**端到端故障智能诊断系统设计文档**

---------DPEagle设计文档

**西安交通大学**

**二零二零年十月**

目录

[一、系统概述 1](#_Toc52227450)

[二、端到端全对可达表生成方法 3](#_Toc52227451)

[2.1关于网络数据平面验证工具HSA的简述 4](#_Toc52227452)

[2.2利用HSA生成端到端全对可达表 6](#_Toc52227453)

[三、定位测试包选择算法 8](#_Toc52227454)

[3.1网络规则的1-识别性 8](#_Toc52227455)

[3.2选择算法的基础版本 9](#_Toc52227456)

[3.3选择算法的加速版本 11](#_Toc52227457)

[3.4可用测试包的生成 11](#_Toc52227458)

[3.5 1-识别下的故障定位方法 12](#_Toc52227459)

[四、系统具体设计与实现 13](#_Toc52227460)

[4.1概述 13](#_Toc52227461)

[4.2网络验证模块 13](#_Toc52227462)

[4.3测试包生成模块 15](#_Toc52227463)

[4.4测试包数据库 17](#_Toc52227464)

[4.5定位测试包选择模块 20](#_Toc52227465)

[4.6诊断分析模块 21](#_Toc52227466)

[五、项目成果展示 24](#_Toc52227467)

[5.1 系统性能表现 24](#_Toc52227468)

# 一、系统概述

DPEagle是一种新型端到端故障智能诊断系统。在主流的网络验证工具和一种定位测试包选择算法的基础上，DPEagle能够端到端式地发送特定的测试包对网络转发设备（交换机、路由器等）中的规则（转发规则、ACL等）进行测试。通过分析测试包的收发情况，DPEagle能够发现并定位到产生问题的网络规则。



图1-1 系统概览

上图给出了DPEagle的系统组成和运行概览。它由四个模块组成，分别是网络验证模块、测试包生成模块、定位测试包选择模块以及诊断分析模块。通过四个模块的有效配合，DPEagle能够对网络转发设备中的转发规则进行动态测试来发现网络设备的故障。在发现问题后，DPEagle能够进一步地对问题进行定位。动态测试能够发现一些网络数据平面特有的故障，如硬件失效、链路故障（如光纤断裂）、软件漏洞（如交换机转发功能实现上的漏洞）等。这些故障本质上可以归因到某些有问题的转发规则或链路（链路可以被视作一种转发规则）上，它们无法被测试包正确探测。下面分别简述各个模块所具有的功能：

1. **网络验证模块。**该模块建立在网络数据平面的验证工具之上。网络数据平面验证工具的主要功能是对真实网络中的拓扑、转发设备以及转发规则进行抽象建模，并在所建立的抽象模型之上，对网络转发中的一些不变量（如可达性、环路、黑洞）开展运行前检验，使得一些常见的错误（如配置错误、安全隔离错误）能够在有误的转发行为发生之前被检测到，大大减少了错误排查的周期和成本。抽象模型在本质上是一种关于网络数据平面的映射，在抽象模型上进行的运算，将能够反映网络数据平面在真实运行时会出现的一些状况。由于映射所具有的一些等价性，验证工具在抽象模型上进行的端到端可达性分析将为后面的测试包生成打下基础。下面概述本模块的主要行为。在图1-1中我们可以看到，网络验证模块会首先获取到网络数据平面的一些关键信息，如拓扑配置、FIBs、ACLs等配置文件。在获取了这些文件后，验证模块将解析其中的信息并生成相应的抽象模型。在抽象模型之上，模块还将定义并实现一些特定的功能，如端到端可达性分析、环路检测、黑洞排查等。
2. **测试包生成模块。**该模块的主要任务是利用网络验证模块提供的端到端可达性分析功能完成边缘测试端口间的可达测试包包头空间的生成工作。可达分析是模拟的数据平面的转发行为和过程。具体地，给定一对用于收发测试包的边缘端口，可达分析将从注入抽象模型给定的一包头空间集，这个集合将通过不停地匹配模型中的转发规则来完成可达运算，最终，在端口处，它以其应有的子集形式从流出。对流出的包头子集再运行验证模块提供的包头空间回溯功能后，将能够获得一个包头集合，这个集合是能够从到达的最大集。在可达分析的过程中，包头集将经历一系列的抽象模型下的转发规则，这些规则与数据平面中真实的转发规则构成映射关系，所以将会与它能够经过的转发规则构成一定的关系，此外还有一些其他的信息，如中间进出端口等。这样的一些信息整合一起后，测试包生成模块将输出一个重要的数据体--端到端全对可达表，该表是后续选择模块的使用基础。所有有价值的信息，最终都被写入一测试包数据库中。更具体的细节将在后面章节展示。
3. **定位测试包选择模块。**该模块主要用于选择特定的测试包集合用于完成对数据平面的动态测试工作。在这个模块中，核心是一种定位测试包选择算法。它首先读取测试包数据库中的端到端全对可达表，然后在这张表的基础上，结合定位测试包选择算法和包头空间采样，完成定位测试包的生成工作，并将结果写入测试包数据库中。定位测试包集合将对网络中的可达转发规则有很好的故障定位识别能力。关于该模块的具体阐述和更多细节将在后面的章节里进行展示。
4. **诊断分析模块。**该模块用于主导定位测试数据包的收发以及分析收发结果。分析结果将用于定位运行时出现的故障。对于定位测试包的收发工作，不同的网络环境可以采取不同的策略。具体细节将在后面的章节中进行阐述。定位出的故障将报送给网络管理员，用于进一步的处理。DPEagle相较于其他工具而言，在故障报告准确率上有更好的表现。该表现将以实验的形式在实验结果展示环节进行展示。

下面将从端到端全对可达表生成方法、定位测试包选择算法以及系统的具体设计与实现三个大方面对DPEagle展开进一步说明。

# 二、端到端全对可达表生成方法

在系统概览中，可以看到，通过网络验证模块和测试包生成模块的配合，最终输出到测试包数据库中的主要内容 是一张端到端全对可达表。这张端到端全对可达表将是后面定位测试包选择模块的基础。

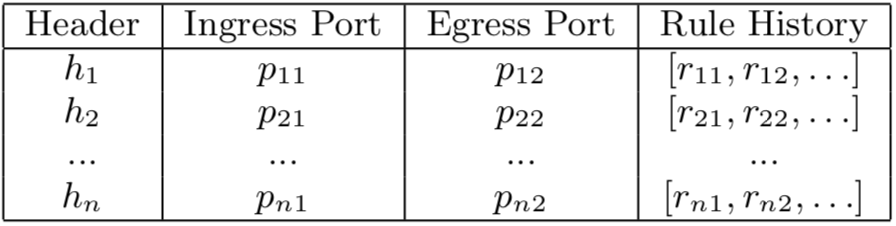


图2-1 端到端可达表的基本形式

上图给出了DPEagle所使用的端到端全对可达表的基本形式。在这张表中，Header指的是包头空间，它是一个关于{0,1,x}的头空间；Ingress Port指的是入端口；Egress Port指的是出端口；Rule History是一个关于网络规则的序列。一条表项所表达的意思为，包头空间从边缘端口进入网络后将从边缘端口出网络并且在这个过程中所经过的网络规则历史序列为[,,...]。换句话说，将从注入网络后，经过[,,...]的转发处理后，其能够从流出网络。

对于传统的网络测试工具，如Ping、traceroute等，整个网络将被看作一个黑盒子，网络管理员无法提前预知到所发送的测试包在网络中将经过哪些规则的处理。在黑盒式的网络视角下，网络管理员往往需要结合多年的测试经验对网络进行手动测试才能发现问题所在，这使得故障排查的效率非常低下。从上一段的描述中可以看到，在DPEagle下，整个网络以白盒视角存在，这样便能够对网络进行一种自动化地测试和故障定位。端到端全对可达表为这种白盒式自动化网络测试提供了基础。为了生成这样的一张全对可达表，便需要一种网络数据平面的验证工具的参与。

在前沿的网络数据平面验证工具中，HSA（Header Space Analysis）是一款开源且功能强大的验证工具。DPEagle在网络验证模块中所采用的验证工具便是HSA。下面将从HSA的基本工作原理出发阐述端到端全对可达表是如何生成的。

## 2.1关于网络数据平面验证工具HSA的简述

HSA的核心思路是将真实网络中的数据平面映射到一抽象模型上。在抽象模型之上所定义相关数据结构和运算将能够模拟数据平面的转发过程。HSA所构建的抽象模型是一种几何模型，在这个几何模型中，主要定义了两个空间，一个是包头空间（Header Space），另一个是网络空间（Network Space）。数据包被看作包头空间中的点，数据平面的转发单元（交换机、路由器等）被看作是定义在网络空间上的一种转移函数，网络拓扑也被看作是定义在网络空间上的一种转移函数。下面阐述HSA中所使用的几何空间以及转移函数的定义：

***包头空间****，*

数据包的包头在HSA所构建的几何模型中是被看作协议无关的，一个数据包的包头将被看作是一个关于0和1的序列。如果数据包包头允许的最大长度为L，那么一个包头将是关于空间中的一个点，一条流将会是该空间中的一个区域。这样一来，便被定义为包头空间。最常在包头空间下使用的对象是通配表达式。通配表达式是一个长度为L的关于{0，1，x}的序列。如此，通配表达式便可看作是下的一个超立方。在下的区域或者流随之而能够被定义为一个关于通配表达式的并集。

***网络空间****，*

HSA将整个网络数据平面建模为一个关于交换单元的集合，每个交换单元都具有一些端口。这里交换单元指代任何可能的转发设备（如交换机、路由器、网桥等）；端口则统一指代这些设备所具有的外部接口。网络空间是这样构建的，将端口空间（即关于交换单元的端口的集合）与包头空间作笛卡尔积后所得到的空间，记为，此时进一步地形式化表示即，这里表示当前网络的所有端口合起来构成的端口序列。这个空间中的一个点表示一个包头为的数据包正处在某个网络端口上，由于往往是网络链路的某一个端点，所以其也能反映数据包关于网络链路的遍历情况。

***网络转移函数****，*

数据包在网络中的移动可以看作是从网络空间中一个点变换到另一个点。比如说，在不考虑包头重写的情况下，对于一个仅仅是将数据包从一个端口发送到另一个端口的二层交换机而言，它实际上是在将数据包沿着的端口轴（即端口空间）作点的变换。又比如，对于一个可以重写匹配域（如MAC地址、TTL、checksum等）的IPv4 路由器而言，它转发数据包的过程便是在的整个空间上作变换。从这些例子中可以发现，这些网络设备均可以被看作一种转移函数，转移函数与设备本身对于网络规则的处理逻辑有关。

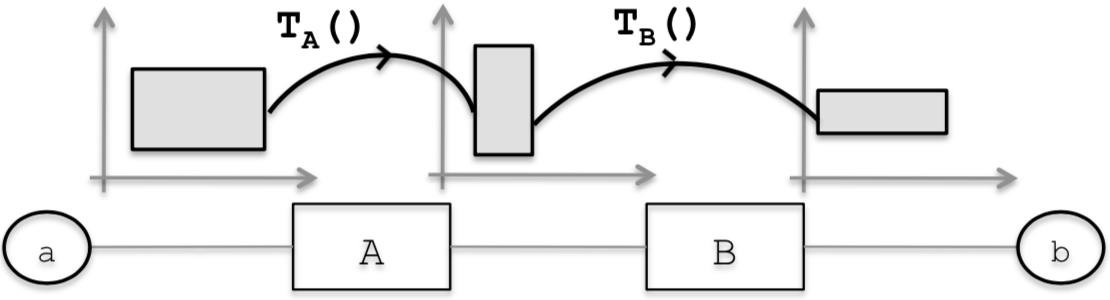


图2-2 转移函数的基本工作工程示例

上面这张图所给出的例子可以形象化地说明转移函数的工作过程。假设包头空间是二维的，即由两个匹配域构成。此时，测试主机a向转发设备A注入了一个在此包头空间下的数据包，当它到达转发设备A时，A的转移函数将数据包的包头进行变换后发送到了设备B。设备B接收并使用其转移函数对数据包的包头进行进一步的变换后，将数据包从其与测试主机b相连的端口发送出去。最终，数据包经过了两次转移函数的变换后来到了测试主机b。

一般地，一个交换单元的转发逻辑可以被建模为一个转移函数，，它是一个关于包头h和到达端口p的一个映射：

有了关于单个交换单元转移函数的定义后，接下来便可以引入关于网络转移函数的定义。网络转移函数聚合了网络中所有交换单元的转移函数。它被用来整体上描述网络的转发行为。假设网络中共存在个交换单元，并且它们的转移函数分别为。那么，便可被定义如下：

***拓扑转移函数****，*

HSA将链路也看作一种转移函数，并给出了拓扑转移函数的定义，其定义如下：

拓扑转移函数模拟的是网络中所有链路的行为。它在链路的一端接收数据包并在另一端返回相同的包头未经修改的数据包。如果是双向链路的话，需要将其一个链路看作两个链路。

有了以上几个关键的定义后，HSA便可以在其构建的抽象模型之上，模拟数据包在网络中的传递行为。在每一跳处，通过使用一种复合函数便可以完成数据包的一次模拟转发。HSA所提供的抽象模型是对于数据平面的一种映射，如果这种映射恰好能够完整还原数据平面中转发设备的主要转发行为，那么在抽象模型上预先进行某些转发行为的模拟实验，将能够尽早的发现并解决一些问题，而不用等到网络已经运行起来后才进行这些问题的搜寻和排查。HSA通过解析网络数据平面的关键信息，如拓扑结构、网络配置文件、FIB表、ACL配置等，来完成抽象模型的构建工作。像HSA这一类的工具被称为网络数据平面验证工具，它们负责在抽象模型上验证网络数据平面中的一些不变量（如无环路、无黑洞、特定节点间可达等）是否有被满足。

## 2.2利用HSA生成端到端全对可达表

虽然HSA主要是用来验证网络数据平面的，但实际上它所提供的强大的抽象模型还可以用来生成端到端全对可达表。这样一来，网络便呈现出了一种白盒化的视角，其使得关于网络数据平面的白盒测试变得可能。

DPEagle主要在HSA提供的端到端全对可达分析方法之上完成对于端到端全对可达表的构建。具体地，在HSA的抽象模型上，向选定的用于测试的端口处分别注入一个全通配的包头，其头空间为all-x。假设进入测试端口，那么接下来便会被对应的网络和拓扑转移函数处理。经过一系列的网络和拓扑转移函数处理后，会转换为一包头空间中的区域，并最终从某一测试端口流出，这里假设为。在这个过程中，每到一个网络转移函数，便会被函数中相应的网络规则所处理，所以，伴随的处理过程，一个相应的网络规则历史序列[,,...]可以被生成。再利用HSA提供的包头回溯方法对从作逆转换后回到，此时输出的包头便是端到端全对可达表中Header表列所需要的结果。在系统概述中，我们提到过端到端全对可达表每条表项的意义，这里结合当前的背景再此解释，即如果将从注入后，当其从流出时，会经过的网络规则历史序列为[,,...]。

为了更好地理解端到端全对可达表的生成过程，这里我们从一个简单的网络出发看一下在其上的全对可达表生成的结果是怎样的。

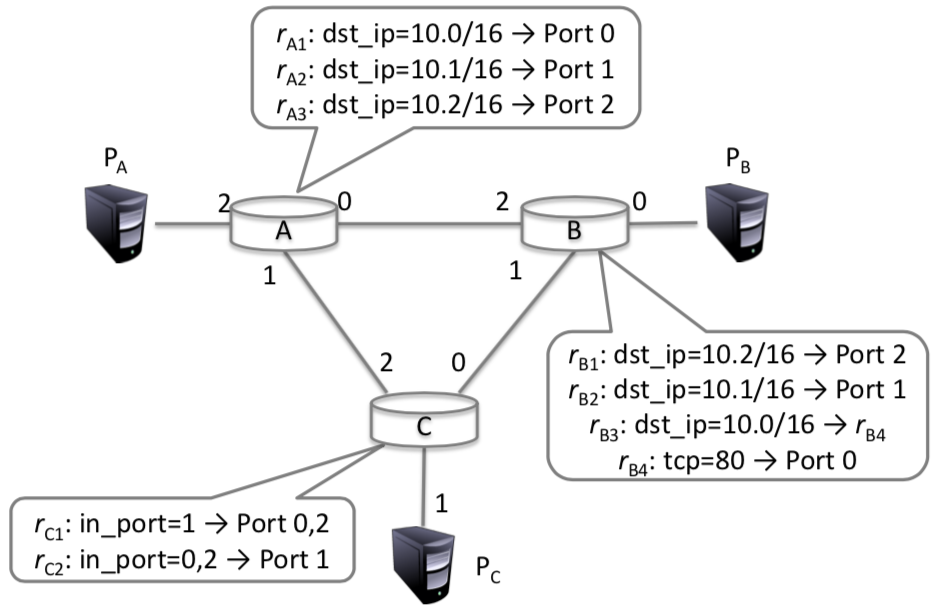


图2-3 一个简单的网络数据平面实例

上图给出了一个由三个网络交换设备A，B，C组成的网络。每个设备具有一些转发规则。在这个网络中，每个交换设备的一个边缘端口用来放置测试主机，三个测试主机为、和分别对应设备A、B和C。那么，接下来的端到端全对可达表的生成便是在这三个安置有测试主机的端口间进行的。为了避免重复性的讨论，我们以生成从端口到端口和的表项为例，展示可达表的生成过程。首先从向网络中注入一个all-x的包头，这样包头首先会来到设备A处。在A上，它的规则dst\_ip=10.0/16会将这部分包头转移到设备B，规则dst\_ip=10.1/16会转移到设备C。在设备B上，其规则dst\_ip=10.0/16和规则tcp=80会将这部分包头转移到测试主机所联结的端口上；在设备C上，其规则dst\_ip=10.1/16会从A来的相同域空间的包头发送到测试主机PC所联结的端口上。这样一来，就有两个可达表的表项被生成了。完整的端到端可达表如下表所示（这里我们假设只有两个匹配域，dst\_ip和tcp目的端口号，其域的长度分别为32位和16位）：

表2-1 该网络下的端到端全对可达表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Header | Ingress Port | Egress Port | Rule History |
| p1 | 0000000001010000,0000101000000000xxxxxxxxxxxxxxxx  (tcp=80, dst\_ip=10.0/16) |  |  | RA1, RB3, RB4, link AB |
| p2 | xxxxxxxxxxxxxxxx,0000101000000001xxxxxxxxxxxxxxxx  (dst\_ip=10.1/16) |  |  | RA2, RC2, link AC |
| p3 | xxxxxxxxxxxxxxxx,0000101000000010xxxxxxxxxxxxxxxx  (dst\_ip=10.2/16) |  |  | RB2, RA3, link AB |
| p4 | xxxxxxxxxxxxxxxx,0000101000000001xxxxxxxxxxxxxxxx  (dst\_ip=10.1/16) |  |  | RB2, RC2, link BC |
| p5 | xxxxxxxxxxxxxxxx,0000101000000010xxxxxxxxxxxxxxxx  (dst\_ip=10.2/16) |  |  | RC1, RA3, link BC |
| p6 | 0000000001010000,0000101000000010xxxxxxxxxxxxxxxx  (tcp=80, dst\_ip=10.2/16) |  |  | RC1, RB3, RB4, link BC |

在上表中，Header列中的值是在HSA下的表达。这里两个域之间用逗号隔开，并且同时给出了HSA表达式所对应的传统表达形式以便对照和查看。前两行的表项即对应着刚才例子中给出的生成结果。可以看到，p1在对应的包头下面，从到的过程中将经过四个网络规则，这里我们将链路也看作网络规则；p2从到的过程中将经过三个规则。

在端到端全对可达表中，最重要的信息便是包头和网络规则历史序列的关系。这一关系本质上反映的是网络中的哪些规则能够被具有对应包头的数据包测试到。整个表中的规则历史序列反映的是全网中所有可达的网络规则，即这些规则在理论上均有对应的数据包能够被构造出来并在发送到网络中时能够经过它们。下面的讨论中，我们把从全对可达表中产生的数据包均称作测试包。这样伴随产生的一个很有价值的问题便是，如何选择某些表项，使得从它们的Header中构造出来的测试包，能够对网络中的所有可达规则进行高效地测试和故障定位。下面一节将阐述DPEagle是如何利用新的算法来解决这一问题的。

# 三、定位测试包选择算法

## 3.1网络规则的1-识别性

有了端到端全对可达表之后，DPEagle接下来要完成的任务是如何选择某些表项，然后生成相应的测试包，以完成对于网络规则的测试和故障定位。这里我们称所选择出的用于测试和定位的测试包为定位测试包，以有别于未被选择的其他测试包。

DPEagle所使用的定位测试包选择算法，是一种能够对网络中的可达规则实现1-识别故障定位的算法。这里首先给出关于网络规则故障1-识别的定义：

* **定义** 对于一个网络规则，如果当网络中只有这一个规则出现问题时，其问题能够被定位到，那么就说关于这个网络规则的故障是可以被1-识别的。

在端到端测试的背景下，问题规则的发现和定位是通过收发定位测试包完成的。在真实的网络中，端到端全对可达表中给出的可达网络规则并不总是均可以被1-识别的，DPEagle的目标便是选择出一定数量的定位测试包使得网络中的可达规则尽可能地被1-识别。

对网络规则实现1-识别的定位测试包选择方法的一般性步骤如下：首先我们将端到端全对可达表中的Rule History列给出的历史序列做集合并运算得到一个并集，这个并集中含有的是网络中的所有可达规则。接下来，选择表中的一个测试包，其历史序列是关于集合的真子集，这样一来，可以对做一个划分，使得存在于中规则被分到一个新的集合中，这样就得到了两个规则集合。不断地重复上面的过程，直到不再有新的测试包能够对当前存在的规则集合展开进一步地划分为止。最终，得到的规则集合间具有这样的特点，即在关于选择的测试包的覆盖模式上，它们两两之前是不同的。换句话说，这样的选择之后，所选择的测试包对规则集合实现了1-识别。如果所有的规则集合中都仅含有一个网络规则，那么这样的选择实际上也完成了对于可达规则的1-识别。下面给出一个例子，用于具体展现一般性方法的步骤。

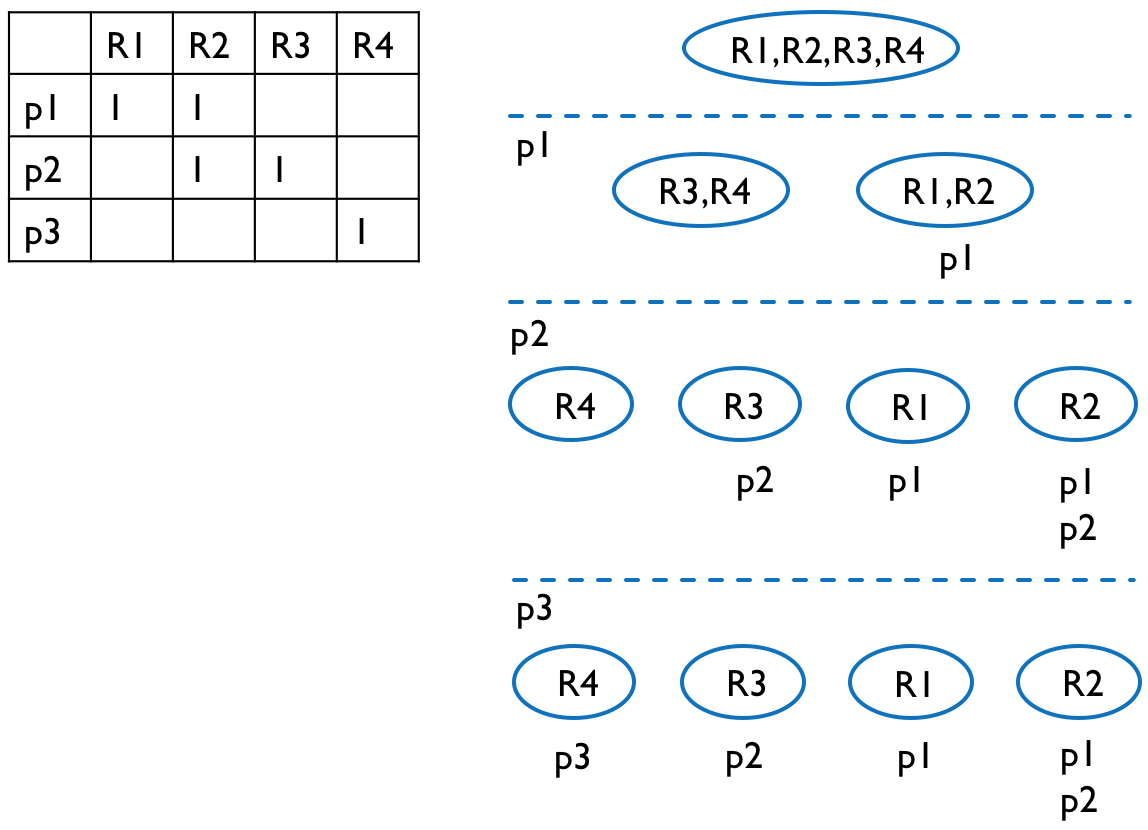


图3-1 分割过程示例

在上图中，左边的表是一个简化的端到端全对可达表，表中只有测试包和可达规则间的关系。其中1表示规则是相应的测试包的历史序列中的元素。可以看到，一共有三个测试包和四个可达规则。划分的过程如右边的子图所示，可以看到，在完成了选择之后，一共有四个规则集合出现。在这个例子中，刚巧每个规则集合都只含有一个规则，并且三个测试包均有被选到。可以看到，规则集合下的覆盖模式之间是不同的，又每个规则集合只含有一个规则，所以每个规则将会是1-识别的。后面将会阐述在1-识别下的问题规则是如何被定位，这里暂时先不做讨论。

在真实的网络环境下，往往会有一些规则集合存在大于一条网络规则的情况，那么在同一个规则集合中的网络规则将不具有1-识别性。在相关实验展示部门我们将看到，即便无法对所有可达规则实现1-识别，这样的选择方法也大大增加了关于网络规则故障的定位准确率，具有很好的应用价值。

## 3.2选择算法的基础版本

在一般性步骤中可以看到，每次划分所使用的定位测试包选择标准将决定最终所选择的包的数量。在选出具有1-识别性的定位测试包的同时，我们希望包的选择数量越少越好，这样可以减小对网络中正常流量的干扰。此外，我们还希望生成定位测试包的速度越快越好。本节，主要介绍DPEagle所使用的选择算法的基础版本。

实际上，网络规则故障1-识别的概念来源于故障1-识别这一基本概念。Mark Brodie等人专门探讨过实现故障1-识别的几种可能的通用算法。为了满足之前提出的期望，DPEagle首先在Mark Brodie等人提出的几种实现1-识别性的通用算法中，选取了定位测试包选择数量最少的一个用来做规则故障1-识别下的选择：

这里，等式的意思是，给定一个规则分割集合序列S，待选测试包packet在S上所具有的比较值为V(packet)。这个等式衡量的意义是如果选了packet之后，新的分割集合序列理论期望上还需要多少的测试包来完成接下来的划分工作。在每次选择中，V(packet)最小的测试包将被选出来作为定位测试包。

由于选取的这一算法具有的算法时间复杂度为（p为端到端全对可达表的表项总数，也即可选测试包的总数；n为可达规则的总数），其具有较高的复杂度，不能满足速度快的目标，因此，DPEagle做了一步改进，利用如下的等式衡量每次迭代中，待选测试包所具有的比较值：



这里基于的观察是，一个被选择的测试包实际上只会对部分规则集合构成有效的划分。更进一步，和在本质上是等价的，即G的最大值将意味V的最小值。通常而言，划分数量相对于整个规则集合的数量要小很多，所以计算的时间将会被有效的缩短。

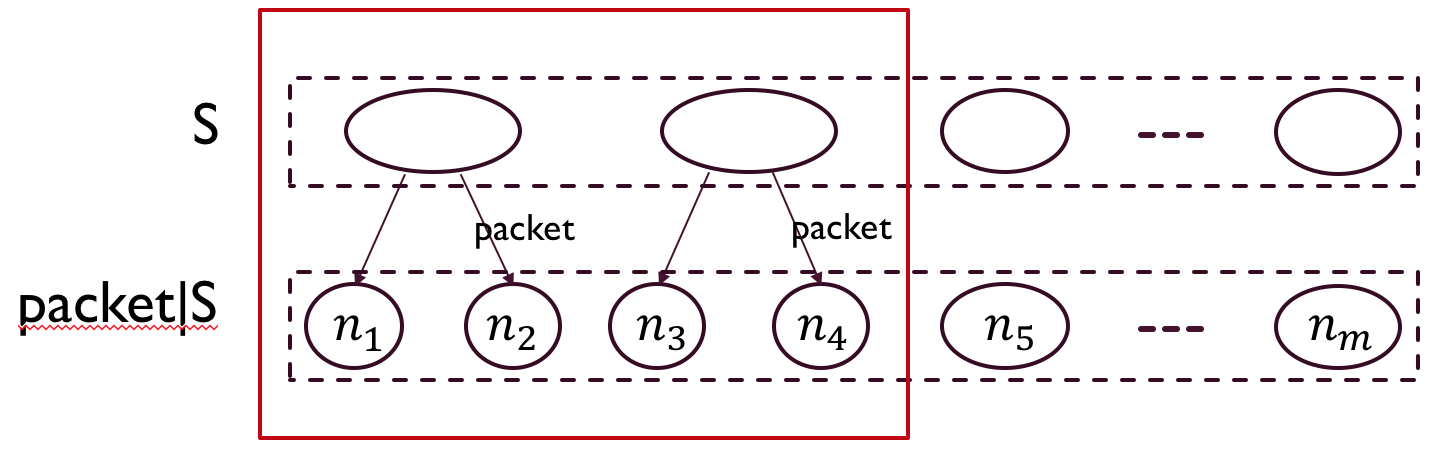


图3-2 部分划分的形象化表示

上图给出了所基于的观察的形象化表现。可以看到，被选择的packet所划分的每个规则集合可以看作被分成了左右两个子集合，这也就是表达式中和的由来。

除了选择时间外，DPEagle又在的基础上提出了如下等式，用于将测试包的数量进一步减少：

这里是关于能够被packet分割的规则集合数量R的函数，依照经验进行调整，通常取。等式是启发性的，即gain值较大的测试包意味着其具有的G值较大或者其能够分割的规则集合的数量较多，这两点都有可能使最终选择的定位测试包的总数减少。

## 3.3选择算法的加速版本

虽然选择算法的基础版本已经在运算时间上做了一定的改进，但是根据实验给出的经验，它在中等规模的网络下，仍然会运行超过几小时甚至一天的时间。为此，DPEagle进一步在速度上对选择算法进行了优化并提出了两个优化策略。其思路如下：

第一个策略的核心思想是建立一个最大堆（Max-Heap），堆中维护的是测试包的增益值，在每次探测包选择中，一个测试包被选择出来的标准为，如果其在堆顶，并在弹出、更新增益值、再推入堆后，堆顶仍为此测试包，那么就选择此测试包为定位测试包并将其再次弹出堆。如此进行定位测试包的选择，直到堆中不再有可供选择的测试包或者每个规则集的信息量为零。这样做的优势在于，尽管大量的测试包的增益值没有被更新，但是由于测试包的增益值随着划分的进行是基本递减的，所以堆顶测试包在更新后再次来到堆顶这一现象意味着即便在全体更新后，这一测试包仍为具有最大增益值的包，符合选取条件。选取堆作为存储测试包增益值的数据结构的好处在于，堆的弹出和推入操作的时间复杂度为)。这一策略称作*Lazy Update*。其次，观察到，测试包的增益值有聚类的特点，即多个测试包在给定的规则集合下，将具有相同的增益值，所以，进一步加速探测包的选取的方法为，引入一个扰动量 加在增益值上。



这里heap\_gain表示加上扰动量之后的增益值。扰动量要保证每个测试包在具有不同的增益值的同时，又不破坏本该具有的关于增益值的相对大小变化，这样，堆在弹出和推入的过程中将很快遇到满足开始时提到的选取条件的测试包。这一策略称作*Gain Perturbation*。

经过加速后，经过实验表明，在中等网络规模下，DPEagle能够在一个小时内完成定位测试包的选择工作，大大缩短了定位测试包的选择时间。

## 3.4可用测试包的生成

在选择了定位测试包之后，未来就需要下发这些测试包到真实的网络中去。这里存在的问题是，选择出的定位测试包还不能直接拿来用，这是因为端到端全对可达表的Header列中的包头是以关于{0,1,x}的表达式被保存的，并且最关键的是，根据HSA的运算特性，它们是以的形式存在的。所表示的是一个包头空间下的某个范围，为了发送测试包，需要从中采样一个点。采样的方法有两种，第一种，是将完全展开，展开的方法HSA有提供，就是一种集合运算，但是，这个方法的缺点是，展开要花费大量的时间，无法满足快速采样的要求。因此，DPEagle采取解SAT的形式来直接从 中采样。由于在实际运用中，通常只有一项，所以只需要考虑。对于SAT表达式，DPEagle采用的是合取范式的形式。表示成合取SAT表达式的方法是这样的，首先将A中的表达式翻译为，这里均是A中不为x的位；对于，由于其在集合减运算之后，所以先将看作，这样对于一属于，其具有的表达式可以翻译为，刚好是析取的形式，其中的均是中不为x的位。所以翻译过来的最终形式为。这样一来，通过采用主流开源的SAT Solver便能够在较短的时间内获得一个可用的包头。

## 3.5 1-识别下的故障定位方法

在获得了可以下发的定位测试包之后，DPEagle接下来便可以利用这些测试包去测试网络规则和以及定位故障。这一节主要说明DPEagle是如何发现和定位故障的。

在定位测试包选择一节，注意到当选择完成时，每个规则集合下面都会有一个关于定位测试包的覆盖模式，这一覆盖模式被称作*定位模式*。DPEagle通过这一模式完成故障的发现和定位。具体地，在选择结束后，对于一个规则集合，假设其关于定位测试包的定位模式为，即有k个被选择的测试包其规则历史序列中含有。定位模式中保存的是测试包的编号，其是一个关于测试包编号的有序列表。测试包编号在模式中按大小顺序来保存。在1-识别性下，定位模式的意义是，若任意时刻网络中只会有一个规则集合出现问题的话，当我们下发并接收了定位测试包之后，如果分析发现无法被正确接收的测试包所具有的编号序列刚好对应某一规则集合下的定位模式的话，那么便可以立即得出结论，即这一规则集合中的规则出现了故障。当规则集合中只有一个规则时，便可以定位到是哪个规则出现了问题。

在定位的过程中，可以看到，如何判断定位测试包是否被正确接收是后续定位的基础。DPEagle是这样做的，对于一个定位测试包p，首先找到其表项中Ingress Port下所记录的入端口号，通过键值在测试包数据库中查找对应的所有表项。由于表项中的Egress Port记录的是测试包的出端口，所以将查找出的表项中的所有出端口组成一个集合，这一集合便是测试包接收正确的标志，即如果p是正确接收的当且仅当得到的关于p被接收的实际端口构成的集合等于。在实际环境中，会存在规则出现故障，但是对应的定位测试包仍然被正确接收的情况，这样的故障，DPEagle将无法处理，DPEagle暂时只采用基于出端口集合的判断方法。

# 四、系统具体设计与实现

## 4.1概述

在图1-1中我们给出了DPEagle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网络验证模块

DPEagle使用HSA网络验证工具完成网络验证模块的构建。系统采用了自动化网络测试工具ATPG的开源代码中提供的HSA源码作为实现的基础。ATPG（Automatic Test Packet Generation）是学界中较早使用网络验证工具进行白盒化网络自动化测试的工具之一。它所依赖的验证工具即是HSA。HSA还有其他的开源版本，但是由于系统在测试包生成模块的实现上还会依赖到ATPG的其他源代码，所以在此采用ATPG中提供的HSA实现。

网络验证模块的主要任务，便是解析网络数据平面中的与转发相关的文件（如配置文件、FIB表、ARP表等）将其转化为对应的抽象模型。ATPG提供了两个真实网络下的数据集，一个是Stanford，另一个是Internet2。Stanford网络所使用的转发设备是Cisco路由器，而Internet2所使用的是Juniper路由器。因此，相应的，在HSA的源码中，有两个网络解析器，一个关于Cisco，一个关于Juniper。不论何种解析器，在实现上，最终都会首先为数据平面下的每一个转发设备生成一个TF文件，再为拓扑结构生成一个TF文件。这里，根据之前对于HSA的介绍中可以知道，TF即是转移函数，转移函数被保存在了文件中。我们这里从HSA源码中的文件tf.py中的一个函数来看TF文件中保存了哪些信息：

def save\_object\_to\_file(self, file):

f = open(file, 'w')

**# 关注点1**

f.write("%d$%s$%d$%d$%d$\n"%(

self.length,self.prefix\_id,self.next\_id,

self.lazy\_eval\_active,self.send\_on\_receiving\_port))

for nibble in self.lazy\_eval\_nibbles:

f.write("%d$"%nibble)

f.write("#\n")

**# 关注点2**

for rule in self.rules:

f.write("%s$"%rule["action"])

f.write("%s$"%rule["in\_ports"])

f.write("%s$"%byte\_array\_to\_hs\_string(rule["match"]))

f.write("%s$"%byte\_array\_to\_hs\_string(rule["mask"]))

f.write("%s$"%byte\_array\_to\_hs\_string(rule["rewrite"]))

f.write("%s$"%byte\_array\_to\_hs\_string(rule["inverse\_match"]))

f.write("%s$"%byte\_array\_to\_hs\_string(rule["inverse\_rewrite"]))

f.write("%s$"%rule["out\_ports"])

f.write("#")

for ra in rule["affected\_by"]:

f.write("%d;%s;%s#"%(self.rules.index(ra[0]),

byte\_array\_to\_hs\_string(ra[1]),

ra[2]))

f.write("$")

f.write("#")

for io in rule["influence\_on"]:

f.write("%d#"%self.rules.index(io))

f.write("$%s$"%rule["file"])

for ln in rule["line"]:

f.write("%d,"%ln)

f.write("$%s$\n"%rule["id"])

f.close()

这里给出了tf.py文件中的一个关键函数save\_object\_to\_file的实现。save\_object\_to\_file主要用于TF文件的最终写入操作。在这段代码中，有两处关注点。在关注点1处，可以看到，TF文件的第一行被写入的信息有，self.length、self.prefix\_id、self.next\_id、self.lazy\_eval\_active、self.send\_on\_receiving\_port等，它们是TF文件的元数据，用来说明TF文件的一些性质。比如，length主要用来指明网络中所使用的各匹配域长度的总和，这里length会是总和的两倍，主要与HSA模型下的运算有关。prefix\_id用来指明TF文件所对应的网络设备的标识符。其他几个信息不太重要，这里不再赘述。关注点2主要用来将网络规则以转移函数的形式写入TF文件中。可以看到，在网络规则的转移函数中，涉及到的变量有，action、in\_ports、match、mask、out\_ports、affected\_by和influence\_on等。这些变量共同在一起组成了转移函数的函数体，在抽象模型处理的过程中，一个包头空间会首先根据in\_ports查找其能够经过的规则，然后通过与match和affected\_by做一定的运算，完成新的包头空间的生成，并在生成之后从out\_ports指定的端口中输出。affected\_by记录该网络规则的匹配域会与哪些高优先级的规则有重叠。influence\_on则记录该网络规则的匹配域会重叠哪些低优先级的规则。

有了TF文件之后，HSA便可以在它们之上实现一些功能，比如端到端可达性分析、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生成端到端全对可达表。在HSA的基础上，该模块通过在HSA提供的抽象模型上进行可达性分析来完成全对可达表的构建。上一节中有提到自动化测试工具ATPG，由于ATPG也需要在HSA的基础上生成端到端全对可达表，所以其源代码中已经给出了端到端全对可达表在HSA基础上的实现。DPEagle采用ATPG提供的端到端全对可达表生成源码作为其测试包生成模块的基础。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 find\_reachability\_test(NTF, TTF, in\_port, out\_ports, input\_pkt):

paths = []

propagation = []

# 关注点1

p\_node = {}

p\_node["hdr"] = input\_pkt

p\_node["port"] = in\_port

p\_node["visits"] = []

propagation.append(p\_node)

# 关注点2

while len(propagation) > 0:

tmp\_propagate = []

for p\_node in propagation:

# 关注点3

next\_hp = NTF.T(p\_node["hdr"], p\_node["port"])

for (next\_h, next\_ps) in next\_hp:

for next\_p in next\_ps:

new\_p\_node = {}

new\_p\_node["hdr"] = next\_h

new\_p\_node["port"] = next\_p

new\_p\_node["visits"] = list(p\_node["visits"])

new\_p\_node["visits"].append(p\_node["port"])

# 关注点4

if next\_p in out\_ports:

paths.append(new\_p\_node)

# 关注点5

linked = TTF.T(next\_h, next\_p)

for (linked\_h, linked\_ports) in linked:

for linked\_p in linked\_ports:

new\_p\_node = {}

new\_p\_node["hdr"] = linked\_h

new\_p\_node["port"] = linked\_p

new\_p\_node["visits"] = list(p\_node["visits"])

new\_p\_node["visits"].append(p\_node["port"])

if linked\_p not in new\_p\_node["visits"]:

tmp\_propagate.append(new\_p\_node)

propagation = tmp\_propagate

return paths

首先看到函数的参数，有NTF、TTF、in\_port、out\_ports和input\_pkt。NTF是关于网络转移函数的对象；TTF是关于拓扑转移函数的对象；in\_port是当前选择的用于进行可达性分析的边缘入端口；out\_ports对应的是所有开放的边缘端口；input\_pkt是注入到in\_port中的一个headerspace对象，在hs.py文件中有具体细节的呈现，它所具有的重要的属性有包头值，其在初始化的通常取all-x；规则历史序列，用于记录每次遍历中被NTF的哪个规则所处理。下面我们来看关于in\_port端口的可达分析是如何进行的。

可达分析是以广度优先遍历的思路开展的。在关注点1处，给出了一个初始节点p\_node，其有三个属性：hdr表示其具有的包头值；port表示该节点当前所处的端口号；visits保存的是到达该节点之前已经经过的端口。在关注点2处，便正式开始了可达分析，可达分析结束的标志是当不再有新的端口可以继续被转移函数处理时，停止遍历。在关注点3处，可以看到开始调用网络转移函数NTF中的变换，完成节点的传递，转移之后将到达相应转发单元的多个出端口，在出端口处，新的节点被生成。在关注点4处，首先判断出端口是否为out\_ports中保存的边缘端口，如果是，那么就有一条可达路径产生。在关注点5处，如果出端口不在给定的边缘端口集中，那么就调用拓扑转移函数TTF，使得新的节点进一步转移变换到多个新的入端口上，这些入端口存在于一个或多个下一跳转发单元上。将下一跳中没有环路产生的新节点保存在广度优先遍历列表中，回到while处，如果遍历列表非空，则继续，否则终止。终止遍历后的函数将返回可达路径集。

在获得了可达路径集后，端到端全对可达表基本已经生成，此时还差另一个重要步骤，便是最终输出的节点所保存的包头不能直接作为可达表Header列中的值。此时，还需要将边缘出端口处得到的包头值进行包头回溯的操作。回溯操作在HSA的学术论文中有具体阐述，这里不再赘述，其在ATPG源代码中也相关的具体实现。在3.4节中，我们曾经阐述了可用测试包的生成方法，该方法用来从HSA包头表达式中采样一个可在真实网络中进行传输的具体包头值。在第一章系统概述中，我们知道，采样工作在理论上，是放在定位测试包模块中实现，但是出于快速原型的考虑，我们决定在测试包生成模块中完成采样工作，即在端到端全对可达表生成的同时，为每一个表项生成了相应的采样包头。下面对采样的具体实现进行阐述。

我们通过使用开源的SAT Solver--PicoSAT的Python Binding--pycosat完成包头的采样工作。首先将HSA表达式转换为DIMACS的形式并适配pycosat关于DIMACS的处理方法，pycosat的README中给出了适配的具体例子，这里不再赘述。接下来，调用pycosat提供的solve方法，解出一个合法的合取范式。最后，再将得到的合取范式转为HSA下的表达式，这样便完成了包头的采样工作。

最终生成的端到端全对可达表以及采样出的包头将写入到测试包数据库中，采样出的包头将作为新的一列，保存在端到端全对可达表中。

## 4.4测试包数据库

测试包数据库是用来存放与网络测试诊断有关的数据，这些数据将会被多个模块使用到。在实现上，我们使用SQLite DBMS来操作测试包数据库。我们这里给出测试包数据库中最关键的几张表的结构。

表4-1 端到端全对可达表（test\_packets）

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Value Example** |
| header | TEXT | {"ip\_dst\_wc": 0, "vlan\_match": [0], "ip\_dst\_match": ["64.57.20.191"], "vlan\_wc": 0} |
| hs | TEXT | 000000000000000001000000001110010001010010111111 |
| input\_port | INTEGER | 800026 |
| output\_port | INTEGER | 820034 |
| ports | TEXT | 800026 820034 |
| no\_of\_ports | INTEGER | 2 |
| rules | TEXT | seat\_1 seat\_41 |
| no\_of\_rules | INTEGER | 2 |

test\_packets是端到端全对可达表在测试包数据库中的表名。它有几个列，这里简单介绍下。header表示的是包头域的文本形式，它是从hs列中给出的HSA表达式中采样一个包头点得到的，这里为了充分利用ATPG的源代码，我们在快速原型实现的时候，将采样的工作放到了端到端全对可达表生成时做，而不是在定位测试包选择之后才做，但是后面会在定位测试包选择模块的设计和实现一节阐述采样是如何做的；input\_port是这个表项所对应的测试包的边缘入端口；output\_port是边缘出端口；ports表示是该测试包在可达分析的过程中会经过的端口序列；rules表示的是可达分析的过程中会经过的网络规则序列。

表4-2 网络规则表（network\_rules）

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Value Example** |
| rule | TEXT | yozb\_rtr\_686 |
| input\_port | TEXT | 1610000 |
| output\_port | TEXT | 1610002 |
| action | TEXT | rw, fwd |
| file | TEXT | ../data/Stanford\_backbone/yozb\_rtr\_route.txt |
| line | TEXT | 6 |

network\_rules表用于记录解析器解析出的规则。其中列file和line共同标记出了一条网络规则表项在网络配置文件中的具体位置，以方便故障的精确定位。该表将在解析器解析网络配置文件时生成。

表4-3 可达规则表（reachable\_rules）

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Value Example** |
| rule | TEXT | yozb\_rtr\_686 |

该表记录的是网络规则中的可达规则。所谓可达规则，指的是端到端全对可达表中测试包所能够测试到的所有网络规则的集合。在实现上，获得该表的方式是，将test\_packets表中的rules做集合并，所获得的并集便是所有的可达规则。

表4-4 拓扑规则表（topology\_rules）

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Value Example** |
| rule | TEXT | link\_1 |
| input\_port | TEXT | 220036 |
| output\_port | TEXT | 600025 |

topology\_rules用于记录拓扑链路。由于链路也可以看作一种网络规则，所以称其为链路规则。该表将在解析器解析网络拓扑配置时生成。

表4-5 定位测试包表（loc\_test\_packets）

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Value Example** |
| rules | TEXT | seat\_1 seat\_41 |
| no\_of\_rules | INTEGER | 2 |

该表是定位测试包模块调用定位测试包算法后获得的。由于选择是在全对可达表上完成的，并且选择是由可达表中的规则历史序列决定的，所以loc\_test\_packets中主要记录两列，规则历史序列rules和序列中的规则数量no\_of\_rules。

表4-6 定位模式表（rule\_cover\_map）

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Value Example** |
| pkt\_pattern | TEXT | 531 2031 3217 |
| rule | TEXT | seat\_2788 |

定位模式表rule\_cover\_map记录了每个可达规则所具有的关于定位测试包的定位模式。pkt\_pattern保存的是关于定位测试包的编号。该表由定位测试包模块生成。诊断分析模块将使用该表提供的定位模式来完成故障规则的定位工作。

## 4.5定位测试包选择模块

这里出于快速原型的考虑，DPEagle使用Python2.7实现了定位测试包选择模块。定位测试包选择模块的主要功能为，在端到端全对可达表的基础上，使用定位测试包选择算法完成测试包的选择，并将选择后的相关信息，如所选择的测试包以及定位模式等，写入测试包数据库供后续的进一步使用。

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4-1 定位测试包选择模块的工作流程图

在定位测试包模块的实现代码中，有几个重要的函数，对应工作流程图中的不同部分，接下来分别对它们进行介绍。

1. **gen\_matrices\_from\_database。**该函数用于读取相关网络的端到端全对可达表，并构建后续要使用的数据对象。有两个数据对象会产生，一个是t\_packetMatrix，另一个是ruleMatrix。t\_packetMatrix是一个关于测试包的规则历史序列的数组，序列的排列顺序与可达表表项的读取顺序一致，序列所在的位置编号即为对应测试包的编号。ruleMatrix是一个关于可达规则的字典对象。每个可达规则将映射到一个关于测试包的数组，该数组存储的是所有能够经过它的测试包的编号。
2. **computeProbes。**该函数用于选择定位测试包。在端到端全对可达表中得到的信息的基础上，该函数主要通过运行定位测试包选择算法来完成定位测试包的选择工作。
3. **get\_ruleset\_to\_packet\_ids。**该函数在computeProbes中被调用。在定位测试包选择算法运行完毕后，该函数将被调用。它返回的是一个关于分割规则集合的字典对象。每个分割规则集合将映射到一个关于定位测试包的数组，该数组记录实际上就是关于分割规则集合的测试包定位模式。
4. **store\_test\_pkts\_to\_database。**该函数在get\_ruleset\_to\_packet\_ids后被computeProbes调用。它将定位测试包所对应的规则历史序列以及历史序列中包含的规则数存入测试包数据库的loc\_test\_packets表中。由于定位测试包的选择只受规则历史序列的影响，所以在实现的过程中，只保存规则历史序列，而不保存完整的端到端全对可达表表项。
5. **store\_RuleCoverMap\_to\_database。**该函数作为最后一个函数被computeProbes调用。它在分割规则集合到定位模式的字典的基础上，将每个可达规则对应的定位模式保存到测试包数据库的rule\_cover\_map表中。

## 4.6诊断分析模块

我们在Beacon SDN控制器上完成了诊断分析模块的构建。该模块以控制器自定义应用程序的形式被实现。诊断分析模块有三个主要部分：测试包构建、测试包下发以及故障定位。三个部分由一个主程序统一调配来诊断分析网络可能存在的故障。下面简要梳理三个部分所提供的主要功能。

测试包构建部分提供的主要功能有：访问测试包数据库、构建可用的定位测试包、构建可达规则的理论定位模式、构建测试包的理论接收模式等。

测试包下发部分提供的主要功能是定位测试包的下发。由于SDN控制器具有操纵SDN交换机进行收发包的功能，因此，我们不依赖边缘测试主机，而是通过SDN控制器发送指令的方式，让SDN交换机从边缘端口直接发送定位测试包。发送的指令为packet-out消息，该消息中封装有需要下发的定位测试包，当SDN交换机收到该消息时，将把包含的定位测试包直接从指定端口发到网络中。

故障定位部分提供的主要功能有：构建测试包的实际接收模式、分析测试包的接收情况、定位故障等。

主程序除了具有协调三个部分的功能外，还具有收取定位测试包的功能，其通过解析SDN交换机发送的packet-in消息来获取其中封装的定位测试包。

在实现上，诊断分析模块的包目录结构主体部分如下：

net.beaconcontroller.mahak

|\_Mahak.java

net.beaconcontroller.repository

|\_DBPacket.java、TestPacket.java

net.beaconcontroller.worker

|\_PinpointerBase.java、PusherBase.java、ReaderBase.java

net.beaconcontroller.workerDPEagle

|\_ PinpointerDPEagle.java、PusherDPEagle.java、ReaderDPEagle.java

其中，Mahak.java是主程序；repository中的两个数据结构与从测试包数据库中读取到的信息有关；work中给出了与三个主要部分有关的Java基类；workerDPEagle中给出了与DPEagle有关的三个部分在基类上的具体实现类。下面我们首先给出三个部分的类图。

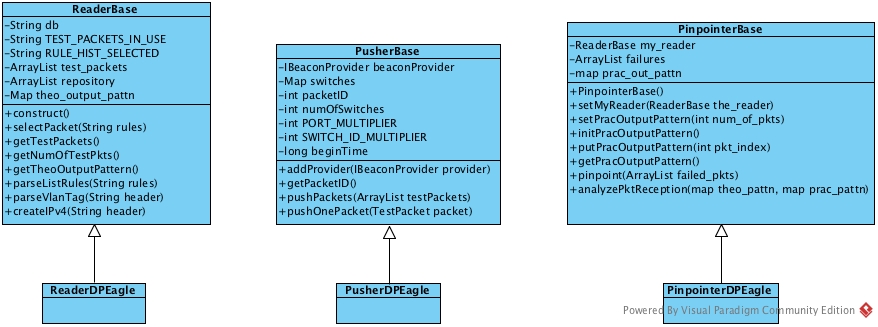


图4-2 诊断分析模块的三个主部分的类图

该类图中给出了每个部分所具有的主要属性和函数，Reader代指构建；Pusher代指下发；Pinpointer代指定位。由于子类中的不同内容较少，这里不再展示，具体细节请查看模块的源代码。

在三个主要部分提供的API的基础上，主程序Mahak便可以完成诊断分析的完整逻辑。下面给出基于具体函数的Mahak执行逻辑图。图中的多数函数均省略具体输入参数，名称与源代码中的一致。该图只展示关键逻辑，在具体的实现中，可能会有一些修改，但是总的运行思路保持不变。



图4-3 Mahak的诊断分析主逻辑图

在执行逻辑图中，红色实线箭头表示Mahak主线程的执行过程；红色虚线箭头表示在Mahak创建的分线程的执行过程。Mahak将从startup函数开始运行。在进行了与所诊断的网络相关的操作后，Mahak为ReaderDPEagle、PusherDPEagle和PinpointerDPEagle分别创建了一个实例，记为pkt\_reader、pkt\_pusher以及pinpointer。之后，Mahak将创建两个线程thread 1和thread 2。thread 1对应测试包构建部分。thread 2对应测试包下发和故障定位部分。在实现上，要保证thread 1先于thread 2执行。下面分别介绍thread 1和thread 2的具体工作流程。

1. **thread 1。**thread 1中的主函数为constructTestPackets，它主要顺序调用两个函数，先是pkt\_reader提供的construct函数，再是pinpointer提供的setPracOutputPattern函数。
2. **thread 2。**thread 2中的主函数为pushAndPinpoint，它主要顺序调用三个函数，首先是利用pkt\_pusher提供的pushPackets方法完成定位测试包的下发工作，在下发完后，线程首先睡眠给定的几秒钟时间，此睡眠时间用于Mahak完成定位测试包的接收工作，接收主要通过其receive函数完成。之后，thread 2利用pinpointer提供的analyzePktReception函数完成定位测试包接收正确性的判断，并返回一个关于未被成功接收的定位测试包的序列failures。最后，通过将failures传递给pinpointer提供的pinpoint的方法，诊断分析模块将定位到可能出现问题的网络规则故障，被报告该故障给网络管理员。

为了方便后续对于诊断分析模块的功能检验，我们在Mahak中又嵌入了与功能测试有关的函数和代码块，功能测试主要是通过把规则的删除看作规则故障，来查看DPEagle能否通过诊断分析成功定位到被删除的规则。功能测试的实现细节请参考Mahak中的相关代码。功能测试的进行还依赖于可用的测试网络环境。这里我们选择在Mininet中用SDN交换机复刻给定的网络数据集。SDN交换机我们选择的是Open vSwitch。Mahak中提供了流表项的下发功能，该功能可编译OpenFlow形式的网络规则，并下发到相应的交换机下。我们在源代码中提供了关于Internet2骨干网的功能测试的实现，具体细节请在源代码中查看。

# 五、项目成果展示

## 5.1 系统性能表现

本节展示DPEagle在故障诊断上的性能表现。按照项目要求，将展示DPEagle在三个指标上的表现，分别是发现准确率、查全率和故障定位准确率。记网络中的所有规则所组成的集合为NR；可达规则组成的集合为RNR；可定位规则组成的集合为PNR。下面分别给出三个指标的定义。

***发现准确率***

对于每个可达规则r属于RNR，以某种方式使其发生故障，故障的发现工作将由DPEagle来完成。统计DPEagle发现故障的次数并记为FN。在发现的故障中，统计确实是故障的次数并记为TFN，则发现准确率的定义如下：

***查全率***

查全率主要关注网络中有多少的规则能够被DPEagle测试到。由于已经知道可达规则集RNR和网络规则集NR，所以查全率的定义如下：

***故障定位准确率***

对于每个可达规则r属于RNR，以某种方式使其发生故障，记DPEagle可精确定位到（没有假阳性的其他规则）的规则所组成的集合为LNR，则故障定位准确率的定义如下：

有了定义之后，接下来就要通过实验来反映DPEagle在三个指标上的具体表现。下面对实验的相关细节展开阐述。

本次实验使用的网络数据集为Internet2，其版本由ATPG的开源代码提供。在Internet2中，一共有9个Juniper路由器，经HSA解析后的网络规则总数为77,841。实验将在开放了Internet2的所有边缘端口的基础上进行。DPEagle所生成的端到端全对可达表共有3,950,929个表项。在该表下，经统计，共有71,692个可达规则，因此，在该数据集上的查全率为：

其中“100-e”表示开放所有边缘端口。

在发现准确率和故障定位准确率的定义中可以看到，需要使每个可达规则产生某种形式的故障，并利用DPEagle对故障进行发现和定位。两个准确率均是建立在1-故障的基础之上，相应地，对DPEagle的衡量也是建立在故障1-识别的基础之上。

我们是这样开展相关实验的，对于每个可达规则，我们假设其故障，并假设经过它的测试包均无法被正确接收到，这样我们可以仅在代码层面来完成模拟故障的发生。具体地，DPEagle在Internet2上总共可以选择出59,546个定位测试包，对于每个可达规则r属于RNR，找到经过r的所有定位测试包，并认定它们无法被正确接收，这样，它们就可以作为诊断分析模块的输入，以完成对于故障的发现和定位。

在前几节的阐述中可以知道，DPEagle是通过定位模式完成故障的发现和定位的。在1-故障的前提下，DPEagle所具有的发现准确率为：

发现准确率为100%的原因为，在每个故障的可达规则下，1-故障的前提使得产生的定位测试包错误模式均能够通过定位模式映射到该可达规则所在的规则集合之上，尽管有些时候，规则集合中会存在其他的一些可达规则，产生假阳性，但是由于出现错误的可达规则一定会被发现，所以准确率为100%。

由于本次实验考查的是1-故障定位准确率，所以，经过计算，故障定位准确率为：

故障定位准确率为100%的原因为，在该数据集上，DPEagle采用的定位测试包选择算法刚好能够使得所有的可达规则具有唯一的定位模式，不存在两个或两个以上的可达规则拥有同一个定位模式的情况，因此，在1-故障的前提下，其故障定位准确率为100%。