P2

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。网络错误往往存在于网络设备配置中，因此对于网络管理者来说，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且修复他们就显得尤为重要。

P3

我们可以把网络看作是一个分布式系统，当然这个分布式系统由于网络设备数量众多和网络设备配置引起的设备之间的交互，系统结构和行为必然非常复杂。因此某个设备上的配置错误不单单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。例如在下图的网络配置下，我们想验证在S节点上的waypoint属性，即数据包从S节点出发，经过B点到达节点D，而实际上数据包并没有通过B点到达D而是通过A点到达D。从结果上来看S的属性被违反了，但是从根因来看，是由于节点B的配置对于目的节点为D的路由更偏好从节点A发来的路由AD而不是从节点D发来的路由D造成的。

P4

如果我们使用现有的网络验证工具，我们可以得到以下结果。左边的运行结果来自目前被广为使用的simulation-based工具batfish，它可以模拟网络配置部署后的运行结果，我们可以知道实际上数据包会按照s-a-d的路径转发，它提供了错误流。而右边是另一个SMT-based工具minesweeper的运行结果，我们只能知道对于我们想要验证的属性，结果为false。由此来看，这些工具仅仅能回答对于验证属性对或者错的答案，但是它们无法诊断出究竟是哪些配置行引发了验证属性的被违反。也就是说，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？

P5

针对以上问题，本方案针对网络运行中常见的故障检测和修复需求，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，构建智能的自动化网络管理系统，可以说具有开创性的意义。

P6:

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（此时播放动画）。

P7：

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。

P8：

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。该网例的配置设置为：S优选来自A的路由，S不宣告任何路由，A, B不向D做任何宣告。一开始D节点会向它的邻居节点发送路由（动画），此时我们记录一条因果关系：表示D从初始状态开始，event为D选择[D]作为最优路由为[D]，产生两个action：[D]发送给A，[D]发送给B。然后A接收到了来自D的路由通告（动画），触发计算，更新路由表，并继续向S发送（动画）。B同理，接收路由（动画），发送给S（动画），但S始终优选来自A的路由，所以路由计算到这里就结束了。

（动画）所以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。

P9：

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。

P10：

图的任一拓扑排序对应的最终计算结果都是等价的，（动画）从而我们可以得到分布式路由程序的串行的计算序列。（动画）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。

P11:

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。

P12：

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。

P13：

可以通过对事件队列监听，仿真代码插桩等方式实现。

P14

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。我们使用Minimal Unsatisfiable Core（MUC，最小不可满足核）和 Error Invariants（EI，错误不变量），对错误相关程序段和错误解释进行了形式化定义。

错误诊断定位的总体思路为：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。最终根据错误类型和关键词映射精确定位配置行错误。

我们用右边的例子具体描述整个定位过程，对于xZero程序的每一行，我们首先会把它们编码成一条条smt语句，接着基于程序执行的路径公式，使用smt求解器计算最小不可满足核可以定位出最小的错误语句集合，去掉集合里的任意一条语句就不会导致错误的产生。最后对该错误集合中的每一条语句计算错误不变量，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。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。

P15-18

回归到实际网络的例子中，我们根据给定的错误序列，编码为一组不可满足的SMT公式，此外，为了确定错误路径中违反网络要求的具体内容，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。对这一组公式求解MUC，我们可以得到黄色框出的部分。也就是说，这些黄色公式的合取必然导致最后assertion的违反。对这些公式计算差值，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。

产生错误的整个过程可以通过错误解释得到：D根据自身配置发出D的路由通告；A在接收了来自D的通告之后更新了自身到D的最优路由，并引发将最优路由发送给S的操作；S收到来自A的路由通告后，因为在此时S没有到达p的路径，因此直接选择来自S的通告作为最优路径；S收到来自B的路由通告时设置的路由优选值为120，小于来自A的路由通告优选值200，因此S不会选择来自B的路由作为最优路由，也就导致了最终属性的违反。

由于MUC和EI的性质，使用MUC和EI进行配置错误定位可以保证我们找到的错误具有可靠性和最小性，同时提供对错误产生的合理解释。

P19

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。

P20

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。

P21

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。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。

P22

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。

P23

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。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。

P24

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，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。

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。

P25

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。从图中可以看出，整个端到端的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。

修复后拓扑上配置行修改的百分比如表所示。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。

P26

基于以上介绍，我们提出的Scalpel系统的创新点主要体现为：

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，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，填补了当前**自动网络配置管理全周期**的空白。

Scalpel设计了因果关系图精确、轻量级捕获机制，**可靠、最小化**故障定位与解释算法和**最小化修复**求解器。解决了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，尽可能少的改动现有配置，实现正确修复的挑战。

Scalpel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，极大地降低了其使用门槛。

目前的实验结果表明Scalpel支持低开销、快速网络配置故障诊断与错误修复，并且该项目相关论文已被CCF C类国际会议APNet 2023录用。