

**Self-organized, Fault Tolerant, Peer-to-Peer Management of  
SNMP Devices**

**By:**

**Michael Da Silva**

**100286974**

**Carleton University**

**COMP 4905 – Honour's Project**

**Supervisor: Dr. Tony White, School of Computer Science**

**August 13, 2004**

## **Abstract**

The goal of this project was to build a Peer-to-Peer environment that can be used for the management of SNMP devices based upon the JXTA framework. The environment supports the ability to self-organize based upon failures by individual management devices.

The Java, Peer-to-Peer, based framework (JXTA) was used with SNMP agent software to have the network capable in organizing itself to keep the management layer alive. A self-organization algorithm that works with the JXTA framework was designed and is implemented in the demonstration application.

## **Acknowledgements**

I would like to thank Dr. Tony White, of the School of Computer Science at Carleton University, for allowing me to investigate the topic and for his guidance in the preparation of this project and report.

I would also like to thank Jonathan Sevy for his SNMP Agent work that was used in the demonstration application of this project.

## Table of Contents

1. Introduction .....	6
1.1 Peer-to-Peer Background .....	6
1.2 JXTA Background.....	7
1.2.1 Peer .....	8
1.2.2 Protocols.....	8
1.2.3 Propagation Pipe .....	9
1.3 SNMP Background .....	10
2. Implementation.....	14
2.1 Concept behind the design of the application.....	14
2.1.1 Peer .....	14
2.1.2 Manager.....	15
2.1.3 Self-organization Algorithm Design.....	15
2.1.4 SNMP Communication.....	18
2.2 Use Cases .....	19
2.2.1 Scenario: Many peers with agents and 1 manager.....	19
2.2.2 Scenario: Many peers with agents and no manager.....	20
2.2.3 Scenario: Peers attempting to organize.....	21
2.2.4 Scenario: Peers determining who should manage .....	22
2.2.5 Scenario: Peers have organized themselves.....	22
2.3 Design Decisions.....	23
3. Problems Encountered.....	26
4. Conclusion .....	27
5. Glossary .....	29
6. References.....	31
7. Appendices .....	32
7.1 Running the Application.....	32
7.2 Ideal network setup .....	33

## List of Figures

Figure 1 – Peer-to-Peer network diagram.....	7
Figure 2 - The pipe advertisement XML file, SNMPNetworkManagerPipeAdv- SNMPNetworkManager-Prop.adv, used in the demonstration application. ....	10
Figure 3 – SNMP query over UDP .....	12
Figure 4 – SNMP communication between manager, peer and agent. ....	19
Figure 5 – Ideal setup of the network.....	20
Figure 6 – Manager was dropped, network needs to organize.....	21
Figure 7 – Peers attempt to self-organize.....	22
Figure 8 – Peers sending one another their number.....	22
Figure 9 – Network has been organized.....	23

# 1. Introduction

## 1.1 Peer-to-Peer Background

*Generally, a peer-to-peer (or P2P) computer network refers to any network that does not have fixed clients and servers, but a number of peer nodes that function as both clients and servers to the other nodes on the network.*

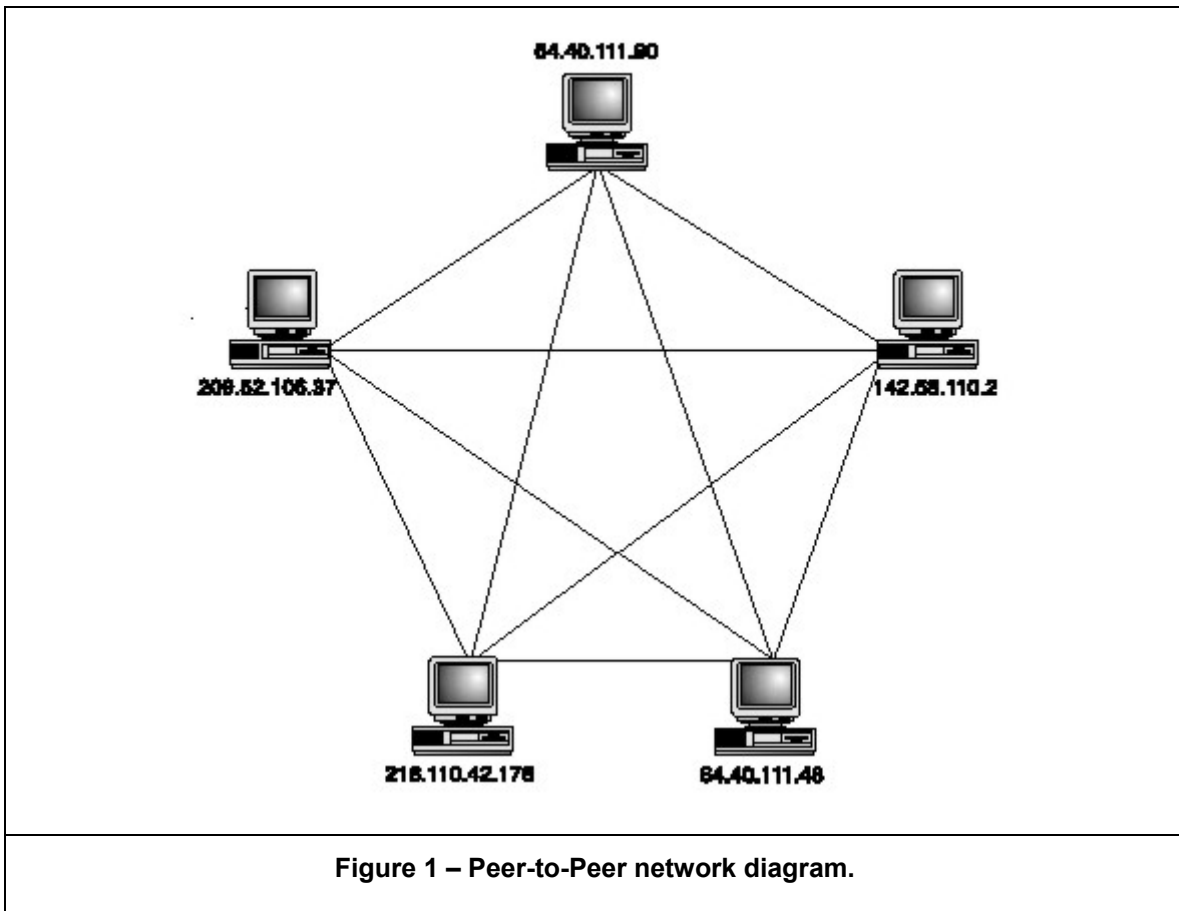
(<http://www.free-definition.com/Peer-to-peer.html>)

As stated in the definition above, a peer-to-peer computer network is a network that does not have fixed clients and servers. The idea is that each peer (a node) on the network is capable of functioning as both a client and a server. Therefore, any peer is able to initiate or complete any transaction that a typical client-server model network would handle.

The main advantage of peer-to-peer networks is that they distribute the responsibility of providing services among all peers on the network without having to rely on a single peer. This in turn eliminates service outages caused by a particular point of failure hence providing a more robust solution of offering services. Furthermore, peer-to-peer networks allow sharing of bandwidth across the entire network by using various communication channels rather than overloading a specific route that is common in traditional client/server communication. By taking a variety of network routes, network congestion would be reduced.

Peer-to-peer networks' many advantages have made the model very popular, especially in file sharing networks such as Gnutella. Such advantages are that there need not be a dedicated server to service client' requests. Since each peer on the network acts as a client and server, requests can be made by one peer and typically any other peer can get it, process it, and return a response.

Looking at Figure 1, each computer represents a peer on the network. Each of these peers is aware of one another and there is no single peer acting as a dedicated server. Requests are made from one peer to another.



Peers can communicate with one another in a peer-to-peer network by means of protocols. These protocols are different from one network to another and are quite often proprietary to that network.

## 1.2 JXTA Background

Realizing the need for a common Peer-to-Peer language, Sun Microsystems developed JXTA, an open source Peer-to-Peer framework. *“JXTA is a set of open protocols that allow any connected device on the network ranging from cell phones and wireless PDAs to PCs and servers to communicate and collaborate in a P2P manner”*. ([www.jxta.org](http://www.jxta.org))

The JXTA framework uses a defined set of protocols to allow communication between Peers where a peer is defined as,

Any entity capable of performing some useful work and communicating the results of that work to another entity over a network, either directly or indirectly.  
(Wilson)

### **1.2.1 Peer**

Each peer in the JXTA network is assigned a unique identifier (peer ID). When a peer is started up, it automatically joins what is known as the 'NetPeerGroup'. All peers that are part of the network are members of this peer group. A peer group is JXTA's way of dividing up the network into separate communities that ideally would serve a separate purpose. Along with the 'NetPeerGroup', each peer can belong to more peer groups in which the peers cooperate and function similarly and under a unified set of capabilities and restrictions. JXTA provides protocols for basic functions such as

- creating groups
- finding groups
- joining groups
- leaving groups
- sending messages

All of these functions are performed by publishing and exchanging XML advertisements and messages between peers.

Through the JXTA protocols, a peer can locate other peers and other peer groups to join. Again using these protocols, a pipe can be opened, a very simple one-way message queue, to another peer or group of peers. By using pipes, distinct parties can communicate with each other.

### **1.2.2 Protocols**

Two of the important protocols in peer communication are the Peer Discovery protocol and the Pipe Binding protocol.

- The peer discovery protocol outlines the format of messages to request peer advertisements and to respond to request for peer advertisements from other peers.
- The pipe binding protocol defines the set of messages that allow a peer to bind a pipe to communicate with another peer.

The idea is that a peer publishes an advertisement on the network providing its details for other peers to discover and be able to contact them. Once peers discover one another, a peer can publish or obtain a pipe advertisement and make a connection to it. Communication pipes such as this are essential for peer-to-peer communication.

### **1.2.3 Propagation Pipe**

Peers that make connections to this pipe can then send and receive messages to other peers that have made the bindings to the pipe as well. This is possible by using a specific type of pipe, the propagation pipe.

Propagation pipes provide the ability to broadcast data to multiple peers that have bindings to the pipe. Therefore, if one peer wants to send a message out to other peers, it can simply send it through the pipe and all the other peers that are bound to it will receive the message. This differs from regular pipes where only one peer would receive the message. An example of a propagation pipe advertisement is shown in Figure 2. This is one of the propagation pipes used in the demonstration application. More specifically, it is used for management communication on the network.

```
<?xml version="1.0"?>
<!DOCTYPE jxta:PipeAdvertisement>
<jxta:PipeAdvertisement xmlns:jxta="http://jxta.org">
  <Id>
    urn:jxta:uuid-
    8AEC0E0FA18C4023B4C997FDB19D0EF2B08776184F134445B8D4A75B7B1C34D304
  </Id>
  <Type>
    JxtaPropagate
  </Type>
  <Name>
    SNMPNetworkManagerPipeAdv-SNMPNetworkManager-Prop
  </Name>
</jxta:PipeAdvertisement>
```

**Figure 2 - The pipe advertisement XML file, `SNMPNetworkManagerPipeAdv-SNMPNetworkManager-Prop.adv`, used in the demonstration application.**

### 1.3 SNMP Background

The Simple Network Management Protocol (SNMP) was designed to reduce the complexity of network management and minimize the amount of resources required to support it.

By specifying the protocol to be used between the network management application and management agent, SNMP allows products (software and managed devices) from different vendors (and their associated management agents) to be managed by the same SNMP network management application.

The main attributes of SNMP are as follows

- It is simple to implement, making it easy for a vendor to integrate it into its device.
- It does not require large computational or memory resources from the devices that do integrate it.

Each agent has the responsibility of collecting information (e.g., performance statistics) pertaining to the device it resides within and storing that information in the agent's own management information base (MIB). This information is sent to the SNMP manager in response to the manager's commands.

Networks can quickly become very complex both in terms of the number of nodes that are involved and in terms of the various protocols that can be running on any one node. Even when dealing with a school's network for instance, one would need to keep track of many routers and lots of hosts and their specific information. Maintaining all of these nodes is the problem of network management, an issue that affects the entire network architecture. SNMP eases the task of network management and provides the flexibility to interact with and manage vendor-specific information.

Distributed nodes used in these networks can therefore be managed by using a protocol that is capable of reading and writing various pieces of state information. SNMP is the most commonly used protocol, running on top of the User Datagram Protocol (UDP), to handle managing these nodes. SNMP supports two kinds of request messages, GET and SET.

- GET\_REQUEST – Used for retrieving data from a node.
- SET\_REQUEST – Used for setting data on a node.

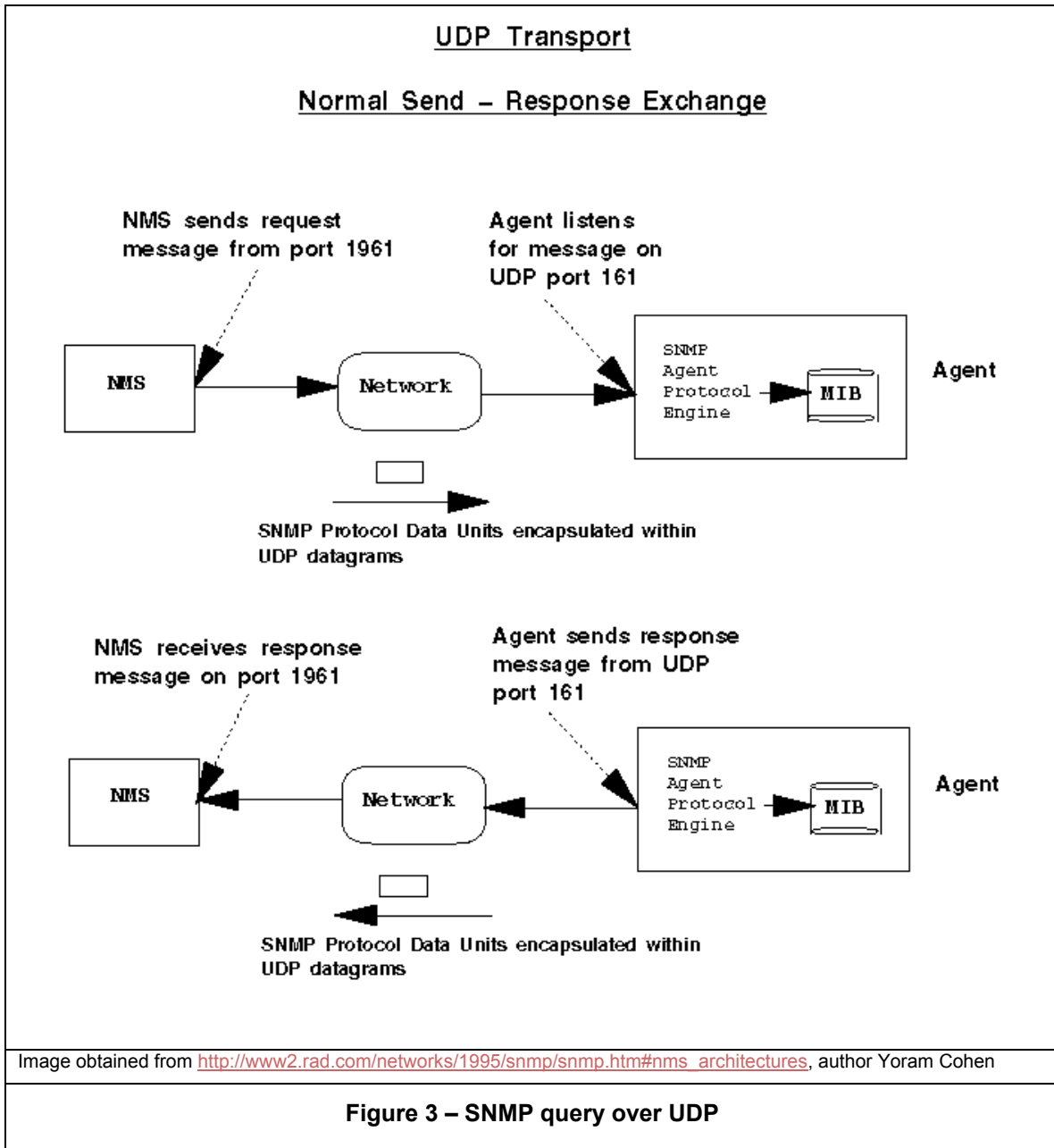
SNMP also has a response message.

- GET\_RESPONSE – Used for returning the requested information.

Each node has software running on it that is known as an SNMP Agent. The agent is what receives SNMP requests, locates the requested piece of information, formats it into an SNMP response message and returns it to the requester.

The request message will contain a globally unique identifier, Object Identifier (OID), that is used to determine what MIB variable should be accessed. This OID is based on ASN.1 and has a dot notation much like an IP address. An example of an ASN.1 OID is 1.3.6.1.2.1.2.1.0 which is a MIB variable 'ifNumber' that contains the number of ports on a router. This example MIB variable is one of many that define specific pieces of information that can be retrieved from a node.

Typically, see Figure 3, network management would consist of a network manager (shown in diagram as NMS) putting together a request message containing the OID for the MIB variable of interest. This message is then sent to the SNMP agent that locates the particular MIB variable corresponding to the received OID. The value of this variable is encoded into a SNMP response message and sent back to the manager.



Details of the MIB and its variables, ASN.1 and the numerous request messages available are beyond the scope of this report. However, more detailed information can be obtained from doing simple searches on the internet.

## 2. Implementation

### 2.1 Concept behind the design of the application

The application can essentially be broken down into four main functions. Firstly, there are peers that are simply members of the network that are running (or know of) agents that need to be managed. Secondly, there is a manager that can access all of these agents through the other peers on the network. Thirdly, there is the organization algorithm implemented on each peer that monitors the network to ensure continuous management. And finally, there is SNMP communication built in that allows for the management of devices.

#### 2.1.1 Peer

The application is designed to demonstrate the ability for the peers to self-organize upon management failure or absence. A peer consists of a basic JXTA client with SNMP agent(s) running on it. A peer has the ability to add or remove an agent by creating one of its own or by adding an agent that is running on another machine. Once agents are added, they will appear in a listing on the peer.

A GUI was added to ease with the usability of the application. On the GUI is a log that keeps the user informed as to what is going on in the background.

Communication between the client and the manager is in the form of messages going through a JXTA communication pipe. Each peer has a input and output pipes to receive and send messages to the manager(s). Peers are made aware of these pipes through the pipe advertisements. A pipe advertisement for the two groups is already available on each peer so they simply need to load it and publish themselves so that others can communicate with them.

Peers can communicate with one another when they are part of the same peer group. All peers (including the managers) in the application are part of the 'SNMPNetwork' group and just the managers are part of the 'SNMPNetworkManager' group. These two groups play an important role in the self-organization of the network.

The 'SNMPNetwork' group is used so all peers can broadcast ping requests to determine if a manager exists. The 'SNMPNetworkManager' group exists so managers can determine if there are other peers acting as a manager as well. The use of these groups is further explained in the following sections.

### **2.1.2 Manager**

A manager is a regular peer with the ability to see all of the other peers on the network and their SNMP agents. The manager has an extra manager UI that allows them to select an agent from one of the peers and send SNMP commands to it. As mentioned in section 2.1.1, the manager is part of the same groups as every other peer but it is also a member of the SNMPNetworkManager group. When a peer is a member of this peer group it is considered a manager.

Managers send their commands over an output pipe that everyone in the SNMPNetwork group will have an input pipe connection to. When a peer receives a command, it responds over a propagation pipe that only managers will have made a connection to. Therefore, only managers are the one's receiving any messages from peers.

Since each peer has the management pack available to them, any peer is capable of being a manager therefore it's important that the self-organizing aspect of the application works properly to ensure the correct number of managers are present on the network (for this implementation there can only be one).

### **2.1.3 Self-organization Algorithm Design**

The idea behind the fault-tolerant, self-organizing algorithm is that each peer is responsible for ensuring that it is being managed. That is, a peer would at all times know if there is a manager on the network. The algorithm is explained in sections with the relevant code shown below each explanation.

Firstly, the peer determines if it itself is a manager or if it has already found a manager on the network. While neither is true, it has no knowledge of a manager, it tries to ping the peers that are part of the SNMPNetworkManager group hoping that a manager will respond to the ping request.

```

if(peer.isManager())
    aliveManager = true;
counter = 0;
peer.log("Pinging the manager(s)");
while(!getAliveManager() && counter < 10) {
    // Ping any active manager(s)
    peer.sendSNMPMessage(SNMPPeer.SNMP_MANAGER_PING, "");
    Thread.sleep(1000);
    counter++;
}
counter = 0;

```

After a set amount of time (chosen for demonstration purposes) and therefore a set number of attempts, the peer goes into a state where it will likely have to try to become a manager.

In this state, the peer joins the management group, creates the management communication pipes it will need to use, generates a number, and publishes itself as a possible manager on the network. Note that at this stage the peer is considered a temporary manager.

```

if(!getAliveManager()) {
    // no manager has replied so try to become one
    peer.log("No ping reply");

    // Join the snmp network manager group
    peer.joinSNMPNetworkManagerGroup();

    peer.log("Joined the SNMPNetworkManager group");

    // Load the SNMPNetworkManager pipe advertisement
    peer.loadSNMPNetworkManagerPipeAdvertisement();
    peer.log("Loaded the manager pipe ad");

    // Create the SNMPNetworkManager input pipe and register the peer
    as a listener
    peer.createSNMPNetworkManagerInputPipe();
    peer.log("Created the manager input pipe");

    peer.createSNMPNetworkManagerOutputPipe();
    peer.log("Creating output pipe");
}

```

```

// Make ourselves known on the network
peer.publishSNMPNetworkManager();
peer.log("Published myself as a possible manager");

// Generate my number
generateRandomNumber();

```

At this point there might be other peers trying to become a manager as well and therefore doing the same thing as this peer. Or, there may even already be a manager but it has not responded due to network delay or heavy traffic. Therefore, the peer must check that while there are other peers in the management group, then it will send them a message requesting to be a manager containing its generate number.

When a peer receives other peer's generated numbers (a management request message), it will check which is greater. The peer with the smaller number will stop trying to become a manager and leave the management group. A peer with the greater number will send each peer that it has received smaller numbers from a message instructing them to leave the management group if they haven't already done so.

```

// Send messages to other peers in the manager group that are
possibly trying to become a
// manager as well...
while(otherManagersExist() && !getAliveManager()) {
    // Send everyone a message with my number
    peer.sendSNMPMessage(SNMPPeer.SNMP_MANAGER_REQUEST,
String.valueOf(this.randomNumber));

    peer.log("Sent other possible managers my number " +
randomNumber);
    Thread.sleep(10000);
}

```

Once the peer has determined that it is the only peer in the management group, it will officially become the manager. In doing so, it starts a manager service and displays the management user interface.

```

if(!getAliveManager()) {

    // If we made it this far then we are going to be a manager
    peer.setManager(true);
    peer.log("I'm now the manager");
    // TODO start the manager (get manager addon)
    peer.startManager();
}

```

```
        peer.snmpManager.showGUI();
    }
}
```

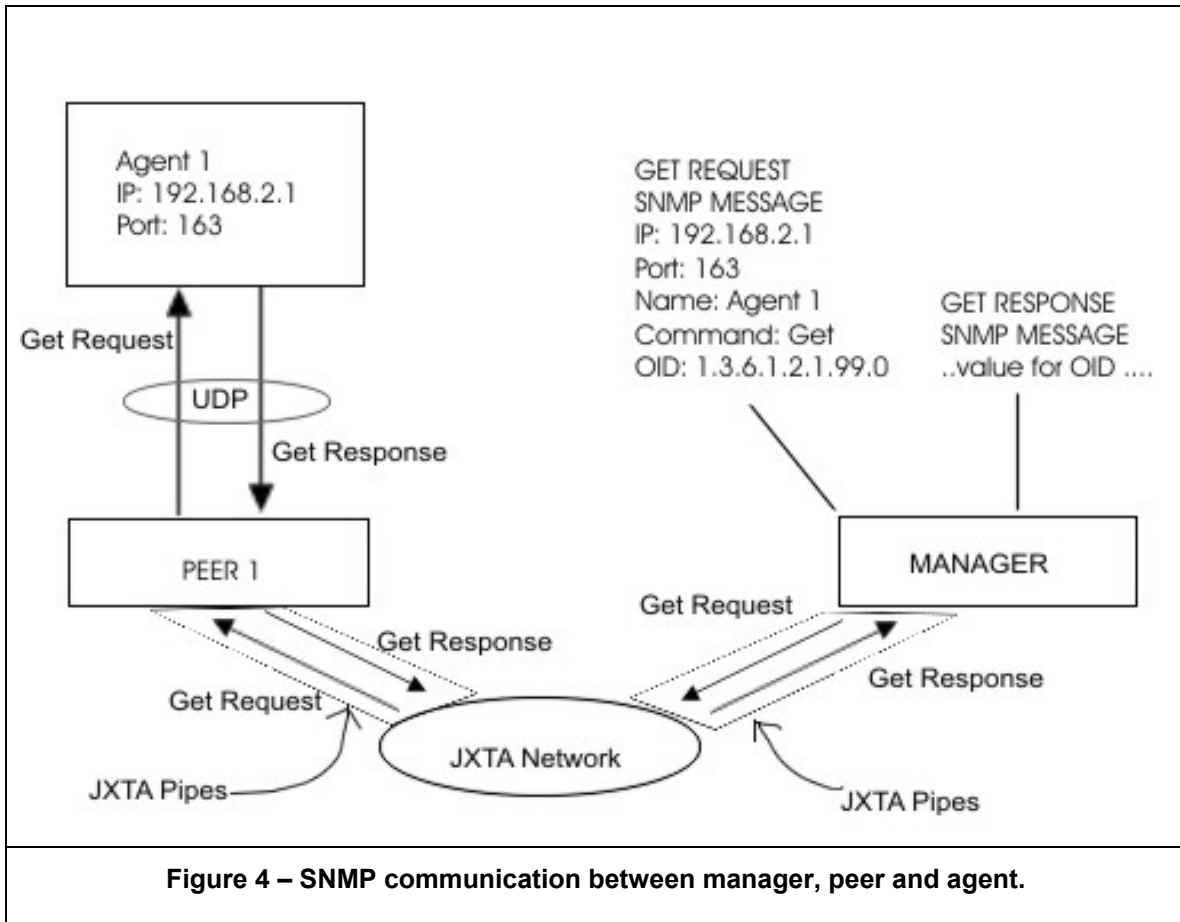
If the peer was not the one that became a manager, it will leave the management group and re-publish itself as a regular peer and continue the checking the network for a manager.

```
if(!peer.isManager()) {
    // this peer is not the manager so exit the group
    peer.leaveGroup(SNMPPeer.SNMPNetworkManagerGroupName);
    //peer.log("Not the manager so exiting group");
    peer.publishSNMPNetwork();
    peer.snmpManager = null;
}
```

#### **2.1.4 SNMP Communication**

The use of open source Java software was used to have SNMP Agents running on the clients to simulate actual devices. The major modifications to this software were to allow a client to run an agent from within the application. Another modification was adding the ability to select which port the agent is to be run on therefore allowing multiple agents to be running on one machine.

Included with this software was a small application that would send SNMP commands to SNMP agents. Part of this was included in the management package to allow managers to send SNMP messages to other peers on the network. Part of this code was also included on each peer to actually send the SNMP command obtained from a manager's SNMP message (a JXTA message over a pipe) to its running agent. Therefore, a manager will format an SNMP message and send it to a peer. The peer then receives this message and sends the appropriate SNMP command to the agent. Figure 4 visually shows how the SNMP messages are communicated.



As it stands, the application includes very little useful SNMP management commands. However, the capability is there to add more as the one's included are simply for demonstrating that sending an SNMP command is possible.

The source of the obtained SNMP software can be found in the references section.

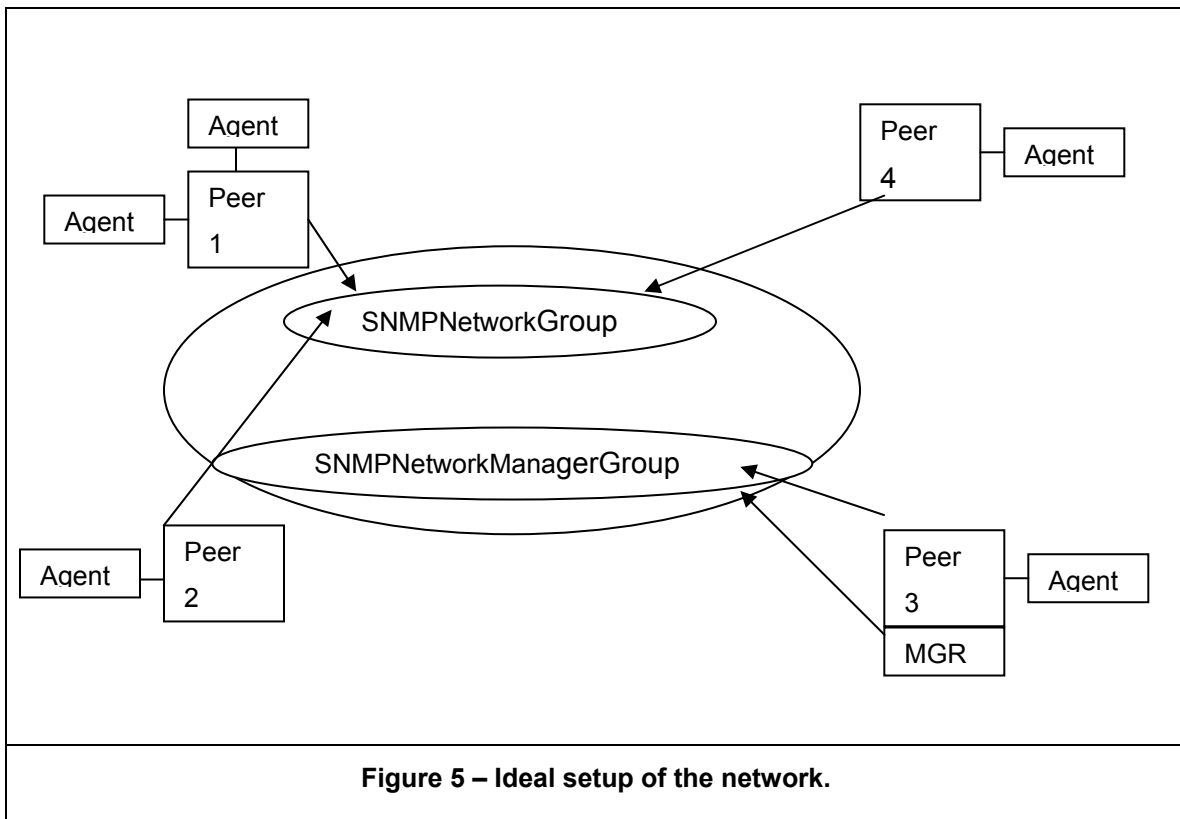
## 2.2 Use Cases

During the runtime of the application, there are several scenarios that may occur. The following outlines the various scenarios that may take place in the application and over the entire network.

### 2.2.1 Scenario: Many peers with agents and 1 manager

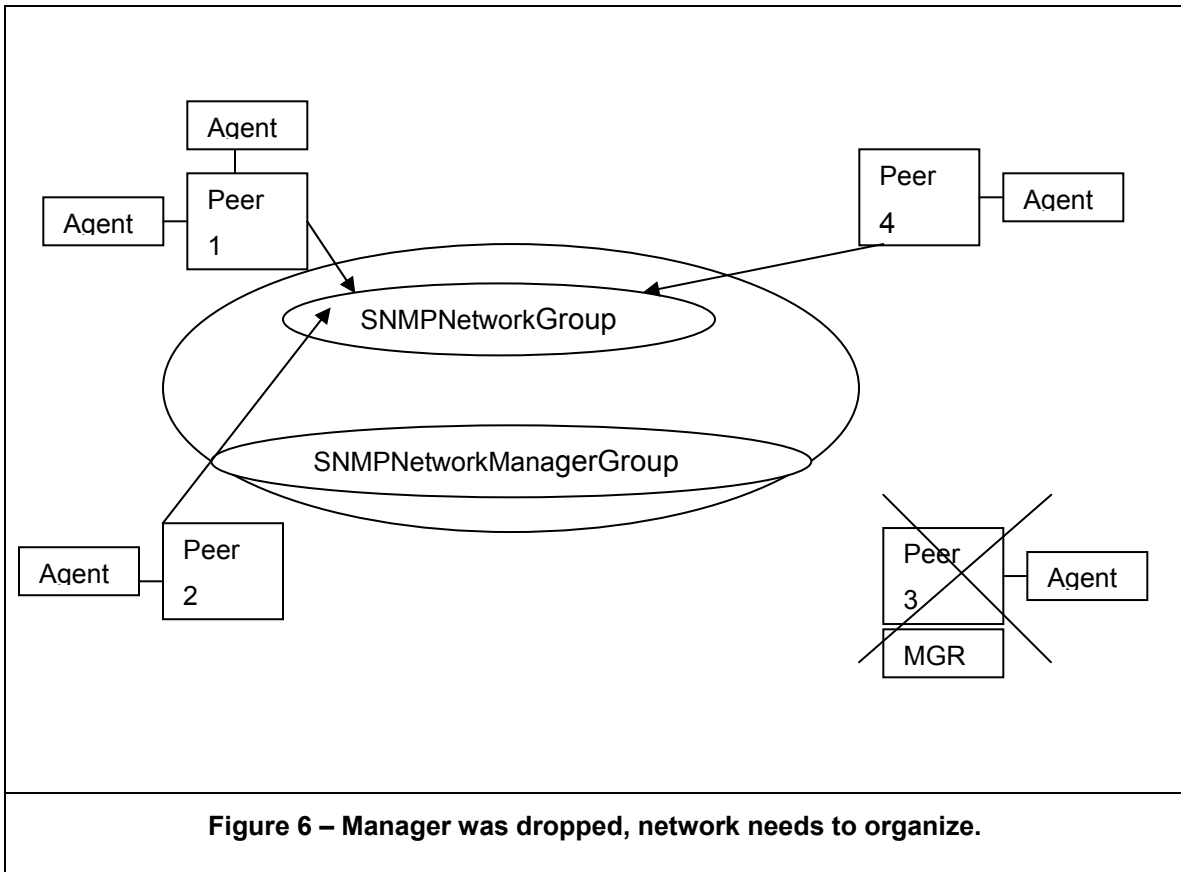
As shown in Figure 5, the network consists of four peers in the SNMPNetwork group, each with one or more agents. Peer 3 is currently the manager of the network

and is therefore the only peer part of the SNMPNetworkManager group. Peer 3 would also be the only peer that would have the management UI displayed.



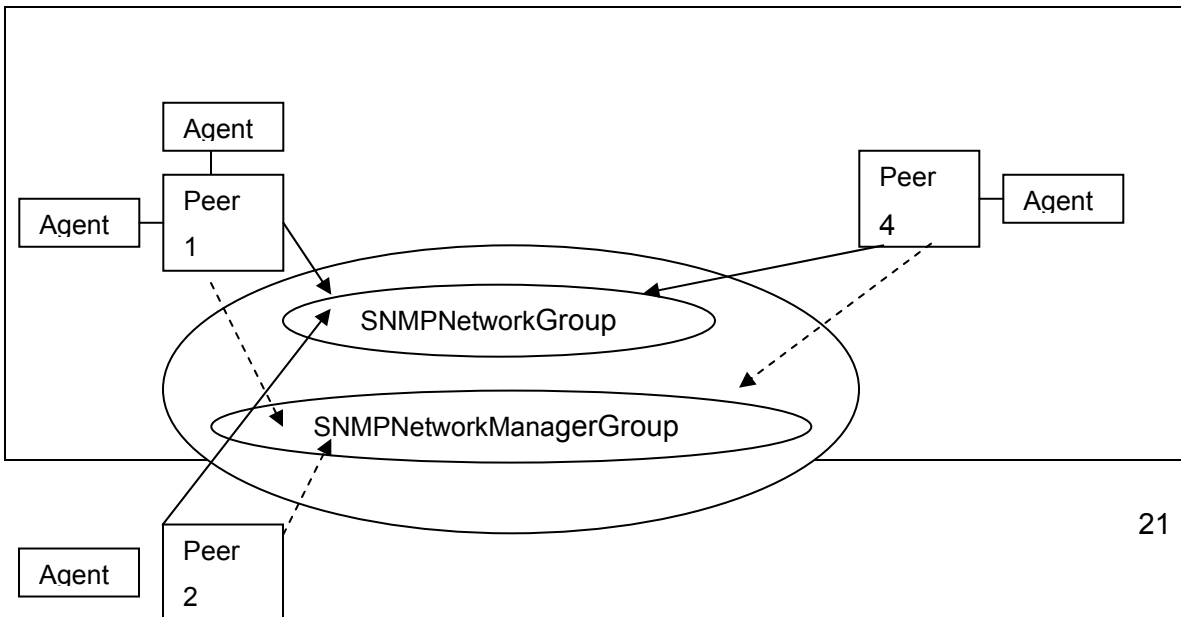
### 2.2.2 Scenario: Many peers with agents and no manager

If the managing peer (peer 3) was disconnected, then the network would be left in an unorganized state. Figure 6 shows how the network would look when it has no manager and therefore needs to be organized. Peer 3 is disconnected therefore that peer is no longer part of the network group nor the manager group.



### 2.2.3 Scenario: Peers attempting to organize

Once the peers realize that they have not received a ping response from any of the managers, they will begin their attempt to organize the network. By doing so they join the SNMPNetworkManager group. Figure 7 shows how the JXTA network would look at this stage.

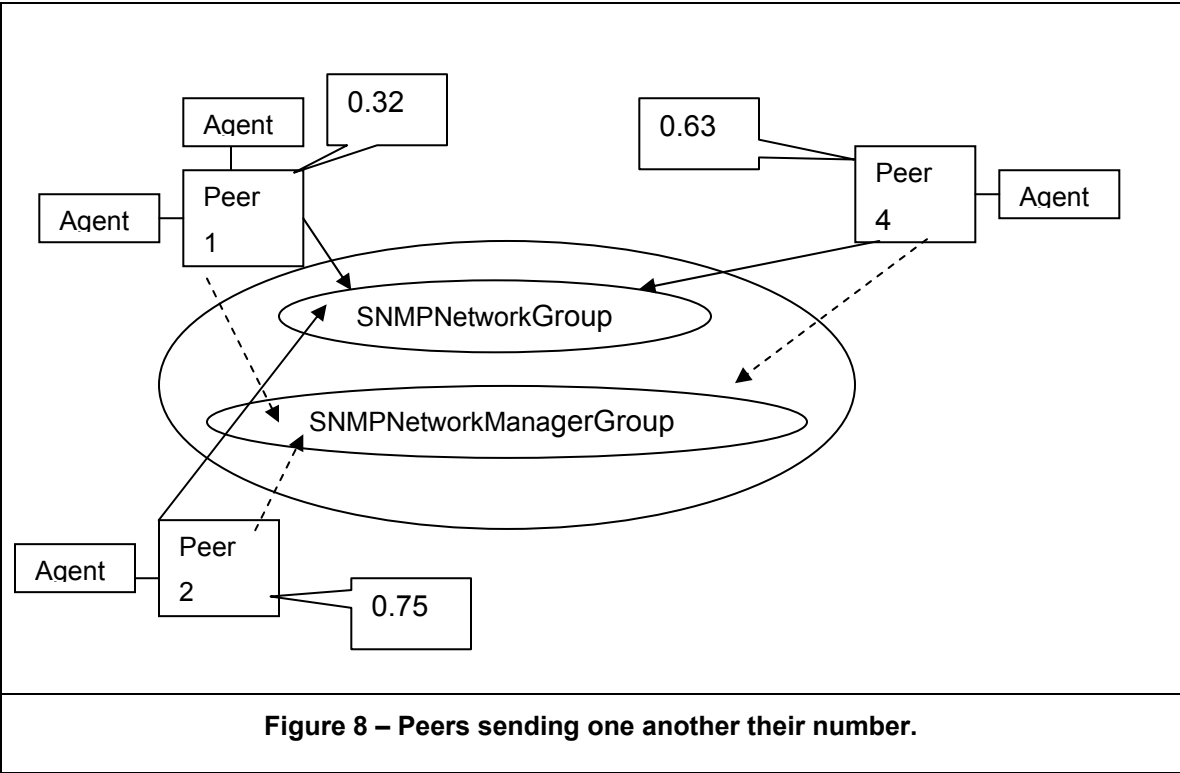




**2.2.4 Scenario: Peers determining who should manage**

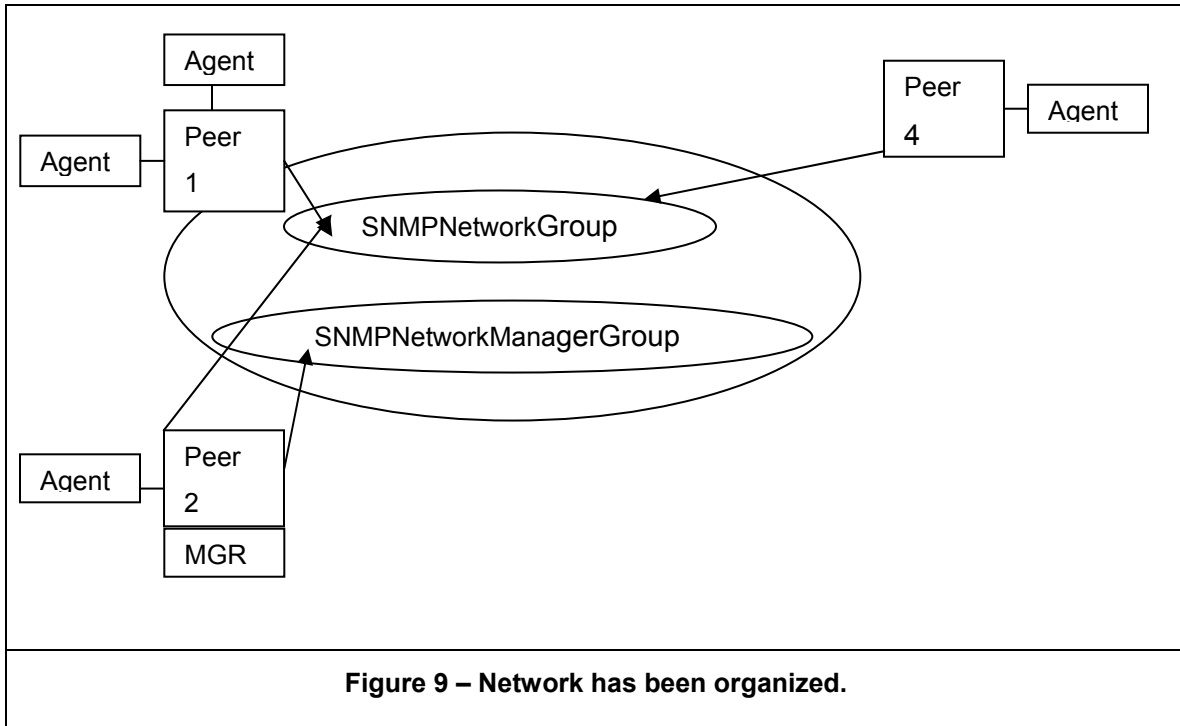
After joining the management group, the peers send each other their random generated number. The outcome of this will determine who the manager will be. Figure 8 shows how the network would look at this stage.

Peer 2 has the highest number therefore it will send the others a ‘leave’ request. The other peers will either leave on their own since their number is smaller or they will leave once they receive a leave request (whichever happens first). Peer 2 will then become the manager once it is the only peer left in the group.



**2.2.5 Scenario: Peers have organized themselves**

Once a manager has been determined the network is back to an organized state. Figure 9 shows what the network will look like. Peer 2 is now the manager and is the only peer in the SNMPNetworkManagerGroup.



## 2.3 Design Decisions

During the design stage of the application there were some decisions to be made as to how the implementation should function. The decisions are described here with their pros and cons, the problem, the solution and the justification to the decision.

<b>Problem</b>	
How the manager should communicate with the peers.	
<b>Alternatives</b>	
<b>Pros</b>	<b>Cons</b>
Broadcast messages to all the peers in the SNMPNetwork group.	
<ul style="list-style-type: none"> <li>• Easy to implement</li> <li>• Reduces number of pipes to maintain</li> </ul>	<ul style="list-style-type: none"> <li>• Creates a lot of traffic</li> <li>• Every peer will constantly be processing messages</li> </ul>

	<ul style="list-style-type: none"> <li>Peers will receive messages that are not for them</li> </ul>
Broadcast messages to all the peers in the SNMPNetwork group but have the message addressed to a specific peer.	
<ul style="list-style-type: none"> <li>Easy to implement</li> <li>Reduces number of pipes to maintain</li> <li>Only a message addressed to the peer will be processed</li> </ul>	<ul style="list-style-type: none"> <li>Creates a lot of traffic</li> <li>Peers will still receive the message that isn't for them but will not process it</li> </ul>
Have the manager maintain a collection of pipes to each peer.	
<ul style="list-style-type: none"> <li>Reduced traffic</li> <li>Peers only receive messages that they need to process</li> </ul>	<ul style="list-style-type: none"> <li>More difficult to implement</li> <li>A large collection of pipes</li> <li>Maintenance issues with non-existent peer pipes</li> <li>Manager will be required to have an output pipe to each peer at all times</li> </ul>
<b>Solution</b>	
It was decided that the manager will broadcast its messages but it each will be addressed to a specific peer.	
<b>Justification</b>	
It would appear that the best solution would be to send a message directly to the peer of interest rather than broadcasting it to everyone however this would in turn cause greater maintenance issues that would need to be solved and are beyond the scope of this project. Therefore, having messages broadcast to all peers and having the peer check to see if the message is addressed to it makes for an easier demonstration of the fault-tolerance on the management level.	

<b>Problem</b>	
How the management will be determined.	
<b>Alternatives</b>	
<b>Pros</b>	<b>Cons</b>
The peer with the highest priority based on arrival will be manager.	
<ul style="list-style-type: none"> <li>Based on fairness</li> </ul>	<ul style="list-style-type: none"> <li>Difficult to determine who was first on the network</li> <li>Difficult to assign priorities without already having a manager</li> </ul>
A peer will generate a random number and compare it with others.	
<ul style="list-style-type: none"> <li>Easy to implement</li> <li>Simple comparison</li> <li>Any peer can become a manager</li> <li>Works very well</li> </ul>	<ul style="list-style-type: none"> <li>Peers without the proper resources may become a manager</li> <li>Management is based on luck not computing power of the peer</li> </ul>
Current manager will designate possible managers in case of failure.	
<ul style="list-style-type: none"> <li>A peer can determine what other peer(s) are best suited for the task</li> </ul>	<ul style="list-style-type: none"> <li>Difficult to decide on initial manager</li> <li>May cause problems if a manager crashes before designating others</li> </ul>

<b>Solution</b>
It was decided that having a peer generate a number and compare it with others would make for the best design.
<b>Justification</b>
This design was the easiest to implement and introduced no obvious downfalls other than the one side-affect that peers that may not be best suited have the same chance of becoming a manager as a very powerful peer. Given that the intent of the project was not to find the best peer, but any peer to become manager, the design would suffice.

<b>Problem</b>	
How SNMP messages will be sent.	
<b>Alternatives</b>	
<b>Pros</b>	<b>Cons</b>
The manager will get the agent information from a peer and send it directly.	
<ul style="list-style-type: none"> <li>• Only managers send SNMP commands</li> </ul>	<ul style="list-style-type: none"> <li>• Difficult to obtain IP information from a peer</li> <li>• Subnet limitations</li> </ul>
The manager will format a message for the peer and the peer will send its agent the actual SNMP command.	
<ul style="list-style-type: none"> <li>• Has the ability to reach more than one network through rendez-vous peers</li> <li>• Management only needs to know the agent information not peer IP information</li> </ul>	<ul style="list-style-type: none"> <li>• None.</li> </ul>
<b>Solution</b>	
It was decided to have the manager send a message that the peer will transform into an SNMP command and send it to the agent itself.	
<b>Justification</b>	
This design made the most sense in terms of being able to reach all peer's agents. Also, using this design takes advantage of JXTA's encapsulation of peer IP addresses and ports.	

### 3. Problems Encountered

One of the issues encountered was with the initial setup of JXTA. Running multiple JXTA sessions on one computer was needed for testing communication between clients, however, since JXTA uses cached settings one needs to copy the client to another folder, go through the setup again and be sure to change each client to be running on a separate port. At first the guide that was used to try this setup did not mention changing the port therefore there were communication issues.

Another problem encountered was in the implementation of one of the designs to self-organize. The idea was to have a peer send a message to all the other peers that were part of a manager group, however, for a peer to message another peer they both must be part of the same group. This was unknown at the beginning of the design and encountered while trying to implement it. It is a limit (or feature) of the JXTA framework.

A third problem encountered was in trying to have the network self-organize. The issue was that if two or more peers were to notice an absence of a manager and therefore were going to take on the task at the same time. Which one would get it? To solve this problem the design of the organization algorithm had to include something that would be unique to each peer to select a peer based on that. As mentioned in the design decision, it was decided to use a random generated number.

Once the generated number was introduced into the self-organization algorithm, a possible issue arose where two or more peers, although not likely, could generate the same number. To correct this problem, the algorithm was re-designed to generate a new number every time it received another peer's that was the same. What prompted this issue was that in the initial design each peer would generate a number when it was started. This in fact was an issue of its own since a peer may generate a very low number and therefore would never become a manager. Only the peers with high numbers would become a manager regardless of their capabilities. To solved this the design was modified to have a peer generate a new number every time it had to go into a state where it was going to try and become a manager.

## 4. Conclusion

The goal of the project was to add fault-tolerance to the management layer of an SNMP network over a peer-to-peer network and this was successfully accomplished. With the use of a Java based peer-to-peer framework, JXTA, as well as open-source Java based SNMP agents, an application was developed to demonstrate the possibility. As the included application shows, a peer running agents over the JXTA framework is capable of self-organizing to ensure that the entire network is always managed.

Using the devised algorithm, if a managing peer were to fail, the remaining peers in the network would recognize this failure and begin to organize themselves such that another peer will take over management.

Due to both time and scope of the project limitations, the developed environment is by no means suitable for real-world use. There are many more enhancements that could be made in order to make the environment more useable. Such enhancements are:

- Allow multiple managers on the network.
- Enforce a minimum or maximum number of managers on the network. Therefore each peer would know of the allowable amount and would organize themselves to ensure that this is always met. Managers would have to keep track of the number currently managing and ensure that the limit is never crossed.
- Refactor the application to further separate the management into a downloadable add-on that a peer can obtain.
- If desired, allow regular peers to send SNMP commands that get routed through the managing peer. (ie: extending the SNMP UI to each peer.)

- Refactor how peers are kept track of and allow peers with the same name on the network.
- Devise a clean-up process in the manager add-on that will clear away any 'dead nodes' on the network.
- User interface re-design with additional features.

With these modifications made, the application will be much closer to an actual useable system.

## 5. Glossary

Abstract Syntax Notation 1 (ASN.1) - A language used for defining datatypes. ASN.1 is used in OSI standards and TCP/IP network management specifications.

Advertisement – A structured representation of an entity, service, or a resource made available by a peer.

Agent - A management entity consisting of hardware and embedded software which responds to SNMP requests over Ethernet from an SNMP manager.

Basic Encoding Rules (BER) - A set of rules for translating ASN.1 values into a stream of octets to be transmitted across a network.

JXTA – A defined set of protocols designed to address the common functionality required to allow peers on a network to form robust networks independent of the specifics of each.

Management Interface Base (MIB) - A logical database made up of the configuration, status and statistical information stored at a device.

Node - Any single computer connected to a network.

Pipe – A basic mechanism for peers to share information with each other. Pipes are responsible for providing network connectivity.

Peer Group – A set of peers formed to serve a common interest or goal.

Peer - Relationship between network devices that have mutual access to each other's resources.

Simple Network Management Protocol (SNMP) – the foundation protocol for the monitoring and potentially the management of network-connected devices.

Rendezvous peer – a peer that provides peers with a network location to use to discover other peers and peer resources.

User Datagram Protocol (UDP) - A connection less, communication transport method that offers a limited amount of service when messages are exchanged over the Internet Protocol.

## 6. References

Wilson, Brendon J. JXTA, New Riders Publishing, Indianapolis, 2002

Davie, Bruce S., Peterson, Larry L. Computer Networks – A Systems Approach Edition 3, Morgan Kaufmann Publishers, San Francisco, 2003

### Software References

Project JXTA

<<http://www.jxta.org>>

SNMP Agent Software

<[http://edge.mcs.drexel.edu/GICL/people/sevy/snmp/snmp\\_package.html](http://edge.mcs.drexel.edu/GICL/people/sevy/snmp/snmp_package.html)>

JAL – JXTA Abstraction Layer

<<http://ezel.jxta.org/jal.html>>

## 7. Appendices

### 7.1 Running the Application

The application has been tested using PC's running the Windows XP operating system with Java Runtime version 1.4.2\_03 installed. Follow the following steps to run the application.

1. Copy all the files from the CD to a folder on the PC(s) harddrive(s).
  - a. To run multiple instances of the application on the same PC, copy the files again onto the PC but in a different folder.
2. On one of the computers run the 'peer1.bat' file.
3. Enter the peer name of the file you are running.
  - a. ie: if running peer1.bat then enter 'peer1' in the name field.
4. Click on the 'advanced' button and uncheck the Enabled checkbox under HTTP Settings.
  - a. Change the port from 9701 to 97xx if running multiple instances on the same PC (fill in numbers other than 01 for xx).
5. You may run the agent\*.bat files as well.
  - a. Each of these start a 'remote' agent under a different port on the PC.
  - b. These can be run on separate PC's as well.
6. When the UI loads you can start more agents if you wish by clicking on the 'Start Agent' button and assigning a name and a port (other than the ports that were used in step 5).
7. If any agents were started in step 5 then for each click on 'Add Remote Agent' and enter the IP address of the machine that the agent is running on and the port.

8. Repeat steps 1-7 but run one of the other peerX.bat files instead of peer1.bat.
9. You can remove agents by selecting them on the right and clicking on 'Remove Agent'.
10. On the manager, when the management UI appears you will see a listing of the other peers on the network and their agents. You can click on one of the agents and then click on 'Send SNMP' to send an SNMP command to the agent.
  - a. A dialog will appear and you can select the type of command to send (currently only supports a GET Request) and the command OID.
11. As the manager you can also click on 'Stop Managing' to simulate a manager being dropped from the network and initiating the self-organization.
  - a. the same affect will take place by closing the client of the peer currently managing

## 7.2 Ideal network setup

To fully demonstrate the capability of the application it is best to run in a setup similar to the following.

### PC 1

- One peer client running
- Two remote agents running

### PC 2

- One peer client running
- Two built-in agents running

### PC 3

- Two peer clients running (separate folders and different ports)
- An agent running on each (on different ports)

All three machines will have to be networked. After the environment has organized and a manager has been set, experiment with sending SNMP commands to the various agents available. Afterwards, get the managing peer to 'Stop Managing' or close the application. Soon afterwards one of the peers on the network will take over management.