Liveness Verification of Stateful Network Functions

论文标题及作者信息

标题： 状态网络功能的活性验证
作者： Farnaz Yousefi, Johns Hopkins University; Anubhavnidhi Abhashkumar 和 Kausik Subramanian, University of Wisconsin-Madison; Kartik Hans, IIT Delhi; Soudeh Ghorbani, Johns Hopkins University; Aditya Akella, University of Wisconsin-Madison

摘要

网络验证工具几乎专注于各种安全属性，例如“可达性”不变量，例如，从主机A到主机B是否存在路径？因此，它们不适用于为越来越依赖状态网络功能的现代可编程网络提供强正确性保证。这类网络的正确操作依赖于更大的属性集，特别是活性属性。例如，一个只允许受邀外部流量的状态防火墙如果最终检测并阻止恶意连接，则工作正常，例如，如果它最终阻止外部主机E在收到来自I的请求之前尝试到达内部主机I。遗憾的是，验证活性属性在计算上昂贵，在某些情况下，甚至是不可判定的。现有的验证技术无法扩展以验证此类属性。在这项工作中，我们提供了一个组合编程抽象，使用紧凑的布尔公式对用此抽象表达的程序进行建模，并展示了这些公式上复杂属性的验证速度很快，例如，对于一个100主机网络，这些公式在验证UDP洪水缓解功能的关键属性时比简单基线快8倍。我们还提供了一个编译器，将使用我们的抽象编写的程序转换为P4程序。

1 引言

近年来，网络验证已成为检查网络是否满足重要属性的关键框架。虽然有多种工具在关注点上有所不同（例如，验证当前数据平面快照与验证网络的控制平面在故障下的情况），但它们都有一个共同的特点：主要关注验证各种风味的可达性不变量：网络中的一个点是否可以从另一个点到达？是否存在无环路径？路径是否拥堵？是否穿越了一个路标？在链接故障或外部消息下是否保持可达性？所有数据中心的最短路径是否可用于路由等。然而，这些工具忽略了一组更丰富的属性，这些属性取决于网络保证活性。

如今的网络越来越多地部署复杂且有状态的网络功能，例如入侵检测/防御系统（IDPS），它们监控流量以检测恶意活动和策略违规，并阻止此类活动。要依赖这样的网络，运营商需要验证“最终会发生一些好事（一个期望的属性）”。例如，考虑一个状态防火墙，其策略是，企业外部的主机E只有在内部主机I首先向E发送请求后，才允许向I发送流量。这里的活性属性是“是否最终会将应该被允许向I发送流量的主机E（即，E首先被I联系）列入白名单？”，或更准确地说，“I向E发送流量的事件导致防火墙将E列入白名单”。现有以可达性为中心的工具无法验证此属性：从I到E的路径的存在并不表明任何数据包实际上已经穿越了那条路径；同样，从E到I的可达性也不能保证它是在I发送流量之前还是之后建立的。

最近的工作表明，通过在等价类（ECs）上操作，可以有效地进行可达性验证，即，经历相同转发行为的数据包组。然而，验证活性不适用于此类技术，因为活性属性推理进展并涉及动态系统中事件的连续性。它们不能在系统的静态快照上进行验证。可以将动态网络（其中状态发生变化）建模为状态机，并且从概念上讲，现有的静态验证方法可以扩展以推理此机器的状态和转换的属性。然而，这种简单的方法会导致随着网络规模的增加而出现状态爆炸，因此在实践中是不切实际的。

在本文中，我们认为，使用自上而下的面向功能的策略来验证活性属性的目标是可实现的，这种策略重新思考了网络抽象，以提高验证效率。为了实现这一愿景：（a）我们为网络程序员提供了一个简单、熟悉的抽象，即一个大开关，以表达他们的意图。这种抽象强制逻辑上分离不同的网络功能。（b）然后我们将程序建模为一个紧凑的“无数据包”的数据结构，与常见的为数据包类建模转发行为的方法不同，它抽象掉了数据包的显式概念，而是专注于实现功能的实体：数据包处理规则。（c）我们构建并评估了我们系统的原型。

抽象：我们为控制平面和数据平面提供了一个统一的大交换机抽象，概念上处理所有数据包。这种抽象与直接连接所有主机的简单、熟悉的数据平面大交换机抽象非常相似。与数据平面的大交换机不同，我们的抽象不需要单独的控制平面来进行编程。为了减少验证时间，我们通过要求用户使用单独的逻辑流表来实现每个功能，强制执行功能分解。

建模和验证：为了验证属性，与以前的工作类似，我们专注于对用户可见的网络行为。这使我们能够使用一个紧凑的“无数据包”数据结构来建模系统行为，该结构抽象了对用户不可见的任何细节，例如数据包在网络内部的逐跳旅行，甚至是数据包或数据包类的显式概念（§3）。无数据包结构将转发行为的变化建模为布尔转换。我们在实验中展示了无数据包模型的验证效率（§4），例如，对于UDP洪水缓解功能，在1000秒的时间限制内，它使得验证一个关键活性属性（主机A最终被阻止）的网络规模比上述简单方法（扩展静态验证以处理网络状态和动态转换）可验证的网络大3.5倍。

实施和评估：我们开发了一个原型设计，该设计公开了两个接口（用于在一个大交换机抽象上表达功能和指定验证属性）和一个P4编译器，将这些功能转换为在当今可编程设备上可执行的程序。我们的评估显示了我们接口的表达性，它们的低开销，以及我们的验证设计的效率，例如，对于一个100主机网络，无数据包模型验证UDP洪水缓解功能的关键活性属性的速度比基于数据包的基线快8倍——这种差距随着网络规模的增加而增加（§4）。

2 统一交换机抽象

为了简化编程并减轻网络程序员编写分布式、多层程序的负担，我们提供了一个概念上处理每个数据包的单一集中式交换机的抽象。这种简化编程的方法通过为控制平面和数据平面提供单一统一的抽象得到启发，类似于Maple并在Flowlog中部署。与Maple允许程序员使用标准编程语言和Flowlog的类SQL语言不同，我们从网络中最基本和熟悉的数据平面抽象开始：一个直接连接所有主机的交换机。然后我们增强这个抽象使其可编程。与Maple和Flowlog类似，我们主动编译在我们的抽象上编写的程序，以控制和数据平面程序的形式在当今网络中执行（§4）。我们描述了我们的抽象，它与常见的大交换机抽象的区别，以及如何在其上编程几个典型的网络功能。

大交换机：网络中常见的数据平面抽象是一个逻辑交换机，它有多个流表，每个流表包含一组规则，直接连接所有用户的主机在一起。一条规则通常包括：（a）匹配字段以匹配数据包头和入口端口，（b）优先级以确定规则的匹配优先级，（c）计数器在匹配数据包时更新，（d）计时器显示规则将过期并从交换机中移除的时间，以及（e）当数据包匹配未过期规则的匹配字段且具有所有未过期规则中最高优先级时执行的动作。这些动作可能导致数据包更改、丢弃或转发。我们使用$R.match$、$R.priority$、$R.counter$、$R.timer$和$R.action$来表示规则R的匹配、优先级、计数器、计时器和动作。

我们的目标是提供一个类似于这个熟悉抽象的类似抽象，同时使其可编程并易于快速验证。特别是，与静态、无状态交换机不同，程序员应该能够实现基于流量随时间变化的动态功能。此外，她应该能够专注于她希望她的交换机提供的高级功能（例如，防火墙策略），而抽象的提供者负责处理功能的低级、分布式实现，包括可达性（例如，确保所有必要的数据包通过防火墙正确转发）。为了进一步帮助用户开发她所需的功能，理想的框架还应该提供模块化编程，并将程序的功能分解为独立的模块。为了实现这些目标，我们对大交换机抽象进行了以下更改：

添加/删除动作：除了标准的SDN动作（转发、丢弃等）外，我们允许规则的执行添加或删除交换机中的规则。例如，为了编程一个有状态防火墙，该防火墙只允许在I向E发送请求后，外部主机E与内部主机I通信，交换机需要一条规则，一旦从I接收到对E的请求，就改变其状态以允许E与I通信。动作$add(R)$和$delete(R)$表明规则执行的结果分别是添加和删除规则R。对于每条规则R，我们添加一个布尔变量$R.active$来显示规则当前是否在交换机上处于活动状态，即数据包是否与之匹配。例如，初始状态$R_0.active = true; R_1.active = false; R_0.action = add(R_1)$表明与$R_0$不同，规则$R_1$最初不是激活的，并且将会作为$R_2$的执行结果被激活。

可操作测量：对于规则R_i，我们允许用户定义匹配字段作为谓词，不仅仅是基于数据包头（例如，src=A），还可以是以下形式的条件：

$$l_i \leq c_j < u_i$$

其中c_j是第j个计数器，l_i和u_i是常数。这个条件表明，为了匹配一个数据包，计数器c_j的值应该在区间$[l_i, u_i)$内。我们假设计数器在$[0, m)$范围内有界。

计数器使得网络程序员能够轻松编写依赖于流量统计数据的关键网络功能，例如安全应用程序检测SYN洪水、端口扫描、DNS放大等，以及校园网络监控器，这些监控器在用户使用超过校园政策规定的配额时阻止用户。每当一个数据包匹配到一个带有计数器的规则时，计数器的值就会增加一。我们允许多个规则共享一个计数器。

其他优化：为了提高可编程性和验证速度，我们还进行了以下优化：

(a) 功能分解：用户被提供了单独的表格来实现她的每个网络功能，例如，一个表格用于她的防火墙，一个表格用于她的负载均衡器等。如果一个数据包匹配到功能F_1中的规则R需要被发送到另一个功能F_2，例如，一个IDPS规则需要将数据包发送到流量清洗器，这将被表示为$R.action = send(F_2)$在表格F_1中。

(b) 声明式路由：我们的抽象提供了路由和转发作为一项服务，我们称之为路由器，给它的用户。这使得用户从计算路径和在基础设施变化后更新它们中解放出来，例如政策、拓扑和地址变化。用户只需要声明数据包应该到达目的地的目标，并将如何完成这项任务的任务委托给提供者。动作规则$send(A)$和$send()$分别简单地表达了用户的意图，将匹配的数据包转发到具有ID A的端点和数据包的目的地。在我们的原型中，为了实现我们的声明式路由服务，我们部署了一个基本的最短路径转发功能。

(c) 符号规则表示：原始的大交换机规则可以表达对显式数据包头的一组有限的谓词，例如$src-IP = 10.0.0.1 \wedge dst-IP = 10.0.0.2$。声明式路由使我们能够提供一个更高级别的抽象来表达对一组端点身份和头字段的任何一般谓词，例如，程序员可以通过单个规则R声明一个转发策略，用于发送到或来自一组称为T的主机的数据包：
$R.match = (src = T \vee dst = T), R.action = send()$，而无需管理$T$的主机的物理位置和IP地址等低级细节。

上述更改将数据平面的大交换机抽象转变为一个可以高效验证的统一抽象：添加/删除动作和可操作测量使得抽象可编程，其他优化通过减少程序大小加速了验证过程。为了进一步帮助高效验证，我们的抽象被设计为具有比图灵完备的控制平面编程语言（如Floodlight和Pyretic）更少的表达能力。我们通过实验发现，尽管在理论上是计算通用的，但在实践中，控制平面只执行有限的操作集，例如，根据流量模式添加和删除规则。我们的大交换机抽象旨在能够执行类似的计算，并因此足够表达性，以编程广泛的网络功能。表1列出了我们在我们的抽象上重写的一些常见控制平面和最近网络抽象的功能（实现在§8中）。为了支持无法在我们的抽象上表达的功能，例如基于内容的安全策略，我们的框架允许在我们的抽象中使用外部库。然而，我们只能验证在我们的抽象上表达的程序。

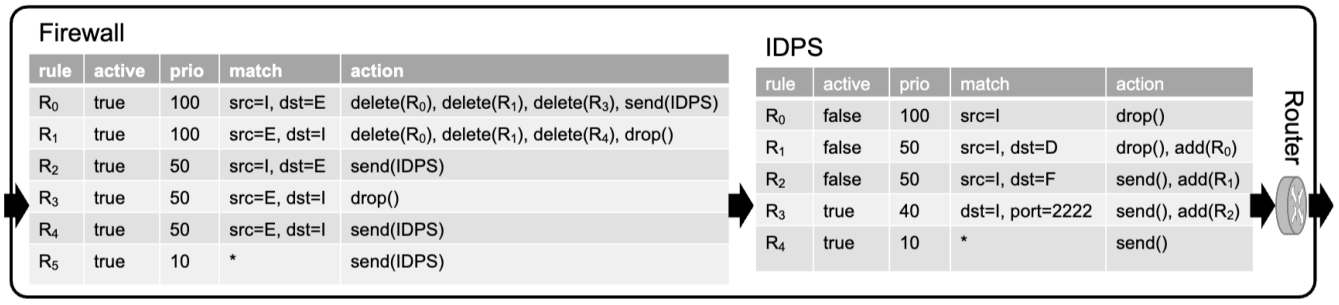


Figure 1: One-big-switch implements firewall and IDPS.

示例。图1展示了一个说明性的例子，其中用户部署了两个表来实现两个典型功能的链式结构。第一个表实现了企业边缘的有状态防火墙，该防火墙只允许端点E（例如，作为外部主机）与I（例如，作为内部主机）通信，前提是I首先向E发送了请求。最初，该表具有高优先级的规则，用于"监视"I和E之间的流量（R_0和R_1）。如果首先观察到从E到I的流量，那么R_1规则的执行将丢弃该流量，并更改状态以阻止E到达I。然而，如果首先接收到从I到E的流量，则会触发R_0的执行，进而允许I和E之间的双向流量通过防火墙。防火墙未丢弃的数据包始终被发送到下一个功能，即下面解释的IDPS，以进行进一步监控。请注意，在定义单个大交换机抽象中的规则时，规则的匹配字段必须定义为至少有一个规则与每个接收到的数据包匹配。

第二个表实现了一个IDPS，用于检测和阻止木马。IDPS根据后门应用程序的可识别指纹确定内部主机I是否被感染并需要被阻止，操作序列如下：（i）I在端口2222上接收到连接，（ii）I尝试连接到FTP服务器F，以及（iii）I尝试连接到数据库服务器D。最初，该表包含两个活动规则：R_3匹配目的地为I且端口为2222的流量，以及优先级较低的规则R_4匹配所有其他流量。两个规则都将流量转发到其目的地。然而，R_3的执行对应于上述的（i）操作。一旦触发，它将激活R_2，一旦I尝试到达F（操作（ii）），R_2就会被执行。R_2的执行反过来激活R_1，一旦I尝试向D发送流量（操作（iii）），它就会被阻止（R_1激活R_0）。流量通过这些表的管道后，将交给路由器以送达其目的地。

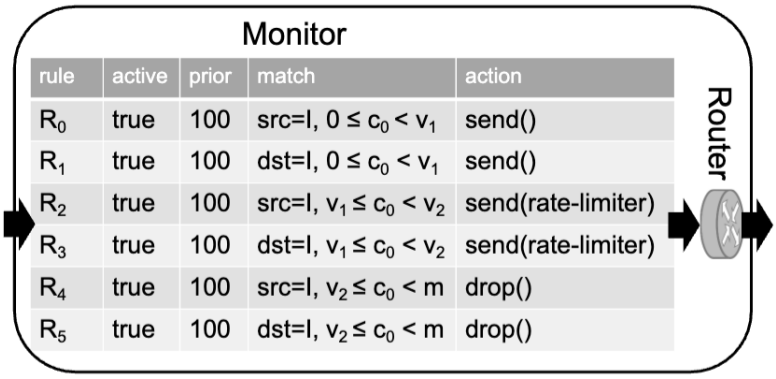


Figure 2: One-big-switch implements a monitor.

图2展示了另一个实现简单校园策略的应用程序的例子，其中校园内的主机I在达到使用上限v_1之前被允许发送和接收数据。一旦其使用量超过v_1，但在达到v_2之前，其流量将通过速率限制器进行路由。在其使用量超过v_2后，其访问将被阻止。校园网络策略的调查显示，大学通常部署此类基于使用量的速率限制策略。虽然为了简化，我们在本节中只提供了功能线性链式的例子（例如，防火墙后面是IDPS），但我们的抽象足够通用，可以表达功能之间的任意依赖关系。一个例子在§8中提供。

3 功能验证

动态系统的活性属性验证的标准方法是将系统的行为建模为转换系统，并使用时序逻辑表达其期望的属性。这种方法的复杂性和可扩展性取决于状态机的大小。在本节中，我们展示如何将有状态的动态网络功能建模为紧凑的布尔"无数据包"转换系统，以便高效验证。在第4节中，我们将通过实验表明，与基于数据包的基线相比，这种方法显著减少了典型应用的平均验证时间。

3.1 网络作为转换系统

我们可以将网络建模为转换系统，这是一个分析框架，用于推理动态系统的行为，其中节点代表系统的状态（每个状态对应于系统变量的一个赋值），边代表状态转换。更正式地说，转换系统TS是一个元组$(S, Act, \rightarrow, I, AP, L)$：

- S是状态集，

- Act是动作集，
- $\rightarrow \subseteq S \times Act \times S$是转换关系，
- $I \subseteq S$是初始状态集，
- AP是原子命题集，
- $L : S \rightarrow 2^{AP}$是标签函数。

为方便起见，我们写$s \rightarrow^{\alpha} s'$代替$(s, \alpha, s')$。直观地说，转换系统从某个初始状态$s_0 \in I$开始，并根据转换关系$\rightarrow$演化。也就是说，如果$s$是当前状态，那么选择一个转换$s \rightarrow^{\alpha} s'$并采取它，即执行动作$\alpha$，系统从状态$s$演化到状态$s'$。在网络中，网络在每个点的状态是其转发状态（例如，规则和计数器），转换是改变状态的事件，例如，策略更新。

接下来，我们展示可以在这些转换系统上表达的属性，解释为什么验证活性很难，并演示如何将网络建模为紧凑的“无数据包”转换系统。

原子命题：原子命题是表达系统状态的简单已知事实的布尔值命题。我们将这些命题定义为(h, a)对，其中h和a分别是数据包的等价类和动作列表（例如，$\mathrm{send(I)}$）。

对于网络转换系统TS，如果动作a适用于h中的所有数据包，则原子命题(h, a)在状态$s \in S$中成立。

标签函数：标签函数L将一组原子命题$L(s) \in 2^{AP}$关联到任何状态$s \in S$。也就是说，如果列表中的动作a适用于h中的所有数据包，则(h, a)存在于$L(s)$中。

属性：程序的执行可以显示为状态的无限序列：$s_0, s_1, \ldots$，其中每个状态s_i是在状态s_{i-1}中执行单个动作的结果。属性也定义为这样的序列集。如果程序定义的序列集包含在属性中，则属性在程序中成立。部分执行是程序状态的有限序列。

时序逻辑表达属性：时序逻辑是普通逻辑的扩展，通过添加关于时间的断言来表达属性。在这里，我们采用线性时序逻辑（LTL），它可以表达各种活性和安全属性。时序逻辑断言是使用普通逻辑运算$\wedge, \vee$和$\neg$以及一些时序运算符构建的——如果P和S是原子命题：(1) GP意味着“现在和将来”P成立，(2) FP意味着“现在或将来某个时候”P成立，(3) $P \rightarrow S$显示逻辑蕴含并意味着如果P现在为真，那么S将永远为真，(4) $P \cup S$意味着P保持真直到S变为真，(5) $\odot P$意味着P在“下一个”状态中成立。使用上述时序运算符，我们可以表达各种属性，例如，
$F((src = E) \wedge dst = I), send(I)$断言最终$E$到$I$的数据包被交付，即$E$最终可以到达$I$。

3.2 活性与安全

在分布式系统中，属性的一个关键分类是安全和活性。这种分类很重要，因为这两组使用不同的技术进行证明。非正式地说，安全属性规定某些“坏事”（死锁，两个进程同时执行关键部分等）永远不会发生，而活性保证“某些好事”（终止，无饥饿自由，保证服务等）最终会发生。也就是说，对于属性P来说，如果P在执行中不成立，那么在某个点必须发生一些不可挽回的“坏事”。今天在网络中验证的大多数属性都是安全的：如果某个属性——如可达性不变量（E始终可以从I到达），路标（某一类流量始终穿越入侵检测系统），无拥塞和无环——被违反，那么就有一个可

识别的点——比如网络的最新快照的变化——在那里“坏事”发生。部分执行γ对于属性P是活跃的，当且仅当存在状态序列β使得P在$\gamma\beta$中成立。对于所有部分执行都是活跃的属性是活性属性。与安全不同，对于活性属性，没有部分执行是不可挽回的——总是有可能在未来发生所需的“好事”。这使得检测活性违规变得具有挑战性，因为它从根本上需要对网络的整个状态空间进行穷尽搜索。在下一节中，我们将讨论通过使用紧凑的转换系统对网络进行建模来克服这一挑战的方法。

网络中的一些活性属性示例包括：(a)入侵检测系统最终检测到所有感染的主机，(b)所有主机最终变得可达，例如，在路由收敛后，(c)显示后门应用程序的可识别指纹导致主机被阻止。更一般地说，“事件A导致事件B”和“事件B最终发生”是两类活性属性，因为总是有可能发生“某些好事”（即事件B）。一个包含活性的属性的例子是完全正确性，它由部分正确性（程序永远不会生成错误的输出；一个安全属性）和终止（程序生成输出；一个活性属性）组成。

3.3 无数据包模型

模型检查的可行性与处理状态爆炸问题密切相关。为了缓解这个问题，我们尝试提供一个紧凑的“无数据包”结构，该结构仅模拟网络中执行功能的实体：数据包处理规则。此外，我们以最抽象的形式建模规则：作为布尔变量，抽象掉所有规则的属性，如它们的匹配字段、动作和优先级。这与普遍的网络验证技术形成对比，后者根据数据包和数据包的等价类（ECs）建模网络行为。

布尔变量和公式提供了一种更紧凑的方式来表示转换系统的状态空间。最先进的模型检查器，如NuSMV，使用符号技术，如二进制决策图（BDDs），有效地探索具有布尔表示的转换系统。这种表示使模型检查器能够有效地探索极大的状态空间。接下来，我们将解释如何将动态网络功能的行为编码为无数据包模型。然后我们将在第4节中展示，这种无数据包建模方法在验证典型网络功能时显著减少了验证时间。

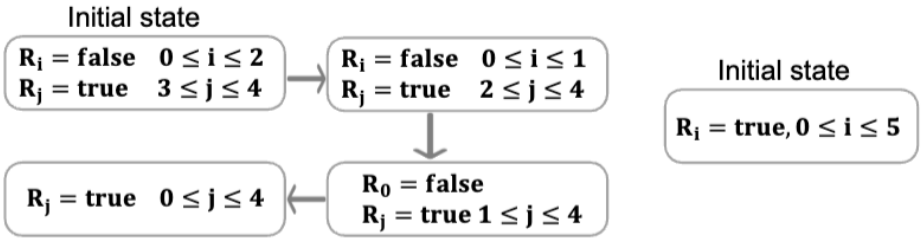


Figure 3: Packet-less models of the (a) IDPS and (b) monitor.

紧凑的无数据包模型：我们最初将网络建模为一个转换系统TS，每个规则对应一个布尔变量。这个变量如果为真，则表示规则在网络中处于活动状态；如果为假，则表示相反。作为起点，我们抽象掉了计数器，假设规则不依赖于它们。（我们稍后会细化这个模型，以纳入计数器的语义。）在这个结构中，每个节点表示网络的一个状态，该状态由布尔规则变量的取值定义。S是所有这些状态的集合。如果在某个状态中，至少有一个与之匹配的数据包，且它是该状态中值为真的最高优先级规则，则该规则可以在该状态中执行。动作是执行规则，通过添加或删除规则来

更新转发状态，转换显示状态如何因这些动作而演变。如果在网络状态$s \in S$中，规则R_i可以执行，并且其执行添加或删除规则，则存在一个转换$s \rightarrow^{R_i} s'$，其中$s' \in S$，对于每个添加的规则R_j，在s'中$R_j = \text{true}$，对于每个删除的规则R_k，在s'中$R_k = \text{false}$。在初始状态中，每个R_i的值等于原始程序中$R_i.active$的初始值。图3显示了图1和图2中IDPS和校园监控器功能的无数据包模型（不包括标签函数）。

尽管更高效，但不显式建模数据包会带来一些关键挑战：（1）抽象掉数据包的概念使得难以纳入流量统计等语义，例如计数器。（2）如§3.1所解释的，属性是在数据包上定义的，一个网络状态中是否持有一个属性取决于与数据包匹配的最高优先级规则的动作。因此，在一个抽象掉任何显式数据包、头部、优先级等概念的结构上验证属性是具有挑战性的。例如，在图1中的IDPS功能，无数据包模型在初始状态有两个规则可以匹配发送到I的端口号为2222的数据包，如图3(a)所示。用布尔变量建模每个规则时，无法确定哪个匹配规则处理了一个数据包（因此无法验证属性）。我们接下来解决这些挑战。

3.3.1 计数器的布尔公式

无数据包建模的第一个挑战是保留有状态程序的计数器语义。回想一下，在无数据包模型中，我们最初抽象掉了计数器。接下来，我们展示如何改进这些模型以纳入计数器的语义。

细化状态：为了建模计数器，我们观察到，如果一组网络状态中唯一变化的变量是计数器值，并且该计数器的值没有通过计数器条件，则所有这些状态中的转发行为保持不变。在监控功能中，对于0到v_1之间的所有计数器值，网络行为是相同的。这允许我们跟踪计数器谓词，即计数器的布尔值函数，而不是实际的计数器值。在监控功能中，我们可以定义以下三个谓词：$(0 \leq c_0 < v_1), (v_1 \leq c_0 < v_2)$和$(v_2 \leq c_0 < m)$。在初始状态中，只有第一个谓词$(0 \leq c_0 < v_1)$为真

转发行为不是由确切的计数器值决定的，而是由计数器通过阈值决定的，这使得计数器适合于谓词抽象，这是一种强大的技术，用于缓解具有大基类型（如整数）的程序验证的挑战。这种技术通过仅跟踪数据上的谓词并消除不可见的数据变量来减小模型的大小。

具体来说，计数器条件将区间划分为可能具有不同转发行为的子区间。让R_i是一个依赖于第j个计数器c_j的规则，即它处于活动状态，如果c_j的值在(l_i, u_i)范围内，并且$P_j = \text{order}(\cup_i (l_i \cup u_i))$，即所有依赖于$c_j$的规则的所有下界和上界的非递减顺序列表。$P_{j,k}, l_{j,k}$和$u_{j,k}$分别表示计数器$c_j$的第$k$个子区间及其下界和上界。在监控程序中，$P_{0,0} = [0, v_1)$是第一个计数器的第一个子区间，$l_{0,0} = 0$，$u_{0,0} = v_1$。当计数器值在此子区间中时，规则$R_0$可以处理数据包，因为其计数器条件$(l_0 \leq c_0 < u_0)$得到满足，即$(l_0 \leq l_{0,0}) \wedge (u_{0,0} \leq u_0)$，其中$l_0 = 0$和$u_0 = v_1$。另一方面，$R_2$不能处理数据包，因为其计数器条件在此子区间中不满足，即$(l_2 \leq\leq l_{0,0}) \wedge (u_{0,0} \not\leq u_2)$，其中$l_2 = v_1$和$u_2 = v_2$。

我们通过为每个规则R_i添加一个布尔变量R_i'来细化TS。我们的目标是在状态s中将此变量的值设置为true，如果R_i的计数器条件在s中得到满足，则为false。对于任何不依赖于计数器的规则R_k，$R_k' = \text{true}$。P_j中的数字是唯一可以改变依赖于c_j的规则R_i'变量的地方。在监控程序中，$P_0 = [0, v_1, v_2, m]$列出了$[0, m)$范围内唯一的点，其中依赖于计数器c_0的规则，例如R_0的条件可以从真变为假，反之亦然。

对于每个计数器c_j在无数据包模型中的状态，我们将状态划分为$|P_j|-1$个状态，其中$|P_j|$是P_j中点的数量，例如，在监控示例中，$P_0=4$。假设R_i是一个依赖于计数器c_j的规则，即R_i可以处理数据包，当计数器的值满足其计数器条件时：$l_i \le c_j < u_i$。在每个细化状态中，R'_i的值应显示规则R_i的计数器条件是否在相应的子区间$P_{j,k}$中得到满足：$R'_i=(l_i \le l_{j,k}) \wedge (u_{j,k} \le u_i)$。

在任何给定的状态$s \in S$中，对于$P_{j,k}$的一个子区间，网络行为由那些(a)在该状态中处于活动状态的规则（即$R_i=\text{true}$）和(b)不依赖于计数器或其计数器条件在该子区间中得到满足的规则决定（即$R'_i=\text{true}$），例如，在监控示例的初始状态中，R_0和R_1是活动的（即$R_0=R_1=\text{true}$）并且它们的计数器条件在第一个子区间$P_{0,0}=[0,v_1)$中得到满足（即$R'_0=R'_1=\text{true}$）。

在总结中，无数据包模型TS中的变量集包括每个规则的一对布尔变量R_i和R'_i。这些变量在任何时点的值定义了该时点无数据包模型的状态。在初始状态中，每个规则R_i的R_i值等于原始程序中$R_i.active$的初始值。如果R_i不依赖于任何计数器，或者如果它依赖于计数器c_j并且其计数器条件在c_j的第一个子区间$P_{j,0}$中得到满足，即$(l_i \le l_{j,0}) \wedge (u_{j,0} \le u_i)$，则$R'_i$在初始状态中为真。

在精炼模型中，如果规则R_i在状态中可以执行，那么必须满足：(a) 对于至少一个与之匹配的数据包，它是优先级最高的规则；(b) R_i在该状态中是活动的，即$R_i=\text{true}$；(c) R_i不依赖于任何计数器，或者其计数器条件在该子区间中得到满足，即$R'_i=\text{true}$。

在精炼模型中的转换：设s_k和s_{k+1}分别是状态s为计数器c_j精炼的第k个和$(k+1)$个精炼状态，即表示P_j的子区间k和$k+1$的布尔公式的状态。如果存在至少一个依赖于c_j的规则R_i在s_k中可以执行，则从状态s_k到状态s_{k+1}添加一个转换。原因是c_j的值是单调递增的，所以如果一个依赖于计数器的规则在s_k中可以执行，它可以导致c_j过渡到下一个子区间（对应于状态s_{k+1}）。对于原始无数据包模型中的每个转换$s \to^{\alpha} s'$，其中α是执行一个添加或删除规则的规则R_i的操作，如果R_i可以在s_k中执行，则存在一个转换$s_k \to^{\alpha} s'_k$。如前所述，对于R_i添加的每个规则R_j，在s'_k中$R_j=\text{true}$；对于R_i删除的每个规则R_k，在s'_k中$R_k=\text{false}$。

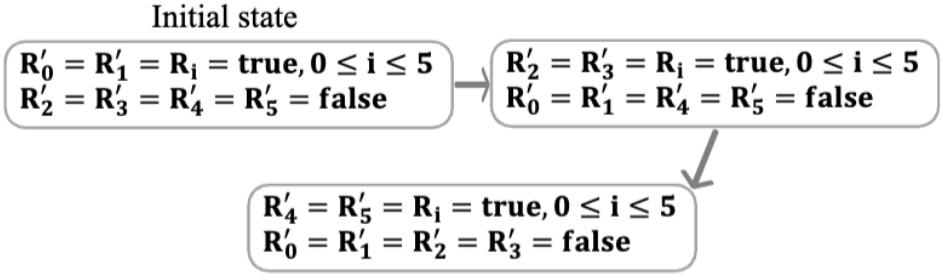


Figure 4: The refined structure for the monitor function.

图4显示了图2中监控功能的布尔无数据包状态和转换（参见图3(b)对应的未精炼无数据包模型）。从初始分区状态（其中$R'_0=R'_1=\text{true}$）到一个状态（其中$R'_0=R'_1=\text{false}$），反映了网络状态从满足这些规则的计数器条件的初始状态演变到不再满足这些规则的计数器条件的状态。如果程序有多个计数器，那么状态和转换的精炼是按计数器顺序依次进行的。

3.3.2 属性的布尔公式

我们之前解释了如何用数据包和期望动作来表达原子命题。属性是由原子命题构建的，在像之前工作中的基于数据包的转换系统中，如果原子命题(h, a)在状态s中成立，那么动作列表a适用于h中的所有数据包。无数据包建模的第二个挑战是在布尔无数据包模型上评估命题。在这部分，我们解释如何将(h, a)命题表达为利用已知规则优先级的规则上的布尔公式。

我们说一个规则R_i可以应用，如果(a) R_i是活动的，即$R_i = \mathrm{true}$，以及(b) R_i要么不依赖于任何计数器，要么其计数器条件得到满足，即$R'_i = \mathrm{true}$。为了执行一个规则，除了满足上述条件外，它应该是至少一个匹配数据包的最高优先级规则。

设$W_{0,n} = [R_0, R_1, \ldots, R_n]$和$W'_{0,n} = [R'_0, R'_1, \ldots, R'_n]$分别是规则$R_i$和变量$R'_i$的列表，按规则优先级的非递增顺序排序，即如果$i < j$，则$R_i.\,\mathrm{priority} \geq R_j.\,\mathrm{priority}$，$R'_i.\,\mathrm{priority} \geq R'_j.\,\mathrm{priority}$。我们的目标是表达一个状态中是否成立命题的布尔公式。我们通过一个递归函数实现这一点：设$K((h, a), W_{0,n}, W'_{0,n})$是一个函数，如果$W_{0,n}$和$W'_{0,n}$满足命题$(h, a)$则为真，否则为假。对于两个特殊情况，其中$(1)$ h为空和(2) h非空且规则列表为空，我们假设$K((h, a), W_{0,n}, W'_{0,n})$分别评估为真和假，因为任何条件对于不存在的数据包都成立(上述第1项，$h = \varnothing$)，而一组空的规则不满足数据包的任何条件(上述第2项，$h \neq \varnothing$且规则列表$=[]$)。在其他条件下，我们有以下情况：

情况1：如果最高优先级规则R_0匹配h中的一些数据包，即如果$(h \cap R_0.\,\mathrm{pkts}) \neq \varnothing$，其中$R_i.\,\mathrm{pkts}$表示$R_i$匹配的数据包集合，并且其动作包括命题中的动作，即$a \subset R_0.\,\mathrm{action}$，那么命题成立的条件是这两个条件之一成立：$(\mathrm{a})$ 要么R_0可以应用$(R_0 \wedge R'_0)$，并且对于所有不匹配R_0的h中的数据包，命题应该对下一个优先级较低的匹配规则成立，即
$(R_0 \wedge R'_0) \wedge K((h - R_0.\,\mathrm{pkts}, a), W_{1,n}, W'_{1,n})$，或者$(\mathrm{b})$ R_0不能应用$(\neg(R_0 \wedge R'_0)$，R_0要么没有在网络中安装，要么其计数器条件不满足)，但在这种情况下，对于h中的所有数据包，命题应该对下一个优先级较低的匹配规则成立，即$\neg(R_0 \wedge R'_0) \wedge K((h, a), W_{1,n}, W'_{1,n})$。换句话说，
$K((h, a), W_{0,n}, W'_{0,n}) = (R_0 \wedge R'_0) \wedge K((h - R_0.\,\mathrm{pkts}, a), W_{1,n}, W'_{1,n}) \vee \neg(R_0 \wedge R'_0) \wedge K((h, a), W_{1,n}, W'_{1,}$
。

情况2：如果最高优先级规则R_0匹配h中的一些数据包，即如果$(h \cap R_0.\,\mathrm{pkts}) \neq \varnothing$，并且其动作不包括命题中的动作，那么命题成立的条件是不可能应用R_0，即$\neg(R_0 \wedge R'_0)$。否则，它匹配数据包但不应用预期的动作。此外，对于h中的所有数据包，命题应该对下一个优先级较低的匹配规则成立，即$K((h, a), W_{0,n}, W'_{0,n}) = \neg(R_0 \wedge R'_0) \wedge K((h, a), W_{1,n}, W'_{1,n})$。

情况3：如果最高优先级规则R_0不匹配h中的任何数据包，那么无论R_0是否可以应用，对于h中的所有数据包，命题应该对下一个优先级较低的匹配规则成立，即
$K((h, a), W_{0,n}, W'_{0,n}) = K((h, a), W_{1,n}, W'_{1,n})$。

通过递归地应用于网络中所有规则的有序列表(根据优先级)，$K((h, a), W_{0,n}, W'_{0,n})$将命题$(h, a)$表达为$R_i$和$R'_i$变量的布尔公式。例如，在图1中的IDPS功能中，命题$((\mathrm{src} = I, \mathrm{dst} = E), \mathrm{send}())$被翻译为以下布尔公式：

$$
\begin{aligned}
K(((\text{src}{=}I, \text{dst}{=}E), \text{send}()), W_{0,4}, W'_{0.4}) = \\
(R_0 \wedge R'_0) \wedge (K(((\text{src}{=}I, \text{dst}{=}E), \text{send}()), W_{1,4}, W'_{1,4})) = \\
\neg(R_0 \wedge R'_0) \wedge K(((\text{src}{=}I, \text{dst}{=}E), \text{send}()), W_{2,4}, W'_{2,4}) = \\
= \\
\neg(R_0 \wedge R'_0) \wedge K(((\text{src}{=}I, \text{dst}{=}E), \text{send}()), W_{4,4}, W'_{4,4}) = \\
\neg(R_0 \wedge R'_0) \wedge \\
(((R_4 \wedge R'_4) \wedge K((\varnothing, \text{send}()), []))\vee \\
(\neg(R_4 \wedge R'_4) \wedge K(((\text{src} = I, \text{dst} = E), \text{send}()), []))) = \\
\neg(R_0 \wedge R'_0) \wedge (R_4 \wedge R'_4).
\end{aligned}
$$

这个布尔公式是根据情况（2）、（3）、（3）、（3）、（1）以及前两个特殊情况指定的规则得出的。在图4中不同状态的真值不同，取决于那些状态中规则的真值，例如，在初始状态S_0中，$R_0 = \text{false}$且$R_4 = \text{true}$，意味着断言（$(\text{src} = I, \text{dst} = E)$, $\text{send}()$）成立（在这个状态中I可以与E通信），但在最终状态（I被阻止）中，$R_0 = \text{true}$且$R_4 = \text{true}$，此时为false。注意，在这个例子中R'_i变量始终为true，因为规则不依赖于计数器。

在网络转换系统TS中，标签函数L将每个状态映射到在该状态中成立的原子命题集合。也就是说，如果动作列表a适用于h中的所有数据包，则(h, a)存在于$L(s)$中，即$K((h, a), W_{0,n}, W'_{0,n}) = \text{true}$，其中$W_{0,n}$和$W'_{0,n}$分别是$s$中规则$R_i$和变量$R'_i$的列表（按规则优先级的非递增顺序排序）。

4 实施和评估

为了评估我们设计的性能，我们构建了一个原型，使网络运营商能够编程和验证他们的功能，并且开发了一个编译器，将这些功能转换为可在可编程交换ASIC上执行的程序。在简要概述我们的原型之后，我们将展示我们的网络抽象和规范语言是如何表达的，并且只引入了最小的开销。我们还将展示与基于数据包的基线相比，无数据包模型验证不同属性的速度更快、更具可扩展性，例如，对于一个有100个主机的网络，无数据包模型在验证UDP洪水缓解功能的生存性属性时，比基于数据包的模型快8倍。

4.1 实施

接口：我们的系统提供了两个接口：一个是允许网络运营商在我们的抽象上编程其功能的一大开关接口（§2），另一个是允许他们表达所需属性的规范接口（§3）。然后我们的生成器自动构建无数据包模型和表示规范的布尔公式，如（§3）中所解释的，并与NuSMV这一最先进的模型检查器交互，以验证规范属性。

将抽象编译为P4程序：P4是一种用于表达可编程数据平面的数据包处理的语言。与可编程数据平面一起，控制平面负责填充P4程序定义的表。我们构建了一个编译器，将使用我们抽象编写的一大开关程序编译为P4_16程序，用于P4行为模型，这是一个开源的可编程软件交换机。与软件交换机一起，我们开发了一个控制平面，它添加和删除表规则。我们描述了编译的一些显著特点：

(a) 功能分解：我们将抽象中的每个表映射到一个P4表，其匹配字段和动作是使用表中的规则构建的。网络功能遍历策略使用P4控制流构造实现，例如，有条件地遍历防火墙表fw：

```
if (meta.visit_fw == 1) {
    fw.apply();
}
```

其中meta.visit_fw是由防火墙表之前的表使用的元数据变量，以确保数据包通过防火墙表。

(b) 可操作的测量：我们使用P4的寄存器来支持增加和匹配跨多个规则共享的计数器。如果抽象中的表使用计数器，我们创建一个单独的P4表，负责更新共享计数器并将计数器状态转移到元数据，以便功能表可以使用该值进行匹配。这有助于我们将寄存器访问限制在单个表中。因此，数据包处理可以以线速进行。

(c) 添加/删除动作：当前，P4数据平面不支持添加/删除动作，即添加或删除其他表规则的规则动作，这是由于现有平台的硬件限制。我们通过克隆数据包到控制平面来支持此功能，类似于OpenFlow中的PacketIn功能。当程序中的规则具有添加/删除动作时，我们的交换程序在数据平面克隆数据包并将其发送到本地交换机控制平面；控制平面（我们也生成的）然后根据程序中的添加/删除动作添加/删除表规则。这种方法的开销是将一些数据包（特别是在连接中匹配特定规则的第一个数据包）转发到本地交换机CPU——这是我们为了缺乏数据平面对添加/删除动作的支持而支付的代价。

Function source	Function	One-big-switch LoC	Packets calling controller [%]	P4 LoC	Compilation time[ms]
Pyretic [44]/Kinetic [35]	Simple counter	1	0	144	1.8
	Port knocking	3	6×10^{-3}	133	1.8
	Simple firewall	2	0	128	1.6
	IP rewrite	2	0	134	1.8
	Simple rate limiter	3	0	141	2.1
Floodlight [4]	Firewall/ACL	2	0	132	1.7
Chimera [13]	Phishing/Spam detection	3	6×10^{-3}	151	2.3
FAST [45]	Simple stateful firewall	3	0.3	133	2.1
	FTP monitor	2	0	135	2.1
	Heavy-hitter detection	2	0	148	2.0
	Super spreader detection	2	0	148	2.0
	Sampling based on flowsize	6	0	189	2.5
Bohatei [23]	Elephant flow detection	3	0	182	2.4
	DNS amplification mitigation	3	7×10^{-3}	135	1.9
	UDP flood mitigation	2	0	148	1.8

Table 1: Applications written on one-big-switch.

4.2 评估

表达性：尽管我们的一大开关抽象很简单，但它使开发者能够表达广泛的应用程序和网络功能。表1显示了我们在框架中开发的功能列表。这些程序的完整描述在（§8）中提供。网络策略可以在我们的一大开关抽象上仅用几行代码简洁地表达（表1中的第3列），例如，一个简单的有状态防火墙策略，只允许由给定部门的主机发起连接的流量，可以在一大开关抽象上用3行代码表

达。同一策略的编译P4代码用133行代码表达（包括100行用于头和解析器的样板代码）和50行代码用于P4控制平面，用于在数据平面克隆数据包并从控制平面添加规则。各种规范属性，包括实践中最常见的规范模式，也可以在（§3）中介绍的时态逻辑中指定。

有限的开销：我们的抽象中的添加/删除动作目前在交换ASIC中不直接支持。我们的P4编译器通过在规则具有此类动作时涉及控制器来实现这些动作。为了测量涉及控制器的开销，我们部署了表1中列出的功能，并回放了一个大学数据中心的数据包跟踪，包含超过102K个数据包和1791个IP地址，并测量了调用控制器的频率。对于所有功能来说，这种开销都是适度的，即0-0.3%（表1中的第4列）。

验证时间：我们测试了有界时间验证无数据包和基于数据包模型的可扩展性，以回答诸如哪些功能和属性可以使用每种方法验证、验证时间如何随网络（因此模型）大小扩展、以及它如何随属性大小扩展等问题。为此，对于表1中具有每个主机策略的功能（例如，重点用户检测器、端口敲击、速率限制器、网络钓鱼/垃圾邮件检测器和UDP洪水缓解功能），我们为网络中的每个主机定义一个策略。例如，重点用户检测器部署了一个每个新SYN数据包都会增加的每源IP计数器，并在计数器超过阈值时开始丢弃数据包（表4）。对于在通信主机对或流上定义策略的功能，例如FTP监控和DNS放大缓解功能，我们运行上述相同的数据中心跟踪以找到通信主机对和匹配流，并为每个定义一个策略。对于将主机分类为集合的功能，例如具有内部和外部主机集的防火墙功能，我们随机将每个主机分配到一个集合中。最后，对于计数器界限，我们从可能值集合上的均匀分布中抽取随机样本。

我们测量了各种功能、属性和网络规模的验证时间，对于无数据包和基于数据包的模型。请注意，以这种方式增加网络大小会导致更大的模型。对于每个实验，我们进行了20次重复试验，每次试验的时间预算为1000秒，即当验证时间超过1000秒时，我们停止验证过程。在展示我们的结果之前，我们将简要概述我们的基于数据包的基线。

基于数据包的转换系统作为基线：扩展静态验证的一种强大技术是将网络切分为一组数据包的等价类（ECs）。每个EC是一组始终在网络中经历相同转发动作的数据包。如先前的工作所示，这种方法可以扩展到模型动态网络。为此，可以使用将数据包分类为ECs的验证器来检测程序的ECs。或者，类似于Kinetic，程序员可能需要提供事件条件（ECs）及其转换系统。

在基于数据包的转换系统中，每个节点代表网络的一个状态，即该状态下的等价类（ECs）。例如，在图1中的IDPS中，初始状态下网络中存在两个ECs：EC0包括所有目的地为I且端口号为2222的数据包，以及EC1包括所有其余数据包。转换是改变网络状态的事件，例如，从ECs接收数据包，其转发动作更新网络的转发行为。在上述示例中，从EC0接收数据包会使系统转换到另一个状态，在该状态中存在三个不同的ECs：EC2包括所有从I发往F的数据包，EC3包括所有不包含在EC2中且发送到I端口号为2222的数据包，以及EC4包括所有不包含在EC2和EC3中的数据包。我们将这种基于数据包的模型作为我们的基线实现。为了在这个基线中将数据包分类为ECs，我们使用VeriFlow开发的启发式方法。

尽管基于数据包的模型和Kinetic在概念上有相似之处，但它们之间存在一些关键差异：我们编程抽象的更大表达性（例如，允许匹配共享数据包计数器）增加了验证问题的难度。此外，与

Kinetic不同，Kinetic要求操作员提供状态机作为输入，而基于数据包的模型会自动从我们的一大开关抽象上编写的规则生成这些状态机。因此，尽管它们在概念上相似，我们避免将我们的基线称为Kinetic。我们的结果表明，与这些提案中部署的基于数据包的方法相比，我们的无数据包方法在验证速度上更快。

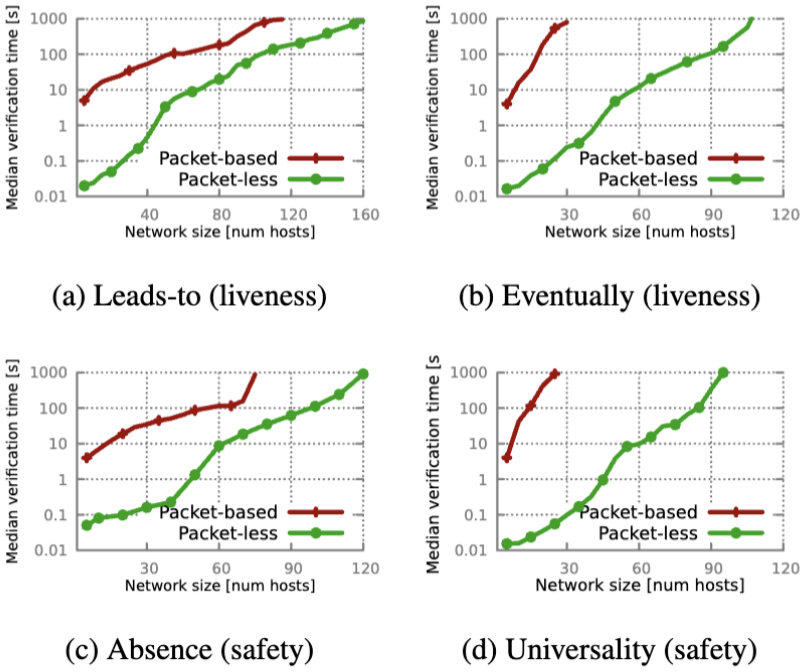


Figure 5: Packet-less verification reduces the verification time of different properties for a flood mitigation function.

我们的无数据包模型比基于数据包的模型在验证不同类别的活性属性（如"leads-to"，例如，主机A发送流量到主机B导致A被阻止）和"eventually"（例如，主机A最终被阻止）时更快。例如，在一个有100个主机的网络中，验证一个leads-to属性——主机A发送的数据包超过阈值导致其被阻止——在无数据包模型中比基于数据包的模型快7.8倍（85秒对比667秒）。在1000秒的时间限制内，无数据包模型可以验证一个不同的活性属性"eventually"（A最终被阻止），对于比基于数据包模型可验证的网络大3.5倍的网络（105个主机对比30个）。通过减小状态机的大小，无数据包模型还提高了安全属性的验证时间。图5（c）和（d）分别展示了一个"absence"属性（主机A永远不可达）和"universality"（A始终可达）的示例。同样，验证相同的活性和安全属性，例如"eventually"和"universality"，在一个有30个主机的网络中的状态防火墙（表13）分别是无数据包模型比基于数据包模型快3.3倍和4.9倍（图未显示）。

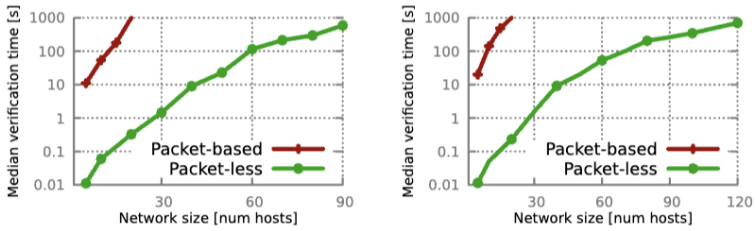
我们观察到，规则之间更大程度的状态共享（例如，多个规则共享的计数器）导致两种方法的验证速度都变慢，但性能下降在基于数据包的模型中更为明显，例如，对于一个速率限制器（表5），在1000秒内，我们可以使用无数据包模型验证一个"eventually"属性的网络，最多可达90个主机（与上面的UDP洪水检测应用相比，为105个主机），而对于基于数据包的模型，最多只能验证15个主机的网络（与上面的UDP洪水检测应用相比，为30个主机）。图6展示了这个功能的一个活性和一个安全属性的结果。无数据包模型可以验证比基于数据包模型大6倍的网络，即使对于小规模网络，它也至少快两个数量级。

Rule	Init	Priority	Match	Action
R_0	true	100	(TCP flag=SYN) & (source IP=A) & $(0 \leq c_0 < X)$	send()
R_1	true	100	(TCP flag=SYN) & (source IP=A) & $(X \leq c_0 < m)$	drop()

Table 4: A heavy-hitter detection function.

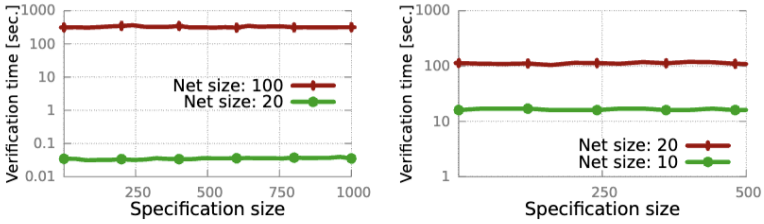
Rule	Init	Priority	Match	Action
R_0	true	100	(source IP=A) & $(0 \leq c_0 < v_1)$	send(port=1)
R_1	true	100	(source IP=A) & $(v_1 \leq c_0 > v_2)$	send(port=2)
R_2	true	100	(source IP=A) & $(v_2 \leq c_0 < v_3)$	send(port=3)
R_3	true	100	(source IP=A) & $(v_3 \leq c_0 < v_4)$	send(port=4)
R_4	true	100	(source IP=A) & $(v_4 \leq c_0 < v_5)$	send(port=5)
R_5	true	100	(source IP=A) & $(v_5 \leq c_0 < m)$	drop()

Table 5: A simple rate limiter.



(a) Eventually (b) Universality

Figure 6: Verification times for a rate limiter.



(a) Packet-less (b) Packet-based

Figure 7: The property size does not impact the verification time.

除了测试与网络规模（因此模型规模）相关的可扩展性外，我们还扩展了属性。验证时间是转换系统大小和属性公式大小的函数，例如，存在一个LTL模型检查算法，其运行时间与转换系统的大小成线性关系，与LTL公式的长度成指数关系。尽管在实践中，结构大小通常是决定验证时间的主要因素，但在我们的方法中，属性公式的大小可能很大，增加了验证时间。为了测试验证时间对属性大小的敏感性，我们首先为随机选择的（有替换的）主机构造原子命题，例如（dst=A, send(A)）。然后，我们使用逻辑运算符根据这些原子命题构建各种大小的公式，如§3中解释的，例如（dst=A, send(A)）∧（dst=B, send(B)）对于大小为2的属性。最后，我们使用这些断言构造表3中列出的活性和安全属性。我们观察到，扩大属性大小并不影响任一方法的验证时间。对于上述相同的UDP洪水缓解功能，在一个有200个主机的网络中，将属性大小从1增加到200，结果的标准偏差小于7.2。即使对于较小的网络，其中属性大小的相对影响预计更大，这一

点也成立。图7展示了两个示例，用于验证$\mathrm{F}(\wedge_{h\in S}(\mathrm{dst}=h,\mathrm{send}(h))$，即最终，集合S中的所有主机都将变得可达，对于有10、20和100个主机的小型网络。

Rule	Init	Priority	Match	Action
R_0	true	100	(protocol=UDP) & (source IP=A) & ($c_0 < X$)	send()
R_1	true	100	(protocol=UDP) & (source IP=A) & ($X \leq c_0 < m$)	drop()

Table 2: A UDP flood mitigation function.

Property	LTL specification [17,21]	Meaning and example
Leads-to (a.k.a Response)	$\mathtt{G}(P \rightarrow \mathtt{F}S)$	P must always be followed by S, a host showing a malicious activity must always be followed by the IDPS blocking the host.
Universality	$\mathtt{G}P$	P always holds, *e.g.,* A is always blocked.
Absence	$\mathtt{G}\neg P$	P never holds, *e.g.,* A can never send traffic to B.
Eventually (a.k.a Existence)	$\mathtt{F}P$	P eventually happens, *e.g.,* A can eventually reach B.
Precedence	$\mathtt{F}P \rightarrow (\neg P \mathtt{U}(S \wedge \neg P))$	P must always be proceeded by S, *e.g.,* blocking a host must always be proceeded by the host exhibiting some malicious activity.

Table 3: List of properties verified.

5 相关工作

静态网络验证器验证了网络快照上的各种可达性不变性，例如无环和无黑洞。最近，可达性验证和执行技术被扩展到包括动态性，例如在故障和策略变化、可变数据平面以及关注控制器而非数据平面快照的情况下。然而，所有这些提议中的目标属性都是安全不变性，如可达性和无环。我们专注于验证计算上更复杂的属性类别（活性），这推动了我们独特的无数据包模型，与以前的模型不同。此外，以前工作中的一些假设限制了它们可以验证的网络功能集。例如，VMN模型了网络功能，其中（a）流是独立的，（b）转发不受事务排序的影响。我们发现许多关键的网络功能，如入侵检测系统和木马检测器，并不具备这些属性。

我们与VeriCon、FlowLog 和 Kinetic 设计可验证编程语言的目标相同，但由于我们的程序被编译为P4程序（而不是VeriCon的CSDN语言、FlowLog和Kinetic程序被编译为的OpenFlow规则），我们能够表达这些框架无法表达的程序，例如，具有多个共享计数器的规则的程序。此外，VeriCon使用一阶逻辑，使得指定动态属性如活性变得不可行。最后，VeriCon、FlowLog和Kinetic都是基于数据包的。例如，Kinetic扩展了Pyretic控制器，增加了基于数据包等价类验证动态网络的支持。在Kinetic中，程序员需要指定“位于数据包等价类”（LPECs），即流空间的最大区域（例如，具有给定源IP的数据包），在每个状态中经历相同的转发行为，并且与编码LPECs处理的有限状态机（FSMs）相关联。我们通过实验和理论证明，与这些提议中部署的基于数据包的方法相比，我们的无数据包方法验证速度更快。

许多最近的网络验证项目利用模型检查的经典概念来控制验证动态系统的状态爆炸挑战，例如切片、符号执行和抽象细化。我们的工作在高层技术上与这些工作相似，用于扩展验证（例如，我们也使用符号建模和抽象）。然而，由于我们的目标属性（活性与安全不变性）不同，我们对这些技术的应用与现有工作有所不同，例如，我们部署符号表示不是为了抽象数据包头字段（如NoD在验证动态网络中的可达性不变性时所做的），而是为了完全抽象掉数据包。通过测试和模

拟验证网络的工作与我们的方法互补。通过应用模型检查，我们努力提供一个完全自动的验证器，与测试和模拟不同，它可以彻底搜索我们抽象的状态空间。

6 限制和未来工作

我们的工作重点是功能正确性；这排除了大量的功能和属性集，包括那些关注路径和流量工程属性的功能（路径是否拥堵？负载是否在多路径上平衡？等）。我们的一大开关抽象不适合编程此类功能。此外，尽管对操作员来说是一个熟悉的抽象，但一大开关抽象相对较低级。未来研究的一个有趣方向是开发适合高效活性验证的更高级别抽象。我们的验证器不是"全栈"的；它没有设计来验证低级属性，如内存安全和崩溃自由，这些属性可以通过像Vigor和VigNAT这样的工具来验证，也不验证编译后的P4代码。因此，像编译器验证器（如p4v）、P4调试器（例如，Vera）和测试器仍然是必需的，以保证我们验证的抽象的忠实实现，例如，检测和调试交换机固件和P4编译器错误。最后，虽然我们的无数据包建模改善了与基于数据包模型相比的验证时间，并使得验证复杂属性如活性成为可能，但对于大规模网络，绝对的验证时间仍然很高（注意图中的对数刻度）。进一步降低有状态功能的验证复杂性仍然是一个开放的挑战。

7 结论

现代网络依赖于各种有状态网络功能来实现丰富的策略。这样的网络的正确操作依赖于确保它们支持关键的活性属性。不幸的是，尽管最近在网络验证方面有令人兴奋的进展，但没有现有的方法是实用的，或适用于验证活性。我们采取自上而下的方法来解决这个问题，首先设计了一个以验证为目的的新编程模型。它提供了对方便的一大开关抽象的自然扩展，并允许不同网络功能的分解。我们开发了一种新的编码方式，使用布尔公式在动态事件（如网络状态变化）下编码这些程序，这些公式可以捕捉丰富的语义（例如，计数器），同时确保编码保持有界大小并适合快速活性验证。我们开发了一个编译器，将我们的程序转换为可在现代硬件上运行的程序。我们的评估显示，编程模型可以简洁地表达各种功能，我们的编译速度快，我们的编码紧凑，并且在验证上比简单编码具有数量级更高的可扩展性，即它导致了显著的验证加速。

Rule	Init	Priority	Match	Action
R_0	true	100	(source IP=A) & (destination IP=B) & ($0 \leq c_0 <$ m)	send()

Table 6: Simple counter.

Rule	Init	Priority	Match	Action
R_0	true	100	(source MAC=M) & (destination port=K)	add(R_1),add(R_2)
R_1	false	100	(source MAC=M) & (destination port=O)	send()
R_2	false	100	(destination MAC=M) & (destination port=O)	send()

Table 7: Port knocking.

Rule	Init	Priority	Match	Action
R_0	true	100	(source MAC=A) & (destination MAC=B)	send()
R_1	true	100	(source MAC=B) & (destination MAC=A)	send()

Table 8: Simple firewall.

Rule	Init	Priority	Match	Action
R_0	true	100	(source IP=A)	modify(source IP=X),send()
R_1	true	100	(destination IP=X)	modify(destination IP=A),send()

Table 9: IP rewrite.

Rule	Init	Priority	Match	Action
R_0	true	100	(source IP=A) & ($0 \leq c_0 < v_1$)	send()
R_1	true	100	(source IP=A) & ($v_1 \leq c_0 < v_2$)	send(rate limiter)
R_2	true	100	(source IP=A) & ($v_2 \leq c_0 <$ m)	drop()

Table 10: Simple rate limiter 2.

Rule	Init	Priority	Match	Action
R_0	true	100	(source IP=A1) & (destination IP=B1) & (source MAC=M1) & (destination MAC=N1) & (source port=P1) & (destination port=Q1)	send()
R_1	true	100	(source IP=A2) & (destination IP=B2) & (source MAC=M2) & (destination MAC=N2) & (source port=P2) & (destination port=Q2)	drop()

Table 11: Floodlight firewall/ACL.

Rule	Init	Priority	Match	Action
R_0	true	100	SMTP.MTA=A	add(R_1),add(R_2),delete(R_0),send()
R_1	false	100	(SMTP.MTA=A) & $0 \leq c_0 <$ X	send()
R_2	false	100	(SMTP.MTA=A) & X $\leq c_0 <$ m	drop()

Table 12: Phishing/spam detection.

Rule	Init	Priority	Match	Action
R_0	true	100	(source IP=I) & (destination IP=E)	add(R_1),delete(R_0),send()
R_1	false	100	(destination IP=I) & (source IP=E)	send()
R_2	true	50	source IP=I	drop()

Table 13: Simple stateful firewall.

Rule	Init	Priority	Match	Action
R_0	true	100	(destination port=$port_{control}$) & (source IP=A) & (destination IP=B)	add(R_1),delete(R_0),send()
R_1	false	100	(source port=$port_{data}$) & (source IP=B) & (destination IP=A)	send()

Table 14: FTP monitor.

Rule	Init	Priority	Match	Action
R_0	true	100	(source IP=A) & (destination IP=B) & $(0 \leq c_0 < v_1)$	modify(flow=small),send(Sampler Table 2)
R_1	true	100	(source IP=A) & (destination IP=B) & $(v_1 \leq c_0 < v_2)$	modify(flow=medium),send(Sampler Table 2)
R_2	true	100	(source IP=A) & (destination IP=B) & $(v_2 \leq c_0 <$ m)	modify(flow=large),send(Sampler Table 2)

Table 15: Sampling based on the flow size - Sampler Table 1.

Rule	Init	Priority	Match	Action
R_0	true	100	(source IP=A) & (destination IP=B) & (flow=small) & $(c_0 = 5)$	$c_0 = 0$,send()
R_1	true	100	(source IP=A) & (destination IP=B) & (flow=medium) & $(c_0 = 500)$	$c_0 = 0$,send()
R_2	true	100	(source IP=A) & (destination IP=B) & (flow=large) & $(c_0 = 50000)$	$c_0 = 0$,send()

Table 16: Sampling based on the flow size - Sampler Table 2.

Rule	Init	Priority	Match	Action
R_0	true	100	(source IP=A) & (destination IP=B) & (destination port=53)	delete(R_0),add(R_2),add(R_3),send()
R_1	true	10	source port=53	drop()
R_2	false	100	(source IP=A) & (destination IP=B) & (destination port=53)	send()
R_3	false	100	(source IP=A) & (destination IP=B) & (source port=53)	send()

Table 17: DNS amplification mitigation.