**Scalpel：使用串行程序分析诊断分布式路由配置**

**所在赛道与赛项：A**

目录

[一、目标问题与意义价值 1](#_Toc138206365)

[1.1 应用领域与关注问题 1](#_Toc138206366)

[1.2 实现目标与基本功能 1](#_Toc138206367)

[1.3 理论意义与应用价值 1](#_Toc138206368)

[二、设计思路与方案 2](#_Toc138206369)

[2.1 设计思路 2](#_Toc138206370)

[2.2 技术路线 2](#_Toc138206371)

[2.3 详细设计方案 3](#_Toc138206372)

[三、方案实现 4](#_Toc138206373)

[3.1 工作流程 4](#_Toc138206374)

[3.2 关键技术实现方案 6](#_Toc138206375)

[3.2.1 捕获因果关系生成错误路径 6](#_Toc138206376)

[3.2.2 配置错误诊断 8](#_Toc138206377)

[3.2.3 配置错误修复 10](#_Toc138206378)

[四、运行结果与分析 11](#_Toc138206379)

[4.1 实验设计 11](#_Toc138206380)

[4.2 实验准备 11](#_Toc138206381)

[4.3 捕获因果关系的时间开销 11](#_Toc138206382)

[4.4 错误配置诊断结果 12](#_Toc138206383)

[五、创新与特色 14](#_Toc138206384)

# 一、目标问题与意义价值

## 1.1 应用领域与关注问题

随着大规模计算机网络的不断发展，网络规模的迅速增长以及网络设备的多样化与配置复杂化使得网络管理者无法对网络行为进行推导或预测，导致网络故障的发生频率越来越高，排除错误难度越来越大。问卷调查显示，网络管理员认为网络规模与网络设备配置错误是网络故障发生的最常见原因。同时，使用传统的网络故障诊断工具（如ping，traceroute等）面对大规模复杂网络中的错误变得捉襟见肘，进行诊断需要大量的重复试错与经验推导，导致效率低下。现有的网络配置验证工具对于配置文件是否出错只能给出是或否的结论。如果配置文件不正确，仍然需要操作员手动查找配置的错误部分并手动修复它们，这既耗时又容易出错。基于此，Scalpel针对如何降低网络设备故障频率，实现快速、高效的网络设备故障排除问题设计解决方案。

## 1.2 实现目标与基本功能

虽然现有的网络配置面验证技术已经日趋成熟，但对于配置错误的诊断和修复很大程度上仍需用户手动完成。本方案针对网络运行中常见的故障检测和修复需求，研究并设计可适用于传统分布式网络的快速高效网络故障诊断与修复技术，提高网络可靠性，实现验证-诊断-修复的网络配置全周期自动管理，构建智能的自动化网络管理系统。

具体来说，Scalpel可以实现当验证出某些网络行为不满足用户需求时，利用配置面验证工具捕获的路由因果关系信息，诊断得到相应的配置错误（错配或漏配），并给出具体的修复方案。系统在实际应用中能够表现良好：能在数秒或数分钟内完成构型诊断且具有较高的准确率；对于单协议场景（路径向量协议（例如BGP）和链路状态协议（例如OSPF））以及多协议融合场景的典型错误都能实现正确诊断；向网络管理员提供具有较强解释性的错误原因，并提供对应的修复指导；在修复配置过程中更改的配置行数尽量少，实现最小化修复；保证修复后的网络满足预设的网络要求，即不会在修复的过程中引发其他错误。

## 1.3 理论意义与应用价值

在国家发展新基建，建设大规模网络的大背景下，保证网络稳定性、可靠性和快速修复性具有重要意义。相比当前业界已有的网络配置面验证技术，Scalpel能够提供错误配置检测、诊断和修复的全周期流程，填补了检错和纠错之间的巨大空缺。同时还具有错误定位精确度高、解释性强、修改范围小等优势，能够有效节约网络配置变更成本，精准快速修复网络故障，降低路由交换设备的维护难度。

据我们所知，目前暂时还没有集验证、诊断、修复于一体的相关的工作，也并没有一个可供网络管理员使用的集验证、诊断、修复于一体的工具。实现一个全周期的网络验证工具，构建智能的自动化网络管理系统，具有开创性的意义。

在系统实现过程中充分考虑了工业生产场景中的典型错误，并进行了针对性的优化，具有较强的产业使用价值。为减轻网络管理人员的工作压力、保证工业生产环境中网络的可靠性和应对故障的快速修复性提供了强有力的支撑。

Scalpel具有模块化的特点，操作人员可以按照自己的需求选择性调用；除此之外，Scalpel还具有很强的兼容性，能够灵活地与不同公司的验证工具相结合。整体而言，具有很大的市场潜力。

# 二、设计思路与方案

## 2.1 设计思路

Scalpel在网络配置验证工具的基础上实现适用于传统分布式网络的轻量级的、操作友好的网络配置诊断与修复工具，从而提高网络的可靠性。

其设计的核心思路是：通过捕获分布式路由计算之间的因果关系，我们可以将其转换为一个单一的串行执行序列，从而使用顺序程序分析实现有效的配置错误诊断和修复。具体来说，Scalpel包括(1)一个错误序列生成器，它在基于仿真的验证过程中生成分布式路由计算的串行执行序列;(2)配置诊断器，给出分布式路由器计算的串行执行序列及错误属性，找到合理的、最小的错误解释，并定位与路由器配置相关的部分;(3)一个配置修复程序，基于模板计算错误属性的正确值来修复发现的配置错误。

为了实现配置错误的精确定位，Scalpel需要对获得的串行执行路径进行错误定位分析。为此，我们通过控制流敏感的程序分析技术找出整个路径上与最终错误输出有关的程序段，去除无关程序段，从而精确定位错误。并通过提取错误执行序列的中间状态来对错误进行解释，从而方便网络管理人员理解配置出错的原因。

## 2.2 技术路线

系统工作流程如图1所示：

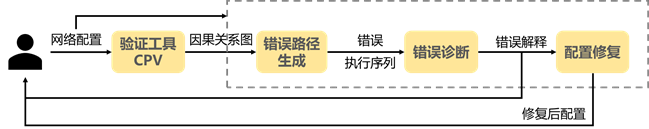


图1：Scalpel系统工作流程图

系统整体包含三个主要部分：错误路径生成模块，错误诊断模块、配置修复模块。在基于仿真技术的配置验证工具输出路由计算间因果关系的基础上，实现自动诊断和修复。具体的技术路线如下：

1. 网络因果关系图精确捕获。

我们通过捕获路由间计算的因果关系来处理网络间复杂的依赖关系，将分布式路由计算转换成因果关系图。

1. 配置错误精确定位。

我们对获得的串行执行路径进行错误定位分析，找出整个路径上和最终错误输出相关的程序段，并去除无关程序段。我们的主要方法是通过控制流敏感的程序分析技术来计算对程序进行切片从而定位错误，并提取错误程序执行的中间状态来给出错误解释。

1. 进行最小化修复。

由于配置的复杂和多样性，很难考虑到所有的解空间（缺失的配置），我们设计利用模板来做合成修复。将待修复的配置变量和网络（包括错误流和正确流）建模成一个约束求解问题。

## 2.3 详细设计方案

1. 生成因果关系图

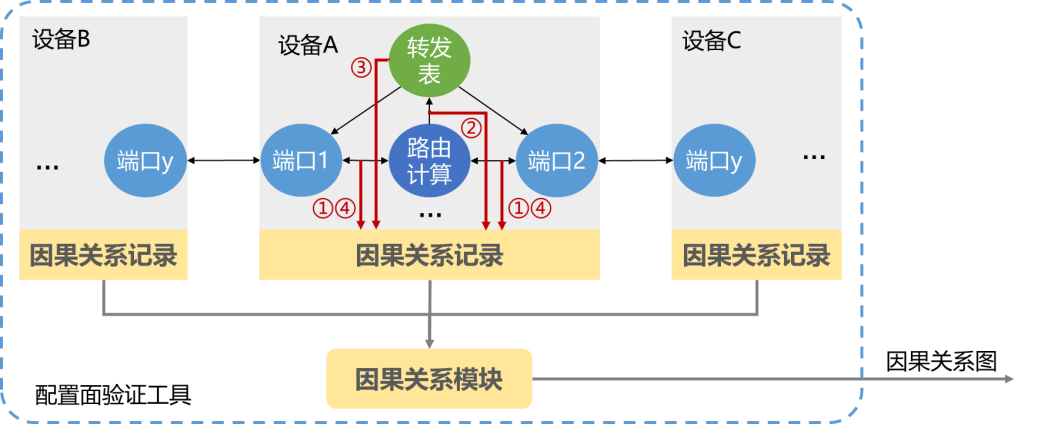


图2：Scalpel因果关系模块

如何捕获不同设备之间的因果关系是诊断步骤中的关键问题。如图2所示，我们通过修改仿真工具（例如Batfish）仿真主循环代码，记录相关信息，捕获不同设备路由计算过程中的因果关系；然后根据路由的收发行为和因果关系子图获得网络完整的因果关系图。每个网络流的计算都是一个有向无环图，表示分布式路由过程的依赖关系。设备本地按关键事件发生的顺序记录，构建因果关系子图。所有子图以事件或行为作为关键字进行结合，获得完整因果关系图。其顶点表示单个网络节点的路由计算，边表示不同路由计算之间的因果关系。给定网络中所有进程的路由计算序列，就可以得到一个特定前缀的因果关系图。

1. 分布式路由计算转化为串行程序

由等价性原理可知，任何满足路由计算因果关系图的串行计算路径，其最终状态与分布式路由计算生成的最终状态相等，所以只要捕获路由计算的因果关系图，即可生成一条满足该图拓扑序的线性串行计算路径。

1. 错误诊断

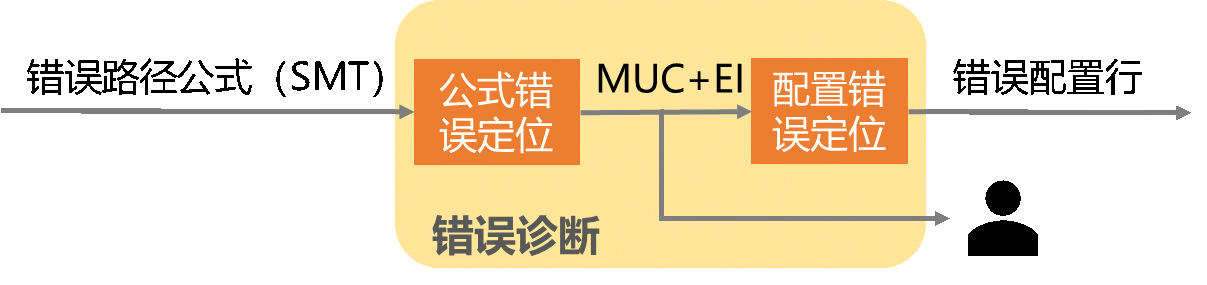


图3：Scalpel错误诊断模块

传统计算路径分析方法或错误定位不精确，或无法对控制流进行分析。我们对错误相关程序段和错误解释进行了形式化定义：Minimal Unsatisfiable Core（MUC，最小不可满足核）和 Error Invariants（EI，错误不变量）。

如图3所示，我们设计错误诊断模块的总体思路为：将串行程序计算路径编码成SMT公式，使用Z3求解器计算其MUC，得到最小错误语句集合。通过差值算法计算错误语句对应的EI，生成错误解释。根据错误类型和关键词映射精确定位配置行错误。

MUC计算最小错误语句：基于错误路径公式，通过计算公式的最小不可满足核定位出最小的错误语句集合，这是整个路径上和最终错误输出相关的程序段。

EI生成错误解释：对该错误集合中的每一条语句计算错误不变量：导致错误产生的每一个关键控制条件和相关变量状态。所有的错误不变量解释了在对应路径下错误被计算的所有关键步骤。

1. 配置文件最小化修复（模板修复）

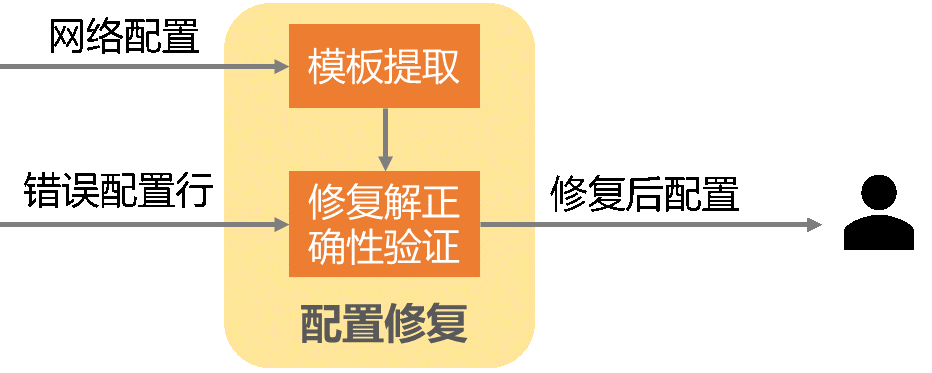


图4：Scalpel配置修复模块

配置修复模块的结构如图4所示。Scalpel使用基于模板的配置综合技术，允许输入配置包含一些符号参数，然后使用SMT求解器自动分配满足要求的具体值。具体做法为：对于可跟踪的错误，Scalpel根据MUC和EI输出的结果对计算路径的错误语句中的配置相关变量做映射，找到错误语句中来自配置的常量，将其配置值设成符号变量（未知值）。将程序执行过程重新编码成带约束的SMT公式，通过求解公式的可满足赋值，得到修复后的配置。

# 三、方案实现

## 3.1 工作流程

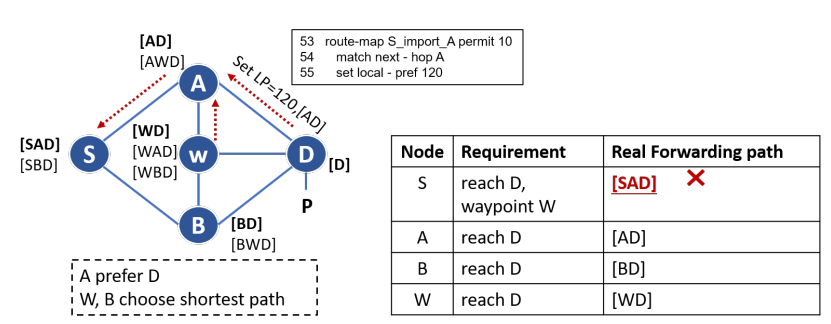


图5：示例拓扑与网络预设要求

我们使用图5中的示例来演示Scalpel的基本工作流程。网络中有5台路由器，它们之间使用eBGP连接。路由器D中有一个目的前缀p，并且网络规定了一系列需要满足的转发要求: A、B、W要到达E, S要通过D到达E，验证结果显示S选择的路径为[S, A, D]，不符合要求。

仅根据验证工具的反馈很难确定是哪部分配置导致了错误的转发路径。结果表明，路由器S不符合要求。但是，只修复S的配置并不能修复错误。因此，Scalpel将路由器的分布式执行转换为集中式的串行执行序列，以诊断配置错误并修复它们。

**生成错误执行序列**：Scalpel捕获网络路由计算过程中不同路由器上路由进程之间的因果关系。每台路由器在本地进行路由计算，计算的结果可以作为其他路由器的输入，从而触发新的路由计算。我们通过跟踪输入和输出之间的关系来恢复不同路由计算之间的因果关系。

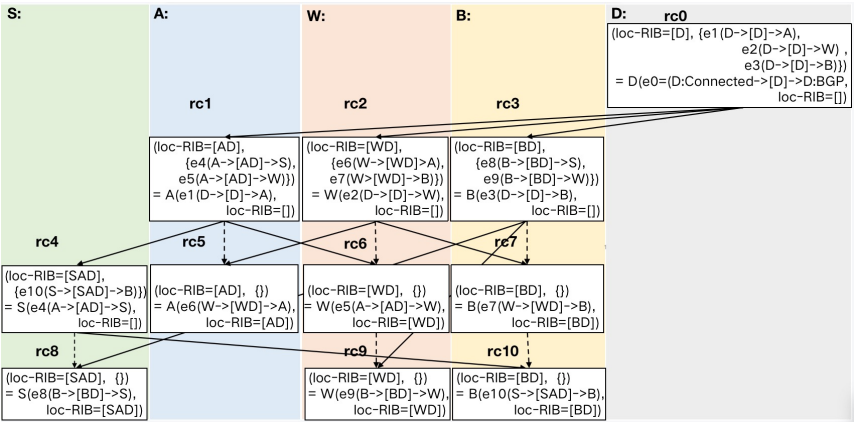


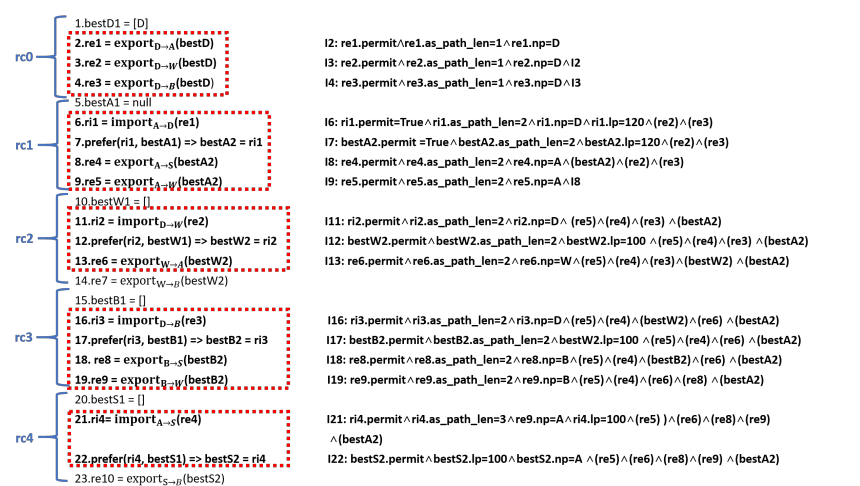
图6：示例网络因果关系图

图6为前缀p的因果关系图，有向图的顶点为一次路由计算，边表示不同路由计算之间的因果关系。在本例中，我们可以看到路由计算从D开始:前缀p源发于D, D将路由[D]发送给所有邻居A、W和B(𝑟𝑐0)。然后，A接收到D传来的路由[D]，触发其本地的路由计算，接着A更新其路由信息库(RIB)，并将路由[A, D]发送给邻居S和W(𝑟𝑐1)。以此类推，直到没有任何路由更新，网络达到收敛状态。

然后使用有向图的拓扑排序，计算一条图遍历路径作为串行执行序列，并将每个路由计算用形式化语句表示，将其转化为错误序列。一个路由计算节点对应多个语句，所有路由计算的语句和网络要求语句的一起构成网络的错误执行序列。

**诊断错误执行序列定位并解释错误**：Scalpel使用**最小不可满足核(MUC)**和**插值法**分析错误序列来给出最小且合理的错误解释。

给定错误序列，我们将其编码为一组不可满足的SMT公式(每个语句都可以视为一个公式)。MUC是不可满足的语句的最小集合。插值作为一种错误解释，它是一组表示错误序列中间状态的标签。对于每个MUC，我们通过计算MUC中每个语句的插值序列来标记最终会导致错误的关键变量。如图7所示，MUC为红色虚线框中的内容，右边的语句解释导致断言(即需求) 被违反的原因。



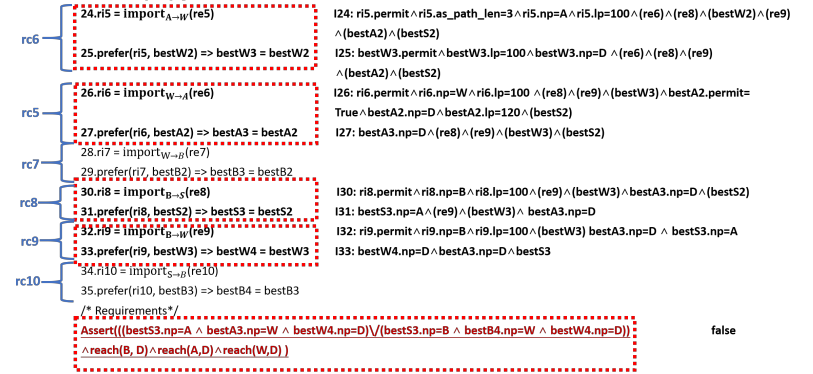


图7：图6因果关系图对应的错误执行序列

最后，我们通过收集来自配置的变量，并将这些变量映射到配置行。这些配置行就是可能导致要求违法的错误位置。至此，Scalpel完成了配置的错误定位。

**修复配置错误**：Scalpel采用基于模板的方法进行错误修复。该方法采用一些配置模板作为输入，并使用SMT工具生成满足网络要求的变量值。例如，在诊断阶段，我们将配置错误定位在A的配置的第53行和第55行。因此，我们将这两行中的值表示为变量，例如，使用模板“set local-pref ?”来替换第55行。最后，通过求解带约束的SMT公式计算出满足要求的变量值90，将其写回配置，并将修复后的配置返回给用户。

## 3.2 关键技术实现方案

### 3.2.1 捕获因果关系生成错误路径

我们首先将路由过程形式化地抽象为事件驱动函数，然后在路由计算过程中捕获路由的因果关系，构建网络的因果关系图(CRG)。然后，我们根据CRG生成错误执行序列，并对其进行SMT编码，以便错误诊断。

* + - 1. 路由过程抽象

不同设备上的路由进程通过特定的协议(如BGP、OSPF)交换路由信息(如BGP UPDATE消息、OSPF链路状态宣告)，同一设备上的路由进程也可以通过路由再分发来相互交换路由信息。

我们将路由过程抽象为一个由配置控制的事件驱动函数来计算路由。它接收到一个事件𝑒触发,使用𝑒和当前状态𝑠作为路由计算的输入,将新状态和新事件𝑒𝑠作为输出。我们省略了过程符号，并使用来表示函数。然后，我们使用以下格式来表示路由过程的计算:

**事件𝑒**：定义为，表示路由信息𝑟从源设备𝑑𝑒𝑣发送到目的设备𝑑𝑒𝑣'。有两种类型的事件:(1)路由重新分配在同一设备(例如，静态/直连路由被重新分配到BGP/OSPF进程);(2)不同设备间的路由通告(如BGP UPDATE消息、OSPF链路状态通告)。

**状态：**状态𝑠定义为local-RIB=[·]。路由器内部维护的所有路由信息构成了路由器的状态，路由器的路由计算依赖于路由器的状态。具体来说，不同的协议维护不同类型的路由信息库(RIB)。这里我们考虑两种协议:(1)链路状态协议(如OSPF);(2)路径矢量协议(如BGP)。它们都有自己的本地RIB来存储最终的路由选择。我们把不同时期的RIB表称为路由器的状态。

**路由计算**：我们定义了一个路由计算作为函数调用。每当一个事件发生时，将有一个相应的路由计算，该路由计算可能导致本地路由器上的状态更新和其他路由器的新事件。我们把相应的路由计算称为行为。有三种类型的行为:(1)状态更新但无新事件，(2)状态更新且有新事件，(3)无状态更新且无新事件。对于情形(3)，表示路由计算会丢弃该路由，不再继续传播。因此，我们省略了对此类行为的处理。

* + - 1. 构建因果关系图

因果关系图(CRG)是一种表示分布式路由进程之间依赖关系的有向无环图。CRG中的顶点是一次路由计算，边表示不同路由计算之间的因果关系。以图5为例，路由计算𝑟𝑐6有两个原因:𝑟𝑐1和𝑟𝑐2。边𝑟𝑐1→𝑟𝑐6表示𝑟𝑐6是由𝑟𝑐1中的事件𝑒5触发的，边𝑟𝑐2→𝑟𝑐6表示状态loc-RIB=[WD]直接继承自𝑟𝑐2。

从CRG中我们可以得到两类因果关系:(1)事件因果关系(图5中的实线):它显示了事件如何传播及其影响范围;(2)状态因果关系(图5虚线):表示一个设备路由过程的状态转换。基于这两种关系，我们可以使用事件因果关系来跟踪事件(例如，𝑒10 - 𝑒4 - 𝑒1 - 𝑒0是事件的轨迹𝑒10)或观察事件如何影响网络;使用状态因果关系可以找出某个设备状态的转变。

CRG显式地显示了每个路由器及其依赖关系上的状态转换。但是， CRG包含的信息远远多于错误信息。因此我们从CRG生成串行执行序列，以实现更细粒度的诊断。

* + - 1. 生成错误执行序列

我们通过拓扑序遍历CRG将分布式网络计算转换为串行执行序列。该序列可以编码为带有SMT约束的执行序列。一个序列是一个元组(𝜋, 𝜙) ，π表示网络的执行顺序，𝜙描述序列期望输出状态的状态公式。序列(𝜋, 𝜙)的执行是序列公式𝑇F(π)=𝑇1∧…∧𝑇𝑛，其中𝑛是序列的长度。如果𝑇F(π)∧𝜙是不可满足的，则该序列称为错误序列。

我们对路由计算全过程——接收路由、路由更新、发送路由分别进行SMT建模，以生成路由计算的SMT约束公式。接下来介绍具体的思路和实现方式。

首先对路由计算的入事件进行处理。本文将入事件进一步形式化为：

从而突出接收端对路由信息的处理。其中表示处理后的路由信息。import函数是实现将接收路由的动作转化为SMT公式的关键，其主要任务是对接收到的路由中需要进行变更的路由属性进行更新，然后形成约束条件。

然后对路由更新进行处理。路由更新包括路由的新增、撤销以及路由选择更新。这里重点考虑路由选择的更新，形式化表示为：

其中best为设备在更新前选择的路由，表示更新后选择的路由；prefer函数根据路由协议的规则判断两条路由哪条更优。整个表达式为蕴含逻辑，即，若新接收到的路由更优，则更新路由选择；否则路由选择保持不变。根据判断的结果，添加保证所选路由优于另一条路由需要满足的约束条件。

最后对路由计算的出事件进行处理。与入事件相对应，我们将出事件形式化表示为：

其中re表示要向外发送的路由信息，best为当前该设备选择的路由。export函数的处理较为简单，其主要任务就是添加发送路由的各属性值等于best路由各属性值的约束条件。

因为判断网络发生错误，一定是网络实际的某些行为违反了网络预设的要求，所以为了确定错误路径中违反网络要求的具体内容，除了将串行执行序列转化为SMT约束外，还需要添加网络正确性要求的约束条件。将串行执行序列的SMT约束与网络正确性要求的SMT约束合取，得到不可满足的SMT公式。具体来说，网络预设要求主要分为可达性要求和路径要求两类。可达性要求，即要求保证网络中源节点到目标节点的路由可达；路径要求，即要求源节点到达目的节点的路由必须经过某些特定节点。

### 3.2.2 配置错误诊断

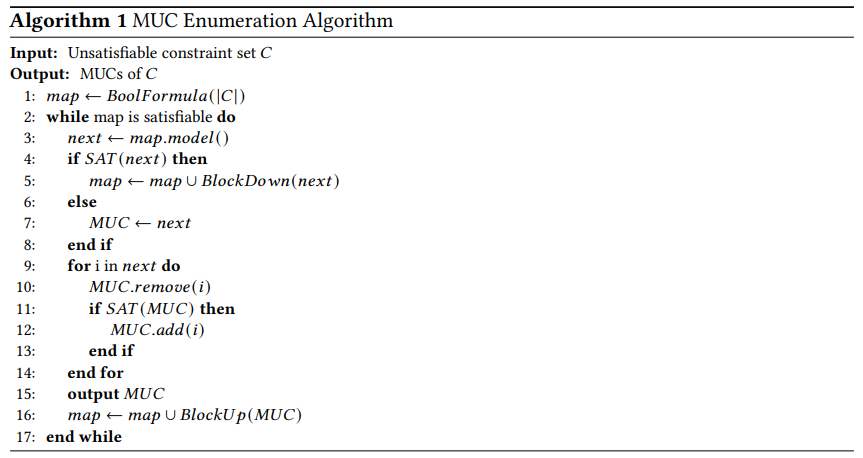
给定一个错误序列Φ，我们使用最小不可满足核(MUC)和插值来诊断错误序列。MUC是一组证明序列不可满足的最小不满足语句集合，其插值是一组表示可能导致网络状态错误的公式。我们将错误序列减少到导致错误的最小语句集，并为它们计算插值，以提供最小且合理的错误解释。

1. 基于MUC枚举算法MARCO的改进算法来计算MUC

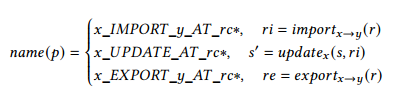
给定一个不可满足的约束集，最小不可满足核(MUC)是其一个子集，它的合取是不可满足的，并且MUC的每个真子集都是可满足的。

我们将配置错误定义为需要修改/删除/添加的错误配置行。前两种类型的错误可以在一些错误执行的语句中观察到，这些语句由错误配置控制，并且在MUC中。最后一种类型的错误不能直接从错误序列中诊断出来，对于这种错误，我们使用启发式的方式修复它们。如果有多个配置错误可能导致违规，那么将有多个MUCs。每个MUC解释需求违反的一个原因。

我们基于MARCO算法计算MUC，并通过按顺序缩小不可满足核，迭代地添加最小不可满足子集并阻塞可满足子集来改进算法。该算法的输入是一个不可满足的约束集，每个约束变量变量都被赋值为一个布尔变量。我们在映射求解器𝑚𝑎𝑝中添加了不受约束的bool变量，使得𝑚𝑎𝑝的可满足赋值是一个未搜索的子集(第1行)。一个未搜索的子集意味着它的可满足性是未知的，我们不断搜索不可满足的子集来计算MUC，直到所有的赋值都被搜索(即𝑚𝑎𝑝是不可满足的)。



接着，将MUC映射到执行序列。我们使用一个布尔常数𝑝来跟踪执行序列子句，这样我们就可以将它映射到初始序列语句。我们布尔常数命名为𝑝，𝑟𝑐\*是语句所涉及的路由计算:



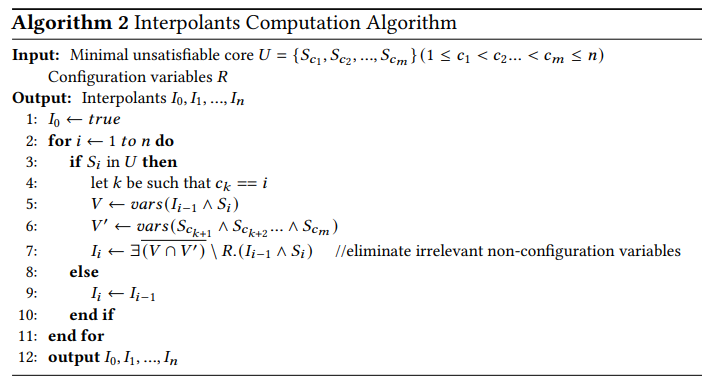
SMT公式编码为:



2. 计算MUC特定变量插值的混合算法

我们使用插值和MUC生成错误序列的错误解释。具体来说，我们利用MUC对错误序列进行切片，并基于混合算法对切片计算插值。错误执行序列中某个位置的插值是执行变量的公式，它捕获如果从该位置继续执行，会产生错误的状态。插值序列显示了导致错误的中间状态抽象。

给定执行序列和最终状态𝜙,一个错误标记𝐿为一个失败的路径由标签组成，形成一个插值序列的方程式。



我们使用𝑇𝐹(𝜋[𝑖, 𝑗])来表示子序列∧……∧。，最终状态Φ在位置i的一个插值公式需要具备以下属性:

(1) 𝐼0 = 𝑡𝑟𝑢𝑒

(2) 𝑇𝐹 (𝜋 [1, 𝑖]) ⇒ 𝐼𝑖

(3) 𝐼𝑖 ∧𝑇𝐹 (𝜋 [𝑖 + 1]) ⇒ 𝐼𝑖 + 1

(4) 𝐼𝑖 ∧𝑇𝐹 (𝜋 [𝑖, 𝑛]) ∧ 𝜙 ⇒ *false*

因为基于MUC和基于插值的方法都可以获得执行路径的一部分。此外，插值序列可以提供中间状态抽象与切片。为了高效地计算最小且合理的归纳插值序列，我们利用由MUC枚举算法生成的MUC来计算基于混合算法的插值，并保留变量的依赖性。

算法2展示如何使用MUC和插值器生成错误解释。输入为MUC𝑈={𝑆𝑐1，𝑆𝑐2，…，𝑆𝑐𝑚}和配置变量𝑅。如归纳插值属性(1)所示，第一个插值总是初始化为𝑡𝑢𝑟𝑒(第1行)。

然后，对于每个语句 (如果不在MUC中)，我们将前面的插值赋值给 (第9行)。

否则，我们计算:(1)在前面的表达式和当前的表达式 (第5行)中出现的变量集合。（2）变量集合出现在𝑆的MUC中（第6行）。集合V和的交集即为同时出现在和中的变量，因此我们需要在中保留。最后，我们消除了它的补集中的变量，但保留了与配置相关的变量，以跟踪由配置控制并导致执行错误的变量赋值(第7行)。

3. 将诊断结果映射到配置行

MUC的插值序列作为错误序列的解释性标签。每个插值都是一个公式，它捕获了会产生误差的中间状态。由于执行序列是一系列路由计算，所以插值序列可以解释为什么会产生错误的路由，以及哪些配置变量导致了错误的路由。图7右侧显示了违反网络路径要求的错误解释。

我们过滤了插值序列中的不是来自配置的变量，并保留了指示配置行的配置相关变量作为待修复的错误。

### 3.2.3 配置错误修复

我们利用基于模板的增量合成器NetComplete来修复错误配置。NetComplete用SMT约束对配置和需求的语义建模，然后使用SMT求解器得到满足需求的变量值代替符号参数。模板的使用允许我们控制初始配置中需要更改的参数。

由于NetComplete生成的修复解决方案只会修改配置行，而不会通过删除其中的一些来修复错误，因此我们通过向存在符号参数的路由策略中的每个语句添加布尔变量来扩展约束建模过程。该布尔变量为true表明对应的语句应该存在;否则，应将其删除。

因为我们已经在诊断部分确定了错误的配置，所以直接的修复方案是将这些错误配置更改为符号参数，并使用基于模板的修复方法自动为它们分配正确的值。但是，SMT求解器可能无法使用一种模板找到修复解决方案。因此，我们提出了三种不同的模板生成方案。首先，我们通过直接将每个错误配置更改为符号参数来生成模板。其次，我们通过学习每个路由器的配置来生成多个模板，并将它们从最低复杂度到最高复杂度排序。最后，我们通过向每个路由器添加静态路由来生成模板。修复配置时依次使用以上三种模板方案，即若前一种模板方案找不到解，则按序尝试下一种模板方案。不同模板方案的基本思想是通过增大搜索空间来提高找到修复解的概率。尽管并不实际，但静态路由可以确保找到一个修复的解决方案。

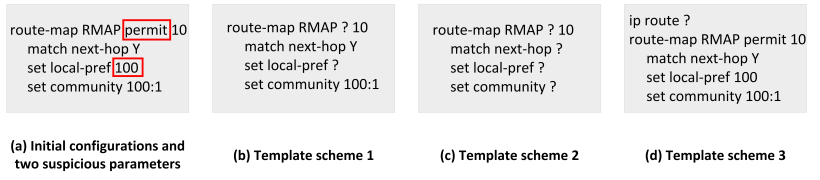


图8：修复模板示例

图8显示了上面提到的三种模板方案的一个简单示例。在这个例子中，我们在初始配置中本地化了两个错误的参数，如图a所示。如图b所示的第一个模板方案是通过将两个错误的配置更改为符号参数(我们使用‘？’来表示这里的符号参数)。图c所示的第二个模板方案是通过学习初始配置的结构并对每个可变参数进行符号化来生成的。为简单起见，本例中只有一个路由器的配置。由于网络中有多个路由器，我们将从不同的路由器学习多个模板。然后，我们将每个模板按照复杂程度的顺序逐个应用到所有路由器上，进行配置修复。如图d所示的第三个模板方案是通过在初始配置中添加静态路由，然后使用配置修复技术完成符号参数生成的。

# 四、运行结果与分析

## 4.1 实验设计

我们对Scalpel的性能从以下三个维度进行了评估:(1)在捕获因果关系时，Scalpel的开销； (2)Scalpel在配置诊断中的表现；(3)Scalpel在配置修复中的性能。Scalpel原型由Java和Python中的5K LoC组成，并使用Z3作为底层SMT求解器。

## 4.2 实验准备

为了证明Scalpel的合理性和正确性，我们在Topology Zoo中的小型真实拓扑上进行了实验。然后，我们将拓扑扩展到更大的尺寸，以便在Scalpel上运行压力测试。

**数据集**：我们使用了10个从0(10)到0(100)个节点的合成网络作为数据集。对于每种拓扑，我们使用NetComplete生成满足网络预设要求的BGP配置，合成配置行数从3,000到20,000行不等。所有配置都是用Cisco的IOS语言编写的。

**网络预设要求**：由于我们已经分配了一些路径需求对来合成初始配置，我们只需使用Scalpel来验证这些路径需求。每个网络要求是从随机选择的源节点到目的节点的随机可行路径，由源节点、目的节点和报文转发路径组成。

**注入配置错误：**对于错误配置，因为通常实际网络配置中的错误是不被公开的，所以无法获得真实的错误配置。为了顺利完成实验，我们采取为正确配置手动注入错误的方式得到错误配置。具体来说，我们在配置中随机添加、删除或修改一些配置行(例如，改变匹配条件，或者拒绝引入路由)。然后，我们使用Batfish模拟控制平面，并检出源节点的RIB表。如果到达目的地的RIB表项与源节点中的转发需求不对应，则认为已经成功注入错误。同时，我们将更改的配置行记录为错误。在压力测试中，我们通过修改拓扑的源路径要求，使原始配置与网络预设要求相违背。换言之，我们选择从源到目的地的**另一条**数据包转发路径作为网络要求，并将原始合成的配置作为当前需求的错误配置。

**实验环境：**所有的实验都是在Linux服务器上进行的，内核版本为5.4.0，两个Intel Xeon Silver 4210R 2.40GHz 的CPU, 13.75MB L3缓存和128GB DDR4 DRAM。考虑到计算时间，在实验中，我们最多只枚举10个MUCs。

## 4.3 捕获因果关系的时间开销

我们用验证过程中记录和存储因果关系的时间表征捕获因果关系的时间。如图9左侧所示，50%的网络因果关系可以在600ms内完成捕获。对于实验的所有网络，捕获因果关系的时间都小于800ms。开销随着拓扑大小的增加呈线性增长。这主要是因为具有较大规模的拓扑总是具有较长的迭代过程。

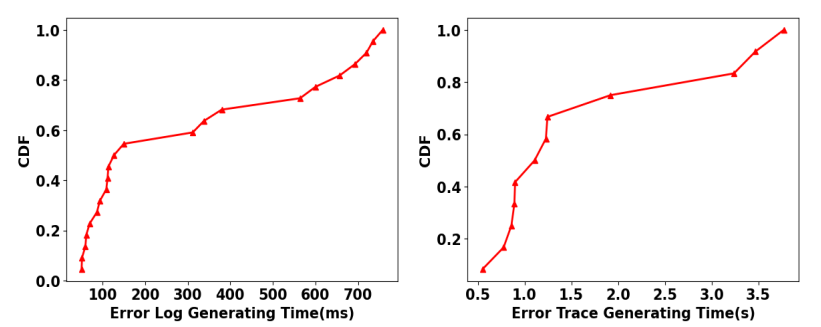


图9：生成因果关系和错误路径的CDF图

图9的右侧给出了生成错误路径的CDF。这个时间包括构造因果关系图的时间、编码错误路径的时间、Z3校验的时间。在80%的场景中，Scalpel生成错误路径的时间≤2.5𝑠。综上所述，以上两个过程在Scalpel的整个过程中占用的时间很少。

## 4.4 错误配置诊断结果

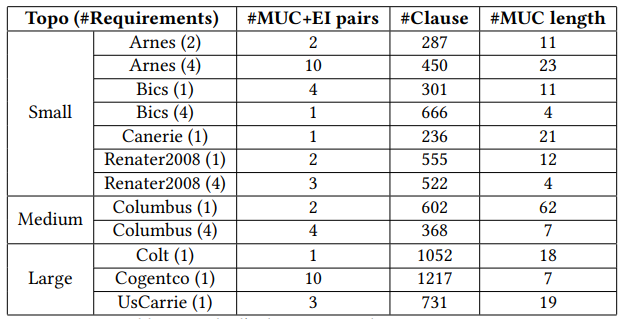


表1：Scalpel诊断结果

我们用每个网络产生错误解释的时间和Scalpel给出的错误解释的个数来评估Scalpel诊断的性能。我们捕获了Scalpel在不同拓扑上的诊断结果，如表1所示。对于数据集中的每个拓扑，平均MUC长度明显短于所有子句的长度，这有效地缩小了修复的范围。我们注意到在单个错误路径中总是存在一对以上的MUC和EI。这是因为错误可能有不止一种解释。为了修复错误，我们必须考虑EI中出现的所有配置作为修复候选。图10给出了Scalpel计算单个MUC、EI以及每个拓扑中所有MUC和EI所花费的时间。对于90%的单个MUC，它们可以在≤10𝑠内计算，最大计算时间≤25𝑠。对于80%的拓扑, MUC计算过程可以在≤30𝑠内完成。因此计算每个拓扑的所有MUCs可以在≤50𝑠中完成。

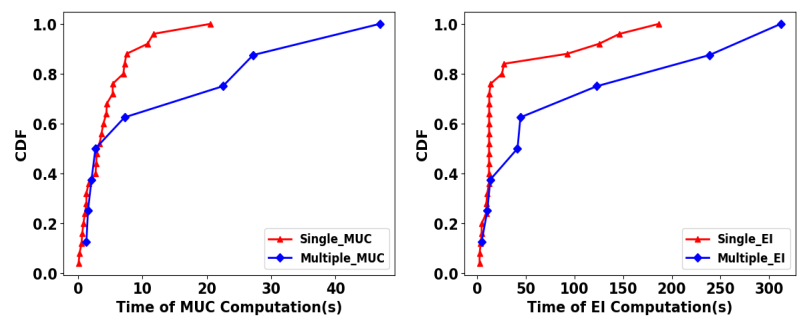


图10：计算MUC和EI的时间

计算EI是诊断过程中最耗时的部分。在80%的情况下，计算一个EI需要≤30𝑠。对于60%的拓扑，所有EI可以在≤50𝑠范围内计算，最高可达≤350𝑠。我们注意到，计算一个EI可能需要近200s的时间，这主要是因为EI的计算时间与MUC的长度和包含的变量密切相关。

4.5 端对端修复表现

对于实验的7个网络拓扑，Scalpel进行了端到端诊断修复配置和配置修改的修复性能评估。我们对修复精度的定义如下:如果被修复的配置符合要求，我们说修复过程的精度为100%。对于所有7个网络及其要求，Scalpel 100%修复了错误的配置。

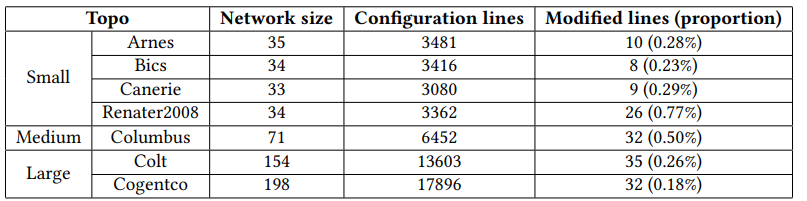


表2：不同拓扑修复配置行数和占比

**修复效率**：我们在图11中列出了记录、诊断和修复错误配置的时间。从图中可以看出，整个时间从十秒到几分钟不等，对于本次实验的所有网络，均可在6分钟内修复。最耗时的部分是诊断错误。根据诊断结果，将再合成的搜索空间缩小到一小部分配置行。因此，我们可以在不引起重大修改的情况下修复配置。

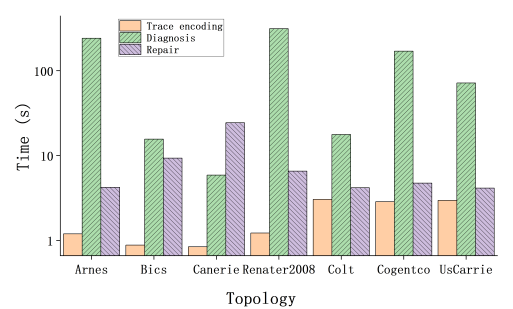


图11：Scalpel端对端修复效率

修复后拓扑上配置行修改的百分比如表2所示。在小型网络中，即使修复了所有的需求，配置的变更行也小于0.77%。更大的拓扑虽然有更多的配置行，但修改配置行的数量仍然很小，不到0.50%。结果表明，用Scalpel进行修复对原有结构的影响很小。

# 五、创新与特色

基于以上介绍，我们提出的Scalpel系统的创新点主要体现为：

1. Scalpel将分布式路由协议的错误定位与修复问题转化为在软件工程领域中被广泛深入研究的**集中式串行程序**的错误定位和修复问题，填补了当前**自动网络配置管理全周期**的空白。

Scalpel通过形式化方法实现对分布式路由配置的有效诊断和修复；将配置出错的位置以及导致网络错误的诊断结论集成到修复过程中，从而能够对运营商未来的配置管理提供指导，降低本系统的部署门槛。

1. Scalpel设计了**可靠、最小化**故障定位与解释算法和**最小化修复**求解器。

网络规模和需求的增长导致网络配置异常复杂，而配置错误通常只是其中很小的一部分。为了实现在海量的配置文件中尽可能快速且准确地定位目标配置中的错误行，Scalpel设计了可靠的最小化故障定位和解释算法，通过数据流的程序分析技术来对程序进行切片从而定位错误，并提取错误程序执行的中间状态来给出错误解释。

即使准确地定位到错误行，对于相同的网络预设要求，通常都有多种配置方法可以实现。为了实现错误修复，完全推翻现有配置文件重新合成显然是不实际的。因此如何尽可能少的改动现有配置，实现正确修复是一大挑战。为此，Scalpel设计了最小化的修复求解器。

1. Scalpel不仅考虑**单协议场景**（路径向量协议（例如BGP）和链路状态协议（例如OSPF）），对**多协议融合场景**的典型错误也设计了相应的诊断方案。

协议间依赖（例如：不同网络协议间可以通过路由重分发进行路由计算）和协议内依赖（例如：不同设备间通过运行同一种协议来协同计算转发路径）是网络配置诊断工作中面临的一大挑战，Scalpel利用模块化的思想来应对多协议融合的场景，一定程度上克服了现有工具适用范围窄的局限。

1. Scalpel具有**很好的扩展性**且**使用门槛较低**。

在真实应用场景中，网络通常具有多个目的IP。Scalpel基于数据流诊断器可以对多种路由协议产生的错误路径进行分析，从而实现针对不同目的IP子空间单独进行路径捕获、诊断和修复。

Scalpel可作为网络验证工具（例如：Batfish）的插件进行使用，**即插即用**，极大地降低了其使用门槛，在一定程度上克服了现有工具使用难度高的缺点。