中国科学技术大学
University of Science and Technology of China

SIMPLE-fying Middlebox Policy Enforcement Using SDN

Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang,
Rui Miao, Vyas Sekar, Minlan Yu

SIGCOMM 2013

授课教师：赵功名

中国科大计算机学院

2025年秋·高级计算机网络

Outline

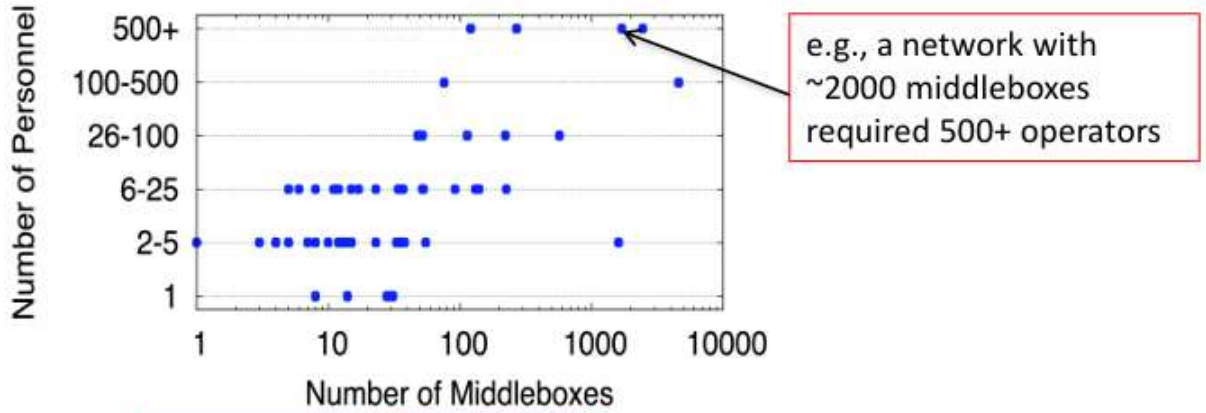
I. Introduction
II. Opportunities and Challenges
III. Simple System Overview
IV. Simple Data Plane Design
V. Resource Management
VI. Simple Dynamics Handler
VII. Implementation
VIII. Evaluations
IX. Review

Introduction

- Middleox（中间件）是一种计算机网络设备，用于转换、检查、过滤或以其他方式操纵流量，常见的设备有：
  - 防火墙：根据预定义的安全规则过滤流量，防止不想要的或恶意的流量进入网络。
  - 网络地址转换器（NAT）：替换数据包的源和/或目标IP地址，允许多个终端主机共享一个公共IP地址。
  - 入侵检测系统（IDS）：监视流量并收集数据，以便对安全异常进行离线分析。
  - 广域网优化器：提高带宽利用率和减少延迟。

Introduction

- 对于中间件的管理很难！
- 对来自57个运营商的调查结果显示 (J. Sherry et al. SIGCOMM 2012)：



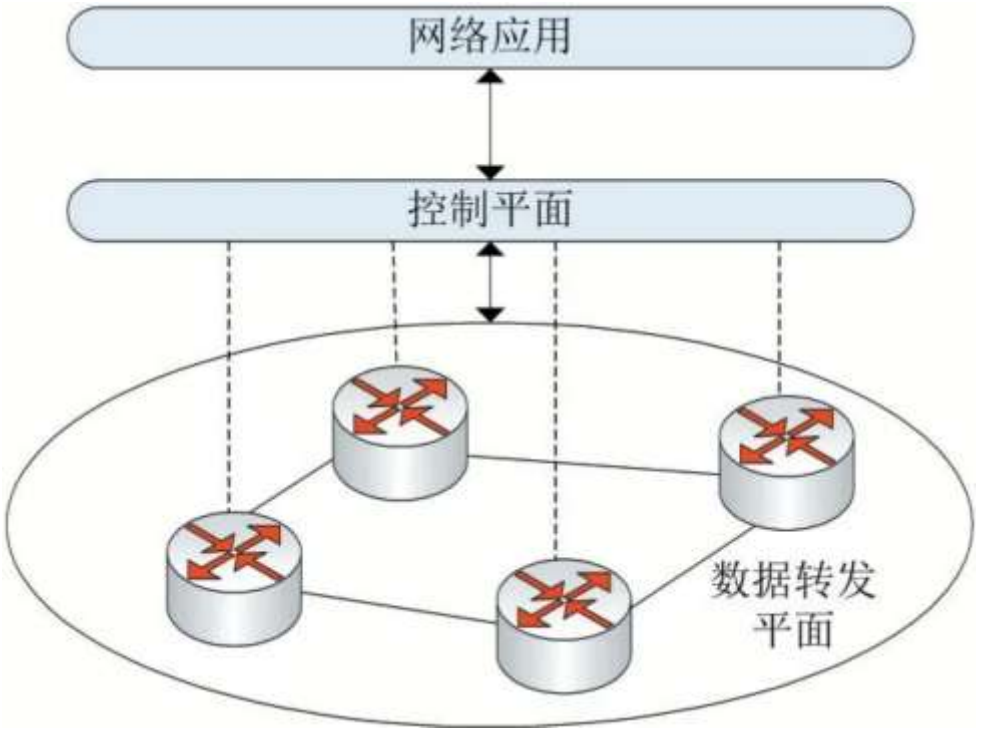
	Misconfig.	Overload	Physical/Electric
Firewalls	67.3%	16.3%	16.3%
Proxies	63.2%	15.7%	21.1%
IDS	54.5%	11.4%	34%

- 随着中间件数量的增加，所需的人员数量也急剧增加（例如，一个拥有约2000个中间件的网络需要500多名操作员）
- 中间件设备经常发生配置错误、过载、物理/电气故障
- 对安全、性能、合规性至关重要，但**昂贵、复杂且难以管理**

Introduction

- 软件定义网络（SDN）通过使用逻辑集中式管理、分离数据平面和控制平面以及提供可编程配置转发规则的能力，为中间件策略实施提供了一种很有前景的方案

  - 控制平面：拥有全局网络拓扑、实时状态和流量信息。如路径计算、访问控制策略、负载均衡策略等
  - 数据平面：根据控制器下发的“流表” 规则，对数据包进行极速匹配和转发



Introduction

- 虽然SDN为中间件管理带来了希望，但中间件本身也给SDN带来了新的难题，这些难题超出了SDN最初设计的L2/L3（二/三层）网络功能的范畴

- 挑战一 Composition (组合)：网络策略通常要求数据包经过一系列中间件（例如，防火墙+入侵检测系统+代理）。SDN能够消除人工规划中间盒部署位置和配置路由的需求，从而**简化策略执行**。然而，若仍采用面向L2/L3应用的基于流的转发规则，可能导致交换机TCAM资源的**低效利用**（例如需安装数千条规则），甚至出现**错误的转发决策**（例如同一数据包需被多个中间盒处理的情况）

Introduction

- 虽然SDN为中间件管理带来了希望，但中间件本身也给SDN带来了新的难题，这些难题超出了SDN最初设计的L2/L3（二/三层）网络功能的范畴

- 挑战二 Load Balancing（负载均衡）：由于中间件运行复杂的数据包处理（例如深度包检测），中间件部署中的一个关键因素是平衡处理负载以避免过载。SDN提供了在网络中实施负载均衡算法的灵活性，避免了运营商手动安装流量分割规则或使用自定义负载均衡解决方案的需求。不幸的是，**SDN交换机中有限的TCAM空间**使得生成此类规则以平衡中间件负载的问题在理论和实践上都难以解决

Introduction

- 虽然SDN为中间件管理带来了希望，但中间件本身也给SDN带来了新的难题，这些难题超出了SDN最初设计的L2/L3（二/三层）网络功能的范畴

- 挑战三 Packet Modifications (数据包修改)：中间件会修改数据包头（例如网络地址转换设备），甚至会改变会话级别的行为（例如广域网优化器和代理会使用持久连接）。如今，运营商必须通过精心部署或手动推断这些修改对路由配置的影响来应对这些情况。通过采用网络全局视角，软件定义网络（SDN）可以消除这一繁琐过程中的错误。然而，由于中间件的专有性质，SDN控制器在设置考虑此类转换的转发规则时，**可见性有限**

Introduction

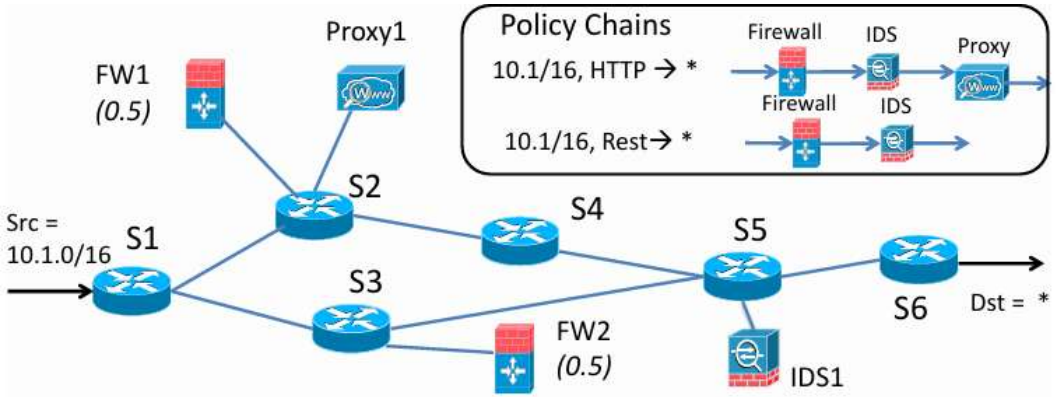
- 因此本文的目标是：设计一个基于SDN的策略执行层，用于特定中间件的流量引导。SIMPLE允许网络运营商指定逻辑中间件路由策略，并自动将其转换为考虑物理拓扑、交换机容量以及中间件资源限制的转发规则。SIMPLE 的设计包含三个关键组成部分：
  - 对组合的高效数据平面支持：本文采用了交换机之间的隧道、以及利用SDN功能在数据包头部添加标签，用这些标签为每个数据包标注其处理状态
  - 实用的统一资源管理：我们将难以处理的优化问题分解为一个复杂的离线组件和一个高效的在线组件，前者用于处理由交换机容量引入的整数约束，后者则根据流量变化平衡中间件负载
  - 中间件修改学习机制：我们利用SDN交换机的报告功能，设计了轻量级的流关联机制，以应对最常见的中间件引发的数据包转换

Outline

I. Introduction
II. Opportunities and Challenges
III. Simple System Overview
IV. Simple Data Plane Design
V. Resource Management
VI. Simple Dynamics Handler
VII. Implementation
VIII. Evaluations
IX. Review

Opportunities and Challenges

- Middlebox composition：典型的中间件策略要求packet (or session)遍历一系列中间件。（这是“服务链”这一更广泛概念的一个实例。）在我们的示例中，管理员希望将所有 HTTP 流量路由到策略链“防火墙-IDS-代理”，其余流量则路由到策略链“防火墙-IDS”。请注意，许多中间件都是有状态的，需要处理session的两个方向以确保正确性

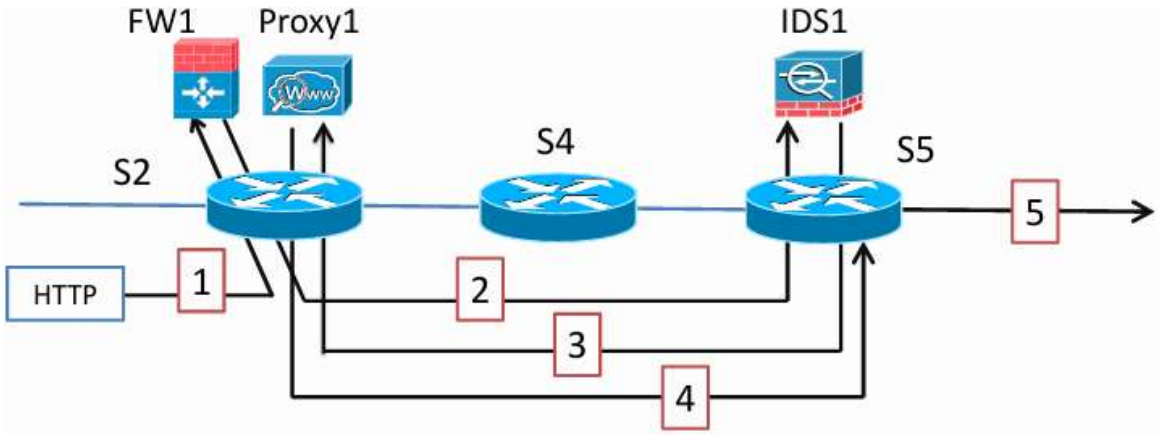


	Physical Sequence
FW1-IDS1-Proxy1	S1 S2 FW1 S2 S4 S5 IDS1 S5 S4 S2 Proxy1 S2 S4 S5 S6
FW2-IDS1-Proxy1	S1 S3 FW2 S3 S5 IDS1 S5 S4 S2 Proxy1 S2 S4 S5 S6
FW1-IDS1	S1 S2 FW1 S2 S4 S5 IDS1 S5 S6
FW2-IDS1	S1 S3 FW2 S3 S5 IDS1 S5 S6

11

Opportunities and Challenges

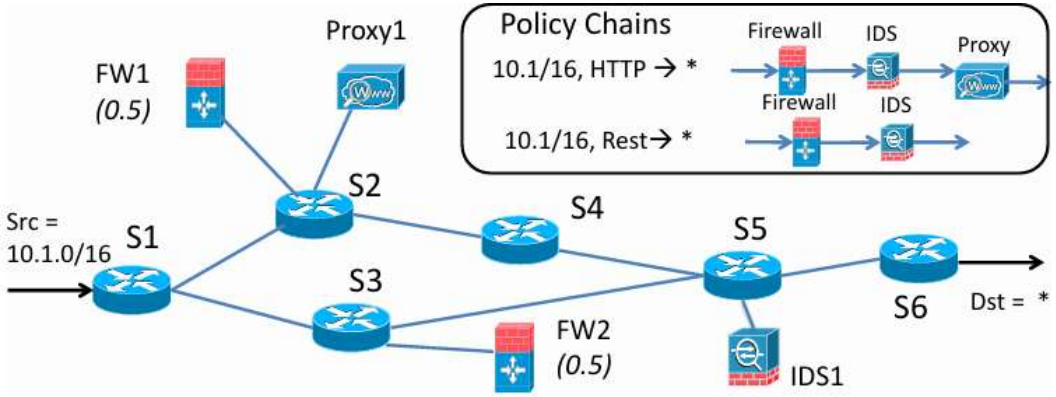
- Opportunity：中间件通常被放置在手动设置的瓶颈处，并经过精心设计，以确保状态遍历。与这种半手动且容易出错的过程相比，**SDN 可以通过编程方式确保中间件遍历的正确性**。SDN 允许管理员专注于他们需要实现的策略，而无需担心在何处执行这些策略，实现更加灵活
- Challenge：同一个SDN交换机多次看到同一个数据包，但每次都需要做出不同的转发决策，而仅凭数据包的头部信息（如IP地址和端口号）是无法区分这些情况的
  - 对于交换机S5来说：如果规则是“转发到IDS1”，那么第4步就会出错，数据包会陷入在S5和IDS1之间的循环；如果规则是“转发到S4”，那么第3步就会出错，数据包根本不会被IDS处理



12

Opportunities and Challenges

- Middlebox resource management：中间件涉及复杂的处理过程，用于捕获应用层语义、使用深度数据包检测。研究表明，中间件过载是导致故障的常见原因，因此，一个重要的考虑因素是平衡中间件之间的负载。例如右图所示，我们可能希望在两个防火墙之间平均分配处理负载



	Physical Sequence
FW1-IDS1-Proxy1	S1 S2 FW1 S2 S4 S5 IDS1 S5 S4 S2 Proxy1 S2 S4 S5 S6
FW2-IDS1-Proxy1	S1 S3 FW2 S3 S5 IDS1 S5 S4 S2 Proxy1 S2 S4 S5 S6
FW1-IDS1	S1 S2 FW1 S2 S4 S5 IDS1 S5 S6
FW2-IDS1	S1 S3 FW2 S3 S5 IDS1 S5 S6

13

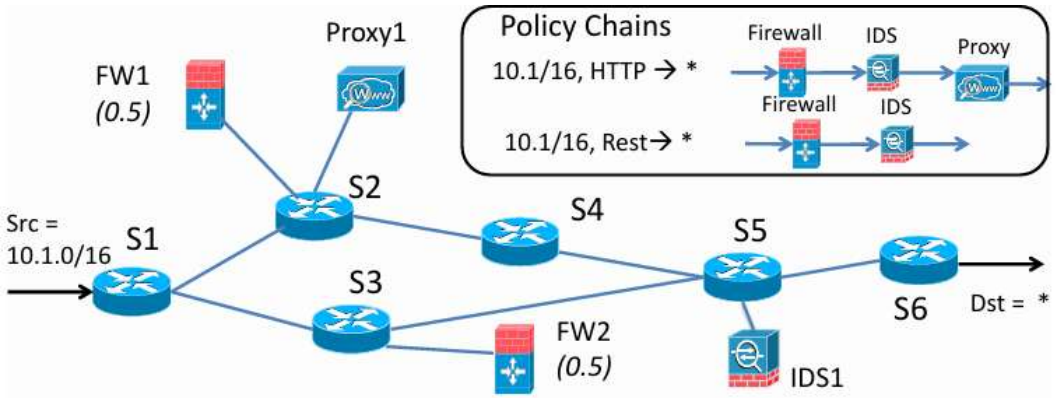
Opportunities and Challenges

- Opportunity： 如今，运营商需要静态设置流量拆分规则或采用自定义负载均衡解决方案。相比之下，**SDN 控制器可以利用数据平面转发规则灵活实施负载均衡策略**，并根据网络动态通过特定的交换机和中间件物理序列路由流量
- Challenge： SDN交换机所能支持的转发规则数量有限；这些规则存储在TCAM中，一台交换机可支持数千条规则（例如，5406zl交换机中有1500个TCAM条目）。在一个拥有O(100)个防火墙和O(100)个入侵检测系统的大型企业网络中，存在O(100 x 100)种可能的防火墙-入侵检测系统序列组合，在最坏情况下，网络中间的一台交换机可能需要配置O(100 x 100)条转发规则
  - 这意味着我们不能直接使用现有的中间件负载均衡算法，因为这些算法没有考虑到交换机的约束。

14

Opportunities and Challenges

- Dynamic traffic transformation：许多中间件会主动修改流量头部和内容。例如，网络地址转换（NAT）会重写单个数据包的IP地址，以映射内部IP和公共IP
- 从代理流出的流量与流入代理的流量相比，可能具有不同的数据包头部、会话和有效载荷。因此，控制器很难在S2上安装规则，以将适当的流量引导至不同的中间件（取决于原始用户组）。



	Physical Sequence
FW1-IDS1-Proxy1	S1 S2 FW1 S2 S4 S5 IDS1 S5 S4 S2 Proxy1 S2 S4 S5 S6
FW2-IDS1-Proxy1	S1 S3 FW2 S3 S5 IDS1 S5 S4 S2 Proxy1 S2 S4 S5 S6
FW1-IDS1	S1 S2 FW1 S2 S4 S5 IDS1 S5 S6
FW2-IDS1	S1 S3 FW2 S3 S5 IDS1 S5 S6

15

Opportunities and Challenges

- Opportunity：为了应对这种动态的数据包转换，如今运营商不得不采取临时措施：（1）将防火墙和入侵检测系统放在代理之后，以确保所有流量都经过所有中间件（2）基于中间件行为的粗略推理，手动计算和预测流量转换。**SDN可以借助网络全局视图，使得网络可以智能地适应中间件带来的流量转换。**

- Challenge：理想情况下，SDN控制器需要了解中间件的内部处理逻辑，以便在安装转发规则之前考虑流量修改。然而，这种逻辑可能是中间件供应商的专有信息。此外，这些转换可能发生在细粒度的时间尺度上，并且取决于流经中间件的特定数据包。这意味着需要自动适应此类由中间件引发的数据包转换。

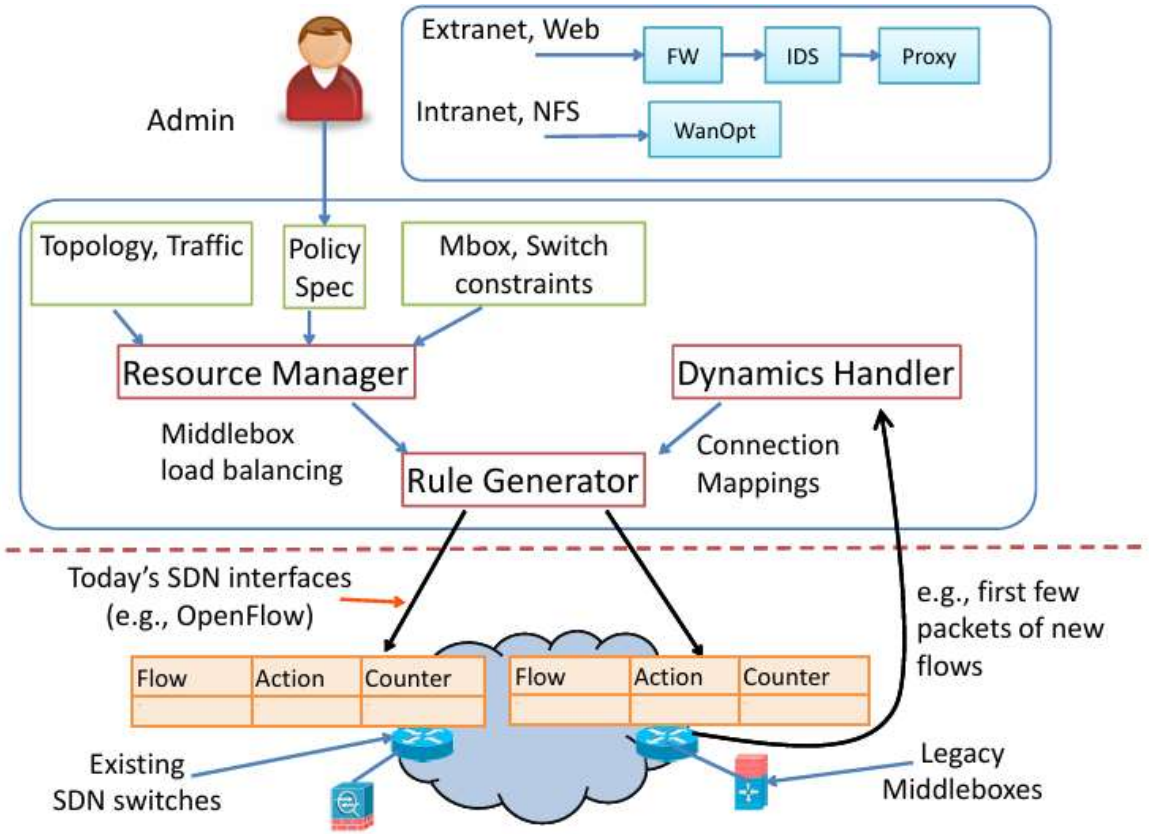
16

Outline

I. Introduction
II. Opportunities and Challenges
III. Simple System Overview
IV. Simple Data Plane Design
V. Resource Management
VI. Simple Dynamics Handler
VII. Implementation
VIII. Evaluations
IX. Review

Simple System Overview

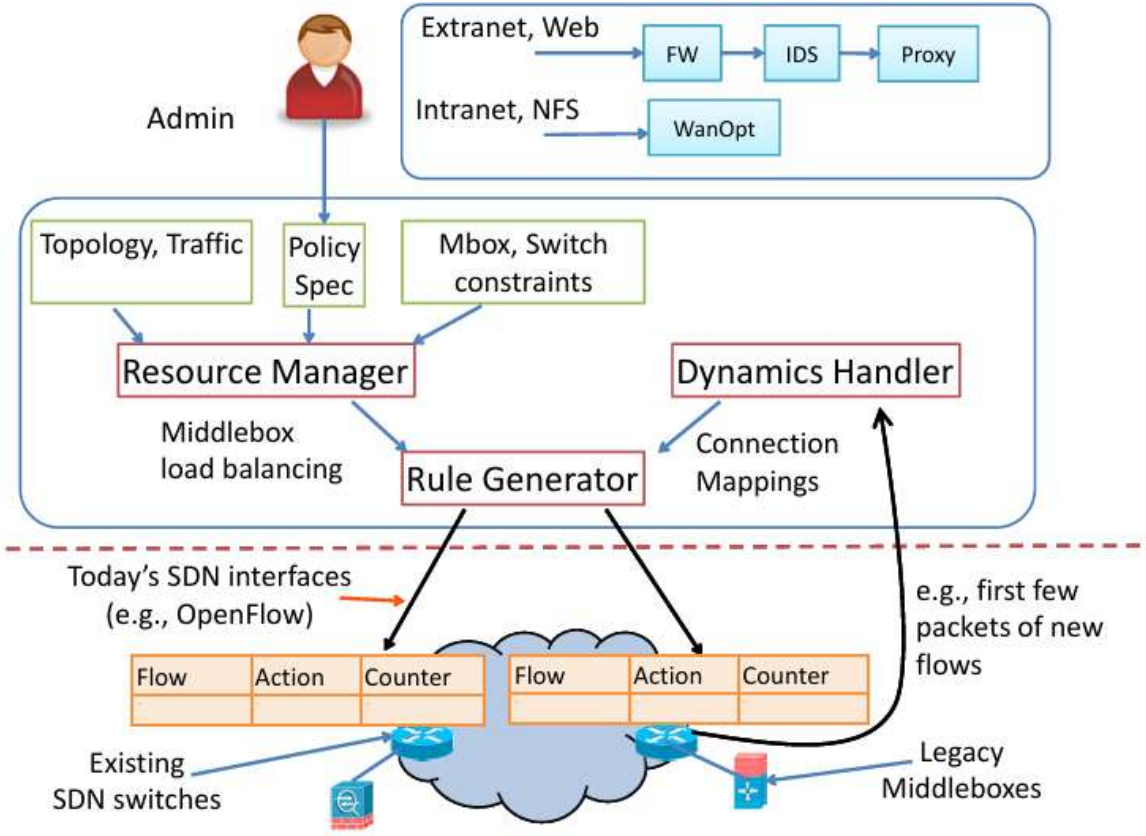
- SIMPLE是一个基于SDN的策略执行层，能够将高层中间件策略转换为高效且负载均衡的数据平面配置，从而引导流量通过所需的中间件序列。
- 右图概述了SIMPLE架构，展示了各个组件所需的输入、模块之间的交互以及与数据平面的接口
- SIMPLE只需配置支持SDN的交换机；中间件无需扩展以支持新的类SDN功能



18

Simple System Overview

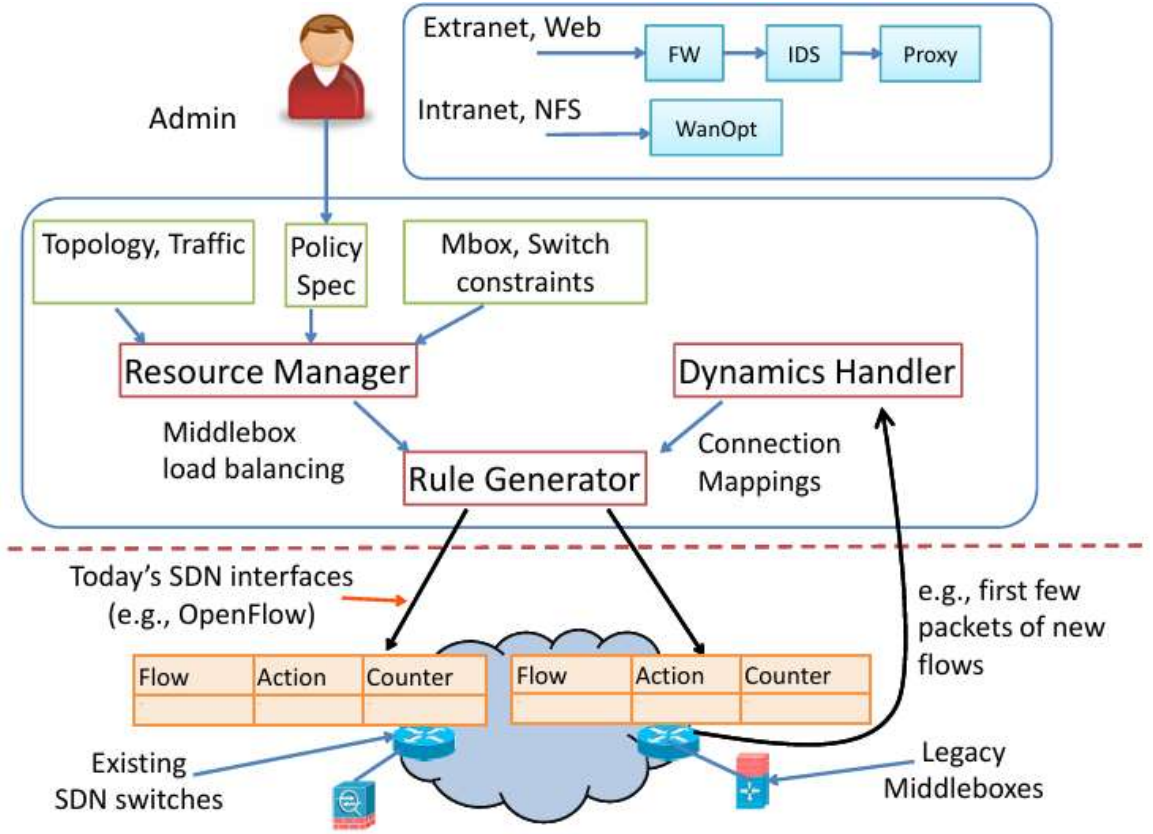
- SIMPLE的High-level的输入：
  - Processing policy：基于SDN的直接控制理念，我们希望网络管理员能够明确需要实现何种处理逻辑，而不必担心这种处理在何处进行或流量需要如何路由
  - 每个类别class c（例如<External,Web>）都标注了其入口和出口位置以及IP前缀。例如，这种外部网络流量可以通过如下流量过滤器来指定（src=内部前缀，dst=外部前缀，src-port =*，dst-port =80，proto = TCP）。PolicyChain表示该类别的所需中间件策略链（例如，防火墙-入侵检测系统）



19

Simple System Overview

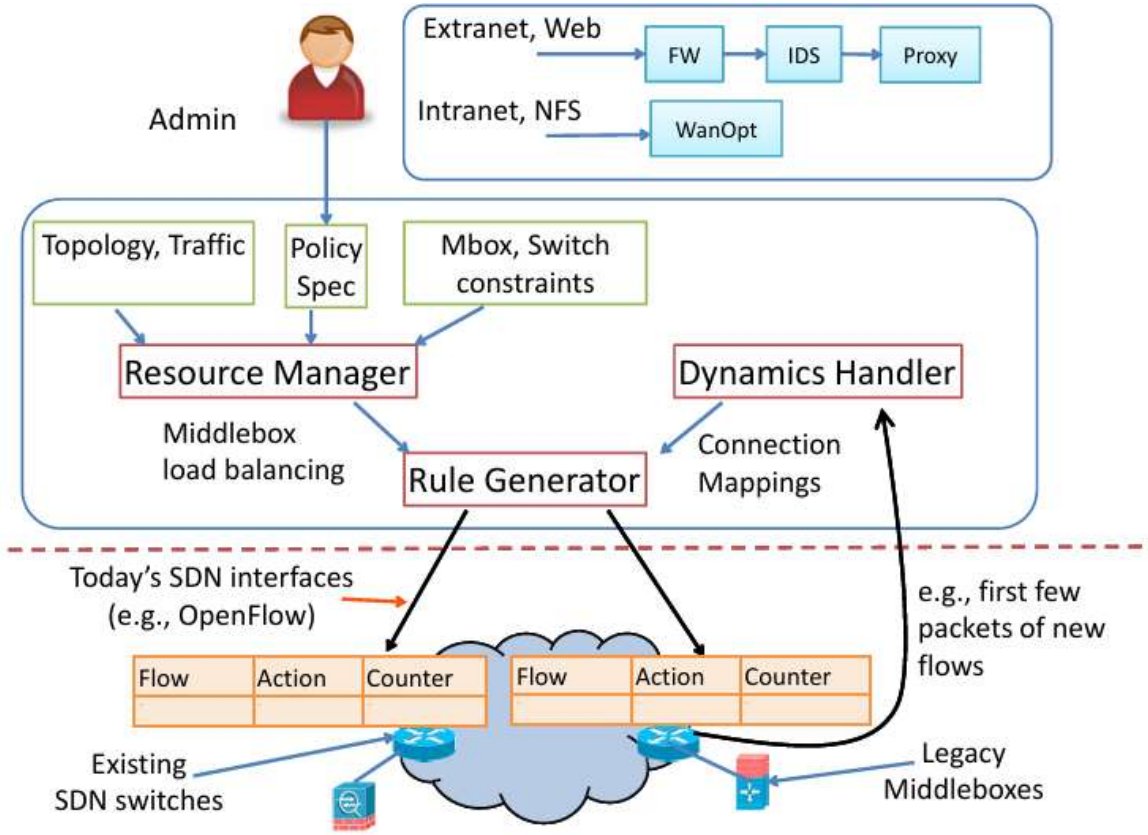
- SIMPLE的High-level的输入：
  - Topology and traffic：SIMPLE 最终必须将逻辑策略规范转换为物理拓扑结构。因此，它需要一张网络地图，标明中间件的位置、交换机之间的链路以及链路容量。我们还需要每个策略类别预期的流量规模。这类输入通常已在网络管理系统中收集
  - 为简化表述，我们假设每个中间件都通过一个支持SDN的交换机连接到网络，我们的技术也适用于中间件作为“串联节点”的部署场景。



20

Simple System Overview

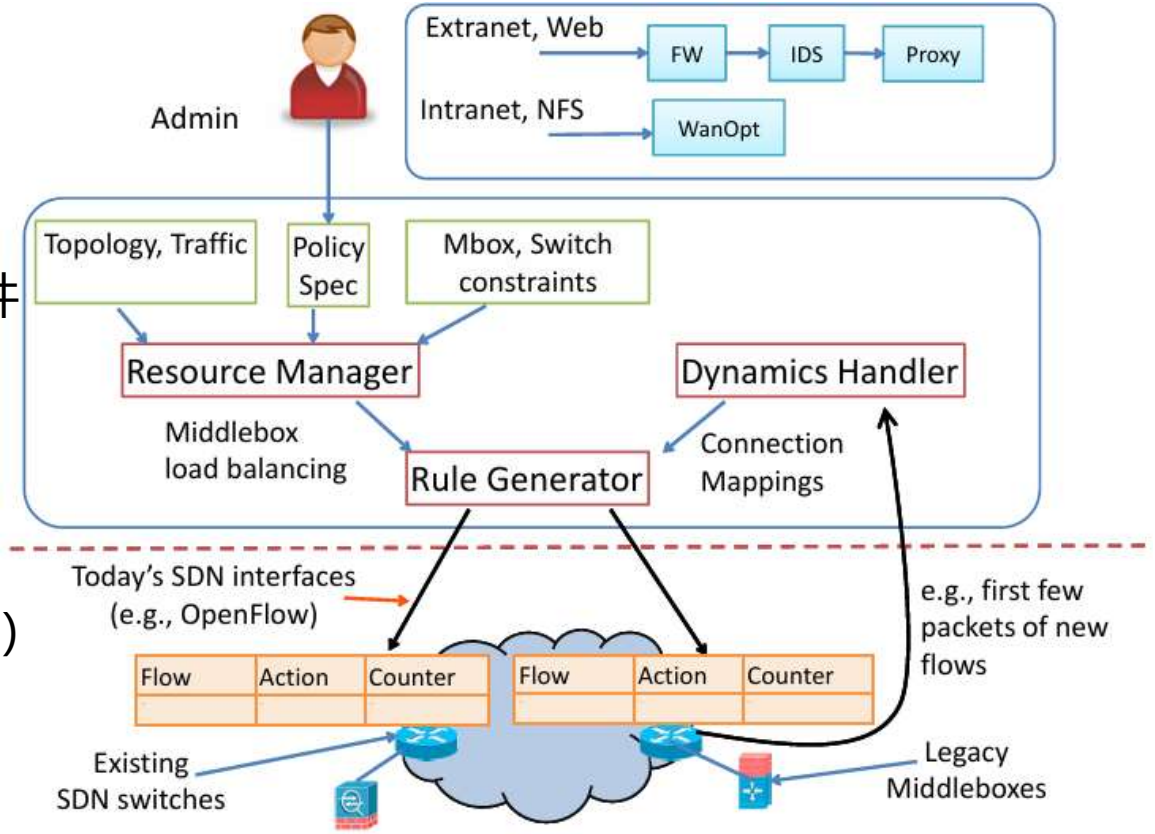
- SIMPLE的High-level的输入：
  - Resource constraints：存在两种类型的受限资源：（1）用于不同中间件数据包处理资源（例如，CPU、内存、加速器）；（2）可用于在SDN交换机中安装转发规则的TCAM数量
  - 另外还需要跨中间件和类别的每数据包处理成本；为了通用性，我们假设这些成本因中间件实例（例如，它们可能具有专用加速器）和策略类别（例如，HTTP与NFS）而异



21

Simple System Overview

- SIMPLE的三个处理模块：
  - ResMgr 将网络的流量矩阵、拓扑结构和策略要求作为输入，并输出一组实现策略要求的中间件处理分配
  - DynHandler 会自动推断能够修改数据包/会话头的中间件的传入和传出连接之间的映射，并将这些映射提供给下文所述的RuleGen模块。
  - RuleGen 接收ResMgr的输出（即不同中间件的处理职责）以及来自DynHandler的连接映射，并生成数据平面配置，以通过适当的中间件序列将流量路由至最终目的地



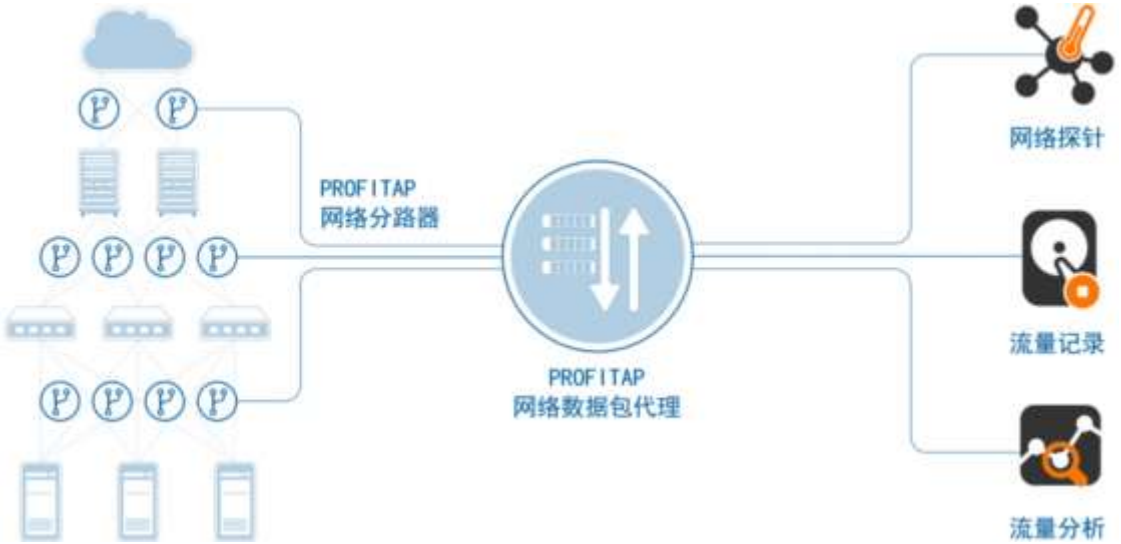
22

Outline

I. Introduction
II. Opportunities and Challenges
III. Simple System Overview
IV. Simple Data Plane Design
V. Resource Management
VI. Simple Dynamics Handler
VII. Implementation
VIII. Evaluations
IX. Review

Simple Data Plane Design

- SIMPLE数据平面有两个高层级要求
  - 交换机不能依赖流的5元组进行转发
  - 需要确保规则能够适配有限的TCAM
- 这对于中间件分布在整个网络中的大型网络而言尤为关键。为解决这些问题，本文提出了一种结合标签和隧道的数据平面解决方案

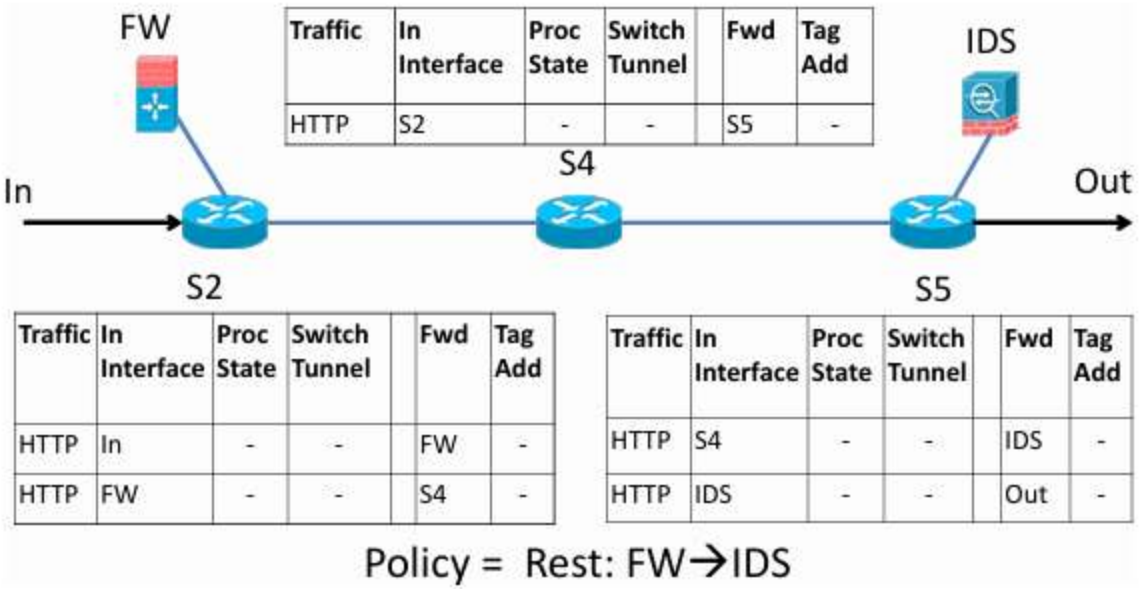


Simple Data Plane Design

- Unambiguous forwarding（无歧义转发）
- 网络中的交换机在任何时候收到一个数据包时，都能根据其所掌握的信息，做出唯一、明确且正确的转发决策
  - 因此我们需要交换机识别数据包当前处于中间件处理链中的哪个段【一个段是指从一个中间件（或入口网关）开始，到逻辑链中的下一个中间件结束的一系列交换机】
  - **我们使用结合拓扑上下文和数据包标记来编码这种处理状态，让它能够区分这两个看似相同的数据包**

Simple Data Plane Design

- **基于输入端口**：适用场景为当服务链的物理路径没有环路时，即每个定向链路在序列中最多出现一次。在这种情况下，交换机可以使用入站接口来识别逻辑段。考虑下图中的序列FW1-IDS1，其中数据包需要经过In–S2-FW1-S2-S4-S5-IDS1-S5–Out。在这种情况下，S2将从“In”端口到达的数据包转发至FW1，并将从FW1端口到达的数据包转发至S4



Traffic	In Interface	Proc State	Switch Tunnel		Fwd	Tag Add
HTTP	S2	-	-		S5	-

Traffic	In Interface	Proc State	Switch Tunnel		Fwd	Tag Add
HTTP	In	-	-		FW	-
HTTP	FW	-	-		S4	-

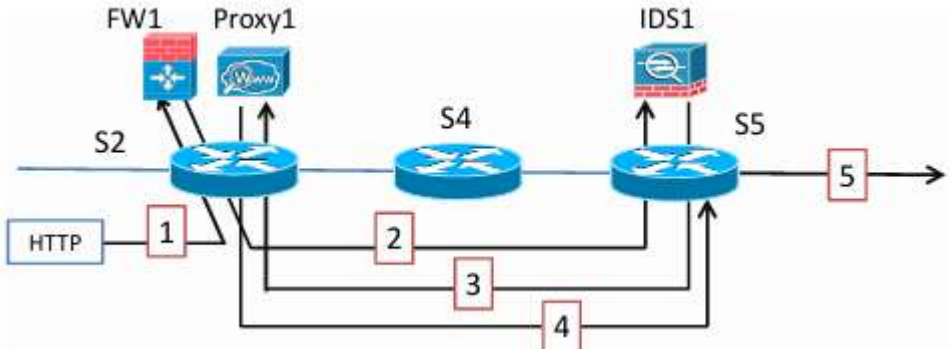
Traffic	In Interface	Proc State	Switch Tunnel		Fwd	Tag Add
HTTP	S4	-	-		IDS	-
HTTP	IDS	-	-		Out	-

(a) Hop-by-hop, No loop

Simple Data Plane Design

- **基于ProcState 标签：**如果物理序列中存在环路，那么输入接口和数据包头字段的组合无法识别中间件段。为解决此问题，我们引入了一种编码数据包处理状态的 ProcState 标签；根据SDN交换机支持的字段，ProcState 标签会利用 VLAN 标签、MPLS 标签或 IP 头中的未使用字段嵌入到数据包头内。用来标记它已经被哪个中间件处理过了，控制器会根据数据包头字段和输入端口，在每个段的第一个交换机上安装标签添加规则。下游交换机在其转发操作中会使用这些标签。

- 右图展示了S2处的标签规则：{HTTP，来自FW1} -> ProcState=FW；{HTTP，来自Proxy1} -> ProcState=Proxy。
- S5处的转发规则为：{HTTP，ProcState=FW} ->转发至IDS1；以及{HTTP，ProcState=代理} ->转发至目的地。S5可以利用ProcState标签来区分到达第二段的数据包的第一个实例（发送至入侵检测系统）和第四段的数据包（发送至目的地）



Simple Data Plane Design

- Compact forwarding tables（紧凑的转发表）
- 在实现复杂的中间件策略时，要让网络中每个交换机所需要存储的转发规则数量尽可能少
- 因为物理交换机中用于存储这些规则的内存（称为TCAM）是非常有限且昂贵的，最朴素的“逐跳转发”方法会使得转发表会变得臃肿：
  - 网络规模效应：如果网络很大，路径很长，一条策略链就会产生大量规则。
  - 策略数量效应：如果网络中有成百上千条不同的策略链（比如HTTP流量走一条路，FTP流量走另一条路），规则数量会成倍增加。

Simple Data Plane Design

- 为了减少转发条目的数量，我们利用了这样一个观察结果：物理序列各段中间的交换机不需要细粒度的转发规则。它们的唯一作用是将数据包路由到序列中下一个中间件所连接的交换机
- 通过**交换机间隧道**来实现，每个交换机维护两种转发表：
  - FwdTable (细粒度转发表)：存储那些需要精确匹配和处理的、针对特定流的“特殊规则”。
  - TunnelTable (隧道转发表)：这是一个更通用的“网络地图”。它只存储如何通过一个“隧道”到达网络中任何其他交换机的信息。这个表是预先计算好的，相对稳定。

Simple Data Plane Design

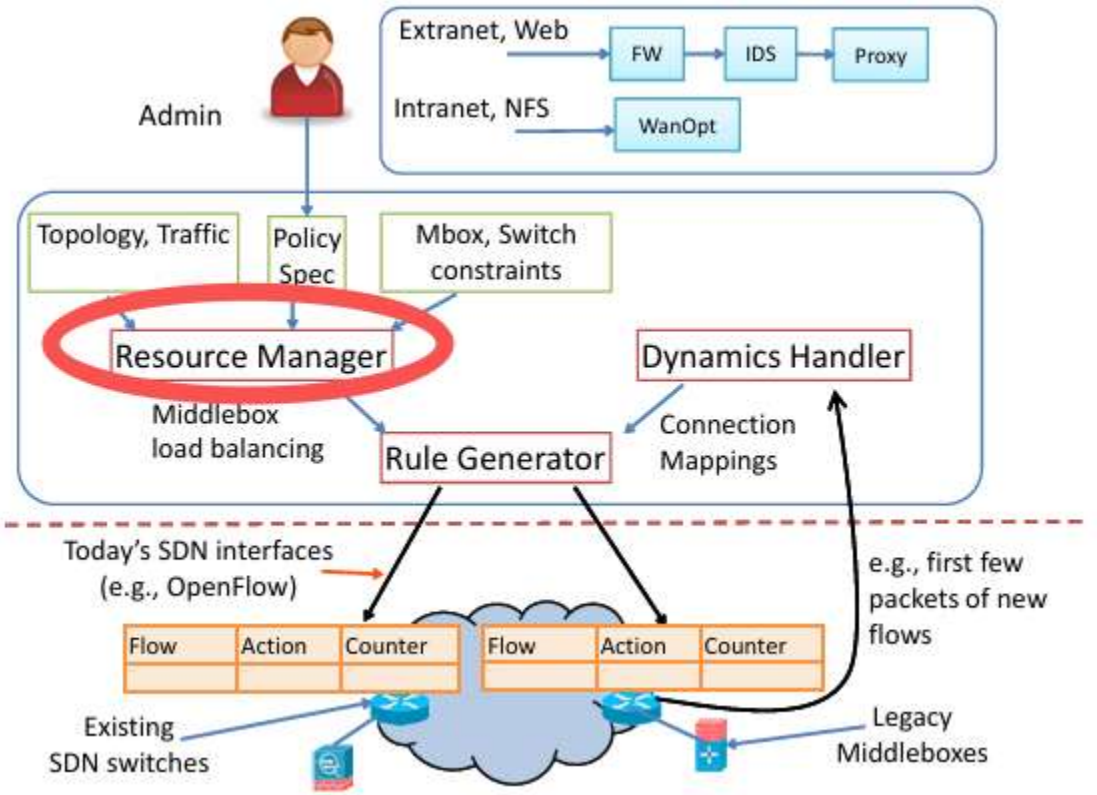
- 通过**交换机间隧道**
- 对于S1 → S2 → S3 → S4 → S5这样一条数据包处理路径：
  - 入口交换机 (Ingress Switch)：它负责执行细粒度的决策。它查看数据包，匹配FwdTable，然后它将数据包“封装”进一个通往 S5 的隧道里，然后发送出去。
  - 中转交换机 (Transit Switches)：路径中间的 S2, S3, S4。它们完全不关心数据包的原始身份（是HTTP还是FTP）。它们只看到最外层的“隧道标签”，于是它们查询自己的TunnelTable，以最高效的方式把数据包接力传向S5。它们不需要为这个特定的流安装任何新规则。
  - 出口交换机 (Egress Switch)：路径的终点（S5）。当它从隧道中收到数据包后，它会“解封装”，恢复数据包的原貌。然后，它再次执行细粒度的决策

Outline

I. Introduction
II. Opportunities and Challenges
III. Simple System Overview
IV. Simple Data Plane Design
V. Resource Management
VI. Simple Dynamics Handler
VII. Implementation
VIII. Evaluations
IX. Review

Resource Management

- Resource Management (资源管理)
- ResMgr中的核心挑战在于需要同时考虑中间件约束和SDN交换机的流表容量。
  - 中间件资源限制：每个中间件（如防火墙、IDS）都有自己的处理能力上限（比如每秒能处理多少Gbps的流量）
  - 交换机资源限制：每个SDN交换机存储转发规则的TCAM内存是有限的（比如只能存1500条规则）

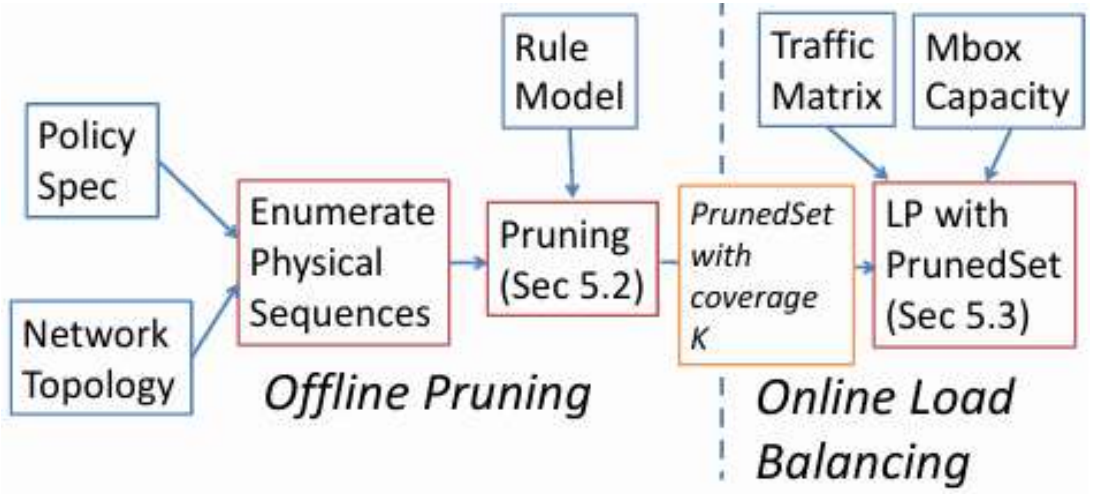


- 这两种约束揽在一起，使得问题变得异常复杂。这个问题在理论上是NP-hard的

Resource Management

➢ Offline-Online Decomposition (离线-在线分解)

- ➢ 作者提出了一个的“分而治之”的策略，把这个复杂的优化问题分解成两个更简单的步骤：
  - ➢ 慢速的离线阶段 (剪枝 Pruning)
  - ➢ 快速的在线阶段 (负载均衡 Load Balancing)



- ➢ 假设：网络的物理拓扑、中间件位置和交换机容量这些“硬件”信息是长期不变的，而网络流量模式是短期、频繁变化的
- ➢ 优点：将耗时、复杂的计算放在不经常运行的离线阶段，而将需要快速响应的在线阶段设计得非常高效。这确保了系统能够对流量变化做出近乎实时的调整。

Resource Management

➢ Offline-Online Decomposition (离线-在线分解)

- ➢ 离线阶段：剪枝 (Offline ILP-based Pruning)
- ➢ 决策变量：这是一个二进制变量 (0或1)，代表“对于策略c，我们是否选择物理路径q”
- ➢ 目标函数：优化的目标是让被选中的路径集合尽可能“分散”，避免所有路径都扎堆经过同一个中间件实例，从而预先防止热点产生
- ➢ 通过求解这个ILP，系统就能得到那个既满足交换机容量限制又保留了足够负载均衡灵活性的“可行路径集”

$$Minimize \;\; MaxMboxOccurs, subject\ to \tag{1}$$

$$\forall c : \sum_{q} d_{c,q} \geq Cov \tag{2}$$

$$\forall k : \sum_{\substack{c,q\ s.t. \\ S_k \in PhysSeq_{c,q}}} Rules_{k,c,q} \times d_{c,q} \leq TCAM_k \tag{3}$$

$$\forall j : \; MboxUsed_j = \sum_{c,q\ s.t. M_j \in PhysSeq_{c,q}} d_{c,q} \tag{4}$$

$$\forall j : \; MaxMboxOccurs \geq MboxUsed_j \tag{5}$$

$$\forall c, q : d_{c,q} \in \{0, 1\} \tag{6}$$

We model this problem as an ILP shown in Figure 6. We use binary indicator variables $d_{c,q}$ (Eq (6)) to denote if a particular physical sequence has been chosen. To ensure we have enough freedom to distribute the load for each chain, we define a target *coverage level* Cov such that each $PolicyChain_c$ will have at least Cov distinct $PhysSeq_{c,q}$ assigned to it in Eq (2). We constrain the total switch capacity used in Eq (3) to be less than the available TCAM space. Here, the number of rules depends on whether a given sequence is “active” or not. (Note that this conservatively assumes

Resource Management

➢ Offline-Online Decomposition (离线-在线分解)

- ➢ 在线阶段：负载均衡 (Online Load Balancing with LP)
- ➢ 决策变量：这是一个连续变量 (0到1之间)，代表“对于策略c，我们应该分配多大比例的流量给物理路径q”
- ➢ 目标函数：最小化网络中所有中间件中的最大负载值，这是典型的负载均衡目标
- ➢ 因为这个模型是线性规划，并且变量是连续的，所以求解速度极快，完全满足在线调整的需求

$$Minimize\ MaxMboxLoad \tag{7}$$

$$\forall c: \sum_{q:PhysSeq_{c,q} \in Pruned} f_{c,q} = 1 \tag{8}$$

$$\forall j: Load_j = \frac{\sum_{\substack{c,q\ s.t. M_j \in PhysSeq_{c,q} \\ PhysSeq_{c,q} \in Pruned}} f_{c,q} \times T_c \times Footprint_{c,j}}{ProcCap_j} \tag{9}$$

$$\forall j: MaxMboxLoad \geq Load_j \tag{10}$$

$$\forall c, q: f_{c,q} \in [0,1] \tag{11}$$

Figure 7: *Linear Program (LP) formulation for balancing load across middleboxes given a pruned set.*

Having selected a set of feasible sequences in the pruning stage, we formulate the middlebox load balancing problem as a *linear program* shown in Figure 7. The main control variable here is $f_{c,q}$, the *fraction of traffic* for $PolicyChain_c$ that is assigned to each (pruned) physical sequence $PhysSeq_{c,q}$.

Resource Management

➢ Extensions

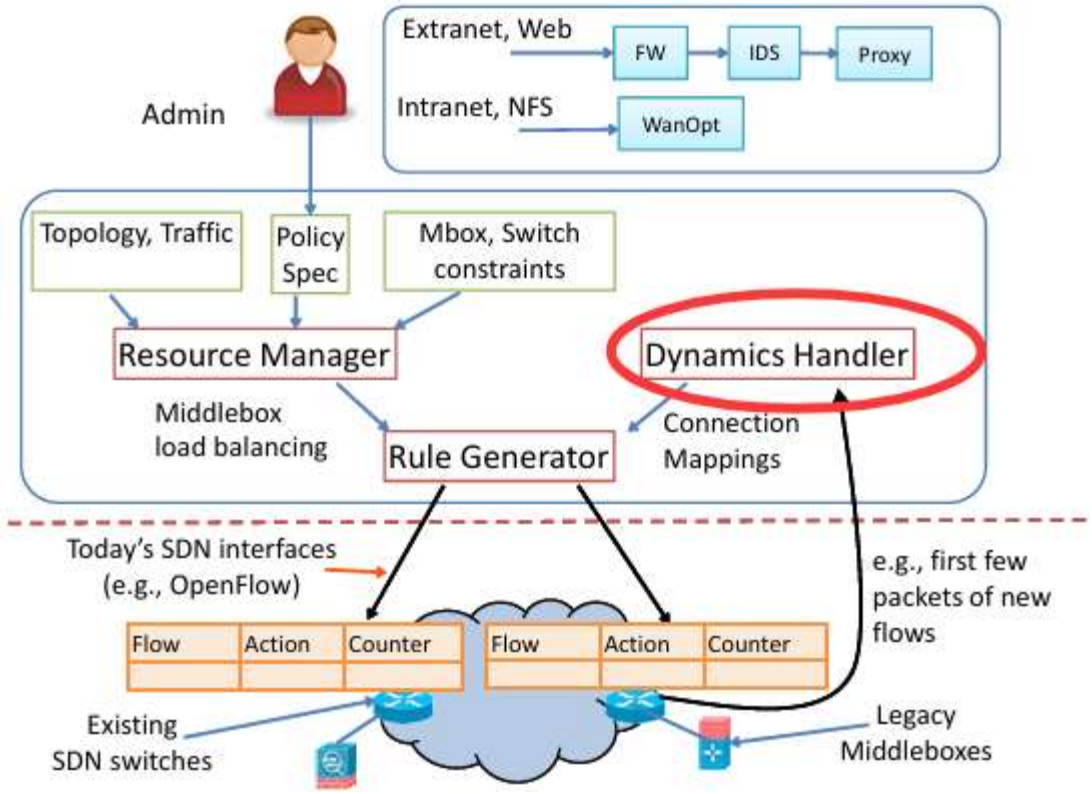
- 处理节点和链路故障：虽然我们期望拓扑结构基本稳定，但可能会出现暂时性的节点和链路故障。在这种情况下，修剪后的集合可能不再满足每个策略的要求。本文通过为不同的交换机、中间件和链路故障场景预先计算修剪序列来解决这个问题
- 处理策略变更：我们还预计中间件策略变更会在相对较粗的时间尺度上发生。然而，SIMPLE所具备的灵活性可能会引入动态策略调用场景；例如，如果我们观察到Web服务器负载较高，就通过数据包清洗器进行路由
- 其他流量工程目标：负载均衡线性规划还可以扩展以纳入其他流量工程目标。例如，在给定流量分配的情况下，我们可以对每条链路的负载进行建模，并对其进行约束，以确保没有链路的拥塞度超过30%

Outline

I. Introduction
II. Opportunities and Challenges
III. Simple System Overview
IV. Simple Data Plane Design
V. Resource Management
VI. Simple Dynamics Handler
VII. Implementation
VIII. Evaluations
IX. Review

Simple Dynamics Handler

- Simple Dynamics Handler(动态处理器)
  - 安装转发规则时中间件可能会动态修改传入流量——当中间件修改流的数据包头部时，下游交换机上的转发规则必须考虑新的头部字段。控制器必须知晓此类转换，并安装正确的转发规则，以将流量导向下一个中间件或出口交换机
  - 让SDN控制器能够自动地、准确地应对那些会修改数据包（特别是修改头部信息）的中间件，确保即使流量被“变身”，下游的转发规则依然正确



Simple Dynamics Handler

- Simple Dynamics Handler(动态处理器)
- Design constraints 设计约束，对问题进行了分类和定义
- 中间件行为的多样性：
  - 有的只读不改（如IDS）、有的会修改头部（如NAT）、有的甚至会修改负载、合并/拆分会话（如Proxy、广域网优化器）
- 核心原则：黑盒处理
  - 要求所有厂商提供标准化的API来暴露内部状态是不现实的，因此，需要将中间件视为黑盒，并尝试从外部自动学习其输入-输出行为。我们只关心“什么进去了，对应着什么出来了”，而不关心“里面发生了什么”。

Middlebox	Input	Actions	Timescale	Info needed	Approach
FlowMon	Header	No change	–	None	–
IDS	Header, Pay-load	No change	–	None	–
IP Firewall	Header	Drop?	–	None	–
IPS	Header, Pay-load	Drop?	–	None	–
Redundancy eliminator	Payload	Rewrite payload	Per-packet	None	–
NAT	Flow	Rewrite header	Per-flow	Header mapping	Payload Match
Load balancer	Flow	Rewrite head-ers & reroute	Per-flow	Session mappings	Payload Match
Proxy	Session	Map sessions	Per-session	Session mappings	Similarity Detector
WAN-Opt	Session	Map sessions	Per-session	Session mappings	Similarity Detector

- 中间件在不同的时间尺度上运行，修改不同的数据包头部，并以不同的粒度运行（例如，数据包、流、会话）

Simple Dynamics Handler

➢ Flow Correlation (流关联)

- 我们不需要理解中间件的全部复杂逻辑，对于转发任务来说，我们只需要知道：进入中间件的流 F_{in} 和离开中间件的流 F_{out}之间，哪个和哪个是“一对儿”？ 这就是流关联
  - 最简单 (No Change)：如果中间件不改变任何东西（如防火墙），F_{in} 和 F_{out} 的头部完全一样，直接匹配即可
  - 中等难度 (Payload Match)：如果中间件只改变头部，但不改变数据包的内容（如简单的NAT），那么我们可以通过比较 F_{in} 和 F_{out} 的负载内容来找到对应关系。如果负载完全一样，它们就是一对
  - 最困难 (Similarity-based)：如果中间件既改变头部，又改变负载（如代理Proxy），精确匹配就不可能了，但它们之间仍然会有很高的“部分重叠”或“相似性”

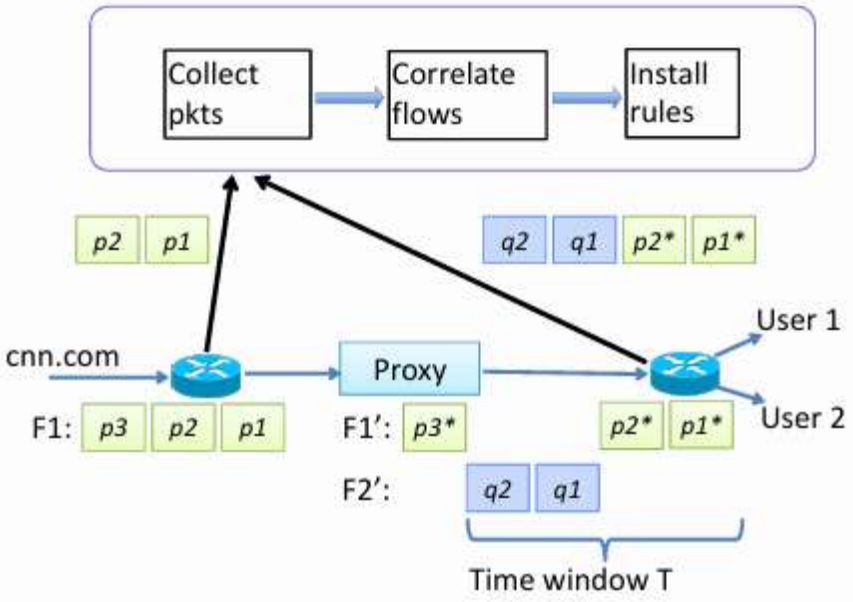
Middlebox	Input	Actions	Timescale	Info needed	Approach
FlowMon	Header	No change	–	None	–
IDS	Header, Pay-load	No change	–	None	–
IP Firewall	Header	Drop?	–	None	–
IPS	Header, Pay-load	Drop?	–	None	–
Redundancy eliminator	Payload	Rewrite payload	Per-packet	None	–
NAT	Flow	Rewrite header	Per-flow	Header mapping	Payload Match
Load balancer	Flow	Rewrite head-ers & reroute	Per-flow	Session mappings	Payload Match
Proxy	Session	Map sessions	Per-session	Session mappings	Similarity Detector
WAN-Opt	Session	Map sessions	Per-session	Session mappings	Similarity Detector

- 中间件在不同的时间尺度上运行，修改不同的数据包头部，并以不同的粒度运行（例如，数据包、流、会话）

Simple Dynamics Handler

- Flow Correlation (流关联)
- 对于最困难情况下的流关联，采用 **Similarity-based 方法，**

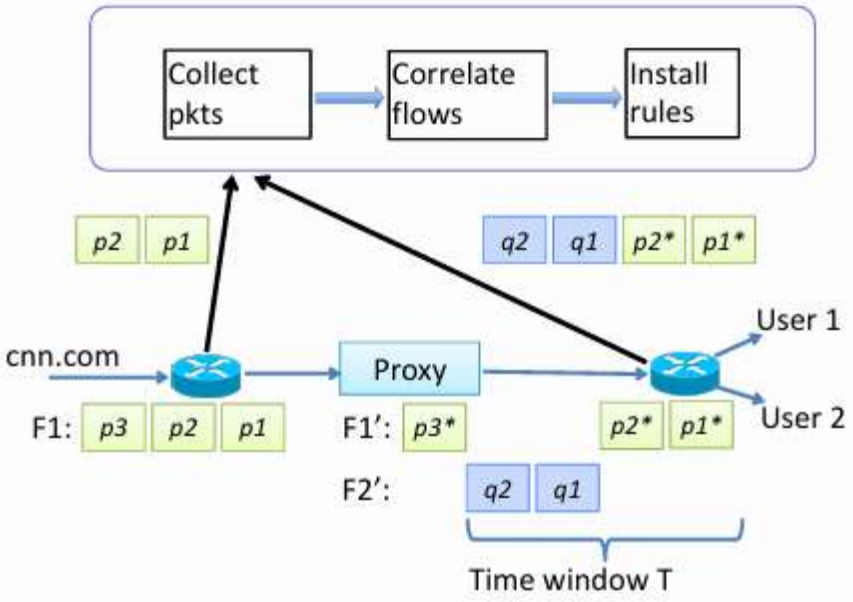
  以代理(Proxy)为例：
  - 1、收集数据包：当一个新的流 F1 到达中间件时，相连的交换机会将这个流的前 **P** 个数据包发送一份给SDN控制器，同时，控制器会监听在一小段时间窗口 **W** 内从这个中间件离开的所有新流（如 F1' 和 F2'），并同样收集它们的前 P 个数据包
  - 2、计算负载相似性：控制器从收集到的P个数据包中重建出每个流的负载数据流，使用一种对内容移位不敏感的技术——Rabin Fingerprints，通过计算两个流之间拥有相同指纹的数据块数量，就可以得出一个量化的相似性分数
  - 3、识别最相似的流：控制器会计算 F1 与时间窗口 W 内所有出向流 (F1 ‘, F2’ ...) 的相似性分数，得分最高的那个（或那些）出向流，就被认为是与 F1 相关联的流



- 控制器可能会发现 F1 和 F1' 的相似性得分远高于 F1 和 F2'，因此它就推断出了 F1 -> F1' 这个映射关系

Simple Dynamics Handler

- Flow Correlation (流关联)
- 另外，我们不需要对所有类型的流量都追求高精度的流关联
- 通过调整 DynHandler 算法中的两个关键参数来实现
  - P (Packets)：控制器为每个新流收集的数据包数量，P 值越大，可用于比较的负载内容就越多，关联的准确性就越高，但控制器处理的计算量也越大
  - W (Time Window)：时间窗口的大小。控制器会将在 W 时间内从中间件出来的所有新流都作为潜在的匹配对象。W 值越大，找到正确匹配的概率就越高（因为它给了出向流更多的时间出现），同理，控制器的计算开销也越大
- 对于某些流量，可以通过降低 P 和 W 的值，就可以显著减少DynHandler的带宽和计算开销



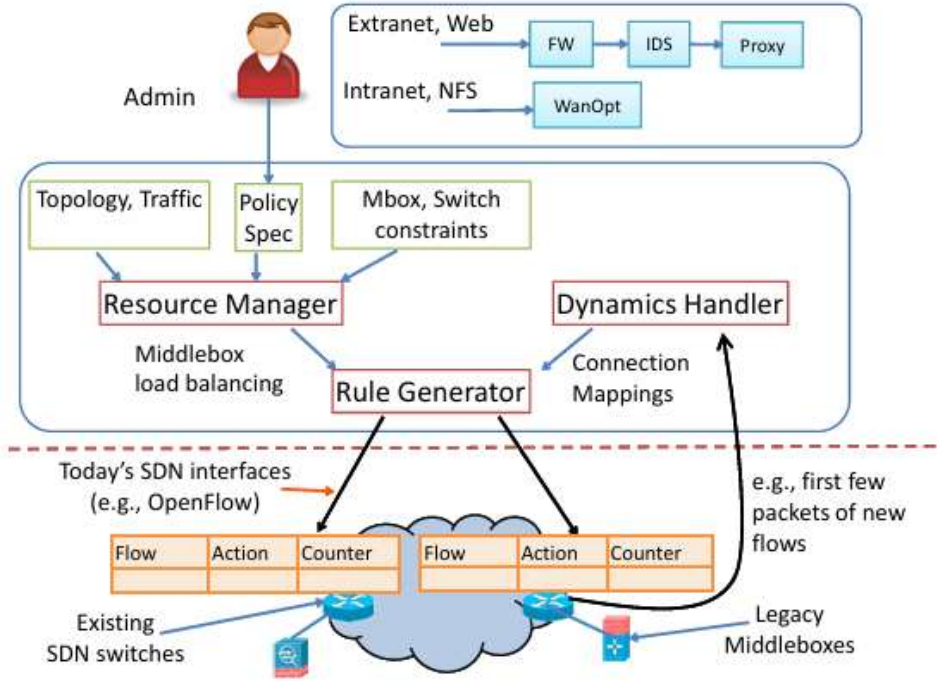
- 控制器可能会发现 F1 和 F1' 的相似性得分远高于 F1 和 F2'，因此它就推断出了 F1 -> F1' 这个映射关系

Outline

I. Introduction
II. Opportunities and Challenges
III. Simple System Overview
IV. Simple Data Plane Design
V. Resource Management
VI. Simple Dynamics Handler
VII. Implementation
VIII. Evaluations
IX. Review

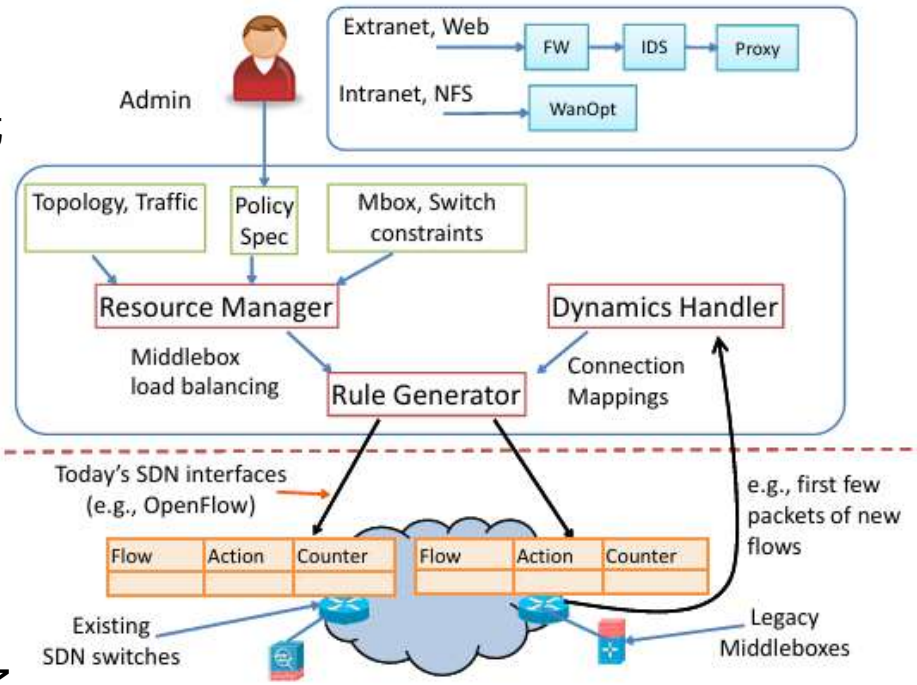
Implementation

- 基础平台：整个SIMPLE控制器原型是基于 POX 实现的【POX是一个用Python编写的、流行的开源SDN控制器平台】
- 组件划分：实现分为三个主要模块：RuleGen (规则生成器), ResMgr (资源管理器), 和 DynHandler (动态处理器)
- RuleGen是负责将上层策略（由ResMgr计算出的流量分配方案）转换成底层交换机转发规则的模块
  - 流量切分 (Traffic Splitting)
  - 转发方案选择 (Forwarding Scheme Selection)
  - 状态中间件处理 (Stateful Middlebox Handling)
  - 规则校验 (Rule Checking)



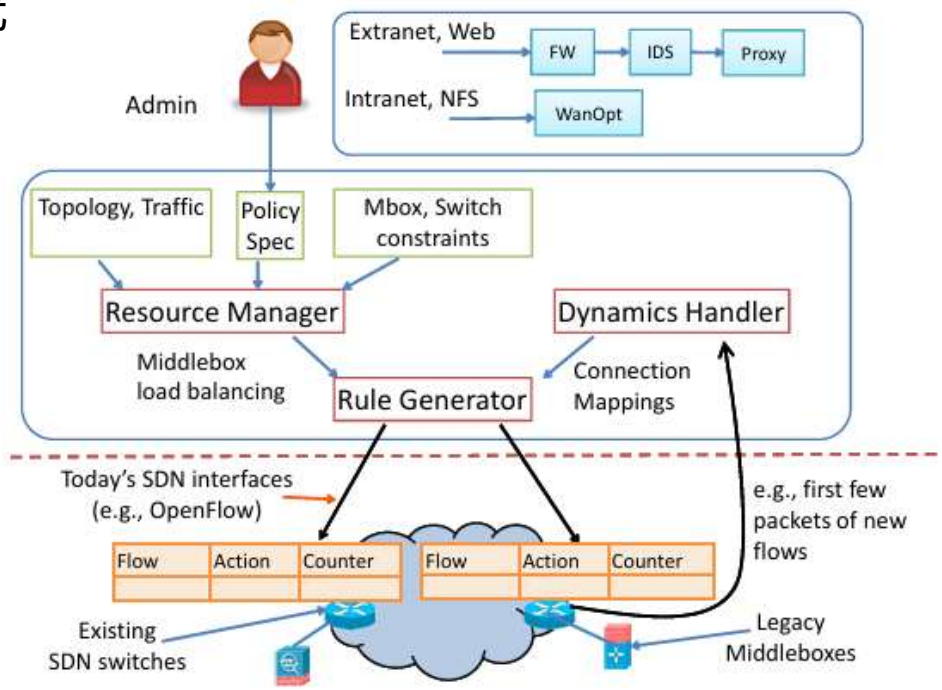
Implementation

- ResMgr是SIMPLE的“大脑”，负责进行复杂的资源优化计算
  - 求解器：对于 LP-based load balancing (在线负载均衡) 和 ILP-based pruning (离线剪枝) 这两个数学规划问题，作者使用了商业优化求解器 CPLEX
  - 故障处理：原型系统支持对所有单个链路、交换机或中间件的故障场景进行响应。这意味着它可以快速重新计算并部署新的流量路径来绕过故障点。
  - 性能优化：为了加快求解速度（特别是耗时的ILP剪枝），作者实现了一个优化：在重新计算时，将上一次的计算结果作为“热启动”的初始解提供给求解器。这比每次都从零开始计算要快得多。



Implementation

- DynHandler是负责处理中间件动态流量转换的模块
  - 数据包收集：它利用了SDN的控制器可以在交换机上安装一条“低优先级”的通配规则。当一个新流（没有匹配到任何高优先级规则）到达时，交换机会将这个流的前几个数据包发送给控制器。
  - 相似性计算实现：作者自己实现了一套Rabin指纹算法
  - 与RuleGen的交互：DynHandler运行关联算法，并将推断出的映射关系（如流F1进去后，出来变成了流F1'，并且F1'的新头部信息是什么）提供给 RuleGen。
  - 规则的动态性 ： 由DynHandler触发生成的规则是按需 (on-demand) 和 瞬态 (transient) 的。这意味着这些规则只在需要时才被创建，并且只为一个流的生命周期而存在（流结束后会过期删除）。



Outline

I. Introduction
II. Opportunities and Challenges
III. Simple System Overview
IV. Simple Data Plane Design
V. Resource Management
VI. Simple Dynamics Handler
VII. Implementation
VIII. Evaluations
IX. Review

Evaluations

- 为了从不同维度全面评估SIMPLE，作者采用了一种多层次的实验方法：
  - Emulab (真实环境小规模测试)：
    - 目的：验证系统在真实物理硬件和网络环境下的基本功能和性能
    - 特点：结果最真实，但规模受限于可用的物理机器数量
  - Mininet (仿真环境中大规模测试)：
    - 目的：在单台机器上仿真出更大规模的网络拓扑（如几十个交换机和主机），以评估系统的可扩展性和通用性
    - 特点：规模比Emulab大，能方便地创建复杂拓扑，结果与真实环境高度一致
  - Trace-driven Simulations (超大规模测试)：
    - 目的：专门用于测试算法在超大规模网络（如上百个节点的真实AS拓扑）下的计算性能和收敛时间
    - 特点：脱离了真实的数据包转发，只关注算法本身的性能，可以测试非常大的问题规模

Evaluations

- 系统基准测试 (System Benchmarks)
- Emulab vs. Mininet 一致性验证
  - 目的：测量SIMPLE系统的基本性能指标
  - 结论：两个平台上的端到端性能指标基本一致，这证明了使用Mininet进行更大规模的实验是可靠和有效的
- 规则安装时间和开销
  - 安装时间：即使对于23个节点的Enterprise拓扑，安装所有规则的时间也仅为约300毫秒
  - 控制流量开销：控制器与交换机之间的通信开销很小，与需要安装的规则总数成正比。

Platform, Config	Time to Install Rules(s)	Overhead (B)	Max MB Load (KB/s)	Max Link Utilization (KB/s)
Emulab, SIMPLE	0.041	5112	25.2	25.2
Mininet, SIMPLE	0.039	5112	25.2	25.2

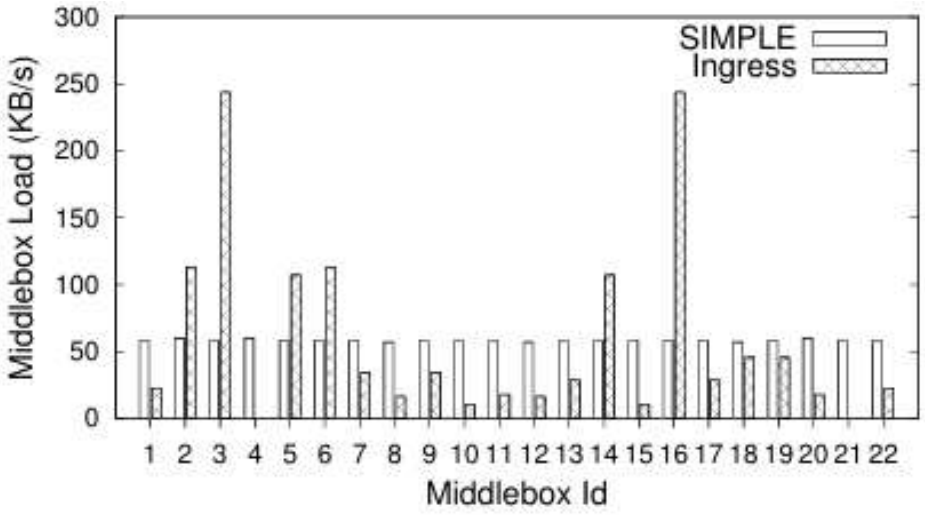
Table 2: *End-to-end metrics for the topology in Figure 1 on Emulab and Mininet. Having confirmed that the results are similar, we use Mininet for larger-scale experiments.*

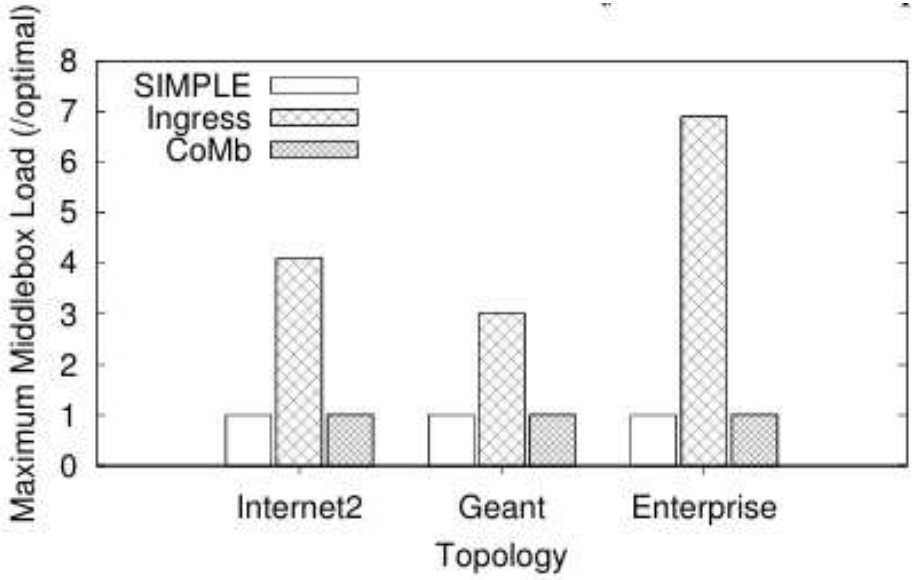
Topology	#Switches, #Hosts, #Mboxes	#Rules	Time (s)	Overhead (KB)
Figure1	6, 2, 4	36	0.04	5
Internet2	11, 11, 10	1699	0.09	180
Geant	22, 20, 20	6964	0.19	820
Enterprise	23, 23, 20	6689	0.31	710

Table 3: *Time and control traffic overhead to install forwarding rules in switches.*

Evaluations

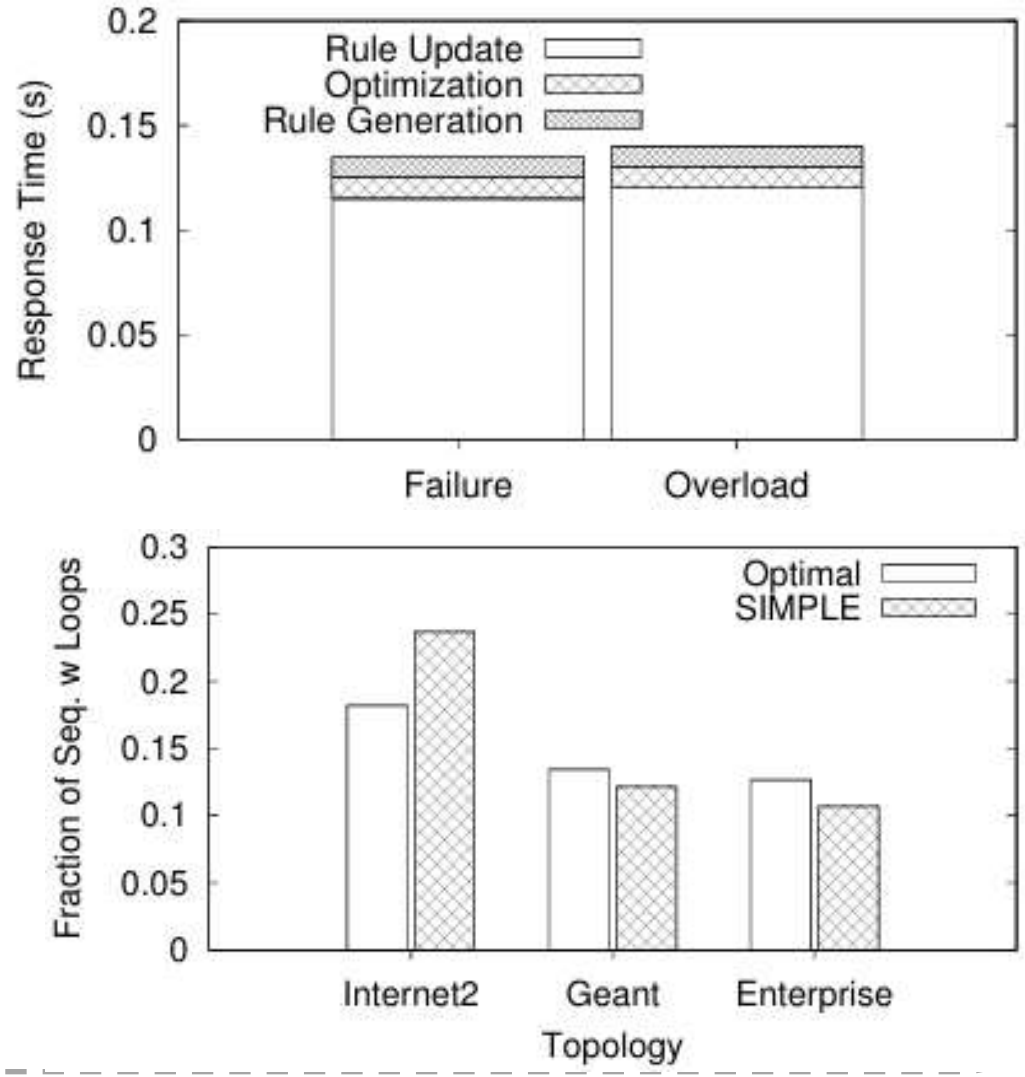
- 通过对比实验，展示SIMPLE相比现有方法的优越性
  - 结论一：**SIMPLE的负载均衡效果远超传统的Ingress-based方法**，可以将网络中的最大负载降低3到6倍。右图直观地显示了SIMPLE能将负载均匀地分散到各个中间件上，而Ingress方法导致了严重的负载倾斜
  - 结论二：**SIMPLE的性能与CoMb（建立中间件联合体）几乎没有差距。**这证明了SIMPLE在不修改中间件的前提下，就能达到与那些需要对中间件进行重大改造的方案相媲美的负载均衡效果





Evaluations

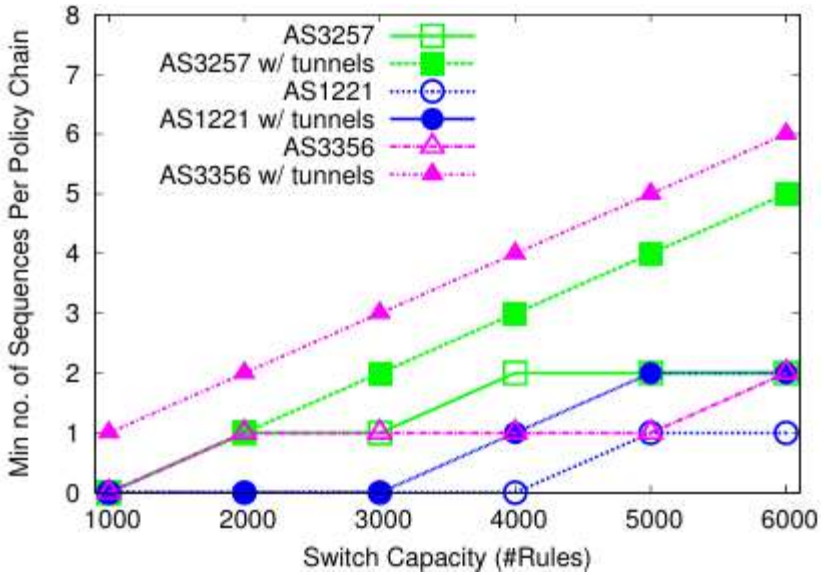
- 通过对比实验，展示SIMPLE相比现有方法的优越性
  - 结论三：SIMPLE重新计算和部署新规则的总响应时间**低于150毫秒**，其中绝大部分时间都花在规则安装上，SIMPLE自身的逻辑（优化和规则生成）开销可以忽略不计。这**证明了其在线调整机制的高效性**
  - 结论四：有相当一部分 (10%-25%) 的最优路径是包含环路的。这有力地证明了论文提出的ProcState标签机制不是“画蛇添足”，而是在追求最优性能时必不可少的



Evaluations

- 可扩展性与最优性：在更大规模的拓扑上测试SIMPLE优化算法的性能
  - SIMPLE的计算时间比理论上的Optimal方法快了4个数量级（几万倍）。即使对于一个包含252个节点的超大型网络(AS3356-aug)，SIMPLE也仅需约1秒就能生成配置
  - SwitchTunnels显著增加了可行路径的数量（增益高达3倍），在某些交换机TCAM非常紧张的场景下，只有使用SwitchTunnels才能找到可行的解决方案。这证明了该技术对于提升系统在资源受限环境下的能力至关重要

Topology	#Switches	Time(s)			
		Opt	Opt w/ tunnel	SIMPLE	SIMPLE w/ tunnel
Internet2	11	0.3	0.3	0.01	0.01
Geant	22	2.29	1.99	0.09	0.14
Enterprise	23	1.76	2.46	0.01	0.01
AS1221	44	23394	91.7	0.04	0.29
AS1239	52	722.7	218.1	0.06	0.2
AS3356	63	122246	3239	0.22	0.48
AS3356-aug	252	-	-	0.92	1.22



Evaluations

- DynHandler的准确性：评估动态处理器在处理最复杂的代理(Proxy)场景时的准确率
- 定义了策略漏报率 (Missed policy rate)和策略误报率 (False policy rate)
  - 随着关联时间窗口的增加，错误率迅速下降。
  - 在一个500毫秒的时间窗口下，对于最复杂的用户特定策略，漏报率和误报率都控制在了10%左右（即准确率约90-95%）。
  - 对于要求不那么高的状态策略，可以使用更小的时间窗口（如200毫秒）达到相似的准确率。
  - 整个过程的开销很小，带宽占用为几十KB，处理1000次关联仅需150毫秒。

Outline

I. Introduction
II. Opportunities and Challenges
III. Simple System Overview
IV. Simple Data Plane Design
V. Resource Management
VI. Simple Dynamics Handler
VII. Implementation
VIII. Evaluations
IX. Review

Review

- 当前系统存在局限吗?
  - 性能开销和用户感知延迟：DynHandler的工作流程是“先观察，后行动”，当一个新流到达时，它需要先将前几个数据包发送给控制器，控制器进行分析计算后，才能安装正确的转发规则。在这个“推断”的过程中，这个新流的前几个数据包是被延迟处理的
  - 对加密或编码负载的无力：DynHandler的核心是基于负载相似性进行推断。它的有效性完全依赖于能够“看懂”数据包的内容，如果流量的负载被加密（如HTTPS, VPN）或被专有协议编码 (encoded)，那么从外部看，这些负载就是一堆毫无意义的随机数据

Review

- 未来架构演进
- SIMPLE的设计是基于一个特定的模型：传统的、非可编程的“黑盒”中间件 + 可编程的SDN交换机
  - SDN交换机集成中间件功能：未来的交换机芯片可能会变得更强大，能够直接在交换机内部执行一些简单的中间件功能（如防火墙ACL）。
  - 中间件变得可编程 (Programmable Middleboxes)：可能会出现像P4语言那样，可以对中间件的数据平面行为进行编程的新范式。

Review

- Policy Management in SDN (SDN中的策略管理)
  - 传统的SDN研究主要集中在L2/L3（二/三层）的策略，如访问控制、限速和路由
  - SIMPLE将SDN的策略管理能力扩展到了L4-L7（四到七层），专门处理由中间件策略定义的服务链遍历 (traversal of middlebox chains) 问题。这是对传统SDN策略的补充和延伸
  - 它的核心贡献在于提出了一个统一的视角，能够同时考虑交换机资源 (switch resource) 和 中间件资源 (middlebox resource) 这两种约束，并进行联合优化，这是之前工作没有做到的。