Peer-To-Peer Virtual Private Networks: Multiple Supernodes Feature for N2N

Costin Lupu
Automatic Control and Computers Faculty
University POLITEHNICA of Bucharest
Splaiul Independenței nr. 313, Bucharest, Romania
costin.lup@gmail.com

June 24, 2012

Abstract

The current research report describes the design and implementation of a decentralizing enhancement technique applied for N2N, a Peer-to-Peer Virtual Private Network solution. The feature addresses two important points for distributed systems: reliability and availability.

1 Introduction

N2N (network to network)[1] is a layer-two over layer-three Peer-to-Peer (P2P) virtual private network (VPN) application that allows users to exploit properties typical of P2P applications at the network level instead of application level. This means that users can gain unrestricted IP visibility and be reachable with the same address regardless of their current network environment. In a nutshell, as OpenVPN moved Secure Sockets Layer (SSL) [2] from application (e.g. used to implement the Hypertext Transfer Protocol Secure protocol) to network protocol, N2N moves P2P from application to network level.

Like most P2P protocols, N2N has one or more supernodes and several edge nodes. Supernodes are used to introduce edge nodes and to cross symmetric Network Address Translation (NAT). Edge-nodes are grouped in communities coordinated by the supernode. Periodically an edge-node sends registration messages to supernode for keeping its record in the supernode peers list. A backup supernode can be assigned to each edge as a command line parameter that will be used if the connection with main supernode fails.

2 Multiple Supernodes - Overview

The current N2N versions uses a very limited set of supernodes that are provided on edges startups as command line parameters. Furthermore, there is no communication between supernodes for distributing the lists of communities and/or edges. As a result, in order to communicate with each other, 2 edges in the same community must register to the same supernode. This also implies that for the current version of N2N the supernode becomes a single point of failure, making the whole virtualized network vulnerable. If the supernodes that were used on edge registration stop running, the whole network setup is lost.

Due to its performances and size, N2N can be also used on mesh mobile or wireless sensor networks. However, different mesh networks cannot be interconnected as long as their coordinating nodes, the supernodes in the N2N case, don't have a communication protocol. For a set of low power mobile nodes it is essential that the communication with the coordinating node to be established based on connectivity performance.

Another key factor for improving communication, especially when relaying is necessary, is load-balancing: the communities should be equally distributed among the set of supernodes based on their edges number and response times.

The problems outlined above raised the demand on the N2N discussion list [3] for implementing multiple supernodes functionality for N2N. The N2N creators suggested the following basic functions would be needed in edge [4]:

- Detecting that a supernode has failed
- Avoiding use of failed supernodes
- Detecting supernode recovery
- Reinstating use of a recovered supernode

Also, regarding feature design, the authors outlined two major concerns: supernode discovery and packet forwarding behaviour. The options suggested for the former were either all edges in a community are statically configured to use the same set of supernodes, or supernodes can be discovered from any one supernode in the group. The options for packet forwarding behaviour were:

- All edges in a community must stay connected to the complete set of supernodes so that any supernode is able to forward packets between any two edges.
- Edges can have different supernode connectivity and the supernodes manage forwarding packets to a supernode that has an open connection to the destination edge.

The current feature implementation includes dynamic supernode discovery. However, for packet forwarding, the chosen solution was keeping all edges in community connected to the subset of coordinating supernodes. The feature design had to be determined from two perspectives: inter-community and intra-community.

Inter-community perspective involved designing the communication between supernodes under a new protocol, on a different port than the one used for the N2N communication protocol. This approach was chosen because the messages of the N2N protocol are edge oriented, each message header containing the community name of the initiating node. The main goal of supernode communication is distributing the peers communities. A starting supernode would have to choose some communities it would voluntarily coordinate, a decision being made depending on the needed level of redundancy for any given community. This stage will be further referred as communities discovery. Every supernode is represented by 2 pairs of addresses: *IP address, UDP Port*. First one, used in the N2N protocol will be further referred as N2N address or advertise address, is the address on which a supernode is receiving edge registrations or performing relayed transfers for peers between symmetric NAT. Second one, used in the Multiple Supernodes protocol will be referred as SNM address.

Intra-community perspective implied choosing the messages to be exchanged between an edge-node and a supernode, before the former is ready to communicate with its peers. The main goal is establishing the set of supernodes that will be coordinating the community based on connection quality and supernodes load level. This stage will be further referred as supernodes discovery.

The feature was implemented [5] in order to alter as less as possible the current version of N2N and to allow further modifications to be made easily. Decisions regarding the design of the two types of communication will be argued next. In following sections of the report, the implemented protocol will be referred as Multiple Supernodes (SNM) protocol.

3 Message Formats

3.1 SNM Message Header

Fig. 1 shows the header used in all SNM messages. *Type* field holds an 1-byte value representing the message type. Current message types values are:

- *0x01* – for SNM requests
- *0x02* – for SNM responses

- *0x03* – for SNM advertise messages

A message holding any other message type will be ignored. *Flags* is a 1-byte value representing the flags associated with each message type. On every request, a 2-bytes *Sequence Number* is generated by the issuer for keeping track of all sent messages.

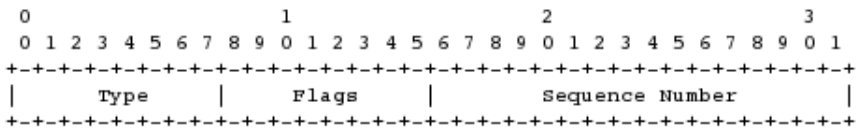

Figure 1: SNM Message Header

3.2 SNM Request Message

The request message flags have the following meanings:

- *S* – all known SNM addresses of running supernodes are requested.
- *C* – all communities coordinated by the receiving supernode are requested.
- *N* – only SNM addresses of supernodes coordinating the provided communities are requested. The request payload will contain a list of community names. Every community name is filled as a variable-length name (name length followed by the string itself). *N* and *C* flags are mutually exclusive.
- *A* – the advertise address (N2N supernode address) is requested.
- *E* – message is initiated by an edge-node.

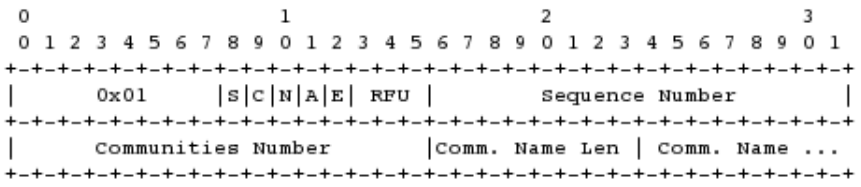

Figure 2: SNM Request Message

3.3 SNM Response Message

Fig. 3 represents the message format of the SNM response message. A SNM response is always sent by a supernode in reply to a SNM request. *Supernodes Number* and *Communities Number* fields will be always set, regardless the type of request that was previously sent. The supernodes addresses are encoded with the following fields:

- *Family* – 1-byte value representing the IP protocol family (IPv4 or IPv6)

- *UDP Port* – 2-bytes value
- *IP Address* – 4-bytes IPv4 address or 16-bytes IPv6 address

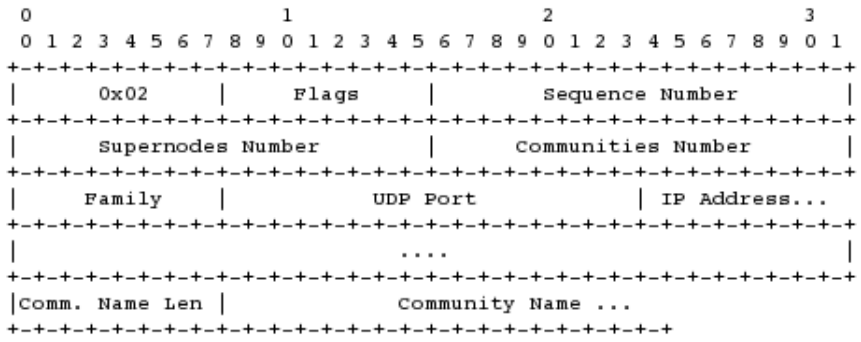

Figure 3: SNM Response Message

If *S* flag is set the response will contain SNM addresses of all known supernodes. If *A* flag is set then the response will contain advertise addresses (N2N addresses) of supernodes coordinating a provided community. If *N* flag is set, the supernodes addresses in the response are associated with an included community name. If *C* flag is set, then the community names included in response are all those coordinated by responding supernode. If the response is sent to an edge-node, then when *C* flag is set, only the community number is provided, considering that an edge will have no interest in other coordinated communities.

3.4 SNM Advertise Message

The advertise message contains the advertise address of sending supernode and the names of communities coordinated by it. If the message is sent to another supernode, it may have the *A* flag set, stating that sending supernode is also requesting for its counter-party advertise address. This may only happen during the community discovery stage of a starting supernode. A receiving supernode will advertise the sender in the registration acknowledgments sent to edges.

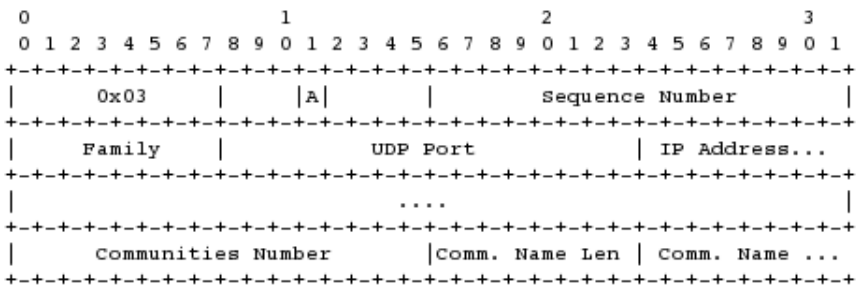

Figure 4: SNM Advertise Message

4 Supernodes Communication

Each supernode keeps a persistent list of all registered supernodes in a file, each entry having the form *IP Address:Port*. On startup, a supernode will load the SNM addresses from file and will sent SNM requests to all of them. On first supernode startup it is mandatory to set the -i command line parameter value to a running supernode address (e.g. *./supernode -i a.b.c.d:12000*). After all information about the other running supernodes is collected, the starting supernode will decide to send advertise messages to all those supernodes that are coordinating the communities it is interested in.

Coordinated communities are also kept in a file, along with the advertise addresses of the coordinating counter-parties. This information is also loaded on startup. The supernode will run through the community discovery stage for finding those communities that have not acquired the needed level of redundancy. The requests will have the C flag set, demanding for communities coordinated by the queried supernode. After all the information is gathered, the new supernode will pick the communities that it will be coordinating, depending on the following parameters:

- *MIN_SN_per_COMM* – the minimum required number of supernodes coordinating a given community, needed for redundancy purpose
- *MAX_SN_per_COMM* – the maximum number of supernodes serving a given community, needed for avoiding a given community being coordinated by all running supernodes
- *MAX_COMM_per_SN* – the soft limit of communities being coordinated by a given supernode

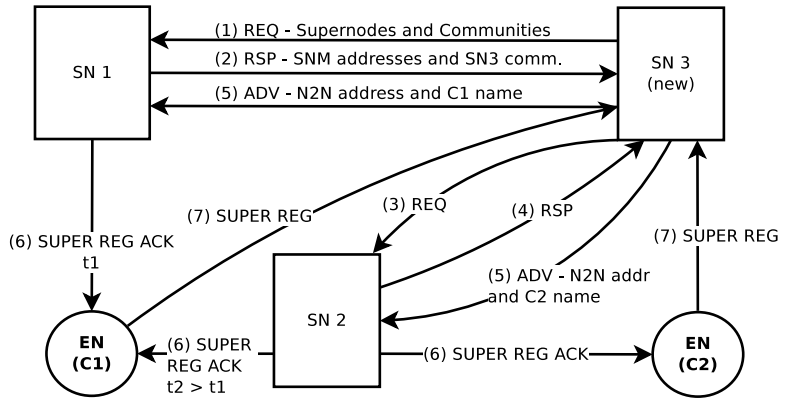

Figure 5: Communities discovery

After deciding what communities it will voluntarily coordinate as well, advertise messages will be send to the coordinating counter-parties. The receivers will send back advertise messages with their N2N addresses. When receiving register messages from an edge-

node, a supernode will fill in the acknowledgement message the advertising addresses of its counter-parties, to which the edge-node will send register messages also. This way, the lists of peers in the same community is also distributed to every coordinating supernode.

An example of communities discovery stage is illustrated in Fig. 5. Supernode SN1 is coordinating communities C1 and C2, while SN2 is coordinating C1. A new supernode, SN3, is started with the SNM address of SN1 set as a command line parameter. In step 1, SN3 is sending a SNM request to SN1, requiring its coordinated communities names and the SNM addresses of all known supernodes. The response contains the SNM address of SN2 and C1 community name. The new supernode is sending a new request to SN2, this time receiving the C2 community name. Next, SN3 sends SNM advertise messages containing its N2N address and the list of chosen communities. After receiving register messages from edge-nodes, SN1 and SN2 will fill the new address in acknowledgments. Note that the edge-node in community C1 is receiving the back-up address from SN1 before receiving it from SN2. This happens because register messages are sent using a round-robin approach. More details are explained in section 5. Finally, the two edge-nodes will send register messages to SN3.

5 Edge - Supernode Communication

The communication between edges and supernodes on the discovery stage was designed considering two options. First option was to integrate the SNM messages semantics in the existing N2N protocol. This solution would have involved that a supernode would have to handle SNM requests and responses under the N2N protocol as well. An edge-node would only connect to advertise addresses of supernodes, meaning that a coordinator assisting the edge bootstrap must keep track of all other supernodes advertise addresses as well.

The second option, that was chosen for the current implementation, implied using the SNM protocol for edge-supernode communication as well. The difference from the first option is that an edge must find the advertise addresses of supernodes coordinating its community. It has the advantage that it doesn't alter the existing N2N protocol and it keeps simplicity in the implementation. Sending back-up supernodes addresses to an edge was implemented by filling the supernode registration acknowledgments.

An edge-node starting for the first time, will begin in a discovery state, meaning that its main target will be to find the supernodes that will coordinate its community. There are 2 possible situations:

- the edge-node is initiating a new community and it will have to choose the supernodes that will coordinate it

- the community already exists and the edge-node will discover the supernodes that are coordinating it

If the edge-node is initiating a new community, it will request the SNM address and communities number of every running supernode. After all information is collected, it will choose the supernodes that will be coordinating the new community. A supernode will be ranked based on its communities number and its response time. Next, edge will start an advertise requesting stage receiving the advertise addresses of supernodes it has chosen. After all addresses are received, it will become ready from communication with community peers.

If the community already exists and the edge-node is starting for the first time, it will start discovering its community supernodes. The supernode discovery process will begin with sending requests to supernodes for that the SNM addresses were provided as command line parameters (e.g. *./edge -l a.b.c.d:5645*). If a receiving supernode is coordinating the community, it will advertise its N2N address along with those of its counter-parties supernodes and the edge will become ready for processing messages from its peers. Otherwise, the supernode will send its communities number and the SNM addresses of all known supernodes.

After getting the complete list of coordinators' advertise addresses, the edge-node will be ready to process any incoming messages from its peers. Registration messages will be send to each supernode of the community in a round-robin fashion, that will help supernodes maintain the complete lists of community peers. A supernode registration acknowledgement will contain the advertising addresses of other coordinating supernodes as well. If a supernode will voluntarily decide to coordinate the community, its address will be advertised in the register acknowledgments. The list of community supernodes is saved in a file and loaded every subsequent next start-up.

The rank of a supernode is computed based on the response time and the load level that is represented by number of coordinated communities. The formula used in ranking is:

$$Rank = \frac{ResponseTime \cdot 100}{n},$$

where $n = MAX\_COMM\_per\_SN - CommNum$ if $CommNum < MAX\_COMM\_per\_SN$ or $n = 1$ otherwise.

The formula was chosen in order to offer a greater chance to supernodes having a lower number of communities. First *MIN_SN_per_COMM* supernodes are chosen in from ascending ordered set of rankings. All supernodes advertise addresses will be kept in a file, each entry having the form *IP Address:Port*.

All SNM messages added apply for edge-supernode communication. A SNM request

initiated by an edge will have the E (edge) flag set and will contain its community name. During supernode discovery stage, the edge-node will start sending requests for SNM addresses until the community supernodes set will be established. If the community is new, the edge will send requests for advertise to all the supernodes it decided to use.

A SNM response payload received by an edge-node may contain either SNM addresses or advertise addresses. The difference is given by the A (advertise) flag, in which case the edge will use the addresses to register to supernodes. Upon receiving such a response, the edge-node will stop quering for other supernodes SNM addresses and it will become ready for communicating with peers.

A SNM advertise message can only be received by an edge-node during discovery stage and only if it had previously requested it from a certain supernode.

6 Experimental Setup

Edge nodes use TAP [6] devices that act as virtual ethernet devices implemented in the operating system kernel. In TAP devices, the ethernet driver is implemented in a user-space application. In order to communicate with another peer, an edge node must open the TAP device as regular file and read/write messages from/to it.

The Multiple Supernodes schema was tested on machines running 64-bit Debian Wheezy distribution. 5 supernodes were launched on different machines coordinating a total of 40 communities. For a given community, each edge was started on a different machine. The values chosen for SNM parameters where:

- MIN_SN_per_COMM = 3
- MAX_SN_per_COMM = 4
- MAX_COMM_per_SN = 3

An edge node will be saving the N2N addresses of its coordinating supernodes in a file named *EDGE_SN_<Community_name>*. A supernode will save the SNM addresses of other supernode in a file named *SN_SNM_<SNM_port>*, while the coordinated communities will be saved in *SN_COMM_<SNM_port>*. For testing, the chosen UDP ports where: *11XYZ* for N2N communication and *12XYZ* for SNM communication.

```
comm_num=33
sn_num=3 name=comm02
         192.168.1.203:11203
         192.168.1.208:11208
         192.168.1.210:11210
sn_num=3 name=comm04
         192.168.1.203:11203
         192.168.1.208:11208
         192.168.1.210:11210
sn_num=2 name=comm101
         192.168.1.210:11210
         192.168.1.208:11208
sn_num=2 name=comm410
         192.168.1.203:11203
         192.168.1.208:11208
...
```

Listing 1: Communities file *SN_COMM_12202* for supernode listening on *192.168.0.202:11202*

The communities file example above illustrates how the supernodes were chosen for communities. The number of supernodes coordinating a community is *<sn_num> + 1*, taking into account the host (e.g *192.168.1.202*). Communities *comm02* and *comm04* were first created and supernodes *192.168.1.[202,203,208]* were chosen to coordinate them. Next, supernode *192.168.1.203* was stopped, so another supernode, *192.168.1.210*, was requested in order to assure the redundancy. Community *com101* was created while *192.168.1.203* was still down and supernodes *192.168.1.[202,208,210]* were chosen. After supernode running on *192.168.1.203* was started again, community *comm410* was created and supernodes *192.168.1.[202,203,208]* were chosen.

7 Conclusion and Further Work

The flexibility of multiple supernodes feature can be enhanced by adding other parameters to be included in the load balancing evaluation. When initiating a community, an edge may expand its set of potential supernodes if response times are under a configurable limit and choose them in a random way. Improving edge-nodes discovery and communication by using distributed hash tables (*DHT*s) may cancel out the need for supernodes and would enable full decentralization.

References

[1] L. Deri and R. Andrews, "N2N: A Layer Two Peer-to-Peer VPN." http://www.ntop.org/products/n2n, 2009.

[2] T. Dierks and C. Allen, "The TLS Protocol." RFC 2246, 1999.

[3] http://listgateway.unipi.it/pipermail/n2n/2008-November/000063.html.

[4] http://listgateway.unipi.it/pipermail/n2n/2009-January/000080.html.

[5] "Multiple Supernodes repository." https://github.com/lukablurr/n2n_v2_fork.

[6] M. Krasnyansky, "Universal TUN/TAP Driver." http://vtun.sourceforge.net, 2001.