Ninux Roma

The Routing Architecture

May, 2012 - Version 0

Authors

- ZioPRoTo - Saverio Proto
- Nino - Antonino Ciurleo
- maruscia - Mara Sorella
- Clauz - Claudio Pisa
- claudyus - Claudio Mignanti
- Hispanico - Marco Giuntini
- ordex - Antonio Quartulli
- Gubi - Alessandro Gubitosi
- G10h4ck - Gioacchino Mazzurco

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

Authors 1

Scope of this document 1

Table of Contents 2

Requirements 3

Glossary 4

Address Space 5
- IPv4 5
- IPv6 5
- Conventions in Addressing 6

Network Topology 7
- Mesh stub areas 8
- Mesh transit areas 9

Routers specification 11
- BGP routers 11
  - Export Policy 12
  - Import Policy 13
- OLSR routers 13
  - Routing operation inside the OLSR router 14
  - Avoid ghost traffic to gateways 14
  - Avoid ADSL Blackholes 14
- ADSL routers 15

NAT - Network Address Translation 16

VPN: OOB, Monitoring and Connection to other cities 16

Migration from legacy Ninux to this routing architecture 17

Conclusion 18

Appendix: OLSR router configuration template 18

Appendix: OLSR Troubleshooting 18

Appendix: BGP router configuration template 18

Requirements

The Ninux network of Rome is designed to be dual stack and to fully support native IPv6 and IPv4. Moreover we 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[1] 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.

[1] in this specific statement with "public" we mean a public IPv4 address belonging to the Ninux Autonomous System, because it is obvious that our private IPv4 addresses cannot be routable from outside our AS.

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[2] 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 following 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
```

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

[2] The RFC 1918 actually defines a wider address space, we use just a /16 to reserve IP addresses to other Ninux cities.

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 a sequential number picked up from our database. However because the IPv6 address space is bigger than IPv4, 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[3], 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. Due to ZIPCODE number 0 is not used in Italy, we chose 10.0.1.X for VPN connection addressing.

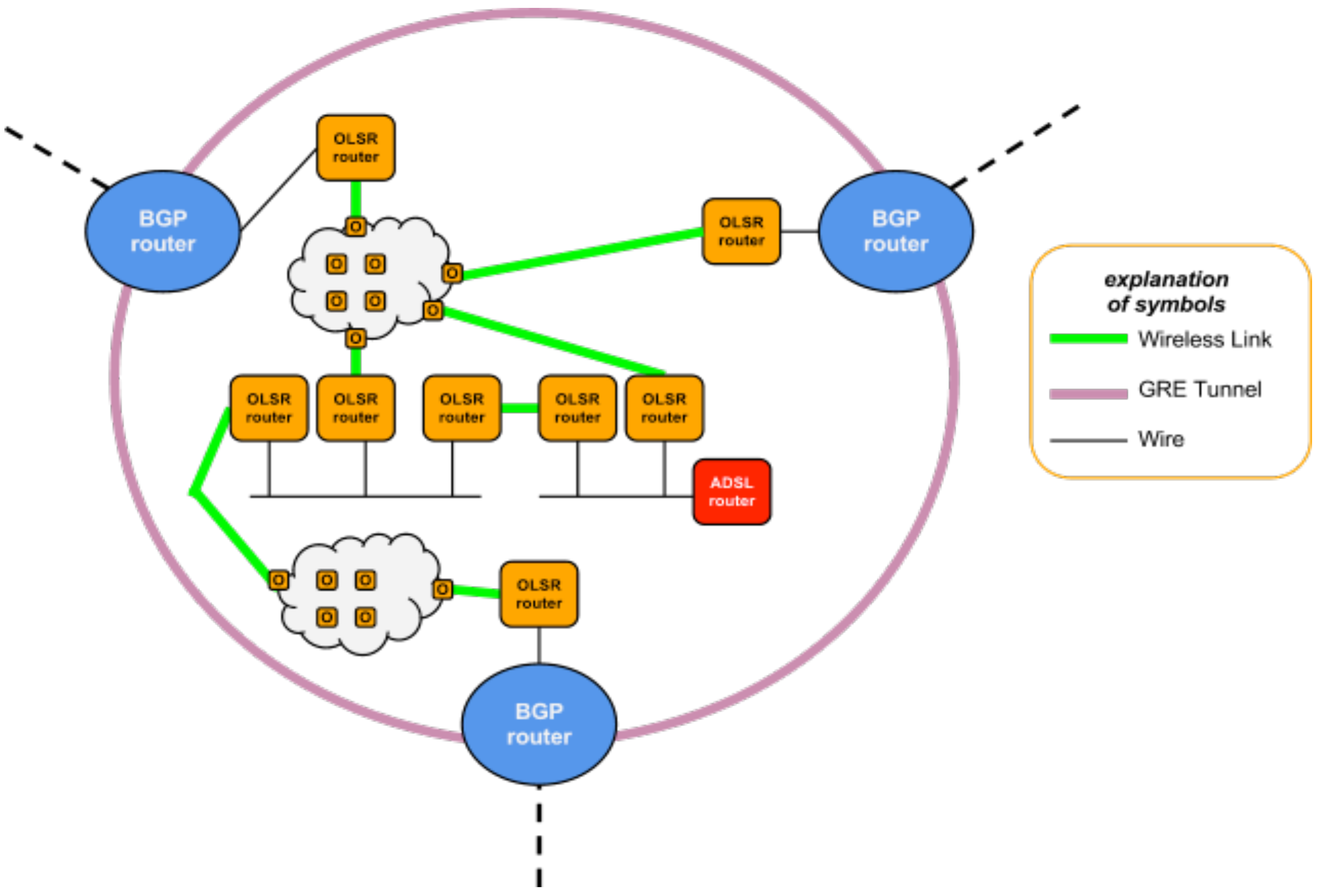
GRE Tunnels Point to Point Links:

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

[3] Explained in next section

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.

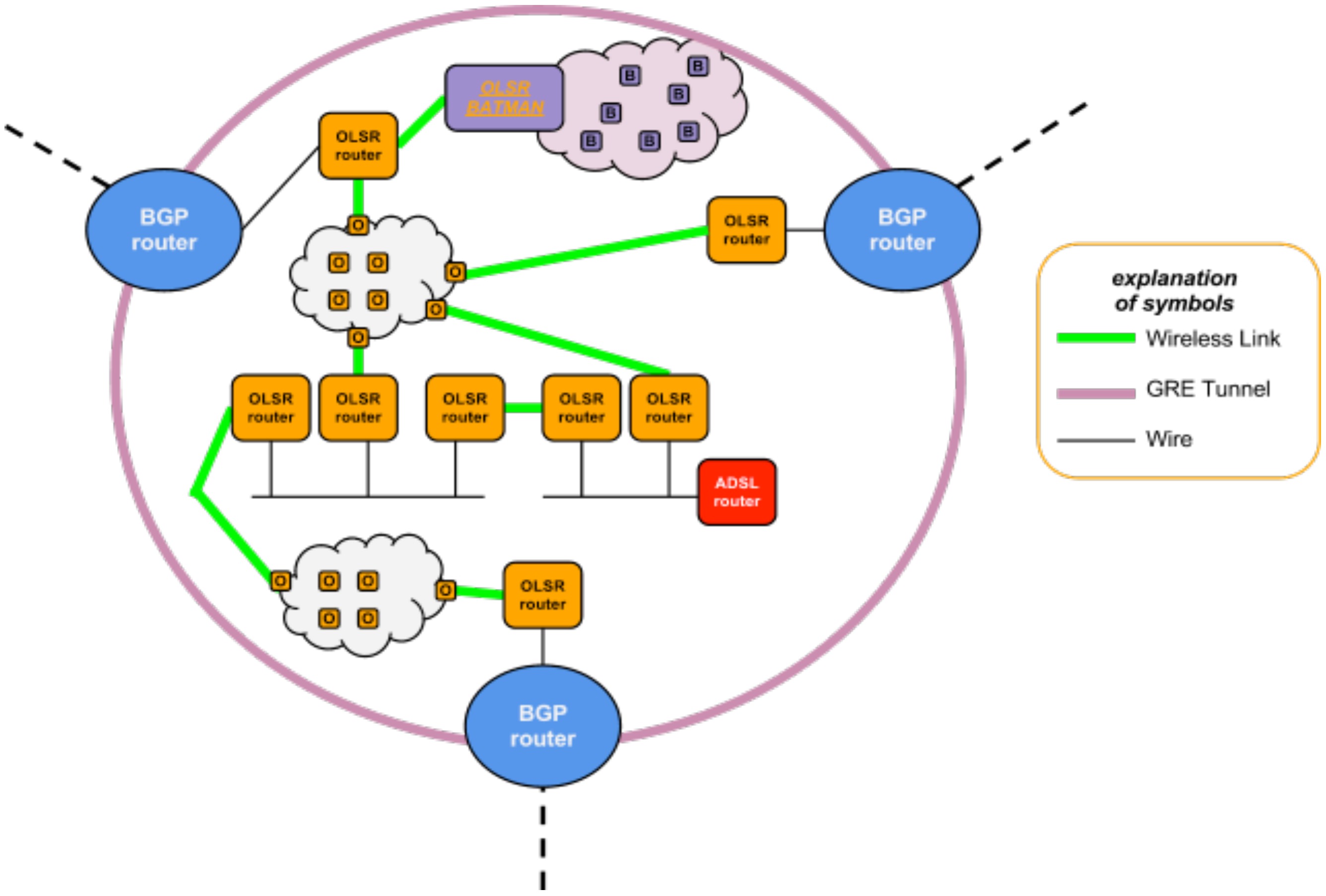


OLSR routers, as we will explain more in deep later, are the devices that the community members have on the roof of their houses. On the same roof there might be more than one OLSR router, but at least one must be present to run a Ninux Node. OLSR routers can be connected to other routers via both wireless 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.

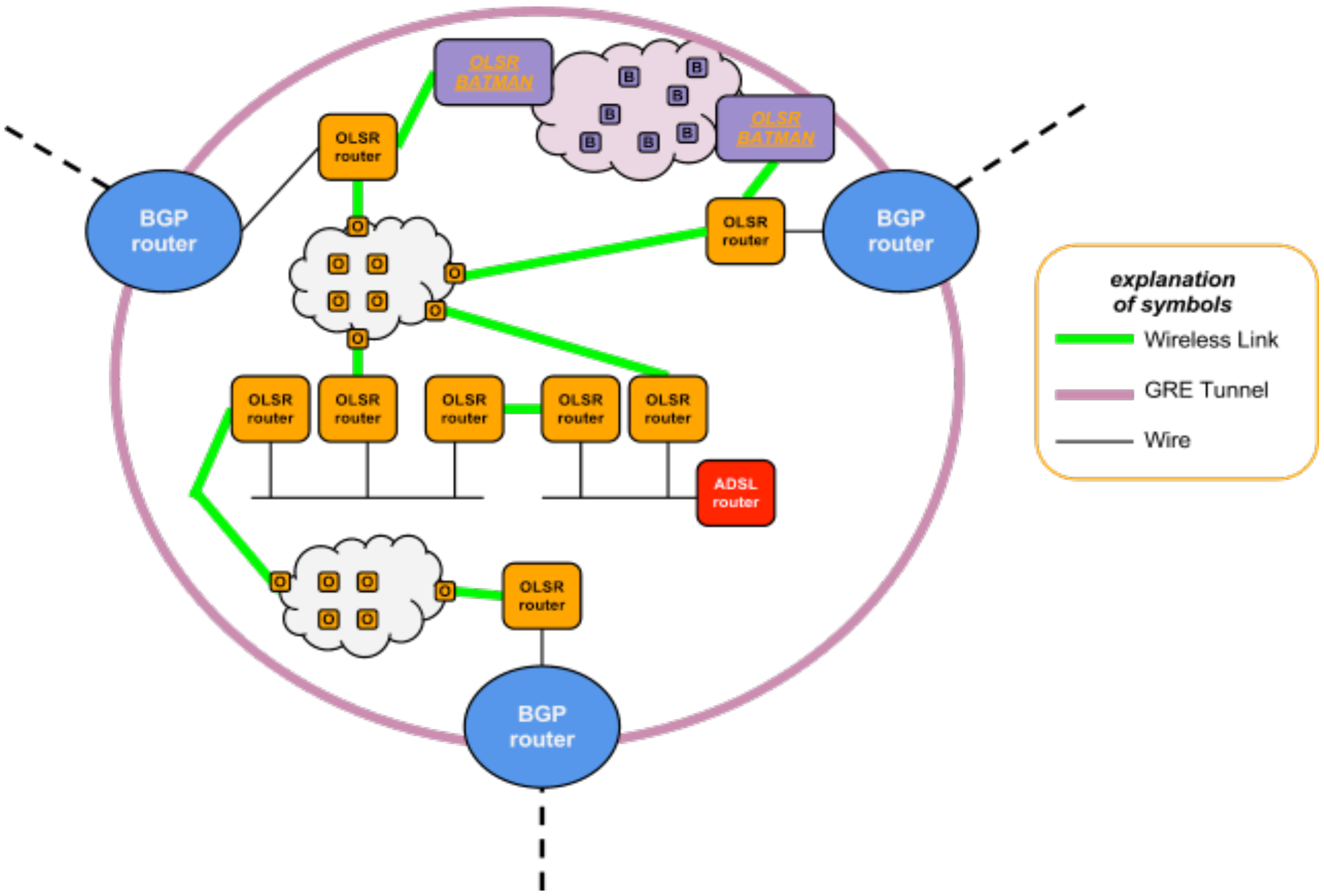


As we see in the figure there will be an edge router between the two domains that speaks both the OLSR and the B.A.T.M.A.N.-Advanced routing protocol. This edge router will announce all the IP addresses used in the batman-adv network with an HNA setting.
Note that is not mandatory to have a single router 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 an 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”[4]: 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.

[4] http://www.open-mesh.org/wiki/batman-adv/Gateways

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.

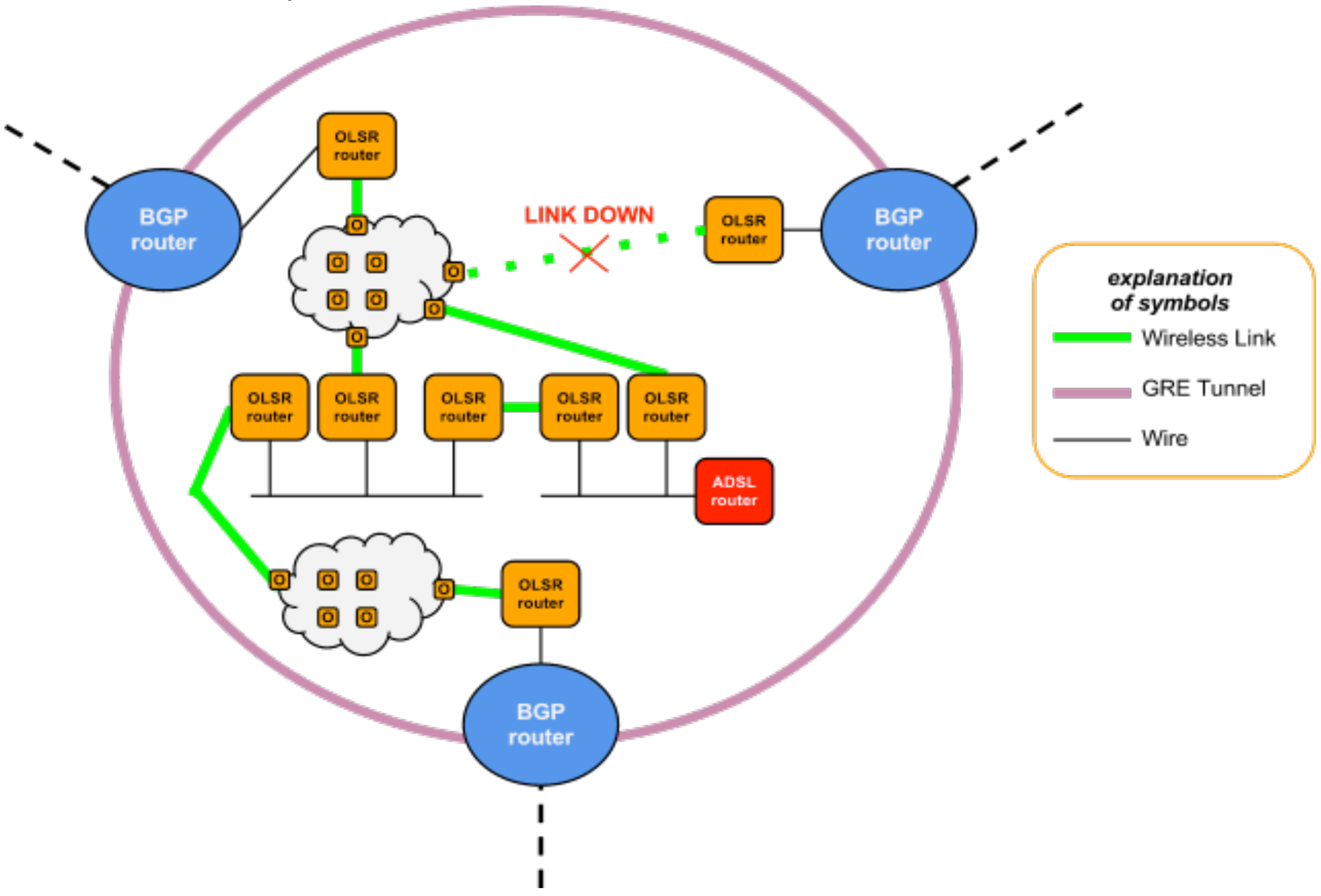
BGP routers

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 tunnels and iBGP is handled with a full mesh of peering.This kind of link is implemented by GRE tunneling over 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.
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[5] and the olsrd daemon version 0.6.3 (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.

[5] https://dev.openwrt.org/browser/packages/net/quagga/patches/120-quagga_manet.patch or here https://github.com/zioproto/quagga-manet

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 others OLSR routers (more later).

It is important also to distribute in OLSR the classical default route, 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. In 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 has the following routing tables.

IPv6Table:	IPv6 routing table, learnt via OLSR
RtTable:	olsrd IPv4 routing table
RtTableDefault:	IPv4 default route learnt via OLSR
RtTableManual:	Routes via web interface applied by user

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*
IPv4:	
If destination is private:	*use RtTable*
If destination is Ninux Public IPv4:	*use RtTable*
If source is Ninux Public IPv4:	*use RtTable*
If source is Transit[6] Public IPv4:	*use RtTable*
Anything else	*use RtTableManual and then*
RtTableDefault	

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 at the end of the RtTable. 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 at the end of the 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 RtTable
ip route add blackhole 172.16.0.0/12 table RtTable
ip route add blackhole 192.168.0.0/16 table RtTable
ip route add blackhole 176.62.53.0/24[7] table RtTable
```

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

[6] This rule is optional. It is needed just in case the Ninux AS gives transit to another AS towards an upstream provider. It stays as a placeholder for the completeness of the architecture, but we will probably never use it.
[7] In this example 176.62.53.0/24 is the public IPv4 space of the Ninux AS

- *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 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.

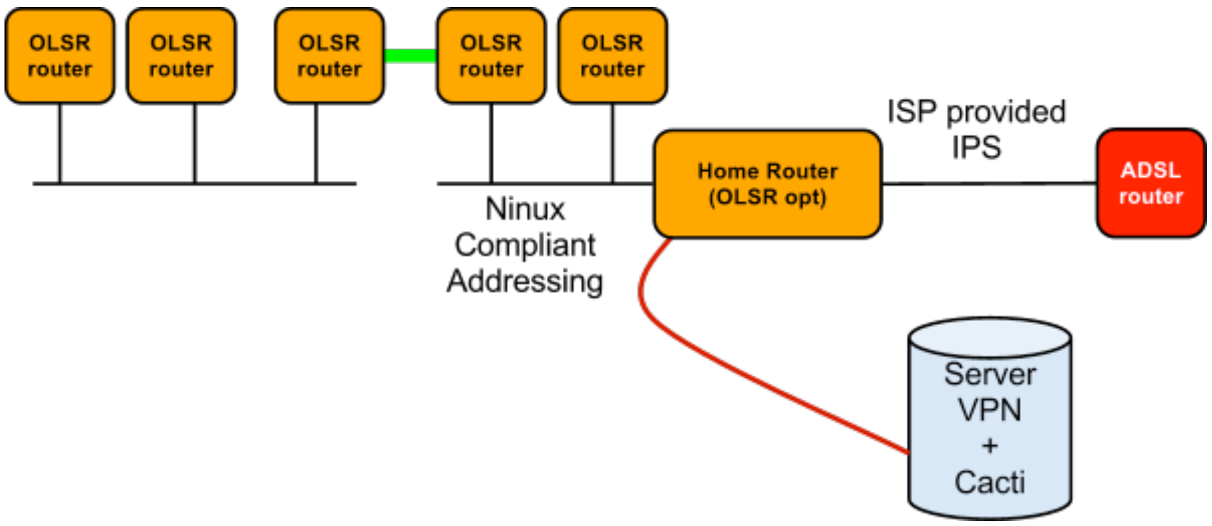
ADSL routers

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.
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).



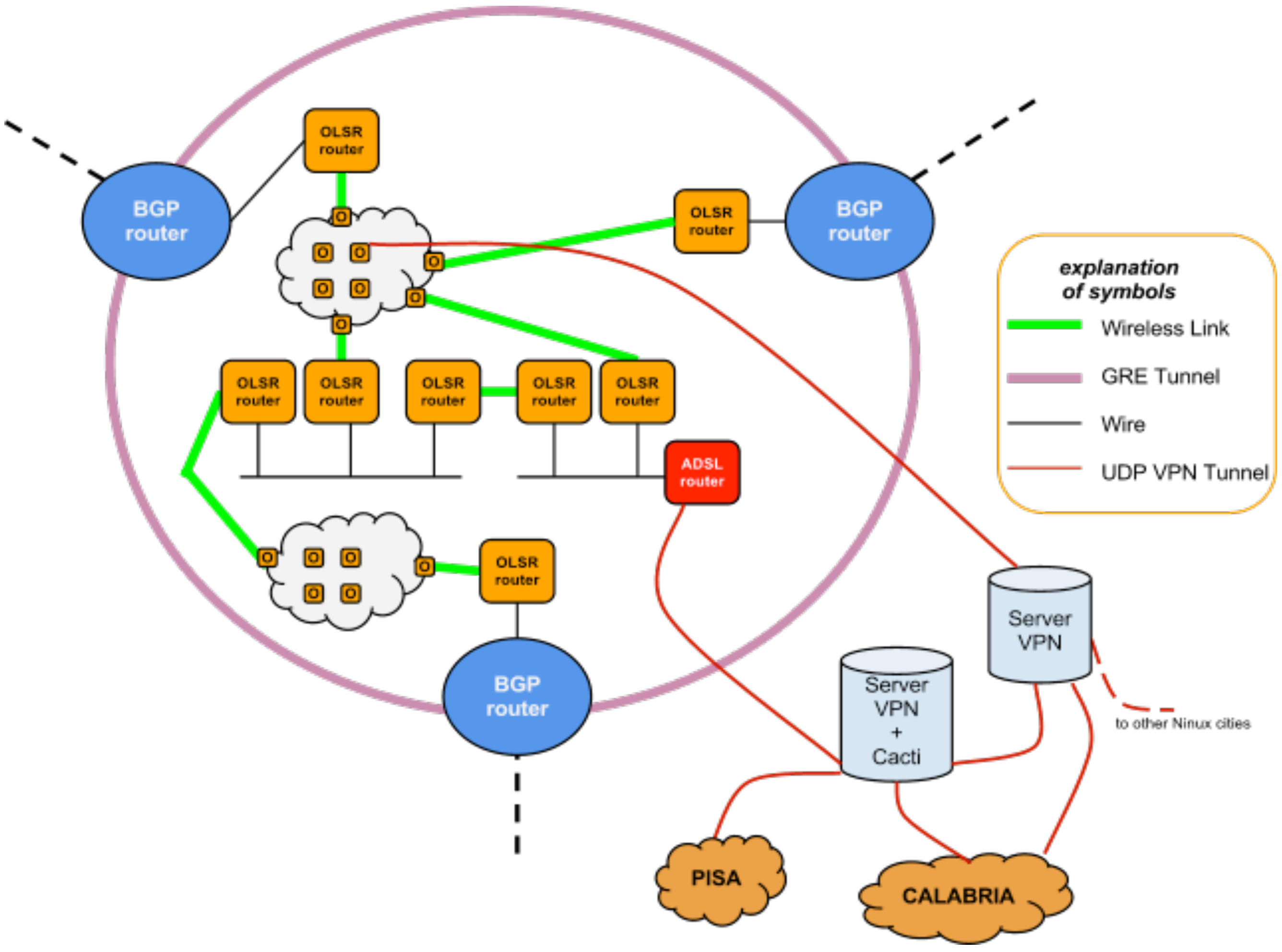
NAT - Network Address Translation

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

The VPN setup regards only the IPv4 network. At the moment we make use of tinc-vpn[8].
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.
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 VPN is meant for signalling (SNMP, Nagios checks) and troubleshooting and never for carrying actual data traffic.

[8] http://www.tinc-vpn.org



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 2012. 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

A template of olsrd.conf will be provided soon with the next release of this document.

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/zioproto/quagga-manet.git

Templates files of zebra.conf and bgpd.conf will also be provided with the next release of this document.