1 mDNS 简介

mDNS（Multicast DNS，多播 DNS）是一个协议，允许在无中央 DNS 服务器的局域网中通过多播 DNS 查询来实现设备的自动服务发现和主机名解析。它是 Zeroconf（零配置网络）的一部分，主要用于在局域网中实现无需手动配置的自动网络发现功能

2 项目简介

利用两台虚拟机模拟两台局域网设备，第一台模拟器主机名 study03 运行服务端，第二台模拟器主机名 study02 运行客户端。客户端输出服务端主机名、IPv4、IPv6 地址、服务器名称（MQTTServer）、类型（_mqtt._tcp）、地址（mqtts://tb.com）

3 配置环境

如果两台虚拟机上运行需要配置环境，如果在一台虚拟机上两个终端运行则不需要此步骤。

①确保虚拟机在同一局域网

命令查看 IP 地址，根据 IP 地址可以确定两台虚拟机是否在同一个子网

Ifconfig

192.168.118.130

192.168.118.129

根据 IP 地址确定两台虚拟机是否在同一个子网

②确认虚拟机的网络模式

两台虚拟机均为 NAT 模式可以通信

③确保防火墙没有阻止 UDP 5353 端口

检查防火墙状态

sudo ufw status

设置允许 5353 端口

sudo ufw allow 5353/udp

也可以直接关闭防火墙

sudo ufw disable

4 安装依赖

sudo apt-get update

sudo apt-get install avahi-daemon libavahi-client-dev libavahi-common-dev

三个包作用？

5 编译

进入程序文件路径，编译程序

gcc -o server.out server.c -lavahi-client -lavahi-common

gcc -o client.out client.c -lavahi-client -lavahi-common

6 开启守护进程

avahi-daemon 包是 Avahi 服务的核心守护进程，它在网络上提供多种服务，包括服务发现、名称解析和服务注册。Avahi 是一个开源的实现了 Zeroconf（零配置网络）标准的服务，主要用于在本地网络中自动发现服务和设备。

命令开启守护进程并检查状态（是否开启）

sudo systemctl start avahi-daemon

sudo systemctl status avahi-daemon

停止/重启/开机自启/关闭开机自启 服务

sudo systemctl stop/restart/enable/disable avahi-daemon

6 运行

分别运行服务端和客户端

./server.out

./client.out

7 报错

如果遇到报错，请检查头文件是否完整等等

检查服务命令

avahi-browse -r _http._tcp

avahi-browse -a

常见问题

报错：Local name collision

客户端多次打印

8 程序

//服务端

```
#include <avahi-client/client.h>
#include <avahi-client/publish.h>
#include <avahi-common/thread-watch.h>
#include <avahi-common/error.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <ifaddrs.h>
#include <arpa/inet.h>

int main() {
    // 创建 AvahiThreadedPoll 实例
    AvahiThreadedPoll *poll = avahi_threaded_poll_new();
    if(!poll) {
        fprintf(stderr, "Failed to create threaded poll.\n");
        return EXIT_FAILURE;
    }

    // 初始化 Avahi 客户端
    AvahiClient *client = avahi_client_new(avahi_threaded_poll_get(poll), AVAHI_CLIENT_NO_FAIL, NULL, NULL, NULL);

    if(!client) {
        fprintf(stderr, "Failed to create Avahi client.\n");
        avahi_threaded_poll_free(poll);  // 释放线程池
        return EXIT_FAILURE;
    }

    // 创建服务
    AvahiEntryGroup *group = avahi_entry_group_new(client, NULL, NULL);

    // 修改服务名称和类型为 mqtt
    const char *service_name = "MQTTServer";
```

```
    const char *service_type = "_mqtt._tcp";
    const char *mqtt_server_address = "mqtts://tb.com";

    // 获取本机的 IP 地址
    struct ifaddrs *ifaddr, *ifa;
    char ipv4_addr[INET_ADDRSTRLEN] = "";
    char ipv6_addr[INET6_ADDRSTRLEN] = "";

    if(getifaddrs(&ifaddr) == -1) {
        perror("getifaddrs");
        return EXIT_FAILURE;
    }

    for(ifa = ifaddr; ifa != NULL; ifa = ifa->ifa_next) {
        if(ifa->ifa_addr == NULL) continue;

        int family = ifa->ifa_addr->sa_family;
        if(family == AF_INET) {
            inet_ntop(AF_INET, &((struct sockaddr_in *)ifa->ifa_addr)->sin_addr, ipv4_addr, INET_ADDRSTRLEN);
        }
        else if(family == AF_INET6) {
            inet_ntop(AF_INET6, &((struct sockaddr_in6 *)ifa->ifa_addr)->sin6_addr, ipv6_addr, INET6_ADDRSTRLEN);
        }
    }
    freeifaddrs(ifaddr);

    // 创建 TXT 记录
    AvahiStringList *txt = NULL;
    txt = avahi_string_list_add_pair(txt, "mqtt_server_address", mqtt_server_address);
    if(strlen(ipv4_addr) > 0) txt = avahi_string_list_add_pair(txt, "ipv4", ipv4_addr);
    if(strlen(ipv6_addr) > 0) txt = avahi_string_list_add_pair(txt, "ipv6", ipv6_addr);

    if(avahi_entry_group_add_service_strlst(group, AVAHI_IF_UNSPEC, AVAHI_PROTO_UNSPEC, 0, service_name, service_type, NULL, NULL, 0, txt) < 0) {
        fprintf(stderr, "Failed to add service TXT record: %s\n", avahi_strerror(avahi_client_errno(client)));
        avahi_client_free(client);
        avahi_threaded_poll_free(poll);
        return EXIT_FAILURE;
    }

    // 提交服务
```

```
    if(avahi_entry_group_commit(group) < 0) {
        fprintf(stderr, "Failed to commit entry group: %s\n", avahi_strerror(avahi_client_errno(client)));
        avahi_client_free(client);
        avahi_threaded_poll_free(poll);  // 释放线程池
        return EXIT_FAILURE;
    }

    // 启动事件循环
    avahi_threaded_poll_start(poll);

    // 让程序持续运行
    printf("Service started. Press Ctrl+C to exit.\n");
    while (1) {
        pause();  // 等待信号
    }

    // 释放资源
    avahi_entry_group_free(group);
    avahi_client_free(client);
    avahi_threaded_poll_free(poll);

    return EXIT_SUCCESS;
}
```

//客户端

```
#include <avahi-client/client.h>
#include <avahi-client/lookup.h>
#include <avahi-common/thread-watch.h>
#include <avahi-common/error.h>
#include <avahi-common/malloc.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <netinet/in.h>
#include <arpa/inet.h>

// 标志位，用于确保只打印一次服务信息
int service_printed = 0;

int ipv4_printed = 0;   // IPv4 解析完成标志
int ipv6_printed = 0;   // IPv6 解析完成标志
```

```
18.
19.  void resolve_callback(AvahiServiceResolver *resolver, AvahiIfIndex interface, AvahiProtocol protoc
     ol, AvahiResolverEvent event,
20.      const char *name, const char *type, const char *domain, const char *hostname,
21.      const AvahiAddress *address, uint16_t port, AvahiStringList *txt, AvahiLookupResultFlags flags,
      void *userdata) {
22.      if(event == AVAHI_RESOLVER_FOUND) {
23.          char address_str[AVAHI_ADDRESS_STR_MAX];
24.          avahi_address_snprint(address_str, sizeof(address_str), address);
25.
26.          if(protocol == AVAHI_PROTO_INET && !ipv4_printed) {
27.              printf("Service Hostname: %s\n", hostname);
28.              printf("Resolved IPv4 IP Address: %s\n", address_str);
29.              ipv4_printed = 1;
30.          }
31.          else if(protocol == AVAHI_PROTO_INET6 && !ipv6_printed) {
32.              printf("Resolved IPv6 IP Address: %s\n", address_str);
33.              ipv6_printed = 1;
34.          }
35.
36.          // 只有在两种地址都打印后，才打印服务信息并设置标志位
37.          if(ipv4_printed && ipv6_printed && !service_printed) {
38.              printf("Service Name: %s\n", name);
39.              printf("Service Type: %s\n", type);
40.
41.              while(txt) {
42.                  char *key, *value;
43.                  avahi_string_list_get_pair(txt, &key, &value, NULL);
44.                  if(strcmp(key, "mqtt_server_address") == 0) {
45.                      printf("Service Address (from TXT record): %s\n", value);
46.                  }
47.                  avahi_free(key);
48.                  avahi_free(value);
49.                  txt = avahi_string_list_get_next(txt);
50.              }
51.              service_printed = 1;
52.          }
53.      }
54.
55.      avahi_service_resolver_free(resolver);
56.  }
57.
58.  void browse_callback(AvahiServiceBrowser *browser, AvahiIfIndex interface, AvahiProtocol protocol,
     AvahiBrowserEvent event,
```

```
59.       const char *name, const char *type, const char *domain, AvahiLookupResultFlags flags, void *us
    erdata) {
60.       AvahiClient *client = userdata;
61.       if(event == AVAHI_BROWSER_NEW) {
62.           avahi_service_resolver_new(client, interface, AVAHI_PROTO_INET, name, type, domain, AVAHI_
    PROTO_UNSPEC, 0, resolve_callback, client);
63.           avahi_service_resolver_new(client, interface, AVAHI_PROTO_INET6, name, type, domain, AVAHI
    _PROTO_UNSPEC, 0, resolve_callback, client);
64.       }
65.   }
66.
67.
68.   int main() {
69.       AvahiThreadedPoll *poll = avahi_threaded_poll_new();
70.       if(!poll) {
71.           fprintf(stderr, "Failed to create threaded poll.\n");
72.           return EXIT_FAILURE;
73.       }
74.
75.       AvahiClient *client = avahi_client_new(avahi_threaded_poll_get(poll), AVAHI_CLIENT_NO_FAIL, NU
    LL, NULL, NULL);
76.       if(!client) {
77.           fprintf(stderr, "Failed to create Avahi client.\n");
78.           avahi_threaded_poll_free(poll);
79.           return EXIT_FAILURE;
80.       }
81.
82.       AvahiServiceBrowser *browser = avahi_service_browser_new(client, AVAHI_IF_UNSPEC, AVAHI_PROTO_
    UNSPEC, "_mqtt._tcp", NULL, 0, browse_callback, client);
83.       if(!browser) {
84.           fprintf(stderr, "Failed to create service browser.\n");
85.           avahi_client_free(client);
86.           avahi_threaded_poll_free(poll);
87.           return EXIT_FAILURE;
88.       }
89.
90.       printf("Service browser started.\n");
91.
92.       avahi_threaded_poll_start(poll);
93.
94.       while(1) {
95.           pause();
96.       }
97.
```

```
    avahi_service_browser_free(browser);
    avahi_client_free(client);
    avahi_threaded_poll_free(poll);

    return EXIT_SUCCESS;
}
```