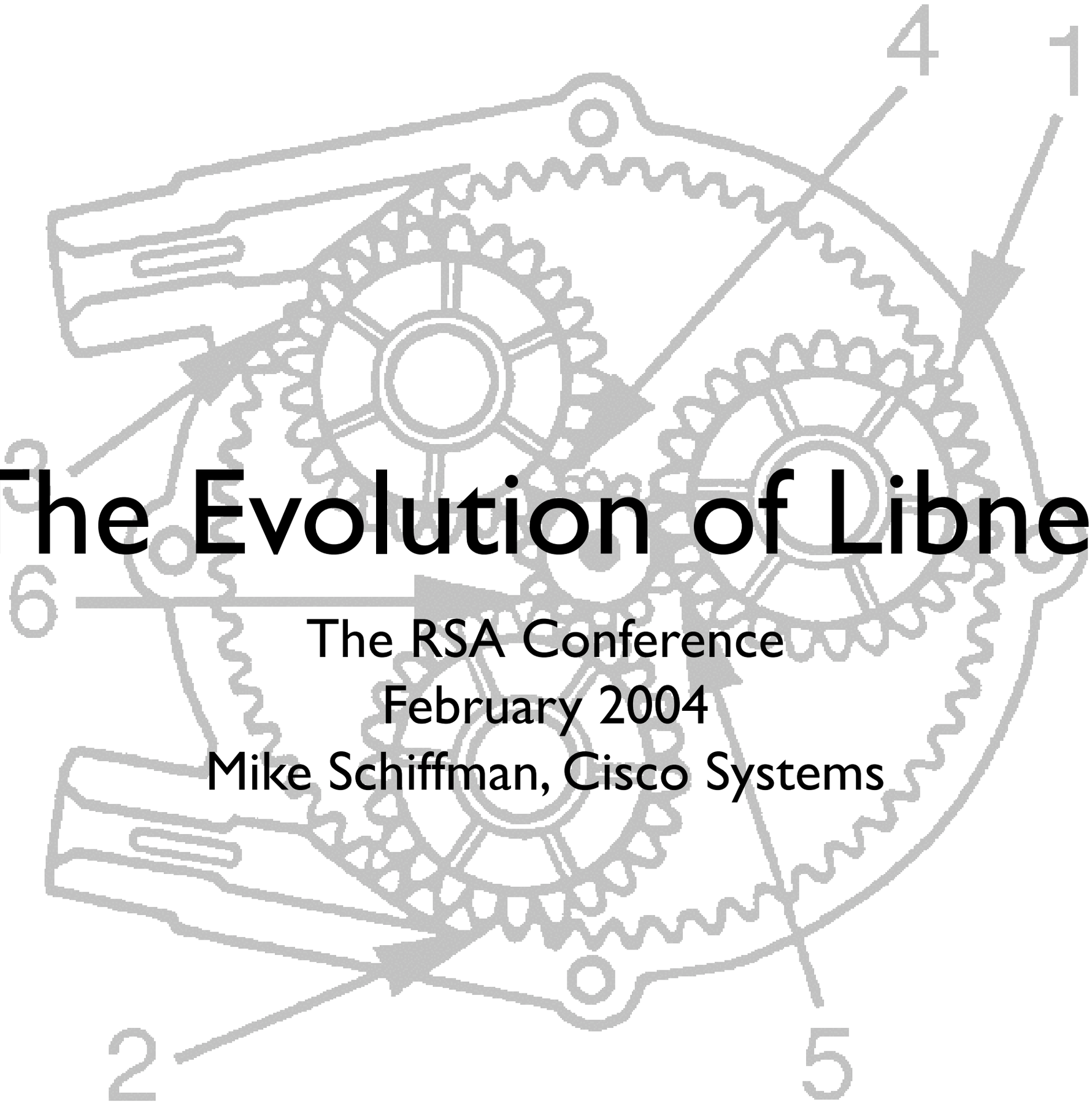


The Evolution of Libnet

The RSA Conference
February 2004

Mike Schiffman, Cisco Systems

Agenda

- Introduction, overview, and what you'll learn
- What is libnet?
- Where we came from: Libnet 1.0.x
  - Process
  - Deficiencies
- Where we are: Libnet 1.1.x
  - Process
  - Key concepts
  - Usage
  - With other components
    - GNIP
    - TRIG
  - Internals
- Closing comments and questions

Mike Schiffman

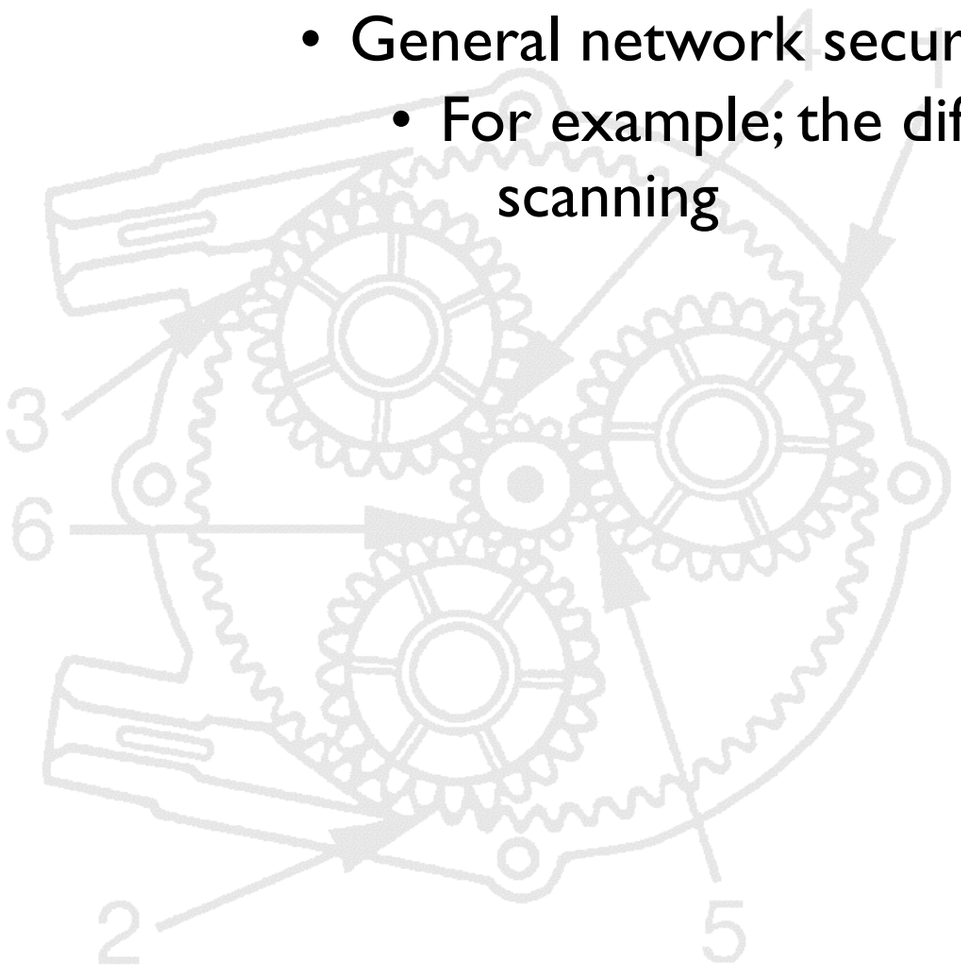
- Researcher for Cisco Systems
  - Critical Infrastructure Assurance Group [CIAG]
- Technical Advisory Boards: Qualys, Sensory Networks, Vigilant, IMG Universal
- Consulting Editor for Wiley & Sons
- R&D, Consulting, Speaking background
  - Firewalk, Libipg, Libnet, Libsf, Libradiate, various whitepapers and reports
- Done time with: @stake, Guardent, Cambridge Technology Partners, ISS
- Current book:
  - Modern Network Infrastructure Security, Addison Wesley (2005)
- Previous books:
  - Building Open Source Network Security Tools, Wiley & Sons
  - Hacker's Challenge Book I, Osborne McGraw-Hill
  - Hacker's Challenge Book II, Osborne McGraw-Hill

What you will learn today

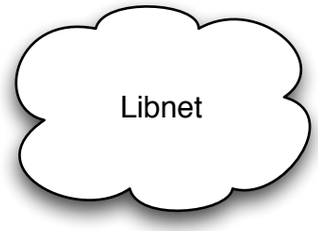
- The Libnet programming library
  - What it is, what it isn't
  - Common network tool techniques and how they are codified
  - How to employ Libnet to rapidly build useful network security tools

What you should already know

- The C programming language
- General understanding of the TCP/IP protocol suite
  - Primarily layers 1 – 3 (OSI layers 2 – 4)
- General network security concepts
  - For example; the difference between packet sniffing and port scanning



What is libnet?

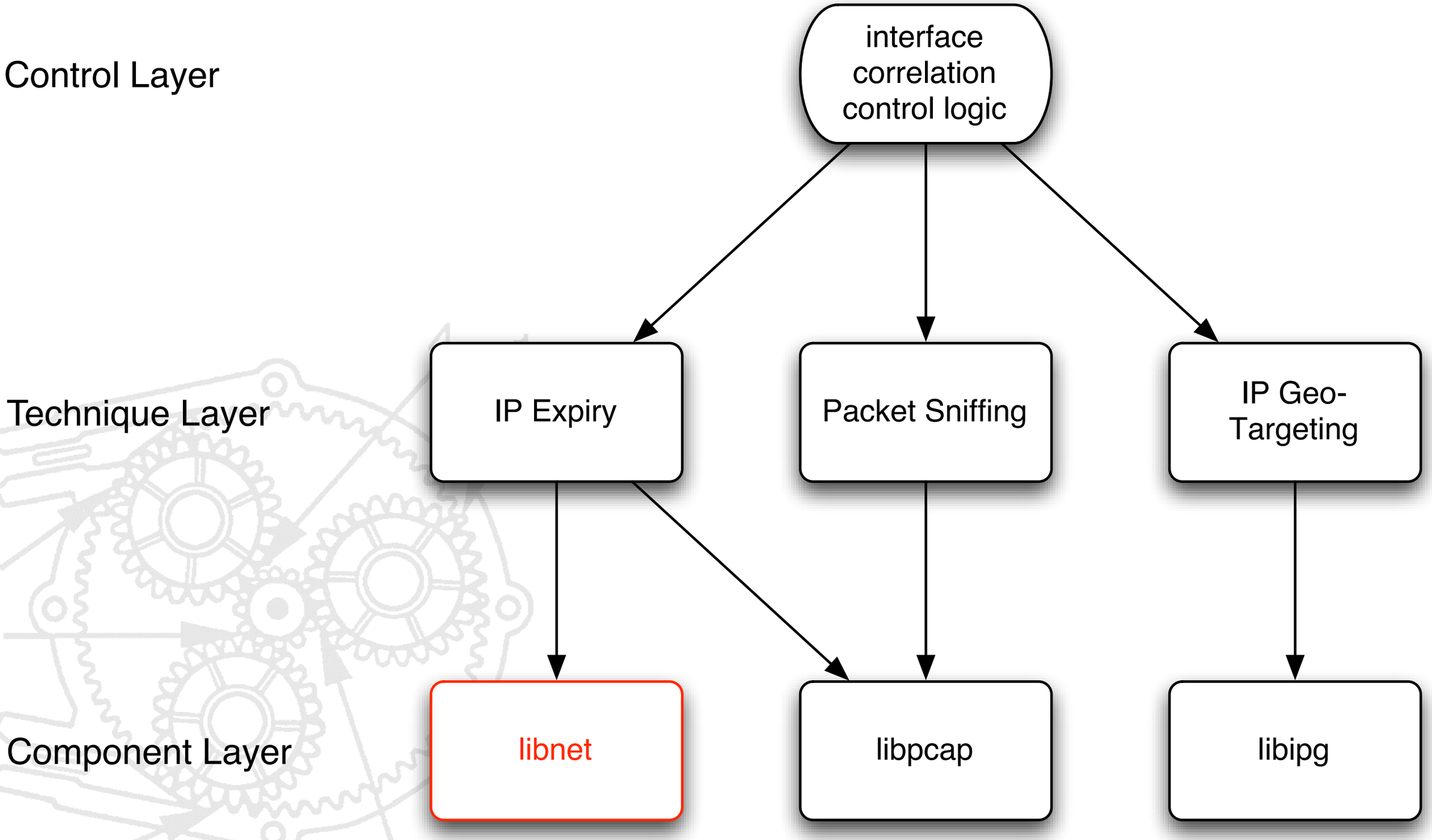


- A C Programming library for packet construction and injection
- The Yin to the Yang of libpcap
- Libnet's Primary Role in Life:
  - A simple interface for packet construction and injection
- Libnet IS good for:
  - Tools requiring meticulous control over every field of every header of every packet
- Libnet IS not well suited for:
  - Building client-server programs where the operating system should be doing most of the work

Components are building blocks

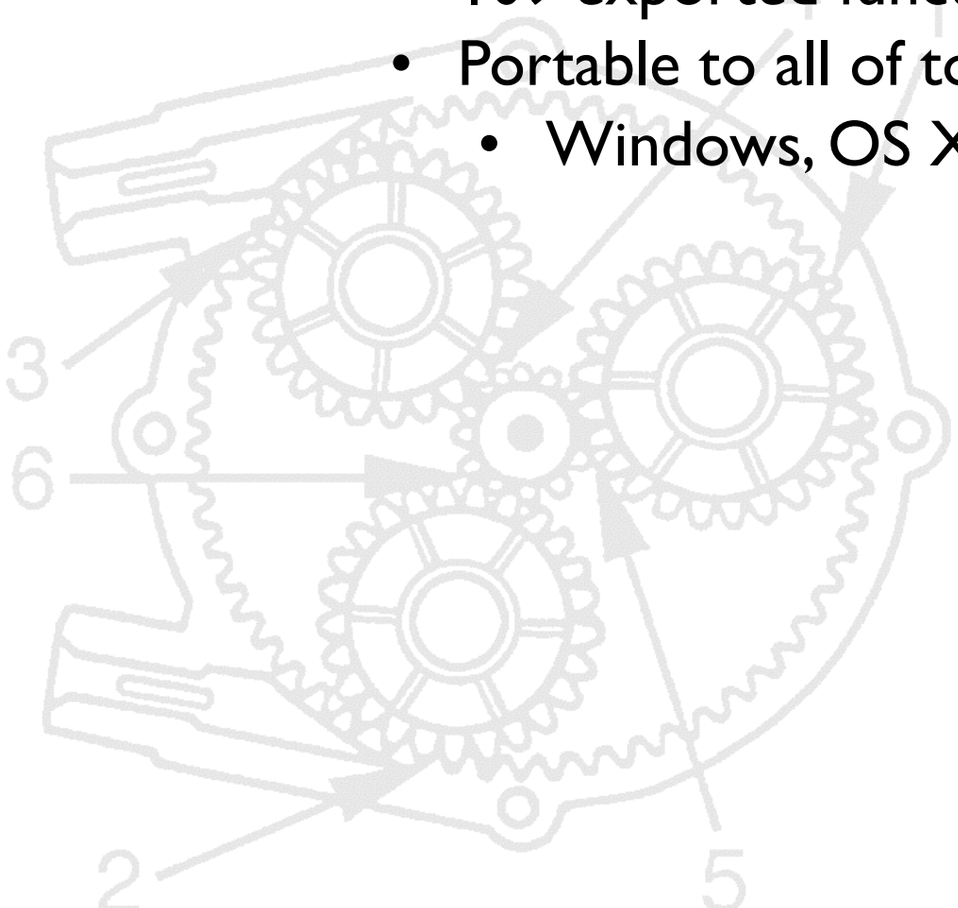
- Libnet is a component
  - That's nice. What is a component?
  - Before we define a component, let's talk about network security tools (toolkits are nice, but they're just enablers)
- A network security tool can be *modularized* and broken down into three layers: Control, Technique, Component
  - Eases conception and fast-tracks construction of tools
  - Control: high level abstract, delivery mechanism for techniques
  - Technique: answers "what does this tool do?"
  - Component: fundamental layer, answers "how does this tool do what it does?"
- Libnet is a fundamental building block used to create network security tools

The Modular Model of Network Security Tools



What's inside of libnet?

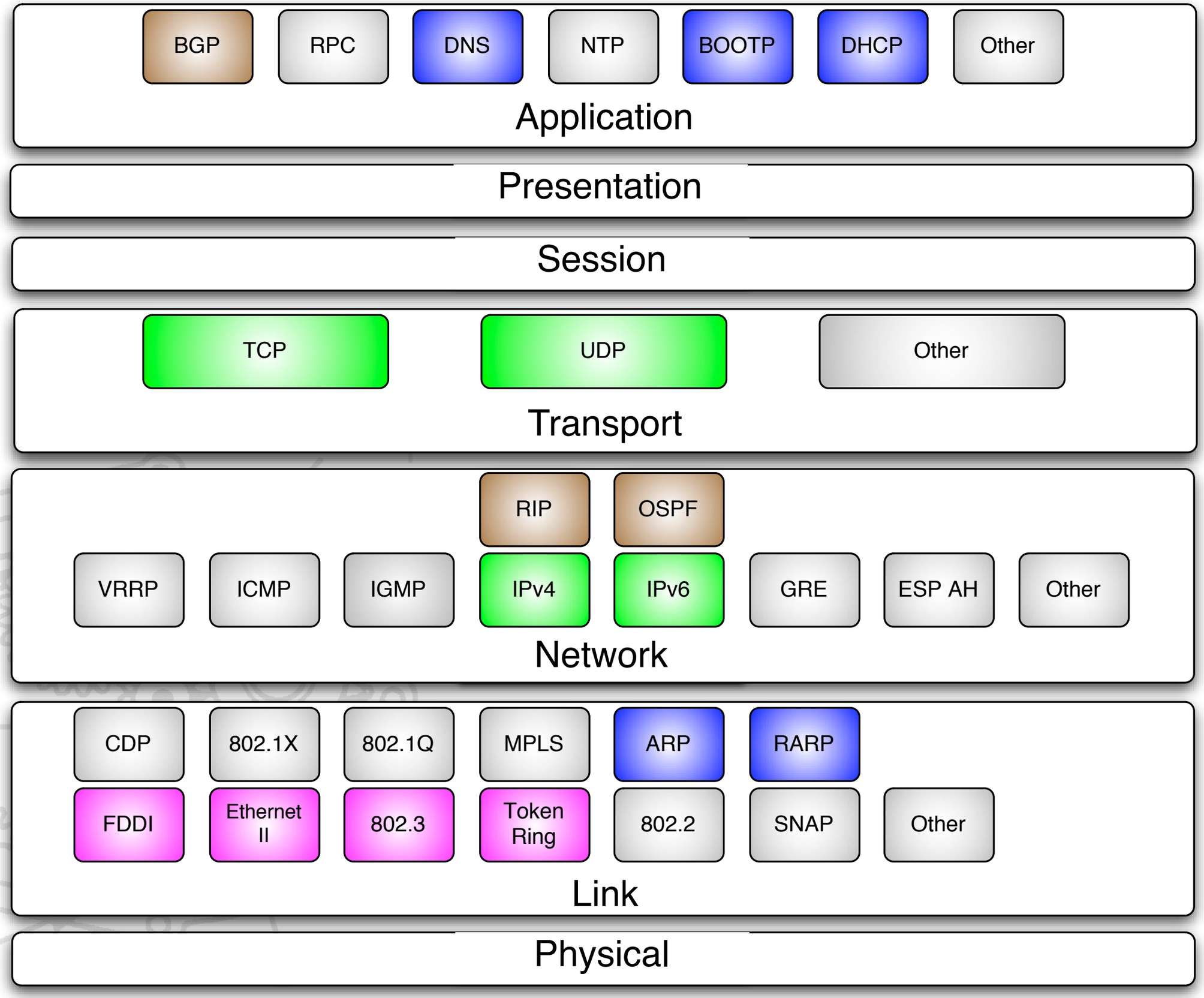
- As of libnet 1.1.2:
  - About 18,000 lines of C source code
  - 109 exported functions, 67 packet builder functions
  - Portable to all of today's hottest operating systems:
    - Windows, OS X, BSD, Linux, Solaris, HPUX



Why use libnet?

- Portability
  - Libnet is portable to all of our favorite and exquisitely cherished operating systems
- Ease of Use
  - As we will see, Libnet 1.1.x exports a braindead simple interface to building and injecting packets (4 easy steps)
- Robustness
  - Libnet supports all of today's in-demand protocols with more added all the time
    - More than 30 supported in Libnet 1.1.2 (see next slide)
    - Several link layers: Ethernet, Token Ring, FDDI, *802.11 planned*
- Open Source
  - Licensing
    - Libnet is released under a BSD license meaning it is basically free to use
  - Response-time in bug fixes
    - Large user-base; bugs are fixed quickly

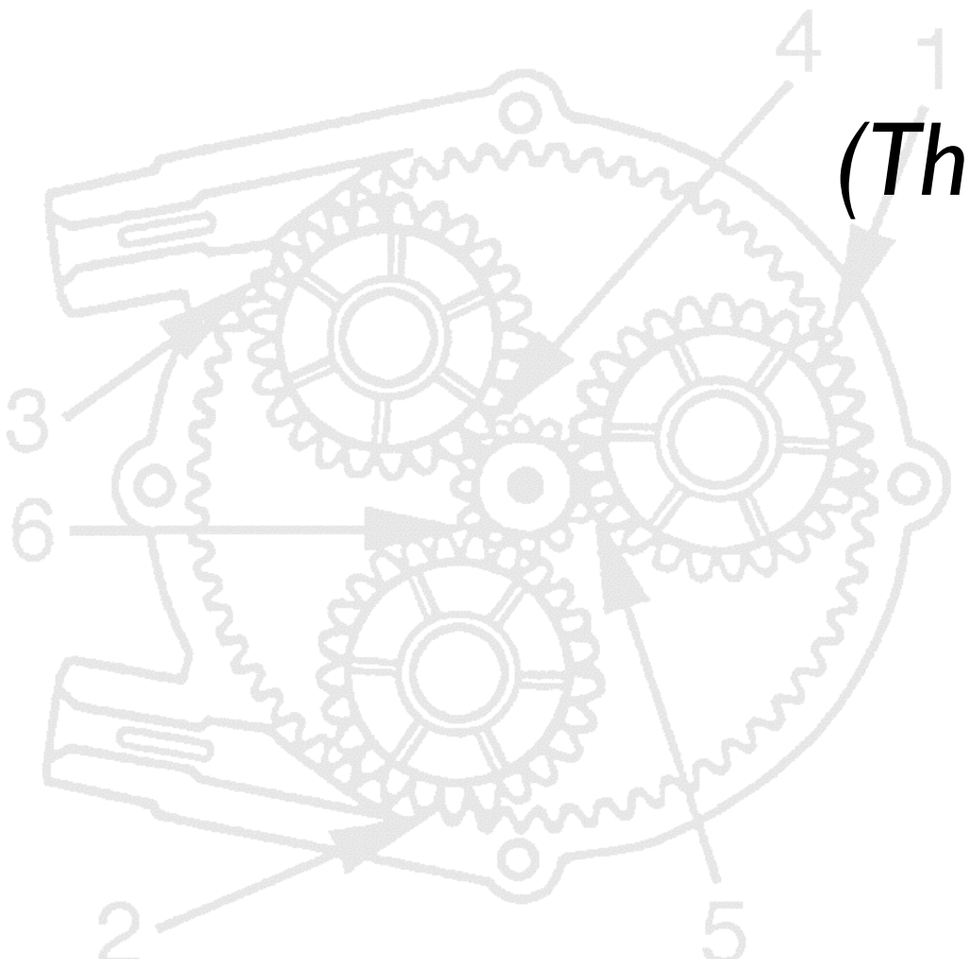
Libnet 1.1.x protocols



A brief history of Libnet

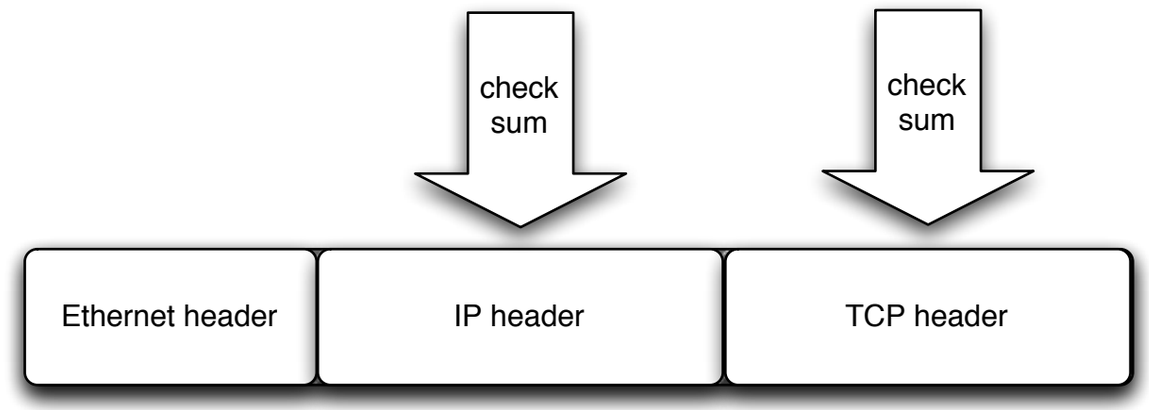
1998 - 2001

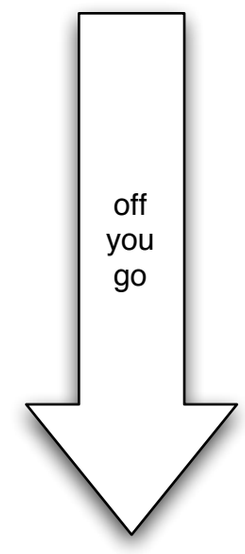
(The formative years)



Libnet 1.0.x process

```
libnet_init_packet(...);
libnet_open_link_interface(...);
libnet_build_ip(...);
libnet_build_ethernet(...);
libnet_build_tcp(...);
libnet_do_checksum(...);
libnet_do_checksum(...);
libnet_write_link_layer(...);
libnet_destroy_packet(...);
libnet_close_link_interface(...);
```





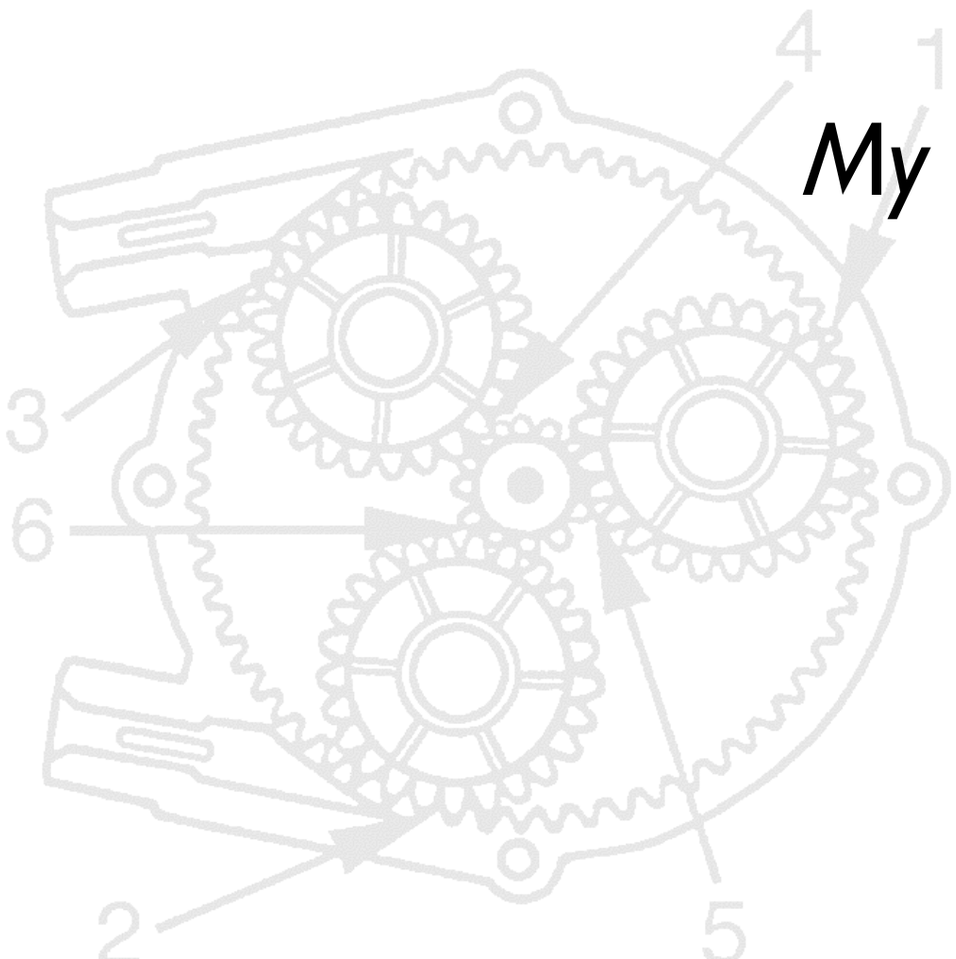
Libnet 1.0.x deficiencies

- Oh wow is that user-unfriendly
  - Too many steps in building a packet (up to 10!)
  - Too much to do, alot can go wrong
- No state maintenance
  - Couldn't track anything
- Over-reliance on the application programmer
  - Memory allocation / de-allocation
  - Memory offsets for packets
  - Checksums
- Libnet 1.1.x was designed to address all of these issues

Libnet 1.1.x

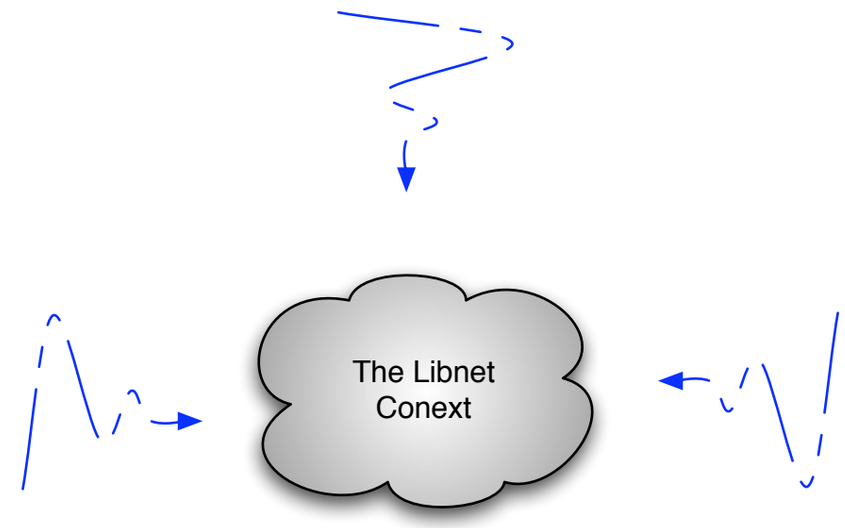
2001 - ...

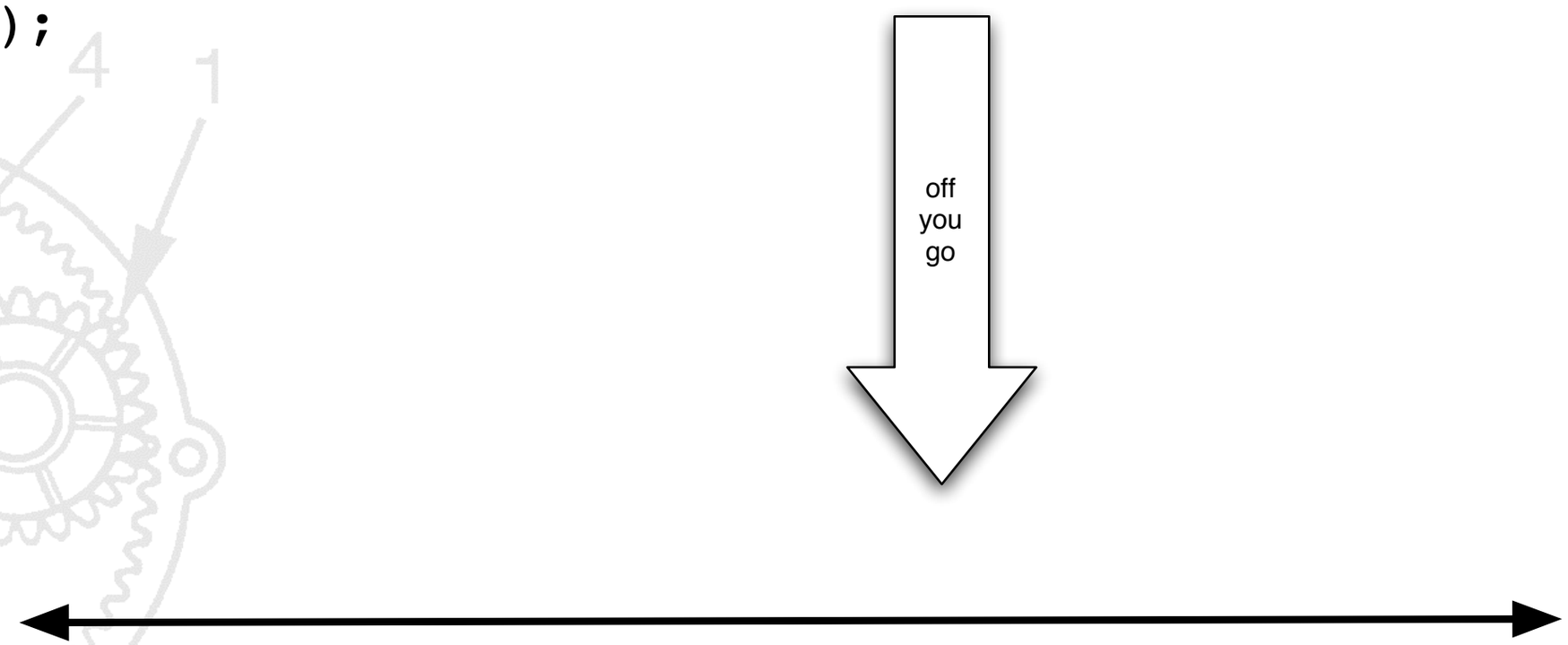
My boy's all grown's up

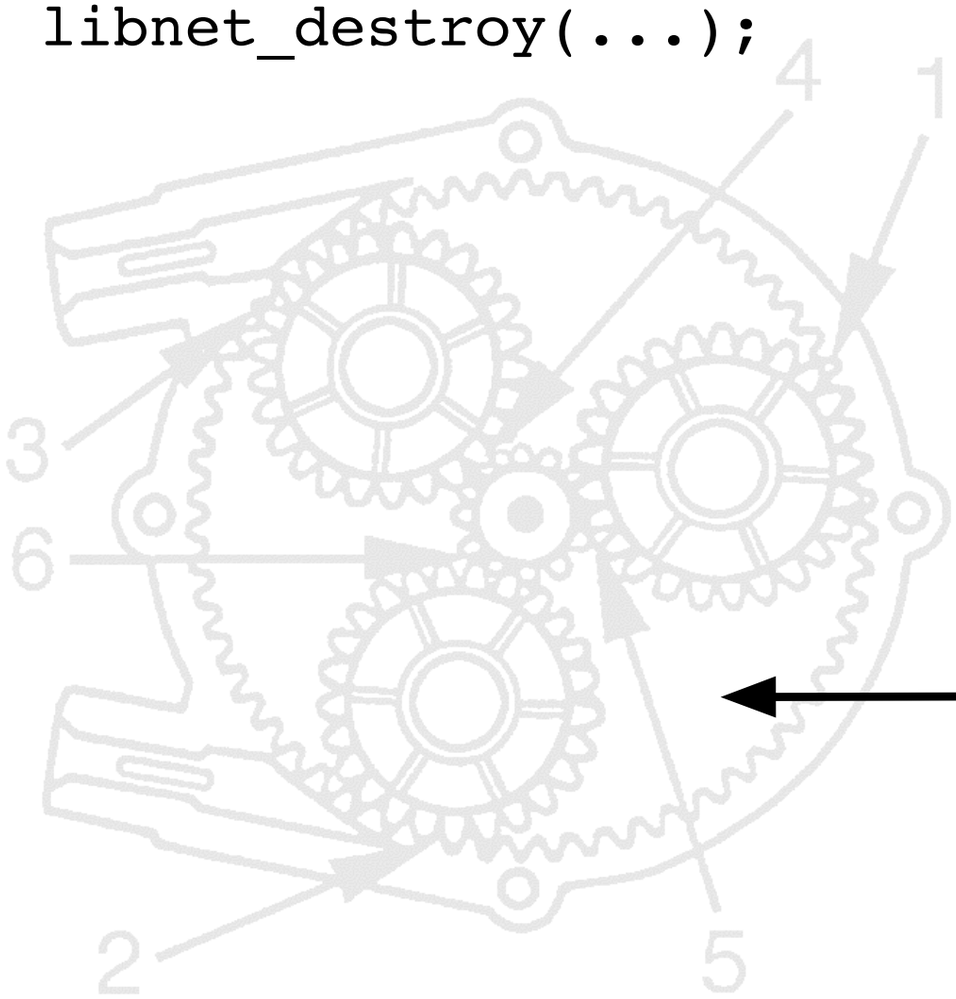


Libnet 1.1.2 process

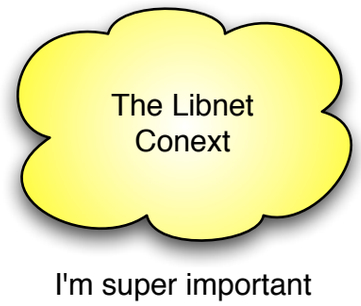
```
libnet_init(...);

libnet_build_tcp(...);

libnet_build_ipv4(...);

libnet_build_ethernet(...);

libnet_build_write(...);

libnet_destroy(...);
```





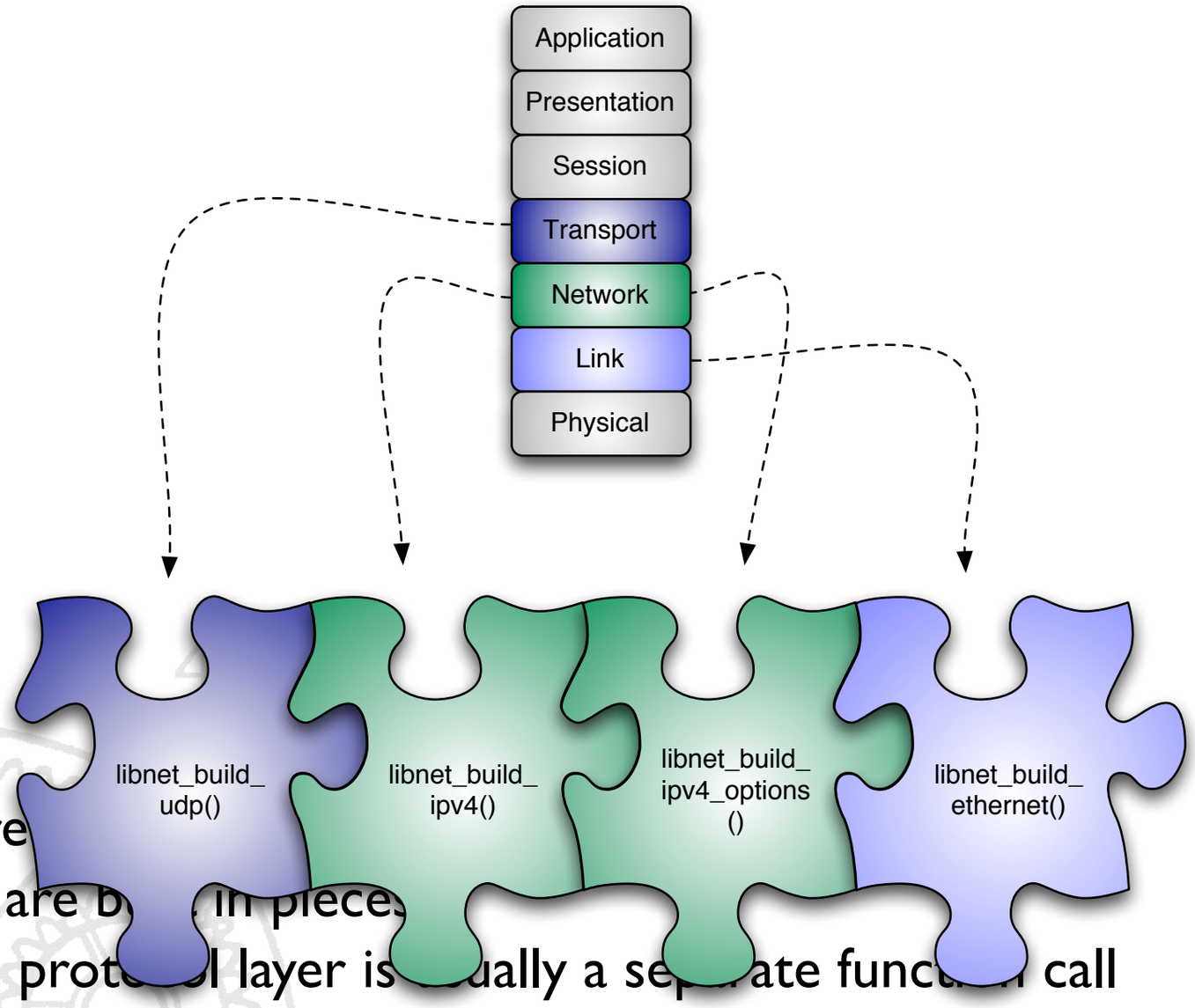


The libnet context



- Opaque monolithic data structure that is returned from `libnet_init();`
  - "`l`"
- Maintains state for the entire session
  - Tracks all memory usage and packet construction
  - Defines and describes a libnet session
- Used in almost every function
- (More detail later)

Packet construction

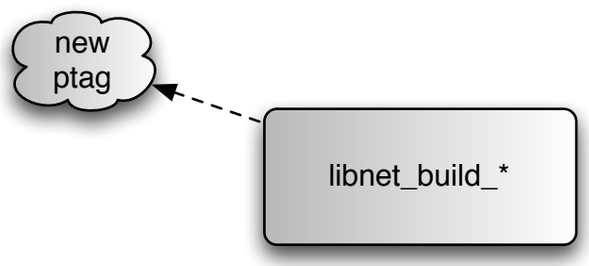


- The core
- Packets are b in piece
  - Each prot l layer is ually a se ate funct n call
    - Generally two - four function calls to build an entire packet
- Packet builders take arguments corresponding to header values
- Approximates an IP stack; must be called in order
  - From the highest on the OSI model to the lowest
- A successful call to a builder function returns a ptag

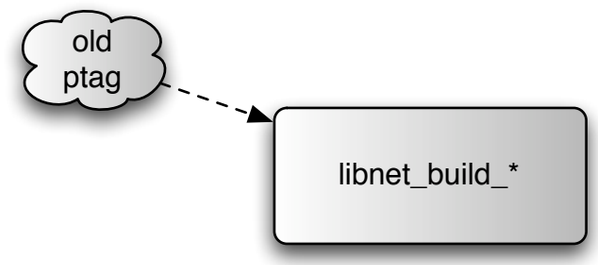
Packet construction

```
tcp = libnet_build_tcp(
      src_prt,                                          /* source port */
      dst_prt,                                          /* destination port */
      0x01010101,                                       /* sequence number */
      0x02020202,                                       /* acknowledgement num */
      TH_SYN,                                           /* control flags */
      32767,                                            /* window size */
      0,                                                /* checksum */
      0,                                                /* urgent pointer */
      LIBNET_TCP_H + payload_s,                         /* TCP packet size */
      payload,                                          /* payload */
      payload_s,                                        /* payload size */
      l,                                                /* context */
      0);                                               /* ptag */
```

Ptags and Pblocks



Creating a new protocol block; a new ptag is returned

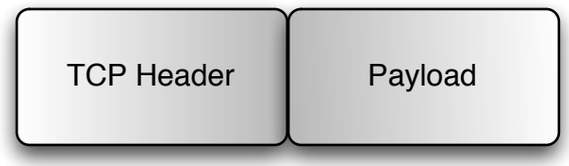


Modifying an existing protocol block; an old ptag is passed in

- Protocol Tag == ptag
- Protocol Block == pblock
- Protocol Tags (ptags) used to track Protocol Blocks (pblocks)
  - Whenever a new packet piece is built it is stored in a pblock and a new ptag is returned
  - Whenever an existing packet piece is modified, an old ptag is used
    - Looped packet updating
- Ptags are handled directly by the user, pblocks are not

```
tcp = libnet_build_tcp(
        src_prt,                          /* source port */
        dst_prt,                          /* destination port */
        0x01010101,                       /* sequence number */
        0x02020202,                       /* acknowledgement num */
        TH_SYN,                           /* control flags */
        32767,                            /* window size */
        0,                                /* checksum */
        0,                                /* urgent pointer */
        LIBNET_TCP_H + payload_s,         /* TCP packet size */
        payload,                          /* payload */
        payload_s,                        /* payload size */
        1,                                /* context */
        0);                               /* ptag */
```

The payload interface



- A simple interface to append arbitrary payloads to packets
  - TCP, UDP, ICMP, IP
- All packet builder functions support this interface
- Use is optional

```
tcp = libnet_build_tcp(
        src_prt,                                  /* source port */
        dst_prt,                                  /* destination port */
        0x01010101,                               /* sequence number */
        0x02020202,                               /* acknowledgement num */
        TH_SYN,                                   /* control flags */
        32767,                                    /* window size */
        0,                                        /* checksum */
        0,                                        /* urgent pointer */
        LIBNET_TCP_H + payload_s,               /* TCP packet size */
        payload,                                  /* payload */
        payload_s,                                /* payload size */
        1,                                        /* context */
        0);                                       /* ptag */
```

Wire injection methods

- Raw socket interface (less complex)
  - Mid-level interface, packets built at the IP layer and above
    - No link header needs to be built
  - Removes all routing and interface decisions
  - Useful for "legitimate" packet tools that do not need to spoof address information
  - Packet passes through kernel's IP stack
    - Routing, checksums, firewalls all an issue
  - Less than granular level of control (next slide)
- Link layer interface (more complex)
  - Low-level interface, packets built at the link layer
  - Packet does not pass through the kernel's IP stack
    - Sovereign control of every field of the packet
    - All address and routing information needs to be provided
    - Some operating systems stamp outgoing MAC address of the Ethernet header (this is bypassable)

Raw Socket Non-Sequitur

	IP Fragmentation	IP Total Length	IP Checksum	IP ID	IP Source	Max size before kernel complains
Linux 2.2+	Performed if packet is larger than MTU	Always filled in	Always filled in	Filled in if left 0	Filled in if left 0	1500 bytes
Solaris 2.6+	Performed if packet is larger than MTU; Sets DF bit		Always filled in			
OpenBSD 2.8+	Performed if packet is larger than MTU		Always filled in			

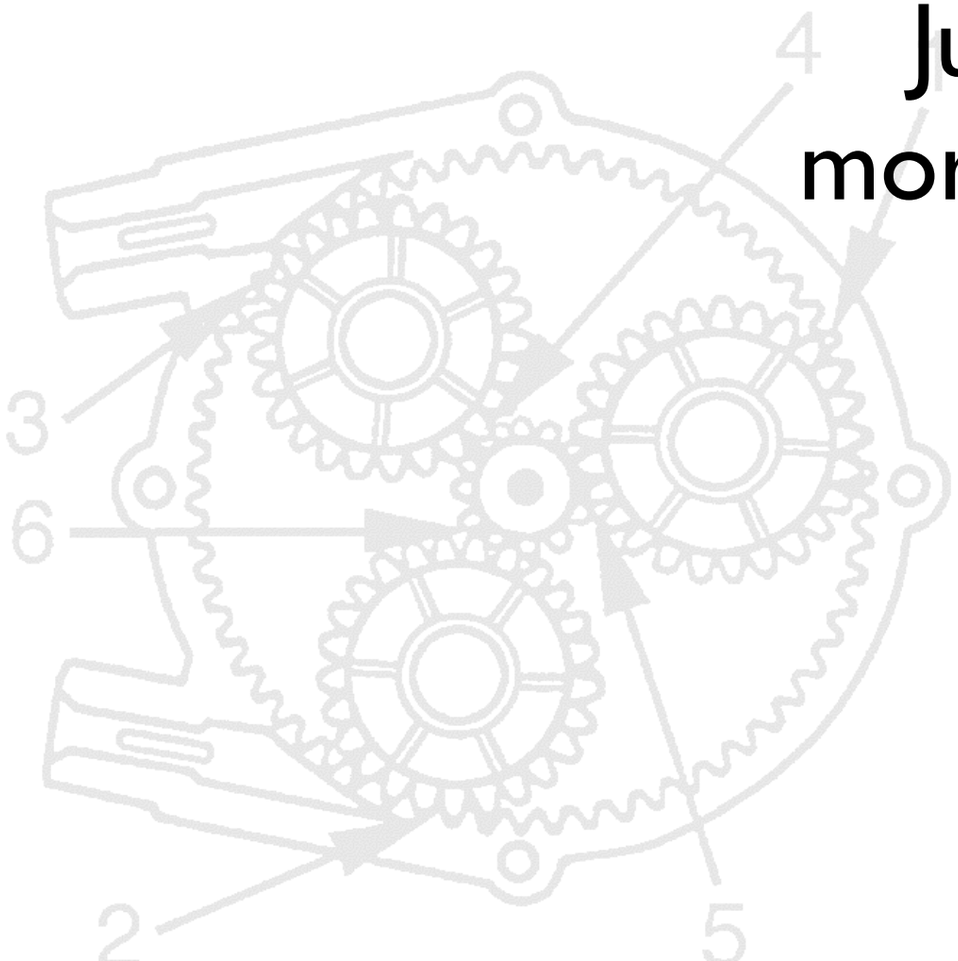
Packet checksums

- Programmer no longer has to worry about checksum computation
- Common usage: programmer specifies a “0”; libnet autocomputes
  - Can be toggled off to use checksum of “0”
- Alternative usage: programmer specifies value, libnet uses that
  - Useful for fuzzing, using pre-computed checksums

```
ip = libnet_build_ipv4(
     LIBNET_IPV4_H + LIBNET_TCP_H + payload_s,   /* length */
     0,                                          /* TOS */
     242,                                        /* IP ID */
     0,                                          /* IP frag */
     64,                                         /* TTL */
     IPPROTO_TCP,                                /* protocol */
     0,                                          /* checksum */
     src_ip,                                     /* source IP */
     dst_ip,                                     /* destination IP */
     NULL,                                       /* payload */
     0,                                          /* payload size */
     1,                                          /* context */
     0);                                         /* ptag */
```

Libnet 1.1.x Functions

Just some of the
more important ones



Initialization

```
libnet_t *
libnet_init(int injection_type, char *device, char *err_buf);
```

Initializes the libnet library and create the environment	
SUCCESS	A libnet context suitable for use
FAILURE	NULL, err_buf will contain the reason
injection_type	LIBNET_LINK, LIBNET_RAW4
device	“fxp0”, “192.168.0.1”, NULL
err_buf	Error message if function fails

```
l = libnet_init(LIBNET_LINK, “fxp0”, err_buf);
if (l == NULL)
{
    fprintf(stderr, “libnet_init(): %s”, errbuf);
}
```

Device (interface) selection

- Happens during initialization
- `libnet_init(LIBNET_LINK, “fxp0”, errbuf);`
  - Will initialize libnet’s link interface using the `fxp0` device
- `libnet_init(LIBNET_LINK, “192.168.0.1”, errbuf);`
  - Will initialize libnet’s link interface using the device with the IP address `192.168.0.1`
- `libnet_init(LIBNET_LINK, NULL, errbuf);`
  - Will initialize libnet’s link interface using the first “up” device it can find
  - `libnet_getdevice(l);`
- `libnet_init(LIBNET_RAW4, NULL, errbuf);`
  - Under the Raw socket interface no device is selected
    - Exception: Win32 does this internally since it is built on top of Winpcap
- New: devices with no IP address can be specified for use (stealth)

Error handling

```
char *
libnet_geterror(libnet_t *l);
```

Returns the last error message generated by libnet	
SUCCESS	An error string, NULL if none occurred
FAILURE	This function cannot fail
l	The libnet context pointer

```
l = libnet_autobuild_ipv4(len, IPPROTO_TCP, dst, l);
if (l == NULL)
{
    fprintf(stderr, “libnet_autobuild_ipv4(): %s”,
        libnet_geterror(l));
}
```

Address resolution

```
u_int32_t
libnet_name2addr4(libnet_t *l, char *host_name, u_int8_t use_name);
```

Converts a IPv4 presentation format hostname into a big endian ordered IP number	
SUCCESS	An IP number suitable for use with libnet_build_*
FAILURE	-1, which is technically "255.255.255.255"
l	The libnet context pointer
host_name	The presentation format address
use_name	LIBNET_REOLVE, LIBNET_DONT_RESOLVE

```
dst = libnet_name2addr4(l, argv[optind], LIBNET_DONT_RESOLVE);
if (dst == -1)
{
    fprintf(stderr, "libnet_name2addr4(): %s", libnet_geterror(l));
}
```

Address resolution

```
char *
libnet_addr2name4(u_int32_t address, u_int8_t use_name);
```

Converts a big endian ordered IPv4 address into a presentation format address	
SUCCESS	A string of dots and decimals or a hostname
FAILURE	This function cannot fail
address	The IPv4 address
use_name	LIBNET_REOLVE, LIBNET_DONT_RESOLVE

```
printf("%s\n", libnet_addr2name4(i, LIBNET_DONT_RESOLVE));
```

Packet construction: UDP

```
libnet_ptag_t
libnet_build_udp(u_int16_t sp, u_int16_t dp, u_int16_t len,
u_int16_t sum, u_int8_t *payload, u_int32_t payload_s, libnet_t *l,
libnet_ptag_t ptag);
```

Builds a UDP header	
SUCCESS	A ptag referring to the UDP packet
FAILURE	`-1`, and `libnet_get_error()` can tell you why
`sp`	The source UDP port
`dp`	The destination UDP port
`len`	Length of the UDP packet (including payload)
`sum`	Checksum, 0 for libnet to autofill
`payload`	Optional payload
`payload_s`	Payload size
`l`	The libnet context pointer
`ptag`	Protocol tag

Packet construction: IPv4

```
libnet_ptag_t
libnet_build_ipv4(u_int16_t len, u_int8_t tos, u_int16_t id,
u_int16_t frag, u_int8_t ttl, u_int8_t prot, u_int16_t sum,
u_int32_t src, u_int32_t dst, u_int8_t *payload,
u_int32_t payload_s, libnet_t *l, libnet_ptag_t ptag);
```

Builds an IPv4 header	
SUCCESS	A ptag referring to the IPv4 packet
FAILURE	`-1`, and `libnet_get_error()` can tell you why
`len`	Length of the IPv4 packet (including payload)
`tos`	Type of service bits
`id`	IP identification
`frag`	Fragmentation bits
`ttl`	Time to live
`prot`	Upper layer protocol
`sum`	Checksum, 0 for libnet to autofill
`src`	Source IP address

Packet construction: IPv4

```
libnet_ptag_t
libnet_build_ipv4(u_int16_t len, u_int8_t tos, u_int16_t id,
u_int16_t frag, u_int8_t ttl, u_int8_t prot, u_int16_t sum,
u_int32_t src, u_int32_t dst, u_int8_t *payload,
u_int32_t payload_s, libnet_t *l, libnet_ptag_t ptag);
```

Builds an IPv4 header	
SUCCESS	A ptag referring to the UDP packet
FAILURE	`-1`, and `libnet_get_error()` can tell you why
`dst`	Destination IP address
`payload`	Optional payload
`payload_s`	Payload size
`l`	The libnet context pointer
`ptag`	Protocol tag

Packet construction: Ethernet

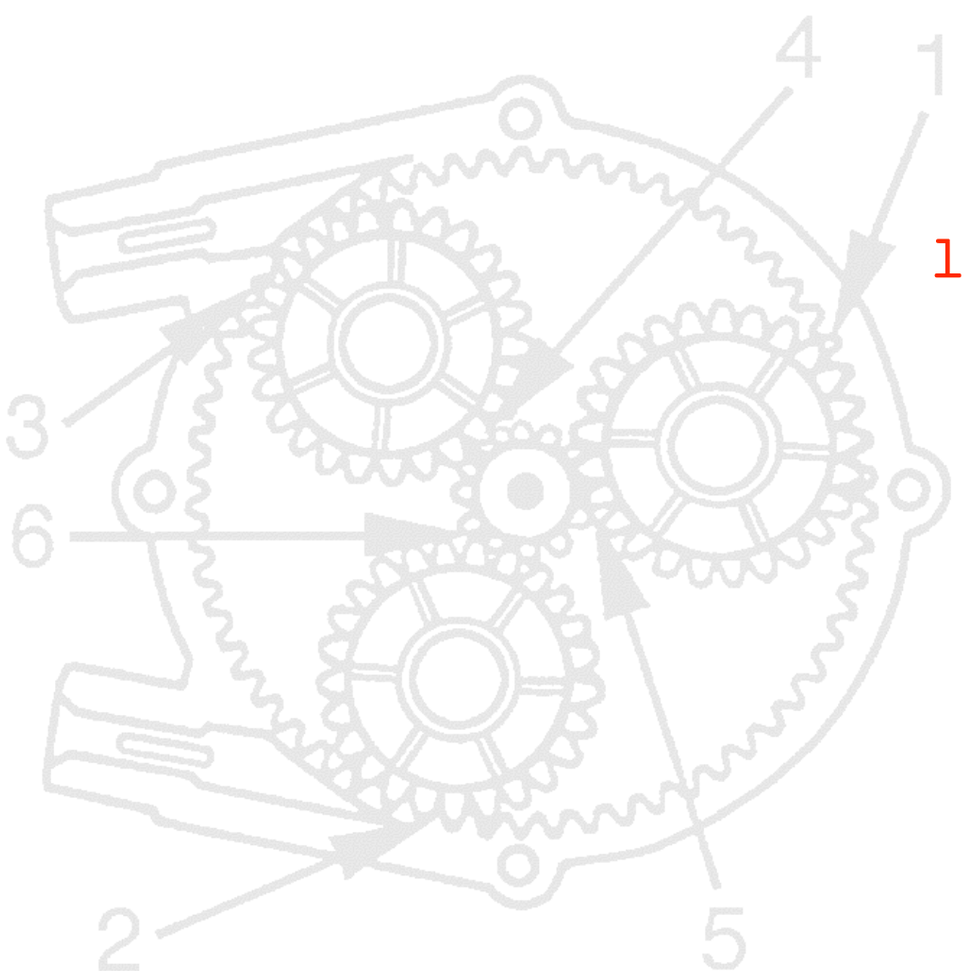
```
libnet_ptag_t
libnet_build_ethernet(u_int8_t *dst, u_int8_t *src,
u_int16_t type, u_int8_t *payload, u_int32_t payload_s, libnet_t *l,
libnet_ptag_t ptag);
```

Builds an Ethernet header	
SUCCESS	A ptag referring to the Ethernet frame
FAILURE	`-1`, and `libnet_get_error()` can tell you why
`dst`	Destination ethernet address
`src`	Source ethernet address
`type`	Upper layer protocol type
`payload`	Optional payload
`payload_s`	Payload size
`l`	The libnet context pointer
`ptag`	Protocol tag

Shutdown

```
void
libnet_destroy(libnet_t *l);
```

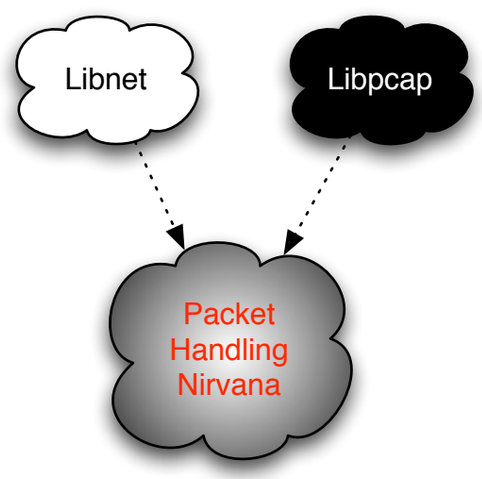
Shuts down the libnet environment	
l	The libnet context pointer




```
libnet_destroy(l);
```

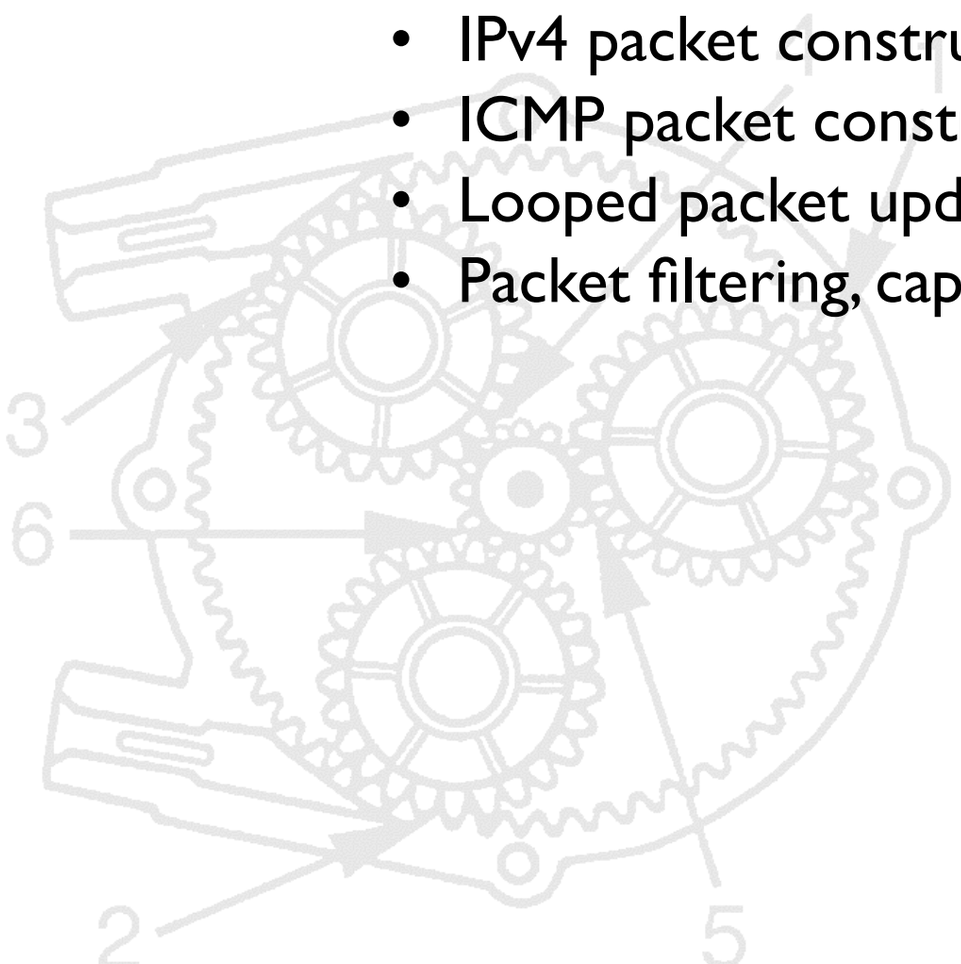
Libnet with other components

GNIP: *A poor man's ping*



A simple application

- Simple ping client
- 250 lines of source
- Illustrates some of libnet's (and libpcap's) core concepts
  - IPv4 packet construction
  - ICMP packet construction
  - Looped packet updating
  - Packet filtering, capturing and dissection



Libpcap packet filter (same as tcpdump)

Monolithic context variables

Side effect of closed interface

```
#include <libnet.h>
#include <pcap.h>

#define GNIP_FILTER "icmp[0] = 0"

void usage(char *);

int
main(int argc, char **argv)
{
    libnet_t *l = NULL;
    pcap_t *p = NULL;
    u_int8_t *packet;
    u_int32_t dst_ip, src_ip;
    u_int16_t id, seq, count;
    int c, interval = 0, pcap_fd, timed_out;
    u_int8_t loop, *payload = NULL;
    u_int32_t payload_s = 0;
    libnet_ptag_t icmp = 0, ip = 0;
    char *device = NULL;
    fd_set read_set;
    struct pcap_pkthdr pc_hdr;
    struct timeval timeout;
    struct bpf_program filter_code;
    bpf_u_int32 local_net, netmask;
    struct libnet_ipv4_hdr *ip_hdr;
    struct libnet_icmpv4_hdr *icmp_hdr;
    char errbuf[LIBNET_ERRBUF_SIZE];

    while((c = getopt(argc, argv, "I:i:c:")) != EOF)
    {
        switch (c)
        {
            case 'I':
                device = optarg;
                break;
            case 'i':
                interval = atoi(optarg);
                break;
            case 'c':
                count = atoi(optarg);
                break;
        }
    }

    c = argc - optind;
    if (c != 1)
    {
        usage(argv[0]);
        exit(EXIT_FAILURE);
    }
```

Libnet Phase One

Libnet context

Pcap context

Setup pcap filter
(ICMP ECHO only)

Resolve IP address

```
/* initialize the libnet library */
l = libnet_init(LIBNET_RAW4, device, errbuf);
if (l == NULL)
{
    fprintf(stderr, "libnet_init() failed: %s", errbuf);
    exit(EXIT_FAILURE);
}

if (device == NULL)
{
    device = pcap_lookupdev(errbuf);
    if (device == NULL)
    {
        fprintf(stderr, "pcap_lookupdev() failed: %s\n", errbuf);
        goto bad;
   }
}

/* handcrank pcap */
p = pcap_open_live(device, 256, 0, 0, errbuf);
if (p == NULL)
{
    fprintf(stderr, "pcap_open_live() failed: %s", errbuf);
    goto bad;
}

/* get the subnet mask of the interface */
if (pcap_lookupnet(device, &local_net, &netmask, errbuf) == -1)
{
    fprintf(stderr, "pcap_lookupnet(): %s", errbuf);
    goto bad;
}

/* compile the BPF filter code */
if (pcap_compile(p, &filter_code, GNIP_FILTER, 1, netmask) == -1)
{
    fprintf(stderr, "pcap_compile(): %s", pcap_geterr(p));
    goto bad;
}

/* apply the filter to the interface */
if (pcap_setfilter(p, &filter_code) == -1)
{
    fprintf(stderr, "pcap_setfilter(): %s", pcap_geterr(p));
    goto bad;
}

dst_ip = libnet_name2addr4(l, argv[optind], LIBNET_RESOLVE);
if (dst_ip == -1)
{
    fprintf(stderr, "Bad destination IP address (%s).\n",
        libnet_geterror(l));
    goto bad;
}
```

Get source IP address

```
src_ip = libnet_get_ipaddr4(l);
if (src_ip == -1)
{
    fprintf(stderr, "Can't determine source IP address (%s).\n",
            libnet_geterror(l));
    goto bad;
}

interval ? interval : interval = 1;
timeout.tv_sec = interval;
timeout.tv_usec = 0;
pcap_fd = pcap_fileno(p);

fprintf(stderr, "GNIP %s (%s): %d data bytes\n",
        libnet_addr2name4(dst_ip, 1), libnet_addr2name4(dst_ip, 0),
        LIBNET_IPV4_H + LIBNET_ICMPV4_ECHO_H + payload_s);
```

Libnet Phase Two

Important: Note ptag usage!

Libnet Phase Three

```
loop = 1;
for (id = getpid(), seq = 0, icmp = LIBNET_PTAG_INITIALIZER; loop; seq++)
{
    icmp = libnet_build_icmpv4_echo(
        ICMP_ECHO,                              /* type */
        0,                                      /* code */
        0,                                      /* checksum */
        id,                                     /* id */
        seq,                                    /* sequence number */
        payload,                                /* payload */
        payload_s,                              /* payload size */
        l,                                      /* libnet context */
        icmp);                                  /* ptag */
    if (icmp == -1)
    {
        fprintf(stderr, "Can't build ICMP header: %s\n",
            libnet_geterror(l));
        goto bad;
    }

    ip = libnet_build_ipv4(
        LIBNET_IPV4_H + LIBNET_ICMPV4_ECHO_H + payload_s, /* length */
        0,                                      /* TOS */
        id,                                     /* IP ID */
        0,                                      /* IP Frag */
        64,                                     /* TTL */
        IPPROTO_ICMP,                           /* protocol */
        0,                                      /* checksum */
        src_ip,                                 /* source IP */
        dst_ip,                                 /* destination IP */
        NULL,                                   /* payload */
        0,                                      /* payload size */
        l,                                      /* libnet context */
        ip);                                     /* ptag */
    if (ip == -1)
    {
        fprintf(stderr, "Can't build IP header: %s\n", libnet_geterror(l));
        goto bad;
    }

    c = libnet_write(l);
    if (c == -1)
    {
        fprintf(stderr, "Write error: %s\n", libnet_geterror(l));
        goto bad;
    }
```

Interface multiplexing

"Is this a response" logic

Libnet Phase Four

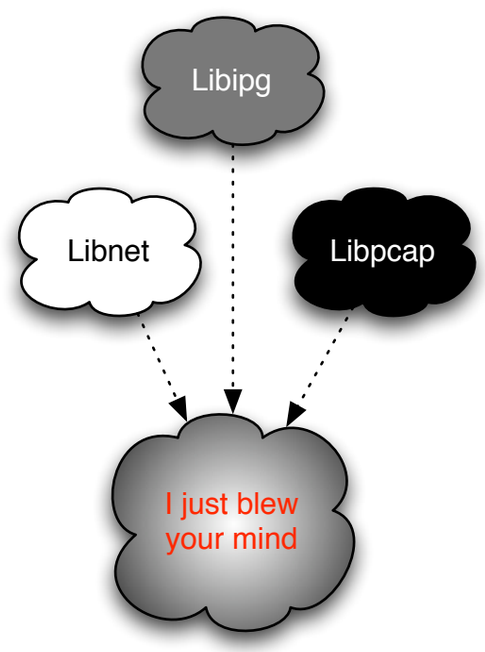
```
FD_ZERO(&read_set);
FD_SET(pcap_fd, &read_set);

for (timed_out = 0; !timed_out && loop; )
{
    c = select(pcap_fd + 1, &read_set, 0, 0, &timeout);
    switch (c)
    {
        case -1:
            fprintf(stderr, "select() %s\n", strerror(errno));
            goto bad;
        case 0:
            timed_out = 1;
            continue;
        default:
            if (FD_ISSET(pcap_fd, &read_set) == 0)
            {
                timed_out = 1;
                continue;
            }
            /* fall through to read the packet */
    }
    packet = (u_int8_t *)pcap_next(p, &pc_hdr);
    if (packet == NULL)
    {
        continue;
    }

    ip_hdr = (struct libnet_ipv4_hdr *)(packet + 14);
    icmp_hdr = (struct libnet_icmpv4_hdr *)(packet + 14 +
            (ip_hdr->ip_hl << 2));
    if (ip_hdr->ip_src.s_addr != dst_ip)
    {
        continue;
    }
    if (icmp_hdr->icmp_id == id)
    {
        fprintf(stderr, "%d bytes from %s: icmp_seq=%d ttl=%d\n",
            ntohs(ip_hdr->ip_len),
            libnet_addr2name4(ip_hdr->ip_src.s_addr, 0),
            icmp_hdr->icmp_seq, ip_hdr->ip_ttl);
    }
  }
}

libnet_destroy(l);
pcap_close(p);
return (EXIT_SUCCESS);
```

GNIP output

```
[rounder:Projects/misc/] root# ./gnip 4.2.2.2
GNIP vnsc-bak.sys.gtei.net (4.2.2.2): 28 data bytes
28 bytes from 4.2.2.2: icmp_seq=0 ttl=247
28 bytes from 4.2.2.2: icmp_seq=1 ttl=247
28 bytes from 4.2.2.2: icmp_seq=2 ttl=247
28 bytes from 4.2.2.2: icmp_seq=3 ttl=247
28 bytes from 4.2.2.2: icmp_seq=4 ttl=247
^C
```

Libnet with other components

TRIG: *A rich man's traceroute*



A simple application

- Simple traceroute client with geo-targeting of IP addresses
- 280 lines of source
- Illustrates more of libnet's, libpcap's) core concepts
  - IPv4 packet construction
  - ICMP packet construction
  - Looped packet updating
  - IP geo-targeting
  - Packet filtering, capturing and dissection

Monolithic context variables

Side effect of closed interface

```
#include <libnet.h>
#include <pcap.h>
#include "./libipg.h"

int
do_lookup(u_int32_t ipn, ipgeo_t *ipg);

u_int8_t do_cc, do_country, do_city, do_region, do_isp, do_lat, do_long;

int main(int argc, char **argv)
{
    pcap_t *p = NULL;
    libnet_t *l = NULL;
    ipgeo_t *ipg = NULL;
    time_t start;
    u_char *packet;
    int c, ttl, done;
    char *device = NULL;
    extern char *optarg;
    extern int optind;
    struct pcap_pkthdr ph;
    libnet_ptag_t icmp, ip;
    u_int32_t dst_ip;
    struct libnet_icmpv4_hdr *icmp_h;
    struct libnet_ipv4_hdr *ip_h, *oip_h;
    char errbuf[LIBNET_ERRBUF_SIZE];

    printf("Trig 1.0 [geo-targeting traceroute scanner]\n");
    do_cc = do_country = do_city = do_region = do_isp = do_lat = do_long = 0;
    while ((c = getopt(argc, argv, "i:CcyrsLl")) != EOF)
    {
        switch (c)
        {
            case 'i':
                device = optarg;
                break;
            case 'C':
                do_cc = 1;
                break;
            case 'c':
                do_country = 1;
                break;
            case 'y':
                do_city = 1;
                break;
            case 'L':
                do_lat = 1;
                break;
            case 'l':
                do_long = 1;
                break;
            case 'r':
                do_region = 1;
                break;
```

Libnet Phase One

Libnet context

Pcap context

Libipg context

Resolve IP address

```
        case 's':
            do_isp = 1;
            break;
    }
}

c = argc - optind;
if (c != 2)
{
    fprintf(stderr, "usage:\t%s\t\t [-i interface][-Ccyrs] host file\n",
            argv[0]);
    goto done;
}

if (do_cc == 0 && do_country == 0 && do_city == 0 && do_region == 0 &&
    do_isp == 0 && do_lat == 0 && do_long == 0)
{
    printf("No IP geo-targeting?\n");
}

l = libnet_init(LIBNET_RAW4, NULL, errbuf);
if (l == NULL)
{
    fprintf(stderr, "libnet: %s\n", errbuf);
    return (EXIT_FAILURE);
}

p = pcap_open_live(device, 60, 0, 500, errbuf);
if (p == NULL)
{
    fprintf(stderr, "pcap: %s\n", errbuf);
    return (EXIT_FAILURE);
}

ipg = ipgeo_init(argv[optind + 1], 0, errbuf);
if (ipg == NULL)
{
    fprintf(stderr, "ipgeo: %s\n", errbuf);
    return (EXIT_FAILURE);
}

dst_ip = libnet_name2addr4(l, argv[optind], LIBNET_RESOLVE);
if (dst_ip == 0)
{
    fprintf(stderr, "libnet: %s\n", libnet_geterror(l));
    goto done;
}
```

Libnet Phase Two

Important: Note ptag usage!

Libnet Phase Three

```
for (done = icmp = ip = 0, ttl = 1; ttl < 31 && !done; ttl++)
{
    icmp = libnet_build_icmpv4_echo(
        ICMP_ECHO,                                  /* type */
        0,                                          /* code */
        0,                                          /* checksum */
        242,                                        /* id */
        ttl,                                        /* sequence */
        NULL,                                       /* payload */
        0,                                          /* payload size */
        l,                                          /* libnet context */
        icmp);                                      /* libnet id */
    if (icmp == -1)
    {
        fprintf(stderr, "libnet: %s\n", libnet_geterror(l));
        return (EXIT_FAILURE);
    }
    ip = libnet_build_ipv4(
        LIBNET_IPV4_H + LIBNET_ICMPV4_ECHO_H,       /* length */
        0,                                          /* TOS */
        242,                                        /* IP ID */
        0,                                          /* IP Frag */
        ttl,                                        /* TTL */
        IPPROTO_ICMP,                               /* protocol */
        0,                                          /* checksum */
        libnet_get_ipaddr4(l),                      /* src ip */
        dst_ip,                                     /* dst ip */
        NULL,                                       /* payload */
        0,                                          /* payload size */
        l,                                          /* libnet context */
        ip);                                        /* libnet id */
    if (ip == -1)
    {
        fprintf(stderr, "libnet: %s\n", libnet_geterror(l));
        return (EXIT_FAILURE);
    }
    c = libnet_write(l);
    if (c == -1)
    {
        fprintf(stderr, "libnet: %s\n", libnet_geterror(l));
        return (EXIT_FAILURE);
    }

    fprintf(stderr, "%02d: ", ttl);
```

"Is this a response" logic

```
/* read loop */
for (start = time(NULL); (time(NULL) - start) < 2; )
{
    packet = (u_char *)pcap_next(p, &ph);
    if (packet == NULL)
    {
        continue;
    }
    /* assume ethernet here for simplicity */
    ip_h = (struct libnet_ipv4_hdr *)(packet + 14);
    if (ip_h->ip_p == IPPROTO_ICMP)
    {
        icmp_h = (struct libnet_icmpv4_hdr *)(packet + 34);
        /* expired in transit */
        if (icmp_h->icmp_type == ICMP_TIMXCEED &&
            icmp_h->icmp_code == ICMP_TIMXCEED_INTRANS)
        {
            oip_h = (struct libnet_ipv4_hdr *)(packet + 42);
            if (oip_h->ip_id == htons(242))
            {
                fprintf(stderr, "%s ",
                    libnet_addr2name4(ip_h->ip_src.s_addr, 0));
                if (do_lookup(ip_h->ip_src.s_addr, ipg) == -1)
                {
                    fprintf(stderr, "ipgeo: %s\n", ipgeo_geterror(ipg));
                }
                break;
            }
        }
        /* terminal response */
        if (icmp_h->icmp_type == ICMP_ECHOREPLY)
        {
            if (icmp_h->icmp_id == 242 && icmp_h->icmp_seq == ttl)
            {
                fprintf(stderr, "%s ",
                    libnet_addr2name4(ip_h->ip_src.s_addr, 0));
                if (do_lookup(ip_h->ip_src.s_addr, ipg) == -1)
                {
                    fprintf(stderr, "ipgeo: %s\n", ipgeo_geterror(ipg));
                }
                done = 1;
                break;
            }
        }
    }
}
}
```

```
int
do_lookup(u_int32_t ipn, ipgeo_t *ipg)
{
    if (ipgeo_lookup(ipn, 0, ipg) == -1)
    {
        return (-1);
    }
    if (do_cc)
    {
        fprintf(stderr, "%s ", ipgeo_get_cc(ipg));
    }
    if (do_country)
    {
        fprintf(stderr, "%s ", ipgeo_get_country(ipg));
    }
    if (do_city)
    {
        fprintf(stderr, "%s ", ipgeo_get_city(ipg));
    }
    if (do_region)
    {
        fprintf(stderr, "%s ", ipgeo_get_region(ipg));
    }
    if (do_isp)
    {
        fprintf(stderr, "%s ", ipgeo_get_isp(ipg));
    }
    if (do_lat)
    {
        fprintf(stderr, "%.4f ", ipgeo_get_lat(ipg));
    }
    if (do_long)
    {
        fprintf(stderr, "%.4f ", ipgeo_get_long(ipg));
    }
    fprintf(stderr, "\n");
    return (1);
}
```

Libnet Phase Four

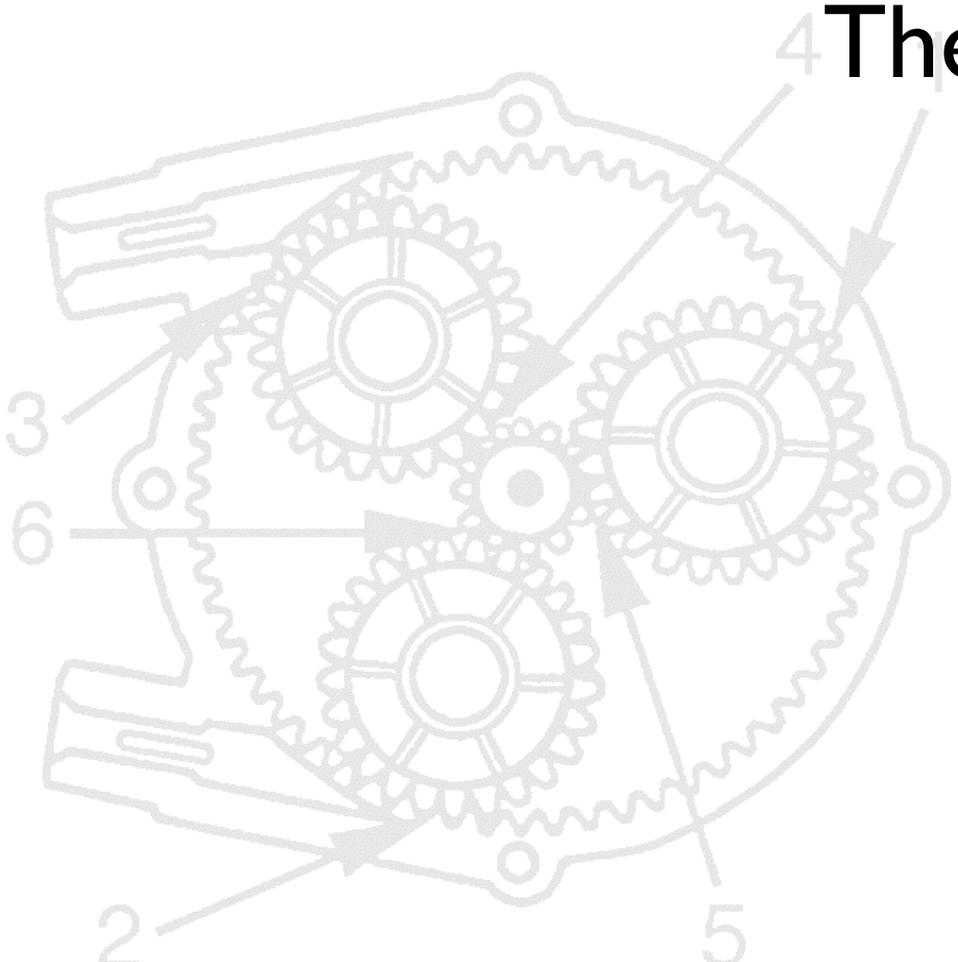
```
done:
    if (l)
    {
        libnet_destroy(l);
    }
    if (p)
    {
        pcap_close(p);
    }
    if (ipg)
    {
        ipgeo_destroy(ipg);
    }
    return (EXIT_SUCCESS);
}
```

TRIG output

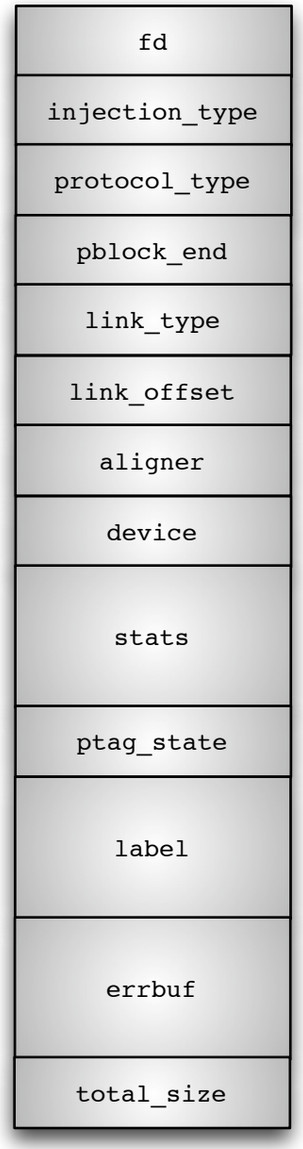
```
[rounder:Projects/libipg/sample] root# ./trig -ien1 -LlCry 4.2.2.2 ../../IP-COUNTRY-REGION-
CITY-LATITUDE-LONGITUDE-ISP-FULL/IP-COUNTRY-REGION-CITY-LATITUDE-LONGITUDE-ISP.CSV
Trig 1.0 [geo-targeting traceroute scanner]
01: 66.123.162.113 US SAN RAMON CALIFORNIA 37.7661 -121.9730
02: 63.203.35.65 US SAN FRANCISCO CALIFORNIA 37.7002 -122.4060
03: 63.203.35.17 US SAN FRANCISCO CALIFORNIA 37.7002 -122.4060
04: 64.161.1.30 CA MONTREAL QUEBEC 45.5000 -73.5830
05: 64.161.1.54 CA MONTREAL QUEBEC 45.5000 -73.5830
06: 144.223.242.81 US KANSAS CITY MISSOURI 39.1749 -94.5804
07: 209.245.146.245 US UNKNOWN UNKNOWN 0.0000 0.0000
08: 209.244.3.137 US BROOMFIELD COLORADO 39.9135 -105.0930
09: 64.159.4.74 US SAN CLEMENTE CALIFORNIA 33.4322 -117.5780
10: 4.24.9.142 EG CAIRO AL QAHIRAH 30.0500 31.2500
11: 4.2.2.2 US PROVIDENCE RHODE ISLAND 41.8231 -71.4204
```

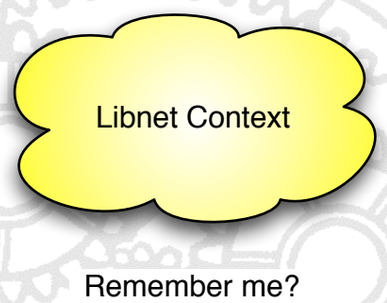
Libnet 1.1.x Internals

The stuff that makes
it go



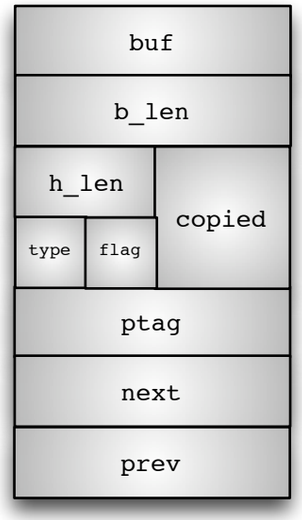
Introduction to the context





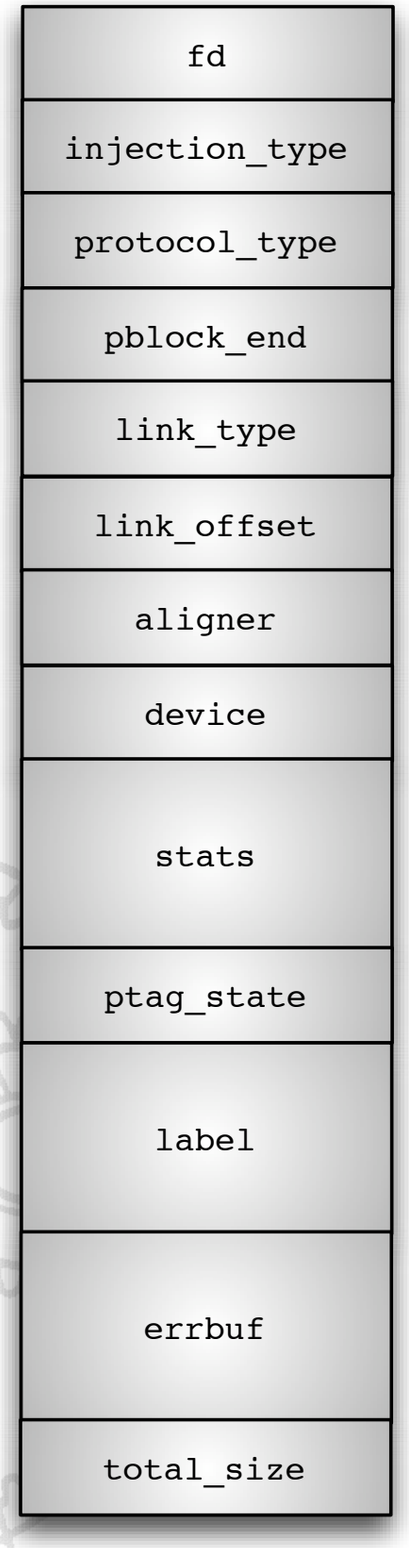
- We've already met.
- Something you don't know: Libnet supports a multiple packet interface
- The "context queue" interface
  - A multiple context interface

Introduction to the pblock

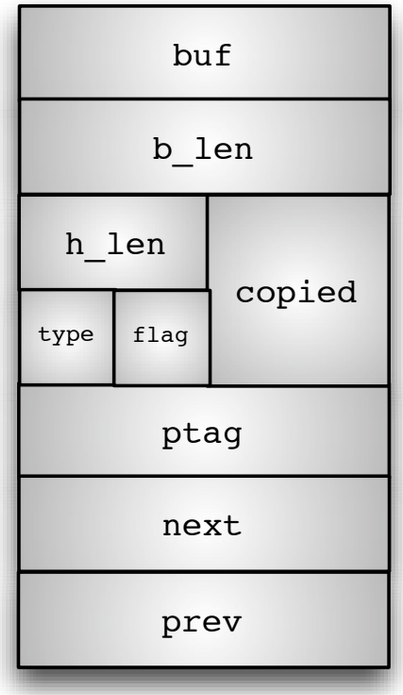


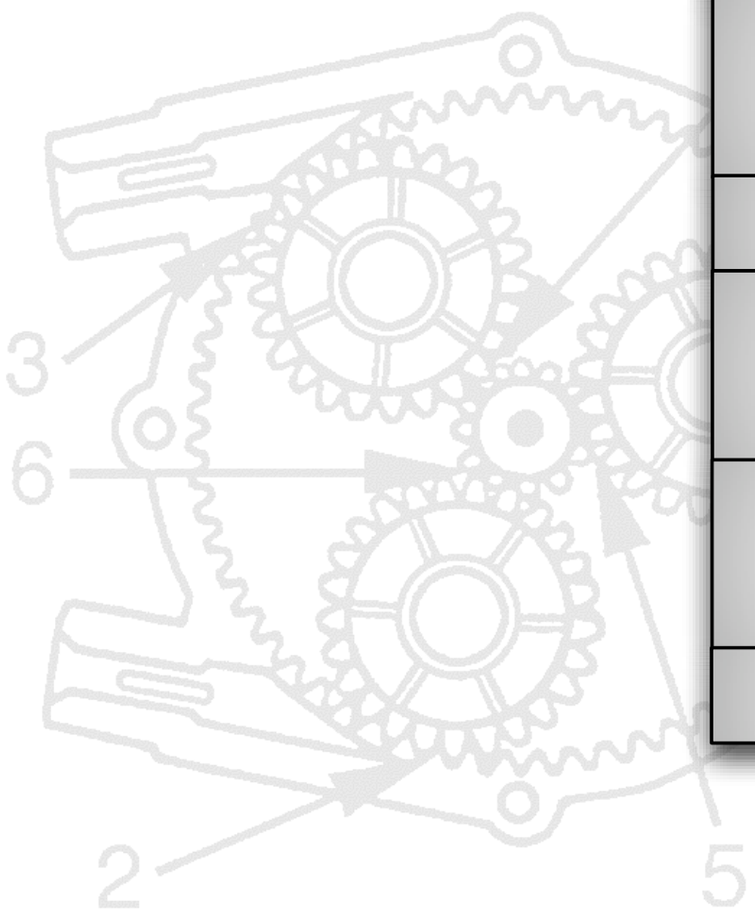
- Libnet's internal packet buffer system
- Header to every packet piece
- Analogous to BSD's mbuf
- All packet memory is handled with one of these babies

The Libnet Context
384 Bytes

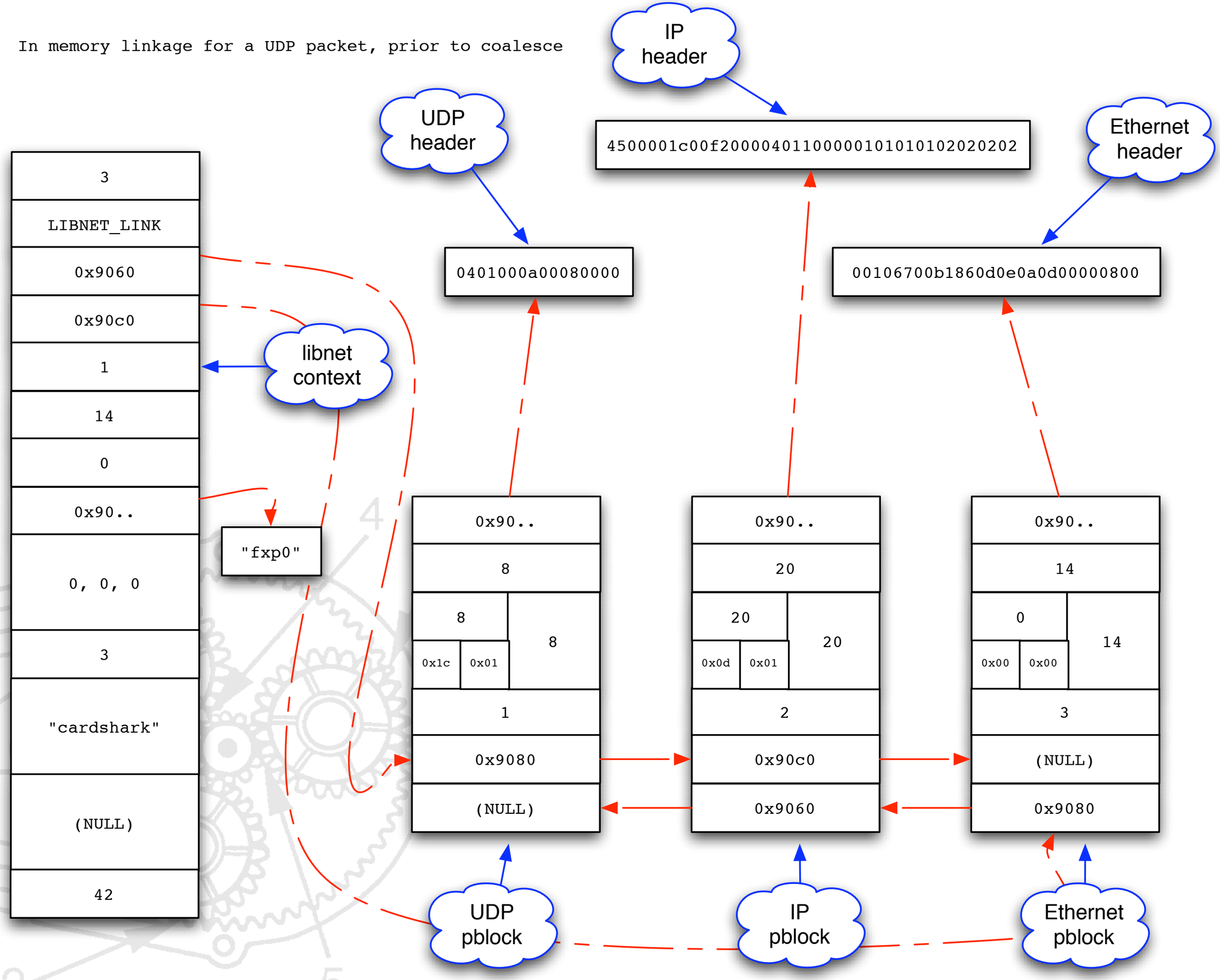


The Libnet Protocol Block
28 Bytes

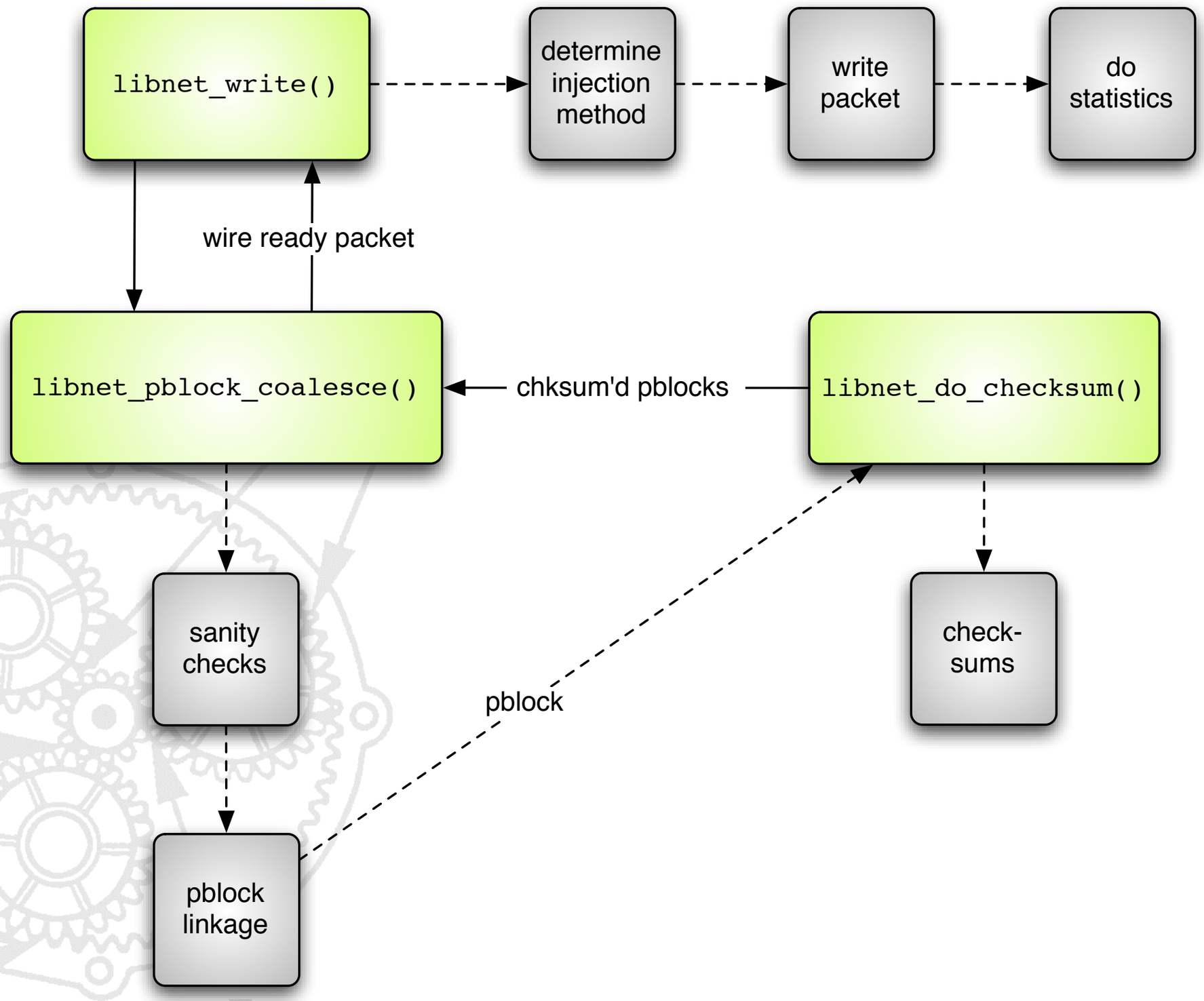




In memory linkage for a UDP packet, prior to coalesce



Internal packet injection logic



Who uses Libnet?

- Ettercap
  - A multipurpose sniffer / interceptor / logger for a switched LAN
  - http://ettercap.sourceforge.net
- Firewalk
  - A gateway portscanning tool
  - http://www.packetfactory.net/firewalk
- ISIC
  - IP stack integrity checker
  - http://www.packetfactory.net/ISIC
- Snort
  - A lightweight network intrusion detection system
  - http://www.snort.org
- Tcpreplay
  - Replays saved tcpdump files at arbitrary speeds
  - tcpreplay.sourceforge.net

Tell your story walking

- We're done.
- Questions? Comments?
- mike@infonexus.com