ARP and ARP Spoofing Simulation

Aaron Whitehead

CPE 400

11/23/2015

1 Introduction

What the general population generally understands to be the internet is actually just a series of communication layers and protocols that allow information to be sent and received between different devices across some distance. Layer two, known as the Data Link Layer is one of the more critical and well known layers. The Data Link Layer is the protocol layer that transfers data between different nodes. The ARP, or Address Resolution Protocol that happens in this layer to allow the data frames to go from a target protocol address to a target physical address. ARP takes care of mapping IP addresses to physical addresses, by handling its own type of ARP message, and using it to request data from the devices on the current network. However, several nefarious individuals have found that by telling the node they have a different protocol address than they actually do, they can cause data to be redirected to them instead of its intended target. This is known as ARP spoofing. The next general step is to then forward the data to the actual target, which is known as a Man in the Middle attack, as you are becoming a middle man in the data transfer chain. This project is around creating a simulation that shows how both of these events take place, including generating ARP messages and handling the messages, along with having a nefarious device poison the ARP cache table to allow them access to someone else's data. This was chosen as the project topic due to the fact that the ARP is something that is commonly observed and something that is important to understand.

2 Contribution / Object explanation

The ARP simulation has been broken up into four main parts, as will this description.

The first part is the main simulation driver. This can be seen as the overall program that creates and uses each other part. The job of the main driver is to create a ARP table, create a list of devices, to choose a device as a target, and attempt to send data to that device. If the ARP spoofing flag is set (see below in how to compile and run to learn more) then the main simulation driver will also mark one of the devices as a hacker and assign him a victim in the form of a different device. The overall structure of the driver is in the form of a while(true) loop that will continuously run until a stop command is provided by the user (ctrl+z), in that the driver will choose target user, pass that command to the ARP table, and then run a tick on the ARP table. It is during the tick command that the ARP spoofing happens, which leads to the second main portion of the project.

The second part of the simulation is the ARP table object. This table represents not only the cache table, but the overall ARP commands as well. This means that when the main simulation driver wants to have the ARP system create a request message to send to the devices, the driver will call a member function of the ARP table and it will handle the actions. The general use of the table is to check if a IP to MAC address mapping exists, add one if it does not, and to slowly age all entries in the list each tick to ensure nothing stale sits on the cache.

The third and most important part of the project is the ARP message object. The ARP message object is created exactly as the real object, including all of the needed data. The more important parts of the message are the opcode, which represents whether the message is a request or a reply, along with the target and source IP and MAC addresses. When the ARP table wishes to request a physical address from each of the devices,

it will create a request message, with its own information as the source, and the target IP address being the desired one, with the target hardware address being blank. Once the device that has the corresponding IP address is found, it creates a new message with its hardware address and ip address in the source locations, and the nodes information in the target position. The table will then parse this returning message and load the newly found physical address into the cache table. This is also how the ARP spoofing works, in that the malicious device will reply with its own physical address when its IP address does not match its own, but its victims, therefore causing the victims hardware address to be set to the attackers.

Fourth and finally, is the device object. This object is created solely to allow the simulation to run correctly, in that it is not in a general ARP system. The devices have the ability to take in ARP request messages, decide if it needs to respond, and then create a reply message when needed. The Device will also decide if now is a good time to reply without getting a request message, in order to put an undesired physical address onto the ARP cache table.

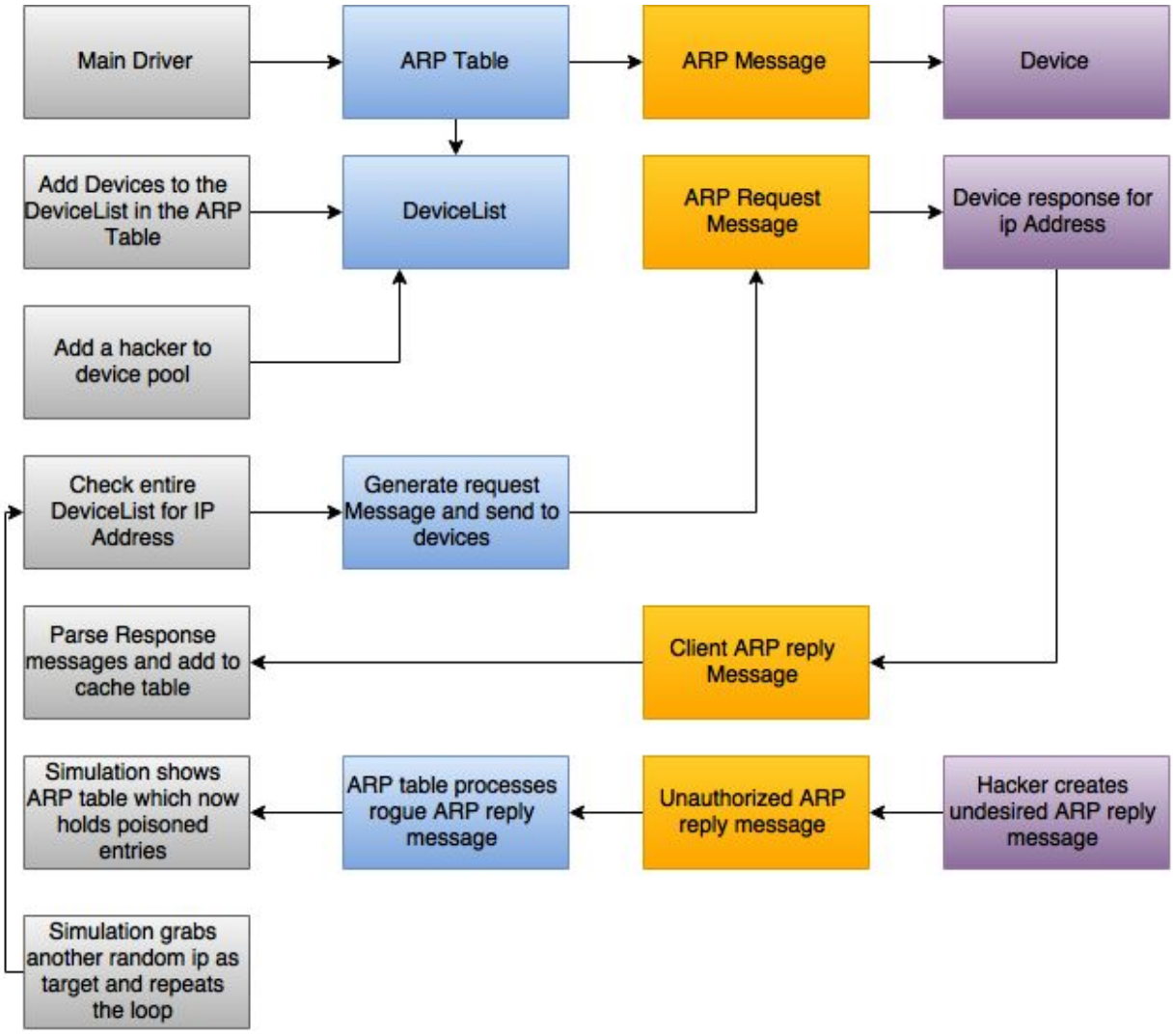
3 Code Explanation

3.1 How to Compile and Run

Instead of utilizing a makefile, this project simply uses a chain of included .h files with their own .cpp implementations alongside them in order to have everything compile together. The code has been written and tested on Ubuntu 64 bit, with the standard command to compile “g++ simulation.cpp -o simulation” and either the command “./simulation” or “./simulation hacking” to run the ARP simulator or the ARP spoofing simulator respectively. It is possible the program will run on other platforms and with slightly different compile commands, but

these are the conditions in which the program was tested and can be guaranteed to work under.

3.2 Code Interaction Diagram



Each column represents each of the main four parts of the system as defined in section 2. See section 4 for a detailed step by step explanation.

4 Run Example

Here is an example of a run of the ARP simulation with ARP spoofing turned on. To begin with, the ARP table is created, and the device list is populated with devices. After this is done, one of the objects is chosen at random as a hacker, and is given the ip address of one of the other devices as a victim.

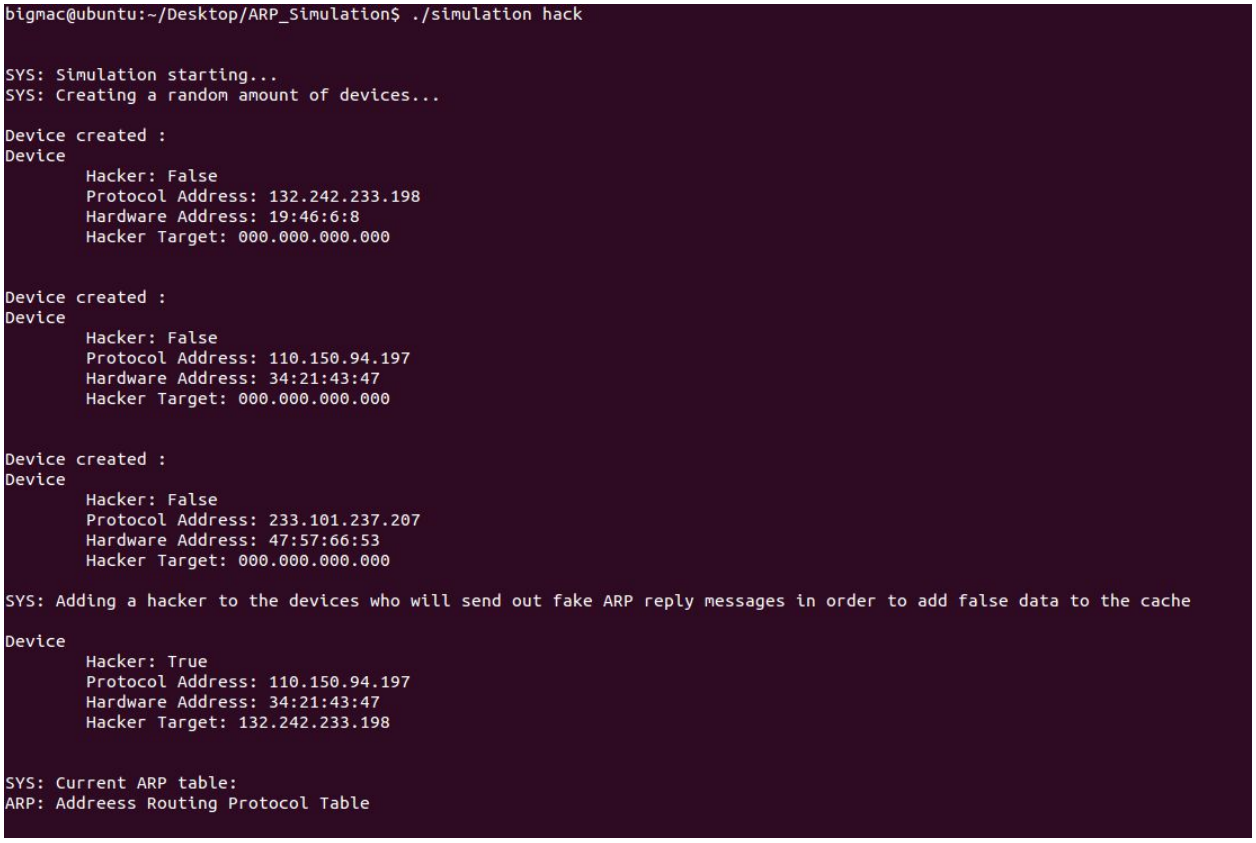


Figure 1: Three devices have been created, with the second one being assigned a hacker tag, along with the ip address of the first device as its "victim"

The next event that happens in the simulation is an ip address is chosen randomly from one of the devices, and the ARP table checks to see if it is in the table. If the address is not in the cache table, then a request message will be created to send to each device.

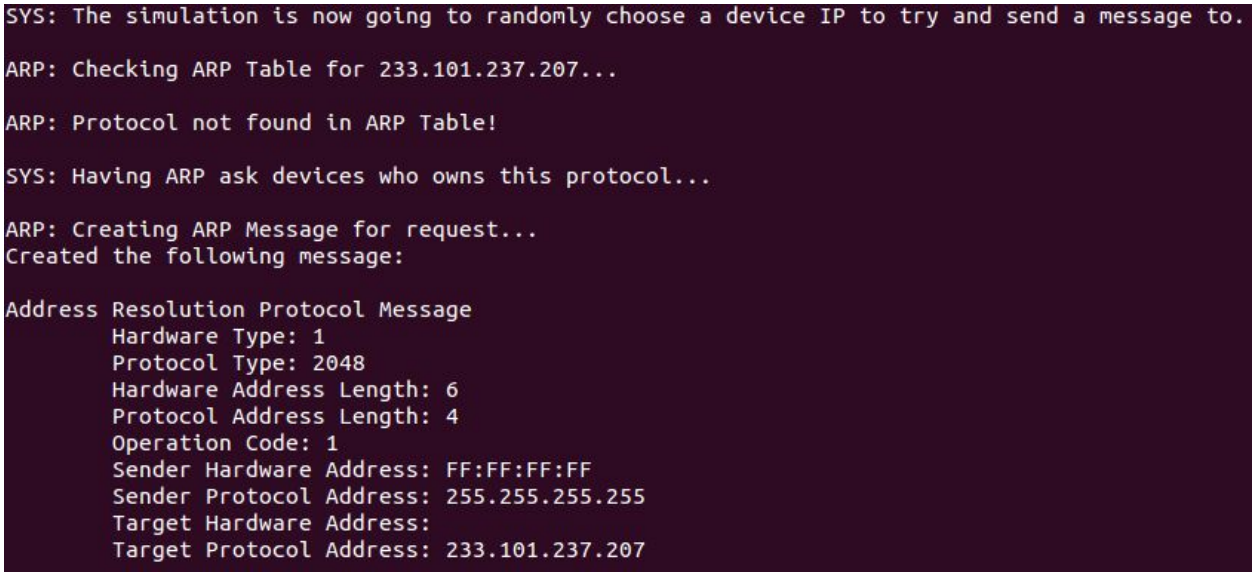


Figure 2: The IP address 233.101.237.207 has been randomly chosen, and was not found in the cache table. A request ARP message was created.

As can be seen in figure 2, the ARP message has all of the desired information, with the operation code signifying that the message is a request, and that the target hardware address field is currently blank. The address FF:FF:FF:FF and 255.255.255.255 are both representive of the node. After this is done, all of the devices are polled to see who has that target protocol address, and that device will create a reply message.

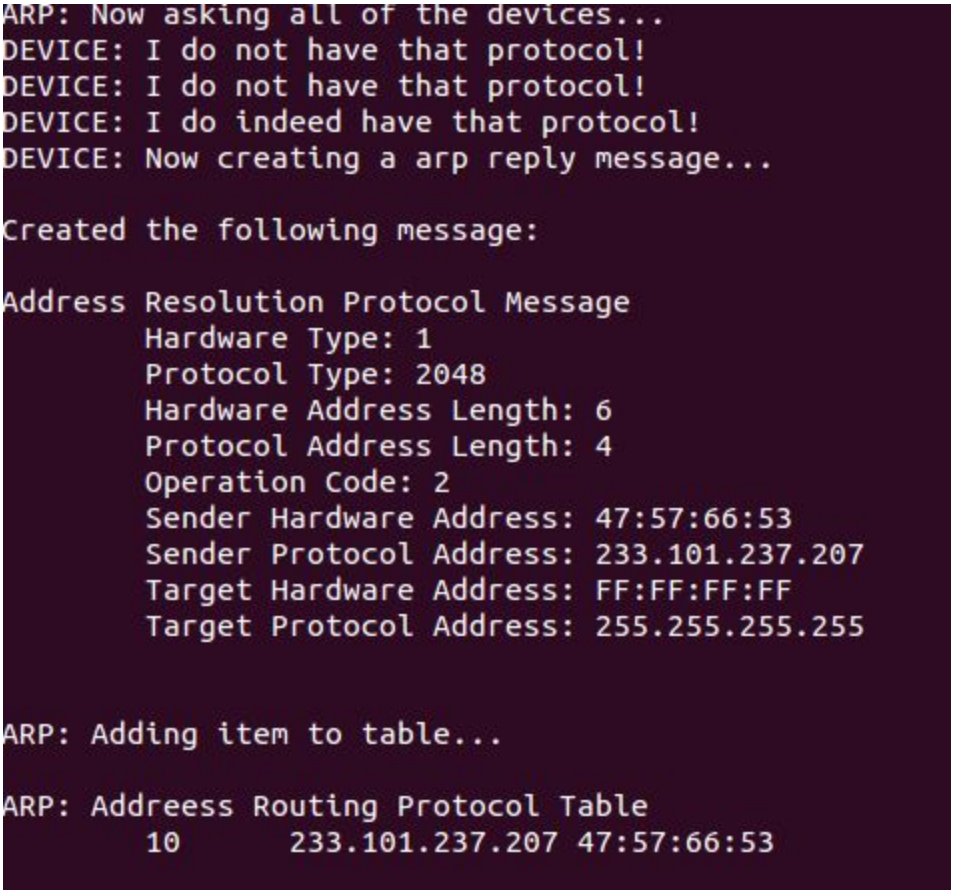


Figure 3: all of the devices are polled, and the correct device creates an ARP reply message, and the ARP table parses this reply and puts the data into the cache table.

As can be seen in figure 3, the information in the ARP reply is correct, notably the sender hardware address, which is what the ARP table is looking for, along with the opcode being set to 2 to signify that this is a reply message. The final step in this "loop" is to allow the nefarious device to send out a "reply" message, although no request message has been sent out. This will allow the hacking user to "poison" the ARP table cache by filling it with incorrect data.

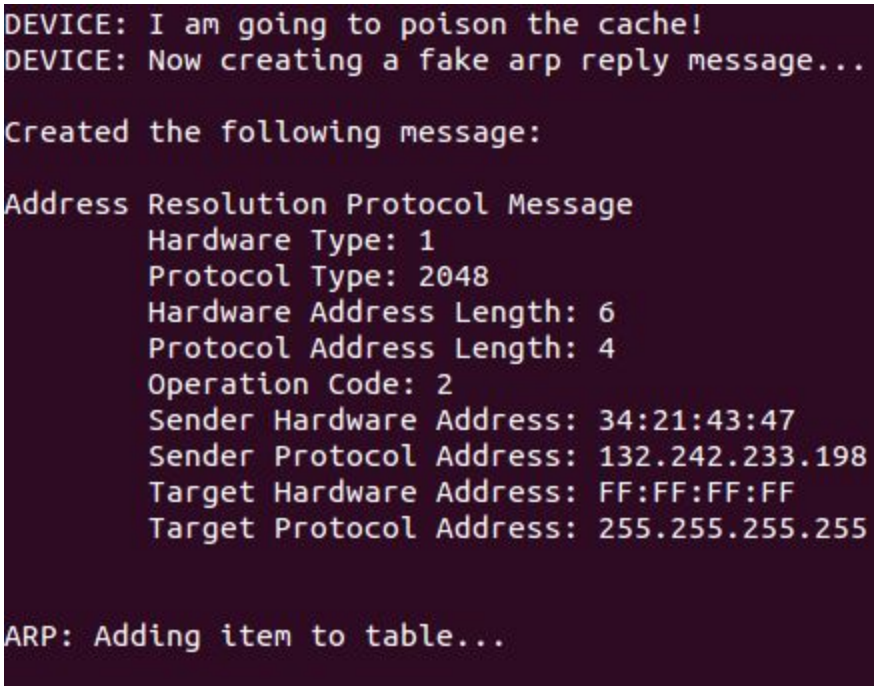


Figure 4: The hacker device has created a fake reply message and the ARP table was tricked into adding the item to the table.

In this particular simulation run, the address 132.242.233.198 is now pointing at physical address 34:21:43:47, even though by referencing figure 1 you can see that that ip address should actually point to a different hardware address. In the next tick of this simulation, it just so happens that someone is attempting to send data to 132.242.233.198. As far as the ARP table is concerned, this isn't a problem as it's already in the cache, but this is the data the poisoner has added, meaning that the data is now being forwarded to the wrong device.

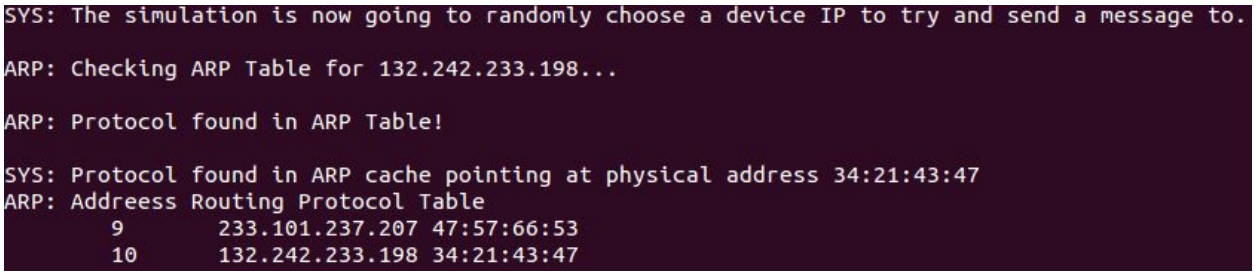


Figure 5: showing that on the second run of the loop the ARP table seems to think it already has the correct physical address

in the cache table, even though it was added illegally in order to spoof the ARP table.

On the next loop of this particular example simulation, it just so happens that the simulation decides to send data to the hacker, which means that the system creates a request ARP message for the hackers physical address, and the hacker kindly supplies it. This ends up with the interesting result that now the hackers ip and the victim's IP address are both in the cache, and both point to the same physical address. If at any point this happens in the cache table in this simulation it is known that a cache poisoning has happened.

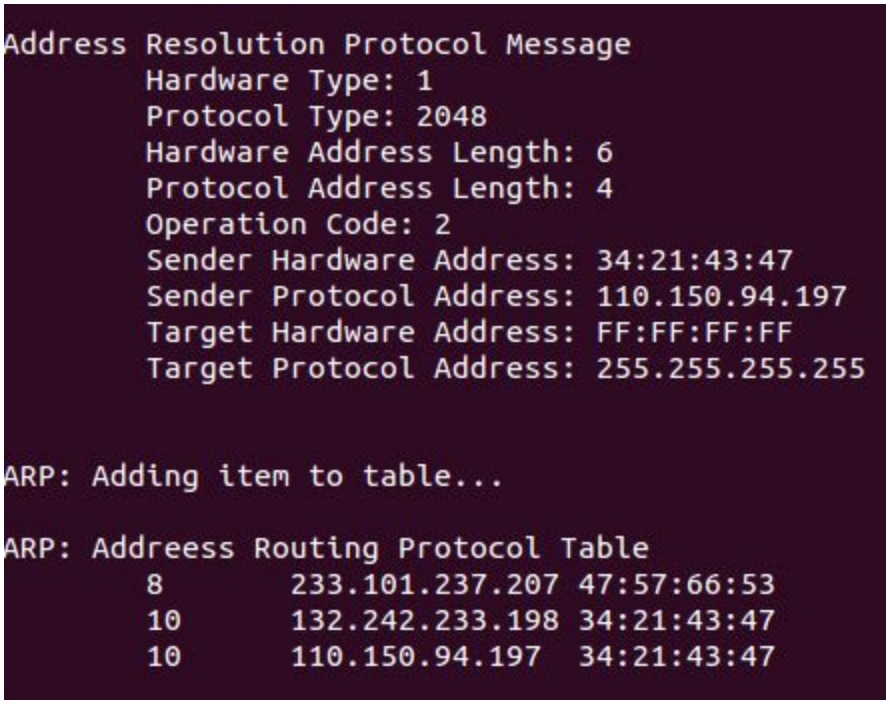


Figure 6: The reply ARP message from the hacker along with a printing of the ARP cache table after the reply message was received, showing both the hackers and the victim's IP address pointing to the same physical address.

5 Conclusion

The overall simulation was a success, with the simulation results being very descriptive and a great example of how ARP tables, messages, and even spoofing works. By utilizing different classes the simulation can and does represent from which section each of the main decisions come from, allowing the observer of the simulation to clearly see which part did what. Furthermore, the cache poisoning method is one that is still used in practice today, and shows exactly how an unauthorized ARP reply message can be used to poison an entire ARP cache table. If this project were going to be upgraded, there are several key points that would absolutely be added, the first of which being converting the arp cache spoofing to a full blown man in the middle attack, which would show the hacker forwarded data frames to the actual victim so the victim is unaware of any wrongdoing. The second priority would be to implement a safety system that would try and stop the ARP spoofing. The first and main method of detection would be to simply not accept any ARP message replies without first sending out an ARP request message, therefore not allowing the current system to work without a slight workaround. The second safety measure would be to scan the ARP table and if at any point there are two IP addresses pointing to the same hardware address, drop all entries with that as the hardware address from the table. However these would additions to the simulations, and not improvements overall, as the simulation still does what it sets out to do in a neat and concise manner.