sip2peer Tutorial
Bootstrap, FullPeer & SBC

Marco Picone

Università degli Studi di Parma
Parma, Italy

Outline

- Introduction
- FullPeer and Bootstrap
- FullPeer Configuration File
- FullPeer run arguments
- SBC and NAT Management
- FullPeer and SBC

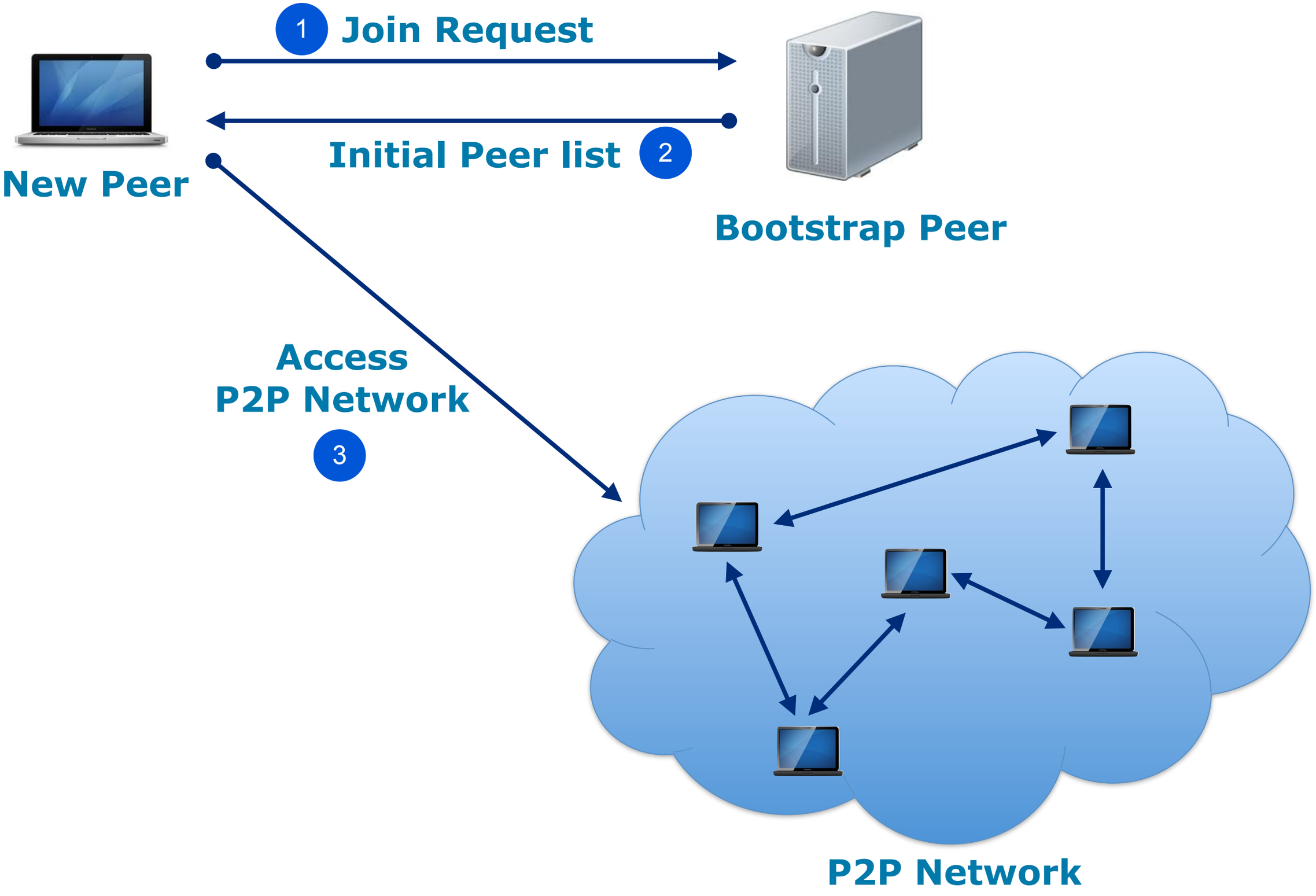
Bootstrap

"A bootstrapping node, also known as a rendezvous host, is a node in an overlay network that provides initial configuration information to newly joining nodes so that they may successfully join the overlay network.

Bootstrapping nodes are predominantly found in decentralized peer-to-peer (P2P) networks because of the dynamically changing identities and configurations of member nodes in these networks."

WIKIPEDIA
The Free Encyclopedia

FullPeer & Bootstrap


1
Join Request
Initial Peer list
2
New Peer
Bootstrap Peer
Access
P2P Network
3
P2P Network

sip2peer Example

BootstrapPeer (BootstrapPeer.java)

- *Example of simple BootstrapPeer Implementation*
- Receives JoinMessage from a new Peer
- Saves the PeerDrescriptor of the new node in a list
- Sends back to the node a list of PeerDescriptor of active peers.

FullPeer (FullPeer.java)

- Sends a JoinMessage request to the Bootstrap with its PeerDescriptor
- Receive the list of PeerDescriptor from the boostrap
- Exchange messages with discovered peers

FullPeer Configuration

```
via_addr=AUTO-CONFIGURATION
host_port=5075
peer_name=kate
test_address_reachability=no
log_path=log/
req_npeer=10
bootstrap_peer=bootstrap@192.168.1.101:5080
sbc=160.78.28.112:6067
keepalive_time=5000
debug_level=1
```

FullPeer uses two additional configuration parameters that can be defined in the configuration file as showed in the example and that could be read using a user-defined object that extends the ***Configure*** base class.
In the example that class is called ***PeerConfig*** and allows to read ***req_npeer*** used the define the number of peer descriptors requested to the bootstrap and ***bootstrap_peer*** to define the address of the bootstrap.

PeerConfig.java

```
public class PeerConfig extends Configure
{
    ...
    public PeerConfig(String file){
        // load configuration
        loadFile(file);
    }
    protected void parseLine(String line)
    {

        String attribute;
        Parser par;
        int index=line.indexOf("=");
        if (index>0) {  attribute=line.substring(0,index).trim(); par=new Parser(line,index+1);  }
        else {  attribute=line; par=new Parser("");  }


        if (attribute.equals("bootstrap_peer"))      {  bootstrap_peer=par.getString();  return;  }
        if (attribute.equals("req_npeer"))      {  req_npeer=par.getInt();  return;  }
    }
}
```

Basic approach to read a sip2peer configuration file. It can be used to add new parameters based on users' needs and node specifications. In this case it is used to read the address of the bootstrap peer and the number of peer descriptors that has to be requested to it with the JoinMessage.

PeerConfig object is used by FullPeer inside the init method specifying the path of the configuration file:

```
private void init(String pathConfig){
        this.peerConfig = new PeerConfig(pathConfig);
        ...
}
```

FullPeer.java

FullPeer's code could work in different modalities in order to show several aspects of the sip2peer library. Looking at the main method inside FullPee.java class there are different parameters configuration that can be used to run the node in different ways. At the moment we will analyze the two main arguments configurations with some additional values that don't involve SBC and NAT problem that will be tackle at the end of the tutorial.

Base Configuration

3 Arguments:

OR

5 Arguments:

+

Additional Arguments

```
//args[0]=file peer configuration args[1]=key
peer = new FullPeer(args[0], args[1]);
```

```
//args[0]=file peer configuration args[1]=key args[2]=peer name args[3]=peer port
peer = new FullPeer(args[0], args[1], args[2], new Integer(args[3]));
```

-j `peer.joinToBootstrapPeer();` join to bootstrapPeer

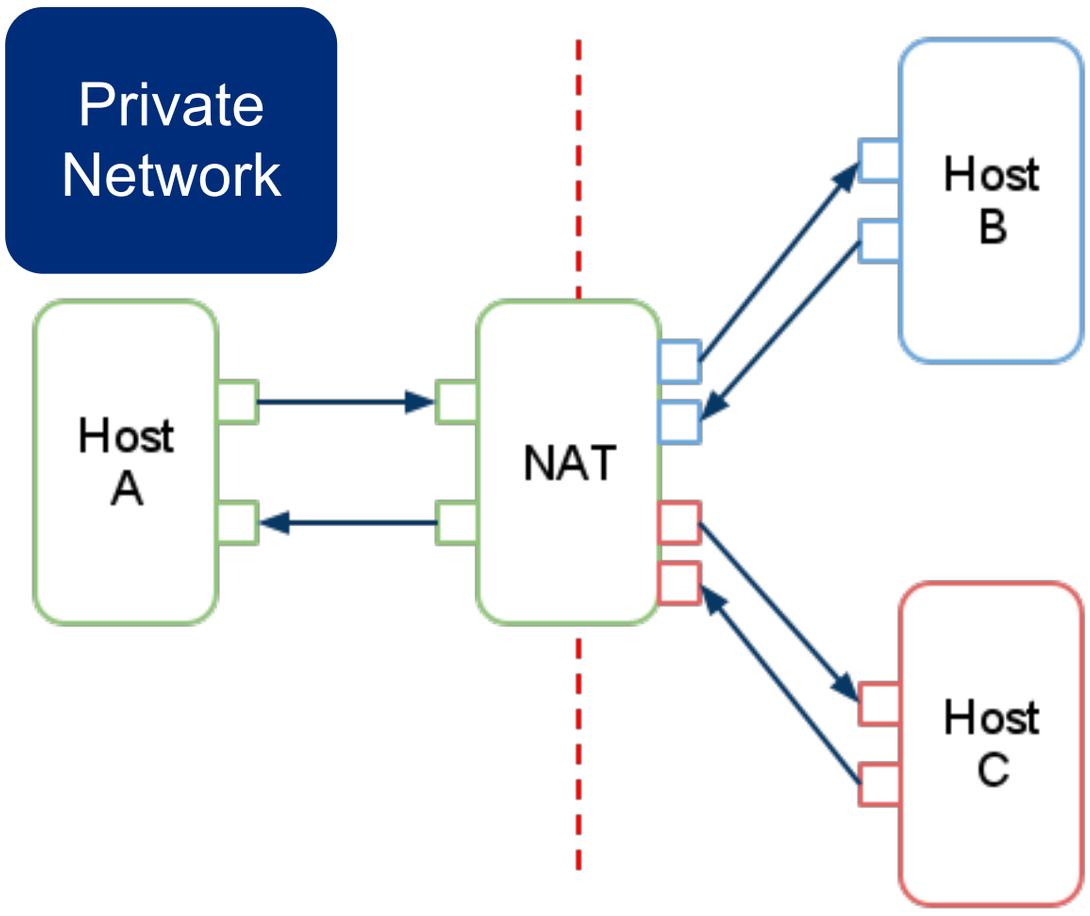
-jp `peer.joinToBootstrapPeer();` **+** `peer.pingToPeerRandomFromList();`

-jr `peer.joinToBootstrapPeer();` **+ recursively** `peer.pingToPeerRandomFromList();`

NAT

Network Address Translation (NAT) is the process of modifying network address information in datagram (IP) packet headers, while in transit across a traffic routing device, for the purpose of remapping one IP address space into another. It is a technique that hides an entire IP address space, usually consisting of private network IP addresses (RFC 1918), behind a single IP address in another, often public address space.

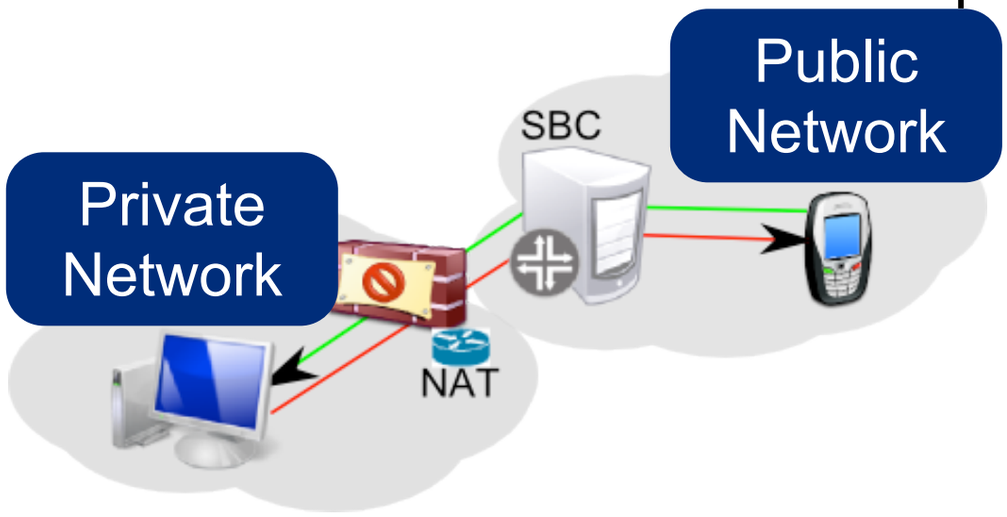
Nowadays NAT is a very common element in computer networking and in particular for peer-to-peer application may represent an big obstacle for the communication.



Example of symmetric NAT

NAT & SBC

In VoIP (Voice over Internet Protocol) networks, a device that is regularly deployed and used to solve NAT traversal problem is the **Session Border Controller (SBC)**. Being sip2peer based on SIP, SBC represents a natural and easily way to solve NAT traversal problems.
Shortly, we can say that in our specific case SBC is a node with public IP that allows a generic peer to check if it is behind a NAT and to request (if necessary) a public IP and port that can be used by the requesting node as contact address and that can be advertised to other peers.



The sip2peer library natively includes an SBC implementation (sip2peerSBC.zip) that can be easily configured and executed on a public IP machine.

Run SBC Node

Sip2PeerSBC.zip file available on sip2peer official page and on Google Code provides our implementation of an SBC peer. In order to run the SBC (on a public IP) and solve NAT problems use the configuration file called ***sbc.cfg*** available in the SBC folder and pass the configuration file name as an argument of the main class called "**SessionBorderController.java**".

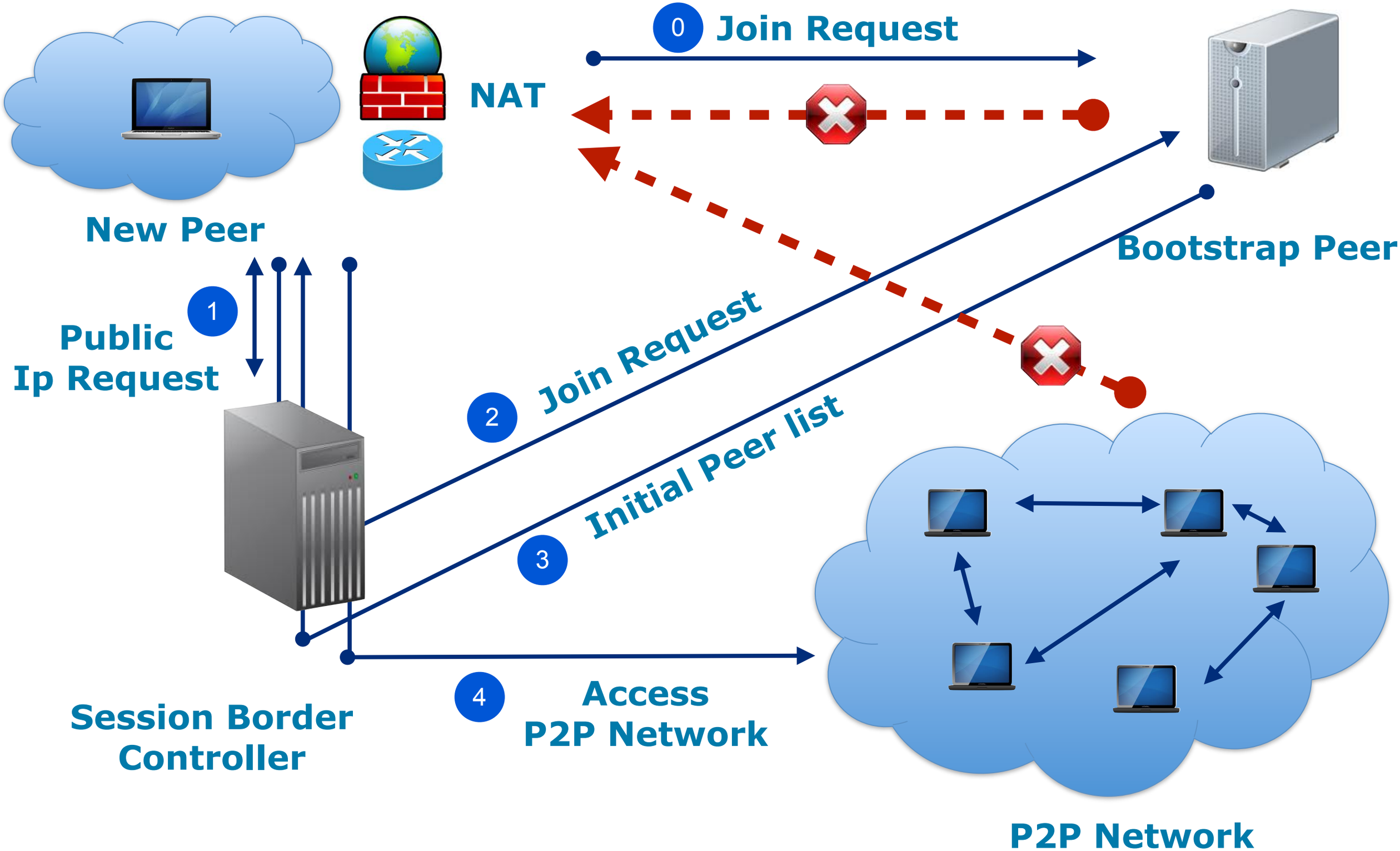
sbc.cfg

```
via_addr=AUTO-CONFIGURATION
host_port=6066
transaction_timeout=2000
test_nat_port=6079
max_gwPeer=15
init_port=6080
debug_level=0
```

SessionBorderController.java

```
public static void main(String[] args) {

      if(args.length!=0){

            String pathFile = "config/"+args[0];

            SipProvider sipProvider = new SipProvider(pathFile);

            ServerProfile serverP = new ServerProfile(null);

            @SuppressWarnings("unused")
            SessionBorderController sbc = new SessionBorderController(sipProvider, serverP, pathFile);
      }
}
```

FullPeer, Bootstrap & SBC


0
Join Request
NAT
New Peer
Bootstrap Peer
1
Public
Ip Request
2
Join Request
3
Initial Peer list
4
Access
P2P Network
Session Border
Controller
P2P Network

FullPeer.java

FullPeer example has some additional arguments used to show how to contact a running SBC on public IP in order to solve NAT related problems. By contacting the SBC, the sip2peer node negotiates and obtains a new **ContactAddress**, that automatically sets in its PeerDescriptor; at this point, the latter can be distributed to other nodes allowing them to communicate with the peer behind the NAT.
You can analyze and reuse the implemented methods to introduce the same functionalities in your sip2peer based projects.

SBC Additional Arguments

-s

```
peer.contactSBC();   request public address from SBC
```

-sd

```
contact SBC, wait a while and then close the opened public ip from the SBC
peer.contactSBC();  +  peer.disconnectGWP();
```

-a

```
contact SBC, wait, join to bootstrapPeer, wait and send ping message to random peer.
peer.contactSBC();  +  peer.joinToBootstrapPeer();  +  peer.pingToPeerRandomFromList();
```

License

http://www.gnu.org/licenses/gpl.html

Contact us for specific releases or additional information about the license.

Contacts

Designer(s)

- Marco Picone (picone@ce.unipr.it)
- Fabrizio Caramia (fabrizio.caramia@studenti.unipr.it)
- Michele Amoretti (michele.amoretti@unipr.it)

Developer(s)

- Fabrizio Caramia (fabrizio.caramia@studenti.unipr.it)
- Marco Picone (picone@ce.unipr.it)

http://dsg.ce.unipr.it

Contacts

http://dsg.ce.unipr.it/?q=node/41

Distributed Systems Group
Dip. di Ingegneria dell'Informazione
Università degli Studi di Parma

About us
- DSG Home
- People
- Research
- Publications
- Students

Projects
- DSG Projects
  - DEUS
  - DGT
  - P2P & Network Coding
  - PATROL
  - sip2peer

sip2peer
SIP-based API for robust connection and communication among peers.

Search projects

Project Home | Downloads | Wiki | Issues | Source | Administer

Summary | Updates | People

Tip: Discuss and then document each teammate's project duties.

Project Information

Star project

Activity: Medium

Code license
GNU General Public License v3

Labels
Android, p2p, peer-to-peer, api, java, Middleware

Feeds
Project feeds

Owners
picone.m
michele.amoretti
fabri.caramia

Committers
0 committers

Contributors

Description

sip2peer is an open-source SIP-based middleware for the implementation of any peer-to-peer application or overlay without constrains on peer nature (traditional PC or mobile nodes) and specific architecture.

Main characteristics are:

- Multiplatform nature
- Simple communication API with notification system
- SIP-Based platform
- NAT traversal management
- Efficient, Scalable and Configurable structure

Supported Platform

At this moment sip2peer is available for Java SE and Android platforms, but we are working on the iOS release.

Featured

Wiki pages
sip2peerTutorial
Show all »

http://code.google.com/p/sip2peer/