ISL

Network Programming #9

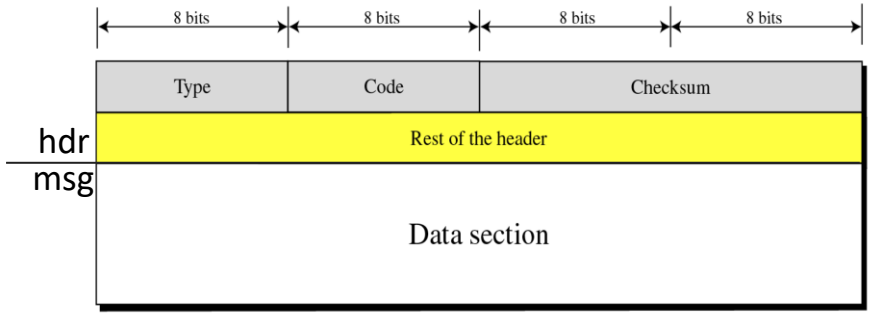
ISL (IoT Standard Lab)

Index

myPing – 프로세스 공통

```
#define PACKETSIZE  64
struct packet
{
    struct icmphdr hdr;
    char msg[PACKETSIZE-sizeof(struct icmphdr)];
};

int pid=-1;
struct protoent *proto=NULL;
```

데이터 크기 = 패킷 크기 – 헤더 크기

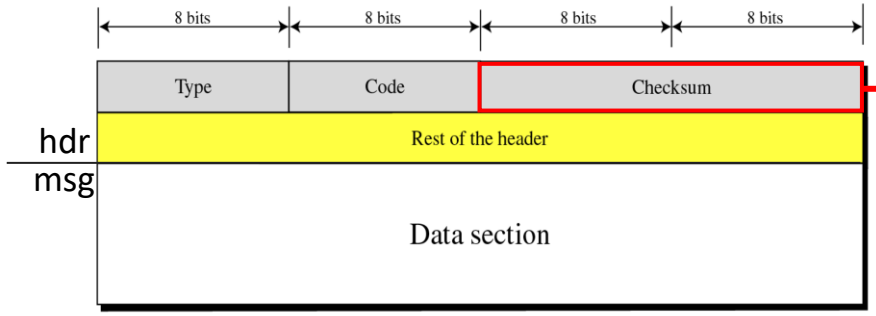


icmp packet format

icmphdr structure

```
struct icmphdr
{
  uint8_t type;   /* message type */
  uint8_t code;   /* type sub-code */
  uint16_t checksum;
  union
  {
    struct
    {
      uint16_t  id;
      uint16_t  sequence;
    } echo;      /* echo datagram */
    uint32_t  gateway;  /* gateway address */
    struct
    {
      uint16_t  __glibc_reserved;
      uint16_t  mtu;
    } frag;      /* path mtu discovery */
  } un;
};
```

myPing – 프로세스 공통



icmp packet format

```
struct icmphdr
{
  uint8_t type;    /* message type */
  uint8_t code;    /* type sub-code */
  uint16_t checksum;
```

```
unsigned short checksum(void *b, int len)
{    unsigned short *buf = b;
    unsigned int sum=0;
    unsigned short result;

    for ( sum = 0; len > 1; len -= 2 )
        sum += *buf++;
    if ( len == 1 )
        sum += *(unsigned char*)buf;
    sum = (sum >> 16) + (sum & 0xFFFF);
    sum += (sum >> 16);
    result = ~sum;
    return result;
}
```

전송된 데이터에 오류가 있는지 체크

myPing – 프로세스 공통

```c
#define PACKETSIZE  64
struct packet
{
    struct icmphdr hdr;
    char msg[PACKETSIZE-sizeof(struct icmphdr)];
};

int pid=-1;
struct protoent *proto=NULL;
```

protoent structure

```c
/* Description of data base entry for a single service.  */
struct protoent
{
  char *p_name;           /* Official protocol name.  */
  char **p_aliases;       /* Alias list.  */
  int p_proto;        /* Protocol number.  */
};
```

```c
pid = getpid();
proto = getprotobyname("ICMP");
hname = gethostbyname(strings[1]);
```

The **getprotobyname**() function returns a *protoent* structure for the entry from the database that matches the protocol name *name*. A connection is opened to the database if necessary.

The **getprotobynumber**() function returns a *protoent* structure for the entry from the database that matches the protocol number *number*. A connection is opened to the database if necessary.

The **setprotoent**() function opens a connection to the database, and sets the next entry to the first entry. If *stayopen* is nonzero, then the connection to the database will not be closed between calls to one of the **getproto***() functions.

The **endprotoent**() function closes the connection to the database.

The *protoent* structure is defined in <*netdb.h*> as follows:

```
struct protoent {
    char  *p_name;       /* official protocol name */
    char **p_aliases;    /* alias list */
    int    p_proto;      /* protocol number */
}
```

```c
sd = socket(PF_INET, SOCK_RAW, proto->p_proto);
```

Raw 소켓에서 사용할 프로토콜 지정

01 Raw socket - ping

myPing – main()

```c
int main(int count, char *strings[])
{   struct hostent *hname;
    struct sockaddr_in addr;

    if ( count != 2 )
    {
        printf("usage: %s <addr>\n", strings[0]);
        exit(0);
    }
    if ( count > 1 )
    {
        pid = getpid();
        proto = getprotobyname("ICMP");
        hname = gethostbyname(strings[1]);
        bzero(&addr, sizeof(addr));
        addr.sin_family = hname->h_addrtype;
        addr.sin_port = 0;
        addr.sin_addr.s_addr = *(long*)hname->h_addr;

        if ( fork() == 0 )
            listener();
        else
            ping(&addr);
        wait(0);

    }
    else
        printf("usage: myping <hostname>\n");
    return 0;
}
```

Network Programming #4 DNS 참조

myPing – listener()

```c
void listener(void)
{   int sd;
    struct sockaddr_in addr;
    unsigned char buf[1024];

    sd = socket(PF_INET, SOCK_RAW, proto->p_proto);
    if ( sd < 0 )
    {
        perror("socket");
        exit(0);
    }
    for (;;)
    {   int bytes, len=sizeof(addr);

        bzero(buf, sizeof(buf));
        bytes = recvfrom(sd, buf, sizeof(buf), 0, (struct sockaddr*)&addr, &len);
        if ( bytes > 0 ) {
            printf("***Got message!***\n");
            display(buf, bytes);
        }
        else
            perror("recvfrom");
    }
    exit(0);
}
```

Raw Socket 타입 + ICMP 프로토콜

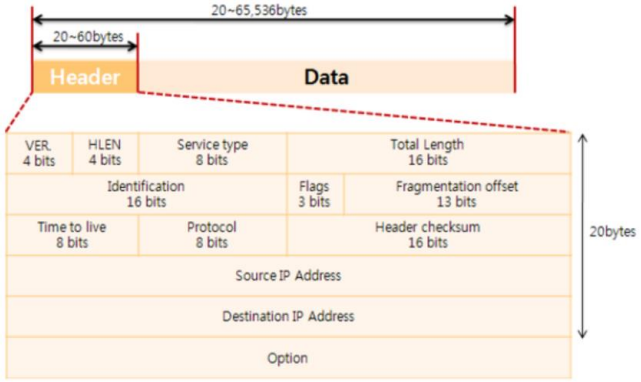
01 Raw socket - ping

myPing – display()

```c
void display(void *buf, int bytes)
{   int i;
    struct iphdr *ip = buf;
    struct icmphdr *icmp = buf+ip->ihl*4;
    struct in_addr addr;

    printf("----------------\n");

    addr.s_addr = ip->saddr;

    printf("IPv%d: hdr-size=%d pkt-size=%d protocol=%d TTL=%d src=%s ",
        ip->version, ip->ihl*4, ntohs(ip->tot_len), ip->protocol,
        ip->ttl, inet_ntoa(addr));

    addr.s_addr = ip->daddr;

    printf("dst=%s\n", inet_ntoa(addr));
    if ( icmp->un.echo.id == pid )
    {
        printf("ICMP: type[%d/%d] checksum[%d] id[%d] seq[%d]\n",
            icmp->type, icmp->code, ntohs(icmp->checksum),
            icmp->un.echo.id, icmp->un.echo.sequence);
    }
}
```

IP 패킷(헤더 + 데이터) 그 자체를 모두 가지고 옴

```c
struct iphdr
  {
#if __BYTE_ORDER == __LITTLE_ENDIAN
    unsigned int ihl:4;
    unsigned int version:4;
#elif __BYTE_ORDER == __BIG_ENDIAN
    unsigned int version:4;
    unsigned int ihl:4;
#else
# error "Please fix <bits/endian.h>"
#endif
    uint8_t tos;
    uint16_t tot_len;
    uint16_t id;
    uint16_t frag_off;
    uint8_t ttl;
    uint8_t protocol;
    uint16_t check;
    uint32_t saddr;
    uint32_t daddr;
    /*The options start here. */
  };
```



IP Packet Header

○ Protocol Identifier (8 bits) ☞ IANA 프로토콜 번호 관리 참조

- 어느 상위계층 프로토콜이 데이터 내에 포함되었는가를 보여줌

. 例) ICMP -> 1, IGMP -> 2, TCP -> 6, EGP -> 8, UDP -> 17, OSPF -> 89 등

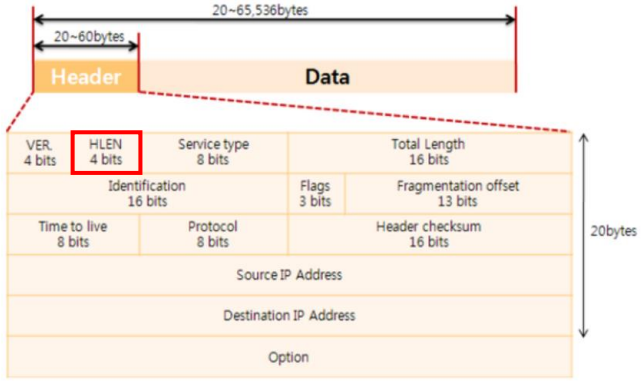
01 Raw socket - ping

myPing – display()

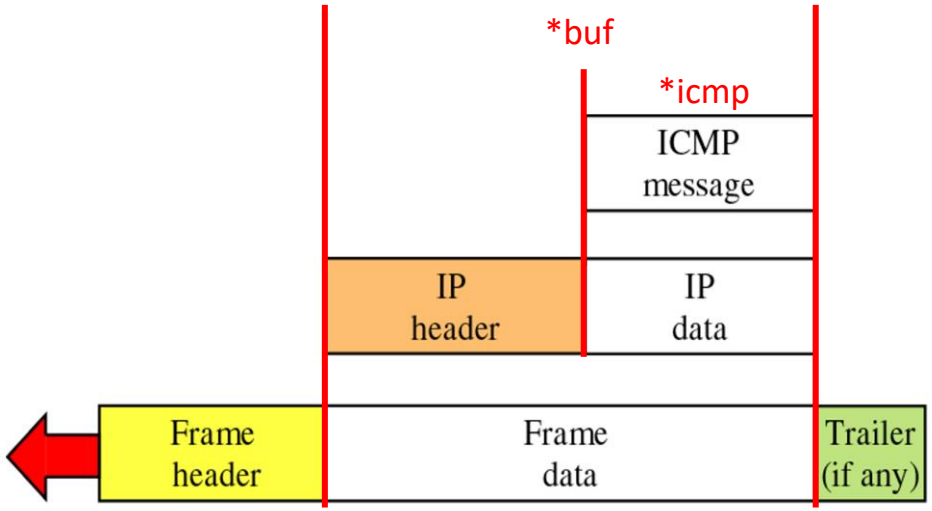
```
void display(void *buf, int bytes)
{   int i;
    struct iphdr *ip = buf;
    struct icmphdr *icmp = buf+ip->ihl*4;
    struct in_addr addr;

    printf("----------------\n");

    addr.s_addr = ip->saddr;

    printf("IPv%d: hdr-size=%d pkt-size=%d protocol=%d TTL=%d src=%s ",
        ip->version, ip->ihl*4, ntohs(ip->tot_len), ip->protocol,
        ip->ttl, inet_ntoa(addr));

    addr.s_addr = ip->daddr;

    printf("dst=%s\n", inet_ntoa(addr));
    if ( icmp->un.echo.id == pid )
    {
        printf("ICMP: type[%d/%d] checksum[%d] id[%d] seq[%d]\n",
            icmp->type, icmp->code, ntohs(icmp->checksum),
            icmp->un.echo.id, icmp->un.echo.sequence);
    }
}
```

IP 헤더 길이만큼 건너 뜀

○ Header Length(HLEN) (4 bits)
- 헤더의 길이
  . 32비트(4 바이트) 워드 단위로 헤더 길이를 표시
- 길이 값 표현
  . 최소 5 (4 × 5 = 20 바이트)부터 15(4 × 15 = 60 바이트,옵션 포함된 경우)까지의 값



IP Packet Header



ICMP Encapsulation

01 Raw socket - ping

myPing – ping()

```
125  void ping(struct sockaddr_in *addr)
126  {   const int val=255;
127      int i, sd, cnt=1;
128      struct packet pckt;
129      struct sockaddr_in r_addr;
130      int bytes;
131
132      sd = socket(PF_INET, SOCK_RAW, proto->p_proto);
133      if ( sd < 0 )
134      {
135          perror("socket");
136          return;
137      }
138      if ( setsockopt(sd, SOL_IP, IP_TTL, &val, sizeof(val)) != 0)
139          perror("Set TTL option");
140      if ( fcntl(sd, F_SETFL, O_NONBLOCK) != 0 )
141          perror("Request nonblocking I/O");
```

Raw Socket 타입 + ICMP 프로토콜

IP 패킷의 TTL 설정

Non-Blocking 소켓 설정

```
144      for (;;)
145      {   int len=sizeof(r_addr);
146
147          printf("Msg #%d\n", cnt);
148          bzero(&pckt, sizeof(pckt));
149          pckt.hdr.type = ICMP_ECHO;
150          pckt.hdr.un.echo.id = pid;
151          for ( i = 0; i < sizeof(pckt.msg)-1; i++ ) {
152              pckt.msg[i] = i+'0';
153              //printf("%c\n", pckt.msg[i]);
154          }
155          pckt.msg[i] = 0;
156          pckt.hdr.un.echo.sequence = cnt++;
157          pckt.hdr.checksum = checksum(&pckt, sizeof(pckt));
158          if ( sendto(sd, &pckt, sizeof(pckt), 0, (struct sockaddr*)addr, sizeof(*addr)) <= 0 )
159              perror("sendto");
160          sleep(1);
161      }
```

```
// if ( bytes = recvfrom(sd, &pckt, sizeof(pckt), 0, (struct sockaddr*)&r_addr, &len) > 0 )
// {
//  printf("\n\n***Got message!***\n");
//  display(&pckt, bytes);
// }
```

```
struct icmphdr
{
  uint8_t type;   /* message type */
  uint8_t code;   /* type sub-code */
  uint16_t checksum;
  union
  {
    struct
    {
      uint16_t  id;
      uint16_t  sequence;
    } echo;      /* echo datagram */
    uint32_t  gateway;  /* gateway address */
    struct
    {
      uint16_t  __glibc_reserved;
      uint16_t  mtu;
    } frag;      /* path mtu discovery */
  } un;
};
```

ICMP Header

8: Echo request
0: Echo reply

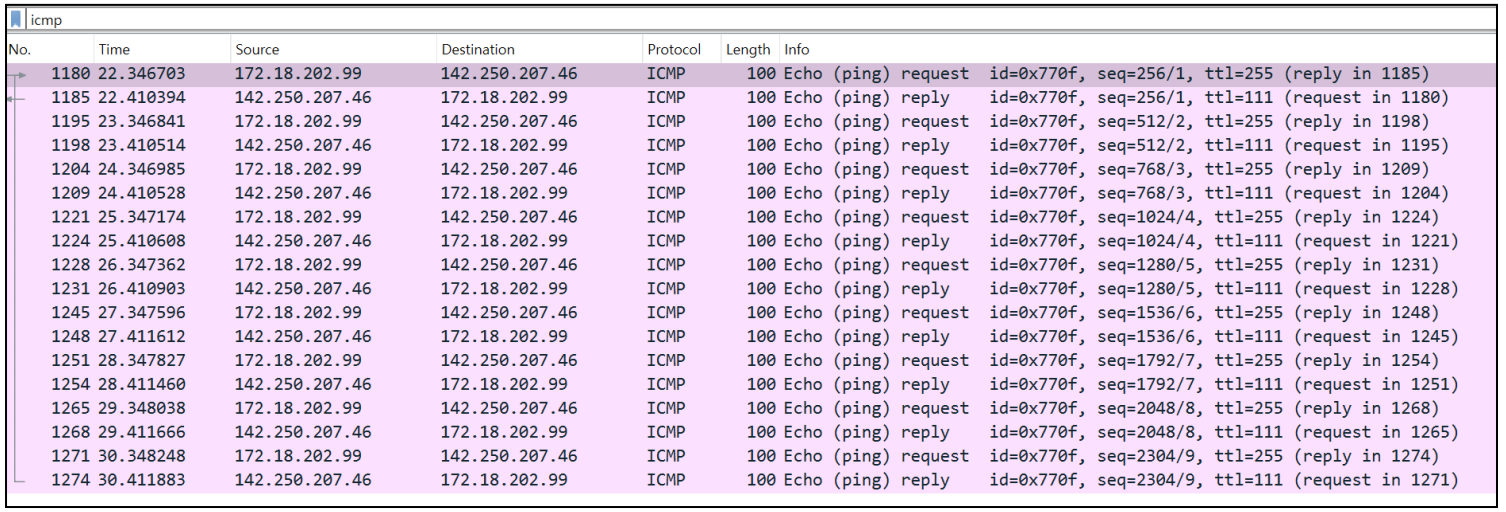
Type: 8 or 0	Code: 0	Checksum
Identifier		Sequence number
Optional data Sent by the request message; repeated by the reply message		

ICMP – Echo request & replay Format

01 Raw socket - ping

myPing – 실행 결과

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket9$ sudo ./myping google.com
Msg #1
***Got message!***
----------------
IPv4: hdr-size=20 pkt-size=84 protocol=1 TTL=111 src=142.250.207.46 dst=172.18.202.99
ICMP: type[0/0] checksum[59905] id[3289] seq[1]

Msg #2
***Got message!***
----------------
IPv4: hdr-size=20 pkt-size=84 protocol=1 TTL=111 src=142.250.207.46 dst=172.18.202.99
ICMP: type[0/0] checksum[59649] id[3289] seq[2]

Msg #3
***Got message!***
----------------
IPv4: hdr-size=20 pkt-size=84 protocol=1 TTL=111 src=142.250.207.46 dst=172.18.202.99
ICMP: type[0/0] checksum[59393] id[3289] seq[3]

Msg #4
***Got message!***
----------------
IPv4: hdr-size=20 pkt-size=84 protocol=1 TTL=111 src=142.250.207.46 dst=172.18.202.99
ICMP: type[0/0] checksum[59137] id[3289] seq[4]

Msg #5
***Got message!***
----------------
IPv4: hdr-size=20 pkt-size=84 protocol=1 TTL=111 src=142.250.207.46 dst=172.18.202.99
ICMP: type[0/0] checksum[58881] id[3289] seq[5]

Msg #6
***Got message!***
----------------
IPv4: hdr-size=20 pkt-size=84 protocol=1 TTL=111 src=142.250.207.46 dst=172.18.202.99
ICMP: type[0/0] checksum[58625] id[3289] seq[6]

Msg #7
***Got message!***
----------------
IPv4: hdr-size=20 pkt-size=84 protocol=1 TTL=111 src=142.250.207.46 dst=172.18.202.99
ICMP: type[0/0] checksum[58369] id[3289] seq[7]

Msg #8
***Got message!***
----------------
IPv4: hdr-size=20 pkt-size=84 protocol=1 TTL=111 src=142.250.207.46 dst=172.18.202.99
ICMP: type[0/0] checksum[58113] id[3289] seq[8]

^C
smalldragon@SD-DESKTOP:~/Workspace/socket/socket9$
```



icmp

No.	Time	Source	Destination	Protocol	Length	Info
1180	22.346703	172.18.202.99	142.250.207.46	ICMP	100	Echo (ping) request id=0x770f, seq=256/1, ttl=255 (reply in 1185)
1185	22.410394	142.250.207.46	172.18.202.99	ICMP	100	Echo (ping) reply id=0x770f, seq=256/1, ttl=111 (request in 1180)
1195	23.346841	172.18.202.99	142.250.207.46	ICMP	100	Echo (ping) request id=0x770f, seq=512/2, ttl=255 (reply in 1198)
1198	23.410514	142.250.207.46	172.18.202.99	ICMP	100	Echo (ping) reply id=0x770f, seq=512/2, ttl=111 (request in 1195)
1204	24.346985	172.18.202.99	142.250.207.46	ICMP	100	Echo (ping) request id=0x770f, seq=768/3, ttl=255 (reply in 1209)
1209	24.410528	142.250.207.46	172.18.202.99	ICMP	100	Echo (ping) reply id=0x770f, seq=768/3, ttl=111 (request in 1204)
1221	25.347174	172.18.202.99	142.250.207.46	ICMP	100	Echo (ping) request id=0x770f, seq=1024/4, ttl=255 (reply in 1224)
1224	25.410608	142.250.207.46	172.18.202.99	ICMP	100	Echo (ping) reply id=0x770f, seq=1024/4, ttl=111 (request in 1221)
1228	26.347362	172.18.202.99	142.250.207.46	ICMP	100	Echo (ping) request id=0x770f, seq=1280/5, ttl=255 (reply in 1231)
1231	26.410903	142.250.207.46	172.18.202.99	ICMP	100	Echo (ping) reply id=0x770f, seq=1280/5, ttl=111 (request in 1228)
1245	27.347596	172.18.202.99	142.250.207.46	ICMP	100	Echo (ping) request id=0x770f, seq=1536/6, ttl=255 (reply in 1248)
1248	27.411612	142.250.207.46	172.18.202.99	ICMP	100	Echo (ping) reply id=0x770f, seq=1536/6, ttl=111 (request in 1245)
1251	28.347827	172.18.202.99	142.250.207.46	ICMP	100	Echo (ping) request id=0x770f, seq=1792/7, ttl=255 (reply in 1254)
1254	28.411460	142.250.207.46	172.18.202.99	ICMP	100	Echo (ping) reply id=0x770f, seq=1792/7, ttl=111 (request in 1251)
1265	29.348038	172.18.202.99	142.250.207.46	ICMP	100	Echo (ping) request id=0x770f, seq=2048/8, ttl=255 (reply in 1268)
1268	29.411666	142.250.207.46	172.18.202.99	ICMP	100	Echo (ping) reply id=0x770f, seq=2048/8, ttl=111 (request in 1265)
1271	30.348248	172.18.202.99	142.250.207.46	ICMP	100	Echo (ping) request id=0x770f, seq=2304/9, ttl=255 (reply in 1274)
1274	30.411883	142.250.207.46	172.18.202.99	ICMP	100	Echo (ping) reply id=0x770f, seq=2304/9, ttl=111 (request in 1271)