

网络工程 本科实验报告

实验名称：基于 OpenFlow 的 SDN 交换机实验

学 员 姓 名	<u>王李烜</u>	学 号	<u>202202001046</u>
学 员 姓 名	<u>廖中煜</u>	学 号	<u>202202001032</u>
学 员 姓 名	<u>王誉潞</u>	学 号	<u>202202001051</u>
指 导 教 员	<u>胡罡</u>	职 称	<u>教授</u>
实 验 室	<u>306-707</u>	实 验 时 间	<u>2025.1.2</u>

《本科实验报告》填写说明

实验报告内容编排应符合以下要求：

(1) 采用 A4 (21cm×29.7cm) 白色复印纸，单面黑字。上下左右各侧的页边距均为 3cm；缺省文档网格：字号为小 4 号，中文为宋体，英文和阿拉伯数字为 Times New Roman，每页 30 行，每行 36 字；页脚距边界为 2.5cm，页码置于页脚、居中，采用小 5 号阿拉伯数字从 1 开始连续编排，封面不编页码。

(2) 报告正文最多可设四级标题，字体均为黑体，第一级标题字号为 4 号，其余各级标题为小 4 号；标题序号第一级用“一、”、“二、”……，第二级用“(一)”、“(二)”……，第三级用“1.”、“2.”……，第四级用“(1)”、“(2)”……，分别按序连续编排。

(3) 正文插图、表格中的文字字号均为 5 号。

目录

1 实验概述	5
1.1 实验内容	5
1.2 实验要求	5
2 编写 SDN 交换机源码处理 OpenFlow 协议消息	5
2.1 本次实验所编写的协议	5
2.1.1 实现的协议	5
2.1.2 组内分工	6
2.2 相关数据结构	6
2.3 消息头构建	6
2.4 消息处理函数	6
2.4.1 处理 Hello 消息	7
2.4.2 处理 Features Request 消息	7
2.4.3 处理 Get Config Request 消息	7
2.4.4 处理 Description Request 消息	8
2.4.5 处理 Flow Stats Request 消息	8
2.4.6 处理 Aggregate Stats Request 消息	9
2.4.7 处理 Table Stats Request 消息	9
2.4.8 处理 Port Stats Request 消息	10
2.4.9 处理 Group Features Request 消息	10
2.4.10 处理 Port Description Request 消息	11
2.4.11 处理 Packet Out 消息	11
2.4.12 处理 Role Request 消息	12
2.5 实验结果	12
2.5.1 OFPT_ROLE_REPLY	12
2.5.2 OFPT_MULTIPART_REPLY	13
3 实验总结	15
参考文献	16

图目录

Figure 1: OFPT_ROLE_REQUEST 消息	13
Figure 2: OFPT_ROLE_REPLY 消息	13
Figure 3: OFPMP_DESC 消息	14
Figure 4: OFPMP_TABLE 消息	14

1 实验概述

1.1 实验内容

本次实验中有三个小实验：

1. 基于 Openbox-S4 测试 SDN 交换功能
2. 基于 SDN 交换机源码处理 openflow 协议消息
3. 实验三 使用 REST API 接口查询交换机的相关功能数据

本实验报告重点分析第二个小实验的实现细节。

1.2 实验要求

1. 熟悉 Openbox-S4 的基本功能
2. 熟悉 OpenFlow 协议的基本消息格式
3. 熟悉 OpenFlow 协议的基本消息处理流程
4. 熟悉 REST API 接口的基本使用

2 编写 SDN 交换机源码处理 OpenFlow 协议消息

本次实验在 `main_user_openflow.c` 文件中实现了 OpenFlow 协议处理逻辑。代码通过多个函数处理不同类型的 OpenFlow 消息，包括 Hello 消息、Features Request 消息、Flow Stats Request 消息等。每个函数都根据消息类型生成相应的回复消息，并通过 `send_openflow_message` 函数发送给控制器。以下将逐一分析每个函数的实现细节。

2.1 本次实验所编写的协议

本次实验实现了 OpenFlow 协议中的大部分消息处理功能。实验代码涵盖了从交换机特性查询到流表统计、端口统计等多种消息类型的处理。以下是实验编写协议的总结。

2.1.1 实现的协议

- OFPT_FEATURES_REQUEST: 用于获取交换机支持的流表数量、缓冲区大小等特性信息。对应的处理函数为 `handle_ofpmsg_features_request`。
- OFPT_GET_CONFIG_REQUEST: 用于查询交换机的配置信息，如 Miss Send Length 等。对应的处理函数为 `handle_ofpmsg_get_config_request`。
- OFPT_MULTIPART_REQUEST: 用于处理多种统计信息请求，包括交换机描述信息、流表信息、端口统计信息等。对应的处理函数包括：
 - `handle_ofpmsg_desc`
 - `handle_ofpmsg_flow_stats`
 - `handle_ofpmsg_aggregate`
 - `handle_ofpmsg_table`
 - `handle_ofpmsg_port_stats`
 - `handle_ofpmsg_group_features`
 - `handle_ofpmsg_port_desc`

- OFPT_PACKET_OUT: 用于处理控制器发送的数据包, 并根据动作指示将数据包从指定端口发送出去。对应的处理函数为 `handle_ofpmsg_packet_out`。
- OFPT_ROLE_REQUEST: 用于配置交换机的角色 (如主控制器、从控制器等)。对应的处理函数为 `handle_ofpmsg_role_request`。

2.1.2 组内分工

本次实验由组内成员共同完成, 每个成员负责不同的消息处理函数。以下是每个成员负责的函数列表。

分工	函数
王李烜	<code>handle_ofpmsg_features_request</code>
	<code>handle_ofpmsg_desc</code>
	<code>handle_ofpmsg_flow_stats</code>
	<code>handle_ofpmsg_aggregate</code>
	<code>handle_ofpmsg_table</code>
	<code>handle_ofpmsg_port_desc</code>
	<code>handle_ofpmsg_role_request</code>
廖中煜	<code>handle_ofpmsg_get_config_request</code>
	<code>handle_ofpmsg_packet_out</code>
王誉潞	<code>handle_ofpmsg_port_stats</code>
	<code>handle_ofpmsg_group_features</code>

2.2 相关数据结构

在 `main_user_openflow.c` 中, 定义了一些关键的数据结构和全局变量, 用于处理 OpenFlow 协议的消息。主要的数据结构包括 `ofp_header`、`ofp_switch_features`、`ofp_flow_stats` 和 `ofp_port_stats` 等。这些数据结构用于处理 OpenFlow 协议中的不同消息类型, 如交换机特性请求、流表统计请求、端口统计请求等。

`ofp_header` 结构体用于表示 OpenFlow 消息头, 包含协议版本、消息类型、消息长度和事务 ID 等字段。`ofp_switch_features` 结构体用于表示交换机的特性信息, 如数据路径 ID、缓冲区数量、流表数量等。`ofp_flow_stats` 结构体用于表示流表统计信息, 如流表项长度、优先级、数据包计数等。`ofp_port_stats` 结构体用于表示端口统计信息, 如端口号、接收数据包数、发送数据包数等。

2.3 消息头构建

在 OpenFlow 协议中, 每个消息都有一个消息头, 用于标识消息的版本、类型、长度和事务 ID。代码中实现了 `build_ofpmsg_header` 函数, 用于构建 OpenFlow 消息头。该函数接收消息长度、消息类型和事务 ID 作为参数, 并填充消息头的各个字段。通过调用该函数, 可以确保消息能够被正确解析和处理。

2.4 消息处理函数

代码中实现了多个 OpenFlow 消息处理函数，每个函数对应一种 OpenFlow 消息类型。以下是每个函数的详细分析。

2.4.1 处理 Hello 消息

`handle_ofpmsg_hello` 函数用于处理控制器发送的 Hello 消息。Hello 消息是 OpenFlow 协议中的基础消息，用于交换控制器和交换机之间的协议版本信息。函数首先检查消息头中的 `version` 字段，判断协议版本是否为 1.3。如果版本匹配，则打印接收到的 Hello 消息；否则，生成一个错误消息并发送给控制器。函数返回 `HANDLE` 表示消息已处理。

```
static enum ofperr handle_ofpmsg_hello(struct ofp_buffer *ofpbuf) {
    if (ofpbuf->header.version == 0x04) {
        printf("RCV HELLO!\n\n");
    } else {
        struct ofp_buffer *ofpbuf_reply = (struct ofp_buffer
*)build_reply_ofpbuf(OFP_ERROR, ofpbuf->header.xid, sizeof(struct ofp_header));
        send_openflow_message(ofpbuf_reply, sizeof(struct ofp_header));
    }
    return HANDLE;
}
```

2.4.2 处理 Features Request 消息

`handle_ofpmsg_features_request` 函数用于处理控制器发送的 Features Request 消息。该消息用于查询交换机的特性信息，如支持的流表数量、缓冲区大小等。函数生成一个 Features Reply 消息，并填充交换机的特性信息，如数据路径 ID、缓冲区数量、流表数量等。通过 `send_openflow_message` 函数将回复消息发送给控制器，并返回 `HANDLE` 表示消息已处理。

```
static enum ofperr handle_ofpmsg_features_request(struct ofp_buffer *ofpbuf) {
    int feature_reply_len = sizeof(struct ofp_switch_features) + sizeof(struct ofp_header);
    struct ofp_buffer *ofpbuf_reply = (struct ofp_buffer
*)build_ofpmsg_reply_ofpbuf(OFP_FEATURES_REPLY, ofpbuf->header.xid, feature_reply_len);
    struct ofp_switch_features *feature_reply_msg = (struct ofp_switch_features
*)ofpbuf_reply->data;

    feature_reply_msg->datapath_id = 0x0100000000000000;
    feature_reply_msg->n_buffers = htonl(46);
    feature_reply_msg->n_tables = 3;
    feature_reply_msg->capabilities = 0x7;

    send_openflow_message(ofpbuf_reply, feature_reply_len);
    return HANDLE;
}
```

2.4.3 处理 Get Config Request 消息

`handle_ofpmsg_get_config_request` 函数用于处理控制器发送的 Get Config Request 消息。该消息用于查询交换机的配置信息，如 Miss Send Length 等。函数生成一个 Get Config Reply 消息，并填充交换机的配置信息。通过 `send_openflow_message` 函数将回复消息发送给控制器，并返回 `HANDLE` 表示消息已处理。

```
static enum ofperr handle_ofpmsg_get_config_request(struct ofp_buffer *ofpbuf) {
    int reply_len = sizeof(struct ofp_switch_config) + sizeof(struct ofp_header);
    struct ofp_buffer *ofpbuf_reply = (struct ofp_buffer
*)build_reply_ofpbuf(OFP_GET_CONFIG_REPLY, ofpbuf->header.xid, reply_len);
    struct ofp_switch_config *switch_config_reply = (struct ofp_switch_config
*)ofpbuf_reply->data;

    switch_config_reply->flags = htons(0x0000);
    switch_config_reply->miss_send_len = htons(32);

    send_openflow_message(ofpbuf_reply, reply_len);
    return HANDLE;
}
```

2.4.4 处理 Description Request 消息

`handle_ofpmsg_desc` 函数用于处理控制器发送的 Description Request 消息。该消息用于查询交换机的描述信息，如制造商、硬件版本等。函数生成一个 Multipart Reply 消息，并填充交换机的描述信息。通过 `send_openflow_message` 函数将回复消息发送给控制器，并返回 `HANDLE` 表示消息已处理。

```
static enum ofperr handle_ofpmsg_desc(struct ofp_buffer *ofpbuf) {
    int reply_len = sizeof(struct ofp_header) + sizeof(struct ofp_multipart) + sizeof(struct
ofp_desc_stats);
    struct ofp_buffer *ofpbuf_reply = (struct ofp_buffer
*)build_reply_ofpbuf(OFP_MULTIPART_REPLY, ofpbuf->header.xid, reply_len);
    struct ofp_multipart *ofpmp_reply = (struct ofp_multipart *)ofpbuf_reply->data;

    static const char *default_mfr_desc = "Wanglixuan";
    static const char *default_hw_desc = "Lixuan_OpenBox";
    static const char *default_sw_desc = "Lixuan_Driver";
    static const char *default_serial_desc = "Lixuan_OpenBox Series";
    static const char *default_dp_desc = "None";

    ofpmp_reply->type = htons(OFPMP_DESC);
    ofpmp_reply->flags = htonl(OFPMP_REPLY_MORE_NO);
    snprintf(ofpmp_reply->ofpmp_desc[0].mfr_desc, sizeof ofpmp_reply-
>ofpmp_desc[0].mfr_desc, "%s", default_mfr_desc);
    snprintf(ofpmp_reply->ofpmp_desc[0].hw_desc, sizeof ofpmp_reply-
>ofpmp_desc[0].hw_desc, "%s", default_hw_desc);
    snprintf(ofpmp_reply->ofpmp_desc[0].sw_desc, sizeof ofpmp_reply-
>ofpmp_desc[0].sw_desc, "%s", default_sw_desc);
    snprintf(ofpmp_reply->ofpmp_desc[0].serial_num, sizeof ofpmp_reply-
>ofpmp_desc[0].serial_num, "%s", default_serial_desc);
    snprintf(ofpmp_reply->ofpmp_desc[0].dp_desc, sizeof ofpmp_reply-
>ofpmp_desc[0].dp_desc, "%s", default_dp_desc);

    send_openflow_message(ofpbuf_reply, reply_len);
    return HANDLE;
}
```

2.4.5 处理 Flow Stats Request 消息

`handle_ofpmsg_flow_stats` 函数用于处理控制器发送的 Flow Stats Request 消息。该消息用于查询交换机的流表统计信息。函数生成一个 Multipart Reply 消息，并设置消息类型为 Flow Stats。通过 `send_openflow_message` 函数将回复消息发送给控制器，并返回 `HANDLE` 表示消息已处理。

```
static enum ofperr handle_ofpmsg_flow_stats(struct ofp_buffer *ofpbuf) {
    int reply_len = sizeof(struct ofp_header) + sizeof(struct ofp_multipart);
    struct ofp_buffer *reply_buffer = (struct ofp_buffer *)
    build_reply_ofpbuf(OFPT_MULTIPART_REPLY, ofpbuf->header.xid, reply_len);
    struct ofp_multipart *multipart_reply = (struct ofp_multipart *)reply_buffer->data;

    multipart_reply->type = htons(OFPMPL_FLOW);
    multipart_reply->flags = htonl(OFPMPL_REPLY_MORE_NO);

    send_openflow_message(reply_buffer, reply_len);
    return HANDLE;
}
```

2.4.6 处理 Aggregate Stats Request 消息

`handle_ofpmsg_aggregate` 函数用于处理控制器发送的 Aggregate Stats Request 消息。该消息用于查询交换机的聚合统计信息。函数生成一个 Multipart Reply 消息，并填充聚合统计信息，如数据包计数、字节计数等。通过 `send_openflow_message` 函数将回复消息发送给控制器，并返回 `HANDLE` 表示消息已处理。

```
static enum ofperr handle_ofpmsg_aggregate(struct ofp_buffer *ofpbuf) {
    int reply_len = sizeof(struct ofp_header) + sizeof(struct ofp_multipart) + sizeof(struct
    ofp_aggregate_stats_reply);
    struct ofp_buffer *ofpbuf_reply = (struct ofp_buffer *)
    build_reply_ofpbuf(OFPT_MULTIPART_REPLY, ofpbuf->header.xid, reply_len);
    struct ofp_multipart *ofpmp_reply = (struct ofp_multipart *)ofpbuf_reply->data;

    ofpmp_reply->type = htons(OFPMPL_AGGREGATE);
    ofpmp_reply->flags = htonl(OFPMPL_REPLY_MORE_NO);

    ofpmp_reply->ofpmp_aggregate_reply[0].packet_count = htonl(46);
    ofpmp_reply->ofpmp_aggregate_reply[0].byte_count = htonl(2025);
    ofpmp_reply->ofpmp_aggregate_reply[0].flow_count = htonl(200);

    send_openflow_message(ofpbuf_reply, reply_len);
    return HANDLE;
}
```

2.4.7 处理 Table Stats Request 消息

`handle_ofpmsg_table` 函数用于处理控制器发送的 Table Stats Request 消息。该消息用于查询交换机的流表统计信息。函数生成一个 Multipart Reply 消息，并填充流表统计信息，如匹配计数、查找计数等。通过 `send_openflow_message` 函数将回复消息发送给控制器，并返回 `HANDLE` 表示消息已处理。

```
static enum ofperr handle_ofpmsg_table(struct ofp_buffer *ofpbuf) {
    int reply_len = sizeof(struct ofp_header) + sizeof(struct ofp_multipart) + sizeof(struct
    ofp_table_stats) * 1;
```

```

        struct ofp_buffer *ofpbuf_reply = (struct ofp_buffer
*)build_reply_ofpbuf(OFPPT_MULTIPART_REPLY, ofpbuf->header.xid, reply_len);
        struct ofp_multipart *ofpmp_reply = (struct ofp_multipart *)ofpbuf_reply->data;

        ofpmp_reply->type = htons(OFPMP_TABLE);
        ofpmp_reply->flags = htonl(OFPMP_REPLY_MORE_NO);

        ofpmp_reply->table_stats[0].matched_count = htonl(2025);
        ofpmp_reply->table_stats[0].table_id = 0;
        ofpmp_reply->table_stats[0].lookup_count = htonl(46);
        ofpmp_reply->table_stats[0].active_count = htonl(1);

        send_openflow_message(ofpbuf_reply, reply_len);
        return HANDLE;
}

```

2.4.8 处理 Port Stats Request 消息

`handle_ofpmsg_port_stats` 函数用于处理控制器发送的 Port Stats Request 消息。该消息用于查询交换机的端口统计信息。函数生成一个 Multipart Reply 消息，并填充端口统计信息，如持续时间、接收数据包数等。通过 `send_openflow_message` 函数将回复消息发送给控制器，并返回 `HANDLE` 表示消息已处理。

```

static enum ofperr handle_ofpmsg_port_stats(struct ofp_buffer *ofpbuf) {
    int reply_len = sizeof(struct ofp_header) + sizeof(struct ofp_multipart) + sizeof(struct
ofp_port_stats) * nmps.cnt;
    struct ofp_buffer *ofpbuf_reply = (struct ofp_buffer
*)build_reply_ofpbuf(OFPPT_MULTIPART_REPLY, ofpbuf->header.xid, reply_len);
    struct ofp_multipart *ofpmp_reply = (struct ofp_multipart *)ofpbuf_reply->data;

    ofpmp_reply->type = htons(OFPMP_PORT_STATS);
    ofpmp_reply->flags = htonl(OFPMP_REPLY_MORE_NO);

    for (int i = 0; i < nmps.cnt; i++) {
        ofpmp_reply->ofpmp_port_stats[i] = nmps.ports[i].stats;
        ofpmp_reply->ofpmp_port_stats[i].duration_sec = htonl(2025);
        ofpmp_reply->ofpmp_port_stats[i].duration_nsec = htonl(51);
    }

    send_openflow_message(ofpbuf_reply, reply_len);
    return HANDLE;
}

```

2.4.9 处理 Group Features Request 消息

`handle_ofpmsg_group_features` 函数用于处理控制器发送的 Group Features Request 消息。该消息用于查询交换机的组表特性信息。函数生成一个 Multipart Reply 消息，并填充组表特性信息，如最大组数等。通过 `send_openflow_message` 函数将回复消息发送给控制器，并返回 `HANDLE` 表示消息已处理。

```

static enum ofperr handle_ofpmsg_group_features(struct ofp_buffer *ofpbuf) {
    int reply_len = sizeof(struct ofp_header) + sizeof(struct ofp_group_features) + 8;
    struct ofp_buffer *ofpbuf_reply = (struct ofp_buffer

```

```

*)build_reply_ofpbuf(OFPT_MULTIPART_REPLY, ofpbuf->header.xid, reply_len);
    struct ofp_group_features *group = (struct ofp_group_features *)ofpbuf_reply->data;

    group->types = htons(OFPPMP_GROUP_FEATURES);
    group->max_groups[0] = htonl(0xdeadbeef);

    send_openflow_message(ofpbuf_reply, reply_len);
    return HANDLE;
}

```

2.4.10 处理 Port Description Request 消息

`handle_ofpmsg_port_desc` 函数用于处理控制器发送的 Port Description Request 消息。该消息用于查询交换机的端口描述信息。函数生成一个 Multipart Reply 消息，并填充端口描述信息，如端口状态等。通过 `send_openflow_message` 函数将回复消息发送给控制器，并返回 `HANDLE` 表示消息已处理。

```

static enum ofperr handle_ofpmsg_port_desc(struct ofp_buffer *ofpbuf) {
    int reply_len = sizeof(struct ofp_header) + sizeof(struct ofp_multipart) + sizeof(struct
ofp_port) * nmps.cnt;
    struct ofp_buffer *ofpbuf_reply = (struct ofp_buffer
*)build_reply_ofpbuf(OFPT_MULTIPART_REPLY, ofpbuf->header.xid, reply_len);
    struct ofp_multipart *ofpmp_reply = (struct ofp_multipart *)ofpbuf_reply->data;

    ofpmp_reply->type = htons(OFPPMP_PORT_DESC);
    ofpmp_reply->flags = htonl(OFPPMP_REPLY_MORE_NO);

    for (int i = 0; i < nmps.cnt; i++) {
        ofpmp_reply->ofpmp_port_desc[i] = nmps.ports[i].state;
    }

    send_openflow_message(ofpbuf_reply, reply_len);
    return HANDLE;
}

```

2.4.11 处理 Packet Out 消息

`handle_ofpmsg_packet_out` 函数用于处理控制器发送的 Packet Out 消息。该消息用于指示交换机将数据包从指定端口发送出去。函数解析消息中的动作列表，判断动作类型是否为 `OFPAT_OUTPUT`，并根据动作指示的端口号调用 `nms_exec_action` 函数将数据包发送出去。函数返回 `HANDLE` 表示消息已处理。

```

static enum ofperr handle_ofpmsg_packet_out(struct ofp_buffer *ofpbuf) {
    struct ofp_packet_out *out = (struct ofp_packet_out *)ofpbuf;
    struct ofp_action_output *action = (struct ofp_action_output *)&out->actions[0];
    int action_len = ntohs(out->actions_len);
    struct eth_header *eth = (struct eth_header *)&ofpbuf->data[sizeof(struct
ofp_packet_out) - sizeof(struct ofp_header) + action_len];
    int send_len = ntohs(ofpbuf->header.length) - sizeof(struct ofp_packet_out) -
action_len;

    if (action_len == 0) {
        nms_exec_action(ntohl(out->in_port), OFPP_FLOOD, eth, send_len, -1);
    }
}

```

```

    } else {
        while (action_len > 0) {
            if (action->type == OFPAT_OUTPUT) {
                nms_exec_action(ntohl(out->in_port), ntohl(action->port), eth, send_len, -1);
            }
            action_len -= sizeof(struct ofp_action_output);
            action++;
        }
    }
    return HANDLE;
}

```

2.4.12 处理 Role Request 消息

`handle_opfmsg_role_request` 函数用于处理控制器发送的 Role Request 消息。该消息用于配置交换机的角色（如主控制器、从控制器等）。函数生成一个 Role Reply 消息，并填充角色配置信息。通过 `send_openflow_message` 函数将回复消息发送给控制器，并返回 `HANDLE` 表示消息已处理。

```

static enum ofperr handle_opfmsg_role_request(struct ofp_buffer *ofpbuf) {
    int reply_len = sizeof(struct ofp_header) + sizeof(struct ofp_role);
    struct ofp_buffer *ofpbuf_reply = (struct ofp_buffer
*)build_reply_ofpbuf(OFPT_ROLE_REPLY, ofpbuf->header.xid, reply_len);
    memcpy(ofpbuf_reply->data, ofpbuf->data, sizeof(struct ofp_role));
    ofpbuf_reply->header.type = OFPT_ROLE_REPLY;

    send_openflow_message(ofpbuf_reply, reply_len);
    return HANDLE;
}

```

2.5 实验结果

在控制器与交换机之间进行抓包，分析报文的交互过程。控制器发送不同类型的 OpenFlow 消息给交换机，交换机接收并处理消息，并返回相应的回复消息。通过抓包分析，可以看到消息的交互过程，包括消息头、消息类型、消息长度等字段的解析和处理。用这种方式来验证协议编写的正确性。

实现的子协议种类较多，这里只展示部分报文内容。

2.5.1 OFPT_ROLE_REPLY

1. OFPT_ROLE_REPLY 类型消息。此消息用于配置交换机的角色（如主控制器、从控制器等）。下面的 Figure 1 是控制器发送的 Role Request 消息。

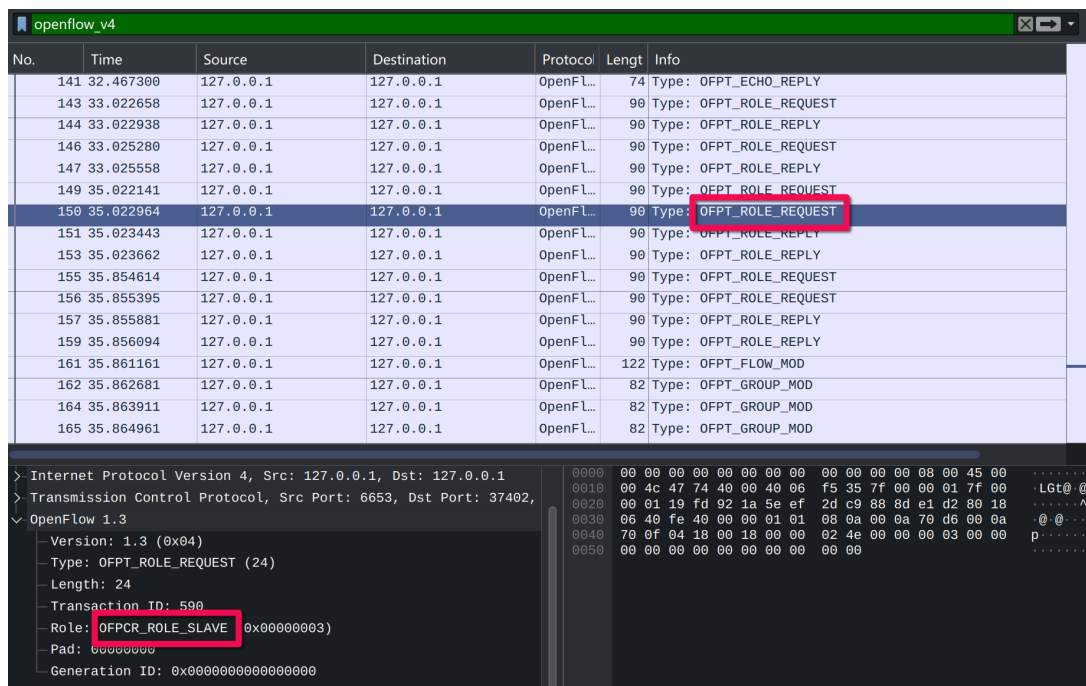


Figure 1: OFPT_ROLE_REQUEST 消息

如图 Figure 2 所示，交换机接收并处理后返回的 Role Reply 消息。

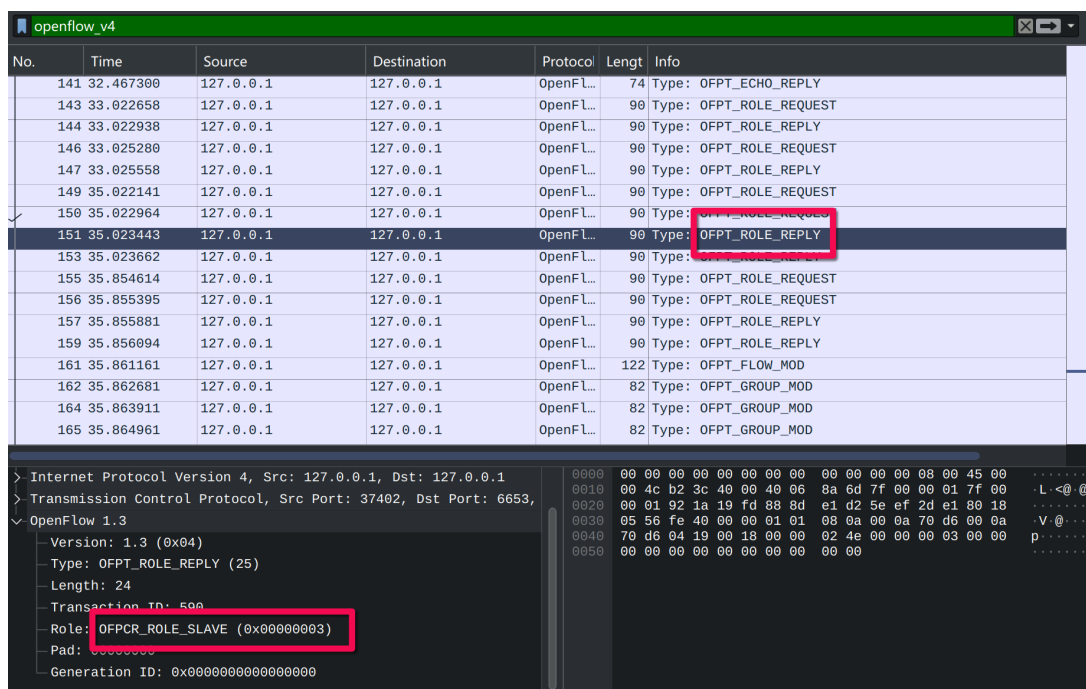


Figure 2: OFPT_ROLE_REPLY 消息

这说明交换机已经接收到了控制器发送的 Role Request 消息，并返回了 Role Reply 消息。

2.5.2 OFPT_MULTIPART_REPLY

1. OFPMP_DESC 类型消息。此消息用于查询交换机的描述信息，包括制造商、硬件版本、软件版本、序列号和数据路径描述等。

No.	Time	Source	Destination	Protocol	Length	Info
4	0.001454	127.0.0.1	127.0.0.1	OpenFL...	74	Type: OFPT_HELLO
8	0.019830	127.0.0.1	127.0.0.1	OpenFL...	74	Type: OFPT_FEATURES_REQUEST
10	0.020242	127.0.0.1	127.0.0.1	OpenFL...	98	Type: OFPT_FEATURES_REPLY
12	0.064691	127.0.0.1	127.0.0.1	OpenFL...	82	Type: OFPT_MULTIPART_REQUEST, OFPMP_PORT_DESC
13	0.064969	127.0.0.1	127.0.0.1	OpenFL...	402	Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_DESC
15	0.068975	127.0.0.1	127.0.0.1	OpenFL...	94	Type: OFPT_GET_CONFIG_REQUEST
16	0.069149	127.0.0.1	127.0.0.1	OpenFL...	74	Type: OFPT_BARRIER_REPLY
18	0.113923	127.0.0.1	127.0.0.1	OpenFL...	78	Type: OFPT_GET_CONFIG_REPLY
20	0.116484	127.0.0.1	127.0.0.1	OpenFL...	82	Type: OFPT_MULTIPART_REQUEST, OFPMP_DESC
21	0.116767	127.0.0.1	127.0.0.1	OpenFL...	1138	Type: OFPT_MULTIPART_REPLY, OFPMP_DESC
22	0.125543	127.0.0.1	127.0.0.1	OpenFL...	82	Type: OFPT_MULTIPART_REQUEST, OFPMP_TABLE_FEATURES
23	0.125851	127.0.0.1	127.0.0.1	OpenFL...	94	Type: OFPT_ERROR
24	0.128167	127.0.0.1	127.0.0.1	OpenFL...	90	Type: OFPT_ROLE_REQUEST
25	0.128412	127.0.0.1	127.0.0.1	OpenFL...	90	Type: OFPT_ROLE_REPLY
26	0.131819	127.0.0.1	127.0.0.1	OpenFL...	122	Type: OFPT_FLOW_MOD
27	0.132692	127.0.0.1	127.0.0.1	OpenFL...	82	Type: OFPT_GROUP_MOD
28	0.133480	127.0.0.1	127.0.0.1	OpenFL...	82	Type: OFPT_GROUP_MOD
29	0.134226	127.0.0.1	127.0.0.1	OpenFL...	82	Type: OFPT_GROUP_MOD

Flags: 0x0000
 Pad: 00000000
 Manufacturer desc.: Wanglixuan
 Hardware desc.: Lixuan_OpenBox
 Software desc.: Lixuan_Driver
 Serial no.: Lixuan_OpenBox Series
 Datapath desc.: None

Figure 3: OFPMP_DESC 消息

可以看到图中的消息包含了制造商、硬件版本、软件版本、序列号和数据路径描述等信息（此处使用了自己的信息进行标记与区分）。

2. OFPMP_TABLE 类型消息。此消息用于查询交换机的流表统计信息，包括匹配计数、查找计数等。

No.	Time	Source	Destination	Protocol	Length	Info
107	16.251793	127.0.0.1	127.0.0.1	OpenFL...	122	Type: OFPT_MULTIPART_REPLY, OFPMP_GROUP_FEATURES
108	16.255617	127.0.0.1	127.0.0.1	OpenFL...	82	Type: OFPT_MULTIPART_REQUEST, OFPMP_DESC
109	16.255900	127.0.0.1	127.0.0.1	OpenFL...	1138	Type: OFPT_MULTIPART_REPLY, OFPMP_DESC
110	16.301001	127.0.0.1	127.0.0.1	OpenFL...	90	Type: OFPT_MULTIPART_REQUEST, OFPMP_PORT_STATS
111	16.301736	127.0.0.1	127.0.0.1	OpenFL...	642	Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_STATS
112	16.311260	127.0.0.1	127.0.0.1	OpenFL...	122	Type: OFPT_MULTIPART_REQUEST, OFPMP_AGGREGATE
113	16.311552	127.0.0.1	127.0.0.1	OpenFL...	106	Type: OFPT_MULTIPART_REPLY, OFPMP_AGGREGATE
114	16.315617	127.0.0.1	127.0.0.1	OpenFL...	122	Type: OFPT_MULTIPART_REQUEST, OFPMP_FLOW
115	16.315936	127.0.0.1	127.0.0.1	OpenFL...	82	Type: OFPT_MULTIPART_REPLY, OFPMP_FLOW
116	16.322070	127.0.0.1	127.0.0.1	OpenFL...	82	Type: OFPT_MULTIPART_REQUEST, OFPMP_TABLE
117	16.322325	127.0.0.1	127.0.0.1	OpenFL...	106	Type: OFPT_MULTIPART_REPLY, OFPMP_TABLE
119	18.324755	127.0.0.1	127.0.0.1	OpenFL...	74	Type: OFPT_ECHO_REQUEST
120	18.324946	127.0.0.1	127.0.0.1	OpenFL...	74	Type: OFPT_ECHO_REPLY
122	20.326846	127.0.0.1	127.0.0.1	OpenFL...	74	Type: OFPT_ECHO_REQUEST
123	20.327006	127.0.0.1	127.0.0.1	OpenFL...	74	Type: OFPT_ECHO_REPLY
125	22.328898	127.0.0.1	127.0.0.1	OpenFL...	74	Type: OFPT_ECHO_REQUEST
126	22.329057	127.0.0.1	127.0.0.1	OpenFL...	74	Type: OFPT_ECHO_REPLY
128	24.329406	127.0.0.1	127.0.0.1	OpenFL...	74	Type: OFPT_ECHO_REQUEST

Pad: 00000000
 Table stats
 Table ID: 0
 Pad: 00000000
 Lookup count: 46
 Match count: 2025

Figure 4: OFPMP_TABLE 消息

可以看到图中的消息包含了匹配计数、查找计数等信息（此处修改为了自己设置的值）。

3 实验总结

本次实验通过编写 SDN 交换机源码处理 OpenFlow 协议消息，深入理解了 OpenFlow 协议的基本消息格式和处理流程。实验涵盖了从交换机特性查询到流表统计、端口统计等多种消息类型的处理，成功实现了大部分 OpenFlow 协议的核心功能。通过实验，我们不仅掌握了 OpenFlow 协议的基本工作原理，还熟悉了如何通过代码实现协议消息的解析与响应。

在实验过程中，我们首先熟悉了 Openbox-S4 的基本功能，并通过编写代码实现了对 OpenFlow 协议消息的处理。实验代码中，我们实现了包括 Hello 消息、Features Request 消息、Flow Stats Request 消息等在内的多种消息处理函数。每个函数都根据消息类型生成相应的回复消息，并通过 `send_openflow_message` 函数发送给控制器。通过抓包分析，我们验证了消息的交互过程，确保协议编写的正确性。

实验中的难点在于如何正确处理不同类型的 OpenFlow 消息，并确保消息的格式和内容符合协议规范。通过查阅 OpenFlow 协议规范和参考相关文献，我们逐步解决了这些问题，并成功实现了协议的核心功能。此外，实验还要求我们熟悉 REST API 接口的使用，这为后续的网络功能扩展奠定了基础。

实验结果表明，我们编写的代码能够正确处理控制器发送的 OpenFlow 消息，并返回相应的回复消息。通过抓包分析，我们验证了消息的交互过程，确保协议编写的正确性。实验的成功不仅加深了我们对 OpenFlow 协议的理解，也提升了我们的编程能力和网络协议分析能力。

总的来说，本次实验达到了预期的目标，成功实现了 OpenFlow 协议的核心功能。通过实验，我们不仅掌握了 OpenFlow 协议的基本工作原理，还熟悉了如何通过代码实现协议消息的解析与响应。未来，我们可以在此基础上进一步扩展功能，如实现流表规则的动态添加和删除，以更好地支持 SDN 网络的灵活性和可扩展性。

参考文献

- [1] OPEN NETWORKING FOUNDATION. OpenFlow Switch Specification Version 1.3.0[EB/OL](2014-10). <https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf>
- [2] GOLDSUNSHINE. OpenFlow 协议详解[EB/OL](2017-07-28). <https://www.cnblogs.com/goldsunshine/p/7262484.html>
- [3] 王小二. OpenFlow 协议简介[EB/OL](2024-12-25). <https://www.jianshu.com/p/acfeae1771b3>
- [4] CODEJUNERY2022. 深入理解 OpenFlow 协议[EB/OL](2024-12-25). <https://www.jianshu.com/p/82e238eb8d14>