# Abstract

在所有虚拟化方法中，软件定义网络在通信级别利用这种范式并为网络提供“操作系统”。因此，数据包处理设备的控制逻辑被转移到外部控制器上，与纯分布式系统相比，这些分离的控制平面提供了更高的可靠性、可扩展性和性能。这一事实正在将网络移向应用层，并在网络内提供应用程序直接交互。然而，构建虚拟化网络的要求相当难以实现，因此，高效的测试平台环境应该会显示成功。在这一领域，Mininet 使用的基于容器的仿真能够在单台计算机上模拟大型拓扑。我们开发了 Mininet 拓扑的自动初始化，并提供了小规模构建真实世界测试套件的能力。为此，我们使用 Internet Topology Zoo 的图形格式根据 Mininet 语法生成网络。此外，我们开发了一个分布式测试套件，用于使用分布式 Internet 流量生成器的评估目的。对生成的流量的统计分析揭示了 Mininet 在拓扑方面的性能。

# INTRODUCTION

在过去的几十年中，虚拟化彻底改变了计算领域的结构。然而，通信基础设施几乎没有受到影响，但也有即将推出的虚拟化这一细分市场的方法。软件定义网络 (SDN) 是一种范例，它使用控制器实例来监督网络拓扑中的跃点。图 1 描绘了 SDN 方法的几个层。应用层持有业务应用程序，并向最终用户介绍网络的透明度。应用层通过应用编程接口 (API) 与提供所需网络服务的 SDN 控制软件进行通信。控制器通过控制数据平面连接到基础设施层，并且可以在运行时更改网络设备配置。在其他方法中，McKeown 等人。 [1] 介绍了 OpenFlow 协议，它是开发 SDN 解决方案的基本要素。在这里，它是第一个独立于供应商的通信接口，位于 SDN 的控制层和转发层之间。因此，OpenFlow 由开放网络基金会 (ONF) 维护，该基金会是一个用户驱动的组织，正在以开放标准实施 SDN [2]。

证明这样一个系统的功能会出现一个主要问题。设置环境成本太高，尤其是在完整的基于硬件的测试平台上部署系统。在这里，虚拟化再次成为游戏的一部分。 Open vSwitch [3] 是一个软件定义的网络堆栈，不需要基于硬件的基础设施，拓扑可以在一台机器内虚拟化。兰茨等人。 [4] 介绍了 Mininet，这是一个在单台计算机上运行的框架，能够根据其条件模拟软件定义的网络拓扑。

现在出现了一个问题：我们如何在这样的测试平台中表示真实世界的拓扑？在我们的论文中，我们定义了一种基于 Internet 拓扑动物园 (ITZ) [5] 的图形地图来构建 Mininet 配置的方法。这是一个平台，它以图形符号提供现实世界的几种拓扑结构，其中包含交换机或路由器的地理位置，以下称为跳数，以及它们的互连。解析器使用这些图为测试台生成拓扑，您可以将所有现有拓扑迁移到 Mininet 到您的计算机上。

对于 Mininet 拓扑的评估，我们使用了分布式互联网流量生成器 (D-ITG)，Avallone 等人。 [6] 发展。它以多种方式产生流量，并能够收集所有相关统计数据。在一个简单的测试拓扑中，我们揭示了 Mininet 在有或没有交叉流量的设置中的表现，以及跳跃故障引起的影响。

# II. SOFTWARE-DEFINED NETWORKING

传统网络由其硬件组件和在这些组件上运行的软件塑造而成。理想情况下，硬件本身应满足以下要求。首先，它很简单，因为构建运行网络的硬件应该很便宜。其次，供应商中立描述了多个供应商的部分应该在混合设置中协同工作。第三，这是一个面向未来的概念，因为不需要突然升级硬件来支持未来的创新。此外，网络的软件方面基本上只有一个要求：可以灵活地轻松更新控制平面，以支持运营商对网络进行廉价的更改。然而，当今任何流行的技术都没有达到一个目标。最初，网络部署不必要地昂贵、难以在运行时更改、维护成本高并且在维护停机时对用户不是很友好。过去，已经尝试了几种方法，但几乎所有方法都没有达到任何重要意义。网络设计取决于两个主要影响因素。一方面，网络需求由用户和运营商提出。用户通常希望他们的数据（他们的数据包）到达所需的目的地，同时满足多个服务质量 (QoS) 属性。然而，网络运营商有不同的需求，如流量工程、虚拟化、隧道或网络分离，这些需求往往对用户是透明的，甚至是不相关的。另一方面，网络接口通过技术实现细节来定义网络。给定协议的数据包头将用户的要求从他们的主机广播到网络，并且对执行的任何类型的工作都是透明的。对于运营商来说，需要完成更繁琐和耗时的网络配置，以便让每个网络节点运行。最后，数据包通过其报头中的特定数据向交换机标识。在最初的互联网运营商要求中可以忽略不计。只关注从源主机流向目的主机的数据包，而每一跳都更新路由信息以转发数据包。由于运营商没有网络接口，硬件在每一跳都手动编程。主机网络和分组交换基本上是一个接口。多协议标签交换 (MPLS) [7] 开始区分网络的“核心”和“边缘”。一个标签被分配给到达网络边缘的数据包，用于在它们穿过核心时转发它们。标签既用于在网络内部转发的目的，也用于满足运营商的要求，如流量工程或隧道。标签信息本身与主机协议（即 IPv4）解耦，该协议表达主机对网络的要求。虽然 Internet 协议 (IP) 仍然是表达主机要求的接口，但标签是负责数据包交换的接口。对于广播，运营商的要求与原来的互联网相比没有任何变化。

SDN带来的最大变化是为网络运营商引入了可编程接口。控制平面与数据平面分离，这意味着传输数据的节点拓扑不同于用于传播控制更改的拓扑。图 2 显示了通过两种不同拓扑共存实现的分离平面。控制器平面基本上是一棵树，控制器作为根节点连接到所有其他交换机和路由器。所有连接的主机都不能影响控制器平面，因为它对网络外部是透明的。网络拓扑由不考虑控制器的实际数据平面表示。有几种具有这种分离级别的网络设计，即 Ethane [8] 或 OpenFlow [1]。 SDN 领域的事实上的标准协议是 OpenFlow，这里将进行更详细的讨论。跃点被构造为将额外的接口单独导出到用于管理转发的控制器，这是通过在数据包头字段和遇到匹配数据包时采取的操作之间的转发表中应用映射来完成的。要匹配的报头部分可以是当今广泛使用的标准协议的一部分，例如 IP 或传输控制协议 (TCP) 和用户数据报协议 (UDP)。动作是将数据包发送到特定端口或重写其协议头数据。这种架构设计有几个优点。网络运营商从一个点控制硬件，并通过对网络控制器进行编程来推出新的设置和更改，而不是手动对每个硬件实例进行编程。 SDN 通过转发表项集体控制数据流的灵活性可用于实现要求，例如隔离不同的网络。 然而，SDN设计也有一些缺点。 OpenFlow 实现中的 SDN 并没有简化网络硬件。 控制器提供的流表匹配应用规则仍然需要解析所有的头部，这在 MPLS 方法中更简单。 如果外部网络协议发生变化（即 IPv4 到 IPv6），则 SDN 中的匹配行为也必须改变，以继续工作。 这是因为头信息和匹配规则不再适合，可以通过 MPLS 方法进行改进。 有了它，只需更新网络边缘，即可将网络切换回工作状态。

## A. OpenFlow

麦基翁等人。 [1] 描述了它们如何使用 OpenFlow 规范在校园网络中实现创新。 OpenFlow 的灵感来自于识别由多个路由器和交换机实现的一组通用功能。因此，该开放协议允许网络管理员将流量划分为不同的流，因为该功能对网络运营商来说是透明的。

如表 I 所示，每个流表由三个字段组成。定义流的数据包头，定义数据包应如何处理和统计的动作。后者是跟踪每个流的数据包和字节数以及自上一个数据包与流匹配以来的时间的计数器，主要用于帮助去除不活动的流。对于匹配过程，表 II 的头字段是相关的。 OpenFlow Switch 的数据路径呈现了一个干净的流表抽象；每个流表条目包含一组要匹配的数据包字段和一个操作（例如发送端口、修改字段或丢弃）。当 OpenFlow Switch 收到一个它以前从未见过的数据包时，它没有匹配的流条目，它会将这个数据包发送到控制器。然后控制器决定如何处理这个数据包。它可以丢弃数据包，也可以添加一个流条目，指导交换机将来如何转发类似的数据包。

控制器通过单独的安全通道连接到网络，如图 2 所示。存在这种控制器的几种实现方式。 OpenFlow 控制器的第一个实现是 NOX [9]，它是用 C++ 和 Python 编写的。一个以多线程基础设施为核心的纯 C 实现是 M¯uL [10]。它专为性能和可靠性而设计，这是从任务关键型网络部署一开始就需要的。 ovs-controller [11] 是一个简单的参考控制器，它与 Open vSwitch [3] 一起打包。 Trema [12] 提供了一个使用 Ruby 的变体，它是一个全栈框架，用于在 Ruby 和 C 中开发 OpenFlow 控制器。 POX [9] 作为 NOX 的继承者是一个高级 SDN API，因为它是经过编程的在 Python 中，因此与平台无关。 POX 提供可查询的拓扑图并支持虚拟化。另一种基于 Python 的方法是 Ryu [13]，它是一种支持 OpenFlow 的开源网络操作系统 (NOS)。最后，缺少一些 Java 控制器。 Jaxon [14] 依赖于 NOX，并且根本不是平台独立的。 Maestro [15] 能够编排网络控制应用程序。 Beacon [16] 支持基于事件的和线程化的操作。最初，Apache 许可的 OpenFlow 控制器 Floodlight [17] 是从 Beacon 控制器分叉出来的。它得到了开发人员社区的支持和改进，并且支持广泛的虚拟和物理 OpenFlow 交换机。在第三节中选择它来控制我们的网络。此外，使用 NodeFlow 有目的控制器。例如，RouteFlow [19] 是一个为支持 OpenFlow 的硬件提供虚拟化 IP 路由服务的开源项目，主要用 C++ 和 Python 编写。 RouteFlow 由一个 OpenFlow 控制器应用程序、一个独立的 RouteFlow 服务器和一个虚拟网络环境组成，该环境再现物理基础设施的连通性并运行 IP 路由引擎。对于使用多个 OpenFlow 控制器，Flowvisor [20] 充当透明代理。简单网络访问控制 (SNAC) [21] 是一种基于 NOX 的 OpenFlow 控制器，它使用基于 Web 的策略管理器来管理网络。 Oflops [22] 是一个标准控制器，可以对 OpenFlow 交换机的各个方面进行基准测试。

有可用的选项，但到目前为止没有一个能够满足原型网络的需求。基于硬件的设置过于昂贵，因此缺乏灵活性、可扩展性、可共享性和可负担性。之前的基于软件的解决方案（如虚拟机网络）由于重量过重和资源过度密集而缺乏可扩展性，因此也缺乏可共享性。软件测试平台与不可部署的模拟代码不匹配，也缺乏交互性。其他方法试图解决分布式测试平台的缺点，从而能够支持大型拓扑并使其可共享。仍然有一个仅在单个工作站上运行的解决方案将进一步解决无法访问这些测试网络的人的可扩展性、可共享性和可负担性。随着 SDN 的出现，出现了另一种解决方案，称为 Mininet。它承诺使用SDN技术一次满足上述所有需求。对此索赔的评估将在第 IV 部分进行。

## B. Mininet

对于原型网络，需要满足进一步的要求。为提高灵活性，应在软件层面而非硬件层面定义拓扑，这将降低设置和开发期间的成本。在软件级别创建的网络设计应该可以部署在实际硬件上，而无需更改代码或配置。通过实时运行、管理和测试设计来提供交互性。描述真实世界的网络需要可扩展性，其中单个工作站模拟具有数百个节点（如果不是数千个）的网络。在测试平台中，网络的协议和特性应该尽可能接近生产环境中部署的网络实例。网络设计应可在协作者之间共享，无需复杂的设置和配置。测试平台应该足够便宜，对于有需要的人来说是负担得起的，无论是在学术界还是在公司工作。

Mininet 幻觉背后的神奇之处在于 Linux 中内置的一组功能，这些功能允许将单个系统拆分为一堆较小的容器，每个容器具有固定的处理能力份额，并结合虚拟链接代码，允许链接具有准确的延迟和速度。在内部，Mininet 采用了 Linux 内核中的轻量级虚拟化特性，包括进程组、CPU 带宽隔离和网络命名空间，并将它们与链路调度程序和虚拟以太网链路相结合。与基于完整虚拟机的模拟器相比，这些功能产生了一个启动速度更快并扩展到更多主机的系统。例如，1, 057 个节点分配了 492MB 的内存并在大约 817 秒时启动 [4]。因此，一个 Mininet 网络由三个元素组成 [23]，如图 3 所示。

• 虚拟主机，这是移入网络命名空间的一组用户级进程。网络命名空间本身是网络状态的容器，并为进程组提供专有的所有权接口、端口和路由表。例如，两个网络命名空间中的两个 Web 服务器可以在一个系统上共存，两者都侦听同一端口上的专用以太网接口。为了保证公平性，Mininet 将每个进程组的处理器带宽限制为可用 CPU 时间的一小部分。

• 通常使用默认Linux 桥接器或Open vSwitch 的虚拟交换机在内核模式下运行以跨接口交换数据包。此外，交换机和路由器可以运行在内核空间以加快数据包交换或在用户空间中轻松修改它们。

• 隔离链路使用明确的数据速率定义，由 Linux 流量控制强制执行，该流量控制由多个数据包调度程序组成，以将流量调整为配置的速率。虚拟以太网对就像连接两个虚拟接口或虚拟交换机端口的电线。通过一个接口发送的数据包被传送到另一个接口，每个接口对于所有系统和应用软件来说都是一个功能齐全的以太网端口。

# III. TOPOLOGIES IN MININET

Subsection II-B 中的 SDN 模拟器需要在 Python 中定义的拓扑才能执行。 尽管来自第 III-A 小节的 ITZ 提供了一个包含许多现实世界拓扑的数据库，但它们是用图形符号定义的。 为了使它们在 Mininet 2.0.0 版中可用，一种可能性是使用来自第 III-B 小节的拓扑解析器。

## A. The Internet Topology Zoo

ITZ [5] 是网络拓扑数据的存储。

通过对网络拓扑的研究给予了很多关注，但由于缺乏准确的数据而停滞不前。测量拓扑的方法存在缺陷，因此所有论点都围绕这些方法进行，并克服了有关网络结构的重要问题。由于运营商发布有关其网络的信息，因此 ITZ 数据库包含从 AboveNet 到 Zamren 的拓扑。因此，ITZ 是公开可用的最准确的网络集合，包括不可测量的元数据。在我们的第 III-B 小节示例中，我们使用德国研究网络 (DFN1) 拓扑，如图 4 所示。

所有拓扑都采用图形格式，使用可扩展标记语言 (XML) 作为描述基础。 GraphML 格式中每个拓扑的有效性由 GraphML2 文件的 XML 模式定义 (XSD) 给出。图形格式提供了足够的信息来建立关于真实世界拓扑的测试平台网络。清单 1 描述了构建网络的重要元素。为了确定关键元素的用途，attr.name 字段包含相关信息。在这里，特别需要节点的标签及其经纬度位置在 Mininet 中重建网络。此外，边缘元素指定节点之间的连接，需要在 Mininet 中链接交换机。

B. The Topology Generator for Mininet

由于 ITZ 中可用的所有拓扑结构都非常相似，因此可以轻松解析它们以生成可执行的 Mininet 拓扑。 Mininet 中使用的拓扑是与 Mininet API 接口的可执行/可加载 Python 类。所以每个可用的 Mininet 拓扑都是相似的，在文件的头部和尾部具有相同的内容。文件基本不同的部分是定义

• 主机条目，

• 切换条目和

• 交换机和主机之间的链接。

可执行文件和可加载文件的区别在于 Python 脚本末尾的代码。如果拓扑是从 Linux shell 执行的，Mininet 会自动启动并使用已定义的拓扑和可用的安全 shell (SSH) 访问。 Loaded 意味着 Mininet 是从 Linux shell 单独启动的，拓扑作为调用参数通过 topo 参数给出。定义对拓扑节点的 SSH 访问的代码没有执行，只执行定义拓扑的部分。作为回报，这意味着仅通过加载 SSH 访问来使用拓扑是不可用的。所以首选的用法是直接从 Linux shell 执行拓扑。有几个命令行参数可用于使用 ITZ 解析器。通过它们，您可以为拓扑定义一个 GraphML 输入文件，该文件应该被解析为 Mininet 可以使用的 Python 拓扑。通过将所有边缘的 bw 值设置为给定值，也可以调整带宽（以 Mbps 为单位）。此外，解析器要求文件位于同一目录中，并且不指定程序将终止的输入参数。但是，某些值可以省略，例如带宽限制，否则会被初始化为 10Mbps。如果省略，则遥控器IP初始化为“10.0.2.2”，这是使用Oracle Virtualbox进行虚拟化时主机操作系统的标准IP。此外，Mininet 的延迟是通过使用 ITZ 拓扑的地理坐标来计算的。

dist(SP, EP) = arccos{sin (LaEP) · sin (LaSP)+ cos (LaEP) · cos (LaSP) · cos (LoEP − LoSP)} · r

(1) 在拓扑距离由球面计算

由方程 1 描述的余弦定律，其中假设半径约为 r = 6, 378, 137m。 SP是起点，EP描述终点。 La 是纬度值，Lo 是地理坐标的经度值，必须以弧度值给出[24]。

tL =

dist(SP, EP) vL

(2)

此外，信号速度近似于光速和光纤的反射系数 1.52，使得 vL = 3·108 1.52

米 s = 1.97·108 米

s [25]。因此，估计

延迟时间 tL 由公式 2 计算。例如，清单 1 在字段 d30 中提供了

纬度和 d33 中的地理定位的经度值，以便我们可以根据此图形格式确定两点之间的延迟。首先，我们提取 CHE 的地理坐标，即 LaCHE = 50.83333° 和 LoCHE = 12.91667°。连接建立到 LEI，其中 LaLEI = 51.33962° 和 LoLEI = 12.37129°。 CHE 和 LEI 之间的距离由公式 1 计算，使得 dist(CHE, LEI) = 68557.871953m。以同样的方式，通过公式 2 计算延迟，使得 tL = 68557.871953m

1.97·105米

延迟被写入清单 2 的 python 脚本，用于初始化 CHE 和 LEI 之间的边缘。

# IV. EVALUATION SUITE FOR MININET

Mininet 拓扑的评估是另一个问题。

在能够建立ITZ的所有拓扑后，需要关注几个指标的评估。对于流量生成，我们在 IV-A 小节中使用 D-ITG，最后在 IV-B 小节中使用 Mininet 拓扑进行了一些试验。

1. Distributed Internet Traffic Generator

为了评估 Mininet D-ITG 在 2.8.0-rc1 版本中的性能，使用了：“分布式互联网流量生成器 (DITG) 是一个能够产生准确遵守由交叉口定义的模式的流量的平台数据包之间的时间 (IDT) 和数据包大小 (PS) 随机过程”[6]。因此，它为流量生成提供了丰富多样的概率分布，并使用了一些建议的模型来模拟各种协议的来源。有了它，就可以生成各种数据包流并使用日志服务器收集统计信息。图 5 描绘了 D-ITG 的所有重要模块。 ITGSend 模块负责流量生成，而 ITGRecv 模块是数据包的接收器，通过数据通道传送。为了收集日志信息，ITGSend 和 ITGRecv 正在通过日志通道与 ITGLog 模块进行通信。对于远程控制，ITGManager 提供了通过信令通道调整 ITGSend 参数的功能。

除了图 5 的模块之外，D-ITG 解码器 (ITGDec) 还分析 ITGLog 模块收集的结果。它计算丢包率、吞吐量、抖动和延迟，包括单向延迟 (OWD) 和往返时间

1. Measurement Trials

图 6 描述了我们测量试验的拓扑结构。 S1 到 S6 是拓扑中的交换机，而 Sender/Receiver 表示处理生成的流量的主机。 在每个发送方调用一个 ITGSend 进程来生成流量，而在每个接收方 ITGRecv 处理数据包的接收。 另外，日志服务器通过运行一个ITGLog实例来收集主机的相关统计数据。 S4 关闭进行几次测量试验，因此所有虚线链接都不可用。 总而言之，我们总共进行了四次试验，分别针对 TCP 和 UDP，每次试验的持续时间为 1 分钟，设置如下：

1) 只有 Sender 1 和 Receiver 1 负责没有交换机 S4 的拓扑上的流量。

2)

发送方 1 和接收方 1 正在处理完整拓扑上的流量。

3) 两个发送方/接收方对都负责没有交换机 S4 的流量。

4) 两个发送方/接收方对都在处理完整拓扑上的流量。

1. Results

Mininet [4] 虚拟机在具有 Vmware3 虚拟化的 HP Proliant DL380p Gen8 服务器上运行，而 Floodlight 控制器 [17] 在标准计算机上执行。 对于试验，SDN 跃点的带宽限制为 10Mbps。 在 D-ITG 中，通过 ITGSend 生成的流量设置为一个数据包的最大有效载荷大小，以便通过改变两个连续数据包的 IDT 来调整流量速率。 测试比特率以 0.5Mbps 的步长从 8Mbps 达到 12Mbps。 为了探测九种不同的比特率，一种协议和一种设置的试验持续时间为 9 分钟。 因此，两个协议需要 18 分钟，完整试验需要 72 分钟。

在图 7 中，试验在具有发送器/接收器对 1 且没有交换机 S4 的拓扑上运行。我们观察到，拓扑的接收吞吐量为 TCP（图 7a）流量约 9Mbps，UDP（图 7b）流量接近 10Mbps。此外，TCP 拥塞控制在图 7b 的延迟图中清晰可见，并试图找到 20 秒以上的阈值。此外，在接收方，如图 7a 和 7b 所示，我们观察到延迟线性增加到某个阈值，对于 UDP，大约为 1.2 毫秒，尽管发送方不对这种行为负责。与图 11 的 UDP 丢失图相比，图 7b 的 1.2ms 延迟的实现发生在 10 秒，这与图 11b 显示第一个数据包丢失（以每秒数据包 pps 为单位）的时间相同。然而，避免阈值表明所有数据包都以 9Mbps 的比特率传输（图 11a）。在图 8 中，试验在具有两个发送器/接收器对的完整拓扑上运行。再次出现类似的观察结果，虽然两个发送方都生成吞吐量为 10Mbps 的流量，但接收到的总比特率对于 TCP（图 8a）约为 18.5Mbps，UDP（图 8b）约为 20Mbps。同样，UDP 场景的延迟显示线性增加，而 TCP 场景中出现剧烈振荡的增加，这可能是由交叉流量生成的拥塞控制机制重叠引起的。 IDT 分布描绘了可能发生的负时间，当一跳（例如，S2）被完全利用时，从发送器 1 到接收器 1 的流量正在使用 S3 进行后续传输。新路由上的数据包比旧路由上的前辈更快到达，这会在 IDT 中产生负面结果。

图 9 和 10 在一张图中收集了几次试验的统计数据。因此，平均值和相应的标准偏差是在图 7 和图 8 最上面三个图的时间安排上计算的。这些值是针对从 8Mbps 到 12Mbps 的不同测试比特率绘制的。第一个图，最上面，描绘了以 Mbps 为单位的吞吐量，第二个以毫秒为单位的延迟，第三个以毫秒为单位的抖动值。

发送方总是在执行测量的每种情况下生成恒定的测试比特率，而不会引入额外的延迟（图 9a 和图 10a），尽管吞吐量生成的方差通过达到 4拓扑生成器，Mininet 的 D-ITG 自动化而增加测量试验的结果可在 http://141.13.92.69/index.php/projects/auto-mininet 在全拓扑的交叉流量试验中，我们观察到吞吐量稳步增加，直到达到图 10 中的两个流均为 10Mbps。例如，当两个发送方都以 10Mbps 的恒定比特率生成流量时，总接收吞吐量估计约为 18.5Mbps。在流量达到阈值之前没有延迟，然后收敛到 1.1 毫秒。在抖动图中，很明显比特率增加得越多，抖动的平均值就越低，因为在阈值之后，使用几乎相同的 IDT 处理数据包。

# V. CONCLUSION & FUTURE WORK

借助 Internet Topology Zoo 解析器，我们为在 SDN 仿真器中评估真实世界的网络拓扑奠定了基础。此外，您现在可以在 GraphML 语法中指定网络并将它们迁移到 Mininet，而无需接触 Python 代码。将几乎每个网络解析到您自己的笔记本电脑的简单性揭示了测试 SDN 行为的多个方面的新可能性。例如，Heller 等人描述的控制器放置问题。 [28] 可以用现实世界或自定义拓扑进一步分析。此外，通过 SSH 访问所有主机的测试套件提供了使用流量生成引擎在一定程度上评估网络行为的能力。在这里，可以在代表真实世界拓扑的可扩展环境中测试 SDN 的监控。根据 SDN 开源计划，拓扑解析器和测量试验的所有结果将公开4。通过使用 D-ITG，我们评估了一个由 Floodlight 控制的示例拓扑。我们的第一次测量试验表明，Mininet 在选定比特率的给定范围内按预期执行。在未来的实验中，需要增加网络规模，特别是通过使用 ITZ 的拓扑结构，以评估 Mininet 可以处理多少个节点并具有可接受的性能。基本上，SSH 访问是将测试模块推送到 SDN 的基础，SDN 是在 ITZ 解析器的使用中建立的。此外，对不同控制器的评估可以揭示它们的应用领域。