# 四、系统具体设计与实现

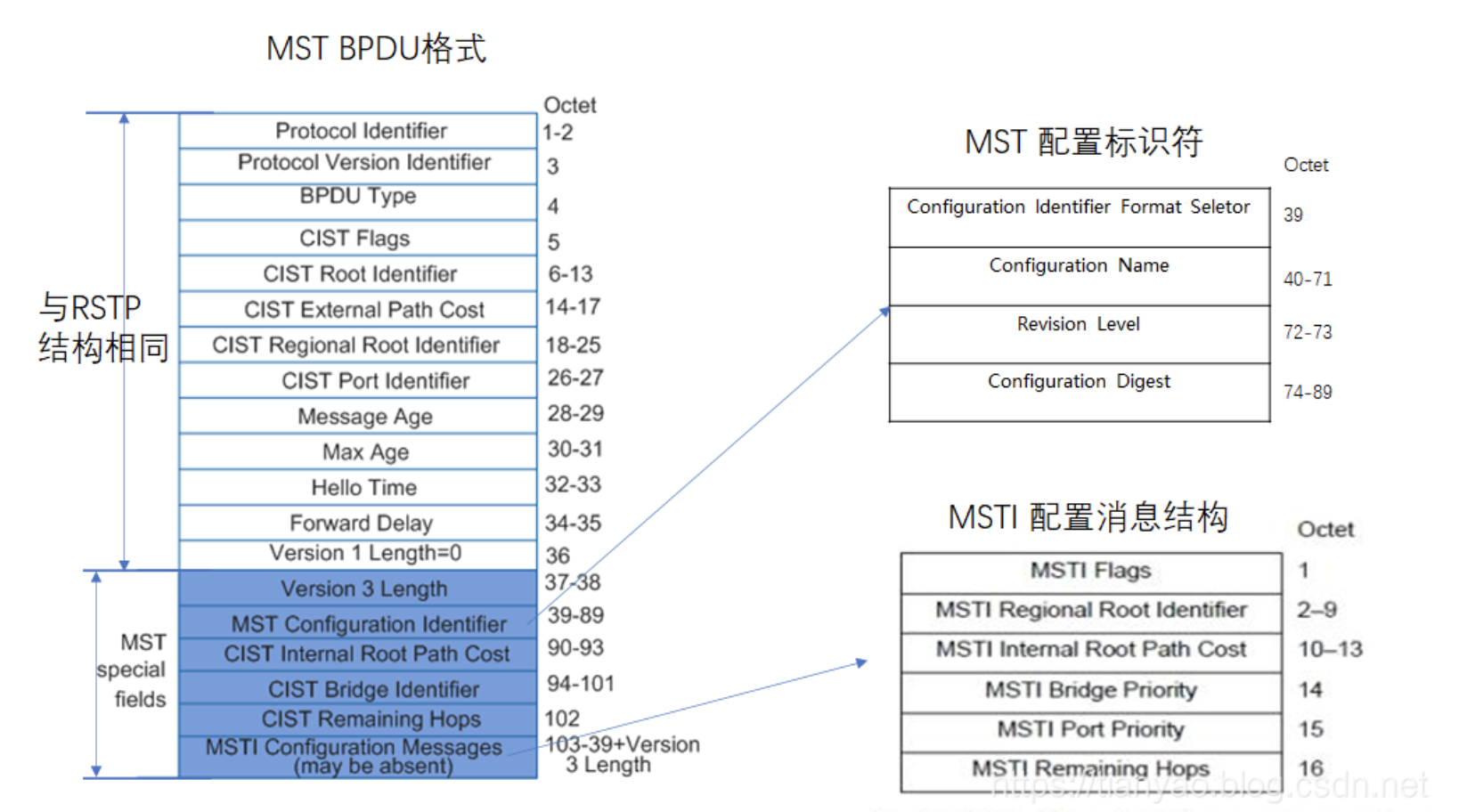
## 4.1概述

图1-1中我们给出了DLH的组成概览，下面将对于每个模块的设计和实现以及模块间如何配合完成系统原型搭建展开详细阐述。

## 4.2配置收集模块

DLH使用使用了多种方式完成计算机网络底层拓扑结构的收集。系统采用了网络结构库networkx开源代码中提供的拓扑生成源码作为拓扑结构实现的基础。networkx是学界中便于用户对复杂网络进行创建、操作和学习的通用工具。利用networkx可以以标准化和非标准化的数据格式存储网络、生成多种随机网络和经典网络、分析网络结构、建立网络模型、设计新的网络算法、进行网络绘制。在此的基础上完成后续的定位产生问题的网络配置。同时为了方便操作我们还提供了一定的范示，帮组用户直接读取网络结构。

配置收集的主要任务，在网络控制平面中解析读取对应的配置文件（如配置文件、MAC表、 data package）将其转化为对应的抽象模型。模块通过自动的读取关键的配置信息如：MAC地址，优先级，链路数据，VLAN，ERPC，IPRC等核心数据完成真实网络的拓扑结构抽象。我们可以IEEE 802.1s中MSTP的BPDU数据单元看看对应的核心数据结构：

图4-1 MSTP BPDU结构

这里给出了配置收集模块关注信息：CIST IRPC, CIST BID, MSTI IPRC等。因此在配置收集模块我们的主要工作就是将网络中的端口等配置信息抽象出来，同时减少不必要的数据，减少计算规模，根据收集到了信息在抽象模型中完成计算，以便于在后期获取真实运行效果。

有了抽象配置文件之后，DLH便可以在它们之上实现一些功能，比如端到端可达性分析、链路故障，环路验证等。

## 4.3测试包生成模块

测试包生成模块主要负责进行数据抽象，模拟data package 在真实网络拓扑上的流通。在networkx提供的抽象模型建立工具上进行模块数据的输入，完成生成端到端的可达性抽象模型。这里选取源码中的重要片段进行阐述。

生成端到端全对可达表的主要函数如下：

def buildStructure(raw\_data):

data = raw\_data.values

##关注点1

for i in range(data.shape[0]):  
 port\_data = data[i]  
 pid = int(str(port\_data[2]) + str(port\_data[1]))  
 g.add\_node(port\_data[0], PID=pid, Root\_bridge\_ID='0', ERPC=0, \  
 Region\_bridge\_ID=0,IRPC=0,Designated\_BID=0, Designated\_PID=0,

\received\_PID=0, Port\_status='None',\  
 switch\_bid=-1, portID=port\_data[1], Port\_priority=port\_data[2], )  
##关注点2   
for switch in switch\_list:  
 name = switch.switch\_name  
 for port in switch.port:  
 g.nodes[name + '-' + str(port)]['switch\_bid'] = switch.bid  
 g.nodes[name + '-' + str(port)]['Root\_bridge\_ID'] = switch.bid  
 g.nodes[name + '-' + str(port)]['Region\_bridge\_ID'] = switch.bid  
  
 g.add\_node(switch.switch\_name,PID=-1, Root\_bridge\_ID=switch.bid, ERPC=-1,Region\_bridge\_ID=switch.bid,IRPC=-1, Designated\_BID=-1, Designated\_PID=-1, received\_PID=-1,Port\_status='switch', switch\_bid=switch.bid, portID=-1, Port\_priority=-1)

##关注点3

data = raw\_data.values

for i in range(data.shape[0]):

data\_edge = list(data[i])

g.add\_edge(data\_edge[0],data\_edge[1],cost=data\_edge[2],\ VLAN=data\_edge[3], region\_name=data\_edge[4])

首先看到函数的参数raw\_data，为配置收集模块收集到的数据。根据收集到的数据我们设定了switch，port，edge等对象，根据数据完成对应结构数据的模块化输入，对应的代码在mstp.py文件中有具体细节的呈现，它所具有的重要的属性有ID，其在初始化时对应为学习到的对应ID，RPC为路径开销，初始时为0，在仿真计算中根据链路带宽进行路径开销计算。最终生成的端到端的可达链路，这样的链路最终构成了抽象的网络拓扑结构。

## 4.4算法仿真模块

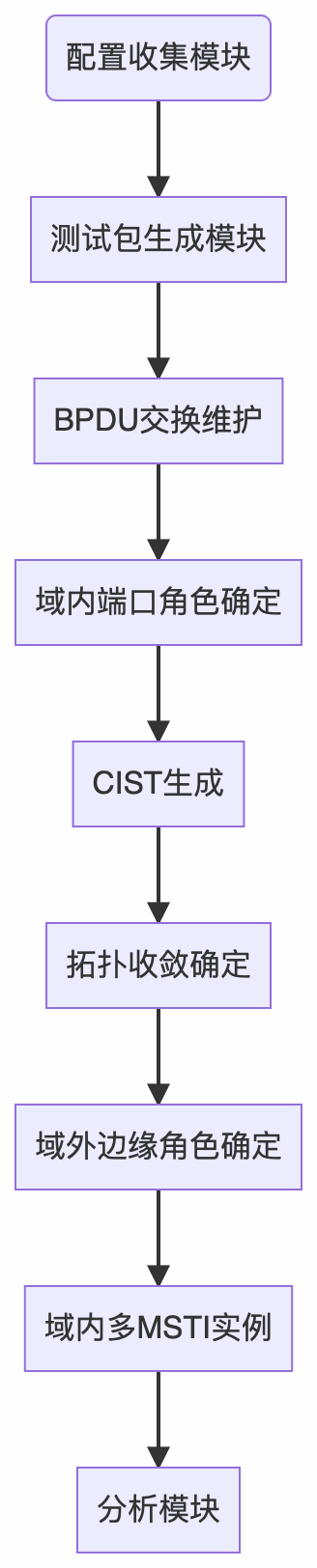
算法仿真模块是用来对测试包模块传递的结果进行完整的模拟分析，核心的算法是自适应我们抽象模块的生成树算法，主要为STP及MSTP算法，这里我们给出我们算法这里给出算法仿真模块BPDU仿真优先级表结构。

表4- 1 BPDU优先级表(优先级从高到低)

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Value Example** |
| header | TEXT | {"ip\_dst\_wc": 0, "vlan\_match": [0], "ip\_dst\_match": ["64.57.20.191"], "vlan\_wc": 0} |
| Root\_bridge\_ID | INTEGER | 000000000000000001000000001110010001010010111111 |
| ERPC | INTEGER | 800000 |
| Region\_bridge\_ID | INTEGER | 000000000000000001000000001110010001010010111111 |
| IRPC | INTEGER | 10000000 |
| Designated\_BID | INTEGER | 000000000000000001000000001110010001010010111111 |
| Designated\_PID | INTEGER | 0000000000000000010000000011100100010100101 |
| Port\_status | TEXT | ‘switch’ |

其中RPC我们根据Dijkstra算法进行计算，各种BID，PID根据BPDU交换包携带信息进行级联更新。我们为每个交换机端口维护一个BPDU队列，通过比较上诉优先级队列完成BPDU队列的更新及丢弃，当数据稳定后及拓扑收敛后我们按照图4-1选取端口见色。

在这个算法中我们采取如下流程图完成了算法仿真模块的过程

图4-2 流程图

其中最为重要的部分为端口角色确定的部分，我们通过BPDU的维护及表4-1优先级规则来确定根端口，指定端口等端口角色。

选举完毕后我们会获得多个表：

表4-2端口角色表（ROOT PORT RULE）

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Value Example** |
| PORT1 | TEXT | ‘ROOT PORT’ |
| PORT2 | TEXT | ‘DESIGNATED PORT’ |
| PORT3 | TEXT | ‘EDGE PORT’ |
| PORT4 | TEXT | ‘MASTER PORT’ |
| PORT5 | TEXT | ‘BACKUP PORT’ |
| PORT6 | TEXT | ‘ALTERNATURE PORT’ |

ROOT PORT RULE：用于端口角色选举完成后的状态表

表4-3 EDGE（reachable\_edge）

|  |  |  |  |
| --- | --- | --- | --- |
| **EDGE\_PORT** | **EDGE\_PORT** | **COST** | **STATUS** |
| PORT1 | PORT2 | 20000 | TRUE |
| PORT1 | PORT3 | 40000 | FALSE |
| PORT2 | PORT5 | 20000 | TRUE |

该表记录的是仿真网络的线路集合。所谓线路集合，指的是端到端的端口节点中测试包所能够到达到的所有网络节点的集合。TRUE代表线路未堵塞，backup代表线路堵塞，在实现上，获得该表的方式是，通过表4-2端口角色的确定来对EDGE进行对应的阻塞，最终形成无环链路。

## 4.5分析模块

这里出于快速原型的考虑，DLH使用Python3.8实现了算法分析选择模块。分析模块的主要功能是，在端到端生成的4-2，4-3中端口角色表及线路角色表生成的无环链路来定位运行时出现的不同故障，同时算法提供两种版本的算法提供无环链路，一是STP算法，二是MSTP算法，可以针对不同版本的网络环境采取不同的定位策略，定位具体故障信息，并选择以后的处理方式，如上报故障管理员，自己测试等。

下面首先给出该模块的工作流程图，用以反映该模块的内部运行逻辑。



图4-3分析模块的工作流程图

通过读取4.4模块的端口角色表，链路角色表，我们可以生成无环链路，根据无环链路及生成树来定位故障

# 五、项目成果展示

## 5.1 基于STP展示

本节展示DLH在针对特定的网路链路针对STP算法进行无环链路生成及测试。按照项目要求，我们将分别展示端口角色表，链路角色表：

**测试网络结构**：

表5-1 Switch对应结构

|  |  |  |
| --- | --- | --- |
| PORT | BID | name |
| 1,2 | 1 | switch1 |
| 3,4,5 | 2 | switch2 |
| 6,7,8 | 3 | switch3 |
| 9,10,11,12,13 | 4 | switch4 |
| 14,15,16,17,18 | 5 | switch5 |
| 19,20 | 6 | switch6 |
| 21,22 | 7 | switch7 |
| 23,24 | 8 | switch8 |

其中表5-1代表网络拓扑结构中交换机的相应结构，其中PORT代表对应的PORT端口号，BID为对应优先级及MAC地址。

表5-2 EDGE对应结构

|  |  |  |
| --- | --- | --- |
| Port1 | Port2 | weight |
| 1 | 3 | 1 |
| 6 | 2 | 1 |
| 5 | 9 | 1 |
| 8 | 14 | 1 |
| 4 | 15 | 2 |
| 7 | 10 | 2 |
| 11 | 16 | 1 |
| 13 | 19 | 1 |
| 18 | 21 | 1 |
| 20 | 22 | 1 |
| 12 | 23 | 2 |
| 17 | 24 | 2 |

表5-2代表的网络链路的对应结构，其中PORT1，PORT2代表对应的节点，WEIGHT代表链路的带宽。

输入上诉结果到DLH中，测试网络结果

**测试网络无环结果：**

表5-3 PORT对应端口角色

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| PORT | STATUS | PORT | STATUS | PORT | STATUS | PORT | STATUS |
| 1 | DP | 7 | DP | 13 | DP | 19 | RP |
| 2 | DP | 8 | DP | 14 | RP | 20 | DP |
| 3 | RP | 9 | RP | 15 | AP | 21 | RP |
| 4 | DP | 10 | AP | 16 | AP | 22 | AP |
| 5 | DP | 11 | DP | 17 | DP | 23 | RP |
| 6 | RP | 12 | DP | 18 | DP | 24 | AP |

可以发现在表5-3，每个端口均有对应的角色，其中出现了AP端口，也就是可以被阻塞的端口，用于最终无环链路的生成。

表5-4 链路对应角色

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| SWITCH1 | SWITCH2 | STATUS | SWITCH1 | SWITCH2 | STATUS |
| Switch1 | Switch2 | FALSE | Switch4 | Switch5 | TRUE |
| Switch1 | Switch3 | FALSE | Switch4 | Switch8 | FALSE |
| Switch2 | Switch5 | TRUE | Switch4 | Switch6 | FALSE |
| Switch2 | Switch4 | FALSE | Switch5 | Switch8 | TRUE |
| Switch3 | Switch4 | TRUE | Switch5 | Switch7 | FALSE |
| Switch3 | Switch5 | FALSE | Switch6 | Switch7 | TRUE |

通过对表5-3的分析最终我们得到，表5-4中对于TRUE的链路的角色图，其中对于TRUE的链路进行阻塞，对于FALSE的链路进行正常维护。

## 5.2基于MSTP展示

本节展示DLH在针对特定的网路链路针对MSTP算法进行无环链路生成及测试。按照项目要求，我们将分别展示端口角色表，链路角色表：

**测试网络结构**：

表5-5 Switch对应结构

|  |  |  |
| --- | --- | --- |
| PORT | BID | name |
| 1,2 | 1 | switch1 |
| 1,2,3,4 | 2 | switch2 |
| 1,2,3 | 3 | switch3 |
| 1,2,3 | 4 | switch4 |
| 1,2 | 5 | switch5 |
| 1,2,3,4 | 6 | switch6 |
| 1,2,3 | 7 | switch7 |
| 1,2,3 | 8 | switch8 |
| 1,2 | 9 | Switch9 |

其中表5-1代表网络拓扑结构中交换机的相应结构，其中PORT代表对应的PORT端口号，BID为对应优先级及MAC地址。

表5-6 EDGE对应结构

|  |  |  |
| --- | --- | --- |
| Port1 | Port2 | weight |
| LSW1-1 | LSW2-1 | 20000 |
| LSW1-2 | LSW3-2 | 20000 |
| LSW2-2 | LSW3-1 | 20000 |
| LSW2-3 | LSW4-3 | 20000 |
| LSW4-1 | LSW5-1 | 20000 |
| LSW5-2 | LSW6-1 | 20000 |
| LSW4-2 | LSW6-2 | 20000 |
| LSW6-3 | LSW8-3 | 20000 |
| LSW8-1 | LSW7-1 | 20000 |
| LSW8-2 | LSW9-2 | 20000 |
| LSW9-1 | LSW7-2 | 20000 |
| LSW7-3 | LSW3-3 | 20000 |

表5-2代表的网络链路的对应结构，其中PORT1，PORT2代表对应的节点，WEIGHT代表链路的带宽。

输入上诉结果到DLH中，测试网络结果

**测试网络无环结果：**

表5-7 PORT对应端口角色

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| PORT | STATUS | PORT | STATUS | PORT | STATUS | PORT | STATUS |
| LSW1-1 | DP | LSW3-2 | RP | LSW5-2 | DP | LSW7-3 | edge\_port |
| LSW1-2 | DP | LSW3-3 | edge\_port | LSW6-1 | AP | LSW8-1 | RP |
| LSW2-1 | RP | LSW4-1 | DP | LSW6-2 | RP | LSW8-2 | DP |
| LSW2-2 | DP | LSW4-2 | DP | LSW6-3 | edge\_port | LSW8-3 | edge\_port |
| LSW2-3 | edge\_port | LSW4-3 | edge\_port | LSW7-1 | DP | LSW9-1 | RP |
| LSW3-1 | AP | LSW5-1 | RP | LSW7-2 | DP | LSW9-2 | AP |
| LSW2-4 | backup | LSW6-4 | backup |  |  |  |  |

可以发现在表5-3，每个端口均有对应的角色，其中出现了AP端口，也就是可以被阻塞的端口，用于最终无环链路的生成。

表5-8 链路对应角色

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| SWITCH1 | SWITCH2 | STATUS | SWITCH1 | SWITCH2 | STATUS |
| Switch1 | Switch2 | FALSE | Switch7 | Switch8 | FALSE |
| Switch1 | Switch3 | FALSE | Switch7 | Switch9 | FALSE |
| Switch2 | Switch3 | TRUE | Switch8 | Switch9 | TRUE |
| Switch4 | Switch5 | FALSE | Switch2 | Switch4 | FALSE |
| Switch6 | Switch4 | FALSE | Switch3 | Switch7 | FALSE |
| Switch6 | Switch5 | TRUE | Switch6 | Switch8 | TRUE |

通过对表5-3的分析最终我们得到，表5-4中对于TRUE的链路的角色图，其中对于TRUE的链路进行阻塞，对于FALSE的链路进行正常维护。