编程作业实验课

陈嘉杰

2021.11

HAL 框架如何工作

- 支持两种工作模式：真实模式和文件模式
- 真实模式会直接在网卡上抓包 & 发送以太网帧
  - 基于 libpcap
- 文件模式则直接从 pcap 文件中读取以太网帧
- 提供了统一的接口

HAL 如何使用

- https://lab.cs.tsinghua.edu.cn/router/doc/framework/

  它提供了以下这些函数：

  1. `HAL_Init`：使用 HAL 库的第一步，**必须调用且仅调用一次**，需要提供每个网口上绑定的 IP 地址，第一个参数表示是否打开 HAL 的测试输出，十分建议在调试的时候打开它
  2. `HAL_GetTicks`：获取从启动到当前时刻的毫秒数
  3. `HAL_GetNeighborMacAddress`：从 NDP 表中查询 IPv6 地址对应的 MAC 地址，在找不到的时候会发出 NDP 请求
  4. `HAL_GetInterfaceMacAddress`：获取指定网口上绑定的 MAC 地址
  5. `HAL_ReceiveIPPacket`：从指定的若干个网口中读取一个 IPv6 报文，并得到源 MAC 地址和目的 MAC 地址等信息；它还会在内部处理 NDP 表的更新和响应，需要定期调用
  6. `HAL_SendIPPacket`：向指定的网口发送一个 IPv6 报文

怎么上手编程作业

- IDE
  - VSCode：跳转到定义
- Linux/macOS 环境
  - Windows 用户安装 WSL2 并配合 VSCode 的 WSL Remote 功能使用
  - WSL 配置：https://oi-wiki.org/tools/wsl/ https://physics-data.meow.plus/faq/env/wsl/
- 阅读文档
  - 作业目录下的 README：文字描述和 RFC 链接
  - 头文件中的注释
  - 参考测试数据

常用的结构体 in6_addr

- 采用 IDE 的跳转功能
- 注意使用跨平台的表示
  - macOS 和 Linux 结构体定义有区别
  - 用 in6_addr.s6_addr
- IPv6 地址是 16 个字节

```
struct in6_addr
  {
    union
      {
  uint8_t __u6_addr8[16];
  uint16_t __u6_addr16[8];
  uint32_t __u6_addr32[4];
      } __in6_u;
#define s6_addr     __in6_u.__u6_addr8
#ifdef __USE_MISC
# define s6_addr16    __in6_u.__u6_addr16
# define s6_addr32    __in6_u.__u6_addr32
#endif
  };
```

常用的结构体 ether_addr

- MAC 地址是 6 个字节
- 用结构体的好处是可以按值传递，而传数组不是
- 用 ether_addr_octet 访问数组

```
struct ether_addr
{
  uint8_t ether_addr_octet[ETH_ALEN];
} __attribute__ ((__packed__));
```

常用的结构体 ip6_hdr

- IPv6 头部字段如右图
- 用下面的名字来访问
  - ip6_plen: Payload Length
  - ip6_nxt: Next Header
  - ip6_hlim: Hop Limit
- 注意字节序
  - Htonl/ntohl/htons/ntohs
  - 如果看到一个不正常的很大的数，就要想到是不是字节序没有处理

```
struct ip6_hdr {
        union {
                struct ip6_hdrctl {
                        u_int32_t ip6_un1_flow; /* 20 b
                        u_int16_t ip6_un1_plen; /* payl
                        u_int8_t  ip6_un1_nxt;  /* next
                        u_int8_t  ip6_un1_hlim; /* hop
                } ip6_un1;
                u_int8_t ip6_un2_vfc;   /* 4 bits versi
        } ip6_ctlun;
        struct in6_addr ip6_src;        /* source addre
        struct in6_addr ip6_dst;        /* destination
} __attribute__((__packed__));

#define ip6_vfc         ip6_ctlun.ip6_un2_vfc
#define ip6_flow        ip6_ctlun.ip6_un1.ip6_un1_flow
#define ip6_plen        ip6_ctlun.ip6_un1.ip6_un1_plen
#define ip6_nxt         ip6_ctlun.ip6_un1.ip6_un1_nxt
#define ip6_hlim        ip6_ctlun.ip6_un1.ip6_un1_hlim
#define ip6_hops        ip6_ctlun.ip6_un1.ip6_un1_hlim
```

常用的结构体 udphdr

- UDP 头部比较简单
- 注意端序
  - 16 位，用 htons/ntohs
- 其中 checksum 计算比较复杂，后面会讲

```
/*
 * Udp protocol header.
 * Per RFC 768, September, 1981.
 */
struct udphdr {
        u_short uh_sport;               /* source port */
        u_short uh_dport;               /* destination port
        u_short uh_ulen;                /* udp length */
        u_short uh_sum;                 /* udp checksum */
};
```

常用结构体 icmp6_hdr

- 编程作业中只涉及 checksum 计算
- 还有一些相关的结构体，用于 ICMPv6 子协议
  - ND/NA
  - RD/RA

```
struct icmp6_hdr
  {
    uint8_t     icmp6_type;   /* type field */
    uint8_t     icmp6_code;   /* code field */
    uint16_t    icmp6_cksum;  /* checksum field */
    union
      {
  uint32_t  icmp6_un_data32[1]; /* type-specific field */
  uint16_t  icmp6_un_data16[2]; /* type-specific field */
  uint8_t   icmp6_un_data8[4];  /* type-specific field */
      } icmp6_dataun;
  };
```

Internet Checksum 计算

- IPv6 Pseudo Header 组装
  - 由于 IPv6 头部没有校验和
  - UDP/ICMPv6 校验和是必需的
- 构造右图的数据
  - 注意 length 的端序
    - 先 ntohs，转为 32 位后再 htonl
- Header 之后紧接 UDP/ICMPv6 头部和载荷
- 这一段数据进行校验和计算

IPv6 pseudo header format

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source IPv6 Address																															
4	32																																
8	64																																
12	96																																
16	128	Destination IPv6 Address																															
20	160																																
24	192																																
28	224																																
32	256	UDP Length																															
36	288	Zeroes																								Next Header = Protocol[11]							
40	320	Source Port																Destination Port															
44	352	Length																Checksum															
48	384+	Data																															

Internet Checksum 计算

- 如何进行拆分成 16 位整数
  - 连续的两个字节合成一个
  - (Data[2*i] << 8) + Data[2*i+1]
  - 判断奇数的情况：最后补 0
- 如何求和
  - 按顺序求和
  - 把高于 16 位的部分挪到低位相加
  - 一次不够，循环直到高于 16 位的部分为 0
- 校验时直接判断结果是否等于 0xFFFF
- 计算时
  - 先把 checksum 设为 0
  - 求和，结果取反填入校验和
  - 注意端序

	Byte-by-byte		"Normal" Order
Byte 0/1:	00	01	0001
Byte 2/3:	f2	03	f203
Byte 4/5:	f4	f5	f4f5
Byte 6/7:	f6	f7	f6f7
	---	---	-----
Sum1:	2dc	1f0	2ddf0
	dc	f0	ddf0
Carrys:	1	2	2
	--	--	----
Sum2:	dd	f2	ddf2
Final Swap:	dd	f2	ddf2

Internet Checksum 计算

- 按照上面的计算方法计算出来的，校验和取值范围是 0x0000-0xFFFE
- 0x0000 和 0xFFFF 计算上是等价的
- UDP 的特殊处理
  - IPv4 中，UDP 的校验和计算是可选的，因此 0x0000 表示没有计算校验和，0xFFFF 对应了原来的 0x0000，校验和取值范围是 0x0000-0xFFFF
  - IPv6 中，UDP 的校验和计算是必需的，因此校验和取值范围是 0x0001-0xFFFF，不允许 0x0000

Internet Checksum 计算

- 对自己严格，对他人宽松
- 接收
  - ICMPv6 进行校验时，只要求和结果等于 0xFFFF 即可
  - UDPv6 进行校验时，求和结果等于 0xFFFF，并且校验和不等于 0x0000
- 发送
  - ICMPv6 计算校验和的时候，当校验和计算结果是 0x0000，就应当写入 0x0000，而不是错误的 0xFFFF
  - UDPv6 计算校验和的时候，当校验和计算结果是 0x0000，就应当写入 0xFFFF，而不是错误的 0x0000

Lookup 实现

- 如何实现路由表?
- 实现什么接口
  - 插入/更新/删除：精确匹配
  - 查询：最长前缀查询
  - 工具函数：前缀和前缀长度之间的转换

Lookup 实现

- 一个简单的实现方法
- 用 std::vector 保存所有的路由表项
- 精确匹配的时候，循环找到前缀地址和前缀长度都匹配的项目，然后更新
- 最长前缀匹配，则对每一项进行判断：利用 len_to_mask 生成掩码，将掩码与查询地址取 AND，然后看是否等于前缀地址。然后记录匹配的项目中前缀长度最长的那一个作为结果
- 如果想减少搜索次数，可以按照前缀长度由长到短进行排序。那么第一次匹配就是最长匹配

Lookup 实现

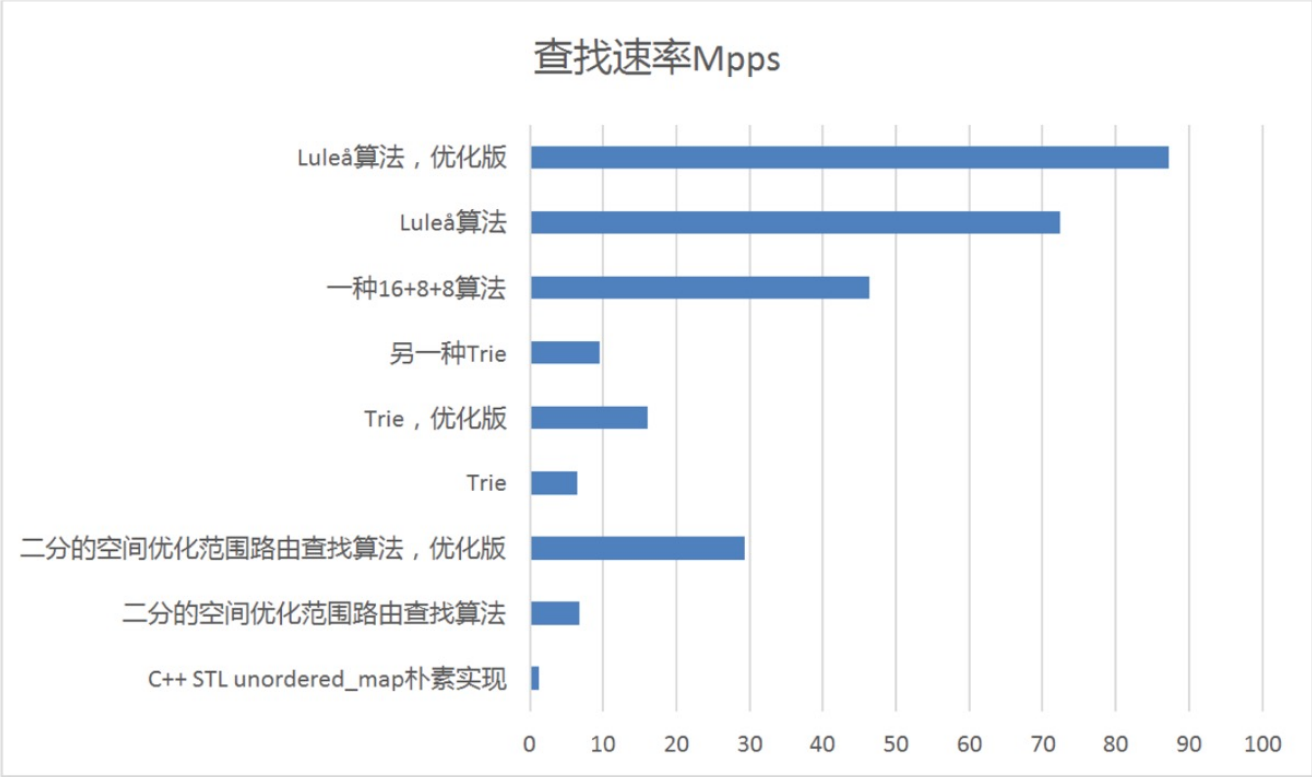
- 基于 Trie 树的查找算法
- 将 IPv6 地址视为长度为 128 的 01 串
- 将 IPv6 前缀对应到 Trie 树上的部分结点
- 最长前缀匹配
  - 从根结点开始，0 向左走，1 向右走
  - 记录当前匹配到的最后一个（最深）的结点
  - 在树上走 128 步

Lookup 实现

- 如何压缩 Trie 深度?
- 走 128 步太多，可以一次走几步
- 比如设计一个 16 叉树，一次用 4 位 IPv6 地址做索引
- 此时一个路由表项可能对应多个树中的结点
  - 比如：::/1 的路由表项，第一级中，0000 ~ 0111 都是匹配的，1000~1111 都不匹配

Lookup 实现

- 其他算法?
- 右图是王逸松、于纪平和谭闻德三位学长做的 IPv4 路由表查询效率结果比较



路由器实现

1. 初始化路由表，加入直连路由
2. 进入路由器主循环
3. 如果距离上一次发送已经超过了 5 秒，就发送完整的路由表到所有的接口
4. 接收 IPv6 分组，如果没有收到就跳到第 2 步
5. 检查 IPv6 分组的完整性和正确性
6. 判断 IPv6 分组需要转发还是进入 RIPng/ICMPv6 协议处理
7. 如果是 RIPng 分组，如果是 Request，就构造对应的 Response；如果是 Response，按照条目更新路由表
8. 如果是 ICMPv6 分组，如果是 Echo Request，就回复 Echo Reply
9. 如果这个分组要转发，判断 Hop Limit，如果小于或等于 1，就回复 ICMP Time Exceeded
10. 如果 Hop Limit 正常，查询路由表，如果找到了，就转发给下一跳
11. 如果不在路由表中，就回复 ICMP Destination Unreachable
12. 跳到第 2 步进入下一次循环处理

路由器实现（数据平面）

1. 初始化路由表，加入直连路由
2. **进入路由器主循环**
3. 如果距离上一次发送已经超过了 5 秒，就发送完整的路由表到所有的接口
4. **接收 IPv6 分组，如果没有收到就跳到第 2 步**
5. 检查 IPv6 分组的完整性和正确性
6. **判断 IPv6 分组需要转发还是进入 RIPng/ICMPv6 协议处理**
7. 如果是 RIPng 分组，如果是 Request，就构造对应的 Response；如果是 Response，按照条目更新路由表
8. 如果是 ICMPv6 分组，如果是 Echo Request，就回复 Echo Reply
9. **如果这个分组要转发，判断 Hop Limit，如果小于或等于 1，就回复 ICMP Time Exceeded**
10. **如果 Hop Limit 正常，查询路由表，如果找到了，就转发给下一跳**
11. **如果不在路由表中，就回复 ICMP Destination Unreachable**
12. **跳到第 2 步进入下一次循环处理**

路由器实现（控制平面）

1. **初始化路由表，加入直连路由**
2. 进入路由器主循环
3. **如果距离上一次发送已经超过了 5 秒，就发送完整的路由表到所有的接口**
4. 接收 IPv6 分组，如果没有收到就跳到第 2 步
5. 检查 IPv6 分组的完整性和正确性
6. **判断 IPv6 分组需要转发还是进入 RIPng/ICMPv6 协议处理**
7. **如果是 RIPng 分组，如果是 Request，就构造对应的 Response；如果是 Response，按照条目更新路由表**
8. **如果是 ICMPv6 分组，如果是 Echo Request，就回复 Echo Reply**
9. 如果这个分组要转发，判断 Hop Limit，如果小于或等于 1，就回复 ICMP Time Exceeded
10. 如果 Hop Limit 正常，查询路由表，如果找到了，就转发给下一跳
11. 如果不在路由表中，就回复 ICMP Destination Unreachable
12. 跳到第 2 步进入下一次循环处理

路由器实现

- 对照代码理解上面讲的路由器的功能实现
- 结合实验文档中的要求实现
  - https://lab.cs.tsinghua.edu.cn/router/doc/requirement/#_5

路由器实现

- 必须实现的有：
  - 转发功能，支持直连路由和间接路由，包括 Hop Limit 减一，查表并向正确的 interface 发送出去。
  - 周期性地向所有端口发送 RIPng Response（周期为 5s，而不是 RFC 2080 Section 2.3 Timers 要求的 30s），目标地址为 RIPng 的组播地址。
  - 对收到的 RIPng Request 生成 RIPng Response 进行回复，目标地址为 RIPng Request 的源地址。
  - 实现水平分割（split horizon）和毒性反转（reverse poisoning）。
  - 收到 RIPng Response 时，对路由表进行维护，处理 RIPng 中 metric=16 的情况。
  - 在发送的 RIPng Response 大小超过 MTU 时进行拆分。

路由器实现

- 必须实现的有：
  - 对 ICMPv6 Echo Request 进行 ICMPv6 Echo Reply 的回复，见 RFC 4443 Echo Reply Message。
  - 在接受到 IPv6 packet，按照目的地址在路由表中查找不到路由的时候，回复 ICMPv6 Destination Unreachable (No route to destination)，见 RFC 4443 Section 3.1 Destination Unreachable Message。
  - 在 Hop Limit 减为 0 时，回复 ICMPv6 Time Exceeded (Hop limit exceeded in transit)，见 RFC 4443 Section 3.3 Time Exceeded Message。

路由器实现

- 可选实现的有（不加分，但对调试有帮助）：
  - 定期或者在更新的时候向 stdout/stderr 打印最新的 RIP 路由表。
  - 在路由表出现更新的时候立即发送 RIPng Response（完整或者增量），可以加快路由表的收敛速度。
  - 路由的超时（Timeout）和垃圾回收（Garbage Collection）定时器。
  - 程序启动时向所有 interface 发送 RIPng Request。

DDL 提醒!

- 编程作业：校历第 1 周-第 9 周周日（2021 年 11 月 14 日）22:00:00
- 真机评测（单人）：校历第 10 周-第 12 周周日（2021 年 12 月 5 日）22:00:00
- 真机评测（组队）：校历第 13 周-第 14 周周日（2021 年 12 月 19 日）22:00:00
- 记得在 TANLabs 上标记最终提交!