre remote $remote_ip local $local_ip
ttl 255
ip link set $greifname up
ip addr add $pub_ip dev $greifname
ip rule add from $pub_ip table 115
ip route add default dev $greifname table 115
```

usage example: tunnel_gre.sh ninuxpub <priv_ipv4> <assigned pub ip>/32

NAT - Network Address Translation

Apart from the situation in which a “double nat” device is needed to separate the Ninux network from the provider’s private IP address space, there is only one situation in which we have to use NAT in the Ninux network: when a IPv4 packet with a private source address is going outside the Ninux Network. This can happen both at the ADSL router or at the BGP router. A NAT must be properly configured to handle this situation.

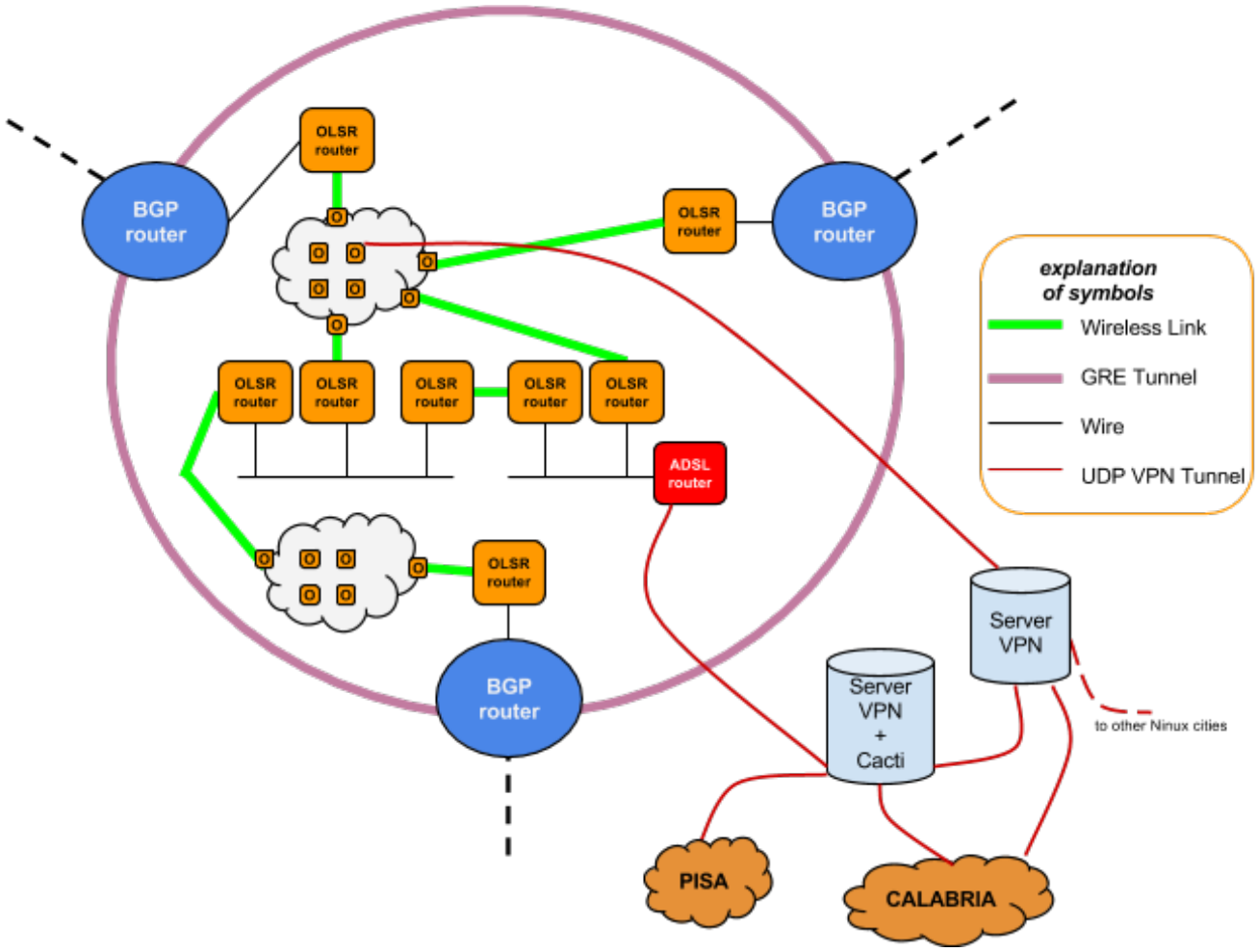
VPN: OOB, Monitoring and Connection to other cities

In the IPv6 Internet all the Ninux cities can exchange traffic using the general Internet routing. But for legacy IPv4 services using private IPv4 addresses we deployed a VPN to have a consistent routing among all the Ninux cities.

There are two different VPN scenarios in place.
RomeVPN. It is a tinc VPN between the nodes of Ninux Roma. We speak OLSR on this VPN and makes possible to have virtual backup links in case the AS is partitioned.
The VPN Links are part of the OLSR domain, the weight of the links is altered with the `LinkQualMult` setting of olsrd.conf to use VPN Links only if no other route to the destination is available.

Islands Ninux VPN: it is a tinc VPN dedicated to the traffic to the other Ninux cities.
On this VPN we use the babel routing protocol and OLSR with the protocol injection plugin.

In the Rome network we deploy this VPN as a tool for troubleshooting and monitoring. The troubleshooting part is in place in case several wireless backbone links begin to fail: the AS will not be partitioned if there are VPN links that keep all the nodes connected in the routing.
We do monitoring (traffic statistics and Nagios) with a single server for all the cities in Italy. This server accesses the various cities exploiting the VPN links.
In this architecture the Rome VPN is meant for signalling (SNMP, Nagios checks) and troubleshooting and **never for carrying actual data traffic**.
At the moment we make use of tinc-vpn to create the overlay topology among the border routers



Migration from legacy Ninux to this routing architecture

What happens when we deploy this architecture? Do we have to reflash all the routers at once with the new firmware?
The answer is no. The new OLSR routers are backward compatible with the legacy OLSR routers that use a single routing table. When the network is mixed, the only traffic that can have problems are IP packets with a Public Ninux source address. If these packets are routed by mistake to ADSL lines by a legacy OLSR router, the results are unpredictable, depending on the ADSL provider's configuration.

Conclusion

This is the very first version of this document where we describe the Ninux Routing Architecture. The deployment of this architecture should be complete by the end of 2013. We are going to release future versions of this document during the roll out of the network, adding templates for configuration files and fixing mistakes we find on the way. If you want to get in contact with us you can send an e-mail to contatti@ninux.org.

Appendix: OLSR router configuration template

To implement Policy Routing this is a template script:

```
#110 Local routes
#111 RtTable
#112 RtTableDefault
#113 Special Table for /1
#114 blackholes table

#Copy local routes only from table main 254 to table 110
ip route show table 254 | grep -Ev ^default | grep -Ev ^blackhole |
while read ROUTE ; do
ip route add table 110 $ROUTE
done

#First evaluate local routes
ip rule add from all lookup 110 pref 3

#Private routes to OLSR table
ip rule add to 10.0.0.0/8 table 111 pref 4
ip rule add to 172.16.0.0/12 table 111 pref 4
ip rule add to 192.168.0.0/16 table 111 pref 4

#Ninux IP Addresses to OLSR table
ip rule add to 176.62.53.0/24 table 111 pref 4

#Evaluate blackholes
ip rule add from all table 114 pref 5

#Send traffic of public addresses to BGP border routers
ip rule add from 176.62.53.0/24 table 113 pref 6
```

```
#Lookup default route first from user and then from OLSR
ip rule add from all lookup 254 pref 7
ip rule add from all lookup 112 pref 8

#Blackhole private aggregates
ip route add blackhole 10.0.0.0/8 table 114
ip route add blackhole 172.16.0.0/12 table 114
ip route add blackhole 192.168.0.0/16 table 114

#Blackhole Ninux aggregate
ip route add blackhole 176.62.53.0/24 table 114
```

The template of olsrd.conf for IPv4 is the following:

```
DebugLevel  0
IpVersion 4
Pollrate  0.025
FIBMetric "flat"
RtTable 111
RtTableDefault 112
UseNiit no
SmartGateway no
#Hna4
#{
#10.xxx.0.0 255.255.255.0
#}
UseHysteresis no
TcRedundancy  2
MprCoverage 7
LinkQualityLevel 2
LinkQualityAlgorithm    "etx_ff"
LinkQualityAging 0.05
LinkQualityFishEye  1

LoadPlugin "olsrd_txtinfo.so.0.1"
{
   PlParam     "port"   "2006"
   PlParam     "Accept"   "0.0.0.0"
}

LoadPlugin "olsrd_mdns.so.1.0.1"
{
 PlParam     "NonOlsrIf"  "eth0"
```

```
 PlParam      "MDNS_TTL"    "20"
 PlParam      "TTL_Check"   "true"
 PlParam      "Network_ID" "1"
 #PlParam      "FilteredHost" "192.168.0.1"
}


InterfaceDefaults {
   HelloInterval 3.0
   HelloValidityTime 125.0
   TcInterval 2.0
   TcValidityTime 500.0
   MidInterval 25.0
   MidValidityTime 500.0
   HnaInterval 10.0
   HnaValidityTime 125.0
}

Interface "ath0" "eth0"
{
    Mode "mesh"

    # LinkQualityMult 192.168.0.1 0.5
    # LinkQualityMult default 0.8
}
```

The template of IPv6 is the following:

```
DebugLevel  0
IpVersion 6
Pollrate  0.025
FIBMetric "flat"
UseNiit no
SmartGateway no
#Hna6
#{
#2001:face:b00c:b00d:: 64
# :: 0
#}

UseHysteresis no
TcRedundancy  2
MprCoverage 7
```

```
LinkQualityLevel 2
LinkQualityAlgorithm    "etx_ff"
LinkQualityAging 0.05
LinkQualityFishEye  1

LoadPlugin "olsrd_txtinfo.so.0.1"
{
   PlParam     "port"   "2007"
   PlParam     "Accept"   "::"
}

LoadPlugin "olsrd_mdns.so.1.0.1"
{
 PlParam     "NonOlsrIf"  "eth0"
 PlParam     "MDNS_TTL"   "20"
 PlParam     "TTL_Check"  "true"
 PlParam     "Network_ID" "1"
 #PlParam     "FilteredHost" "2001::1"
}


InterfaceDefaults {
   HelloInterval 3.0
   HelloValidityTime 125.0
   TcInterval 2.0
   TcValidityTime 500.0
   MidInterval 25.0
   MidValidityTime 500.0
   HnaInterval 10.0
   HnaValidityTime 125.0
}

Interface "ath0" "eth0"
{
    Mode "mesh"

    IPv6Multicast FF02::6D

}
```

Appendix: OLSR Troubleshooting

We expect txtinfo plugin to be accessible by anyone on our routers, on port 2006 for IPv4 and on port 2007 for IPv6.
Using this tool we can access the OLSR routing table, links information etc etc, so that any Ninux Member can perform troubleshooting on the backbone without need of passwords.
https://github.com/ninuxorg/misc_tools/blob/master/ninux-lg.py

Appendix: BGP router configuration template

First of all you have to tune the Linux Kernel parameters to store enough routes for a full routing table of the Internet.

```
echo 32768 >/proc/sys/net/ipv6/route/max_size
echo 8192 >/proc/sys/net/ipv6/route/gc_thresh

sysctl -w net.core.wmem_default=1048576
sysctl -w net.core.wmem_max=1048576
```

Download quagga sources and apply this patch:
https://dev.openwrt.org/browser/packages/net/quagga/patches/120-quagga_manet.patch

Or use our git repository:
https://github.com/ninuxorg/quagga-manet.git

On the BGP router also OLSR is running, make sure you enable the quagga plugin with the following configuration

```
LoadPlugin "olsrd_quagga.so.0.2.2"
{
      PlParam "ExportRoutes" "only"
      PlParam "SockPath" "/var/run/zserv.api"
      PlParam "Version" "2"
}
```

This is a template for the `zebra.conf` file. Values to be changed are in bold.

```
hostname Router
password yourpassword
enable password yourenablepassword
log file /var/log/quagga/zebra.log
interface eth0
 ipv6 nd suppress-ra
```

```
interface lo
ip forwarding
ipv6 forwarding
```

This is a template for the `bgpd.conf` file. Values to be changed are in bold.

```
hostname bgpd
password yourpassword
enable password yourpassword
log stdout
log syslog

router bgp 197835
 bgp router-id youripaddress
 network 176.62.53.0/24
 network 176.62.53.0/25
 network 176.62.53.0/27
 network 176.62.53.128/25
 aggregate-address 176.62.53.0/24
 aggregate-address 176.62.53.0/25
 aggregate-address 176.62.53.0/27
 aggregate-address 176.62.53.128/25
 neighbor ibgp_neighbor_ipv4_address remote-as 197835
```

```
 neighbor ibgp_neighbor_ipv4_address description bgpmara
 neighbor ibgp_neighbor_ipv4_address next-hop-self
 neighbor ibgp_neighbor_ipv4_address route-map IBGP in
 neighbor ebgp_neighbor_ipv4_address remote-as 35131
 neighbor ebgp_neighbor_ipv4_address description ydea
 neighbor ebgp_neighbor_ipv4_address shutdown
 neighbor ebgp_neighbor_ipv4_address ebgp-multihop 10
 neighbor ebgp_neighbor_ipv4_address soft-reconfiguration inbound
 neighbor ebgp_neighbor_ipv4_address route-map LP150 in
 neighbor ebgp_neighbor_ipv4_address IPv4-53-128-PREPEND-OUT out
 neighbor 2001:4c00:893b:ffff::2 remote-as 197835
 neighbor 2001:4c00:893b:ffff::2 route-map LP50 in
 neighbor 2a02:688:1::5 remote-as 5394
 neighbor 2a02:688:1::5 description UNIDATA TECNOPOLO
address-family ipv6
 aggregate-address 2001:4c00:893b::/48
 redistribute static
 redistribute olsr
neighbor 2001:4c00:893b:ffff::2 activate
 neighbor 2001:4c00:893b:ffff::2 route-map LP50 in
neighbor 2a02:688:1::5 activate
 neighbor 2a02:688:1::5 route-map LP150 in
 neighbor 2a02:688:1::5 route-map IPv6-UNI-OUT out
 exit-address-family

ip prefix-list ALL-IPv4 seq 5 permit 176.62.53.0/24
ip prefix-list ALL-IPv4 seq 10 permit 176.62.53.0/25
ip prefix-list ALL-IPv4 seq 20 permit 176.62.53.128/25
ip prefix-list ALL-IPv4 seq 30 permit 80.79.62.80/29
ip prefix-list ALL-IPv4 seq 40 permit 80.79.62.186/32
ip prefix-list IPv4-53-0 seq 10 permit 176.62.53.0/25
ip prefix-list IPv4-53-128 seq 10 permit 176.62.53.128/25
ip prefix-list IPv4-YDEA-OUT seq 10 permit 80.79.62.80/29
ip prefix-list IPv4-YDEA-OUT seq 20 permit 176.62.53.0/24
ip prefix-list IPv4-YDEA-OUT seq 30 permit 80.79.62.185/32
!
ip as-path access-list SOLONINUX permit ^$
ip as-path access-list SOLONINUX deny any
!
ip community-list 100 permit 1:100
!
route-map LP50 permit 10
 set local-preference 50
!
```

```
route-map SOLONINUX permit 10
 match as-path SOLONINUX
!
route-map IPv4-53-128-PREPEND-OUT permit 10
 match ip address prefix-list IPv4-53-128
 match as-path SOLONINUX
 set as-path prepend 197835
!
route-map IPv4-53-128-PREPEND-OUT permit 20
 match ip address prefix-list ALL-IPv4
 match as-path SOLONINUX
!
route-map LP150 permit 10
 set local-preference 150
!
route-map IBGP permit 10
 match community 100
 set local-preference 200
!
route-map IBGP permit 20
 set local-preference 50
!
route-map LP30 permit 10
 set local-preference 30

ipv6 prefix-list IPv6-UNIDATA-OUT seq 10 permit 2001:4c00:893b::/48

ip as-path access-list IBGP-NINUX deny 197835
ip as-path access-list NINUX-UNIDATA-TECNOPOLO permit ^$
!
route-map NINUX-TECNOPOLO permit 10
 match as-path NINUX-UNIDATA-TECNOPOLO
!
route-map IPv6-UNI-OUT permit 10
 match ipv6 address prefix-list IPv6-UNIDATA-OUT
!
route-map LP50 permit 10
 match as-path IBGP-NINUX
 set local-preference 50
!
route-map LP150 permit 10
 set local-preference 150
```