大家好，我们组的决赛作品是Scalpel：使用串行程序分析诊断分布式路由配置。

随着网络规模的迅速增长，网络设备多样化，和配置的复杂化导致网络故障频发。经过调研，网络错误往往存在于网络设备配置中，因此在将设备配置部署于实际网络之前，找到可能引起网络错误的设备配置并修复就显得尤为重要。

网络是一个由众多网络设备和设备交互行为构成的复杂分布式系统。某个设备上的配置错误不仅可能会引起该设备的错误行为，往往也会造成一系列连锁反应，引起该设备前后多个设备的属性违反。例如下图s节点属性的违反是由B节点的配置造成的。

而现有的网络验证工具，仅仅能得到对于验证属性对或者错的答案，无法诊断出究竟是哪些配置行引发了验证属性的被违反。batfish只能提供了实际数据包的错误转发流。而通过右边的工具minesweeper只能验证属性的正确与否。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。至此，我们提出一个问题，在网络验证之后我们应该做什么？

针对以上问题，本方案的目标是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。我们发现目前的网络检测工具或针对验证，或针对修复，缺少一个集验证-诊断-修复于一体的网络配置全周期自动管理。实现一个全周期的网络验证工具，可以说具有开创性的意义。

Scalpel的核心思想在于：可以通过捕获不同路由进程计算间的因果关系，将分布式路由计算转换成集中式串行程序进行分析。

每个路由进程都可以抽象成一个event driven的函数：event是触发计算的事件，如接收到一条来自邻居设备的路由，基于当前进程的路由表状态，每次函数调用后都会产生一系列action，表示路由计算的结果，如丢弃路由、更新路由表、发送路由给其他设备。例如右上角的网例，4个设备节点间通过eBGP交换路由信息，每个设备上的eBGP进程都可以抽象成这样一个函数。

我们通过记录每次函数调用的输入输出来捕获路由计算间的因果关系。每次路由计算的因包括两部分：外部事件和内部状态，果为计算后产生的一系列动作，触发后续的计算。

最终，我们把所有设备上记录的因果关系日志通过输入输出匹配串联起来，构造路由计算因果关系图，这是一个有向无环图，节点表示路由计算，边表示它们的因果关系。

图的任一拓扑排序对应的最终计算结果都是等价的，从而我们可以得到分布式路由程序的串行的计算序列。而对串行程序的分析与诊断，已是软工领域被广泛研究的问题之一。

基于将分布式路由计算转化成串行程序分析这一思想，我们提出了配置自动化诊断与修复的工具：scalpel，这里主要需要解决三个挑战：一是精确捕获因果关系图，二是精确定位错误配置，三是最小化修复配置。

对于1因果关系的捕获，基本原理是按序记录每个设备上的各个路由进程的event和action的计算，由于每次计算的event都是其他计算的action，我们可以通过这个作为键值将所有分布式计算连结起来，获得完整因果关系图。

可以通过对事件队列监听，仿真代码插桩等方式实现。

对计算路径的诊断与定位，我们使用Minimal Unsatisfiable Core（MUC，最小不可满足核）和 Error Invariants（EI，错误不变量），总体思路为：将串行程序计算路径编码成SMT公式，接着基于程序执行的路径公式，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，也就是导致错误产生的每一个关键控制条件和相关变量状态。最终根据错误类型和关键词映射精确定位配置行错误。

对于实际网络的例子，我们根据给定的错误序列，编码为一组不可满足的SMT公式。对这一组合取公式求解MUC，我们可以得到黄色框出的部分。对这些公式计算插值，可以捕获了会产生误差的中间状态。其中的配置变量就是导致了产生错误路由的原因。

由错误解释，我们便可以轻松定位到设备配置上由于localpreference导致的错误。

我们通过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。

在找到导致错误违反的配置变量后，我们的修复思路就是将这些变量设为未知的符号变量并将程序执行过程重新编码成带约束的SMT公式，使用求解器求解满足约束的变量赋值。

实际中，我们利用基于模板的增量合成器NetComplete来修复错误配置，其核心原理就是符号化变量并求解。在这个例子中，我们通过对节点s的两个routemap中的local preference符号化并求解符合满足约束的数值，最终调整将var2大于var1实现修复。

我们通过大约5千行的 Python+Java代码实现了Scalpel并作为Batfish的插件，并使用了Z3作为smt求解器。我们利用topology zoo提供的10个拓扑并使用net complete生成不同规模、不同配置行数的网络作为数据集，通过在配置中随机添加、删除或修改一些配置行手动注入错误，使用scalpel对路径属性进行验证。

scalpel记录和存储因果关系的时间如下所示。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≤2.5。以上两个过程在Scalpel的整个过程中占用的时间很少。

我们捕获了Scalpel在不同拓扑上的诊断结果，如下表所示。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，这表明我们有效地缩小了修复的范围，因此提高了诊断性能。

这两张cdf图给出了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可以在10秒内计算，计算每个拓扑的所有MUCs可以在50秒内完成。

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会在30秒内完成。对于60%的拓扑，所有EI可以在50秒内完成计算。我们认为耗时的长短主要与MUC的长度和包含的变量相关。

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修复性能评估。从图中可以看出，整个端到端的时间从十秒到几分钟不等，所有网络均可在6分钟内修复。

修复后拓扑上配置行修改的百分比如表所示。在各种规模的网络中，配置行修改的比例均不到0.50%。结果表明，用Scalpel进行修复对原有结构的影响很小。

基于以上介绍，我们提出的Scalpel系统的创新点主要体现为：

Scalpel通过形式化方法实现了网络配置最小化可靠性诊断与修复，具有很好的扩展性且使用门槛较低。可作为网络验证工具（例如：Batfish）的插件进行使用。

该项目相关论文已被CCF C类国际会议APNet 2023录用。

恳请各位评委专家批评指正！