Implementation of a Mobile Ad-hoc Mesh Network

Sudipto Das

Co-worker: Rajesh Roy

Department of Computer Science & Engineering
Jadavpur University
Kolkata – 32
{sudipto.das.rajesh.roy}@rediffmail.com

Under the Guidance of

Dr. Pradip K. Das

Professor
Department of Computer Science & Engineering
&
Coordinator
Centre for Mobile Computing & Communication
Jadavpur University
Kolkata – 32
We would like to take the opportunity to thank all those people who had directly or indirectly helped us through our project titled "Implementation of a Mobile Ad-hoc Mesh Network" during the academic year 2004-05.

To be precise, we would like to thank our esteemed professor Dr. P. K. Das for his wise lectures that enabled us to comprehend various aspects of the topic and examine our work thoroughly & with great patience and give valuable advices whenever required to upgrade our skills. We also thank him for providing us the opportunity and the necessary resources required to carry out the work.

And last, but by no means the least, we are proud to declare that we are fortunate enough to thank everybody associated with The Centre for Mobile Computing and Communication, Jadavpur University, for extending their co-operation during the project and without mention of whom our efforts would have been incomplete; again thanks to all concerned.
Contents

1. Ad-hoc Mesh Network – an overview
   1.1. Advantages
   1.2. Disadvantages
   1.3. Motivation and Practical applications

2. Implementation at Jadavpur University Campus
   2.1. Platform used
   2.2. Wireless Interface and its driver
      2.2.1. The Wireless Interface
      2.2.2. Driver for the Interface
      2.2.3. Optional Range Extension Antenna
   2.3. An Implementation of the AODV routing protocol
   2.4. Application program for monitoring
      2.4.1. The Server Application
      2.4.2. Client Application

3. Looking ahead: Possible enhancements

Appendix A:
Snippets of the driver Initialization Script

Appendix B:
Installation of the driver for the interface

Appendix C:
Installation of the AODV Kernel

Appendix D:
Configuring the gateway
1. Ad-hoc Mesh Network – an Introduction

Mobile Ad-hoc Network (MANET) is a distributed system consisting of many wireless mobile stations/ nodes with no predetermined topology or central control. Stations in a MANET communicate through wireless devices in self-organized manner. Such networks can be useful in situations where there is no other wireless infrastructure present, or where a wireless infrastructure cannot be used, like a battlefield or an emergency operation and even in robotics.

The ad-hoc mesh network architecture is based on mesh devices co-operating and acting as “forwarders” of their own traffic as well as traffic originating at some other mesh node, towards a “sink” node that provides external connectivity.

Each device has to manage and maintain known optimal paths which can change very often due to mobility, in order to route data-traffic to the particular node that provides access to the external IP backbone.

1.1 Advantages:

- Dynamic Route Discovery and Route Maintenance:
  The main advantage of using an ad-hoc network is that no initial information about the routes from one node to another has to be provided. The nodes can dynamically determine the appropriate route to forward the packet to the destination nodes.

- Adaptive Backbone provisioning:
  One of the best features of a wireless mesh is the lack of the requirement to provide a wired backhaul connection to every node. Rather user traffic is relayed using wireless radios between nodes, until it reaches its destination, or a node with a connection to another network (like the internet). Thus one could deploy e.g., a Wi-Fi Mesh to provide service over a large geographic area but only very limited backhaul initially.

- Fault-tolerance:
  Adaptive to failure in nodes or drop-outs in radio coverage – traffic is simply re-routed dynamically

- Bandwidth Scaling:
  Adding more nodes to a mesh increases the overall network capacity and total available band-width.

1.2 Disadvantages:

- Backhaul/user traffic competition at radio nodes using a single interface:
  If a Wi-Fi channel is used to implement a connection between two wireless mesh nodes, capacity of that link cannot be used (at that moment, anyway) for user traffic. To deal with licensed frequencies can be used for backhaul while leaving unlicensed bands for user traffic, or use 802.11a frequencies to provide backhaul for 802.11b/g traffic.

- Latency:
  If traffic is being relayed with a large number of intermediate nodes the latency involved in this relaying can affect time bounded traffic like Voice or video with stringent delay constants. This is because packets may suffer longer delays over the Multi-Hop wireless routes before they reach their final destination.

- Unwillingness to forward others’ packet:
  In the absence of incentives, intermediate nodes may be unwilling to forward packets for other nodes because they have to spend their resources for forwarding
these packets. In the case of non-co-operation from the neighboring nodes, the mesh network may collapse.

**Security:**

Finally, if user traffic is flowing through intermediate nodes in a mesh (as it most often will be) security is an issue. Intermediate nodes might be able to eavesdrop on data not intended for them.

### 1.3 Motivation and practical applications:

The past decade has seen steady exponential growth of the Internet as well as cellular telephony. A region-by-region study of this reveals that the growth rate has been phenomenal especially in the developing world. The cost of large-scale communication deployment is out of reach of developing countries. While cellular wireless technologies may help in quick deployment, like land-lines, their cost structure too is suited for the developed world. The service, and more importantly the equipments, is value-priced for markets where the users are willing to pay a high price. In rural settings, the density of population as well as its paying capacity is small and hence a deployment cannot pay for itself. The same is true of data based wireless access networks such as 802.16 Wireless MAN.

The 802.11 networks in this context suits best for deployment in the rural regions of the developing countries like India as setting up the individual nodes requires much less initial investment. In India the prospects of such a network is even brighter due to presence the vast stretches of rural areas between big cities, where, using a similar network, internet connectivity can be provided with ease.

## 2. Implementation at the Jadavpur University Campus

We, at the Jadavpur University Campus, have implemented a multi-hop Ad-hoc mesh network. The network consists of a network of individual Mobile Nodes (MN), each of which can route data independently on its own. These nodes can route the data generated by themselves, or can forward data from other nodes, thereby acting as Forwarding Nodes (FN). The network also consists of a node that has external connectivity, thereby acting as a Gateway Node (GN). In our implementation, we have chosen AODV as the routing protocol in the ad-hoc network.

Our implementation consists of a package that has the following important components:

- A workstation with Linux Fedora Core-I kernel (2.4.22.2115nptl) installed
- A Wireless PCI Network adapter (D-Link g520+) with an Open Source driver
- An Implementation on the AODV routing protocol
- A Client Server Application that monitors the performance of the other nodes in the network.

The entire package is installed by executing the shell script ./install.sh present in the root directory obtained after unpacking it, provided that all the necessary requirements for each of the individual packages are met. Each of the components of the above package is discussed briefly as under:
2.1 **Platform Used:**

Our Implementation requires an easily deployable (preferably single board computer) workstation that can support wireless interface(s) and has Linux with Fedora Core – I kernel (kernel version 2.4.22-1.2115nptl) installed. For the workstation to act as a node in the network, it should have processing power enough to process the various packets which it has to transmit.

For ease of deployment, the hardware platform for the mesh nodes need to have a small form factor, low power consumption, yet powerful enough to handle the protocols and wireless transmissions.

In our implementation, we have used workstations with Intel Pentium 4 processors (x86 architecture).

2.2 **Wireless Interface and its driver:**

Our Implementation uses the D-Link g520+ PCI Adapter as the network radio interface.

2.2.1 **The Wireless Interface:**

![D-Link g520+ PCI Adapter](image)

The main features of this interface card can be summarized as follows:

- Speeds up to 2X faster, with data transfer rates up to 22Mbps, when used with other D-Link AirPlus products
- Fully compatible with standard 802.11b-compliant devices
- Supports infrastructure networks via an Access Point and peer-to-peer communication in ad-hoc mode
- **Standards:**
  - IEEE 802.11
  - IEEE 802.11b
- **Range:**
  - Indoors – up to 328 feet (100 meters)
  - Outdoors – up to 1,312 feet (400 meters)
- **Network Architecture:**
  - **Supports Ad-Hoc Mode** (Peer-to-Peer without Access Point) or Infrastructure Mode (Communications to wired networks via Access Points with Roaming)
  - Compliant with IEEE 802.11b Standards

Ref: [http://www.dlink.co.in/dlink/Products/wireless/dwl520+.htm](http://www.dlink.co.in/dlink/Products/wireless/dwl520+.htm)

2.2.2 **Driver for the Interface:**

The open source driver for this card is included in our package. We have used the open source driver from a renowned Linux Open Source Project. For proper compilation and installation of the driver, the following are the necessary requirements:
- A Linux kernel higher that 2.4.18 with kernel source code
- Wireless extensions enabled
- Super User permission i.e., root permission
- No SMP configurations
- Proper kernel header files / kernel source (packages) installed for the exact kernel running.
- It is recommended that this kernel is new enough to support wireless extensions version WIRELESS_EXT > 13
- A make package has to be installed
- A gcc package has to be installed
- The /bin/bash shell is required for the start_net/stop_net scripts
- The module packages: module-init-tools for 2.5.x/2.6.x, modutils for Linux 2.4.x
- The package containing iwconfig, iwpriv etc., needs to be installed on the system

Once the driver is installed, the startup script ~/acx.../scripts/start_net should be modified to reflect the proper IP settings. To bring up the interface, the above script has to be executed.

2.2.3 **Optional Range Extension Antenna:**

The nodes may use an optional antenna for the purpose of extending the ranges to up to 1Km. In our implementation we envision to use the D-Link ANT24-0800 antenna the main features of which are:

- 360° Horizontal Spread
- Mounting Kit Supplied
- Extends Wireless Range up to 1 km
- Suitable for all IEEE 802.11b (D-Link Air and D-Link AirPlus) products with detachable antenna

Ref: [http://www.dlink.co.in/dlink/Products/wireless/ant240800.htm](http://www.dlink.co.in/dlink/Products/wireless/ant240800.htm)

**Outdoor Omni-Directional Antenna**

Ant24-0800

*Extend the coverage of a wireless local area network (WLAN)*

2.3 **An Implementation of the AODV routing protocol:**

In our implementation we have used AODV implementation, Kernel AODV by Luke Klein-Berndt, National Institute of Standards and Technology [http://w3.antd.nist.gov/wctg/aodv_kernel/](http://w3.antd.nist.gov/wctg/aodv_kernel/)

We have chosen AODV as the routing protocol due to the obvious advantages of AODV over the other routing protocols such as DSDV. This AODV implementation is a program that runs in the kernel space and uses the Linux NetFilter architecture for the purpose of routing packets.
Kernel AODV is a loadable kernel module for Linux. It implements AODV routing between computers equipped with WLAN devices.

The most important features of this implementation are:

- Since it is a Kernel module it runs in the Kernel space instead of the user space, which allows for better access to resources.
- Uses Netfilters from the 2.4 Kernel to capture packets going in and out of the node instead of using the libcap library
- Uses a Proc file to update the user about current routes and statistics for that node
- Supports multiple interfaces
- Based on draft 11 of AODV
- Runs on ARM based PDAs
- Completely redesigned and improved internal message handling system
- Managed internet gatewaying (even multi-hop)
- Monitor wireless signal strength
- Fixed buffer size for /proc Route Table
- Fixed Expanding Ring Search

The program maintains four files that the user can refer for various kinds of information, these files are listed below:

- AODV Routing table - /proc/aodv/routes
- List of timers - /proc/aodv/timers
- List of RREQs - /proc/aodv/flood_id
- List of neighbors - /proc/aodv/neigh

In our implementation there are two different types of nodes, Gateway Nodes (GN) and Client Nodes which can either be the Mobile Nodes or Forwarding Nodes. A Gateway node requires two interfaces, one which hooked up to the normal network and another which will connect to the AODV network. The Client is simply a regular AODV node which has been configured with the correct setting to communicate on the network which the Gateway node is attached to.

A Gateway node should be configured should that it can communication with the external network. This means it non-AODV interface should have a correctly configured IP address. The correctly route towards the subnet gateway of the external network should also be configured. It is also important to make sure that the subnets of the two interfaces do not overlap.

The Gateway node will generate a Route Reply to all Route Request which are not in the AODV subnet. The Gateway node considers the AODV subnet to be equivalent to the subnet settings for the interface being used by AODV. So if the interface is configured for 192.168.1.34/255.255.255.0, AODV would consider everything from 192.168.1.0 to 192.168.1.255 to be part of the AODV subnet. Of course all of the Client nodes have to be configured so that their IP address is in this subnet. The Gateway node assumes every address in the AODV subnet can be routed by AODV and all other addresses can be routed locally through the routing table.

IPTABLES is used to do NAT, allowing AODV nodes to access the external network, even if their IP address is not externally routable. IPTABLES should be installed on the Gateway node.

After a Client node’s network settings have been correctly configured it is started using ‘~/kernel_aodv2.2.2/start.sh’. Before a Gateway node can be started, its startup script ‘~/kernel_aodv2.2.2/start_gateway.sh’ must first be edited to reflect the current network configurations and then be executed.

2.4 Application program for monitoring:

The package includes a Client Server application the can be used to monitor the performance of the various other nodes in the network. The Server application must run on every node from which statistics have to be gathered, while the client can be used in a node from where the monitoring is done. It is a GUI based program developed in JAVA
and for the node to execute this application, must have the X – Server running and the Java Virtual Machine installed. A brief description of the application is given as under:

### 2.4.1 The Server Application:

The Server application, known as **StatisticsServer**, is much like a Daemon process that continually listens to a port assigned to this application (we have used Port No. 2048 which, as far as our knowledge, is not used by any other well-known application). On receiving a connection, the Server services it and delivers the client the proper information according to the request received. The requests have been classified into two major categories viz.

- **Request for Introductory Information:**
  
  On receiving the request for Introductory Information, the server retrieves some information (like Name of the node, its location etc.) about the node on which it is running and sends them back to the Client that had requested.

- **Request for Status Information:**
  
  On receiving the request for Status Information, the Server obtains the various network statistics (like the MAC Address of the interface of that node, the rate of data transmission, the ESSID name of the node etc.) of the node in which it is running and sends them to the requesting Client Node.

The connections established with the client are much like HTTP connections i.e., the connection is released as soon as the request has been serviced. The Server does not implement multi-threading, and so can service only one request at a time, but it can buffer about 100 requests so they can be serviced as soon as the resources are available. The Server displays various status messages to indicate the current status of its working.

The following figure shows a typical run of the server application.

#### 2.4.2 The Client Application:

The Client Application, known as **StatisticsClient**, is a completely user friendly GUI based application that gives a rough graphical representation of the present network scenario (by placing icons corresponding to the different types nodes) and can be used for the purpose of monitoring the other nodes in the network. The application graphically
displays only those nodes to which a valid path, from the node on which it is running, exist. But information can be gathered from any node in the network, irrespective of whether it is displayed on the Canvas or not. The display of the nodes on the canvas can be broadly divided into three categories viz. the **Local Host** i.e., the node on which the program is running, **Neighboring Nodes** i.e., the nodes to which a direct Single Hop route exists, and **Reachable Nodes** i.e., the nodes which are not neighbors, but to which a valid route exists.

On requesting information about a particular node, the Client application opens a connection with the **StatisticsServer** running on the destination node (assuming that the Sever is up and running on the destination. If the server is not running on the destination, the connection will be refused and an error will be reported in the Output Status are of the application), gathers information from that node and finally displays them. The following figure gives a typical sample run of the Client Application.

The textbox in the IP Pane can be used to obtain information from a node that is not shown in the list, i.e., from a node to whom path is not known. On clicking the “Go” button, the application connects to the server running on the destination node, and gathers both introductory as well as status information about the node.
Status Information can also be gathered from the nodes by clicking the right mouse button over the icon for the corresponding node and clicking on “Gather”. The application then gathers Status information from that node and displays it.

The application also displays the route table of the node on which it is running. To view any changes in the rapidly changing network scenario, the network scenario can be reloaded as well as the route table refreshed whenever the user desires to do so.

3. Looking Ahead: Possible Enhancements

Our implementation uses static configuration of IP addresses. In a truly dynamic real scenario, it would be ideal to dynamically obtain an IP from a central authority as in Dynamic Host Configuration Protocol. Hence using Distributed and Dynamic Address Configuration in the network would impart a truly dynamic nature to the network.

Furthermore, as the node uses the same wireless interface for both sending and receiving packets, there is contention at each node between the incoming and outgoing packets. This will result in longer delays and subsequent throughput degradation. This feature requires some exploration in future.
Appendix A

Snippets of the Driver Initialization Script

The following snippet of the ~/acx.../scripts/start_net script shows the locations that have to be modified by the user to reflect the current network settings of the node. The exact locations are marked and entries should be made or modified according to the instructions provided alongside the corresponding field.

```
ESSID="network_down" # THIS IS CASE Sensitive!! any == associate to any ESSID
# Default rate configured as 11Mbps to not cause speed problems (while
# using auto rate) or connection problems (while not using auto rate)
# with non-22Mbps hardware...
RATE=11M
AUTORATE=1 # only disable auto rate if you know what you're doing...
CHAN=1 # it's useful to try to stick to channels 1, 6 or 11 only, since these don't
overlap with other channels
#SHORTPREAMBLE=1 # set a value of 1 in order to force "Short Preamble"
(incompatible with very old WLAN hardware!) instead of peer autodetect
#TXPOWER=20 # 0..20 (dBm) (18dBm is firmware default) overly large setting might
perhaps destroy your radio eventually!
MODE=Ad-hoc # Managed for infrastructure, Ad-hoc for peer-to-peer, or Auto to
auto-select depending on environment
DEBUG=0xb # 0xffff for maximum debug info, 0 for none

# WEP Key(s)
# ascii keys (passphrase) should look like this: KEY="s:asciikey"
# hex keys should look like this: KEY="4378c2f43a"

# most wep users will want to use this line
KEY=""

# [ *** NOTE ***: WEP still doesn't work with acx111 cards yet! ]

# alternatively, you can uncomment and use these lines to
# set all 4 possible WEP keys
#KEY1="1234567890" #WEP64
#KEY2="1234567890"
#KEY3="1234567890"
#KEY4="1234567890"
# you must select which of the 4 keys above to use here:
#KEY="[1]" # for KEY1, "[2]" for KEY2, etc

ALG=open # open == Open System, restricted == Shared Key

#IP address
USE_DHCP=0 # set to 1 for auto configuration instead of fixed IP setting

IP=192.168.1.98 # set this if you did not set USE_DHCP=1
NETMASK=255.255.255.0 # set this if you did not set USE_DHCP=1
GATEWAY=192.168.1.254 # set this if you did not set USE_DHCP=1

LED_OFF=1 # set to 1 to turn off the power LED to save power

MTU_576=0 # set to 1 if you have buffer management problems
```
The above mentioned changes should be made before loading the driver module. After the changes are made, executing the script loads the driver.

Appendix B

Installing the Driver for the wireless Interface:

The Linux driver for the wireless interface is available in our package. The requirements for the driver are given in the section that introduces the driver. The following is the step by step procedure to install the driver:

Detecting the interface:

One must first determine that a device that contains a chipset that is compatible with the acx100/111 driver is plugged in. The command "lspci -n" gives a listing of the PCI devices. If any one of the entries contains the following at the end, then a compatible device has been plugged in. The possible values are:

104c:8400 (acx100 CardBus)
104c:8401 (acx100 PCI)
104c:9066 (acx111 Cardbus/PCI)

Finding and removing older drivers:

Type the command: find /lib/modules/`uname -r` -name "*acx*". If the command returns without any output, then there is no previous version of the driver installed, proceed to the next section of driver installation. If the command returns anything like acx100_pci or acx_pci, then an older version of the driver is installed which has to be removed.

The older version of the driver is removed by the following command:
find /lib/modules/`uname -r` -name "*acx*" -exec mv {} /root \; to move the older driver to your /root directory. Then, the command: "depmod -a" to update the module dependencies, it may take a while to complete.

Verifying the build environment:

The following step by step procedure can be used to verify the build environment:

Presence of "gcc with proper version":
The command "gcc --version" provides the command gcc's version. It should be above 3.2.3. An unsuccessful return implies the absence of the command. The package corresponding to the command should be installed first.

Presence of "make" with proper version:
The command "make --version" provides make's version. It should be above 3.80. An unsuccessful return implies the absence of the command. The package corresponding to the command should be installed first.

Presence of kernel source code:
To verify the presence of the correct version of the kernel source, the command: "ls /lib/modules/`uname -r`/build/include/linux/version.h" is used. A successful listing ensures the presence of kernel source code. An unsuccessful return implies the absence of kernel source. The package corresponding to the kernel source should be installed.
Presence of wireless tools:
To verify the presence of wireless tools, the command “iwconfig” is typed. The successful output of the command is like:

```
lo    no wireless extensions.
```

There may be other lines in addition to “lo” describing any other active ethernet devices in the machine that do not have wireless capability.

Checking for proper configurations:
Once the presence of the kernel-source package has been verified, make sure that a file named .config (that's a leading dot) exists in /lib/modules/`uname -r’/build directory (those are backtics, not single-quotes). Verify that it exists with ls -l /lib/modules/`uname -r’/build/.config.
An unsuccessful listing indicates the absence of the file in proper location. Copy the appropriate config file from the folder /lib/modules/`uname -r’/build/configs/ to the appropriate location with the name .config. By appropriate, we mean that the file corresponding to the processor architecture of the computer on which the driver is being installed. If it is an Intel x86 processor, then the file containing ‘i386’ or ‘i686’ can be used.
The .config file has to be checked for the proper settings. Search for the string “CONFIG_NET_RADIO=y”. If the string is not present in the file, the driver installation cannot proceed further. After that, for the string “CONFIG_NET_WIRELESS=y”. If the string is not present, just insert the line below the entry for “CONFIG_NET_RADIO=y”.

Unpacking and compiling the driver kernel:
Just copy the driver file “acx100-0.2.0pre8_....tar.gz” to the /root directory, unzip and untar the file. Then move to the new folder created. Execute the “make” command and oddly enough, successful output has these as it’s last 2 lines:

```
*** Compilation finished. Make sure to copy required firmware files to /usr/share/acx/ before proceeding! ***
make[1]: Leaving directory `/root/acx100-0.2.0pre8_plus_fixes_51/src'
```

Installing the Firmware and the Driver:
Copy the firmware file TIACX111.BIN (for the PCI card) or WLANGEN.BIN (for the PCMCIA card) available with the package to the folder /usr/share/acx. If the folder is not present (which is the most likely case) then create it.
After the firmware has been placed in the appropriate location, execute “make install”. This will copy the files just compiled to a location under /lib/modules/`uname -r’ and then run depmod to update the module dependencies. It will not do any configuration.

After all this hard work, the driver is hopefully ready to be used. The startup script `~/acx.../scripts/start_net must be modified, as hinted in Appendix A, to reflect the current network settings.

Appendix C
Installing the AODV Kernel:
The AODV Kernel is available in our package. The requirements for the kernel are given in the section that introduces the kernel. The following is the step by step procedure to compile the kernel:

绯 Red Checking Proper Configurations: 

For proper compilation of the kernel module, an entry "FORWARD_IPV4=yes" should be made in the file "/etc/sysconfig/network".

绯 Red Unpacking and compiling the kernel:

Just copy the driver file “kernel_aodv... .tar.gz” to the /root directory, unzip and untar the file. Then move to the new folder created. Before the make command is executed, the Makefile should be modified to reflect the proper processor architecture. The Makefile should contain a label target:. Modify it to the appropriate value according to the processor architecture of the host computer as guided by the hists provided alongside. e.g. for an Intel x86 processor, the line should read as target:=x86. Execute the “make” command.

The AODV kernel can be started by executing the script ~/kernel_aodv.../start.sh.

Appendix D

Configuring the Gateway:

The Gateway Node (GN) or the Border Node is one that has external connectivity to the nodes of the AODV subnet. For an AODV node to act as a GN, it must have at least two network interfaces, one for the AODV subnet, and another that is connected to an external network.

We assume that wlan0 is the interface to the internal network while eth0 (or ppp0) is the interface for the external network. The following commands are used to configure the gateway.

绯 Red Linux Connected via PPP:

This is used for a Linux computer connected to the internet by a Dial-up Modem (PPP). We assume that IPTABLES is up and running. Execute the following commands in the specified sequence.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iptables --flush</td>
<td>Flush all the rules in filter and NAT tables</td>
</tr>
<tr>
<td>iptables --table nat --flush</td>
<td>Delete all chains that are not in default filter and NAT table</td>
</tr>
<tr>
<td>iptables --delete-chain</td>
<td></td>
</tr>
<tr>
<td>iptables --table nat --delete-chain</td>
<td></td>
</tr>
<tr>
<td># Set up IP FORWARDing and Masquerading</td>
<td></td>
</tr>
<tr>
<td>iptables --table nat --append POSTROUTING</td>
<td>OUT INTERFACE ppp0 -j MASQUERADE</td>
</tr>
<tr>
<td>iptables --append FORWARD --in-interface</td>
<td>Assuming one NIC to local WLAN</td>
</tr>
<tr>
<td>echo 1 &gt; /proc/sys/net/ipv4/ip_forward</td>
<td>Enables packet forwarding by kernel</td>
</tr>
</tbody>
</table>
Linux Connected via DSL, Cable, T1:

This is used for a Linux computer connected to the internet using an Ethernet card. We assume that IPTABLES is up and running. Execute the following commands in the specified sequence.

```
# Delete and flush. Default table is "filter". Others like "nat" must be explicitly stated.
iptables --flush   # Flush all the rules in filter and nat tables
iptables --table nat --flush
iptables --delete-chain   # Delete all chains that are not in default filter and nat table
iptables --table nat --delete-chain

# Set up IP FORWARDing and Masquerading
iptables --table nat --append POSTROUTING --out-interface eth0 -j MASQUERADE
iptables --append FORWARD --in-interface wlan0 -j ACCEPT

echo 1 > /proc/sys/net/ipv4/ip_forward   # Enables packet forwarding by kernel
```