Tema 1 PCOM

Generated by Doxygen 1.8.13

Contents

Chapter 1

File Index

1.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 2

File Documentation

2.1 arp.c File Reference

This is the implementation for ARP protocol. It contains a setter for the ARP header, a search of the ARP Table for a given IP address, a function that adds a new entry to the table and request / reply functions with a handler.

```
#include "router_funs.h"
#include "arp.h"
#include "skel.h"
```

Include dependency graph for arp.c:

2.2 arp.h File Reference

Functions prototypes for ARP.

```
#include "skel.h"
#include "queue.h"
```

Include dependency graph for arp.h: This graph shows which files directly or indirectly include this file:

Macros

- #define **ARP_MAX_SIZE** 20

Functions

- void arp_hdr_setter (struct arp_header ∗arp_hdr, uint16_t arp_op, uint8_t dest_mac[ETH_ALEN], uint32_t dest_ip, uint8_t source_mac[ETH_ALEN], uint32_t source_ip)

  Sets the ARP header fields to the given values.
- struct arp_entry ∗ search_arp (arp_entry ∗∗arp_table, size_t arp_len, uint32_t ip)

  Searches the ARP table for an IP. If found, returns the entry, otherwise returns NULL.
- void arp_requester (rtable_entry ∗route)

  Starting from a given packet, creates an ARP Request and sends it.
- void arp_replier (packet base)

  Starting from a given ARP REQUEST, creates an ARP REPLY and sends it.

- void add_arp_entry (arp_entry **arp_table, size_t *arp_max_len, size_t *arp_len, uint32_t ip, uint8_t mac[ETH_ALEN])

  Creates a new entry and adds it to the table. If the maximum length of the table is reached, then it also reallocate memory for the table.
- void search_queue (queue no_mac_dest_queue, rtable_entry *rtable, size_t rtable_len, arp_entry **arp_table, size_t arp_len)

  When a ARP Reply is recieved, search the ARP Queue in order to send the waiting packet.
- void arp_handler (packet pkt, queue no_mac_dest_queue, arp_entry **arp_table, size_t *arp_max_len, size_t *arp_len, rtable_entry *rtable, size_t rtable_len)

  Calls reply / request functions depending of the type of the ARP recieved.

2.2.1 Detailed Description

Functions prototypes for ARP.

Author

Radu-Andrei Dumitru

2.2.2 Function Documentation

2.2.2.1 add_arp_entry()

```
void add_arp_entry (
            arp_entry ** arp_table,
            size_t * arp_max_len,
            size_t * arp_len,
            uint32_t ip,
            uint8_t mac[ETH_ALEN] )
```

Creates a new entry and adds it to the table. If the maximum length of the table is reached, then it also reallocate memory for the table.

Parameters

arp_table	The ARP table of the current router.
arp_max_len	The maximum length of the ARP table (needed for realloc).
arp_len	The current length of the ARP table.
ip	The IP that should be added in the entry.
mac	The MAC that should be added in the entry.

Definition at line 48 of file arp.c.

```
{
    /* If the maximum length has been reached, then increase the size of the
    table.
    */
    if (*arp_len == *arp_max_len) {
        arp_entry **aux = realloc(arp_table, (++(*arp_max_len)) *
                            sizeof(arp_entry *));

        DIE(!aux, "Realloc failed.\n");

        arp_table = aux;
    }

    // Create a new entry and add it to the table.
    arp_entry *new = malloc(sizeof(arp_entry));
    DIE(!new, "New entry failed.\n");

    new->ip = ip;
    memcpy(new->mac, mac, ETH_ALEN);
    arp_table[(*arp_len)++] = new;
}
```

2.2.2.2 arp_handler()

```
void arp_handler (
              packet pkt,
              queue no_mac_dest_queue,
              arp_entry ** arp_table,
              size_t * arp_max_len,
              size_t * arp_len,
              rtable_entry * rtable,
              size_t rtable_len )
```

Calls reply / request functions depending of the type of the ARP recieved.

Parameters

pkt	The recieved ARP packet.
no_mac_dest_queue	The queue that holds the packet without destination MAC.
arp_table	The ARP table of the current router.
arp_max_len	The maximum length of the ARP table (needed for realloc).
arp_len	The current length of the ARP table.
rtable	The routing table.
rtable_len	The length of the routing table.

Definition at line 170 of file arp.c.

```
{
    struct arp_header *arp_hdr = (struct arp_header *) (pkt.payload +
                                        sizeof(struct ether_header));
    // Identifying the ARP Operation Type.
    if (ntohs(arp_hdr->op) == ARPOP_REQUEST) {
        arp_replier(pkt);

    } else if(htons(arp_hdr->op) == ARPOP_REPLY) {
        uint32_t interf_ip = inet_addr(get_interface_ip(pkt.interface));
        if (arp_hdr->tpa == interf_ip) {
            add_arp_entry(arp_table, arp_max_len, arp_len,
                        arp_hdr->spa, arp_hdr->sha);
            search_queue(no_mac_dest_queue, rtable, rtable_len, arp_table,
                            *arp_len);
        }
    }
}
```

2.2.2.3 arp_hdr_setter()

```
void arp_hdr_setter (
             struct arp_header * arp_hdr,
             uint16_t arp_op,
             uint8_t dest_mac[ETH_ALEN],
             uint32_t dest_ip,
             uint8_t source_mac[ETH_ALEN],
             uint32_t source_ip )
```

Sets the ARP header fields to the given values.

Parameters

arp_hdr	The base ARP header.
arp_op	The ARP Operation (Request / Reply).
dest_mac	The MAC destination where the packet should arrive. (00:00:00:00:00:00 for a Request)
dest_ip	The IP of the destination where the packet should arrive.
source_mac	The MAC of the packet's starting point.
source_ip	The IP of the packet's starting point.

Definition at line 14 of file arp.c.

```
{
    // The default values for ARP.
    arp_hdr->htype = htons(ETHER_HTYPE);
    arp_hdr->ptype = htons(ETHERTYPE_IP);
    arp_hdr->hlen = ETH_ALEN;
    arp_hdr->plen = IPV4_PROTO_LEN;

    // The operation can be ARPOP_REQUEST / ARPOP_REPLY.
    arp_hdr->op = htons(arp_op);

    // Setting the source MAC and IP.
    memcpy(arp_hdr->sha, source_mac, ETH_ALEN);
    arp_hdr->spa = source_ip;

    // Setting the target MAC and IP.
    memcpy(arp_hdr->tha, dest_mac, ETH_ALEN);
    arp_hdr->tpa = dest_ip;
}
```

2.2.2.4 arp_replier()

```
void arp_replier (
             packet base )
```

Starting from a given ARP REQUEST, creates an ARP REPLY and sends it.

Parameters

base	The ARP REQUEST.

Definition at line 136 of file arp.c.

```
137 {
138     struct arp_header *base_arp = (struct arp_header *) (base.payload +
139                                 sizeof(struct ether_header));
140 
141     // Create a new packet for the reply.
142     packet reply;
143     struct ether_header *reply_eth = (struct ether_header *) reply.payload;
144     struct arp_header *reply_arp = (struct arp_header *) (reply.payload +
145                                     sizeof(struct ether_header));
146 
147 
148     reply.len = sizeof(struct ether_header) + sizeof(struct arp_header);
149     memset(reply.payload, 0, sizeof(reply.payload));
150     /* The reply should be sent on the same interface as the one where the
151     request came on.
152     */
153     reply.interface = base.interface;
154 
155     /* The target becomes the source of the request and the found host becomes
156     the source.
157     */
158     uint8_t source_mac[6];
159     get_interface_mac(base.interface, source_mac);
160 
161     eth_hdr_setter(reply_eth, base_arp->sha, source_mac,
162      ETHERTYPE_ARP);
163 
164     arp_hdr_setter(reply_arp, ARPOP_REPLY, base_arp->sha,
165                     base_arp->spa, source_mac, base_arp->tpa);
166 
167     send_packet(&reply);
168 }
```

2.2.2.5 arp_requester()

```
void arp_requester (
              rtable_entry * route )
```

Starting from a given packet, creates an ARP Request and sends it.

Parameters

route	The route where the packet should be sent on.

Definition at line 101 of file arp.c.

```
102 {
103     packet request;
104 
105     struct ether_header *eth_hdr = (struct ether_header *) request.payload;
106     struct arp_header *arp_hdr = (struct arp_header *) (request.payload
107                                 + sizeof(struct ether_header));
108 
109     request.len = sizeof(struct ether_header) + sizeof(struct arp_header);
110     memset(request.payload  , 0, sizeof(request.payload));
111     request.interface = route->interface;
112 
113     // The request should be sent to Broadcast. (FF:FF:FF:FF:FF:FF).
114     uint8_t broadcast_mac[ETH_ALEN];
115     memset(broadcast_mac, 0xFF, ETH_ALEN);
116 
117     /* The request source is the MAC from the interface that it will be sent
118     on.
119     */
120     uint8_t source_mac[6];
121     get_interface_mac(route->interface, source_mac);
122     eth_hdr_setter(eth_hdr, broadcast_mac, source_mac, ETHERTYPE_ARP);
123 
124     // The Target MAC Address in ARP Header is 00:00:00:00:00:00.
125     uint8_t no_mac[ETH_ALEN];
```

```
    memset(no_mac, 0x00, ETH_ALEN);
    // Extracting the source IP.
    uint32_t source_ip = inet_addr(get_interface_ip(route->interface));

    arp_hdr_setter(arp_hdr, ARPOP_REQUEST, no_mac, route->next_hop,
                source_mac, source_ip);

    send_packet(&request);
}
```

2.2.2.6 search_arp()

```
struct arp_entry* search_arp (
            arp_entry ** arp_table,
            size_t arp_len,
            uint32_t ip )
```

Searches the ARP table for an IP. If found, returns the entry, otherwise returns NULL.

Parameters

arp_table	The ARP table of the router.
arp_len	The length of the ARP table.
ip	The IP that it will be searched in the ARP table.

Returns

struct arp_entry∗ The found entry in the ARP table.

Definition at line 36 of file arp.c.

```
{
    size_t i;
    for (i = 0; i < arp_len; i++) {
        if (arp_table[i]->ip == ip)
            return arp_table[i];
    }

    return NULL;
}
```

2.2.2.7 search_queue()

```
void search_queue (
            queue no_mac_dest_queue,
            rtable_entry * rtable,
            size_t rtable_len,
            arp_entry ** arp_table,
            size_t arp_len )
```

When a ARP Reply is recieved, search the ARP Queue in order to send the waiting packet.

Parameters

no_mac_dest_queue	The queue that holds the packets without destination MAC.
rtable	The routing table.
rtable_len	The length of the routing table.
arp_table	The ARP table of the current router.
arp_len	The length of the ARP table.

Definition at line 72 of file arp.c.

```
{
    queue aux = queue_create();

    while (!queue_empty(no_mac_dest_queue)) {
        packet *crt = (packet *) (queue_deq(no_mac_dest_queue));
        struct iphdr *ip_hdr = (struct iphdr *)
                                (crt->payload + sizeof(struct ether_header));
        struct ether_header *eth_hdr = (struct ether_header *) crt->payload;

        rtable_entry *route = get_best_route(rtable, rtable_len,
                                             ip_hdr->daddr);
        arp_entry *existing = search_arp(arp_table, arp_len, route->next_hop);

        if (existing) {
            memcpy(eth_hdr->ether_dhost, existing->mac, ETH_ALEN);
            get_interface_mac(route->interface, eth_hdr->ether_shost);
            send_packet(crt);
        } else {
            queue_enq(no_mac_dest_queue, crt);
        }
    }

    while (!queue_empty(aux)) {
        queue_enq(no_mac_dest_queue, queue_deq(aux));
    }
}
```

2.3 ip_icmp.c File Reference

This is the implementation for IP and ICMP functions. There are setters for the headers, a function that sends ICMP replies, a handler, a integrity validator for IP header and the incremental checksum function.

```
#include "ip_icmp.h"
#include "skel.h"
```

Include dependency graph for ip_icmp.c:

Functions

- void ip_hdr_setter (struct iphdr ∗reply_iphdr, struct iphdr ∗req_iphdr, uint32_t interf_ip)

  Sets all the fields of an iphdr struct (The frag_off is taken from an ICMP request).
- void icmp_hdr_setter (struct icmphdr ∗reply_icmp, struct icmphdr ∗req_icmp, uint8_t icmp_type)

  Sets the code to 0 and the type to the given icmp_type. If the type is destination unreachable / time exceeded, then the id and sequence will be set to 0. Else if the type is an Echo Reply, it sets id and sequence to the same that were in the Echo Request.
- void send_icmp (packet req_pkt, uint8_t icmp_type)

  Starting from a Echo Request, creates a new packet, completes it with the setter functions and sends it.
- int icmp_handler (packet pkt, struct iphdr ∗ip_hdr)

  Deciding if it should send a reply and calls the functions needed for sending one.
- int is_garbage (packet pkt, struct iphdr ∗ip_hdr)

  Decides if a packet should be thrown away (invalid checksum / ttl $<$ 2). Also if the problem is with ttl, sends an ICMP Time Exceeded reply.
- void incremental_checksum (struct iphdr ∗ip_hdr)

  Updates the IPv4 checksum in an efficient way, avoiding using recalculation.

2.3.1 Detailed Description

This is the implementation for IP and ICMP functions. There are setters for the headers, a function that sends ICMP replies, a handler, a integrity validator for IP header and the incremental checksum function.

Author

Radu-Andrei Dumitru

2.3.2 Function Documentation

2.3.2.1 icmp_handler()

```
int icmp_handler (
              packet pkt,
              struct iphdr * ip_hdr )
```

Deciding if it should send a reply and calls the functions needed for sending one.

Parameters

pkt	The recieved packet.
ip_hdr	The header of the recieved packet.

Returns

int 1 (ICMP_REPLY_SENT) if a reply has been sent, 0 (ICMP_NO_REPLY_SENT), otherwise.

Definition at line 96 of file ip_icmp.c.

```
{
    if (ip_hdr->protocol == IPPROTO_ICMP) {
        struct icmphdr *icmp_hdr = (struct icmphdr *) (pkt.payload +
        sizeof(struct ether_header) + sizeof(struct iphdr));

        // If is a REQUEST and is for the current router, send a reply.
        if (icmp_hdr->type == ICMP_ECHO) {
            uint32_t interf_ip = inet_addr(get_interface_ip(pkt.interface));
            if (ip_hdr->daddr == interf_ip) {
                send_icmp(pkt, ICMP_ECHOREPLY);
                return ICMP_REPLY_SENT;
            }
        }
    }

    return ICMP_NO_REPLY_SENT;
}
```

2.3.2.2 icmp_hdr_setter()

```
void icmp_hdr_setter (
             struct icmphdr * reply_icmp,
             struct icmphdr * req_icmp,
             uint8_t icmp_type )
```

Sets the code to 0 and the type to the given icmp_type. If the type is destination unreachable / time exceeded, then the id and sequence will be set to 0. Else if the type is an Echo Reply, it sets id and sequence to the same that were in the Echo Request.

Parameters

reply_icmp	The base IP header.
req_icmp	The ICMP header of the packet recieved.
icmp_type	The type of the ICMP. (Reply / Time Exceeded / Destination unreachable)

Definition at line 32 of file ip_icmp.c.

```
{
    reply_icmp->type = icmp_type;
    reply_icmp->code = ICMP_CODE;

    /* Check if it is destination unreachable / time exceeded / echo reply. For
    reply, keep the same id and sequence.
    */
    if (reply_icmp->type == ICMP_UNREACH ||
        reply_icmp->type == ICMP_TIME_EXCEEDED) {
        reply_icmp->un.echo.id = 0;
        reply_icmp->un.echo.sequence = 0;

    } else if (reply_icmp->type == ICMP_ECHOREPLY) {
        reply_icmp->un.echo.id = req_icmp->un.echo.id;
        reply_icmp->un.echo.sequence = req_icmp->un.echo.sequence;
    }

    // Compute the checksum.
    reply_icmp->checksum = 0;
    reply_icmp->checksum = ip_checksum((void *) reply_icmp,
                                       sizeof(struct icmp));
}
```

2.3.2.3 incremental_checksum()

```
void incremental_checksum (
             struct iphdr * ip_hdr )
```

Updates the IPv4 checksum in an efficient way, avoiding using recalculation.

Parameters

ip_hdr	The IP Header of the packet.

Definition at line 132 of file ip_icmp.c.

```
{
    ip_hdr->ttl--;

    uint32_t aux_sum;
    uint16_t mask = ((uint16_t) 1) << 8; // 0000 0001 0000 0000

    // Incrementing the first byte of the check sum.
    aux_sum = ntohs(ip_hdr->check) + mask;

    // Adding the carry to the end. (To perform one's complement sum).
    ip_hdr->check = htons((aux_sum + (aux_sum >> 16)));
}
```

2.3.2.4 ip_hdr_setter()

```
void ip_hdr_setter (
            struct iphdr * reply_iphdr,
            struct iphdr * req_iphdr,
            uint32_t interf_ip )
```

Sets all the fields of an iphdr struct (The frag_off is taken from an ICMP request).

Parameters

reply_iphdr	The base IP header.
req_iphdr	The IP header of the packet recieved.
interf_ip	The IP of the current interface.

Definition at line 12 of file ip_icmp.c.

```
{
    reply_iphdr->ihl = 5;
    reply_iphdr->version = 4;

    reply_iphdr->tos = BEST_EFFORT;
    reply_iphdr->tot_len = htons(sizeof(struct iphdr) + sizeof(struct icmphdr));
    reply_iphdr->id = htons(ICMP_ID);
    reply_iphdr->frag_off = req_iphdr->frag_off;
    reply_iphdr->ttl = req_iphdr->ttl;
    reply_iphdr->protocol = IPPROTO_ICMP;
    reply_iphdr->saddr = interf_ip;
    reply_iphdr->daddr = req_iphdr->saddr;

    reply_iphdr->check = 0;
    reply_iphdr->check = ip_checksum((void *) reply_iphdr,
                         sizeof(struct iphdr));
}
```

2.3.2.5 is_garbage()

```
int is_garbage (
            packet pkt,
            struct iphdr * ip_hdr )
```

Decides if a packet should be thrown away (invalid checksum / ttl $<$ 2). Also if the problem is with ttl, sends an ICMP Time Exceeded reply.

Parameters

pkt	The packet that should be inspected.
ip_hdr	The IP header of the packet that should be inspected

Returns

int 1 if an ICMP Time Exceeded reply has been sent, 0 otherwise.

Definition at line 115 of file ip_icmp.c.

```
{
    // If the checksum is not correct, throw it away.
    if (ip_checksum((void *) ip_hdr, sizeof(struct iphdr)) != 0)
        return 1;

    /* If the time to live is 0 or 1, then send an ICMP to tell that the given
     * time is exceeded.
    */
    if (ip_hdr->ttl < 2) {
        send_icmp(pkt, ICMP_TIME_EXCEEDED);
        return 1;
    }

    return 0;
}
```

2.3.2.6 send_icmp()

```
void send_icmp (
               packet req_pkt,
               uint8_t icmp_type )
```

Starting from a Echo Request, creates a new packet, completes it with the setter functions and sends it.

Parameters

req_pkt	The recieved packet.
icmp_type	The type of the ICMP.(Reply / Time Exceeded / Destination unreachable)

Definition at line 57 of file ip_icmp.c.

```
{
    // Extracting the headers from the REQUEST packet.
    struct ether_header *req_eth = (struct ether_header *) (req_pkt.payload);
    struct iphdr *req_ip = (struct iphdr *)
                            (req_pkt.payload + sizeof(struct ether_header));
    struct icmphdr *req_icmp = (struct icmphdr *) (req_pkt.payload +
                         sizeof(struct ether_header) + sizeof(struct iphdr));

    packet reply;

    reply.len = sizeof(struct ether_header) + sizeof(struct iphdr) +
                sizeof(struct icmphdr);
    memset(reply.payload, 0, MAX_LEN);
    reply.interface = req_pkt.interface;

    // Extracting the headers from the in progress reply packet.
    struct ether_header *rply_eth = (struct ether_header *) (reply.payload);
    struct iphdr *rply_ip = (struct iphdr *)
                            (reply.payload + sizeof(struct ether_header));
    struct icmphdr *rply_icmp = (struct icmphdr *)
                        (reply.payload + reply.len - sizeof(struct icmphdr));

    /* The old source becomes the new target and the old target becomes the new
    target
    */
    uint8_t source_mac[ETH_ALEN];
    get_interface_mac(req_pkt.interface, source_mac);

    eth_hdr_setter(rply_eth, req_eth->ether_shost, source_mac, ETHERTYPE_IP);

    uint32_t this_ip = inet_addr(get_interface_ip(req_pkt.interface));
    ip_hdr_setter(rply_ip, req_ip, this_ip);

    icmp_hdr_setter(rply_icmp, req_icmp, icmp_type);

    send_packet(&reply);
}
```

2.4 ip_icmp.h File Reference

Function prototypes for interacting with IP and ICMP headers and for managing ICMP request and replies.

```
#include "skel.h"
#include "router_funs.h"
```

Include dependency graph for ip_icmp.h: This graph shows which files directly or indirectly include this file:

Macros

- #define **BEST_EFFORT** 0
- #define **ICMP_CODE** 0
- #define **ICMP_ID** 1337
- #define **ICMP_REPLY_SENT** 1
- #define **ICMP_NO_REPLY_SENT** 0

Functions

- void ip_hdr_setter (struct iphdr ∗reply_iphdr, struct iphdr ∗req_iphdr, uint32_t interf_ip)

  Sets all the fields of an iphdr struct (The frag_off is taken from an ICMP request).
- void icmp_hdr_setter (struct icmphdr ∗reply_icmp, struct icmphdr ∗req_icmp, uint8_t icmp_type)

  Sets the code to 0 and the type to the given icmp_type. If the type is destination unreachable / time exceeded, then the id and sequence will be set to 0. Else if the type is an Echo Reply, it sets id and sequence to the same that were in the Echo Request.
- void send_icmp (packet req_pkt, uint8_t icmp_type)

  Starting from a Echo Request, creates a new packet, completes it with the setter functions and sends it.
- int icmp_handler (packet pkt, struct iphdr ∗ip_hdr)

  Deciding if it should send a reply and calls the functions needed for sending one.
- int is_garbage (packet pkt, struct iphdr ∗ip_hdr)

  Decides if a packet should be thrown away (invalid checksum / ttl $<$ 2). Also if the problem is with ttl, sends an ICMP Time Exceeded reply.
- void incremental_checksum (struct iphdr ∗ip_hdr)

  Updates the IPv4 checksum in an efficient way, avoiding using recalculation.

2.4.1 Detailed Description

Function prototypes for interacting with IP and ICMP headers and for managing ICMP request and replies.

Author

Radu-Andrei Dumitru

2.4.2 Function Documentation

2.4.2.1 icmp_handler()

```
int icmp_handler (
              packet pkt,
              struct iphdr * ip_hdr )
```

Deciding if it should send a reply and calls the functions needed for sending one.

Parameters

pkt	The recieved packet.
ip_hdr	The header of the recieved packet.

Returns

int 1 (ICMP_REPLY_SENT) if a reply has been sent, 0 (ICMP_NO_REPLY_SENT), otherwise.

Definition at line 96 of file ip_icmp.c.

```
{
    if (ip_hdr->protocol == IPPROTO_ICMP) {
        struct icmphdr *icmp_hdr = (struct icmphdr *) (pkt.payload +
        sizeof(struct ether_header) + sizeof(struct iphdr));

        // If is a REQUEST and is for the current router, send a reply.
        if (icmp_hdr->type == ICMP_ECHO) {
            uint32_t interf_ip = inet_addr(get_interface_ip(pkt.interface));
            if (ip_hdr->daddr == interf_ip) {
                send_icmp(pkt, ICMP_ECHOREPLY);
                return ICMP_REPLY_SENT;
            }
        }
    }

    return ICMP_NO_REPLY_SENT;
}
```

2.4.2.2 icmp_hdr_setter()

```
void icmp_hdr_setter (
             struct icmphdr * reply_icmp,
             struct icmphdr * req_icmp,
             uint8_t icmp_type )
```

Sets the code to 0 and the type to the given icmp_type. If the type is destination unreachable / time exceeded, then the id and sequence will be set to 0. Else if the type is an Echo Reply, it sets id and sequence to the same that were in the Echo Request.

Parameters

reply_icmp	The base IP header.
req_icmp	The ICMP header of the packet recieved.
icmp_type	The type of the ICMP. (Reply / Time Exceeded / Destination unreachable)

Definition at line 32 of file ip_icmp.c.

```
34 {
35     reply_icmp->type = icmp_type;
36     reply_icmp->code = ICMP_CODE;
37 
38     /* Check if it is destination unreachable / time exceeded / echo reply. For
39     reply, keep the same id and sequence.
40     */
41     if (reply_icmp->type == ICMP_UNREACH ||
42         reply_icmp->type == ICMP_TIME_EXCEEDED) {
43         reply_icmp->un.echo.id = 0;
44         reply_icmp->un.echo.sequence = 0;
45 
46     } else if (reply_icmp->type == ICMP_ECHOREPLY) {
47         reply_icmp->un.echo.id = req_icmp->un.echo.id;
48         reply_icmp->un.echo.sequence = req_icmp->un.echo.sequence;
49     }
50 
51     // Compute the checksum.
52     reply_icmp->checksum = 0;
53     reply_icmp->checksum = ip_checksum((void *) reply_icmp,
54                                  sizeof(struct icmp));
55 }
```

2.4.2.3 incremental_checksum()

```
void incremental_checksum (
             struct iphdr * ip_hdr )
```

Updates the IPv4 checksum in an efficient way, avoiding using recalculation.

Parameters

ip_hdr	The IP Header of the packet.

Definition at line 132 of file ip_icmp.c.

```
133 {
    ip_hdr->ttl--;

    uint32_t aux_sum;
    uint16_t mask = ((uint16_t) 1) << 8; // 0000 0001 0000 0000

    // Incrementing the first byte of the check sum.
    aux_sum = ntohs(ip_hdr->check) + mask;

    // Adding the carry to the end. (To perform one's complement sum).
    ip_hdr->check = htons((aux_sum + (aux_sum >> 16)));
}
```

2.4.2.4 ip_hdr_setter()

```
void ip_hdr_setter (
               struct iphdr * reply_iphdr,
               struct iphdr * req_iphdr,
               uint32_t interf_ip )
```

Sets all the fields of an iphdr struct (The frag_off is taken from an ICMP request).

Parameters

reply_iphdr	The base IP header.
req_iphdr	The IP header of the packet recieved.
interf_ip	The IP of the current interface.

Definition at line 12 of file ip_icmp.c.

```
{
    reply_iphdr->ihl = 5;
    reply_iphdr->version = 4;

    reply_iphdr->tos = BEST_EFFORT;
    reply_iphdr->tot_len = htons(sizeof(struct iphdr) + sizeof(struct icmphdr));
    reply_iphdr->id = htons(ICMP_ID);
    reply_iphdr->frag_off = req_iphdr->frag_off;
    reply_iphdr->ttl = req_iphdr->ttl;
    reply_iphdr->protocol = IPPROTO_ICMP;
    reply_iphdr->saddr = interf_ip;
    reply_iphdr->daddr = req_iphdr->saddr;

    reply_iphdr->check = 0;
    reply_iphdr->check = ip_checksum((void *) reply_iphdr,
                         sizeof(struct iphdr));
}
```

2.4.2.5 is_garbage()

```
int is_garbage (
               packet pkt,
               struct iphdr * ip_hdr )
```

Decides if a packet should be thrown away (invalid checksum / ttl $<$ 2). Also if the problem is with ttl, sends an ICMP Time Exceeded reply.

Parameters

pkt	The packet that should be inspected.
ip_hdr	The IP header of the packet that should be inspected

Returns

int 1 if an ICMP Time Exceeded reply has been sent, 0 otherwise.

Definition at line 115 of file ip_icmp.c.

```
{
    // If the checksum is not correct, throw it away.
    if (ip_checksum((void *) ip_hdr, sizeof(struct iphdr)) != 0)
        return 1;

    /* If the time to live is 0 or 1, then send an ICMP to tell that the given
     * time is exceeded.
    */
    if (ip_hdr->ttl < 2) {
        send_icmp(pkt, ICMP_TIME_EXCEEDED);
        return 1;
    }

    return 0;
}
```

2.4.2.6 send_icmp()

```
void send_icmp (
            packet req_pkt,
            uint8_t icmp_type )
```

Starting from a Echo Request, creates a new packet, completes it with the setter functions and sends it.

Parameters

req_pkt	The recieved packet.
icmp_type	The type of the ICMP.(Reply / Time Exceeded / Destination unreachable)

Definition at line 57 of file ip_icmp.c.

```
{
    // Extracting the headers from the REQUEST packet.
    struct ether_header *req_eth = (struct ether_header *) (req_pkt.payload);
    struct iphdr *req_ip = (struct iphdr *)
                        (req_pkt.payload + sizeof(struct ether_header));
    struct icmphdr *req_icmp = (struct icmphdr *) (req_pkt.payload +
                      sizeof(struct ether_header) + sizeof(struct iphdr));

    packet reply;

    reply.len = sizeof(struct ether_header) + sizeof(struct iphdr) +
              sizeof(struct icmphdr);
    memset(reply.payload, 0, MAX_LEN);
    reply.interface = req_pkt.interface;

    // Extracting the headers from the in progress reply packet.
    struct ether_header *rply_eth = (struct ether_header *) (reply.payload);
    struct iphdr *rply_ip = (struct iphdr *)
                            (reply.payload + sizeof(struct ether_header));
    struct icmphdr *rply_icmp = (struct icmphdr *)
                        (reply.payload + reply.len - sizeof(struct icmphdr));

    /* The old source becomes the new target and the old target becomes the new
    target
    */
    uint8_t source_mac[ETH_ALEN];
    get_interface_mac(req_pkt.interface, source_mac);

    eth_hdr_setter(rply_eth, req_eth->ether_shost, source_mac, ETHERTYPE_IP);

    uint32_t this_ip = inet_addr(get_interface_ip(req_pkt.interface));
    ip_hdr_setter(rply_ip, req_ip, this_ip);

    icmp_hdr_setter(rply_icmp, req_icmp, icmp_type);

    send_packet(&reply);
}
```

2.5 router_funs.c File Reference

This contains a comparator needed for qsort and the Longest Prefix Match done with Binary Search. Also, it contains a function that sets the Ether header to given parameters.

`#include "router_funs.h"`
Include dependency graph for router_funs.c:

Functions

- int lpm_comparator (const void ∗p1, const void ∗p2)

  Comparator function needed for qsort. If the prefixes are equal then sort by mask. Otherwise, sort by prefixes. (Descending)
- rtable_entry ∗ get_best_route (rtable_entry ∗rtable, size_t rtable_len, uint32_t dest_ip)

  Performs a binary search in order to find the longest prefix match.
- void eth_hdr_setter (struct ether_header ∗eth_hdr, uint8_t dest_mac[ETH_ALEN], uint8_t source_mac[ET↩H_ALEN], uint16_t type)

  Sets the destination and source MAC of an Ether header to the given MACs.

2.5.1 Detailed Description

This contains a comparator needed for qsort and the Longest Prefix Match done with Binary Search. Also, it contains a function that sets the Ether header to given parameters.

Author

Radu-Andrei Dumitru

2.5.2 Function Documentation

2.5.2.1 eth_hdr_setter()

```
void eth_hdr_setter (
              struct ether_header * eth_hdr,
              uint8_t dest_mac[ETH_ALEN],
              uint8_t source_mac[ETH_ALEN],
              uint16_t type )
```

Sets the destination and source MAC of an Ether header to the given MACs.

Parameters

eth_hdr	The base Ether header.
dest_mac	The MAC destination where the packet should arrive.
source_mac	The MAC of the packet starting point.
type	The type of the packet. (ARP / IPv4 in this case)

Definition at line 47 of file router_funs.c.

```
{
    memcpy(eth_hdr->ether_dhost, dest_mac, ETH_ALEN);
    memcpy(eth_hdr->ether_shost, source_mac, ETH_ALEN);
    eth_hdr->ether_type = ntohs(type);
}
```

2.5.2.2 get_best_route()

```
rtable_entry* get_best_route (
              rtable_entry * rtable,
              size_t rtable_len,
              uint32_t dest_ip )
```

Performs a binary search in order to find the longest prefix match.

Parameters

rtable	The routing table.
rtable_len	The length of the routing table.
dest_ip	The IP where the packet should arrive.

Returns

rtable_entry* The entry in the table with the longest prefix match.

Definition at line 22 of file router_funs.c.

```
{
    unsigned int mid, left = 0, right = rtable_len - 1;
    int best = -1;

    while (left <= right) {
        mid = left + (right - left) / 2;

        if ((dest_ip & rtable[mid].mask) == rtable[mid].prefix) {
            best = mid;
            right = mid - 1;
        } else if (((dest_ip & rtable[mid].mask) > rtable[mid].prefix)) {
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }

    if (best == -1)
        return NULL;

    return &rtable[best];
}
```

2.5.2.3 lpm_comparator()

```
int lpm_comparator (
            const void * p1,
            const void * p2 )
```

Comparator function needed for qsort. If the prefixes are equal then sort by mask. Otherwise, sort by prefixes. (Descending)

Parameters

p1	A pointer to a router table entry.
p2	A pointer to a router table entry.

Returns

int A positive or negative number, depending on the substraction of the prefixes / masks (if the prefixes are equal).

Definition at line 11 of file router_funs.c.

```
{
    struct route_table_entry *e1 = (struct route_table_entry *) p1;
    struct route_table_entry *e2 = (struct route_table_entry *) p2;

    if (e1->prefix == e2->prefix)
        return e2->mask - e1->mask;

    return e2->prefix - e1->prefix;
}
```

2.6 router_funs.h File Reference

Function prototypes for those needed by the router.

`#include "skel.h"`

Include dependency graph for router_funs.h: This graph shows which files directly or indirectly include this file:

Functions

- int lpm_comparator (const void *p1, const void *p2)

  Comparator function needed for qsort. If the prefixes are equal then sort by mask. Otherwise, sort by prefixes. (Descending)
- rtable_entry * get_best_route (rtable_entry *rtable, size_t rtable_len, uint32_t dest_ip)

  Performs a binary search in order to find the longest prefix match.
- void eth_hdr_setter (struct ether_header *eth_hdr, uint8_t dest_mac[ETH_ALEN], uint8_t source_mac[ETH_ALEN], uint16_t type)

  Sets the destination and source MAC of an Ether header to the given MACs.

2.6.1 Detailed Description

Function prototypes for those needed by the router.

Author

Radu-Andrei Dumitru

2.6.2 Function Documentation

2.6.2.1 eth_hdr_setter()

```
void eth_hdr_setter (
            struct ether_header * eth_hdr,
            uint8_t dest_mac[ETH_ALEN],
            uint8_t source_mac[ETH_ALEN],
            uint16_t type )
```

Sets the destination and source MAC of an Ether header to the given MACs.

Parameters

eth_hdr	The base Ether header.
dest_mac	The MAC destination where the packet should arrive.
source_mac	The MAC of the packet starting point.
type	The type of the packet. (ARP / IPv4 in this case)

Definition at line 47 of file router_funs.c.

```
{
    memcpy(eth_hdr->ether_dhost, dest_mac, ETH_ALEN);
    memcpy(eth_hdr->ether_shost, source_mac, ETH_ALEN);
    eth_hdr->ether_type = ntohs(type);
}
```

2.6.2.2 get_best_route()

```
rtable_entry* get_best_route (
            rtable_entry * rtable,
            size_t rtable_len,
            uint32_t dest_ip )
```

Performs a binary search in order to find the longest prefix match.

Parameters

rtable	The routing table.
rtable_len	The length of the routing table.
dest_ip	The IP where the packet should arrive.

Returns

rtable_entry∗ The entry in the table with the longest prefix match.

Definition at line 22 of file router_funs.c.

```
{
    unsigned int mid, left = 0, right = rtable_len - 1;
    int best = -1;

    while (left <= right) {
        mid = left + (right - left) / 2;

        if ((dest_ip & rtable[mid].mask) == rtable[mid].prefix) {
            best = mid;
            right = mid - 1;
        } else if (((dest_ip & rtable[mid].mask) > rtable[mid].prefix)) {
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }

    if (best == -1)
        return NULL;

    return &rtable[best];
}
```

2.6.2.3 lpm_comparator()

```
int lpm_comparator (
            const void * p1,
            const void * p2 )
```

Comparator function needed for qsort. If the prefixes are equal then sort by mask. Otherwise, sort by prefixes. (Descending)

Parameters

p1	A pointer to a router table entry.
p2	A pointer to a router table entry.

Returns

int A positive or negative number, depending on the substraction of the prefixes / masks (if the prefixes are equal).

Definition at line 11 of file router_funs.c.

```
{
    struct route_table_entry *e1 = (struct route_table_entry *) p1;
    struct route_table_entry *e2 = (struct route_table_entry *) p2;

    if (e1->prefix == e2->prefix)
        return e2->mask - e1->mask;

    return e2->prefix - e1->prefix;
}
```

Index