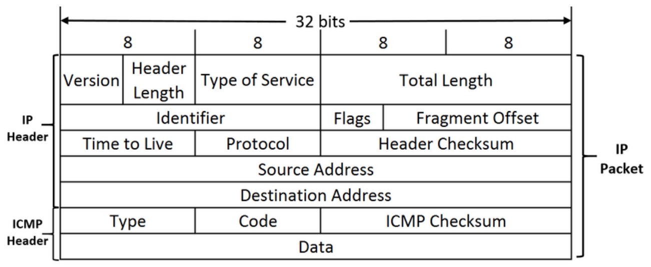
L6-Computer Network Lab



```
#include <stdio.h> // ip_header size
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <sys/socket.h>
#include <sys/types.h>

#define BUFFER_SIZE 1024

int main() {
    int sockfd;
    char buffer[BUFFER_SIZE];
    struct ip *ip_header;
    struct icmphdr *icmp_header;
    socklen_t addr_len;
    // Create a raw socket to capture ICMP packets
    sockfd = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
    if (sockfd < 0) {
        perror("socket");exit(1);
    }
    while (1) {
        addr_len = sizeof(struct sockaddr);
        int bytes_received = recvfrom(sockfd, buffer, BUFFER_SIZE, 0, NULL, &addr_len);
        if (bytes_received < 0) {
            perror("recvfrom");close(sockfd);exit(1);
        }
        // Extract IP header
        ip_header = (struct ip *)buffer;
        // Check if the packet is ICMP
        if (ip_header->ip_p == IPPROTO_ICMP) {
            // Calculate the length of the IP header
            int ip_header_length = ip_header->ip_hl * 4;
            // Print the length of the IP header
            printf("Length of IP header: %d bytes\n", ip_header_length);
            // Extract ICMP header
            icmp_header = (struct icmphdr *)(buffer + ip_header_length);
            // You can print other ICMP header fields if needed
            // printf("Type: %d, Code: %d\n", icmp_header->type, icmp_header->code);
        }
    }
    close(sockfd);
    return 0;
}
```

```
//All parameters

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <sys/types.h>

#define MAX_PACKET_SIZE 65536
```

```
void process_packet(unsigned char *buffer, int size) {
    struct ip *ip_header = (struct ip *)buffer;
    struct icmphdr *icmp_header = (struct icmphdr *)(buffer + (ip_header->ip_hl << 2));

    printf("IP Header:\n");
    printf("  Version: %d\n", ip_header->ip_v);
    printf("  Header Length: %d words (%d bytes)\n", ip_header->ip_hl, (ip_header->ip_hl << 2));
    printf("  TOS: %d\n", ip_header->ip_tos);
    printf("  Total Length: %d bytes\n", ntohs(ip_header->ip_len));
    printf("  Identification: %d\n", ntohs(ip_header->ip_id));
    printf("  Flags: 0x%02x\n", (ntohs(ip_header->ip_off) & IP_RF));
    printf("  Fragment Offset: %d\n", (ntohs(ip_header->ip_off) & IP_OFFMASK));
    printf("  TTL: %d\n", ip_header->ip_ttl);
    printf("  Protocol: %d (ICMP)\n", ip_header->ip_p);
    printf("  Checksum: 0x%04x\n", ntohs(ip_header->ip_sum));
    printf("  Source IP: %s\n", inet_ntoa(ip_header->ip_src));
    printf("  Destination IP: %s\n", inet_ntoa(ip_header->ip_dst));

    printf("ICMP Header:\n");
    printf("  Type: %d\n", icmp_header->type);
    printf("  Code: %d\n", icmp_header->code);
    printf("  Checksum: 0x%04x\n", ntohs(icmp_header->checksum));
    printf("  Identifier: %d\n", ntohs(icmp_header->un.echo.id));
    printf("  Sequence Number: %d\n", ntohs(icmp_header->un.echo.sequence));

    // You can also print the payload of the ICMP packet if needed.

    printf("\n");
}

int main() {
    int raw_socket;
    unsigned char buffer[MAX_PACKET_SIZE];

    raw_socket = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
    if (raw_socket == -1) {
        perror("socket");
        exit(EXIT_FAILURE);
    }

    while (1) {
        int size = recv(raw_socket, buffer, sizeof(buffer), 0);
        if (size == -1) {
            perror("recv");
            close(raw_socket);
            exit(EXIT_FAILURE);
        }
        printf("Packet Size: %d bytes\n", size);
        process_packet(buffer, size);
    }

    close(raw_socket);

    return 0;
}
```

```
// Packet Data
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <sys/types.h>

#define MAX_PACKET_SIZE 65536

void process_packet(unsigned char *buffer, int size) {
    struct ip *ip_header = (struct ip *)buffer;
    struct icmphdr *icmp_header = (struct icmphdr *)(buffer + (ip_header->ip_hl << 2));
    unsigned char *icmp_data = (unsigned char *)(buffer + (ip_header->ip_hl << 2) + sizeof(struct icmphdr));

    printf("IP Header:\n");
    // ... (Same as before)

    printf("ICMP Header:\n");
    // ... (Same as before)

    printf("ICMP Data:\n");
    printf("  ");
    for (int i = 0; i < size - (ip_header->ip_hl << 2) - sizeof(struct icmphdr); i++) {
        printf("%02x ", icmp_data[i]);
    }
    printf("\n");

    printf("\n");
}

int main() {
    int raw_socket;
    unsigned char buffer[MAX_PACKET_SIZE];

    raw_socket = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
    if (raw_socket == -1) {
        perror("socket");
        exit(EXIT_FAILURE);
    }

    while (1) {
        int size = recv(raw_socket, buffer, sizeof(buffer), 0);
        if (size == -1) {
            perror("recv");
            close(raw_socket);
            exit(EXIT_FAILURE);
        }
        process_packet(buffer, size);
    }

    close(raw_socket);

    return 0;
}
```

```
// Verification
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <sys/types.h>

#define MAX_PACKET_SIZE 65536

unsigned short calculate_checksum(unsigned short *data, int length) {
    unsigned long sum = 0;
    unsigned short *temp = data;

    while (length > 1) {
        sum += *temp++;
        length -= 2;
    }
    // Add any remaining byte if the length is odd
    if (length > 0) {
        sum += *(unsigned char *)temp;
    }
    // Fold the 32-bit sum to 16 bits
    while (sum >> 16) {
        sum = (sum & 0xFFFF) + (sum >> 16);
    }
    return (unsigned short)(~sum);
}

void process_packet(unsigned char *buffer, int size) {
    struct ip *ip_header = (struct ip *)buffer;
    struct icmphdr *icmp_header = (struct icmphdr *)(buffer + (ip_header->ip_hl << 2));
    unsigned char *icmp_data = (unsigned char *)(buffer + (ip_header->ip_hl << 2) + sizeof(struct icmphdr));

    printf("IP Header:\n");
    // ... (Same as before)

    printf("ICMP Header:\n");
    // ... (Same as before)

    // Calculate the received checksum
    unsigned short received_checksum = icmp_header->checksum;

    // Calculate the expected checksum
```

```c
    unsigned short expected_checksum =calculate_checksum((unsigned short*)icmp_header,ntohs(ip_header->ip_len)-(ip_header->ip_hl<<2));

    printf("ICMP Data:\n");
    printf("  ");
    for (int i = 0; i < size - (ip_header->ip_hl << 2) - sizeof(struct icmphdr); i++) {
        printf("%02x ", icmp_data[i]);
    }
    printf("\n");

    // Verify checksum
    if (received_checksum == expected_checksum) {
        printf("Checksum Verification: Passed\n");
    } else {
        printf("Checksum Verification: Failed\n");
    }
    printf("\n");
}

int main() {
    int raw_socket;
    unsigned char buffer[MAX_PACKET_SIZE];

    raw_socket = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
    if (raw_socket == -1) {
        perror("socket");
        exit(EXIT_FAILURE);
    }
    while (1) {
        int size = recv(raw_socket, buffer, sizeof(buffer), 0);
        if (size == -1) {
            perror("recv");
            close(raw_socket);
            exit(EXIT_FAILURE);
        }
        process_packet(buffer, size);
    }
    close(raw_socket);
    return 0;
}
```

```c
// sending data using icmp
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <sys/types.h>

#define PACKET_SIZE 64
#define ICMP_TYPE 8  // ICMP Echo Request
#define ICMP_CODE 0
#define ICMP_HEADER_SIZE 8

unsigned short calculate_checksum(unsigned short *data, int length) {
    unsigned long sum = 0;
    unsigned short *temp = data;

    while (length > 1) {
        sum += *temp++;
        length -= 2;
    }
    // Add any remaining byte if the length is odd
    if (length > 0) {
        sum += *(unsigned char *)temp;
    }
    // Fold the 32-bit sum to 16 bits
    while (sum >> 16) {
        sum = (sum & 0xFFFF) + (sum >> 16);
    }
    return (unsigned short)(~sum);
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <destination_ip>\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    const char *destination_ip = argv[1];

    int raw_socket;
    unsigned char packet[PACKET_SIZE];

    // Create a raw socket
    raw_socket = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
    if (raw_socket == -1) {
        perror("socket");
        exit(EXIT_FAILURE);
    }

    // Set up the destination address
    struct sockaddr_in destination;
    memset(&destination, 0, sizeof(destination));
    destination.sin_family = AF_INET;
    if (inet_pton(AF_INET, destination_ip, &destination.sin_addr) != 1) {
        perror("inet_pton");
        close(raw_socket);
        exit(EXIT_FAILURE);
    }

    // Construct the ICMP Echo Request packet
    struct icmphdr *icmp_header = (struct icmphdr *)packet;
    memset(packet, 0, PACKET_SIZE);

    icmp_header->type = ICMP_TYPE;
    icmp_header->code = ICMP_CODE;
    icmp_header->checksum = 0;
    icmp_header->un.echo.id = getpid(); // Use the process ID as the identifier
    icmp_header->un.echo.sequence = 0; // Sequence number (you can increment this for each packet)

    // Fill the packet with a pattern for data
    char *data = (char *)(packet + ICMP_HEADER_SIZE);
    for (int i = 0; i < PACKET_SIZE - ICMP_HEADER_SIZE; i++) {
        data[i] = i % 10 + '0';
    }

    // Calculate the ICMP checksum
    icmp_header->checksum = calculate_checksum((unsigned short *)icmp_header, PACKET_SIZE);

    // Send the packet
    if (sendto(raw_socket, packet, PACKET_SIZE, 0, (struct sockaddr *)&destination, sizeof(destination)) == -1) {
        perror("sendto");
        close(raw_socket);
        exit(EXIT_FAILURE);
    }

    printf("ICMP Echo Request sent to %s\n", destination_ip);
    close(raw_socket);
    return 0;
}
```

```
// client
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
int main(int argc, char **argv)
{
    if (argc != 2)
        printf("usage: tcpcli <ipaddress>\n");

    int sockfd, n;
    struct sockaddr_in servaddr;
    char sendline[512], recvline[512];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(6000);
    servaddr.sin_addr.s_addr = inet_addr(argv[1]);
    connect(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr));

    fgets(sendline, sizeof(sendline), stdin);
    int len = strlen(sendline);
    write(sockfd, sendline, len);

    n = read(sockfd, recvline, sizeof(recvline));

    if (n < 0)
```

```
        printf("error reading\n");

    recvline[n] = 0;
    fputs(recvline, stdout);
    close(sockfd);

    exit(0);
}

// tcp server

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define SERV_PORT 6000

int main(int argc, char **argv)
{
    int listenfd, connfd, clilen;
    pid_t childpid;
    struct sockaddr_in servaddr, cliaddr;
    char msg1[512];

    listenfd = socket(AF_INET, SOCK_STREAM, 0);
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(SERV_PORT);
    bind(listenfd, (struct sockaddr *)&servaddr, sizeof(servaddr));
    listen(listenfd, 5);

    for (;;)
    {
        clilen = sizeof(cliaddr);
        connfd = accept(listenfd, (struct sockaddr *)&cliaddr, &clilen);

        if ((childpid = fork()) == 0)
        {
            close(listenfd);
            int n1 = read(connfd, msg1, sizeof(msg1));
            write(connfd, msg1, n1);
            close(connfd);
            exit(0);
        }
        close(connfd);
    }

    return 0;
}
```