Technische Universität Berlin
Fakultät IV - Elektrotechnik und Informatik
Fachgebiet AOT

Bachelor Thesis

Agent-based Approach for Cross-subnet Communication

by Rubén Hervás Fernández
Matrikelnummer: 0334513

Prof. Dr. Sahin Albayrak
Betreuung: Marcel Patzlaff, Claus Schenk

August 2, 2011

Eidesstattliche Erklärung

Die selbstständige und eigenhändige Anfertigung versichere ich an Eides Statt.

August 2, 2011

Abstract

The establishment of point-to-point connections between hosts behind NAT boxes have been always a problem due to the problem with the private IP addresses and NAT firewalls. To solve this problem, there are different techniques to make possible that hosts behind NAT boxes can establish a point-to-point connection rounding NAT firewalls and using solutions to exchange their IPs.

The goal of this thesis is to present an implementation of a communication protocol which establishes a communication between agents that are behind NAT boxes avoiding all the problems that could occur during the communication, such as duplicated IP addresses or host unreachable errors and introduce to the reader some related work about NAT traversal. There is also in this thesis a little introduction to the existing techniques used to round a NAT firewall and the explanation of why we use NAT boxes even though they sometimes are a problem.

Zusammenfassung

Die Einrichtung einer Point to Point Verbindungen zwischen den NAT-Boxes mit den dazugehörigen Hosts war immer ein Problem, ebenso wie das Problem mit den privaten IP Adressen und den NAT- Firewalls.

Um dieses Problem zu lösen gibt es verschiedene Techniken und Lösungen, damit Hosts mit dazugehrigen NAT-Boxes eine Point to Point Verbindung herstellt können, die die Firewall umgeht, um deren IP-Adressen auszutauschen.

Das Ziel dieser Thesis ist die Implementierung eines Kommunikationsprotokolls, welches die Kommunikation zwischen dem Hosts mit den dazugehörigen NAT Boxes herstellt und auftretende Probleme wie beispielsweise doppelte IP-Adressen oder nicht erreichbare Hosts dadurch vermeidet.

Es erfolgt aurdem eine Einführung für den Leser in die Arbeitsweise von NAT-Traversal.

Es handelt sich also in dieser Thesis um eine kleine Einführung zu bestehenden Techniken die verwendet werden, um eine NAT Firewall zu umgehen, und die Erklärung warum wir NAT-Boxes benutzen, obwohl es manchmal Probleme verursacht.

Abstract

L'establiment de connexions punt-a-punt entre hosts sempre ha sigut un problema quan eren darrera d'un dispositiu NAT a causa del desconeixement de les adreces IP. Per solucionar aquest

problema hi ha diferents tècniques i solucions que aconsegueixen establir una connexió punt-a-punt evitant els firewalls dels dispositius NATs i així fer possible la comunicació com si els hosts tinguessin IP pública.

En aquesta tesis s'explica una breu introducció a les tècniques més usades i que he trobat més interessants. El propòsit d'aquesta tesis es presentar una solució implementada per mi que aconsegueix establir una connexió punt-a-punt entre tres host que són darrere dispositius NAT sense que hi hagi cap problema amb els firewalls de ningun dels dispositius NAT i introduir breument que és el NAT traversal i algunes aplicacions que utilitzen aquests tipus de tècniques. També s'expliquen breument els mètodes UDP i TCP hole punching.

Contents

1 Introduction **1**

2 Description of the problem **6**

3 Overview of the solution **8**

4 Description of the communication protocol **11**

4.1 Application agent 11

4.2 Mobile agent 14

4.3 Webserver agent 16

5 Implementation **17**

5.1 Mobile agent 17

5.2 Application agent 20

5.3 Webserver agent 22

6 Related work **25**

6.1 NUTSS 25

6.1.1 STUN 25

7 Conclusion **28**

7.1 Security 28

7.2 Standards 28

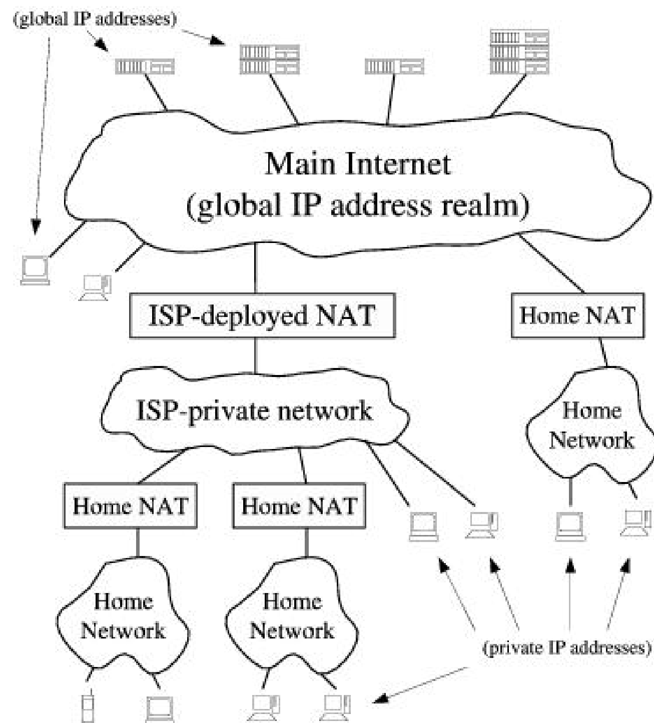
7.3 Future developments in technology 28

1 Introduction

Due to the growth of the number of devices connected to the Internet, connecting devices with each other has become a problem. It is no longer possible that every device can be directly addressable. This is because the IP version that we use nowadays has not enough possible addresses for all the devices. That is why IPv6 is going to be implemented. The fact that devices are not directly addressable is a problem because if a device wants to send a message to another device there is no possible way to do it directly if the message has to be transmitted through NAT. Thanks to the implementation of the IPv6 this problem will gradually disappear, as with IPv6 the range of IP addresses will increase considerably.

With IPv4 the IP addresses have 32 bit, so the range is 2^{32} IP addresses. With IPv6 the format of the IP addresses will change and they will have 128 bits, so the range will be 2^{128} possible IP addresses, this number is sufficient to assign an IP address to every device connected to the Internet. Meanwhile with IPv4 the devices are behind a NAT box, which uses address translation. A NAT box is usually a router that has a public IP. It has some other agents which share its public IP but when we want to establish a communication between agents, the problem is that routers reject most of the incoming connection attempts.

Despite the implementation of IPv6 there still will be lots of devices which will be behind NAT boxes because of the difficulty of implementing a new IP version. We can not assume that all routers and application are IPv6 compatible, so the problem trying to connect agent behind NAT boxes will still exists.[6]



Routers have a NAT(network address translator) table inside, they use this tables to manage and route the traffic properly. Adjacent routers share their NAT tables, so in the case that a communication between agents which are in local networks it would not be a problem to establish a communication because they can know the IP of the other agent thanks to NAT tables. The problem appears when an agent out of the local network wants to establish a connection to an specific agent which is behind a router. The mobile agent (the agent out of the local network) does not know the IP of the agent which wants to connect to. It can only access directly the device with the public IP, the router. There is also a problem with the ports that allow traffic because routers usually have the ports for incoming connections attempts closed for security reasons. In summary, when we try to establish a communication between two agents behind NAT boxes we have these two problem, the inability of access directly to the agent, and the problem of the NAT box firewall that will reject the connections attempts. As it is said in [9], an agent is a piece of software that acts for a user or other program in a relationship of agency (http://en.wikipedia.org/wiki/Software_agent, 07/2011).

In order to solve these problems there are mechanism to establish communication between agents that are not in the same local network. If you want to connect one agent behind a NAT box and another behind another nonlocal NAT box you can configure your router NAT table and firewall to allow traffic through the ports you want to use. This solution has some problems. It is necessary to have administrative permissions and also is a static solution. If you change the IP of one agent this solution would not work unless you change the configuration and your

firewall rules.

Another way to establish a communication between the agents is to use a relay intermediate server that both agents behind NAT boxes have access to. The relay server then will forward the package from one agent to another. The problem with this solution is that the relay server has a lot of traffic load and also the communication depends on a third agent all the time, so if the bandwidth of the relay server is limited the communication could be very slowly.

To solve the scenario raised in this thesis I have used a hybrid model of the P2P protocol, as it is explained in [3]. I have used a webserver agent, which is between the application agent and the mobile agent, which provides enough information to the mobile agent to connect with the application agent, so that it can start a communication directly with the application agent. I have done a similar solution as the TURN(Traversal Using Relay NAT) or STUN server-client protocols made, as it is explained in [10] and [7]. My solution will be explained in section 3.

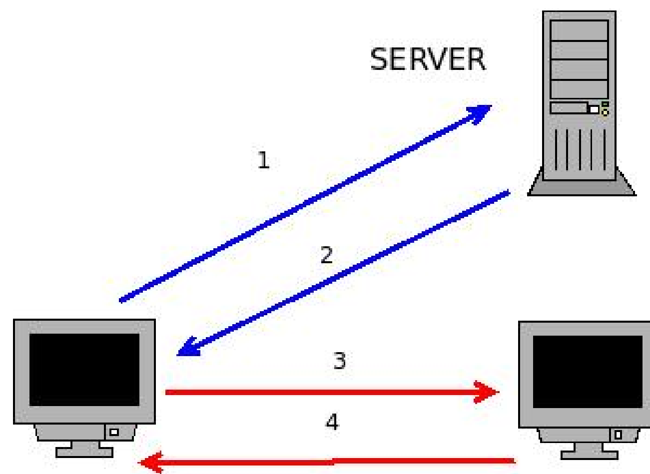


Figure 2: Hybrid P2P solution

As you can see in figure 2, a hybrid P2P solution uses an intermediate agent that helps the agents to establish the communication. First of all the agent that wants to start the communication asks the intermediate agent for the necessary information. After receiving this information, the agent can send a message to the other agent and both can communicate with each other other without more help of the intermediate server.

P2P application and other software have lots of problems when they want to establish connections with a specific device behind a NAT box without knowing the IP of it. In order to solve this, we can use some solutions such as UDP hole punching, TCP hole punching and other solutions in order to exchange information with the agents so they are able to communicate with each other directly even when they are behind NAT boxes. There are some projects and applications that have achieved this, I am going to do a little introduction of the solutions and

techniques that have been done.

As it is said in [4], the key to establish a connection through NAT boxes is that the remote host knows which public port and IP address has been assigned by the NAT for a given flow. To do that they use a STUN server[10](Simple Traversal of User Datagram Protocol Through Network Address Translators). The problem is that the STUN server knows the public IP and port that has been assigned by the NAT for the flow between the host and the STUN server, but STUN have to make guesses about what public IP and port will be assigned for the next flow with the other agent(http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.134.5527; http://www.guha.cc/ saikat/pub/fdna04-nutss.pdf).

There are different ways of how NATs treat the port assignment. This could be a problem when an agent wants to do hole punching. Some NAT boxes assign the public port to the same as the local port if the local port was not already assigned to another flow. These NATs are called Port Preserving Cone NATs, this kind of NATs allow hole punching. On the other hand, there is another type that is called Symmetric NATs that assign a different port for every new flow. This would not be a problem if the Symmetric NATs assign port numbers in uniform increment, usually 1 or 2, in that case is possible to predict the next port number assignment. But not all Symmetric NATs work like this. So in the case that the NAT assign public ports randomly, making a port prediction is impossible.

One of the techniques used to establish connections between two agents behind NAT boxes is UDP hole punching. Various proprietary protocols, such as those for on-line gaming use UDP hole punching. As it is explained in [1], this technique consists of two steps. The first step consists of each agent that wants to communicate discovers the type of NATs and firewalls between them and the Internet, to do that they use a Rendezvous Server. They send a message to the Rendezvous Server, when an agent registers with the Rendezvous Server, the server records two endpoints for that client: the (IP address, UDP port) pair that the client believes itself to be using to talk with the server, and the (IP address, UDP port) pair that the server observes the client to be using to talk with it. We refer to the first pair as the client's private endpoint and the second as the client's public endpoint. In second step the public IP address and the NAT behavior obtained in the first step are used to predict the address and port number for the future connection between the agent. When one agent receives the public and private endpoints of the other agents it wants to communicate with, the agent start sending UDP packets to both of these endpoints, it finally uses the endpoint which first return back a valid response(http://www.usenix.org/events/usenix05/tech/general/full_papers/ford/ford_html/).

UDP hole punching does not compromise the security policy of most NATs, but, as it is explained above, it does not work always due to the lack of a standard NAT. TCP hole punching is also used by some applications. The process is similar but it is used for TCP connections.

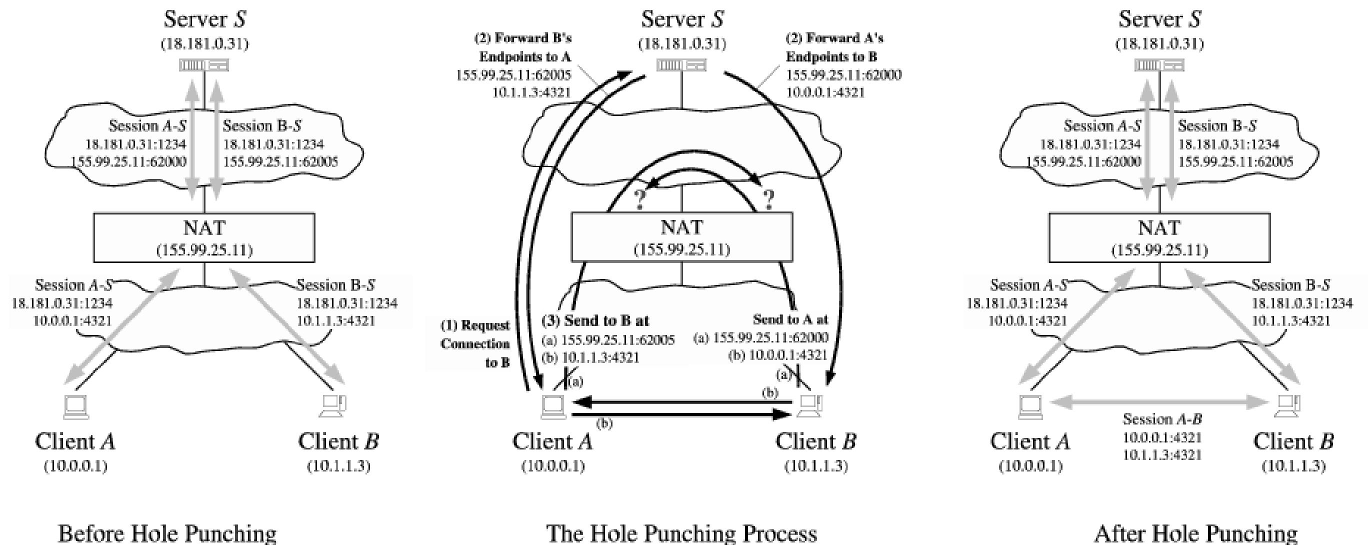


Figure 3: UDP hole punching process, source: http://www.brynosaurus.com/pub/net/p2pnat/

Lets show some situations where NAT is a problem.

If we have some files that we want to share with some other people NAT does not allow people to connect to our computer. This situation is solved by programs like Azureus or eMule for example in [8]. Another situation could be when you want to establish a connection to do a video conference with another agent. You can not do that because, most of the times, you do not know the IP of the other agent you want to connect. You should previously know the IP address and the port which the agent will be listening to, also the agent has to open a hole in its firewall, after this process you are able to establish a connection and start the stream of data with the agent.

If we want to establish a connection with an agent that is behind two local subnets we have another problem. Local subnets in most of the cases uses the same private IP addresses, so if we want to send a message to an agent behind two local subnets it could happen that our message goes to an agent that is in the first local subnet because there is one agent in the first subnet with the same IP address that the agent that was supposed to receive the message.

The main goal of this work is to find a solution to a scenario where an agent wants to establish communication with another agent which is running an application. To achieve that, we have to solve some problems and use similar techniques that are explained above in this chapter. The first problem we have to deal with is the fact that the application agent has not a global IP, so it is not directly addressable for the agent which wants to connect. The second problem that occur is that our agent which runs the application is in a local subnet which is not connected directly to the Internet. This fact makes the scenario more difficult because it can be problems dealing with IPs.

2 Description of the problem

The scenario solved in this thesis is an agent based distributed system. There are three agents involved in this scenario. We have an application agent which is running our application and another agent which wants to access it. There is another agent which has a public address. Every agent needs to know the address of the other agents to establish a communication between them. So we have a problem because two of the three agents have private addresses. When the communication takes place in a local subnet, the remote agent can be accessed via a local IP. A solution is to use multicast messages in order to publish a communication address in a local subnet. But when the agent is out of the local network and tries to communicate with one which has a private address establishing a communication is more difficult for the following reasons.

The first problem is that we can not use the multicast messages to go through the routers barriers when we try to communicate with agents out of our local network. Apart from this problem we also have another problem, such as the impossibility to access the other agents because of the configuration of the router or the problem that we can not access directly to an agent which does not have a public IP. There are techniques and mechanism that solve the problem of routing messages from different subnet which are not in the same local network.

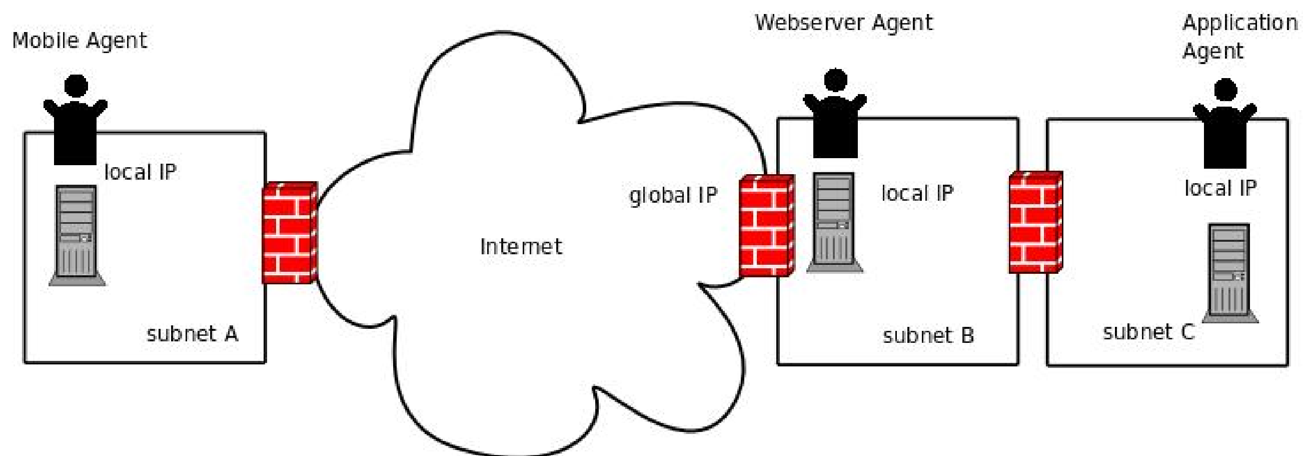


Figure 4: scenario

Problem: As you can see in figure 4, our Application agent is on a separate subnet (subnet C) and it is no possible to access to it directly. There is a Webserver agent which has a global address, it can be directly addressed, and it also has a local address in the subnet B. The Webserver runs a web application that is used to interact with the Application agent. The scenario that we want to solve is that a Mobile agent on a subnet A connected to the Internet can communicate with the Application agent only via the global address of the Werbserver agent.

We want to find a solution with which communication between the agents can be made as if all had public address. To achieve that, the solution has to avoid the problems of network address translation and the limitation of multicast message.

3 Overview of the solution

In the solution I use three different agents. There is a webserver agent which is responsible to help the mobile agent and the application agents to establish the connection. An application agent which has to be in contact with the webserver agent and publish its port and its IP and a mobile agent which starts the communication and wants to access to the application.

The main goal of the webserver agent is help the two other agents to establish the communication. This agent works similar as a STUN server [7]. It has an updated list of all the active application agents, when an application agent goes down it has to delete that application agent of the list. When it receives an attempt of connection from a mobile agent there are two possible ways to continue with the process. The first way is returning to the mobile agent the list of the active application agents which the mobile agents could connect. The second way is that the webserver agent sends directly a request to the application agent and the communication is established. The return of the webserver agent depends on the type of request done by the mobile agent.

The application agent has to search for the webserver agent. It already knows in which subnet the webserver agent is, so it has to send broadcast messages to this subnet with the port which the webserver agent will be listening to and the IP of its network. Apart from doing that, it has to be listening for a request for connection from the webserver agent. When it receives a request of connection from the webserver agent it has to check which are the port and the IP of the mobile agent which wants to connect to it and send a message to that mobile agent. The IP and the port of the mobile agent are in the request message received from the webserver agent.

The mobile agent sends a message to the webserver agent, which has a public IP and port. When it tries to connect to the webserver agent, it can use a parameter to specify the identifier of the application agent which it wants to connect. If in the connect method an identifier is provided, the webserver agents connects directly with the application agent. If not, the webserver agent responses with the list of the available application agents. When the mobile agent receives the list, it chooses for the agent which is running the application that it wants or, if there are more than one agent running the application, it checks which is the more suitable agent. After establishing a connection, the mobile agent and the application agent keeps the channel with activity to avoid NAT boxes to close the channel.

From my point of view the most important is the webserver agent. It has the responsibility to help the other agents to establish the communication. So, without it, the two other agents are not able to reach each other. It is also important to remark that once the communication between the application agent and the mobile agent is established it is independent from the webserver agent.

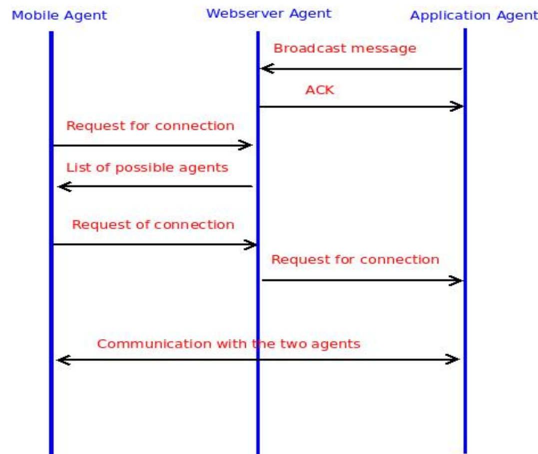


Figure 5: Overview of the operation

As you can see in figure 5, after the webserver agent provides the information of the application agent to the mobile agent and sends to the application agent that a mobile agent wants to connect with it, the communication between the agents is possible without using a relay server because both agent know the IP and the port of the other agent.

So the main goal of the webserver agent is to provide enough information of the application agent to the mobile agent and then do the same with the information of the mobile agent to the application agent. After the two agents have the information they can communicate directly with each other through their NAT boxes.

Using this solution we are able to have different agents running applications in a local subnet which is not connected to the Internet but they can provide service to agent out of our local network as they were "on-line". This is very useful if we are interested in having our agents less vulnerable to malevolent attacks from the external network, so they are not "visible" before you get the list of agents from the webserver agent.

This solution could be very useful if we want to offer several on-line applications but we only have one public IP address. Using the solution presented here, we can have only one agent with a global IP which manage the connection attempts to the application agents and as many as we want of agents running the applications.

Before doing this solution other possible solutions were taken into account, but at the end for several reasons (administrative permissions, efficiency or traffic load in an agent), this was the solution that I have done.

One possible solution could have been to use the webserver agent as a relay server and forward every message from the mobile agent to the application agent but this solution is less efficient and also makes the webserver agent have a lot of traffic load.

Another solution could have been do some scripts that do port forwarding from the first local net to the second local net, but to do this administrative permissions are needed so I could not do that.

4 Description of the communication protocol

In this section the steps of the main agents involved in the communication protocol will be explained and some pictures which illustrates the steps to make easy the understanding will be provided. Also the goals of every agent will be explained.

4.1 Application agent

The application agent is behind a private network that the mobile agent can not access directly, so before the two agent can communicate directly with each other we have to use some method to make this agent directly addressable.

First of all, the application agent connects with the webserver agent in order to provide the webserver agent its IP and port which it will be listening on. After that, the webserver agent registers the PORT and the IP address in an array of application agents. To do this the application agent sends a broadcast message to the network where the webserver agent is with its PORT and its IP address. When the webserver agent receives the message it sends an ACK message to the application agent. After doing that the application agent waits for a request from the webserver agent that will contain the IP address and the PORT of the mobile agent that want to connect to one application agent.

When the webserver agent receives the request from the mobile agent, it sends a request to the application agent with the IP and port of the mobile agent. Afterwards the application agent can start the communication directly to the mobile agent. This solution is similar of how a STUN server works [7].

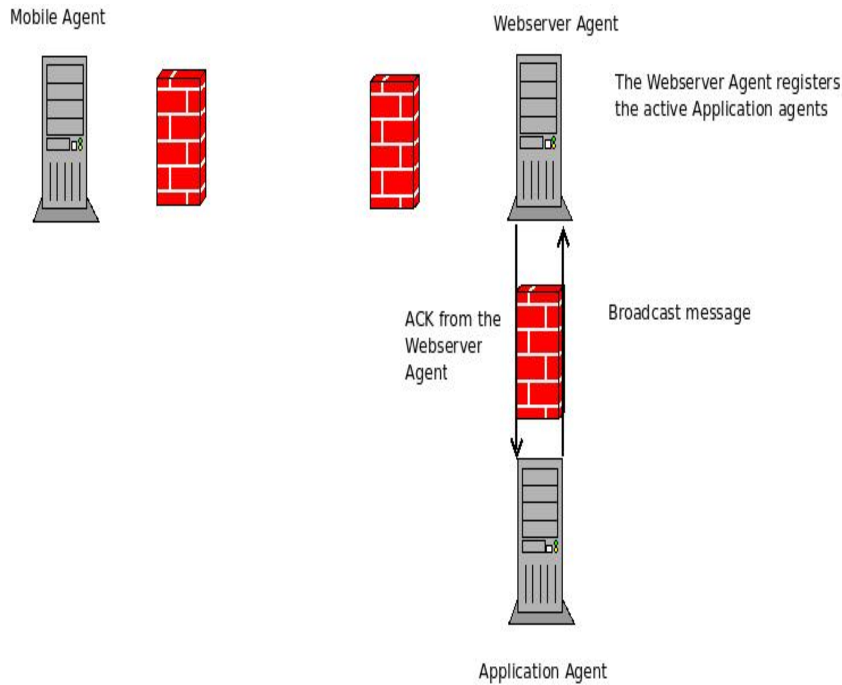


Figure 6: Application Agent : Step 1

The application agent sends a message to the webserver agent every ten seconds in order to signal that it is available and ready to receive connections from a mobile agent. In that messages its port and IP address are included. The messages have to be sent every ten seconds because there is a thread that every forty seconds deletes the application agents from the list if the webserver agent has not received any message indicating that this agent is available.

Apart from doing that, the application agent is waiting for a request, when it receives a request from the webserver agent it reads the data of the package, that package contains the IP and the port of the mobile agents is trying to connect to it. Once it has the information of the mobile agent the application agent sends a message to the mobile agent.

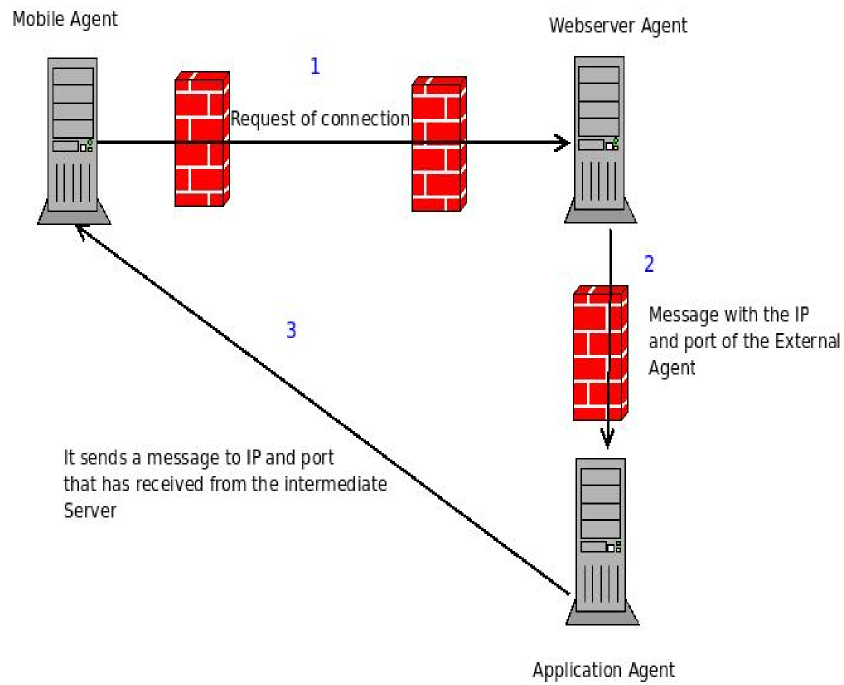


Figure 7: Application Agent : Step 2

This agent is running an application. The fact that is not directly connected to the Internet is a problem because, as it is explained in section 2, in local subnets we have private IP addresses and they are most of the times the same, so we have a problem of duplicate IP addresses in local subnets.

4.2 Mobile agent

The mobile agent is also behind a NAT box, it tries to connect to the webserver agent in order to get a connection to an application agent, it has two options. It can send an identifier to automatically get a connection to one existing application agent, or it can request the updated list of the application agents available and then request a connection to one of them.

First of all, it sends a request to the webserver agent. As it is explained in section 3, there are two possibilities of doing that, I am going to explain the possibility which the mobile agent first ask for the list of active application agents and, after receiving the list, it establishes a connection to the agent which is running the application it wants to connect.

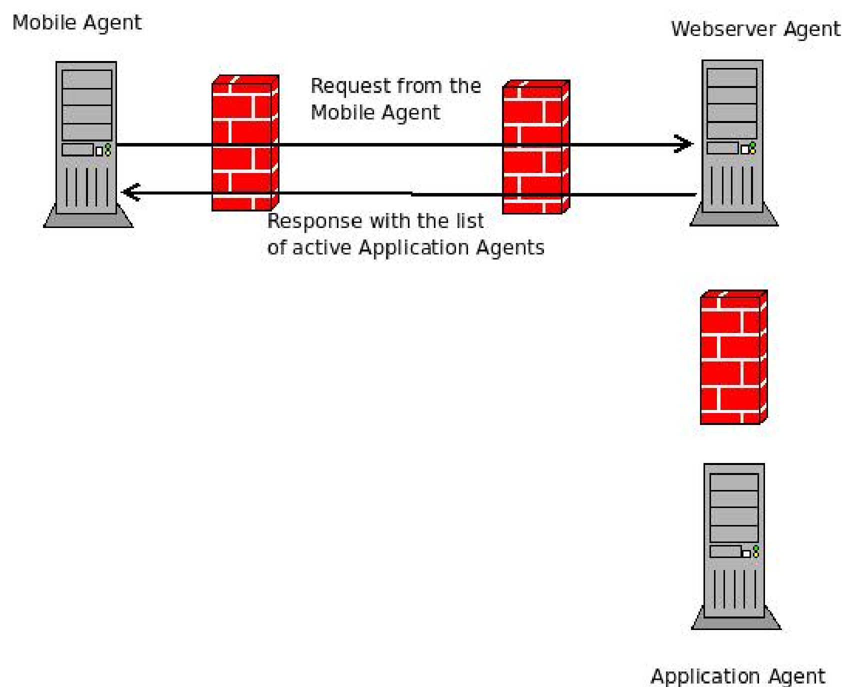


Figure 8: Mobile Agent : Step 1

The mobile agent sends a request to the webserver agent, with the message already containing its IP and its port. When it receives a response from the webserver agent with the list of the available application agents, it chooses an application agent and sends another message to the webserver agent with the identifier of the application agent it wants to connect to and waits for a response of the application agent.

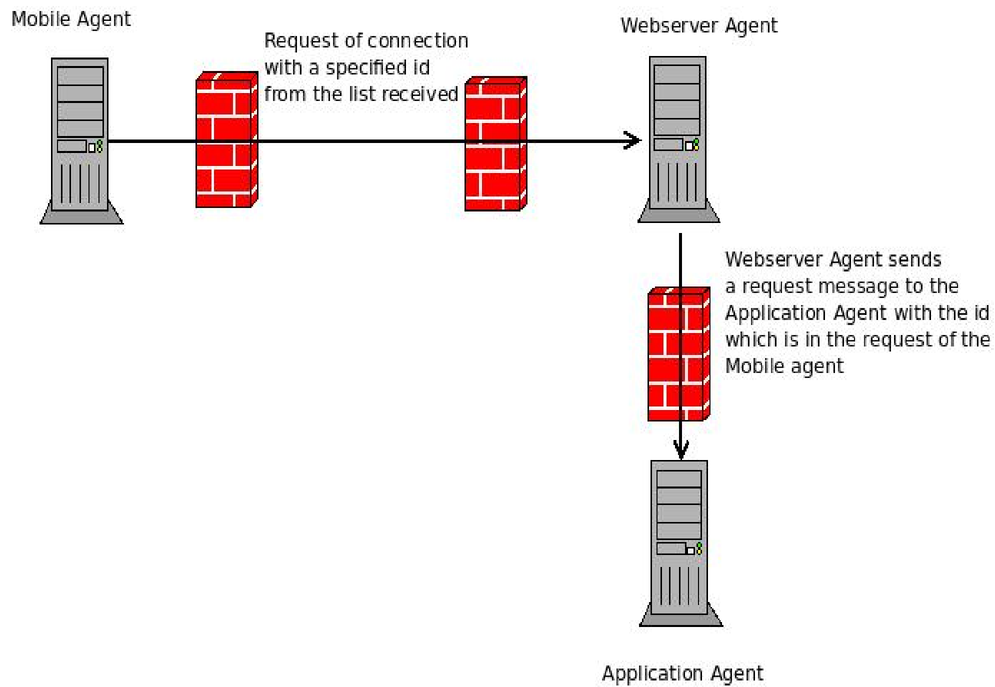


Figure 9: Mobile Agent : Step 2

After receiving the response message from the application agent the communication is established and is the responsibility of both agents, the mobile and the application agents to keep that channel opened in order to avoid the closure of that channel by the NAT boxes through the communications due to inactivity.

To keep the channel opened the mobile agent sends a message every twenty seconds if there is no activity in the channel. Every time there is activity through that channel, a timer, that is always running in the mobile agent, is reseted to zero.

4.3 Webserver agent

The webserver agent is the essential agent for establishing communication between the other agents. First, it has to have an updated list of the available application agents that are within its local network and provide to the mobile agents, when necessary, the list. It has a public IP and port where the mobile agents can connect to. In this case the webserver agent is winterser.dai-lab.de. And the mobile agent tries to connect to 6062.

It has two sockets, one for the internal connection (the application agents) which are behind its NAT and one for the incoming connections from the mobile agents.

When it receives an internal connection it registers the IP and the port of the application agent to a list. It also saves the number of connections to that application agents, to enable possible future load balancing to avoid application agents to be very busy and distribute the traffic load over the available application agents. When it receives an incoming connection it checks if the connection request has an identifier. If it has one, it search for a registered application agent with that identifier and sends the request message of an incoming connection to the application agent. If the incoming connection has not an identifier the webserver agent sends the list of the available application agents to the mobile agent and waits for a connection with an identifier. If the incoming connection has an identifier but there is no application agent with that identifier the webserver agent sends an error message to the mobile agent.

To keep the list of application agents updated the webserver agent uses the following mechanism. When it receives a connection from one application agent it registers that agent as available and save the number of connection to the application agent. There is one thread that every twenty seconds check all the available agents list and sets their availability to false. If the next time the thread checks the list and finds that there is any application agent that has not set its availability to true, the thread deletes it from the list. The time I use for this thread is twenty seconds because then, there is very little possibility of one active application agent being deleted when it should not be. For example, lets explain the worst case. At moment X the webserver agent receives one broadcast message from an application agent, it registers it and and moment X+1 the thread sets the availability of the agent to false. It could happen that if the time was ten that the thread will run before the message of the application agents is received and the availability would not be set to true. In that case one application agent would be deleted when it should not be. That is why the thread runs every twenty seconds. This duration could be changed but I have use this duration in order no to overload the traffic between the webserver agent and the application agents.

To send the request to the application agent, it takes the IP and the port from the message of the mobile agents and it puts that information in the data of the message for the application agent.

5 Implementation

In this section I am going to explain and justify how I have implemented my solution and show the classes and methods I have used to implement the agents.

5.1 Mobile agent

The mobile agent is implemented with a set of classes. In figure 10 the classes that implement the mobile agent are shown and after that there is a little explanation of the behavior of every class.

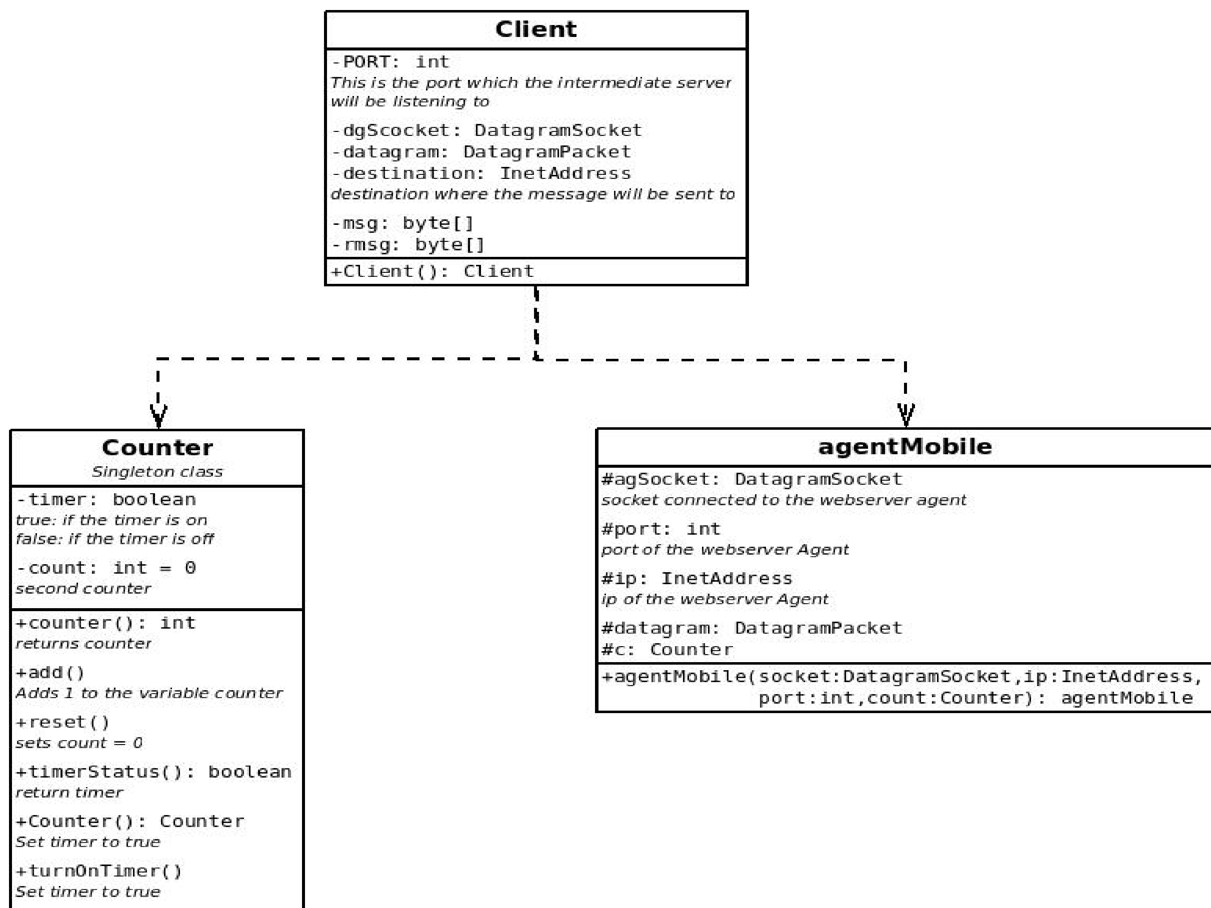


Figure 10: Mobile Agent

The class Client

The class Client is in charge of establishing the connection with the webserver agent and receive the first message of the application agent.

The class agentMobile

The class agentMobile is a thread class that sends packages to the webserver agent and keeps the channel open between the mobile agent and the application agent. Every time there is data exchanged between these agents the counter is set to zero. This class has a creator method and a run method. It is called by the Client class.

The class Counter

The class counter is the responsible of controlling the seconds elapsed. This class is used by the mobile agent and also by the application agent. The class Counter has six methods. One creator method which initializes a new Counter. Then there are modifier and consultant methods.

The **counter()** method returns the number of seconds that have passed.

The **add()** method is called when a second has passed and it changes the variable counter to counter+1.

The **TimerStatus()** method returns the state of the counter, it returns true if it is switched on or false if it is switched off.

The **reset()** method sets the counter variable to zero, this method is called when there is data exchange between the application agent and the mobile agent.

The **turnOnTimer()** method starts the counter of seconds.

The Class timer

This class is a thread that runs every second and is it used to count the seconds elapsed. This class is used by the agentMobile and sends every twenty seconds a message if there is no data exchange between the application mobile and the mobile agent in order to avoid NAT boxes from closing the channel. As a thread it has two methods, the creator and the run method.

The Interface IClient

The mobile agent has an interface that is called IClient. The interface has 3 important methods.

The first method is called connect. This method makes a connection to the specified IP and port that has provided in the parameters, this IP and port are the IP and port of the webserver agent which will provide to the mobile agent the list of the available application agents.

list<Node> connect(IP: String, port: int);

After this method is called, the mobile agent waits for a list of application agents with different information about every webserver.

The second method is called connect too, but it has different parameters, this method sends a request to the webserver agent with an identifier to which application agent wants to connect to.

void connect(IP: String, port: int, id: int);

After this method is called, the mobile agent wait for a message from the application agents that has the identifier id.

The third method is called close. This method close the connection with the application agent.

void close();

This method sends a message to the application agent saying 'end' and close the connection with it.

5.2 Application agent

The application agents has three classes that implement it.

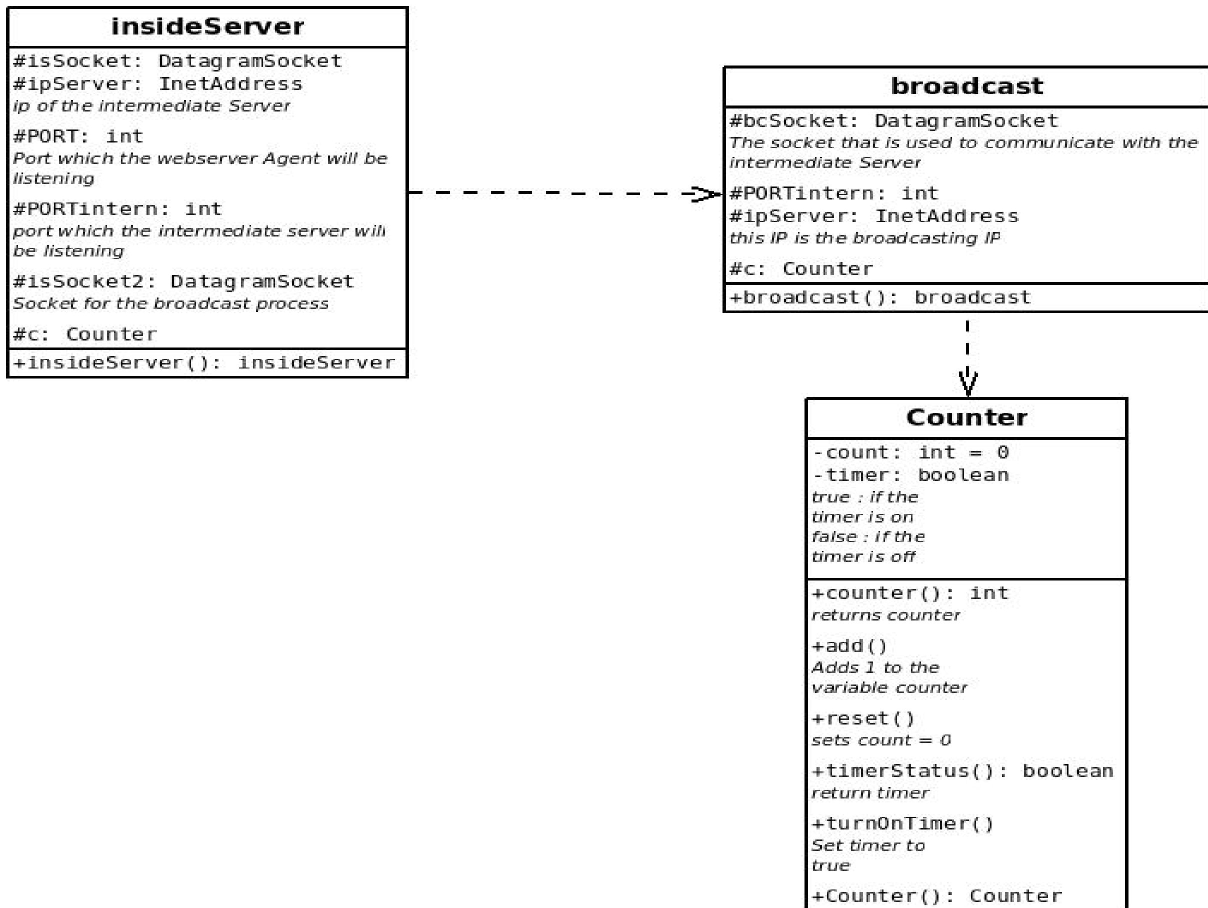


Figure 11: Class insideServer

The class insideServer

The class insideServer is a thread class. It has two sockets, one which will be the socket for the broadcasting process. This socket is used to send message to the webserver agent saying that the application agent is active and another socket for the communication with the mobile agent. This class also receives a request from the webserver agent and reads the information of it. After doing this, it sends a message to the mobile agent, whose IP and port are in the request message data. It has two methods, the creator and the run method. This class calls the broadcast class.

The class broadcast

The class broadcast sends a broadcast message every ten seconds through the network where the webserver agent is. Doing this allows the webserver agent to know if the application agent is available. It has a creator method and a run method. It is called by the insideServer class. This class also uses the Counter and timer class to control how many seconds has passed.

The Interface IinsideServer

The interface IinsideServer provide one method for registering the application agent to the webserver agent and another method to unregister the application of the webserver agent.

The methods are like this.

void register(IP: String)throws Exception;

void unregister() throws Exception;

The IP of the method register is the IP of the webserver agent where the application agent would be registered. This method will send a structure with information of the application agent. Here in this document I mention only an identifier, but it could be the name of the application, some information about the computer or other information that could be useful for a mobile agent to chose to which server it wants to connect.

For example, we have 5 application agents and every agent runs a different application. In this situation it would be necessary to give the webserver agent some information about the application is running in every application agent so that it can provide this information to the mobile agent and then the mobile agent chose the right application agent. Another situation that could be imagined nowadays could be that there are one application agent running the application for the mobile devices (smartphones, tablets, PDA's) and another application agent is running the same application but modified for the other devices, then it will be very useful to add information in the struct saying which agent is for mobile devices and which for the others.

5.3 Webserver agent

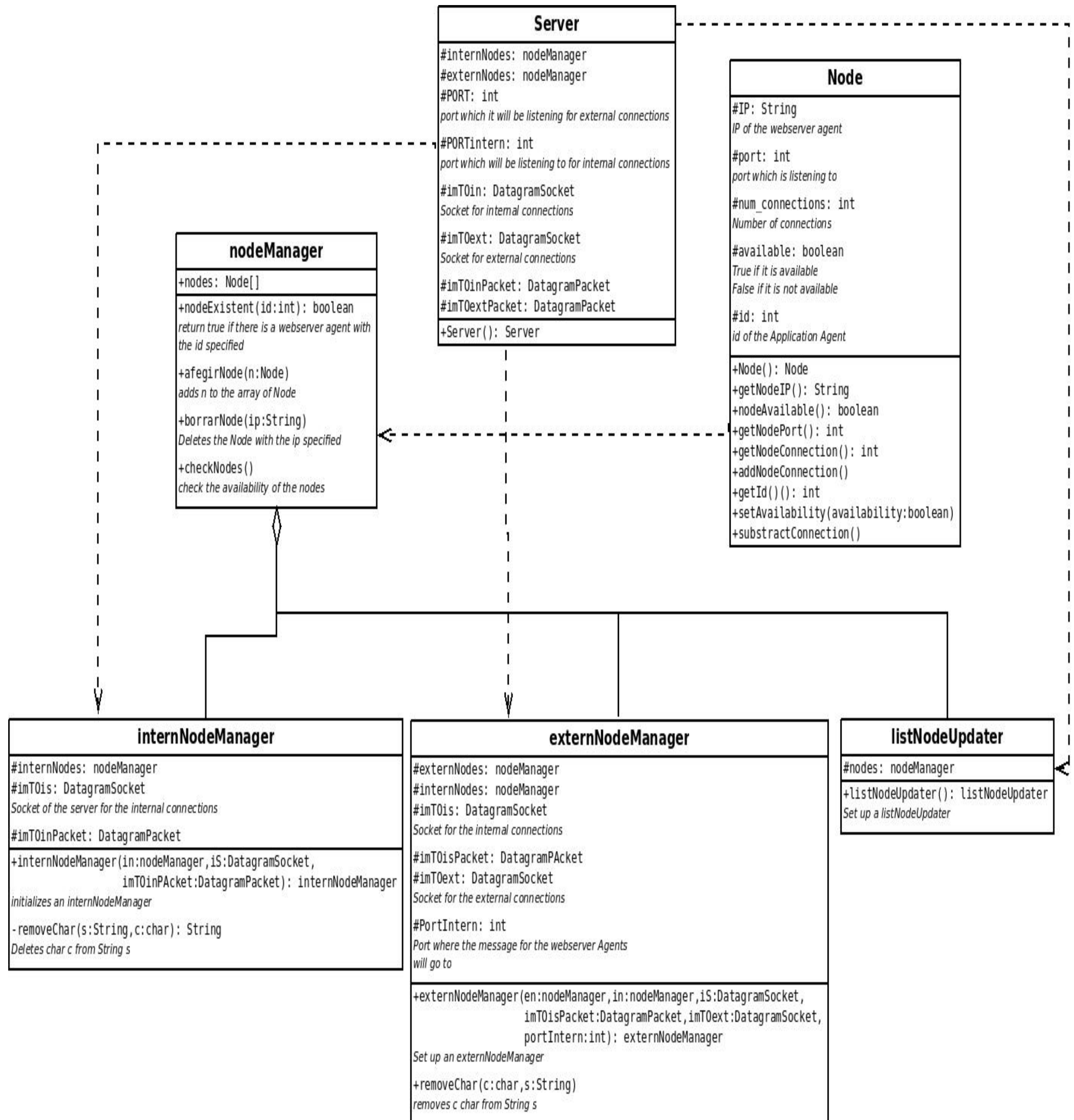


Figure 12: Webserver Agent Diagram

The class Server

This class is the main class of the webserver agent. This class is in charge of opening the two socket that will be listening to the incoming connections of the mobile agents and the incoming connections of the application agents. It will register into a list of Node the active application agents. This class calls internNodemanager, externNodemanger and listNodeUpdater and then keeps listening for incoming connections. It has a creator method and a run method, it also has a private method that removes a char from a String in order to make easier for the application agent to read the data sent.

The class Node

This class has the information about an application agent. This class as it is explained above, section 5.2, could be modified depending on the situation of the scenario. It has a creator method and also some modifier and consultant methods.

The **getNodeIP()** method returns the IP of the application agent.

The **getNodePort()** method returns the port which the application agent is listening to.

The **nodeAvailable()** method returns the availability of the application agent, true if it is available or false if not.

The **getNodeconnection()** method returns the number of connections to the application agent.

The **addNodeConnection()** method adds one connection to the application agent.

The **getId()** method returns the identifier of the application agent.

The **setAvailability(availability:boolean)** method sets the availability of the node to availability parameter value.

The **substractConnection()** method subtracts one connection of the application agent.

The class NodeManager

This class is an array of Nodes. It has four methods.

The method **checkNodes()** method works like this, it looks all the application agents, if the application agent's availability is true, it turns it to false, and if the availability is false the node is deleted from the array.

The **nodeExistent(id:int)** method returns true if there is an application agent with the id given as a parameter otherwise it returns false.

The **afegirNode()** method registers an application agent to the array.

The **borrarNode()** method deletes an application agent of the array.

It is used by internNodeManager and externNodeManager.

The class internNodeManager

This class is used by the Server class, it is a thread class and is the responsible to add to a NodeManager all the application agents that connect to the webserver agent.

The class externNodeManager

This class is used by the Server class , it is thread class and is the responsible to send the request to the application agent that has the id which the mobile agents wants to connect. The request has the IP and the port of the mobile agent.

The class listNodeUpdater

This class is a thread class, every twenty seconds checks the availability of the application agents registered for the webserver agent, it uses the following process. If the availability of the application agents is true, it sets the availability to false, and if the availability of the application agent is false then the application agent is deleted from the list. This class is used by the Server class and has a creator method and a run method.

6 Related work

6.1 NUTSS

NUTSS is a SIP-based(Session Initiation Protocol) approach to UDP and TCP network connectivity. NUTSS uses NAT traversal by using STUN server and a proxy server that is accessible from the two agents that want to connect with each other. First of all I am going to explain what is STUN and its components.

6.1.1 STUN

As it is explained in [10], STUN(Session Traversal Utilities for NAT) is a client-server protocol. It supports two types of transactions. One is a request/response transaction in which a client sends a request to a server, and the server returns a response. The second is an indication transaction in which either agent - client or server - sends an indication that generates no response. STUN protocol is used by some VoIP application and also by mobiles. A possible configuration of a STUN protocol would be like this.

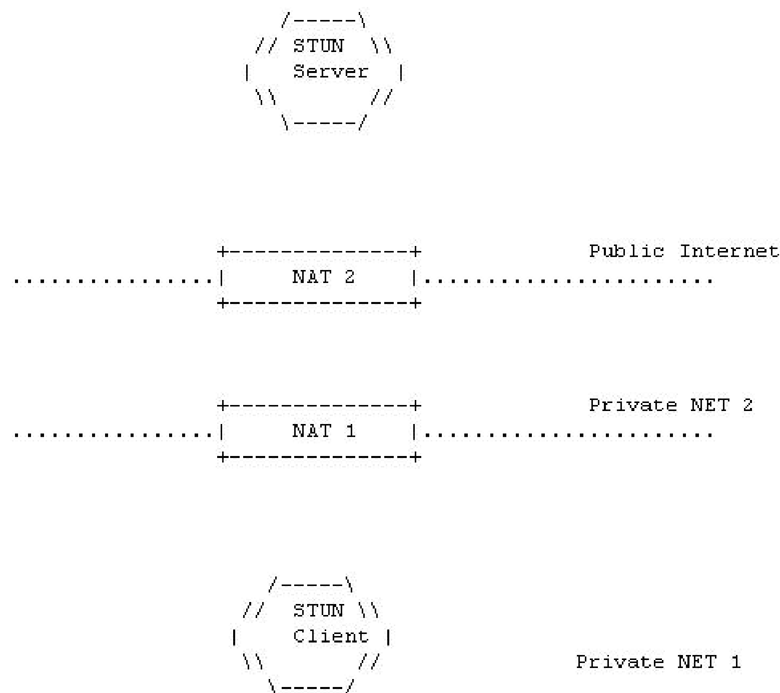


Figure 13: STUN configuration, source : http://www.ietf.org/rfc/rfc3489.txt

In this configuration, we have two entities that implement the STUN protocol. The lower agent in the figure is the client, and is connected to a private network 1. This network connects to private network 2 through NAT 1. Private network 2 connects to the public Internet through NAT 2. The upper agent in the figure is the server, and resides on the public Internet(http://tools.ietf.org/html/rfc5389).

STUN Server

A STUN server, as it explained in [10] is a server that receives STUN requests and sends STUN responses. Both STUN request and STUN response are a type of packet named STUN and have a special configuration(http://tools.ietf.org/html/rfc5389).

STUN Client

A STUN client is an entity that sends STUN request and receives STUN responses from the STUN server.(http://tools.ietf.org/html/rfc5389)

Explanation of the algorithm used by STUN

First of all, the STUN client sends a message to the STUN server. After that, the STUN server returns to the STUN client its public IP and the port that is opened. From now on, the STUN client knows its public IP and the port it has to listen, so the STUN client could send to another hypothetical STUN client his public IP and the port that the other agent could send the messages and start a communication with each other directly, this is how some VoIP applications round firewalls and manage to connect through firewalls, applications such as Skype use this solution.

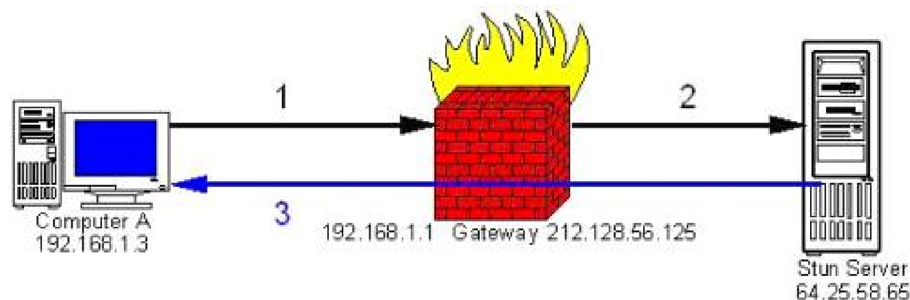


Figure 14: STUN Algorithm source: http://www.3cx.com/blog/voip-howto/stun-voip-1/

In figure 14 you can see the steps explained the the paragraph above.

After explaining how STUN works and which are the components of the protocol I am going to explain how NUTSS establish a TCP connection through NAT boxes.

NUTSS uses a proxy server that is accessible from both agents. First they get theirs global IP and global port from the STUN server, the STUN servers could be independents. After that, one host sends to the proxy server his global IP and global port and open a raw socket, a socket that allows direct sending and receiving of network packets by applications, in the global port. The other agent does the same. The proxy then exchange the global port and IP, it sends the global IP and port from A to B and the global port and IP from B to A. After doing this step, both hosts can communicate each other directly. The full procedure is described in RFC [4].

7 Conclusion

In this thesis an approach to a cross subnet communication is explained. I have looked into some previous techniques of how P2P applications and other applications are able to establish connections through NAT boxes. I have used these techniques as a basis for the solution presented in this work. This solution is very useful when a point-to-point communication is required by an application or for a specific scenario. But I think that with the implementation of the IPv6 these techniques will be less useful because there will be more devices with global IPs. Even though, I think that they will not become obsolete because there will be devices that will still be behind NAT boxes.

7.1 Security

There are several very interesting techniques such as UDP hole punching and TCP hole punching[1]. These allow to open temporary holes in firewalls and solve the problem of the closed port of the NAT boxes very good. The problem with these solutions is that you have always to use an intermediate server as a relay that helps the agents behind NAT boxes to exchange their IP and ports. So there is a security problem, maybe the relay server could use this information to do illegal activities or somebody could steal information of the agents.

To avoid this problem there are some applications that use username and passwords, such as NUTTS[2] [4] or STUN[10]. The problem still exists, because someone could hack the relay server and then get the IP and port and use this information for illegal activities.

7.2 Standards

There is a problem with the existing NAT techniques, as there are different types of NAT. This is a problem for people who try to apply protocols or try to deal with NAT. This is because different methods may have to be applied for different NAT types. There should be a standard NAT technique so then all software and techniques will work always and for developers it will be easier. [5]

7.3 Future developments in technology

In the next few years I think that the paradigm of the Internet is going to undergo strong changes with the implementation of the IPv6. But anyway solutions of exchanging IP addresses and ports between agents will still make sense, because although there will be enough IP for all

the devices there will be always devices that, for security reasons or another reasons, would be behind another device that will filter the traffic or protect the device from malevolent attacks.

References

[1] B. Ford, P. Srisuresh, and D. Kegel, *Peer-to-peer communication across network address translators*, USENIX Annual Technical Conference, vol. 2005, 2005.

[2] Paul Francis, *Is the internet going NUTSS?*, IEEE Internet Computing **7** (2003), no. 6, 94–96.

[3] Heinz Waldemar Herlitz Gatica, *Transversalidad en nat/firewalls*, Master's thesis, Universidad Catolica de temuco Facultad de ciencias, 2005.

[4] Saikat Guha, *NUTSS: a SIP-based approach to UDP and TCP network connectivity*, 2004.

[5] Saikat Guha and Paul Francis, *Characterization and measurement of tcp traversal through nats and firewalls*, 2005.

[6] Juha Lehtovirta, *Transition from ipv4 to ipv6*.

[7] J. Rosenberg, J. Weinberger, and C.R.Mahy Huitema, *" stun-simple traversal of user datagram protocol (udp) through network address translators (nats)*, Tech. report, RFC 3489, March, 2003.

[8] Jürgen Schmidt, *http://www.h-online.com/security/features/how-skype-co-get-round-firewalls-747197.html*.

[9] wikipedia, *http://en.wikipedia.org/wiki/software_agent*, 2011.

[10] D. Wing, P. Matthews, R. Mahy, and J. Rosenberg, *Session traversal utilities for nat (stun)*, 2008.