NetBSD Kernel Topics:
IP Processing
mbuf structure
Loadable Kernel Modules
Interrupts
Miscellaneous
Goals

• **IP**
  – Understand the structure of IP processing in NetBSD
  – Understand the IP packet format
  – Become familiar with IP packet field access
    » accessing src and dst addresses, etc…

• **mbufs**
  – Understand the mbuf structure
  – Become familiar with mbuf access routines
    » packet size
    » data location

• **Stevens TCP/IP books**
  – impress upon you the importance of these ref. books
What we are NOT going to talk about

- **IPv6**
  - Everything we talk about will be in terms of IPv4
  - Router Plugins also supports IPv6
  - The code can be confusing
    » use IPv6 structures since they are superset of IPv4

- **IP Options**

- **Fast Forward Path**

- **Details of interaction between device drivers and IP**
  - actually, we’ll talk about some of this…

- **Packet Scheduler**

- **Routing protocols**

- **daemons: routed, rsvpd, …**


NetBSD Kernel Topics:

IP Processing
mbuf structure
Loadable Kernel Modules
Interrupts
Miscellaneous
Vocabulary: Terms Used in Stevens

- **message**: Transport protocol data given to IP
- **datagram**: message + IP header
- **fragment**: if datagram is too large for network: IP splits it. Each fragment contains its own IP header
- **packet**: fragment or datagram small enough for network
- **frame**: packet + data-link layer header
Protocol messages, IP Packets, Fragments, ...

Derived from Figure 8.7 from Wright/Stevens, Vol. 2
IP Packet Format

IP Header (with no Options fields)

Transport Protocol (e.g. TCP, UDP) Header

Packet Payload
IPv4 Forwarding in the NetBSD Kernel (APIC)
User IP in the NetBSD Kernel

User Program

socket layer

tcp/udp_input → tcp/udp_output

ipintr

ipintrq:

ip_forward → ip_output

atmc_input

atmc_output

apic_isr

apic_intr

apic_devoutput

apic_start

User Program

Waits in Socket Buffer for User Program to pick up

Software Interrupt

input packets

interrupts

output packets

socket layer

TCP/UDP Input

TCP/UDP Output

Socket Layer
IP Packet Handling in the CB Kernel

Where is the route lookup done?

User Space

SW Interrupt

HW Interrupt

Network

Kernel

User Space

Socket layer

Input packets

Interrupts

Output packets

SW Interrupt

HW Interrupt

NETWORK

Kernel

User Space

Where are the gates?

This is the gate that you will use.

Here are the rest of the gates?

Where is the route lookup done?
Important Source Files (usr/src/sys/netinet/ip*)

• ip.h
  – struct ip { }
  – define’s

• ip_input.c
  – ipintr() (This is ip_input routine)
  – ip_forward()

• ip_output.c
  – ip_output()
IP Packet Handling in CB Kernel with APIC

• Device Driver -- Device Dependent
  – `apic_intr()`
    » read INTR_ACK: 0 => not for us, return
    » `apic_isr()`
  – `apic_isr()`
    » read NOTIFY_LIST, next rcv channel needing attention
      • Process descriptors for RCV channel
        – swap words in each received mbuf (APIC BUG)
        – **link mbufs until we find end of frame**
        – verify CRC (just check flag set by APIC)
        – `atmc_input(packet)`
      – repeat
      • repeat

January 9, 2001  Router Plugins (Crossbow)
IP Packet Handling in CB Kernel (continued)

• Device Driver - Dev. Independent `atmc_input()`:
  – LLC/SNAP processing (if needed)
    » extract type (IP/RATM/…)
  – AAL5 processing
    » extract length
    » strip trailer
  – `aiu_getafix` (packet) (flow table will be discussed later)
    » `aiu_getafix` stores a ptr to FTE in packet
  – Enqueue in IP Queue
IP Packet Handling in CB Kernel (continued)

• IP Input -- \textit{ipintr}():
  – Get next packet from IP Queue
  – Do some basic checks, header, length, checksum…
  – process IP options
  – \texttt{aiu\_dgate(&m,2)} Router Plugins Dynamic Gate #2
  – if packet is not for us and we can forward it:
    » forward packet \texttt{ip\_forward()} -- upcoming slide...
    » return
  – \texttt{aiu\_dgate(&m,4)} Router Plugins Dynamic Gate #4
  – protocol specific input routine e.g. \texttt{tcp\_input}()
IP Packet Handling in CB Kernel (continued)

- IP Forwarding -- *ip_forward() :
  - decide if we need to send any redirects to sender
  - *aiu_dgate(&m,5) Router Plugins Dynamic Gate #5
  - route lookup
  - *ip_output()
IP Packet Handling in CB Kernel (continued)

- **IP Output -- *ip_output()*
  - *aiu_dgate(&m,3)* Router Plugins Dynamic Gate #3
  - if (no route yet)
    » *get route*
  - check for special processing
    » ANEP Options
      • *anep_output()*
        - if DAN then *afd_handle_dan_packet() : ACTIVE PROCESSING*
  - send on interface
**IP Packet Handling in CB Kernel (continued)**

- **Device Driver Device Indep.** -- `atmc_output()`:
  - LLC/SNAP processing
  - AAL5 processing (if needed)
  - if `packet_scheduling`: enqueue for PS
  - if `!packet_scheduling`: send to `devoutput()`

- **Device Driver Device Dep.** -- `apic_devoutput()`
  - configure an APIC descriptor for each `mbuf` in packet
  - Resume APIC TX channel
Note about Addresses

• The address structures in the Crossbow Kernel are IPv6
  – IPv6 address structures are a superset of IPv4
  – IPv6 address are 128 bits
• We will be using IPv4 addresses
  – e.g. 192.168.5.2
  – IPv4 addresses are 32 bits
• Notation for using IPv4 address in IPv6:
  – Use double colon before address:
    ::192.168.5.2
  – Double colon tells the utilities to set everything to the left to 0’s
NetBSD Kernel Topics

IP Processing
mbuf structure
Loadable Kernel Modules
Interrupts
Miscellaneous
The mbuf Data Structure

```
mbuf{}
  m_next
  m_nextpkt
  m_len
  0-108
  m_data
  m_type
  MT_xxx
  M_PKTHDR
  M_EXT
  M_PKTHDR
  M_EXT
  m_flags
  0
  M_PKTHDR
  m_pkthdr.len
  m_pkthdr.rcvif
  m_ext.ext_buf
  m_ext.ext_free
  m_ext.ext_size

Normal mbuf
  Normal mbuf with Packet Hdr
  Cluster mbuf
  Cluster mbuf with Packet Hdr

Normal mbuf
  Normal mbuf with Packet Hdr
  Cluster mbuf
  Cluster mbuf with Packet Hdr
```

2048-byte cluster

2048-byte cluster
• **Highlight**: (add Stevens’ like figures…)
  - m_next vs m_nextpkt (figure 2.2 page 35)
    » m_next: points to next mbuf in THIS packet
    » m_nextpkt: points to the first mbuf of the NEXT packet
  - m_len vs. m_pkthdr.len
  - m_ext.ext_buf vs. m_data
  - Which mbuf structure we use with APIC

• **Example**:
  - do m_pullup
  - get data pointer
  - print out src addr, dst addr
  - get Transport header pointer
  - print out dst port

*Sorry: This was just one of my notes slides that should have been removed before printing*
The mbuf Data Structure

Sorry: This was just one of my notes slides that should have been removed before printing.
The mbuf Data Structure (example)
The mbuf Data Structure (example, continued)

mbuf{}

next mbuf in chain

mbuf{}

NULL

54

MT_HEADER

M_PKTHDR

1514

NULL

mbuf packet header

Ethernet header, IP header, TCP header

mbuf{}

NULL

m_next

m_nextpkt

m_len

m_data

m_type

m_flags

m_pkthdr.len

m_pkthdr.rcvif

2048-byte cluster

Ethernet header, IP header, TCP header

1460 bytes of data

m_ext.ext_buf

m_ext.ext_size

2048

1460 bytes of data

m_ext.ext_free
NetBSD Kernel Topics

IP Processing
mbuf structure

Loadable Kernel Modules
Interrupts
Miscellaneous
Loadable Kernel Modules

- Mechanism in NetBSD to dynamically load code into running kernel
- NetBSD System Utilities:
  - modload(8)
  - modunload(8)
  - modstat(8)
- Module Types supported:
  - System Call modules
    » When unloaded, returns to original system call
    » Any system call can be replace
    » Take care when replacing ioctl(2) since LKM uses it to load/unload modules!!
  - Virtual File System modules
  - Device Driver modules: Block and character device drivers
  - Execution Interpreters: For binaries not normally usable by OS
  - Miscellaneous modules
    » No defined interfaces
    » Up to user/developer to provide hooks to get to the code
    » This is what Router Plugins uses
Loadable Kernel Modules: modload

- Open /dev/lkm
  - ioctl’s will be performed on the open file descriptor
- Prelink module, open it and calculate size info
  - system("ld -A /netbsd -e _<entry> -o <outfile> -T <addr=0> <module>");
- Reserve Kernel memory:
  - ioctl(lkm_fd, LMRESERV, size_info)
    - returns kernel load address
- Relink at kernel load address
- Open relinked module and load it into kernel
  - ioctl(lkm_fd, LMLOADBUF, ...)
- Adjusting symbol table entry pointers and load it.
- sync();
- Call the module’s entry function to test it.
- Post-install if called for...
  - ioctl(lkm_fd, LMSTAT, ...)
NetBSD Kernel Topics

IP Processing
mbuf structure
Loadable Kernel Modules
Interrupts
Miscellaneous
Interrupts

- **8 levels**
  - `spl0`  normal
  - `splsoftclock`  low-priority clock processing
  - `splnet`  network protocol processing
  - `splttty`  terminal I/O
  - `splbio`  disk and tape I/O
  - `splimp`  network device I/O
  - `splclock`  high-priority clock processing
  - `splhigh`  all interrupts blocked

- **Typical usage:**
  ```c
  int i;
  i = splimp();
  [do something at device I/O level]
  splx(i);
  ```
NetBSD Kernel Topics

IP Processing
mbuf structure
Loadable Kernel Modules
Interrupts
Miscellaneous
Other kernel details you must know

• No floating point
• No `#include` in header files
• More caveats and info from Fred later…