kernal configuration parameter help—networking part

All the network related options are selected either directly to the kernel or compiled as modules.

In Prague, if there's module option, compile it as modules
In Dublin, all options are directly compiled to kernel.

CONFIG_PACKET:

The Packet protocol is used by applications which communicate directly with network devices without an intermediate network protocol implemented in the kernel, e.g. tcpdump. If you want them to work, choose Y.

This driver is also available as a module called af_packet.o (= code which can be inserted in and removed from the running kernel whenever you want). If you want to compile it as a module, say M here and read Documentation/modules.txt; if you use modprobe or kmod, you may also want to add "alias net-pf-17 af_packet" to /etc/modules.conf.

CONFIG_PACKET_MMAP:

If you say Y here, the Packet protocol driver will use an IO mechanism that results in faster communication.

In this project: YES
I am not sure which command will use it. Just want to try

CONFIG_NETLINK:

This driver allows for two-way communication between the kernel and user processes. It does so by creating a new socket family, PF_NETLINK. Over this socket, the kernel can send and receive datagrams carrying information. It is documented on many systems in netlink(7), a HOWTO is provided as well, for example on http://snafu.freedom.org/linux2.2/docs/netlink-HOWTO.html

So far, the kernel uses this feature to publish some network related information if you say Y to "Routing messages", below. You also need to say Y here if you want to use arpd, a daemon that helps keep the internal ARP cache (a mapping between IP addresses and hardware addresses on the local network) small. The ethertap device, which lets user space programs read and write raw Ethernet frames, also needs the network link driver.

CONFIG_RTNETLINK:

If you say Y here, user space programs can receive some network related routing information over the netlink. 'rtmon', supplied with the iproute2 package (ftp://ftp.inr.ac.ru), can read and interpret this data. Information sent to the kernel over this link is ignored.

CONFIG_NETLINK_DEV:

This option will be removed soon. Any programs that want to use character special nodes like /dev/tap0 or /dev/route (all with major number 36) need this option, and need to be rewritten soon to use the real netlink socket. This is a backward compatibility option, choose Y for now.

CONFIG_NETFILTER:

Netfilter is a framework for filtering and mangling network packets that pass through your Linux box.

The most common use of packet filtering is to run your Linux box as a firewall protecting a local network from the Internet. The type of firewall provided by this kernel support is called a "packet filter", which means that it can reject individual network packets based on type, source, destination etc. The other kind of firewall,a "proxy-based" one, is more secure but more intrusive and more bothersome to set up; it inspects the network traffic much more closely, modifies it and has knowledge about the higher level protocols, which a packet filter lacks. Moreover, proxy-based firewalls often require changes to the programs running on the local clients. Proxy-based firewalls don't need support by the kernel, but they are often combined with a packet filter, which only works if you say Y here.

You should also say Y here if you intend to use your Linux box as the gateway to the Internet for a local network of machines without globally valid IP addresses. This is called "masquerading": if one of the computers on your local network wants to send something to the outside, your box can "masquerade" as that computer, i.e. it forwards the traffic to the intended outside destination, but modifies the packets to make it look like they came from the firewall box itself. It works both ways: if the outside host replies, the Linux box will silently forward the traffic to the correct local computer. This way, the computers on your local net are completely invisible to the outside world, even though they can reach the outside and can receive replies. It is even possible to run globally visible servers from within a masqueraded local network using a mechanism called portforwarding. Masquerading is also often called NAT (Network Address Translation).

Another use of Netfilter is in transparent proxying: if a machine on the local network tries to connect to an outside host, your Linux box can transparently forward the traffic to a local server, typically a caching proxy server.

Various modules exist for netfilter which replace the previous masquerading (ipmasqadm), packet filtering (ipchains), transparent proxying, and portforwarding mechanisms. Please see Documentation/Changes under "iptables" for the location of these packages.

Make sure to say N to "Fast switching" below if you intend to say Y here, as Fast switching currently bypasses netfilter.

Chances are that you should say Y here if you compile a kernel which will run as a router and N for regular hosts. If unsure, say N.

CONFIG_NETFILTER_DEBUG:

You can say Y here if you want to get additional messages useful in debugging the netfilter code.

CONFIG_FILTER:

The Linux Socket Filter is derived from the Berkeley Packet Filter. If you say Y here, user-space programs can attach a filter to any socket and thereby tell the kernel that it should allow or disallow certain types of data to get through the socket. Linux Socket Filtering works on all socket types except TCP for now. See the text file Documentation/networking/filter.txt for more information.

You need to say Y here if you want to use PPP packet filtering (see the CONFIG_PPP_FILTER option below).

CONFIG_UNIX:

If you say Y here, you will include support for Unix domain sockets; sockets are the standard Unix mechanism for establishing and accessing network connections. Many commonly used programs such as the X Window system and syslog use these sockets even if your machine is not connected to any network. Unless you are working on an embedded system or something similar, you therefore definitely want to say Y here.

However, the socket support is also available as a module (= code which can be inserted in and removed from the running kernel whenever you want). If you want to compile it as a module, say M here and read Documentation/modules.txt. The module will be called unix.o. If you try building this as a module and you have said Y to "Kernel module loader support" above, be sure to add 'alias net-pf-1 unix' to your /etc/modules.conf file. Note that several important services won't work correctly if you say M here and then neglect to load the module.

Say Y unless you know what you are doing.

CONFIG_INET:

These are the protocols used on the Internet and on most local Ethernets. It is highly recommended to say Y here (this will enlarge your kernel by about 144 KB), since some programs (e.g. the X window system) use TCP/IP even if your machine is not connected to any other computer. You will get the so-called loopback device which allows you to ping yourself (great fun, that!).

For an excellent introduction to Linux networking, please read the NET-3-HOWTO, available from http://www.linuxdoc.org/docs.html#howto .

This option is also necessary if you want to use the full power of term (term is a program which gives you almost full Internet connectivity if you have a regular dial up shell account on some Internet connected Unix computer; for more information, read http://www.bart.nl/~patrickr/term-howto/Term-HOWTO.html).

If you say Y here and also to "/proc file system support" and "Sysctl support" below, you can change various aspects of the behavior of the TCP/IP code by writing to the (virtual) files in /proc/sys/net/ipv4/*; the options are explained in the file Documentation/networking/ip-sysctl.txt.

Short answer: say Y.

CONFIG_IP_MULTICAST:

This is code for addressing several networked computers at once,enlarging your kernel by about 2 KB. You need multicasting if you intend to participate in the MBONE, a high bandwidth network on top of the Internet which carries audio and video broadcasts. More information about the MBONE is on the WWW at http://www-itg.lbl.gov/mbone/ . Information about the multicast capabilities of the various network cards is contained in Documentation/networking/multicast.txt. For most people, it's safe to say N.

CONFIG_IP_ADVANCED_ROUTER:

If you intend to run your Linux box mostly as a router, i.e. as a computer that forwards and redistributes network packets, say Y; you will then be presented with several options that allow more precise control about the routing process.

The answer to this question won't directly affect the kernel: answering N will just cause this configure script to skip all the questions about advanced routing.

Note that your box can only act as a router if you enable IP forwarding in your kernel; you can do that by saying Y to "/proc file system support" and "Sysctl support" below and executing the line

```
  echo "1" > /proc/sys/net/ipv4/ip_forward
```

at boot time after the /proc file system has been mounted.

If you turn on IP forwarding, you will also get the rp_filter, which automatically rejects incoming packets if the routing table entry for their source address doesn't match the network interface they're arriving on. This has security advantages because it prevents the so-called IP spoofing, however it can pose problems if you use asymmetric routing (packets from you to a host take a different path than packets from that host to you) or if you operate a non-routing host which has several IP addresses on different interfaces. To turn rp_filter off use:

```
      echo 0 > /proc/sys/net/ipv4/conf/<device>/rp_filter
or
      echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter
```

CONFIG_IP_MULTIPLE_TABLES:

Normally, a router decides what to do with a received packet based solely on the packet's final destination address. If you say Y here, the Linux router will also be able to take the packet's source address into account. Furthermore, if you also say Y to "IP: use TOS value as routing key" below, the TOS (Type-Of-Service) field of the packet can be used for routing decisions as well. In addition, if you say Y here and to "IP: fast network address translation" below, the router will also be able to modify source and destination addresses of forwarded packets.

If you are interested in this, please see the preliminary documentation at http://www.compendium.com.ar/policy-routing.txt and ftp://post.tepkom.ru/pub/vol2/Linux/docs/advanced-routing.tex . You will need supporting software from ftp://ftp.inr.ac.ru/ip-routing/

CONFIG_IP_ROUTE_FWMARK:

If you say Y here, you will be able to specify different routes for packets with different mark values (see iptables(8), MARK target).

CONFIG_IP_ROUTE_NAT:

If you say Y here, your router will be able to modify source and destination addresses of packets that pass through it, in a manner you specify. General information about Network Address Translation can be gotten from the document http://www.csn.tu-chemnitz.de/~mha/linux-ip-nat/diplom/nat.html

CONFIG_IP_ROUTE_MULTIPATH:

Normally, the routing tables specify a single action to be taken in a deterministic manner for a given packet. If you say Y here however, it becomes possible to attach several actions to a packet pattern, in effect specifying several alternative paths to travel for those packets. The router considers all these paths to be of equal "cost" and chooses one of them in a non-deterministic fashion if a matching packet arrives.

CONFIG_IP_ROUTE_TOS:

The header of every IP packet carries a TOS (Type Of Service) value with which the packet requests a certain treatment, e.g. low latency (for interactive traffic), high throughput, or high reliability. Ifyou say Y here, you will be able to specify different routes for packets with different TOS values.

CONFIG_IP_ROUTE_VERBOSE:

If you say Y here, which is recommended, then the kernel will print verbose messages regarding the routing, for example warnings about received packets which look strange and could be evidence of an attack or a misconfigured system somewhere. The information is handled by the klogd daemon which is responsible for kernel messages("man klogd").

CONFIG_IP_ROUTE_LARGE_TABLES:

If you have routing zones that grow to more than about 64 entries,you may want to say Y here to speed up the routing process.

CONFIG_IP_PNP:

This enables automatic configuration of IP addresses of devices and of the routing table during kernel boot, based on either information supplied on the kernel command line or by BOOTP or RARP protocols. You need to say Y only for diskless machines requiring network access to boot (in which case you want to say Y to "Root file system on NFS" as well), because all other machines configure the network in their startup scripts.

CONFIG_NET_IPIP:

Tunneling means encapsulating data of one protocol type within another protocol and sending it over a channel that understands the encapsulating protocol. This particular tunneling driver implements encapsulation of IP within IP, which sounds kind of pointless, but can be useful if you want to make your (or some other) machine appear on a different network than it physically is, or to use mobile-IP facilities (allowing laptops to seamlessly move between networks without changing their IP addresses; check out http://anchor.cs.binghamton.edu/~mobileip/LJ/index.html).

Saying Y to this option will produce two modules (= code which can be inserted in and removed from the running kernel whenever you want). Most people won't need this and can say N.

CONFIG_NET_IPGRE:

Tunneling means encapsulating data of one protocol type within another protocol and sending it over a channel that understands the encapsulating protocol. This particular tunneling driver implements GRE (Generic Routing Encapsulation) and at this time allows encapsulating of IPv4 or IPv6 over existing IPv4 infrastructure. This driver is useful if the other endpoint is a Cisco router: Cisco likes GRE much better than the other Linux tunneling driver ("IP: tunneling" above). In addition, GRE allows multicast redistribution through the tunnel.

CONFIG_IP_MROUTE:

This is used if you want your machine to act as a router for IP packets that have several destination addresses. It is needed on the MBONE, a high bandwidth network on top of the Internet which carries audio and video broadcasts. In order to do that, you would most likely run the program mrouted. Information about the multicast capabilities of the various network cards is contained in Documentation/networking/multicast.txt. If you haven't heard about it, you don't need it.

CONFIG_IP_PIMSM_V1:

Kernel side support for Sparse Mode PIM (Protocol Independent Multicast) version 1. This multicast routing protocol is used widely because Cisco supports it. You need special software to use it (pimd-v1). Please see http://netweb.usc.edu/pim/ for more information about PIM.

Say Y if you want to use PIM-SM v1. Note that you can say N here if you just want to use Dense Mode PIM.

CONFIG_IP_PIMSM_V2:

Kernel side support for Sparse Mode PIM version 2. In order to use this, you need an experimental routing daemon supporting it (pimd or gated-5). This routing protocol is not used widely, so say N unless you want to play with it.

CONFIG_INET_ECN:

Explicit Congestion Notification (ECN) allows routers to notify clients about network congestion, resulting in fewer dropped packets and increased network performance. This option adds ECN support to the Linux kernel, as well as a sysctl (/proc/sys/net/ipv4/tcp_ecn) which allows ECN support to be disabled at runtime.

Note that, on the Internet, there are many broken firewalls which refuse connections from ECN-enabled machines, and it may be a while before these firewalls are fixed. Until then, to access a site behindsuch a firewall (some of which are major sites, at the time of this writing) you will have to disable this option, either by saying N now or by using the sysctl.

CONFIG_SYN_COOKIES:

Normal TCP/IP networking is open to an attack known as "SYN flooding". This denial-of-service attack prevents legitimate remote users from being able to connect to your computer during an ongoing attack and requires very little work from the attacker, who can operate from anywhere on the Internet.

SYN cookies provide protection against this type of attack. If you say Y here, the TCP/IP stack will use a cryptographic challenge protocol known as "SYN cookies" to enable legitimate users to continue to connect, even when your machine is under attack. There is no need for the legitimate users to change their TCP/IP software; SYN cookies work transparently to them. For technical information about SYN cookies, check out
ftp://koobera.math.uic.edu/syncookies.html .

If you are SYN flooded, the source address reported by the kernel is likely to have been forged by the attacker; it is only reported as an aid in tracing the packets to their actual source and should not be taken as absolute truth.

SYN cookies may prevent correct error reporting on clients when the server is really overloaded. If this happens frequently better turn them off.

If you say Y here, note that SYN cookies aren't enabled by default; you can enable them by saying Y to "/proc file system support" and "Sysctl support" below and executing the command

```
  echo 1 >/proc/sys/net/ipv4/tcp_syncookies
```

at boot time after the /proc file system has been mounted.

If unsure, say Y.

CONFIG_IP_NF_CONNTRACK:

Connection tracking keeps a record of what packets have passed through your machine, in order to figure out how they are related into connections.

This is required to do Masquerading or other kinds of Network Address Translation (except for Fast NAT). It can also be used to enhance packet filtering (see `Connection state match support' below).

If you want to compile it as a module, say M here and read Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP_NF_FTP:

Tracking FTP connections is problematic: special helpers are required for tracking them, and doing masquerading and other forms of Network Address Translation on them.

If you want to compile it as a module, say M here and read
Documentation/modules.txt. If unsure, say `Y'.

CONFIG_IP_NF_QUEUE:

Netfilter has the ability to queue packets to user space: the netlink device can be used to access them using this driver.

If you want to compile it as a module, say M here and read Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP_NF_IPTABLES:

iptables is a general, extensible packet identification framework. The packet filtering and full NAT (masquerading, port forwarding, etc) subsystems now use this: say `Y' or `M' here if you want to use either of those.

If you want to compile it as a module, say M here and read
Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP_NF_MATCH_LIMIT:

limit matching allows you to control the rate at which a rule can be matched: mainly useful in combination with the LOG target ("LOG target support", below) and to avoid some Denial of Service attacks.

If you want to compile it as a module, say M here and read
Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP_NF_MATCH_MAC:

mac matching allows you to match packets based on the source ethernet address of the packet.

If you want to compile it as a module, say M here and read
Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP_NF_MATCH_MARK:

Netfilter mark matching allows you to match packets based on the `nfmark' value in the packet. This can be set by the MARK target (see below).

If you want to compile it as a module, say M here and read
Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP_NF_MATCH_MULTIPORT:

Multiport matching allows you to match TCP or UDP packets based on a series of source or destination ports: normally a rule can only match a single range of ports.

If you want to compile it as a module, say M here and read
Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP_NF_MATCH_TOS:

TOS matching allows you to match packets based on the Type Of Service fields of the IP packet.

If you want to compile it as a module, say M here and read
Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP_NF_MATCH_TCPMSS:

This option adds a `tcpmss' match, which allows you to examine the MSS value of TCP SYN packets, which control the maximum packet size for that connection.

If you want to compile it as a module, say M here and read
Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP_NF_MATCH_STATE:

Connection state matching allows you to match packets based on their relationship to a tracked connection (ie. previous packets). This is a powerful tool for packet classification.

If you want to compile it as a module, say M here and read
Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP_NF_MATCH_UNCLEAN:

Unclean packet matching matches any strange or invalid packets, by looking at a series of fields in the IP, TCP, UDP and ICMP headers.

If you want to compile it as a module, say M here and read

Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP_NF_MATCH_OWNER:

Packet owner matching allows you to match locally-generated packets based on who created them: the user, group, process or session.

If you want to compile it as a module, say M here and read
Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP_NF_FILTER:

Packet filtering defines a table `filter', which has a series of rules for simple packet filtering at local input, forwarding and local output. See the man page for iptables(8).

If you want to compile it as a module, say M here and read
Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP_NF_TARGET_REJECT:

The REJECT target allows a filtering rule to specify that an ICMP error should be issued in response to an incoming packet, rather than silently being dropped.

If you want to compile it as a module, say M here and read
Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP_NF_TARGET_MIRROR:

The MIRROR target allows a filtering rule to specify that an incoming packet should be bounced back to the sender.

If you want to compile it as a module, say M here and read
Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP_NF_NAT:

The Full NAT option allows masquerading, port forwarding and other forms of full Network Address Port Translation. It is controlled by the `nat' table in iptables: see the man page for iptables(8).

If you want to compile it as a module, say M here and read Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP_NF_TARGET_MASQUERADE:

Masquerading is a special case of NAT: all outgoing connections are changed to seem to come from a particular interface's address, and if the interface goes down, those connections are lost. This is only useful for dialup accounts with dynamic IP address (ie. your IP address will be different on next dialup).

If you want to compile it as a module, say M here and read Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP_NF_TARGET_REDIRECT:

REDIRECT is a special case of NAT: all incoming connections are mapped onto the incoming interface's address, causing the packets to come to the local machine instead of passing through. This is useful for transparent proxies.

If you want to compile it as a module, say M here and read Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP_NF_MANGLE:

This option adds a `mangle' table to iptables: see the man page for iptables(8). This table is used for various packet alterations which can effect how the packet is routed.

If you want to compile it as a module, say M here and read Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP_NF_TARGET_TOS:

This option adds a `TOS' target, which allows you to create rules in the `mangle' table which alter the Type Of Service field of an IP packet prior to routing.

If you want to compile it as a module, say M here and read Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP_NF_TARGET_MARK:

This option adds a `MARK' target, which allows you to create rules in the `mangle' table which alter the netfilter mark (nfmark) field associated with the packet packet prior to routing. This can change the routing method (see `IP: use netfilter MARK value as routing key') and can also be used by other subsystems to change their behavior.

If you want to compile it as a module, say M here and read Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP_NF_TARGET_LOG:

This option adds a `LOG' target, which allows you to create rules in any iptables table which records the packet header to the syslog.

If you want to compile it as a module, say M here and read Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP_NF_TARGET_TCPMSS:

This option adds a `TCPMSS' target, which allows you to alter the MSS value of TCP SYN packets, to control the maximum size for that connection (usually limiting it to your outgoing interface's MTU minus 40).

This is used to overcome criminally braindead ISPs or servers which block ICMP Fragmentation Needed packets. The symptoms of this problem are that everything works fine from your Linux firewall/router, but machines behind it can never exchange large packets:

1) Web browsers connect, then hang with no data received.
2) Small mail works fine, but large emails hang.
3) ssh works fine, but scp hangs after initial handshaking.

Workaround: activate this option and add a rule to your firewall configuration like:

```
      iptables -A FORWARD -p tcp --tcp-flags SYN,RST SYN \
             -j TCPMSS --clamp-mss-to-pmtu
```

If you want to compile it as a module, say M here and read Documentation/modules.txt. If unsure, say `N'.

CONFIG_IPV6:

This is experimental support for the next version of the Internet Protocol: IP version 6 (also called IPng "IP next generation").Features of this new protocol include: expanded address space, authentication and privacy, and seamless interoperability with the current version of IP (IP version 4). For general information about IPv6, see http://playground.sun.com/pub/ipng/html/ipng-main.html ; for specific information about IPv6 under Linux read the HOWTO at http://www.bieringer.de/linux/IPv6/ and the file net/ipv6/README in the kernel source.

If you want to use IPv6, please upgrade to the newest net-tools as given in Documentation/Changes. You will still be able to do regular IPv4 networking as well.

This protocol support is also available as a module (= code which can be inserted in and removed from the running kernel whenever you want). The module will be called ipv6.o. If you want to compile it as a module, say M here and read Documentation/modules.txt.

It is safe to say N here for now.

CONFIG_IP6_NF_IPTABLES:

ip6tables is a general, extensible packet identification framework. Currently only the packet filtering and packet mangling subsystem for IPv6 use this, but connection tracking is going to follow. Say 'Y' or 'M' here if you want to use either of those.

If you want to compile it as a module, say M here and read Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP6_NF_MATCH_LIMIT:

limit matching allows you to control the rate at which a rule can be matched: mainly useful in combination with the LOG target ("LOG target support", below) and to avoid some Denial of Service attacks.

If you want to compile it as a module, say M here and read Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP6_NF_MATCH_MARK:

Netfilter mark matching allows you to match packets based on the `nfmark' value in the packet. This can be set by the MARK target (see below).

If you want to compile it as a module, say M here and read

Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP6_NF_FILTER:

Packet filtering defines a table `filter', which has a series of rules for simple packet filtering at local input, forwarding and local output. See the man page for iptables(8).

If you want to compile it as a module, say M here and read Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP6_NF_MANGLE:

This option adds a `mangle' table to iptables: see the man page for iptables(8). This table is used for various packet alterations which can effect how the packet is routed.

If you want to compile it as a module, say M here and read Documentation/modules.txt. If unsure, say `N'.

CONFIG_IP6_NF_TARGET_MARK:

This option adds a `MARK' target, which allows you to create rules in the `mangle' table which alter the netfilter mark (nfmark) field associated with the packet packet prior to routing. This can change the routing method (see `IP: use netfilter MARK value as routing key') and can also be used by other subsystems to change their behavior.

If you want to compile it as a module, say M here and read Documentation/modules.txt. If unsure, say `N'.

CONFIG_ATM:

ATM is a high-speed networking technology for Local Area Networks and Wide Area Networks. It uses a fixed packet size and is connection oriented, allowing for the negotiation of minimum bandwidth requirements.

In order to participate in an ATM network, your Linux box needs an ATM networking card. If you have that, say Y here and to the driver of your ATM card below.

Note that you need a set of user-space programs to actually make use of ATM. See the file Documentation/networking/atm.txt for further details.

Why not?

No ATM in this project at all.

The IPX protocol
Appletalk protocol support
DECnet Support
802.1d Ethernet Bridging
CCITT X.25 Packet Layer (EXPERIMENTAL)

For this project, select NO to the above options. Because all protocols involved here is IP or IPv6.

Part of my project will be on Queueing topics. The following options on QoS topics will be chosen.

CONFIG_NET_SCHED:

When the kernel has several packets to send out over a network device, it has to decide which ones to send first, which ones to delay, and which ones to drop. This is the job of the packet scheduler, and several different algorithms for how to do this "fairly" have been proposed.

If you say N here, you will get the standard packet scheduler, which is a FIFO (first come, first served). If you say Y here, you will be able to choose from among several alternative algorithms which can then be attached to different network devices. This is useful for example if some of your network devices are real time devices that need a certain minimum data flow rate, or if you need to limit the maximum data flow rate for traffic which matches specified criteria. This code is considered to be experimental.

To administer these schedulers, you'll need the user-level utilities from the package iproute2+tc at ftp://ftp.inr.ac.ru/ip-routing/ . That package also contains some documentation; for more, check out http://snafu.freedom.org/linux2.2/iproute-notes.html .

This Quality of Service (QoS) support will enable you to use Differentiated Services (diffserv) and Resource Reservation Protocol (RSVP) on your Linux router if you also say Y to "QoS support", "Packet classifier API" and to some classifiers below. Documentation and software is at http://icawww1.epfl.ch/linux-diffserv/ .

If you say Y here and to "/proc file system" below, you will be able to read status information about packet schedulers from the file /proc/net/psched.

The available schedulers are listed in the following questions; you can say Y to as many as you like. If unsure, say N now.

CONFIG_NET_SCH_CBQ:

Say Y here if you want to use the Class-Based Queueing (CBQ) packet scheduling algorithm for some of your network devices. This algorithm classifies the waiting packets into a tree-like hierarchy of classes; the leaves of this tree are in turn scheduled by separate algorithms (called "disciplines" in this context).

See the top of net/sched/sch_cbq.c for references about the CBQ algorithm.

CBQ is a commonly used scheduler, so if you're unsure, you should say Y here. Then say Y to all the queueing algorithms below that you want to use as CBQ disciplines. Then say Y to "Packet classifier API" and say Y to all the classifiers you want to use; a classifier is a routine that allows you to sort your outgoing traffic into classes based on a certain criterion.

This code is also available as a module called sch_cbq.o (= code which can be inserted in and removed from the running kernel whenever you want). If you want to compile it as a module, say M here and read Documentation/modules.txt.

CONFIG_NET_SCH_CSZ:

Say Y here if you want to use the Clark-Shenker-Zhang (CSZ) packet scheduling algorithm for some of your network devices. At the moment, this is the only algorithm that can guarantee service for real-time applications (see the top of net/sched/sch_csz.c for details and references about the algorithm).

Note: this scheduler is currently broken.

This code is also available as a module called sch_csz.o (= code which can be inserted in and removed from the running kernel whenever you want). If you want to compile it as a module, say M here and read Documentation/modules.txt.

CONFIG_NET_SCH_PRIO:

Say Y here if you want to use an n-band priority queue packet "scheduler" for some of your network devices or as a leaf discipline for the CBQ scheduling algorithm. If unsure, say Y.

This code is also available as a module called sch_prio.o (= code which can be inserted in and removed from the running kernel whenever you want). If you want to compile it as a module, say M here and read Documentation/modules.txt.

CONFIG_NET_SCH_RED:

Say Y here if you want to use the Random Early Detection (RED) packet scheduling algorithm for some of your network devices (see the top of net/sched/sch_red.c for details and references about the algorithm).

This code is also available as a module called sch_red.o (= code which can be inserted in and removed from the running kernel whenever you want). If you want to compile it as a module, say M here and read Documentation/modules.txt.

CONFIG_NET_SCH_SFQ:

Say Y here if you want to use the Stochastic Fairness Queueing (SFQ) packet scheduling algorithm for some of your network devices or as a leaf discipline for the CBQ scheduling algorithm (see the top of net/sched/sch_sfq.c for details and references about the SFQ algorithm).

This code is also available as a module called sch_sfq.o (= code which can be inserted in and removed from the running kernel whenever you want). If you want to compile it as a module, say M here and read Documentation/modules.txt.

CONFIG_NET_SCH_TEQL:

Say Y here if you want to use the True Link Equalizer (TLE) packet scheduling algorithm for some of your network devices or as a leaf discipline for the CBQ scheduling algorithm. This queueing iscipline allows the combination of several physical devices into one virtual device. (see the top of net/sched/sch_teql.c for details).

This code is also available as a module called sch_teql.o (= code which can be inserted in and removed from the running kernel whenever you want). If you want to compile it as a module, say M here and read Documentation/modules.txt.

CONFIG_NET_SCH_TBF:

Say Y here if you want to use the Simple Token Bucket Filter (TBF) packet scheduling algorithm for some of your network devices or as a leaf discipline for the CBQ scheduling algorithm (see the top ofnet/sched/sch_tbf.c for a description of the TBF algorithm).

This code is also available as a module called sch_tbf.o (= code which can be inserted in and removed from the running kernel whenever you want). If you want to compile it as a module, say M here and read Documentation/modules.txt.