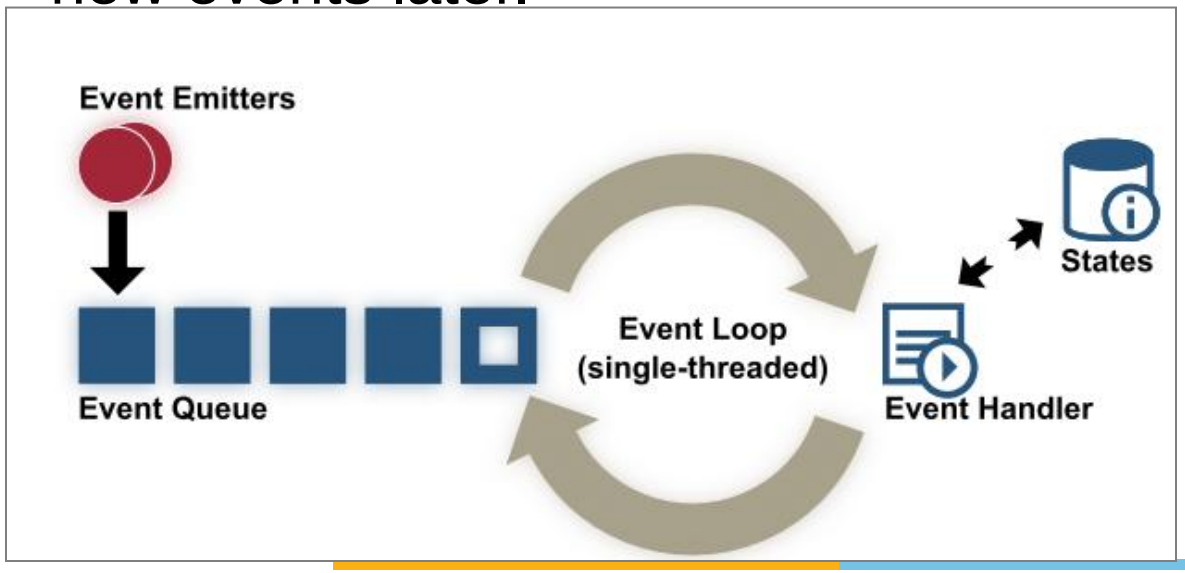
Network Programming

BITS Pilani
Pilani Campus

K Hari Babu
Department of Computer Science & Information Systems

Event Driven Architectures

- A single threaded event loop consumes event after event from the queue and sequentially executes associated event handler code.
- New events are emitted by external sources such as socket or file I/O notifications.
- Event handlers trigger I/O actions that eventually result in new events later.




```
while (1) {
events = getEvents();
for (e in events)
processEvent(e);
}
```

Event Driven Architectures

- Thread is very similar to a scheduler, multiplexing multiple connections to a single flow of execution.
- The states of the connections are organized in appropriate data structures— using finite state machines etc.
- Event-driven server architectures is dependent on the availability of asynchronous/non-blocking I/O operations at OS level.

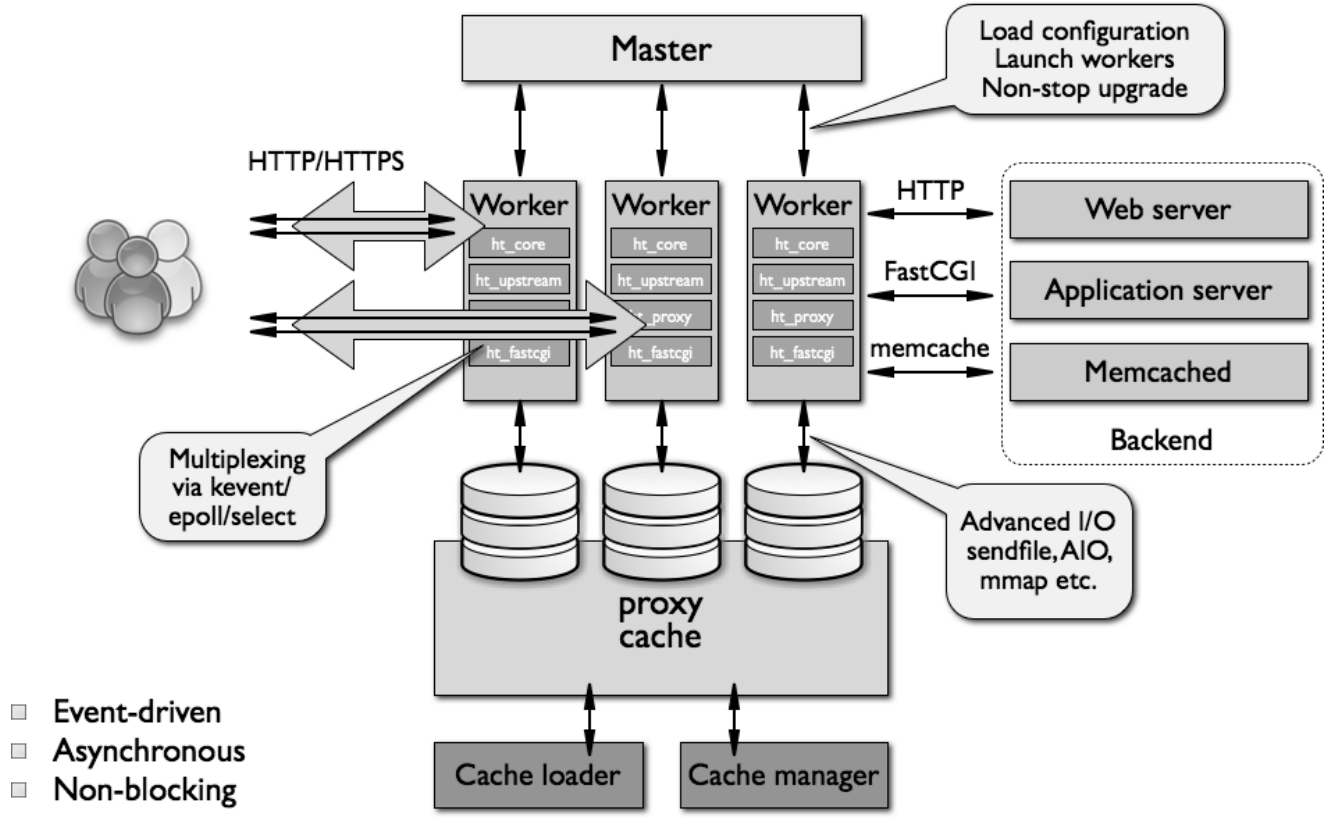
	Blocking	Non-blocking
Synchronous	read/write	read/write using O_NONBLOCK
Asynchronous	I/O multiplexing (select/poll/epoll)	AIO

IO Notification

- IO Multiplexing with Threads
  - Availability of data
  - Copying through a helper thread or process.
  - Notification through a call back function.
- AIO
  - Completion of data.
  - Call back functions which modify the state of the connection and generate events.

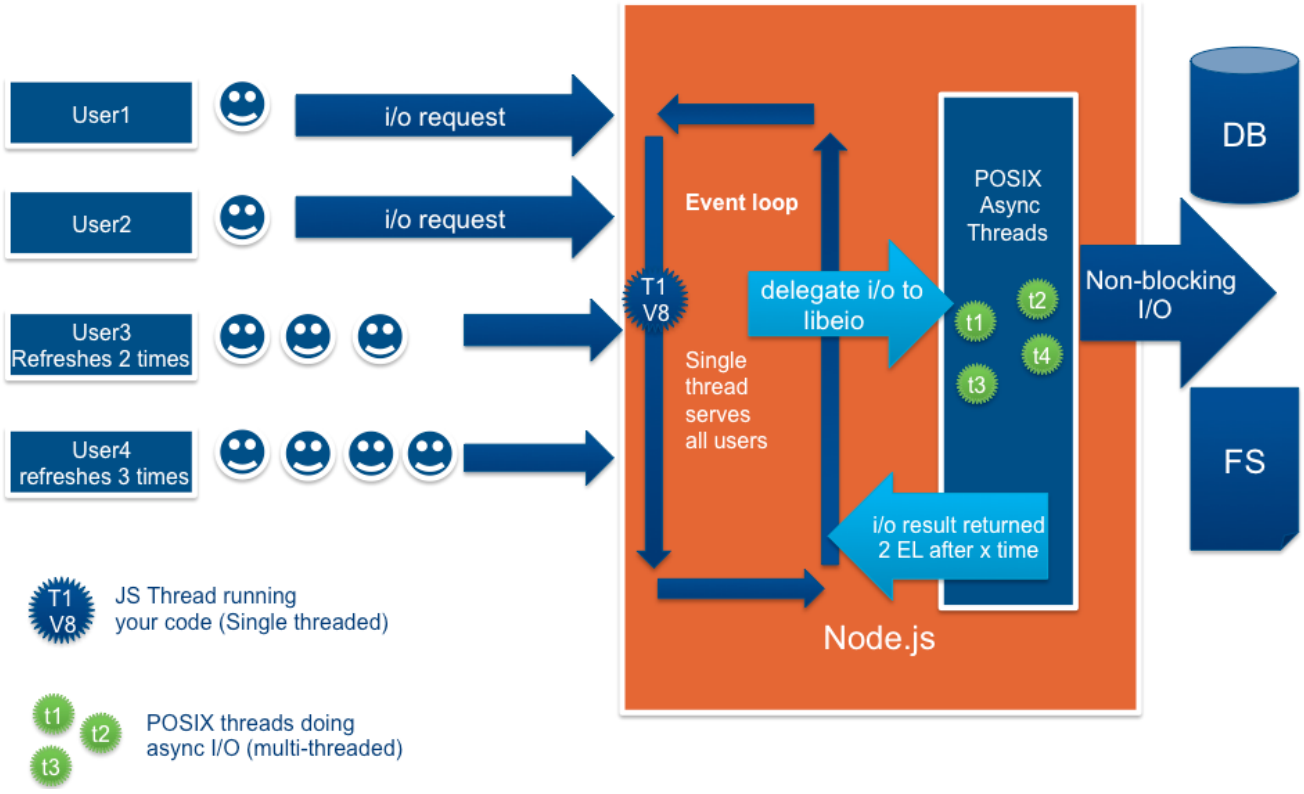
Nginx

- Multiple worker processes
- Shared listening socket
- Uses Asynchronous callback functions.



Node.js

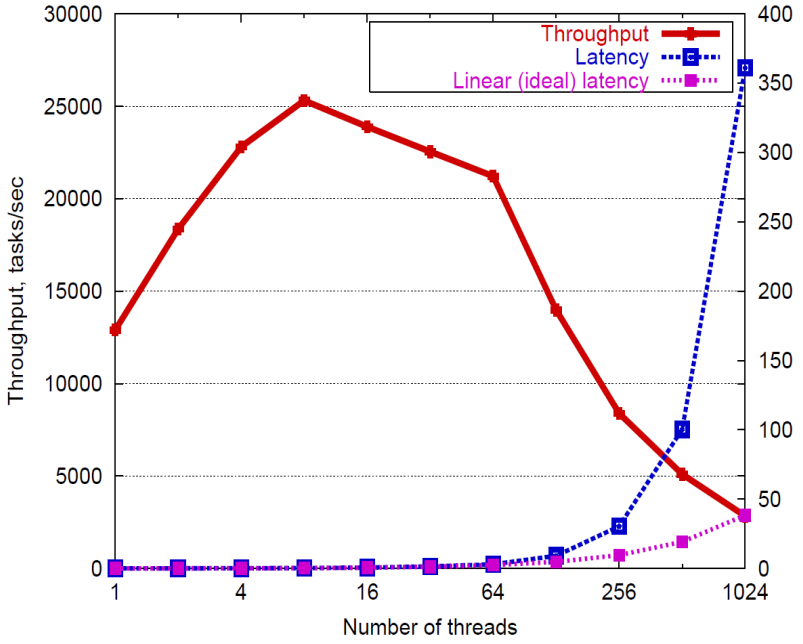
- Single thread
- Asynchronous callback functions



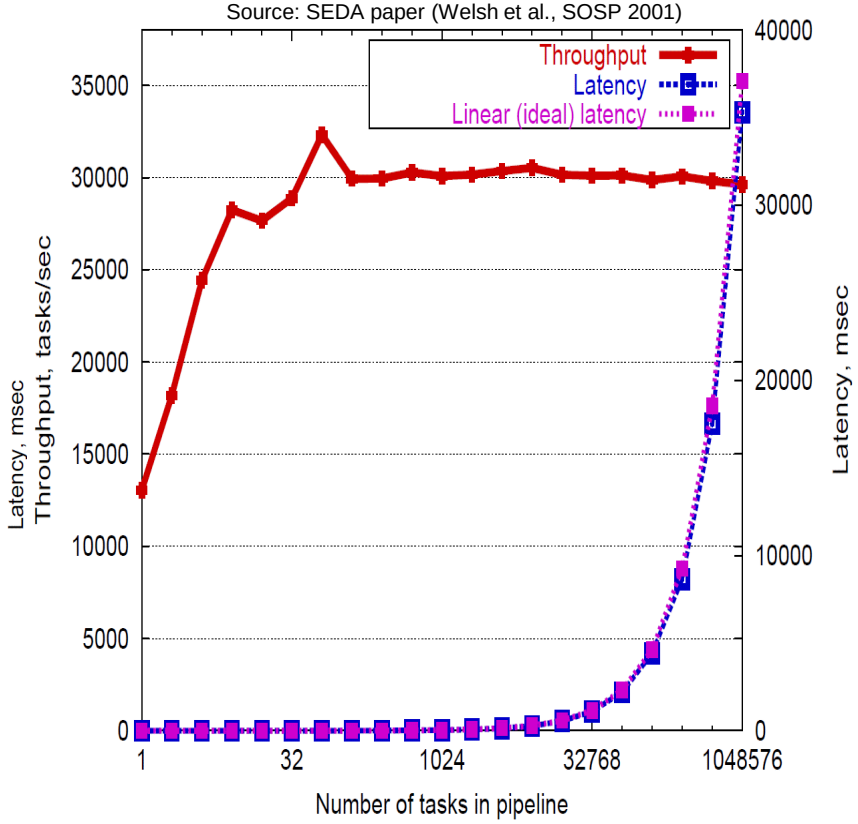
Event Driven Architectures

- Two patterns that involve event demultiplexors are called Reactor and Proactor.
  - The Reactor patterns involve synchronous I/O, whereas the Proactor pattern involves asynchronous I/O.
    - In Reactor, the event demultiplexor waits for events that indicate when a file descriptor or socket is ready for a read or write operation.
    - The demultiplexor passes this event to the appropriate handler, which is responsible for performing the actual read or write.
  - Proactor pattern, the event demultiplexor initiates asynchronous read and write operations.
    - The event demultiplexor waits for events that indicate the completion of the I/O operation, and forwards those events to the appropriate handlers.

Threads vs events



Source: SEDA paper (Welsh et al., SOSP 2001)



- No throughput degradation under load
- Peak throughput is higher

innovate achieve lead

BITS Pilani

Pilani Campus

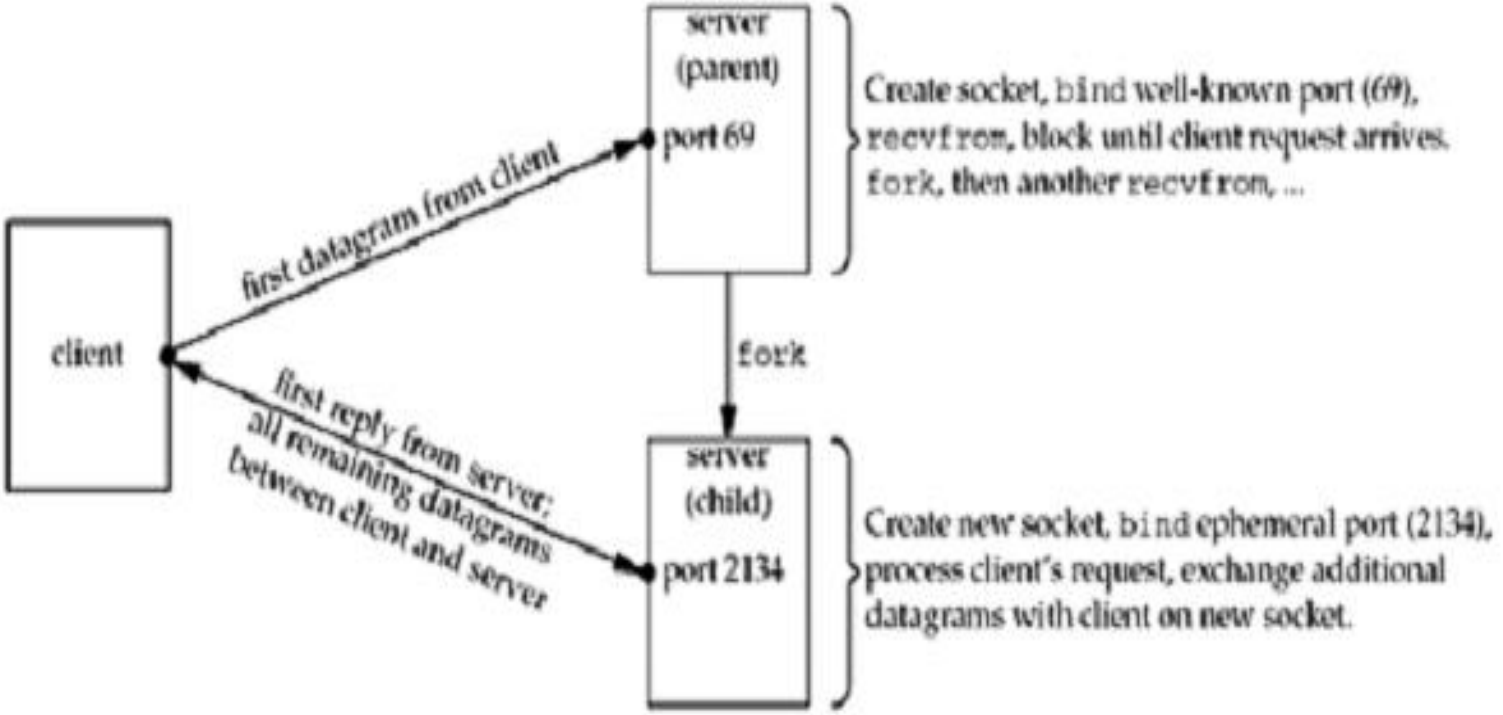
Concurrency UDP Servers

T1: ch 22.7

Concurrent UDP Servers

- Two different types of servers:
- First is a simple UDP server that reads a client request, sends a reply, and is then finished with the client
  - Concurrency: fork a child and let it handle the request
- Second is a UDP server that exchanges multiple datagrams with the client. Extended conersation.
  - Create a new socket for each client, bind an ephemeral port to that socket, and use that socket for all its replies.
  - The client looks at the port number of the server's first reply and send subsequent datagrams to that port.

Concurrency in UDP for Extended Conversations



innovate achieve lead

BITS Pilani

Pilani Campus

Daemons

R1: ch 37

Daemons

- A daemon is a process with the following characteristics:
  - It is long-lived. Often, a daemon is created at system startup and runs until the system is shut down.
  - It runs in the background and has no controlling terminal. The lack of a controlling terminal ensures that the kernel never automatically generates any job-control or terminal-related signals (such as SIGINT , SIGTSTP , and SIGHUP) for a daemon.
- Examples
  - cron: a daemon that executes commands at a scheduled time.
  - sshd: the secure shell daemon, which permits logins from remote hosts using a secure communications protocol.
  - httpd: the HTTP server daemon (Apache), which serves web pages.
  - inetd: the Internet superserver daemon which listens for incoming network connections on specified TCP/IP ports and launches appropriate server programs to handle these connections.

How to create a Daemon?

innovate achieve lead

```
/* daemons/become_daemon.h*/
#ifndef BECOME_DAEMON_H
#define BECOME_DAEMON_H
/* Bit-mask values for 'flags' argument of becomeDaemon() */
#define BD_NO_CHDIR           01    /* Don't chdir("/") */
#define BD_NO_CLOSE_FILES     02    /* Don't close all open files */
#define BD_NO_REOPEN_STD_FDS  04    /* Don't reopen stdin, stdout, and
                                       stderr to /dev/null */
#define BD_NO_UMASK0         010    /* Don't do a umask(0) */
#define BD_MAX_CLOSE  8192          /* Maximum file descriptors to close if
                                       sysconf(_SC_OPEN_MAX) is indeterminate */
int becomeDaemon(int flags);
#endif
```

How to create a Daemon?

innovate achieve lead

```
1  int /* Returns 0 on success, -1 on error */
2  becomeDaemon(int flags)
3  {
4      int maxfd, fd;
5      switch (fork()) {   /* Become background process */
6      case -1: return -1; /*ensure not a proces sgroup leader*/
7      case 0:  break;     /* Child falls through... */
8      default: _exit(EXIT_SUCCESS); /* while parent terminates */
9      }
10     if (setsid() == -1) /* Become leader of new session */
11         return -1;
12     switch (fork()) {   /* Ensure we are not session leader */
13     case -1: return -1;
14     case 0:  break;
15     default: _exit(EXIT_SUCCESS);
16     }
```

How to create a Daemon?

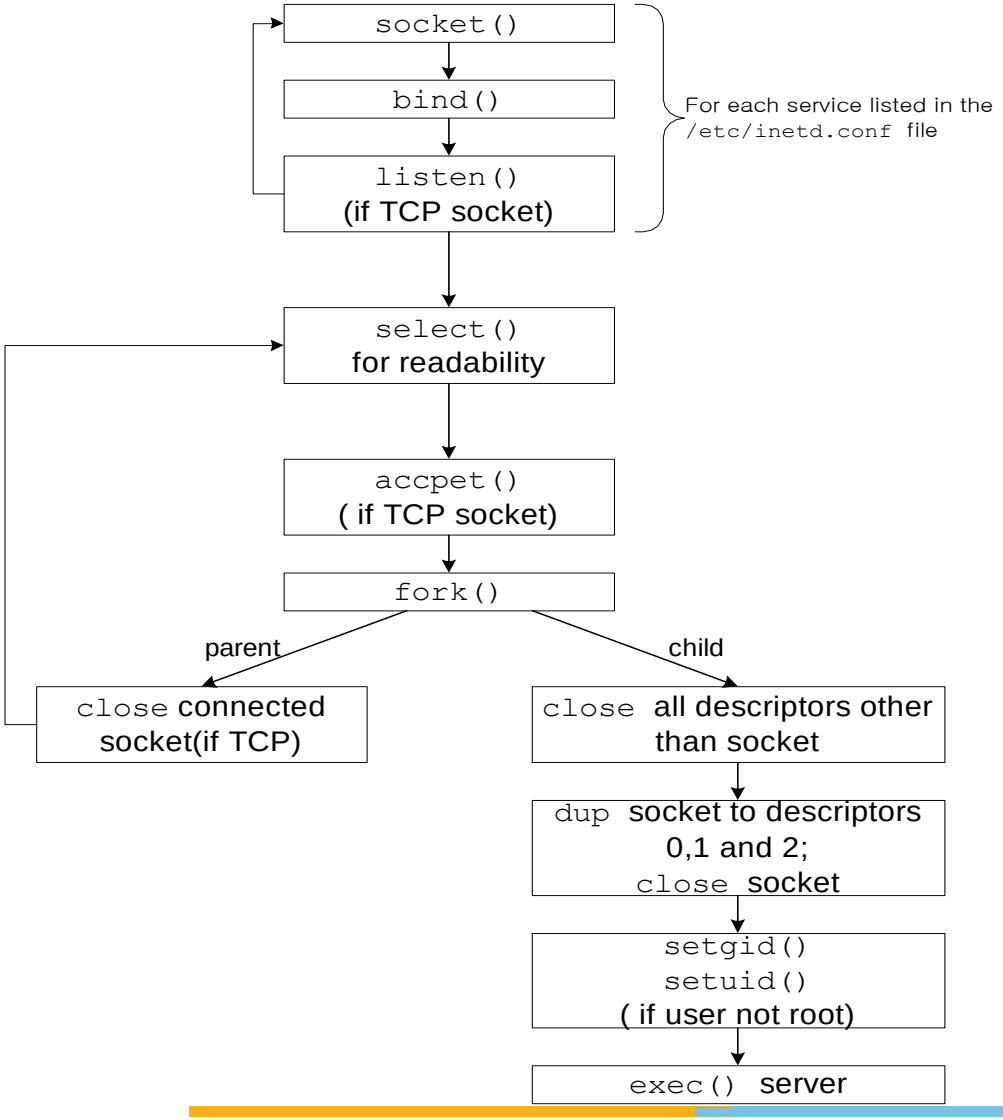
innovate achieve lead

```
15      default: _exit(EXIT_SUCCESS);
16      }
17       if (!(flags & BD_NO_UMASK0))
18          umask(0);       /* Clear file mode creation mask */
19      if (!(flags & BD_NO_CHDIR))
20          chdir("/");  /* Change to root directory */
21      if (!(flags & BD_NO_CLOSE_FILES)) { /* Close all open files */
22          maxfd = sysconf(_SC_OPEN_MAX);
23          if (maxfd == -1)/* Limit is indeterminate... */
24              maxfd = BD_MAX_CLOSE; /* so take a guess */
25          for (fd = 0; fd < maxfd; fd++)
26              close(fd);
27      }
28      if (!(flags & BD_NO_REOPEN_STD_FDS)) {
29          close(STDIN_FILENO);/* Reopen standard fd's to /dev/null */
30          fd = open("/dev/null", O_RDWR);
31          if (fd != STDIN_FILENO)  /* 'fd' should be 0 */
32              return -1;
33          if (dup2(STDIN_FILENO, STDOUT_FILENO) != STDOUT_FILENO)
34              return -1;
35          if (dup2(STDIN_FILENO, STDERR_FILENO) != STDERR_FILENO)
36              return -1;
37      }
38      return 0;
39  }
```

Inted Super Server

- The inetd daemon is designed to eliminate the need to run large numbers of infrequently used servers. Using inetd provides two main benefits:
  - Instead of running a separate daemon for each service, the inetd daemon monitors a specified set of socket ports and starts other servers as required. Thus, the number of processes running on the system is reduced.
  - The programming of the servers started by inetd is simplified, because inetd performs several of the steps that are commonly required by all network servers on startup.

Inted Super Server



`inetd` service specification

- For each service, `inetd` needs to know:
  - the socket type and transport protocol
  - wait/nowait flag.
  - login name the process should run as.
  - pathname of real server program.
  - command line arguments to server program.
- Servers that are expected to deal with frequent requests are typically *not* run from inetd
  - mail, web, NFS.

/etc/inetd.conf

innovate achieve lead

```
1  # Syntax for socket-based Internet services:
2  #  <service_name> <socket_type> <proto> <flags> <user> <server_pathname> <args>
3  #
4  # comments start with #
5  echo      stream  tcp  nowait  root  internal
6  echo      dgram   udp  wait    root  internal
7  chargen   stream  tcp  nowait  root  internal
8  chargen   dgram   udp  wait    root  internal
9  ftp       stream  tcp  nowait  root  /usr/sbin/ftpd ftpd -l
10 telnet    stream  tcp  nowait  root  /usr/sbin/telnetd telnetd
11 finger    stream  tcp  nowait  root  /usr/sbin/fingerd fingerd
12 # Authentication
13 auth      stream  tcp  nowait  nobody  /usr/sbin/in.identd in.identd -l -e -o
14 # TFTP
15 tftp       dgram   udp   wait    root   /usr/sbin/tftpd tftpd -s /tftpboot
```

wait/nowait

- WAIT specifies that **`inetd`** should not look for new clients for the service until the child (the real server) has terminated.
- TCP servers usually specify **`nowait`** - this means inetd can start multiple copies of the TCP server program - providing concurrency
- Most UDP services run with **`inetd`** told to wait until the child server has died.

Server Loaded by Inetd

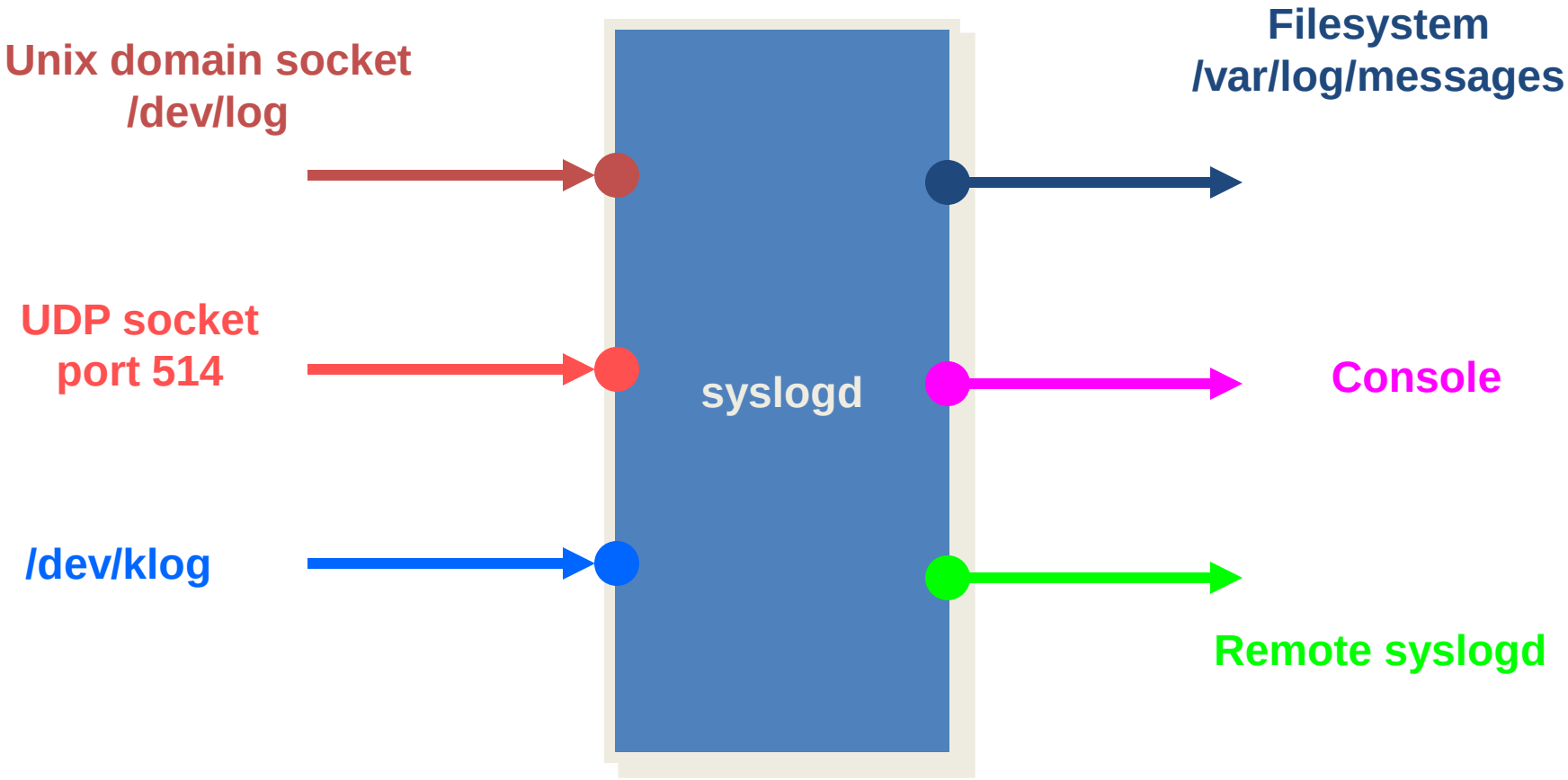
innovate achieve lead

```
1  /* daytime server*/
2  int
3   main(int argc, char **argv)
4   {
5       socklen_t len;
6       struct sockaddr *cliaddr;
7       char    buff[MAXLINE];
8       time_t  ticks;
9       daemon_inetd(argv[0], 0);
10      cliaddr = malloc(sizeof(struct sockaddr_storage));
11      len = sizeof(struct sockaddr_storage);
12      getpeername(0, cliaddr, &len);
13      err_msg("connection from %s", Sock_ntop(cliaddr, len));
14      ticks = time(NULL);
15      snprintf(buff, sizeof(buff), "%.24s\r\n", ctime(&ticks));
16      write(0, buff, strlen(buff));
17      close(0);                       /* close TCP connection */
18      exit(0);
19  }
```

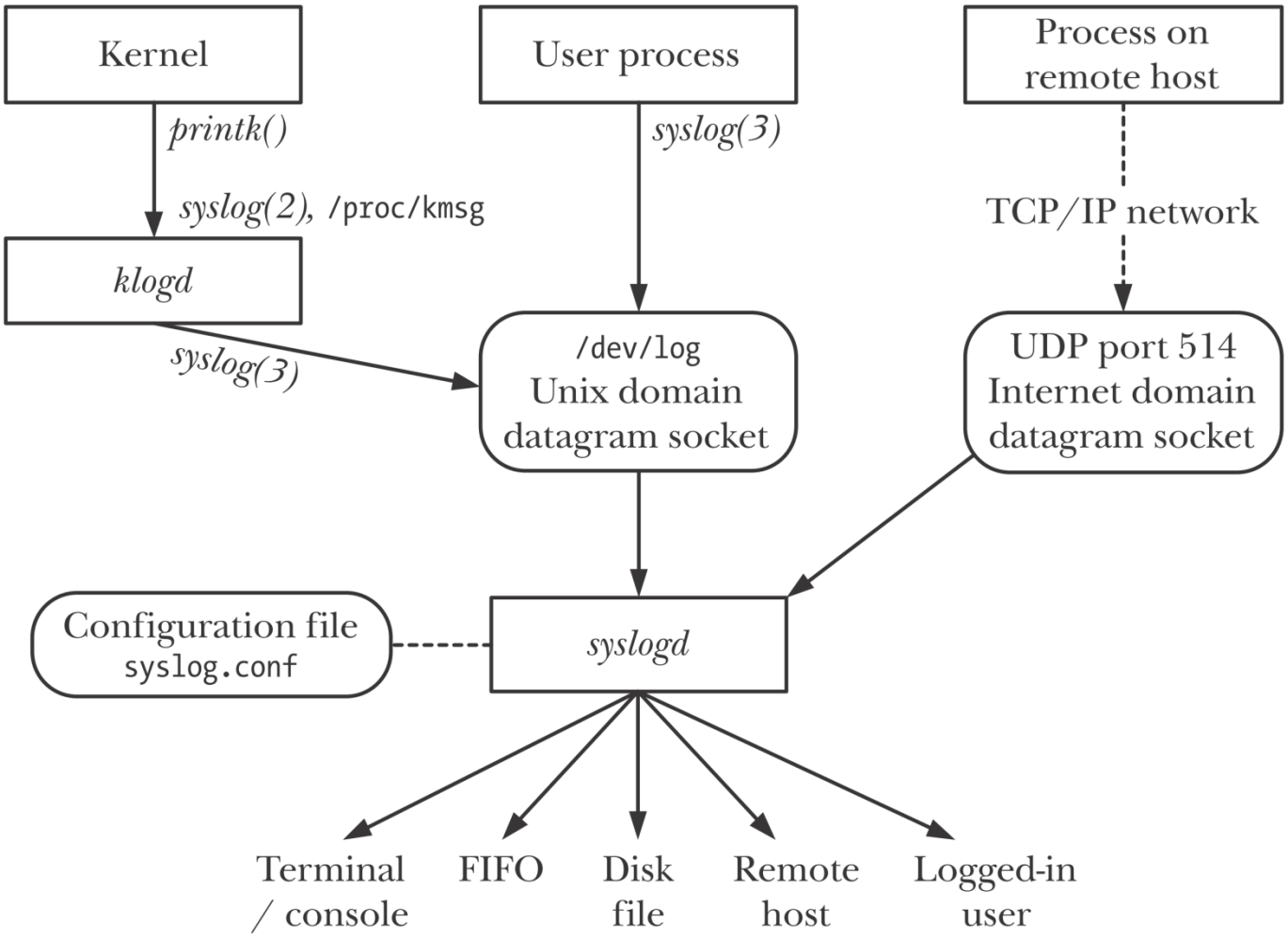
syslogd daemon

- Berkeley-derived implementation of `syslogd` perform the following actions upon startup.
  1. The configuration file is read, specifying what to do with each type of log message that the daemon can receive.
  2. A Unix domain socket is created and bound to the pathname `/var/run/log` (`/dev/log` on some system).
  3. A UDP socket is created and bound to port 514
  4. The pathname `/dev/klog` is opened. Any error messages from within the kernel appear as input on this device.
- We could send log messages to the `syslogd` daemon from our daemons by creating a Unix domain datagram socket and sending our messages to the pathname that the daemon has bound, but an easier interface is the `syslog` function.

syslogd


Unix domain socket
/dev/log
UDP socket
port 514
/dev/klog
syslogd
Filesystem
/var/log/messages
Console
Remote syslogd

syslogd


Kernel
User process
Process on remote host
printk()
syslog(3)
syslog(2), /proc/kmsg
TCP/IP network
klogd
syslog(3)
/dev/log
Unix domain
datagram socket
UDP port 514
Internet domain
datagram socket
Configuration file
syslog.conf
syslogd
Terminal / console
FIFO
Disk file
Remote host
Logged-in user

Syslogd

- Each message processed by syslogd has a number of attributes, including a *facility*, which specifies the type of program generating the message, and a *level*, which specifies the severity (priority) of the message.
- The syslogd daemon examines the facility and level of each message, and then passes it along to any of several possible destinations according to the dictates of an associated configuration file, /etc/syslog.conf .

- The syslog API consists of three main functions:
  - The openlog() function establishes default settings that apply to subsequent calls to syslog().
    - The use of openlog() is optional. If it is omitted, a connection to the logging facility is established with default settings on the first call to syslog().
  - The syslog() function logs a message.
  - The closelog() function is called after we have finished logging messages, to disestablish the connection with the log.

```
1  #include <syslog.h>
2  void openlog(const char * ident , int  log_options , int  facility );
```

  - The ident argument is a pointer to a string that is included in each message written by syslog();

syslog function


```
#include <syslog.h>
void syslog(int priority, const char *message, . . . );
```

- the *priority* argument is a combination of a level and a facility.
- The *message* is like a format string to `printf`, with the addition of a `%m` specification, which is replaced with the error message corresponding to the current value of `errno`.

```
Ex) Syslog(LOG_INFO|LOG_LOCAL2, "rename(%s, %s):
  %m",file1,file2);
```

Facility

Table 37-1: *facility* values for *openlog()* and the *priority* argument of *syslog()*

Value	Description	SUSv3
LOG_AUTH	Security and authorization messages (e.g., *su*)	•
LOG_AUTHPRIV	Private security and authorization messages	
LOG_CRON	Messages from the *cron* and *at* daemons	•
LOG_DAEMON	Messages from other system daemons	•
LOG_FTP	Messages from the *ftp* daemon (*ftpd*)	
LOG_KERN	Kernel messages (can't be generated from a user process)	•
LOG_LOCAL0	Reserved for local use (also LOG_LOCAL1 to LOG_LOCAL7)	•
LOG_LPR	Messages from the line printer system (*lpr*, *lpd*, *lpc*)	•
LOG_MAIL	Messages from the mail system	•
LOG_NEWS	Messages related to Usenet network news	•
LOG_SYSLOG	Internal messages from the *syslogd* daemon	
LOG_USER	Messages generated by user processes (default)	•
LOG_UUCP	Messages from the UUCP system	•

syslog function

level	value	description
LOG_EMERG	0	system is unusable (highest priority)
LOG_ALERT	1	action must be taken immediately
LOG_CRIT	2	critical conditions
LOG_ERR	3	error conditions
LOG_WARNING	4	warning conditions
LOG_NOTICE	5	normal but significant condition (default)
LOG_INFO	6	informational
LOG_DEBUG	7	debug-level message (lowest priority)

level of log message.

- Log message have a level between 0 and 7.

Outline

- Raw Sockets
  - Creation
  - Output
  - Input
  - Ping program
  - Traceroute
  - ICMP daemon
- Data Link Access
  - BSD Packet Filter (BPF)
  - DLPI
  - SOCK_PACKET/PF_PACKET
  - UDP Checksum
- Broadcasting
  - example
  - Race conditions when using signals
- Multicasting
  - Addresses
  - Multicasting on LAN
  - Multicasting on WAN
  - Source Specific Multicast
  - Multicast Socket Options
  - Simple Network Time Protocol

innovate achieve lead

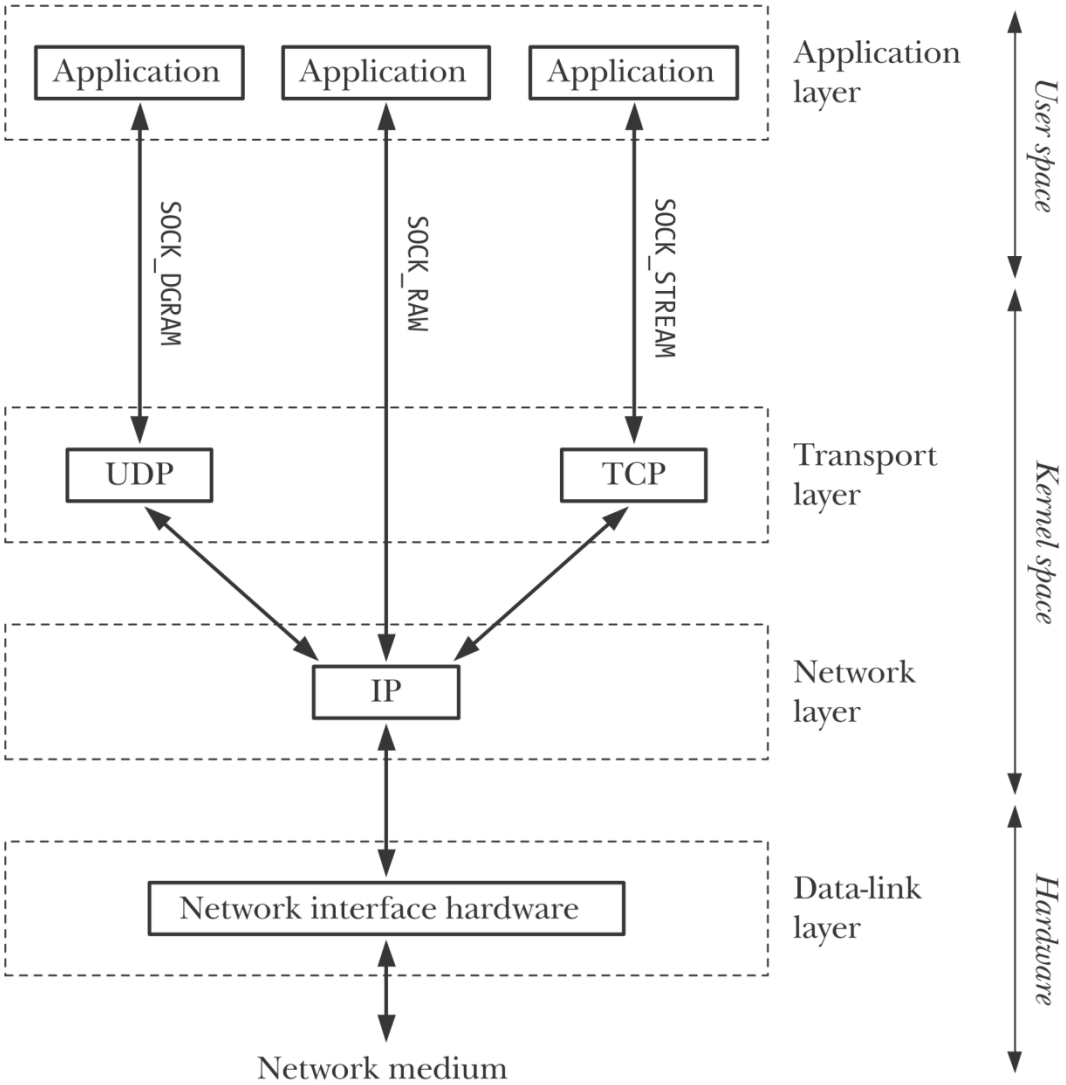
BITS Pilani

Pilani Campus

Raw Sockets

T1: Ch 28

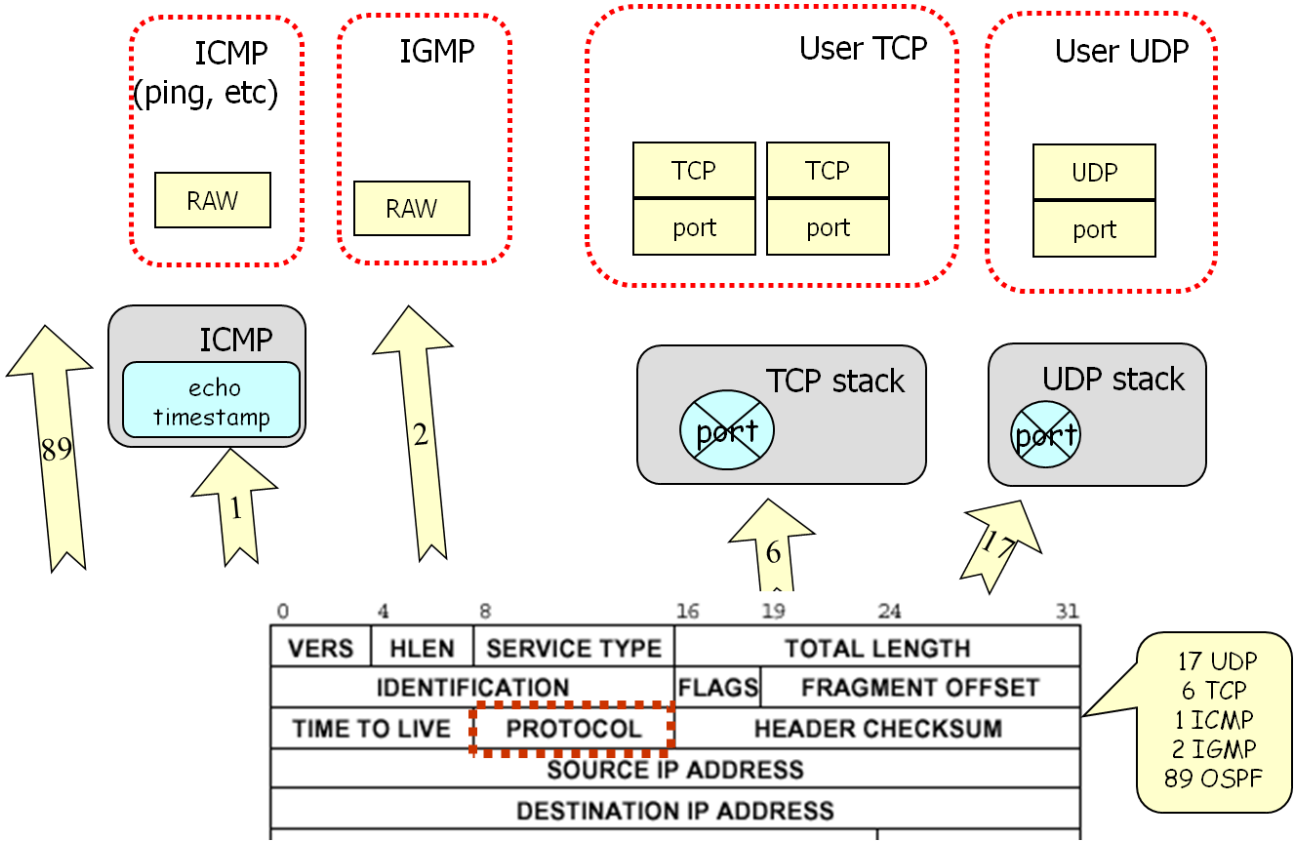
What is Raw Socket?



What can raw sockets do?

- Bypass TCP/UDP layers
- Read and write ICMP and IGMP packets
  - ping, traceroute, multicast routing daemon
- Read and write IP datagrams with an IP protocol field not processed by the kernel
  - OSPF
- Send and receive your own IP packets with your own IP header using the IP_HDRINCL socket option
  - can build and send TCP and UDP packets
  - testing, hacking only superuser can create raw socket though
- You need to do all protocol processing at user-level

Protocol Field in IPv4



Creating Raw Sockets

- Only Superuser can create

```
int fd= socket(AF_INET, SOCK_RAW, protocol);
```

  - where protocol is one of the constants, IPPROTO_xxx, such as IPPROTO_ICMP.
- *bind()* can be called on the raw socket. This function sets only the local address: There is no concept of a port number with a raw socket.
- *connect()* can be called on the raw socket, but this is rare. This function sets only the foreign address: Again, there is no concept of a port number with a raw socket.

Raw Socket Output

- Output is performed by calling sendto or sendmsg and specifying the destination IP address.
  - If the IP_HDRINCL option is not set, kernel prepends the IP header.
- The kernel sets the protocol field of the IPv4 header to the third argument from the call to socket.
  - If the IP_HDRINCL option is set, the starting address of the data for the kernel to send specifies the first byte of the IP header.
- The process builds the entire IP header, except:
  - the IPv4 identification field can be set to 0 which tells the kernel o set this value;
  - the kernel always calculates and stores the IPv4 header checksum;
  - IP options may or may not be included
- The kernel fragments raw packets that exceed the outgoing interface MTU.

Raw Socket Input

- Which received IP datagrams does the kernel pass to raw sockets?
  - UDP packets and TCP packets are never passed to a raw socket.
    - read at the datalink layer
  - ICMP packets passed to a raw socket after the kernel has finished processing the ICMP message.
    - Except echo request, timestamp request, and address mask request
  - IGMP packets passed to a raw socket after the kernel has finished processing the IGMP message.
  - All IP datagrams with a protocol field that the kernel does not understand are passed to a raw socket.
  - If the datagram arrives in fragments, nothing is passed to a raw socket until all fragments have arrived and have been reassembled.

Raw Socket Input

- When the kernel has an IP datagram, all raw sockets for all processes are examined, looking for all matching sockets.
  - A copy of the IP datagram is delivered to each matching socket.
- the datagram delivered to the socket:
  - If a nonzero protocol is specified in *socket()*, protocol field must match
  - If bind() is used, then the destination IP address of the received datagram must match with local IP.
  - If connect() is used, then the source IP address of the received datagram must match destination IP.
- if a raw socket is created with a protocol of 0, and neither bind nor connect is called, then that socket receives a copy of every raw datagram the kernel passes to raw sockets.

Raw Sockets & IPv6

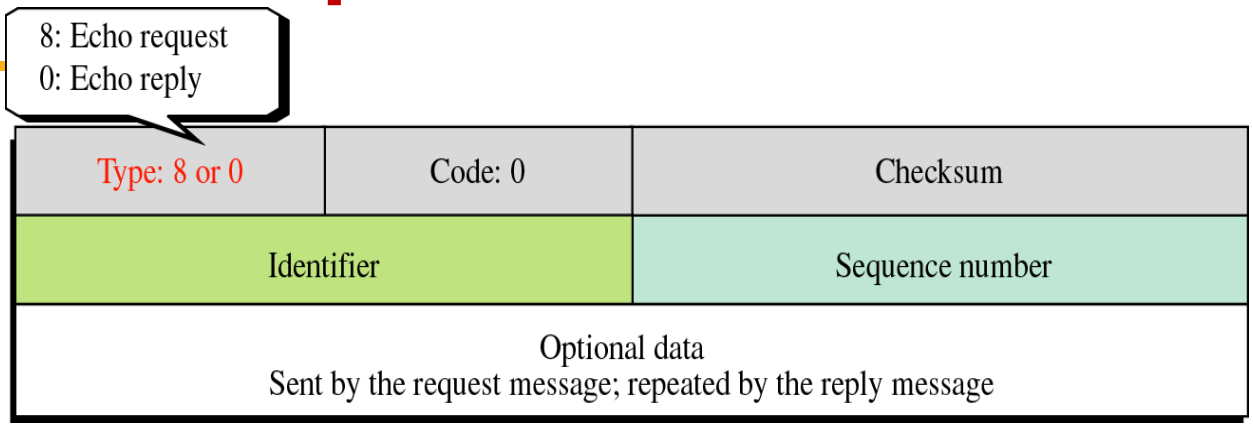
- Whenever a received datagram is passed to a raw IPv4 socket, the entire datagram, including the IP header, is passed to the process
- For a raw IPv6 socket, only the payload (i.e., no IPv6 header or any extension headers) is passed to the socket .
  - All fields in the header are available as socket options or ancillary objects. [RFC 3542]
- There is nothing similar to the IPv4 IP_HDRINCL socket option with IPv6.
- Complete IPv6 packets (including the IPv6 header or extension headers) cannot be read or written on an IPv6 raw socket.

Example : Ping

```
1  $ ping www.google.com
2  PING www.google.com (216.239.57.99): 56 data bytes
3  64 bytes from 216.239.57.99: seq=0, ttl=53, rtt=5.611 ms
4  64 bytes from 216.239.57.99: seq=1, ttl=53, rtt=5.562 ms
5  64 bytes from 216.239.57.99: seq=2, ttl=53, rtt=5.589 ms
6  64 bytes from 216.239.57.99: seq=3, ttl=53, rtt=5.910 ms
```

- Ping is used to check the status of the device.
- Ping is also used to measure RTT (Round Trip Time).
- Ping application is developed on ICMP protocol messages.
  - ICMP echo request
  - ICMP echo reply

ICMP Echo Request

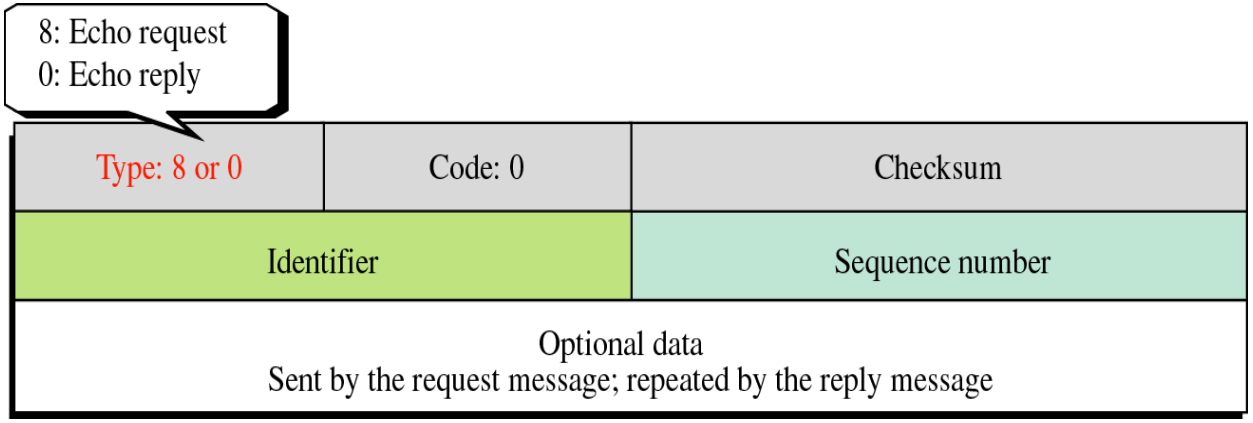


...e Field	ICMP Message Type
0	Echo Reply
3	Destination Unreachable
4	Source Quench
5	Redirect (change a route)
8	Echo Request
9	Router Advertisement
10	Router Solicitation
11	Time Exceeded for a Datagram
12	Parameter Problem on a Datagram
13	Timestamp Request
14	Timestamp Reply
15	Information Request (obsolete)
16	Information Reply (obsolete)
17	Address Mask Request
18	Address Mask Reply

Ping Program

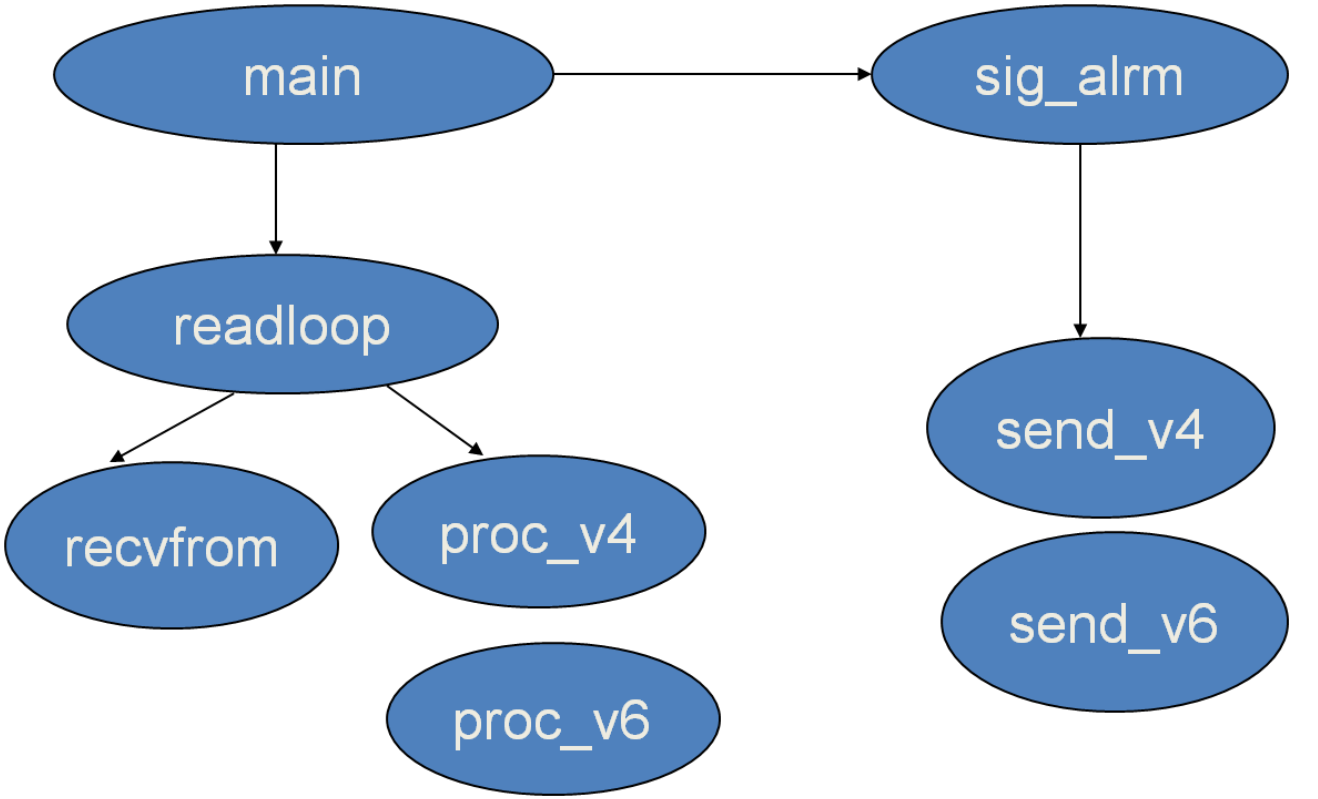
- Create a raw socket to send/receive ICMP echo request and echo reply packets
- Install SIGALRM handler to process output or using multiple threads.
  - Sending echo request packets every t second
  - Build ICMP packets (type, code, checksum, id, seq, sending timestamp as optional data)
- Enter an infinite loop processing input
  - Use recvmsg() to read from the network
  - Parse the message and retrieve the ICMP packet
  - Print ICMP packet information, e.g., peer IP address, round-trip time

ICMP Message



- set the identifier to the PID of the ping process and we increment the sequence number by one for each packet we send
- we store the 8-byte timestamp of when the packet is sent as the optional data. The rules of ICMP require that the identifier, sequence number, and any optional data be returned in the echo reply.
- Storing the timestamp in the packet lets us calculate the RTT when the reply is received.

Overview of Functions



- send_v4 or send_v6 send ICMP echo message.
- recvfrom() and proc_v4 receives and processes the ICMP echo reply message.

ping.h


```
#define BUFSIZE         1500

                        /* globals */
char     sendbuf[BUFSIZE];

int              datalen;                       /* # bytes of data following ICMP header */
char    *host;
int              nsent;                         /* add 1 for each sendto() */
pid_t    pid;                                /* our PID */
int              sockfd;
int              verbose;

                        /* function prototypes */
void     init_v6(void);
void     proc_v4(char *, ssize_t, struct msghdr *, struct timeval *);
void     proc_v6(char *, ssize_t, struct msghdr *, struct timeval *);
void     send_v4(void);
void     send_v6(void);
void     readloop(void);
void     sig_alrm(int);
void     tv_sub(struct timeval *, struct timeval *);

struct proto {
  void   (*fproc)(char *, ssize_t, struct msghdr *, struct timeval *);
  void   (*fsend)(void);
  void   (*finit)(void);
  struct sockaddr  *sasend;     /* sockaddr{} for send, from getaddrinfo */
  struct sockaddr  *sarecv;     /* sockaddr{} for receiving */
  socklen_t         salen;              /* length of sockaddr{}s */
  int               icmpproto;  /* IPPROTO_xxx value for ICMP */
} *pr;
```

Structure for handling differences in ipv4 and ipv6

Ping Code: main.c

ipv4 functions

ipv6 functions

```
#include        "ping.h"
struct proto    proto_v4 = { proc_v4, send_v4, NULL, NULL, NULL, 0, IPPROTO_ICMP };
#ifdef  IPV6
struct proto    proto_v6 = { proc_v6, send_v6, init_v6, NULL, NULL, 0, IPPROTO_ICMPV6 };
#endif
int     datalen = 56;           /* data that goes with ICMP echo request */
int
main(int argc, char **argv)
{
        int                             c;
        struct addrinfo *ai;
        char *h;
        opterr = 0;             /* don't want getopt() writing to stderr */
        while ( (c = getopt(argc, argv, "v")) != -1) {
                switch (c) {
                case 'v':
                        verbose++;
                        break;
                case '?':
                        err_quit("unrecognized option: %c", c);
                }
        }
        if (optind != argc-1)
                err_quit("usage: ping [ -v ] <hostname>");
        host = argv[optind];
```

Ping Code: main.c

innovate achieve lead

```
host = argv[optind];
pid = getpid() & 0xffff;          /* ICMP ID field is 16 bits */
Signal(SIGALRM, sig_alrm);
ai = Host_serv(host, NULL, 0, 0);
h = Sock_ntop_host(ai->ai_addr, ai->ai_addrlen);
printf("PING %s (%s): %d data bytes\n",
                    ai->ai_canonname ? ai->ai_canonname : h,
                    h, datalen);
```

getaddrinfo()

Ping Code: main.c

innovate achieve lead

```
                /* 4initialize according to protocol */
        if (ai->ai_family == AF_INET) {
                pr = &proto_v4;
#ifdef  IPV6
        } else if (ai->ai_family == AF_INET6) {
                pr = &proto_v6;
                if (IN6_IS_ADDR_V4MAPPED(&(((struct sockaddr_in6 *)
                                                    ai->ai_addr)->sin6_addr)))
                        err_quit("cannot ping IPv4-mapped IPv6 address");
#endif
        } else
                err_quit("unknown address family %d", ai->ai_family);
        pr->sasend = ai->ai_addr;
        pr->sarecv = Calloc(1, ai->ai_addrlen);
        pr->salen = ai->ai_addrlen;
        readloop();
        exit(0);
}
```

If ipv4

If ipv6

Ping Code: readloop.c

```
void
sig_alrm(int signo)
{
        (*pr->fsend)();

        alarm(1);
        return;
}
```

```
readloop(void)
{
        int                     size;
        char                    recvbuf[BUFSIZE];
        char                    controlbuf[BUFSIZE];
        struct msghdr   msg;
        struct iovec    iov;
        ssize_t                 n;
        struct timeval  tval;
        sockfd = Socket(pr->sasend->sa_family, SOCK_RAW, pr->icmpproto);
        setuid(getuid());               /* don't need special permissions any more */
        if (pr->finit)
                (*pr->finit)();
        size = 60 * 1024;               /* OK if setsockopt fails */
        setsockopt(sockfd, SOL_SOCKET, SO_RCVBUF, &size, sizeof(size));
        sig_alrm(SIGALRM);      /* send first packet */
        iov.iov_base = recvbuf;
        iov.iov_len = sizeof(recvbuf);
        msg.msg_name = pr->sarecv;
        msg.msg_iov = &iov;
        msg.msg_iovlen = 1;
        msg.msg_control = controlbuf;
```

Set protocol to ICMP

setting recv buffer size to accommodate many replies

call signal handler

Socket addr structure for rcv address.

Ping Code: readloop.c


```
for ( ; ; ) {
        msg.msg_namelen = pr->salen;
        msg.msg_controllen = sizeof(controlbuf);
        n = recvmsg(sockfd, &msg, 0);
        if (n < 0) {
                if (errno == EINTR)
                        continue;
                else
                        err_sys("recvmsg error");
        }
        Gettimeofday(&tval, NULL);
        (*pr->fproc)(recvbuf, n, &msg, &tval);
}
```

receive ICMP

Store current time in *tval* global var

call protocol specific processing function: proc_v4 or proc_v6

Ping Code: proc_v4.c

innovate achieve lead

```
proc_v4(char *ptr, ssize_t len, struct msghdr *msg, struct timeval *tvrecv)
{
        int                             hlen1, icmplen;
        double                  rtt;
        struct ip               *ip;
        struct icmp             *icmp;
        struct timeval  *tvsend;

        ip = (struct ip *) ptr;         /* start of IP header */
        hlen1 = ip->ip_hl << 2;         /* length of IP header */
        if (ip->ip_p != IPPROTO_ICMP)
                return;                                 /* not ICMP */

        icmp = (struct icmp *) (ptr + hlen1);   /* start of ICMP header */
        if ( (icmplen = len - hlen1) < 8)
                return;                                 /* malformed packet */
```

Ip_hl is multiplied by 4

Check protocol field in IP hdr

Ping Code: proc_v4.c

innovate achieve lead

```
if (icmp->icmp_type == ICMP_ECHOREPLY) {
        if (icmp->icmp_id != pid)
                return;                 /* not a response to our ECHO_REQUEST */
        if (icmplen < 16)
                return;                 /* not enough data to use */

        tvsend = (struct timeval *) icmp->icmp_data;
        tv_sub(tvrecv, tvsend);
        rtt = tvrecv->tv_sec * 1000.0 + tvrecv->tv_usec / 1000.0;

        printf("%d bytes from %s: seq=%u, ttl=%d, rtt=%.3f ms\n",
                        icmplen, Sock_ntop_host(pr->sarecv, pr->salen),
                        icmp->icmp_seq, ip->ip_ttl, rtt);

} else if (verbose) {
        printf("  %d bytes from %s: type = %d, code = %d\n",
                        icmplen, Sock_ntop_host(pr->sarecv, pr->salen),
                        icmp->icmp_type, icmp->icmp_code);
}
```

Validate msg type

Get the timestamp from the packet

Compute RTT

Print message to console

Ping Code: proc_v6.c


```
proc_v6(char *ptr, ssize_t len, struct msghdr *msg, struct timeval* tvrecv)
{
#ifdef  IPV6
        double                          rtt;
        struct icmp6_hdr        *icmp6;
        struct timeval          *tvsend;
        struct cmsghdr          *cmsg;
        int                                     hlim;

        icmp6 = (struct icmp6_hdr *) ptr;
        if (len < 8)
                return;                         /* malformed packet */

        if (icmp6->icmp6_type == ICMP6_ECHO_REPLY) {
                if (icmp6->icmp6_id != pid)
                        return;                 /* not a response to our ECHO_REQUEST */
                if (len < 16)
                        return;                 /* not enough data to use */

                tvsend = (struct timeval *) (icmp6 + 1);
                tv_sub(tvrecv, tvsend);
                rtt = tvrecv->tv_sec * 1000.0 + tvrecv->tv_usec / 1000.0;
```

Starting of the message itself is ICMP header because no IPv6 hdr

Validate msg type

Get the timestamp from the packet

Compute RTT

Ping Code: proc_v6.c

innovate achieve lead

```
                hlim = -1;
                for (cmsg = CMSG_FIRSTHDR(msg); cmsg != NULL; cmsg = CMSG_NXTHDR(msg, cmsg)) {
                        if (cmsg->cmsg_level == IPPROTO_IPV6 &&
                                cmsg->cmsg_type == IPV6_HOPLIMIT) {
                                hlim = *(u_int32_t *)CMSG_DATA(cmsg);
                                break;
                        }
                }
                printf("%d bytes from %s: seq=%u, hlim=",
                                len, Sock_ntop_host(pr->sarecv, pr->salen),
                                icmp6->icmp6_seq);
                if (hlim == -1)
                        printf("???");  /* ancillary data missing */
                else
                        printf("%d", hlim);
                printf(", rtt=%.3f ms\n", rtt);
        } else if (verbose) {
                printf("  %d bytes from %s: type = %d, code = %d\n",
                                len, Sock_ntop_host(pr->sarecv, pr->salen),
                                icmp6->icmp6_type, icmp6->icmp6_code);
        }
#endif  /* IPV6 */
}
```

Get hop limit using ancillary data

Print msg to console

Ping Code: send_v4.c

innovate achieve lead

```
void
send_v4(void)
{
        int                     len;
        struct icmp     *icmp;

        icmp = (struct icmp *) sendbuf;
        icmp->icmp_type = ICMP_ECHO;
        icmp->icmp_code = 0;
        icmp->icmp_id = pid;
        icmp->icmp_seq = nsent++;
        memset(icmp->icmp_data, 0xa5, datalen); /* fill with pattern */
        Gettimeofday((struct timeval *) icmp->icmp_data, NULL);

        len = 8 + datalen;                      /* checksum ICMP header and data */
        icmp->icmp_cksum = 0;
        icmp->icmp_cksum = in_cksum((u_short *) icmp, len);

        Sendto(sockfd, sendbuf, len, 0, pr->sasend, pr->salen);
}
~
~
```

Set ICMP fields

Set data to timstamp

Ping Code: init_v6.c

innovate achieve lead

```
void
init_v6()
{
#ifdef IPV6
        int on = 1;

        if (verbose == 0) {
                /* install a filter that only passes ICMP6_ECHO_REPLY unless verbose */
                struct icmp6_filter myfilt;
                ICMP6_FILTER_SETBLOCKALL(&myfilt);
                ICMP6_FILTER_SETPASS(ICMP6_ECHO_REPLY, &myfilt);
                setsockopt(sockfd, IPPROTO_IPV6, ICMP6_FILTER, &myfilt, sizeof(myfilt));
                /* ignore error return; the filter is an optimization */
        }

        /* ignore error returned below; we just won't receive the hop limit */
#ifdef IPV6_RECVHOPLIMIT
        /* RFC 3542 */
        setsockopt(sockfd, IPPROTO_IPV6, IPV6_RECVHOPLIMIT, &on, sizeof(on));
#else
        /* RFC 2292 */
        setsockopt(sockfd, IPPROTO_IPV6, IPV6_HOPLIMIT, &on, sizeof(on));
#endif
#endif
}
```

IPv6 offers finer control: we specify which message types to receive

Enable recving hop limit through ancillary data

Ping Code: send_v6.c

innovate achieve lead

```
void
send_v6()
{
#ifdef  IPV6
        int                                     len;
        struct icmp6_hdr        *icmp6;

        icmp6 = (struct icmp6_hdr *) sendbuf;
        icmp6->icmp6_type = ICMP6_ECHO_REQUEST;
        icmp6->icmp6_code = 0;
        icmp6->icmp6_id = pid;
        icmp6->icmp6_seq = nsent++;
        memset((icmp6 + 1), 0xa5, datalen);     /* fill with pattern */
        Gettimeofday((struct timeval *) (icmp6 + 1), NULL);

        len = 8 + datalen;                      /* 8-byte ICMPv6 header */

        Sendto(sockfd, sendbuf, len, 0, pr->sasend, pr->salen);
                /* 4kernel calculates and stores checksum for us */
#endif  /* IPV6 */
}
```

Set vales to ICMP fields

Store current timestamp into data field

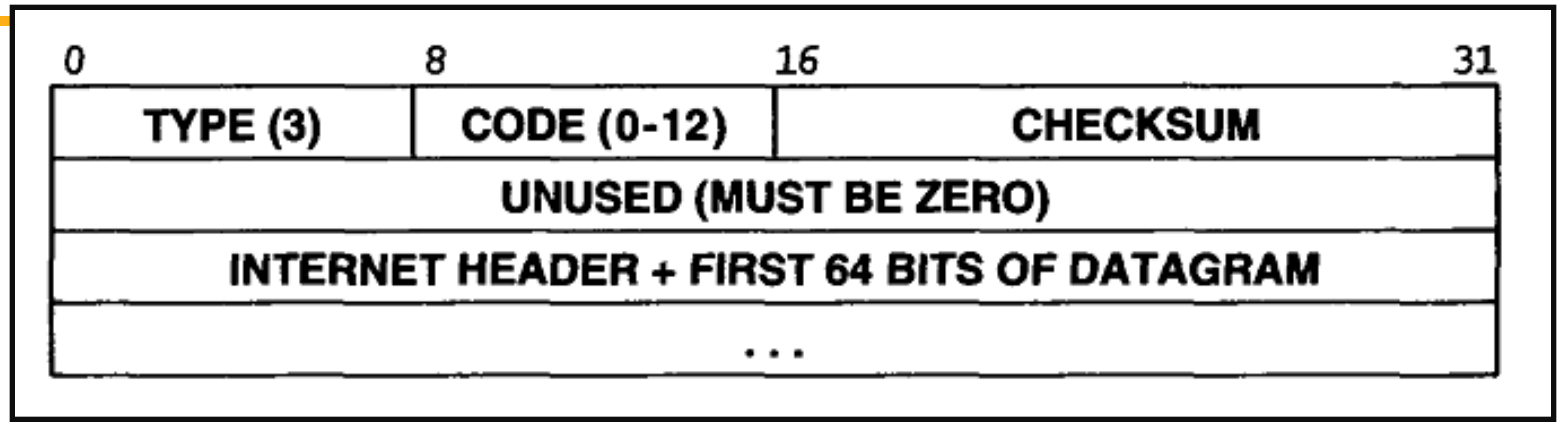
Example: Trace Route

- Determines the path IP datagrams follow.
- Uses TTL field(IPv4) or hop limit(IPv6) and two ICMP messages
- One UDP datagram is sent by the host with TTL=1 to the destination
- 1st hop router sends an ICMP “time exceed in transit” error
- TTL is increased to 2, and another datagram is sent
- Process repeats with a final datagram with a port number not in use on the destination, so that destination can send “ICMP port unreachable” error

Trace Route Program

- Create a UDP socket and bind source port
  - To send probe packets with increasing TTL
  - For each TTL value, use timer to send a probe every three seconds, and send 3 probes in total
- Create a raw socket to receive ICMP packets
  - If timeout, printing “ *”
  - If ICMP “port unreachable”, then terminate
  - If ICMP “TTL expired”, then printing hostname of the router and round trip time to the router

ICMP Destination Unreachable Message



e Field	ICMP Message Type
0	Echo Reply
3	Destination Unreachable
4	Source Quench
5	Redirect (change a route)
8	Echo Request
9	Router Advertisement
10	Router Solicitation
11	Time Exceeded for a Datagram
12	Parameter Problem on a Datagram
13	Timestamp Request
14	Timestamp Reply
15	Information Request (obsolete)
16	Information Reply (obsolete)
17	Address Mask Request
18	Address Mask Reply

Code Value	Meaning
0	Network unreachable
1	Host unreachable
2	Protocol unreachable
3	Port unreachable
4	Fragmentation needed and DF set
5	Source route failed
6	Destination network unknown
7	Destination host unknown
8	Source host isolated
9	Communication with destination network administratively prohibited
10	Communication with destination host administratively prohibited
11	Network unreachable for type of service
12	Host unreachable for type of service

Traceroute: trace.h

innovate achieve lead

```
#define BUFSIZE         1500

struct rec {                                    /* format of outgoing UDP data */
  u_short       rec_seq;                        /* sequence number */
  u_short       rec_ttl;                        /* TTL packet left with */
  struct timeval        rec_tv;         /* time packet left */
};

                        /* globals */
char     recvbuf[BUFSIZE];
char     sendbuf[BUFSIZE];

int              datalen;                       /* # bytes of data following ICMP header */
char    *host;
u_short  sport, dport;
int              nsent;                         /* add 1 for each sendto() */
pid_t    pid;                           /* our PID */
int              probe, nprobes;
int              sendfd, recvfd;        /* send on UDP sock, read on raw ICMP sock */
int              ttl, max_ttl;
int              verbose;

                        /* function prototypes */
const char      *icmpcode_v4(int);
const char      *icmpcode_v6(int);
int              recv_v4(int, struct timeval *);
int              recv_v6(int, struct timeval *);
void     sig_alrm(int);
void     traceloop(void);
void     tv_sub(struct timeval *, struct timeval *);
```

Traceroute: trace.h


```
struct proto {
  const char    *(*icmpcode)(int);
  int    (*recv)(int, struct timeval *);
  struct sockaddr  *sasend;     /* sockaddr{} for send, from getaddrinfo */
  struct sockaddr  *sarecv;     /* sockaddr{} for receiving */
  struct sockaddr  *salast;     /* last sockaddr{} for receiving */
  struct sockaddr  *sabind;     /* sockaddr{} for binding source port */
  socklen_t             salen;  /* length of sockaddr{}s */
  int                   icmpproto;      /* IPPROTO_xxx value for ICMP */
  int      ttllevel;            /* setsockopt() level to set TTL */
  int      ttloptname;          /* setsockopt() name to set TTL */
} *pr;
```

Traceroute: main.c


```
struct proto     proto_v4 = { icmpcode_v4, recv_v4, NULL, NULL, NULL, NULL, 0,
                                                    IPPROTO_ICMP, IPPROTO_IP, IP_TTL };

#ifdef  IPV6
struct proto     proto_v6 = { icmpcode_v6, recv_v6, NULL, NULL, NULL, NULL, 0,
                                                    IPPROTO_ICMPV6, IPPROTO_IPV6, IPV6_UNICAST_HOPS }
#endif

int              datalen = sizeof(struct rec);    /* defaults */
int              max_ttl = 30;
int              nprobes = 3;
u_short dport = 32768 + 666;
```

Traceroute: main.c

innovate achieve lead

```
        pid = getpid();
        Signal(SIGALRM, sig_alrm);

        ai = Host_serv(host, NULL, 0, 0);

        h = Sock_ntop_host(ai->ai_addr, ai->ai_addrlen);
        printf("traceroute to %s (%s): %d hops max, %d data bytes\n",
                   ai->ai_canonname ? ai->ai_canonname : h,
                   h, max_ttl, datalen);

                /* initialize according to protocol */
        if (ai->ai_family == AF_INET) {
                pr = &proto_v4;
#ifdef  IPV6
//      } else if (ai->ai_family == AF_INET6) {
//              pr = &proto_v6;
//              if (IN6_IS_ADDR_V4MAPPED(&(((struct sockaddr_in6 *)ai->ai_addr)->sin6_addr)))
//                      err_quit("cannot traceroute IPv4-mapped IPv6 address");
#endif
        } else
                err_quit("unknown address family %d", ai->ai_family);

        pr->sasend = ai->ai_addr;               /* contains destination address */
        pr->sarecv = Calloc(1, ai->ai_addrlen);
        pr->salast = Calloc(1, ai->ai_addrlen);
        pr->sabind = Calloc(1, ai->ai_addrlen);
        pr->salen = ai->ai_addrlen;

        traceloop();

        exit(0);
}
```

Traceroute: traceloop.c

innovate achieve lead

```
traceloop(void)
{
        int                                     seq, code, done;
        double                          rtt;
        struct rec                      *rec;
        struct timeval          tvrecv;

        recvfd = Socket(pr->sasend->sa_family, SOCK_RAW, pr->icmpproto);
        setuid(getuid());               /* don't need special permissions anymore */

#ifdef  IPV6
        if (pr->sasend->sa_family == AF_INET6 && verbose == 0) {
                struct icmp6_filter myfilt;
                ICMP6_FILTER_SETBLOCKALL(&myfilt);
                ICMP6_FILTER_SETPASS(ICMP6_TIME_EXCEEDED, &myfilt);
                ICMP6_FILTER_SETPASS(ICMP6_DST_UNREACH, &myfilt);
                setsockopt(recvfd, IPPROTO_IPV6, ICMP6_FILTER,
                                        &myfilt, sizeof(myfilt));
        }
#endif

        sendfd = Socket(pr->sasend->sa_family, SOCK_DGRAM, 0);

        pr->sabind->sa_family = pr->sasend->sa_family;
        sport = (getpid() & 0xffff) | 0x8000;   /* our source UDP port # */
        sock_set_port(pr->sabind, pr->salen, htons(sport));
        Bind(sendfd, pr->sabind, pr->salen);

        sig_alrm(SIGALRM);
```

Traceroute: traceloop.c

innovate achieve lead

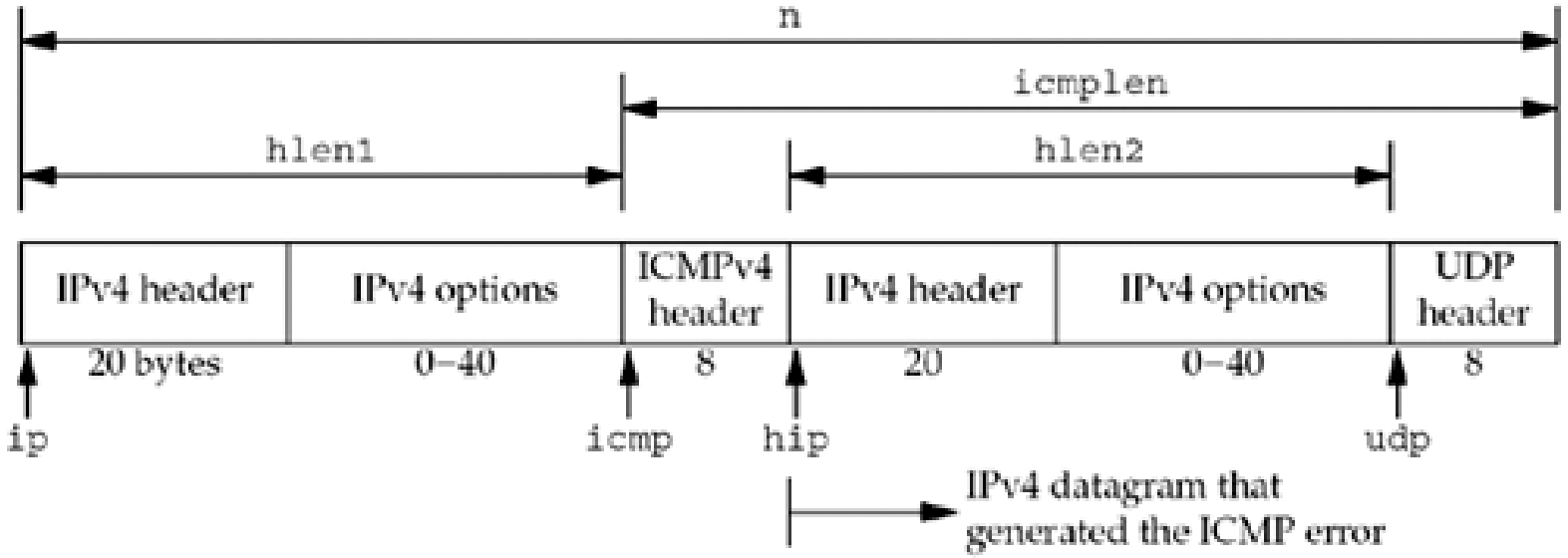
```
for (ttl = 1; ttl <= max_ttl && done == 0; ttl++) {
        Setsockopt(sendfd, pr->ttllevel, pr->ttloptname, &ttl, sizeof(int));
        bzero(pr->salast, pr->salen);

        printf("%2d ", ttl);
        fflush(stdout);

        for (probe = 0; probe < nprobes; probe++) {
                rec = (struct rec *) sendbuf;
                rec->rec_seq = ++seq;
                rec->rec_ttl = ttl;
                Gettimeofday(&rec->rec_tv, NULL);

                sock_set_port(pr->sasend, pr->salen, htons(dport + seq));
                Sendto(sendfd, sendbuf, datalen, 0, pr->sasend, pr->salen);
```

Traceroute: traceloop.c

innovate achieve lead

```
if ( (code = (*pr->recv)(seq, &tvrecv)) == -3)
        printf(" *");           /* timeout, no reply */
else {
        char    str[NI_MAXHOST];

        if (sock_cmp_addr(pr->sarecv, pr->salast, pr->salen) != 0) {
                if (getnameinfo(pr->sarecv, pr->salen, str, sizeof(str),
                                                NULL, 0, 0) == 0)
                        printf(" %s (%s)", str,
                                        Sock_ntop_host(pr->sarecv, pr->salen));
                else
                        printf(" %s",
                                        Sock_ntop_host(pr->sarecv, pr->salen));
                memcpy(pr->salast, pr->sarecv, pr->salen);
        }
        tv_sub(&tvrecv, &rec->rec_tv);
        rtt = tvrecv.tv_sec * 1000.0 + tvrecv.tv_usec / 1000.0;
        printf("  %.3f ms", rtt);

        if (code == -1)         /* port unreachable; at destination */
                done++;
        else if (code >= 0)
                printf(" (ICMP %s)", (*pr->icmpcode)(code));
}
```

ICMPv4 packet



Traceroute: recv_v4.c


```
/*
 * Return: -3 on timeout
 *                 -2 on ICMP time exceeded in transit (caller keeps going)
 *                 -1 on ICMP port unreachable (caller is done)
 *               >= 0 return value is some other ICMP unreachable code
 */

int
recv_v4(int seq, struct timeval *tv)
{
        int                             hlen1, hlen2, icmplen, ret;
        socklen_t               len;
        ssize_t                 n;
        struct ip               *ip, *hip;
        struct icmp             *icmp;
        struct udphdr   *udp;

        gotalarm = 0;
        alarm(3);
        for ( ; ; ) {
                if (gotalarm)
                        return(-3);             /* alarm expired */
                len = pr->salen;
                n = recvfrom(recvfd, recvbuf, sizeof(recvbuf), 0, pr->sarecv, &len);
                if (n < 0) {
```

Traceroute: recv_v4.c

innovate achieve lead

```
ip = (struct ip *) recvbuf;     /* start of IP header */
hlen1 = ip->ip_hl << 2;         /* length of IP header */

icmp = (struct icmp *) (recvbuf + hlen1); /* start of ICMP header */
if ( (icmplen = n - hlen1) < 8)
        continue;                           /* not enough to look at ICMP header */

if (icmp->icmp_type == ICMP_TIMXCEED &&
        icmp->icmp_code == ICMP_TIMXCEED_INTRANS) {
        if (icmplen < 8 + sizeof(struct ip))
                continue;                   /* not enough data to look at inner IP */

        hip = (struct ip *) (recvbuf + hlen1 + 8);
        hlen2 = hip->ip_hl << 2;
        if (icmplen < 8 + hlen2 + 4)
                continue;                   /* not enough data to look at UDP ports */

        udp = (struct udphdr *) (recvbuf + hlen1 + 8 + hlen2);
        if (hip->ip_p == IPPROTO_UDP &&
                udp->source == htons(sport) &&
                udp->dest == htons(dport + seq)) {
                ret = -2;               /* we hit an intermediate router */
                break;
        }
```

Traceroute: recv_v4.c

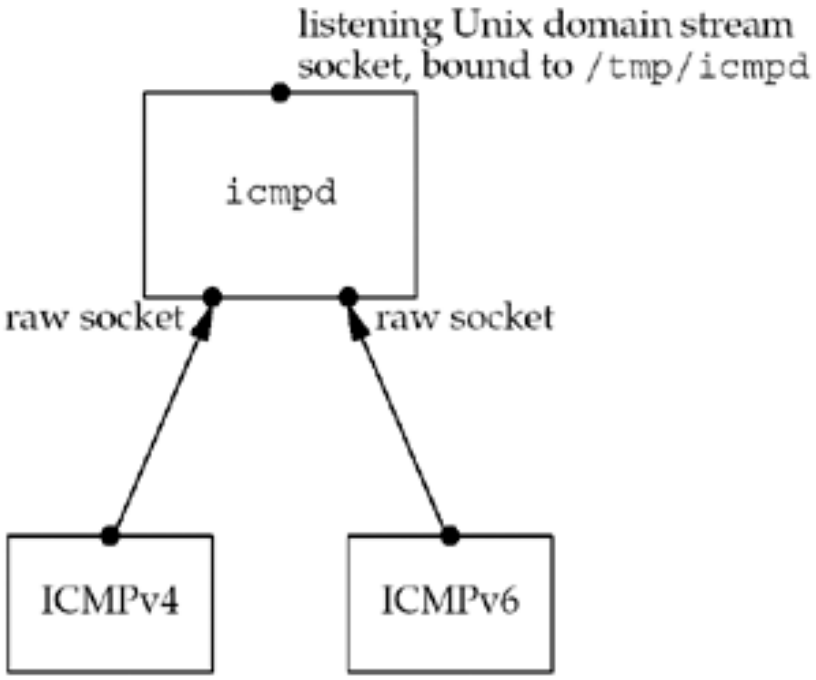
innovate achieve lead

```
} else if (icmp->icmp_type == ICMP_UNREACH) {
        if (icmplen < 8 + sizeof(struct ip))
                continue;                               /* not enough data to

        hip = (struct ip *) (recvbuf + hlen1 + 8);
        hlen2 = hip->ip_hl << 2;
        if (icmplen < 8 + hlen2 + 4)
                continue;                               /* not enough data to

        udp = (struct udphdr *) (recvbuf + hlen1 + 8 + hlen2);
        if (hip->ip_p == IPPROTO_UDP &&
                udp->source == htons(sport) &&
                udp->dest == htons(dport + seq)) {
                if (icmp->icmp_code == ICMP_UNREACH_PORT)
                        ret = -1;           /* have reached destination */
                else
                        ret = icmp->icmp_code;  /* 0, 1, 2, ... */
                break;
        }
}
if (verbose) {
```

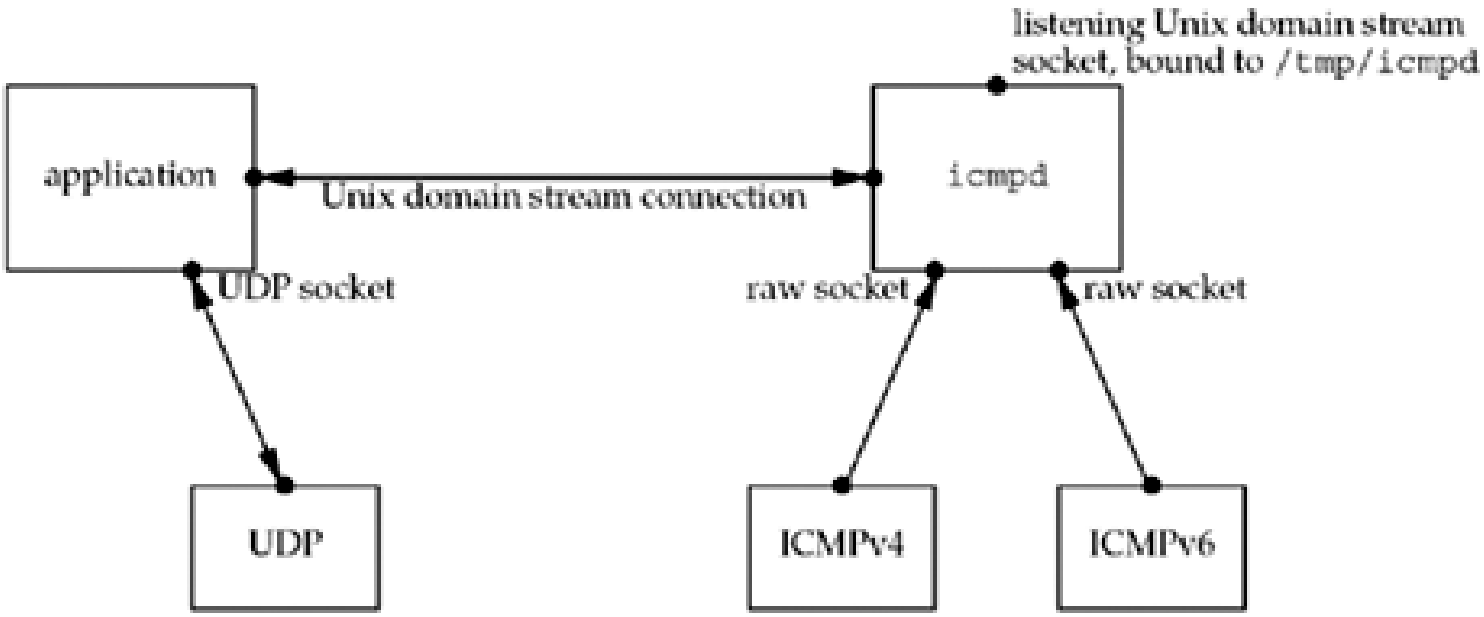
ICMP Daemon

- Receiving asynchronous ICMP errors on a UDP socket has been, and continues to be, a problem.
- ICMP errors are received by the kernel, but are rarely delivered to the application that needs to know about them.
- In the sockets API, it requires connecting the UDP socket to one IP address to receive these errors .
- The reason for this limitation is that the only error returned from recvfrom is an integer errno code, and if the application sends datagrams to multiple destinations and then calls recvfrom, this function cannot tell the application which datagram encountered an error.

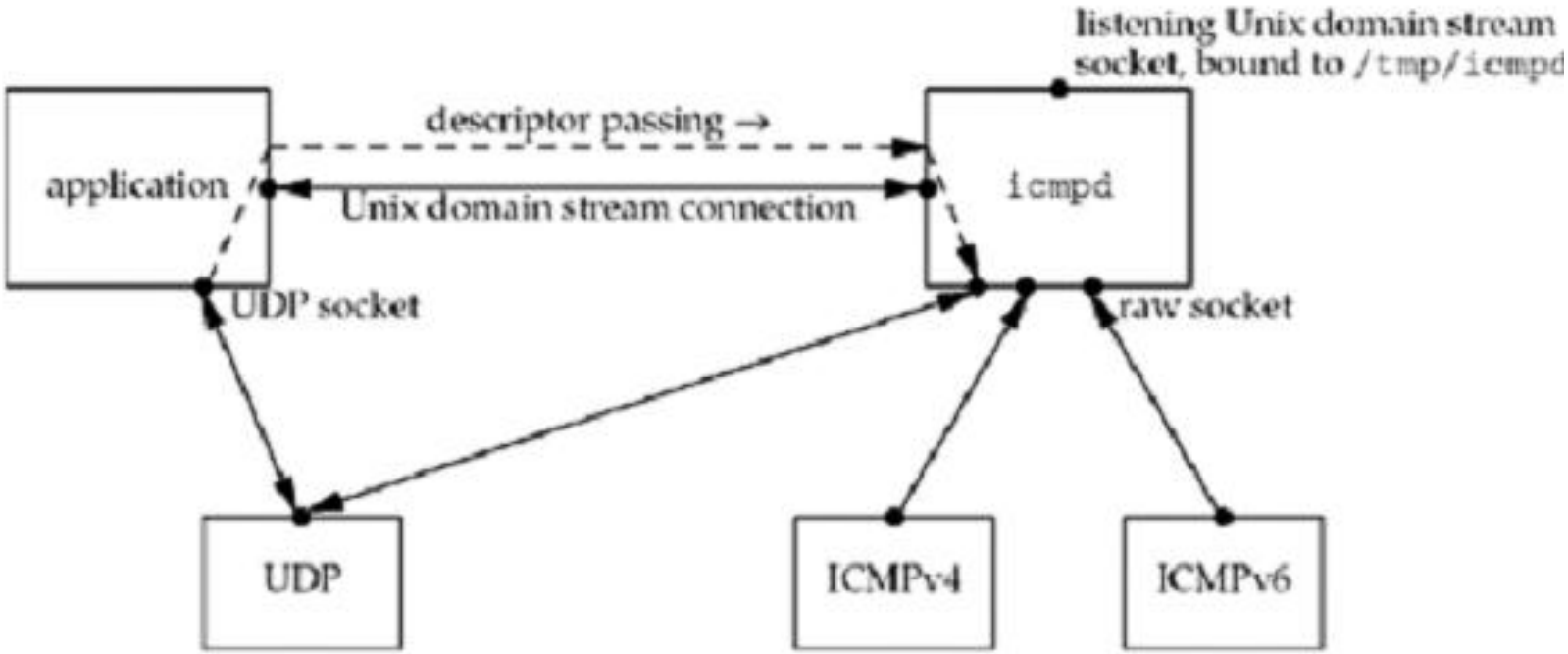
ICMP Daemon


listening Unix domain stream
socket, bound to /tmp/icmpd
icmpd
raw socket
raw socket
ICMPv4
ICMPv6

ICMP Daemon


listening Unix domain stream
socket, bound to /tmp/icmpd
application
Unix domain stream connection
icmpd
UDP socket
raw socket
raw socket
UDP
ICMPv4
ICMPv6

ICMP Daemon


listening Unix domain stream socket, bound to /tmp/icmpd
descriptor passing →
application
Unix domain stream connection
icmpd
UDP socket
raw socket
UDP
ICMPv4
ICMPv6

innovate achieve lead

BITS Pilani

Pilani Campus

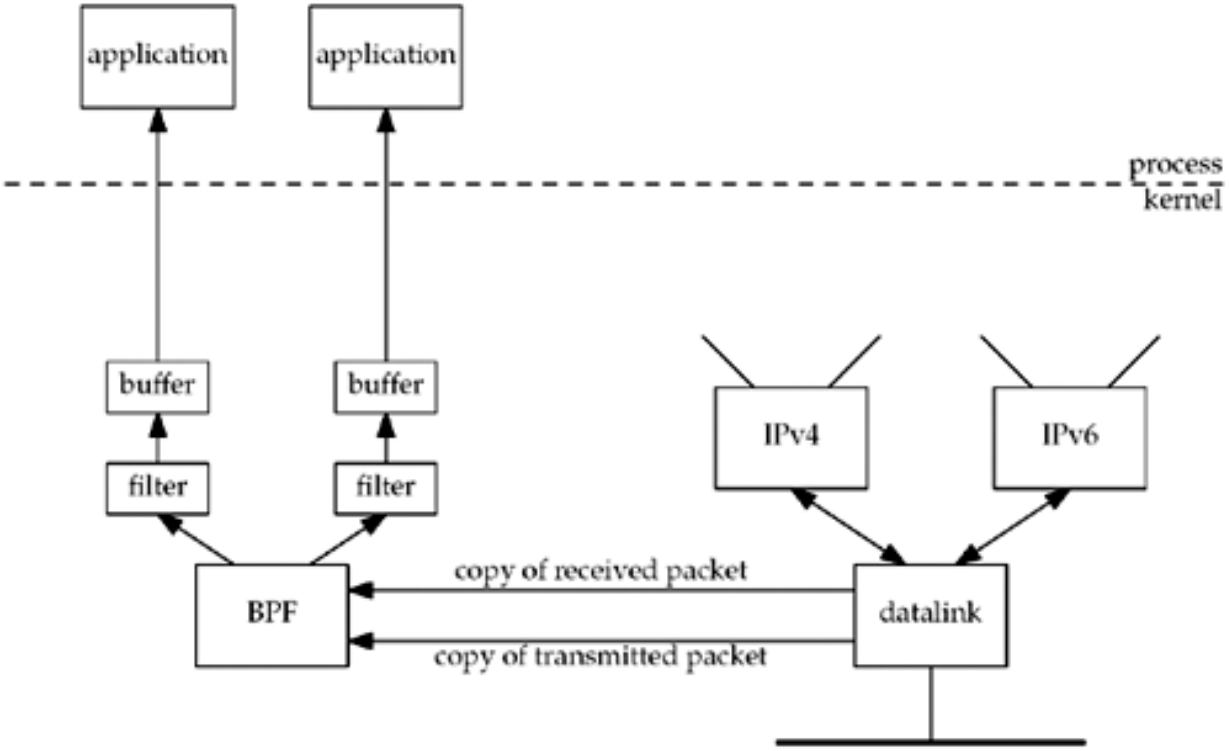
Datalink Access

T1: Ch 29

Data Link Access

- Provides access to the datalink layer for an application
- Capabilities
- Ability to watch the packets received by the datalink layer
- Run certain programs as applications instead of kernel. Ex: RARP
- 3 common methods
  - BSD Packet Filter (BPF)
  - SVR4 Datalink Provider Interface (DLPI)
  - Linux SOCK_PACKET interface
- Libpcap library
  - Publicly available packet capture library
  - Works with all the above three methods.
  - Writing programs with this makes them OS independent

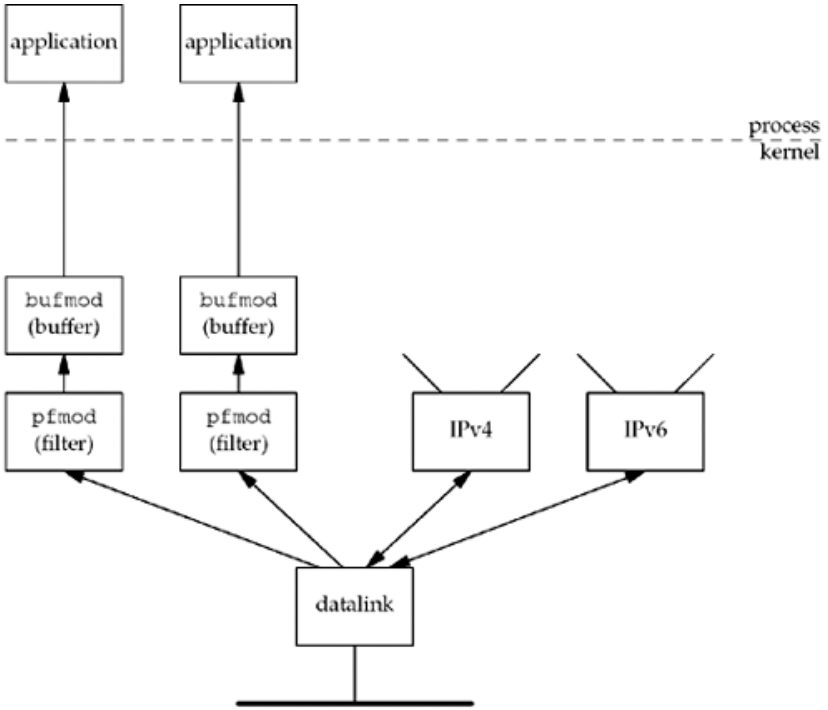
BSD Packet Filter (BPF)



BSD Packet Filter (BPF)

- Three techniques to reduce overhead:
  - Filters within kernel. Avoids data copy into user appl.
  - Only a portion of each packet is copied. (14+40+20+22=96 bytes)
  - BPF buffers the data. It is copied to appl buffer only when full or read timeout expires.

Datalink Provider Interface (DLPI)



- Similar to BPF
- Differences:
  - Filter implementation in BPF is 3 to 20 times faster than DLPI. Directed acyclic control flow graph an boolean expression tree.
  - BPF always make the filtering decision before copying the packet. DLPI may first copy the packet to pfmod and then make the decision.

Linux Support

- Two methods:
  - Socket of type: SOCK_PACKET
    - Available only in older Linux kernel (2.0 or earlier)
    - Socket of family: PF_PACKET
    - Return only complete link layer packet.
  - Family of PF_PACKET:
    - *Newer method* with more filtering and performance features.
    - Type is SOCK_RAW to receive complete link layer packet.
    - Type is SOCK_DGRAM to receive packet without link layer header.
    - *pcap* library uses this method.
- In both cases
  - Third argument must specify the frame type.
  - Admin privileges are required to create a socket.

Linux Support

innovate achieve lead

```
fd = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL));        /* newer systems*/
or
fd = socket(AF_INET, SOCK_PACKET, htons(ETH_P_ALL));      /* older systems*/
```

```
//If we wanted only IPv4 frames, the call would be
fd = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_IP));          /* newer systems */
or
fd = socket(AF_INET, SOCK_PACKET, htons(ETH_P_IP));         /* older systems */
//Other constants for the final argument are ETH_P_ARP and ETH_P_IPV6 etc.
```

Linux Support

- Differences with BPF and DLPI:
  - The Linux feature provides no kernel buffering and kernel filtering is only available on newer systems (via the SO_ATTACH_FILTER socket option).
    - In receive buffer, but multiple frames cannot be buffered together and passed to the application with a single read. So it increases no. of sys calls.
  - SOCK_PACKET provides no filtering by device. (PF_PACKET sockets can be linked to a device by calling bind.)
    - If ETH_P_IP is specified in the call to socket, then all IPv4 packets from all devices (Ethernets, PPP links, SLIP links, and the loopback device, for example) are passed to the socket. A

libpcap library

- Provides implementation independent access to the underlying packet capture facility.

```c
#include <pcap.h>
pcap_t *handle;     /* Session handle */
char dev[] = "rl0";     /* Device to sniff on */
char errbuf[PCAP_ERRBUF_SIZE];  /* Error string */
struct bpf_program fp;      /* The compiled filter expression */
char filter_exp[] = "port 23";  /* The filter expression */
bpf_u_int32 mask;       /* The netmask of our sniffing device */
bpf_u_int32 net;        /* The IP of our sniffing device */

if (pcap_lookupnet(dev, &net, &mask, errbuf) == -1) {
 fprintf(stderr, "Can't get netmask for device %s\n", dev);
 net = 0;
 mask = 0;
}
handle = pcap_open_live(dev, BUFSIZ, 1, 1000, errbuf);
if (handle == NULL) {
 fprintf(stderr, "Couldn't open device %s: %s\n", somedev, errbuf);
 return(2);
}
if (pcap_compile(handle, &fp, filter_exp, 0, net) == -1) {
 fprintf(stderr, "Couldn't parse filter %s: %s\n", filter_exp, pcap_geterr(handle));
 return(2);
}
if (pcap_setfilter(handle, &fp) == -1) {
 fprintf(stderr, "Couldn't install filter %s: %s\n", filter_exp, pcap_geterr(handle));
 return(2);
}
```

Promiscuous mode , timeout in milli seconds

Optimize?, network mask

```
1  /* Grab a packet */
2  packet = pcap_next(handle, &header);
3  /* Print its length */
4  printf("packet with length of [%d]\n", header.len);
5  /* And close the session */
6  pcap_close(handle);
7  }
8
9  struct pcap_pkthdr header;  /* The header that pcap gives us */
```

- To receive multiple packets we need to use the following function with a callback function.

```
int pcap_loop(pcap_t *p, int cnt, pcap_handler callback, u_char *user)
```

- Please see usage of it here:
  http://www.tcpdump.org/sniffex.c
  http://www.tcpdump.org/pcap.html

Acknowledgements

Q&A

innovate achieve lead

BITS Pilani

Pilani Campus

Thank You