Linux in TCP/IP Internets

Get Connected!

Linux is much more than just a multi-user/multitasking operating system; it is the product of a community with a variety of interests and wishes. It thrives on communication, on the exchange of know-how and technologies. And communication across networks is a central component of this exchange.

BY IAN TRAVIS

Communication on networks, and above all on the Internet has become a fundamental part of our lives. It is hard to imagine life without networks, or can you imagine going back to the "bad old days" of the early 80s, when people used to exchange floppies?

But have you ever thought about how it all works? And I don't mean launching KMail or Evolution to send messages out into the world. What I mean is, have you ever stopped to consider how the bits and bytes that make up data communications actually cross those thousands of miles to arrive safely at their destination? You may be interested in using the same technologies to set up a home or small office network. This article provides the basic networking knowledge you will need to understand your SOHO network, avoiding trial and error, and explaining why a certain configuration works.

From Babylon to TCP/IP

When we put two computers on a network, we typically want to achieve an exchange of information. This task appears almost trivial from our modern point-of-view. System interoperability can be taken for granted. Choosing components has become more or less a question of taste, or budget. The functionality we now take for granted has its roots in the design laid down by internetworking pioneers many years ago. Their search for a way to merge a variety of widely different technologies to create a co-ordinate whole, led to the concept of the *internetwork*, an open system that allowed computers with various architectures to talk to each other. Standards that specify how networked computers communicate, and conventions that specify how networks are interconnected, and how information travels across these networks, are a major part of this concept. We refer to these standards and conventions as the TCP/IP Internet Protocol Suite, or TCP/IP for short

TCP/IP a Bird's Eye View

TCP/IP was named after the most important protocols in the suite: *Transmission Control Protocol/Internet Protocol*. To allow network communications to be viewed separately from the underlying hardware, the developers of TCP/IP subdivided the individual communication tasks into separate units, known as layers:

- **Physical**: Device drivers and network cards
- **Network**: IP, ICMP, IGMP
- **Transport**: TCP, UDP
- **Application**: Telnet, FTP, HTTP, etc.

The components within the individual units are interchangeable. They fulfill the same basic task, but in different ways. One of these communication tasks involves handling the physical aspects of the network, the network cards and transport media. Ethernet cards are the best-known example of this. When you change your network card, you simply load a new driver for the card, but you do not need to change the Network Layer or the application.

Another task on the physical layer is to format the data to allow it to be transferred across the wire. The signals that occur must reflect the physical structure of the transport medium.

While the Physical Layer simply sends data from one end of the wire to the other, the Network Layer ensures that the data will actually reach the right destination at the other end. The Transport Layer selects the appropriate service on the sending computer and – in the case of TCP – ensures that the data arrive at their destination safely and in the right order. If a packet does happen to get lost, TCP will re-transmit the packet.

Encapsulation

When an application uses TCP/IP to transmit information, the information progresses through the various layers of the protocol stack, until it is finally sent across the wire as a bitstream by the network card. Each layer adds control information. Imagine your data as the smallest doll in a set of Russian dolls. At each level, a so-called header is attached to your data. An application that writes data encapsulates the data in an application header, before passing the data on to TCP, for example. TCP adds its own TCP header, thus creating a TCP segment, which it passes on to IP. And IP – as you will have guessed – adds another header to create a so-called IP datagram. In an Ethernet network, an Ethernet header and tail are added to the IP datagram to create the physical frame.

Addresses

When we talk about addresses in a TCP/IP internet, it makes sense to define which addresses we actually mean. At the lowest level, the hardware interface has a unique physical address, for exam-

ple, a 48-bit Ethernet address, with a format like 00:E0:29:27:B1:75. This address is assigned by the manufacturer and globally unique.

The Ethernet address is needed for low-level communication. An Ethernet wire can house many computers. A network card will inspect the Ethernet address to decide whether an incoming packet is for its host. As we are looking at access to the wire (the Ether), this address is referred to as the Media Access Control or MAC address.

But MAC addresses would not allow us to create large-scale networks. Although the hardware manufacturer assigns the MAC address, there is no way of knowing where the card will be used, and thus routing in a network of this kind would be impossible. Imagine a situation where each house has a random number, but cities and streets have no names. And if we apply this to MAC addresses, the house numbers would not even be in sequence.

IP to the Rescue

The IP address organizes this seemingly chaotic landscape. Each computer is assigned an IP address as its Internet number, which is valid in the Network Layer. The IP address is structured. The first part of the address contains the number of the network where the computer resides, and the second part refers to the computer itself.

If a computer wants to talk to another, it means sending information across the Ether. As the network adapter only recognizes MAC addresses, there must be some way of discovering the unknown MAC address by reference to the known IP address. This is what the Address Resolution Protocol (ARP) does.

If the sender does not know the target's ARP address, it simply sends an ARP request to all the computers to find out where the target is. If the target hears this request, it responds with its own address and the computers involved remember the MAC/IP address mappings for a while (typically for about 10 minutes). You can use the *ifconfig* command to output your own network adapter's physical address:

```
ifconfig eth0
eth0 Link encap:Ethernet Hwaddr
 00:E0:29:27:B1:75
inet addr:192.168.41.207 Bcast:
192.168.41.255 Mask:
255.255.255.0
```

or type *arp -a* to view the entries in your computer's ARP cache:

```
arp -a
samba (192.168.41.3) at 00:40:
05:36:FD:4E [ether] on eth0
sqlse (192.168.41.4) at 00:04:
75:C2:7A:54 [ether] on eth0
work1 (192.168.41.24) at 00:04:
75:93:14:F8 [ether] on eth0
tech-2 (192.168.41.132) at 00:
00:E8:22:50:F8 [ether] on eth0
```

Routing

Asking every other computer on the network the way to your target only makes sense within a local network. Imagine having to ask every single computer on the Internet – public networks would be so busy handling ARP requests and responses that nothing else could possibly be transported. The answer to this dilemma is routing. Routing means choosing the best path for a data packet through a given network. Unsurprisingly, the computers that make these decisions are called routers.

From the viewpoint of the local network, the router is the gateway to the world outside. Each host must know the gateway, or default route. And this means configuring your computer to know the IP address of its default router.

If the target computer does not live on the same network as the sending computer, the sender will simply contact the router, and rely on the router to discover the rest of the path to the target and send the packets to that destination. This is where the two parts of the IP address come in useful. The router needs the network part to find the target network, and the host part designates the target within the target network.

If the source and target network are identical, this means that the target computer is on the sender's local network. In this case, the sender uses ARP to discover the MAC address and sends the data directly to this address. The subnet mask is used to find out which part of the address belongs to the network and which part to the host.

Putting on a Mask

Although there are many ways of writing IP addresses, the dotted decimal format is the most common. The 32 bits of the IP address are divided up into 4 groups

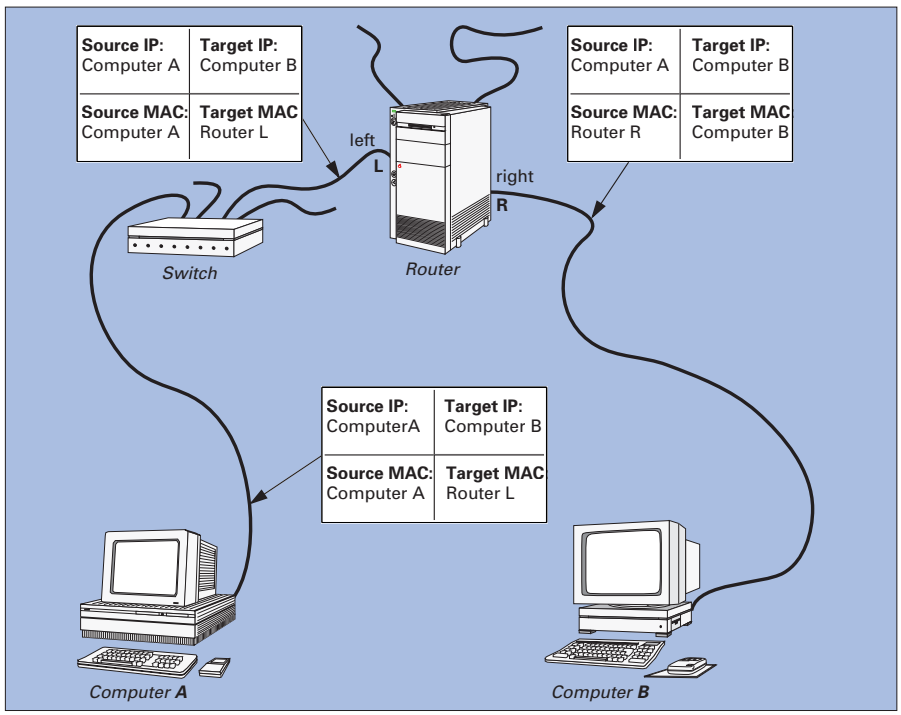

Figure 1: Computer A sends a packet to Computer B, using B's IP address, and the MAC address of the left port on the router. The router forwards the packet keeping the IP addresses but changing the MAC addresses. The switch has no influence on the packet

of 8 bits. Each group is separated from the next by a dot. The numbers in each octet are written in decimal format. A typical address could appear as 192.168.41.207.

The network (or subnet) mask follows the same pattern. A bit with a value (1) means that this part of the mask is part of the network address, a zero indicates that it belongs to the host. When all 8 bits in an octet are set to one, the decimal value is 255. Let's look at a typical mask: 255.255.255.0.

The first three bytes are all 255. In other words, the first three bytes of the IP address make up the network address. The last byte is zero, so the last byte in the IP address contains the host address. Let's take another look at the output from *ifconfig* to apply this to our example:

```
ifconfig eth0
eth0 Link encap:Ethernet Hwaddr➥
 00:E0:29:27:B1:75
inet addr:192.168.41.207 Bcast:➥
192.168.41.255 Mask:➥
255.255.255.0
```

This time we are interested in line starting with *inet addr:192.168.41.207*. As the mask for this address is 255.255.255.0, we can see that the computer resides on the network 192.168.41.0 and has the host ID 207 on that network.

The traditional IP addressing scheme divided IP addresses into different classes. At the time, the experts assumed that a handful of extremely large enterprises would need a lot of Internet addresses, a fair number of medium-sized enterprises would need a fair number of addresses, and that a large number of smaller enterprises would need a few addresses each. This led to the definition of three IP address classes, simply known as Class A, B, and C, for general use, and two special classes for multicast (Class D) and experimental (Class E) addresses. The **IP Address Classes** box shows how these address classes were assigned.

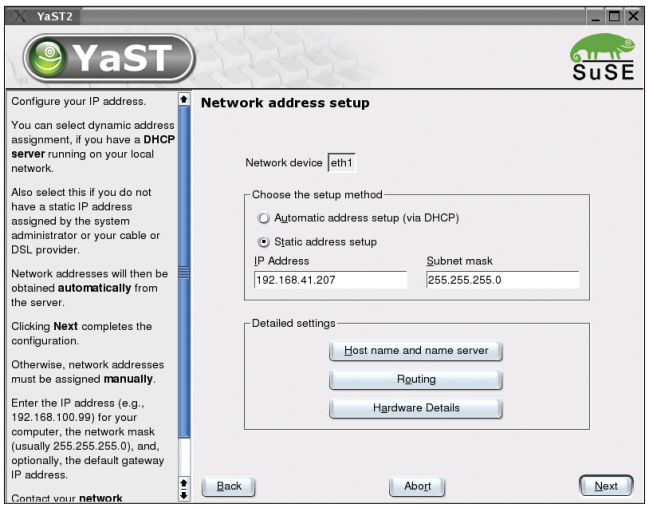

Figure 2: Customizing network settings on SuSE with YaST2

The ROADS Problem, CIDR and Private Addresses

When the internetworking pioneers devised the addressing scheme way back in the 70s, they had know way of knowing how important the Internet would soon become. By 1993 it was already apparent that a classful scheme was incapable of providing a unique address to allow every computer in the world to attach to the global Internet. This problem became known as the ROADS (Running Out of ADdress Space) problem. Looking at scientific publications from the early 90s reveals that scientists expected to run out of Internet addresses by the turn of the century. So what happened to change that?

For one thing, Classless Inter-Domain Routing (CIDR) or "supernetting" was introduced. This allowed the Internet authorities to assign enterprises that had applied for a Class B network address block multiple Class C network addresses instead. People also started asking themselves if it made sense to assign unique Internet addresses to all the computers on their networks.

Private Networks

There is no need to worry about routing or the lack of Internet addresses, if your local network is not connected to the Internet. You can simply assign the same network ID to all your computers, and use an appropriate mask. But if you tried to connect this local network up to the Internet, this could cause chaos. To prevent this, the IETF (Internet Engineering Task Force) designated a number of addresses as private.

You can experiment with private addresses without any danger. As private addresses are not assigned on the Internet and not routed onto the Internet, address conflicts can be ruled out. The addresses shown in Table 1 have been classified as private.

Accessing the Internet with a Private Address

You may already have noticed that the IP address in our example (192.16.4.207) is a private address, and may be asking yourself how the computer in this example can connect up to the Internet. Each data packet the computer produces contains the sender's address, and as we know, this address is not valid outside the local network.

There are two solutions to this. Of course you could use public addresses for your network. But this would mean you renting these address from your Provider. The other, and more common approach, is to use a modem, an ISDN card, or a cable or DSL modem to connect to the Internet. In this case you are typically assigned a single temporary address for the external connection. On the surface, it might look like you have a problem: a whole address block is too expensive, but you want all of your computers to have Internet access. Again there is a simple solution.

This solution assumes that the router, whose address you configure as the de-

GLOSSARY

IP Address Classes:

Class A 0.0.0.0 – 127.255.255.255 – subnet mask 255.0.0.0, host ID max. 24 bits

Class B 128.0.0.0 – 191.255.255.255 – subnet mask 255.255.0.0, host ID max. 16 bits

Class C 192.0.0.0 – 223.255.255.255 – subnet mask 255.255.255.0, host ID max. 8 bits

Table 1: Private addresses

Lowest address	Highest address
10.0.0.0	10.255.255.255
172.16.0.0	172.31.255.255
192.168.0.0	192.168.255.255
169.254.0.0	169.254.255.255

Table 2: IP routing table

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
192.168.41.0	*	255.255.255.0	U	0	0	0	etho
default	ROUTER_ONE.dd-i	0.0.0.0	UG	0	0	0	etho

fault route for all the other computers on your network, can handle Network Address Translation (NAT). The NAT gateway simply replaces the IP source address in each outgoing packet with its own public, and thus routable, address. This process is reversed for the incoming packets. The gateway replaces the target address with the private IP address of the machine waiting for a response from the Internet.

From the viewpoint of the machine on the internal network, the NAT device is a router, through which it can access external addresses. And the Internet sees only the NAT device – nobody knows that it forwards the packages and translates their addresses.

If you are interested in finding out what your machine's default route looks like, you can simply type *route* in command line. The output will show you the same details as you can see in Table 2.

Memory Hooks

You may be asking yourself if you will be expected to remember all those IP addresses. The answer is no – thanks to DNS! It is the Domain Name System that allows us humans, who are very good at remembering intuitive names, such as *www.linux-magazine.com*, to avoid difficult IP addresses, such as 192.76.144.15.

In smaller networks you might prefer to add a few lines to */etc/hosts*, and distribute this file to all your computers, rather than going to the trouble of setting up DNS. You can use your favorite editor (or vi) to create this file, which will look something like the following.

```
# cat /etc/hosts
#sample hosts file
192.168.41.3   samba
192.168.41.4   sqlse
192.168.41.7   kyocera
192.168.41.8   oki
...
```

When an application wants to talk to another computer, whose name it knows, it asks the operating system to look for an appropriate entry in */etc/hosts*. If that entry can be found, the application can use the IP address mapped by the entry – you might liken this to a private address book.

In contrast, DNS is like a global phone book. The DNS servers on the Internet are organized in a name hierarchy. If a server is unable to resolve a specific name, it knows the address of a DNS server higher up in the hierarchy that can answer the query, or even pass the answer on to the requesting machine.

DHCP:Temporary IP Addresses

The network communication tasks we have talked about so far, all assume that your local machine has been set up with an IP address, the net mask, the IP address of your router and in most cases the IP address of at least one DNS server. In many cases, there is no need to enter all these details manually. Larger networks will tend to use the Dynamic Host Configuration Protocol (DHCP) to set up the configuration for both the local network and Internet access. And as addresses are scarce on the largest of networks, the Internet, providers tend to work with temporary addresses that are assigned for the duration of an Internet session. This is a task that a flat file like */etc/hosts* cannot hope to handle.

DHCP allows a client that needs an IP address to search for a DHCP server by broadcasting a request on the local network. Any DHCP server that hears this request will respond with an offer. The computer accepts the first offer it hears, and the DHCP server confirms by repeating the terms of its offer.

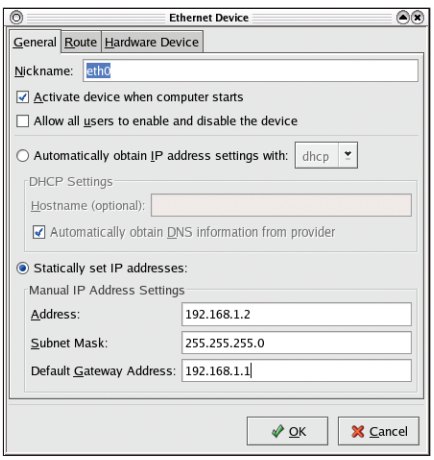

Figure 3: Customizing network settings on Red Hat with redhat-config-network

Configuration

Depending on your distribution, your TCP/IP network configuration can be stored in a number of different locations. The *hostname* command will tell you the hostname for your local machine:

```
#hostname
linux
```

Smaller networks that do not use a local DNS server will store hostname and IP address mappings in */etc/hosts*. The file may not be populated if you have a DNS server. The */etc/resolv.conf* file provides the name of the nearest name server, which will be contacted when your machine needs to resolve a DNS name to reach a target. SuSE 8.2 stores network settings in the */etc/sysconfig/network* directory. This path contains a */etc/sysconfig/network/ifcfg-xxxx* file for each network adapter, where xxxx represents the name of the network card, such as *eth0*. The following command outputs the settings for a local Ethernet card, *eth0*, which is set up to use DHCP:

```
#cat /etc/sysconfig/network➥
/ifcfg-eth0
BOOTPROTO='dhcp'
MTU=''
REMOTE_IPADDR=''
STARTMODE='onboot'
UNIQUE=''
WIRELESS='no'
```

In addition to the traditional file-based approach, modern distributions also provide GUI tools. On SuSE Linux you launch *YaST2* and, after typing your superuser password, select *Network Devices/Network Card*. The corresponding tool for Red Hat Linux is called *redhat-config-network*.

After this background knowledge, you might want to get down to some serious configuration work. The Gateway article on page 80 takes you through the steps required to connect your local SOHO network up to the Internet. ■