Enhance TCP performance with multiple path routing; Use Eclipse to debug Linux Kernel Networking Code

October 5, 2004

Frank Watson

User Mode Linux Overview

User Mode Linux (acro: uml) is composed of two different parts:

1) Linux executable

- Created by taking the kernel source tree and applying a patch from UML's website.
- The Linux kernel as an application. Advantages: modifications do not crash host machine and able to attach a debugger. Disadvantage: currently unable to test device drivers

2) Root file system

- A byte per byte copy of an operating system's file system includes: libraries, compilers, shells, and anything that makes an operating system work. Conceptual example: CD Iso images.

How User Mode Linux works

- Commands
- linux umid=lamb udb0=root_fs ubd=mmap eth0=tuntap,,,172.31.0.130 eth1=tuntap,,,172.31.0.131 mem=32M udb2=swap

linux	Executable
udb0=root_fs	Name of root file system
umid=lamb	UML ID – for interface with UML console
ubd=mmap	Not necessary -- use /dev/anon for mem
eth0=tuntap,,,172.31.0.130	Use tuntap driver with ip address 0.130
eth1=tuntap,,,172.31.0.131	2nd device driver
mem=32M	Amount of memory to use
udb2=swap	Swap file

- Networking – tuntap

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
172.31.0.172    *               255.255.255.255 UH    0      0        0 tap0
10.0.1.172      *               255.255.255.255 UH    0      0        0 tap1
172.31.0.0      *               255.255.255.0   U     0      0        0 eth0
169.254.0.0     *               255.255.0.0     U     0      0        0 eth0
127.0.0.0       *               255.0.0.0       U     0      0        0 lo
default         172.31.0.1      0.0.0.0         UG    0      0        0 eth0
```

Debugging

- Allows for Back Trace
  - When kernel panics calls kernel/panic.c:panic. This helps in determining what caused the kernel crash.
- Examination of variables
  - sk_buff and the sock (INET socket) are the most important variable while tracing networking code.
- Stepping through code to find the code path
  - What is populated in the sk_buff is not as important as when. Being able to step through the code allows the developer to see when a method is populated.

Additional features of UML

- Two minute build time (pending on machine)
  - Not all the device drivers are built. Saves a lot of time.
- Instance results
  - Do not have to mess with the system map, move the bzImage (kernel file), and play with modules.

How to install UML

·Download 3 main files

Kernel source code, UML kernel patch, and Root file system

·Installation

Unpack the kernel – "tar xvjf <kernelSourceCode>.tar.bz2”

Apply the UML patch -- "patch -p1 < patchFile" at top of the source tree

Build the "linux“ executable from the linux source – "make xconfig ARCH=um; make dep; make linux ARCH=um”

·Running UML – executable is located at the top of the source tree
execute uml with the following command line:

linux mem=128M udb=root_fs_slackware_7.0_big udb2=swap debug=go eth0=tuntap,,,<IP address # 1>

·Setting up the internet connection once UML opens
Use the following commands:

Ifconfig eth0 <IP address #2>

Route del –net 172.31.0.0 dev eth0 netmask 255.255.0.0

Route add –host <IP of host machine> dev eth0

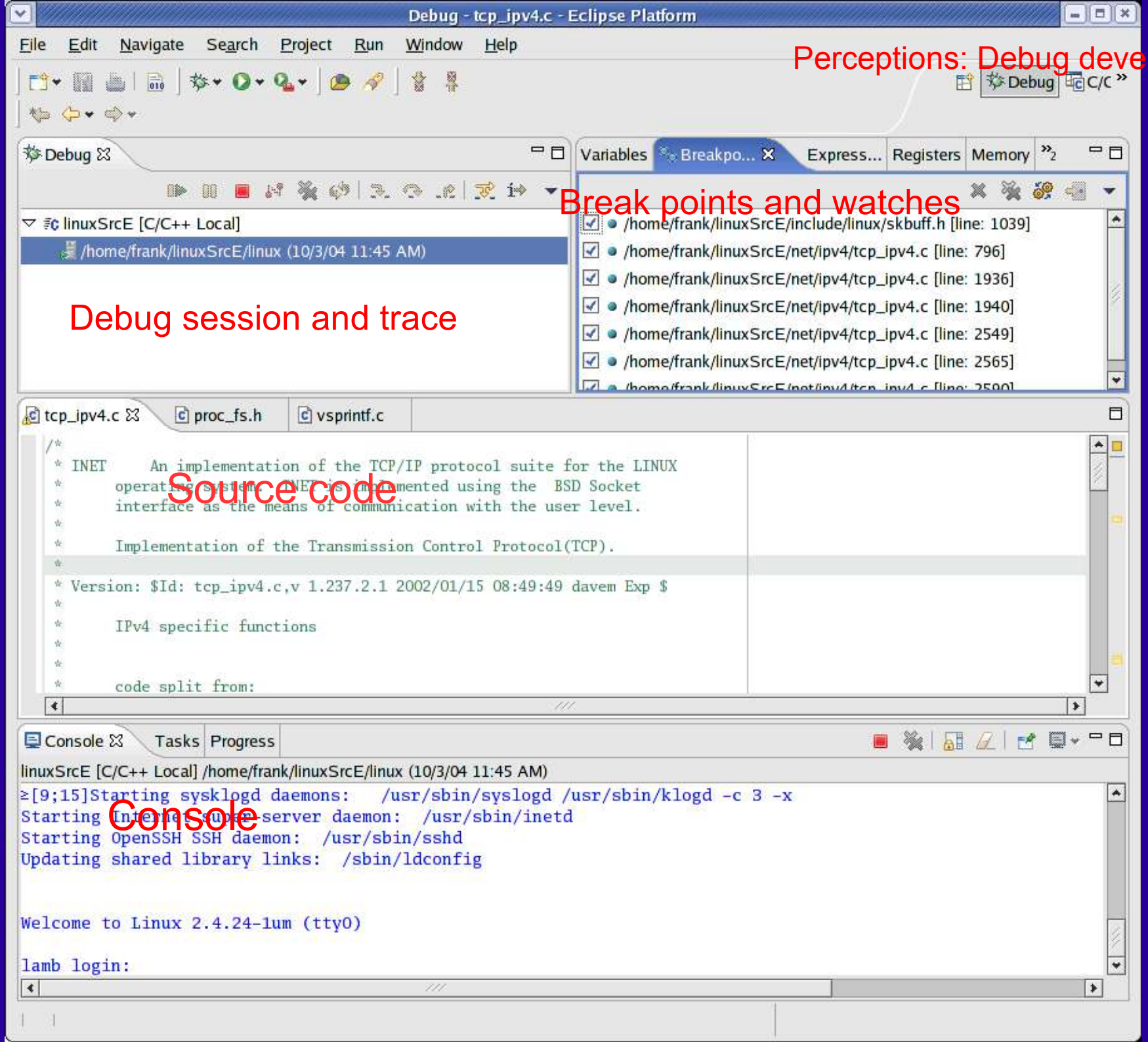
Route add default gw <IP of host machine>

Eclipse

- Description
  - Started by IBM. Open Source project and has over thirty companies (to name a few: Borland, Rational – before being merging with IBM, Red Hat, SuSE, Intel,Compuware, Novell, Oracle, PalmSource, Fujitsu, Genuitec, Hitachi Software)
- Plug-ins
  - Eclipse works on a plug-in scheme and allows additional functionality. For example, if you like the features in Borland's Jbuilder. Download (for a price) the plug-in and get these features incorporated into Eclipse.

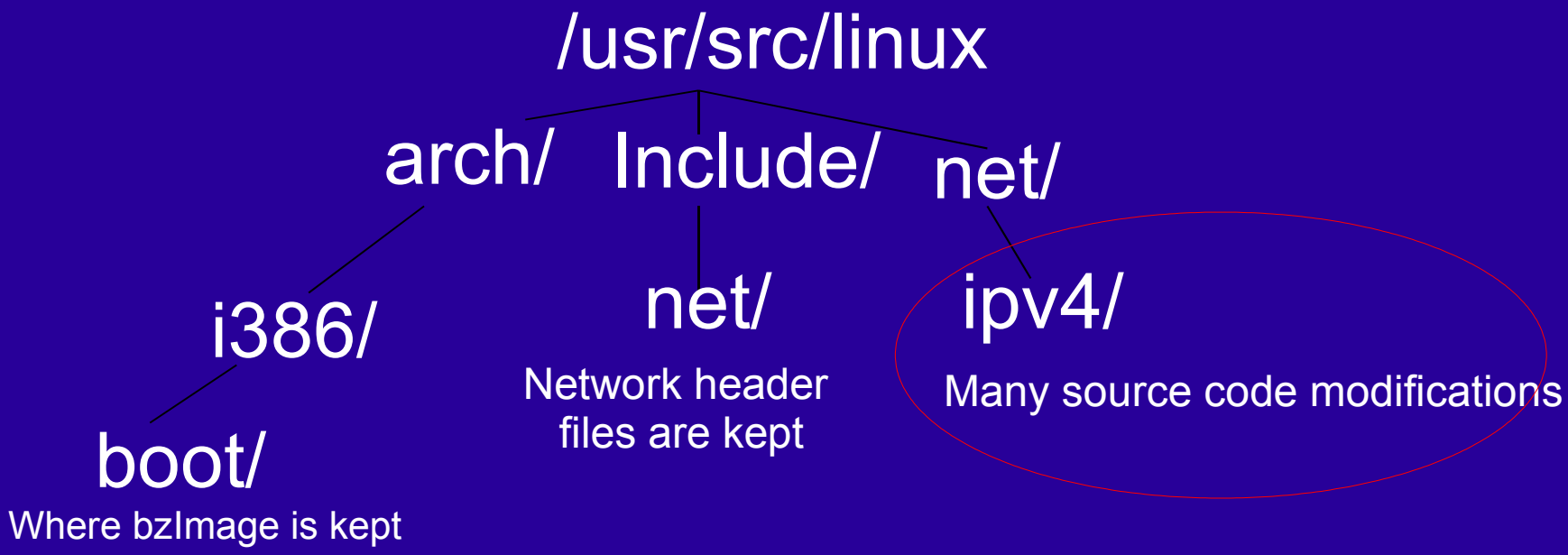
Eclipse CDT Plugin

- Allows for C/C++ development
- Perspective windows
  - Partitions out the functionality – keeps everything from being cluttered
- Scanning
  - Four different scanners which look at the source code, make files, and binary code.
- Interfaces with GDB
  - Allows the use of a .gdbinit file to initialize GDB

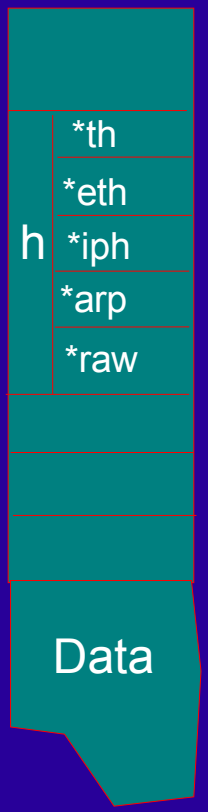

Debug - tcp_ipv4.c - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Perceptions: Debug development
Debug C/C
Debug
linuxSrcE [C/C++ Local]
/home/frank/linuxSrcE/linux (10/3/04 11:45 AM)
Debug session and trace
Variables Breakpo... Express... Registers Memory
Break points and watches
/home/frank/linuxSrcE/include/linux/skbuff.h [line: 1039]
/home/frank/linuxSrcE/net/ipv4/tcp_ipv4.c [line: 796]
/home/frank/linuxSrcE/net/ipv4/tcp_ipv4.c [line: 1936]
/home/frank/linuxSrcE/net/ipv4/tcp_ipv4.c [line: 1940]
/home/frank/linuxSrcE/net/ipv4/tcp_ipv4.c [line: 2549]
/home/frank/linuxSrcE/net/ipv4/tcp_ipv4.c [line: 2565]
tcp_ipv4.c proc_fs.h vsprintf.c
Source code
/*
* INET An implementation of the TCP/IP protocol suite for the LINUX
operating system. INET is implemented using the BSD Socket
interface as the means of communication with the user level.
*
Implementation of the Transmission Control Protocol(TCP).
*
* Version: $Id: tcp_ipv4.c,v 1.237.2.1 2002/01/15 08:49:49 davem Exp $
*
IPv4 specific functions
*
*
code split from:
Console Tasks Progress
linuxSrcE [C/C++ Local] /home/frank/linuxSrcE/linux (10/3/04 11:45 AM)
Console
≥[9;15]Starting sysklogd daemons: /usr/sbin/syslogd /usr/sbin/klogd -c 3 -x
Starting Internet Super-server daemon: /usr/sbin/inetd
Starting OpenSSH SSH daemon: /usr/sbin/sshd
Updating shared library links: /sbin/ldconfig
Welcome to Linux 2.4.24-1um (tty0)
lamb login:

Where to start (Linux source tree)

- Explanation of /usr/src/linux)

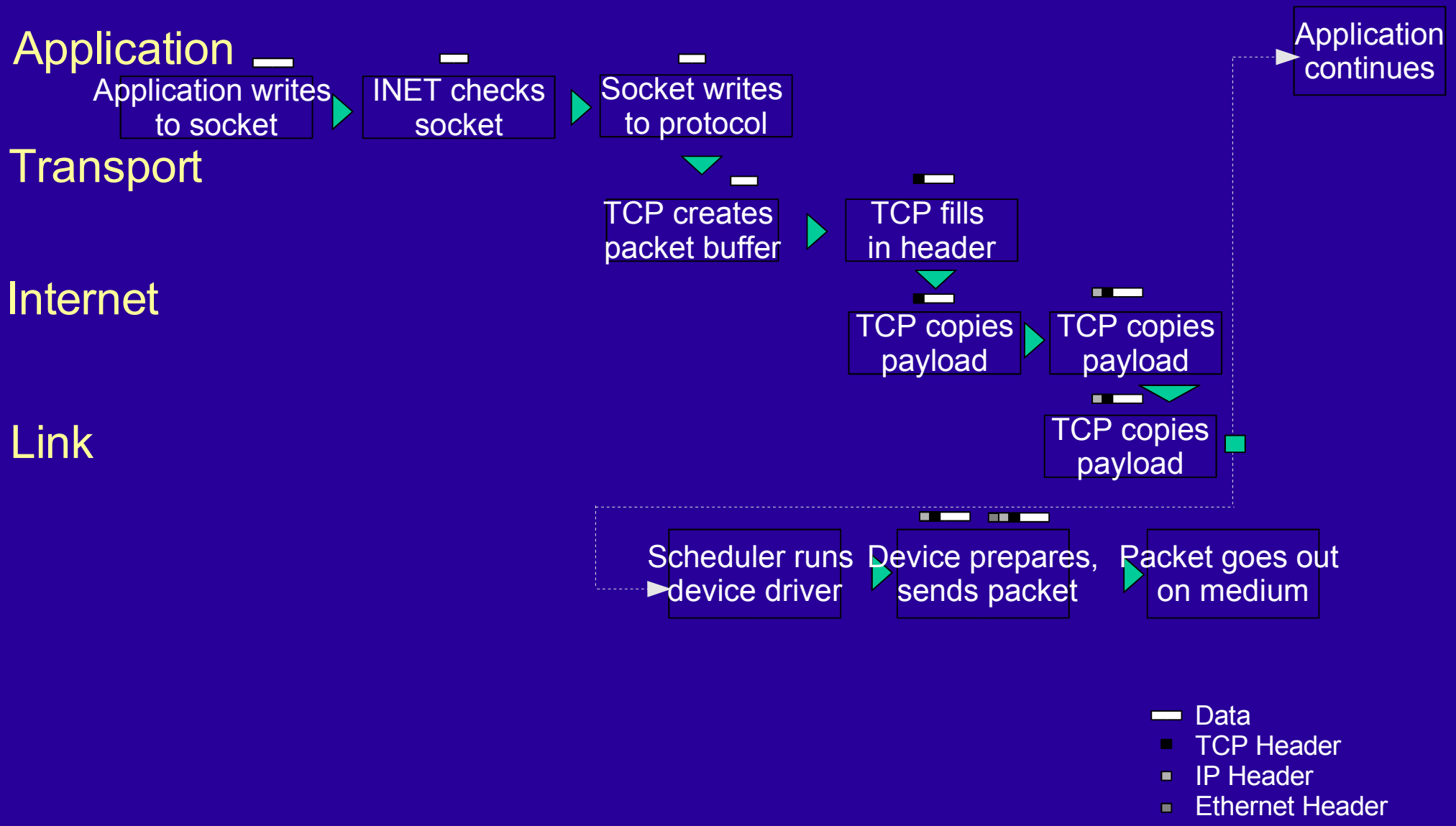


Sk Buff / Socks



- Structure that is interwoven through the entire network delivery of an internet packet.
- This data structure is fine tuned and works really well.
- Located in the **/usr/src/linux/include/linux** directory. Declared in a header file called **skbuff.h**
- Sockets are the data structures used to route the header-less packets when they are first created.
- There are two different types of Sockets in Linux: BSD and INET
- BSD is the socket interface which interacts with the user; within the BSD socket an INET socket (can also be multiple INET sockets link listed) . INET sockets do the rest of the work and are sent with the packet's receiving or destination info.

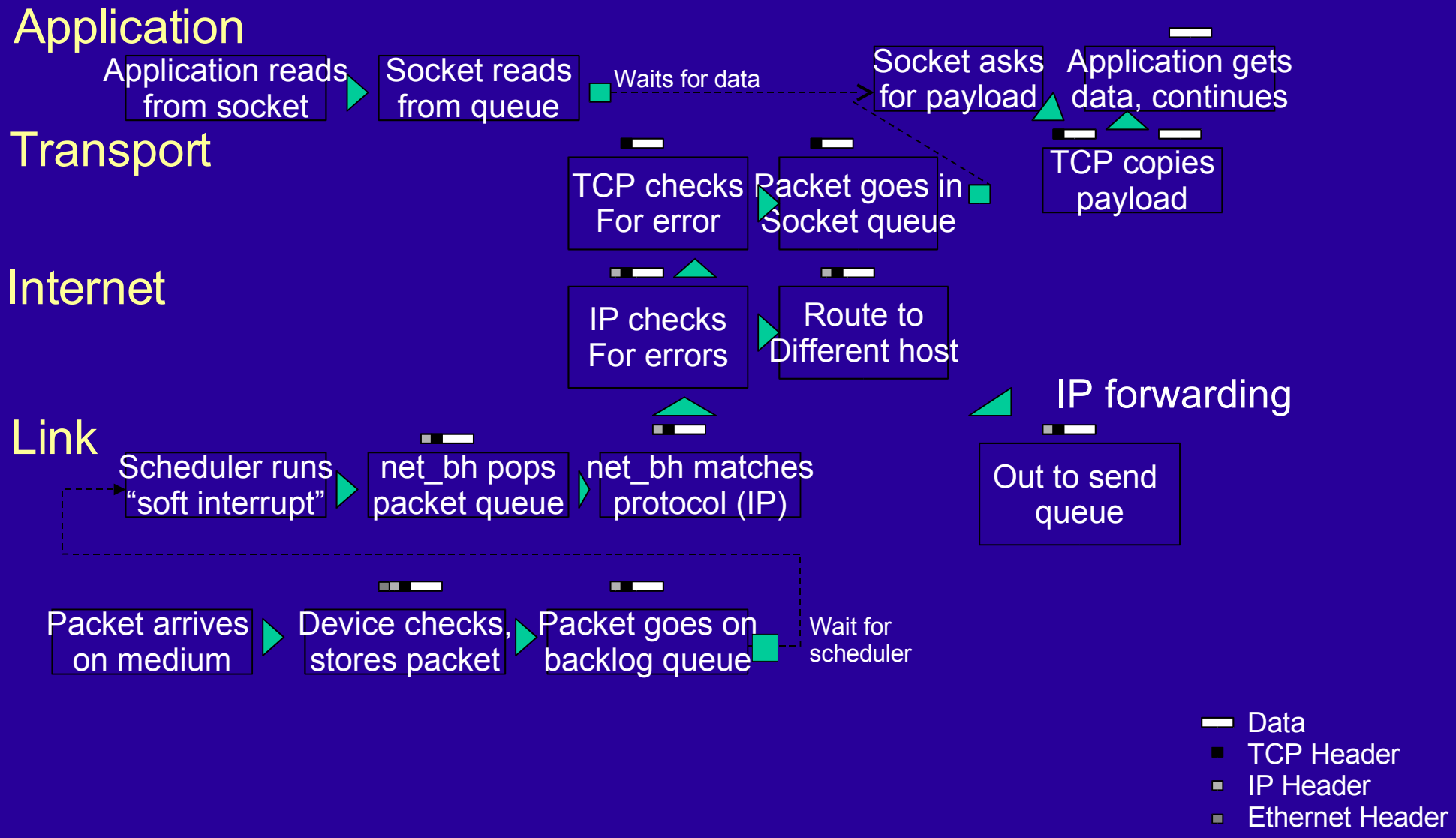
Sending a packet



Back trace (sending a packet)

dev_queue_xmit	→	skb freed
...		
ip_finish_output2	→	skb->hh (hardware header) determines if packet is ipip
...		
ip_output	→	Increment's SNMP stats
ip_queue_xmit2	→	Adds IP checksum; sets the sk peer and IP ID field.
ip_queue_xmit	→	Rt is copied to skb's dst_entry; IP header is built
tcp_trasmit_skb	→	Tcp header is built; tcp_option is built/updated; adds TCP checksum;l sets INET sock to skb->sk
tcp_connect	→	Sets the window and populates the tcp_option (init values)
tcp_v4_connect	→	INET socket's destination IP/port are set. Dst entry is created and set in INET socket.
inet_stream_connect	→	Marks the inet sock state. At this point, INET socket is sent apart from the BSD socket
sys_connect	→	System call. Looks up BSD Socket.
sys_socket_call	→	Copies info from user level

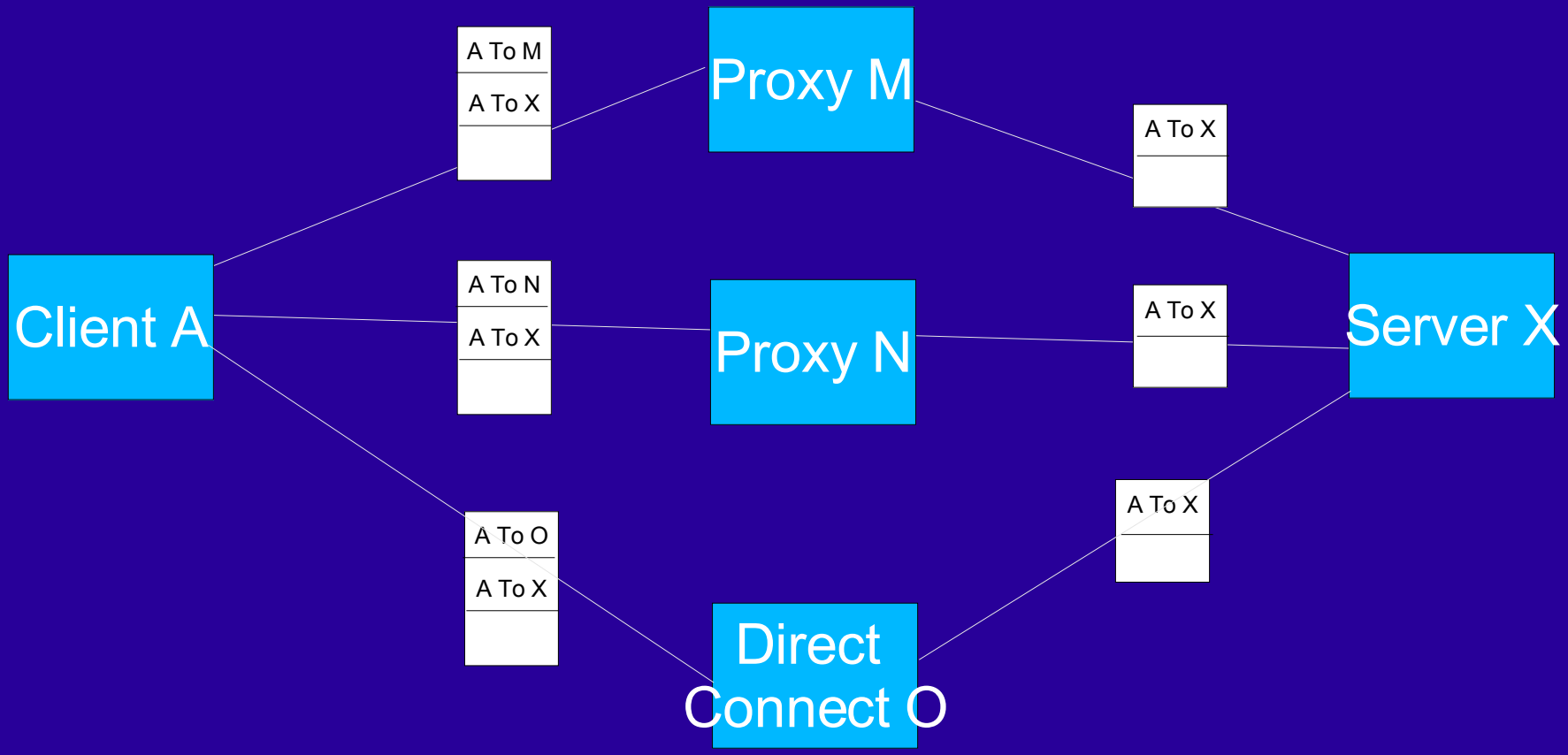
Receiving a packet



Enhanced TCP

- IP tunneling creates two IP headers on one packet. When the gateway or proxy server receives the packet, it strips off the first IP header and sends it to the back-end destination.
- To create a multi-pass routing using TCP, we use IP tunneling to trick the end server into thinking the packet comes through the same route.

Enhanced TCP (cont.)



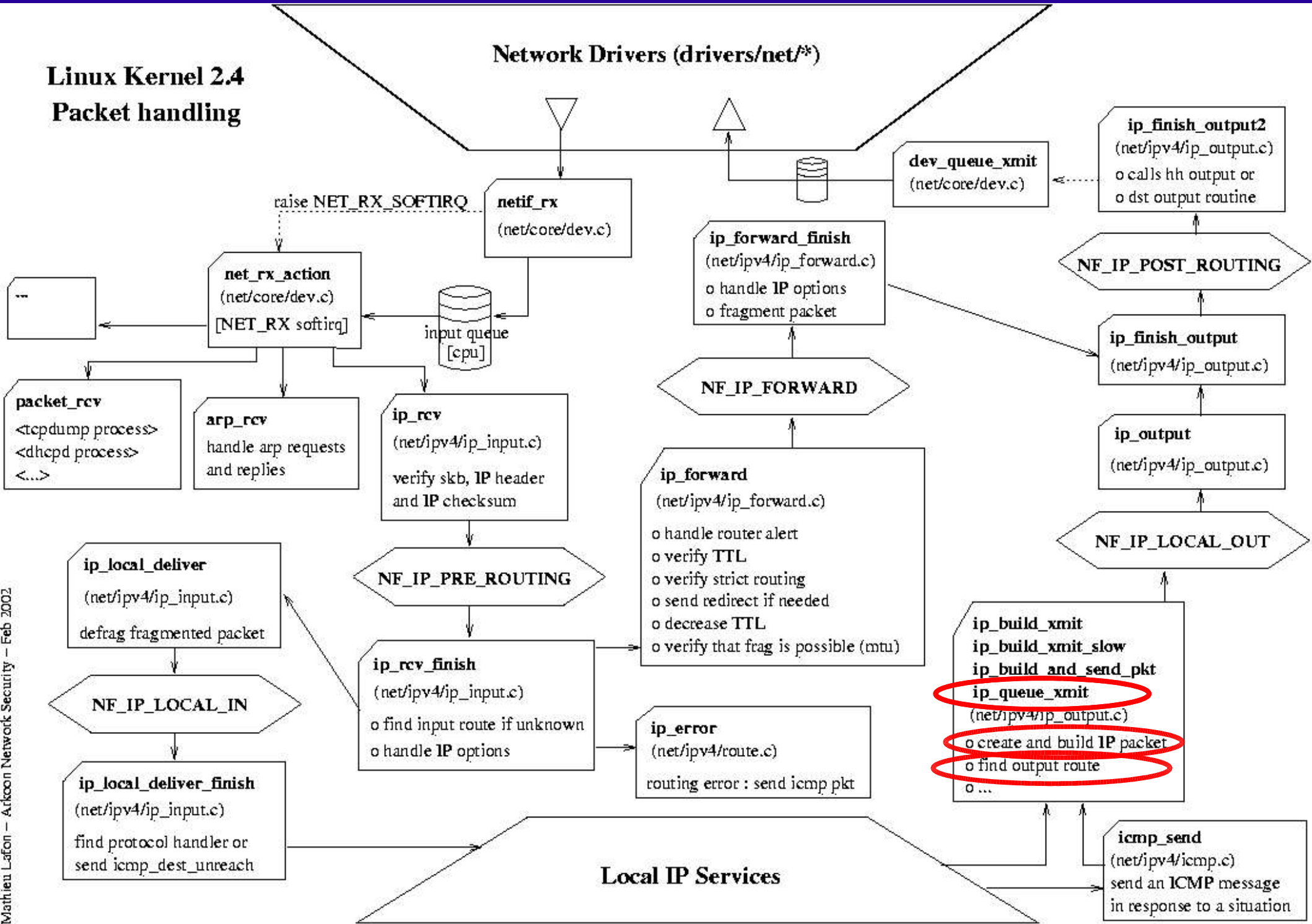
Solution

- Setup the client to have two IP tunnels (tunl0 and tunl1). Write the code in the ip_queue_xmit and switch the dev on the sk_buff.
- ip_queue and ip_queue2 are the last places in the ip/tcp code before sending up to lower device levels. Changing the device to a tunnel oppose an ethernet will also change the functions called, meaning the sk_buff will travel through functions in the ipip.c adding the additional IP header.
- Yu Cai made this break through.

Demo

Picture of the packet through the source code.

- We will insert break points at:
  1) tcp_v4_connect – IP and port are populated dst_entry is created
  2) tcp_connect – sets initial window and tcp_option
  3) tcp_transmit_skb – tcp header built
  4) ip_queue_xmit – IP header is built
  5) ip_queue_xmit2 – sets sk peer and IP's ID and checksum field


Linux Kernel 2.4
Packet handling
Network Drivers (drivers/net/*)
dev_queue_xmit
(net/core/dev.c)
ip_finish_output2
(net/ipv4/ip_output.c)
o calls hh output or
o dst output routine
raise NET_RX_SOFTIRQ
netif_rx
(net/core/dev.c)
ip_forward_finish
(net/ipv4/ip_forward.c)
o handle IP options
o fragment packet
NF_IP_POST_ROUTING
net_rx_action
(net/core/dev.c)
[NET_RX softirq]
input queue
[cpu]
ip_finish_output
(net/ipv4/ip_output.c)
packet_rcv
<tcpdump process>
<dhcpd process>
<...>
arp_rcv
handle arp requests
and replies
ip_rcv
(net/ipv4/ip_input.c)
verify skb, IP header
and IP checksum
NF_IP_FORWARD
ip_output
(net/ipv4/ip_output.c)
ip_forward
(net/ipv4/ip_forward.c)
o handle router alert
o verify TTL
o verify strict routing
o send redirect if needed
o decrease TTL
o verify that frag is possible (mtu)
NF_IP_LOCAL_OUT
ip_local_deliver
(net/ipv4/ip_input.c)
defrag fragmented packet
NF_IP_PRE_ROUTING
ip_build_xmit
ip_build_xmit_slow
ip_build_and_send_pkt
ip_queue_xmit
(net/ipv4/ip_output.c)
o create and build IP packet
o find output route
o ...
NF_IP_LOCAL_IN
ip_rcv_finish
(net/ipv4/ip_input.c)
o find input route if unknown
o handle IP options
ip_error
(net/ipv4/route.c)
routing error : send icmp pkt
ip_local_deliver_finish
(net/ipv4/ip_input.c)
find protocol handler or
send icmp_dest_unreach
Local IP Services
icmp_send
(net/ipv4/icmp.c)
send an ICMP message
in response to a situation
Mathieu Lafon – Arkoon Network Security – Feb 2002

References

- http://www.linux-mag.com/2001-04/user_mode_01.html An extremely helpful article about setting up UML with a step by step example
- http://user-mode-linux.sourceforge.org The user mode linux webpage
- http://kernelnewbies.org/documents/ipnetworking/linuxipnetworking.html An extremely valuable document about the linux IP networking layer
- Linux IP Neworking – A guid to implementation and modification of the Linux Protocol Stack – Glenn Herrin
- Interworking with TCP/IP – Douglas Comer