第一章　sdiv 章节

1.1　引言

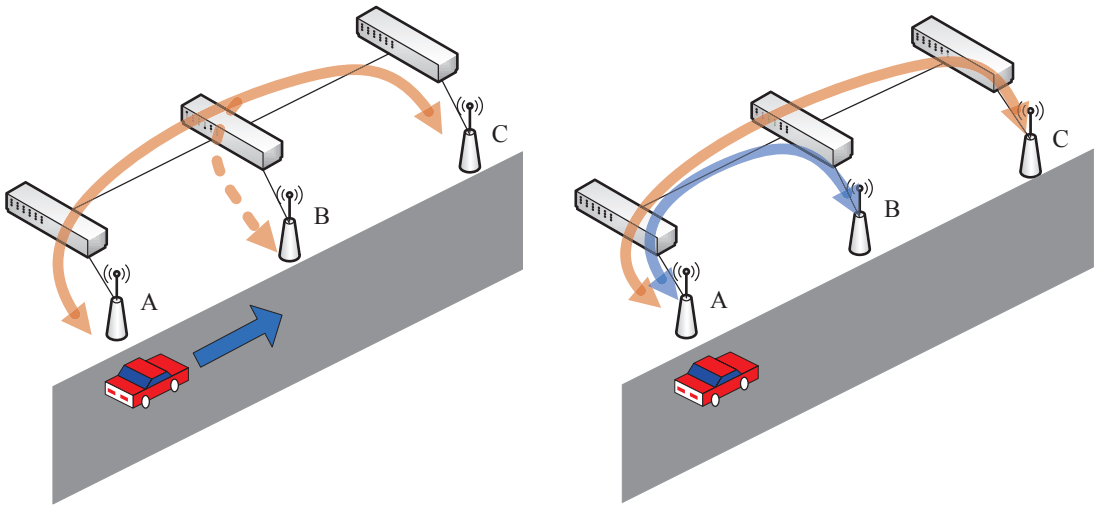
车联网（Internet of Vehicles，IoV）目前受到了学术界和工业界的巨大关注。通信技术和智能城市的巨大发展给 IoV 带来了更多可能的服务，提高了车辆驾驶的质量和安全性。为了 IoV 的进一步发展，目前已有多个研究和工业组织在实施各种相关项目 [?, ?]。例如，欧盟领导的 CVIS [?] 致力于发展车与车之间以及车与路旁设施之间的通信技术；Smartway [?] 的目标是集成多个智能传输系统并提供一个统一的平台。

虽然 IoV 有着很好的发展前景，但其中网络设备的封闭性阻碍了 IoV 上新服务的快速部署和更新。网络设备（如交换机、路由器）目前由不同的厂商开发，设备的更改需要专业人员的手工配置，使网络管理变得昂贵并且易出错。由于缺少一个开放和统一接口，无法实现网络的灵活和动态的配置，从而影响大规模 IoV 上新服务的部署和拓展。为了 IoV 的发展，一个新的网络架构是必要的。

关于新的网络架构以及下一代互联网，目前有如下相关的研究。命名数据网络 [?] 的目标为开发一个新的互联网架构。与传统互联网的从哪里获取服务相比，命名数据网络关注获取什么服务。为了实现移动过程中的无缝隙数据传输，MobilityFirst [?] 将节点的移动性作为一个普遍场景进行设计。NEBULA [?] 则设计了一个基于云计算和数据中心的新的网络架构。SDN 将控制层和数据层进行了分离，并提供了配置网络的统一的接口。一个统一的配置网络设备的接口，使大规模可灵活配置的网络变得可行，从而可以加速 IoV 上新服务的部署。

考虑其开放和统一的接口，我们采用 SDN 作为 IoV 中底层网络架构，并提出软件定义车联网（Software-Defined Internet of Vehicles，SDIV）的架构。除了其开放统一的接口，SDIV 在以下方面有较大优势：1. 通过将控制平面和数据平面的分离，SDIV 有着非常高的可拓展性；2. 通过逻辑上集中式的控制器，SDIV 简化了网络的管理；3. SDIV 中的控制器可以根据当前网络状况计算数据传输的最优路径并协调车与路旁电子设备之间的通信。

关于 SDIV 上的编程框架，由于车辆的移动性，我们提出将底层每一个交换机的数据通路分为两个部分：面向静态转发路径部分和面向动态转发路径部分。其中静态部分则由高级 SDN 程序编译生成（该部分不会随着车辆移动而变化）。对于动态部分，由于车辆的不断移动，提前生成流水线结构以及流表项并不是一个好的设计。因此 SDIV 将动态转发路径部分设计为单流表的响应式下发流规则。鉴于单流表的有限的流表容量以及 IoV 中的移动特性，如何构建紧凑的流表是 SDIV 的主要挑战，也是本章的主要技术内容。



(a) (b)

图 1.1: （a）：当车辆从 A 移到 B 时，它需要重新建立连接。虚线表示连接尚没建立；（b）当车辆同时连接多个摄像头时，对于每个连接建立路径下发规则并不高效由于所有的路径目的一样（即所有数据流目的地址为 A）。

本工作中，我们考虑 IoV 中的实时查询服务，来给出 SDIV 中优化的流规则设置细节。通过实时查询服务，驾驶员可以获取实时的道路或其他信息来选择正确的驾驶方向。道路信息来自于路旁的监控摄像头并传给有需求的车辆。在此场景中，具体问题如下：1. 由于车辆的移动，车辆和摄像头之间需要不断建立新的连接，从而增加了流规则的数量；2. 当车辆与多个摄像头进行连接时，由于它们具有相同的目的，对每一个连接设置流规则并不高效。图 1.1描述了上述的问题。如果控制器简单地对每一个请求都下发流规则，则交换机中流表容量会成为系统瓶颈，并最终影响实时查询服务的性能。

为了解决实时查询服务中流规则设置问题，通过借助 SDN 的集中式细粒度的流控制，我们提出了以下技术：1. 为了减少车辆发出的请求数量，我们在数据从摄像头到车辆的最后一跳时采用多播地址；2. 为了保证在车辆移动过程中数据传输不会中断，我们根据车辆的行驶状况提前在交换机中设置流规则；3. 为了减少流规则数量以及保证数据传输的正确性，我们在分支节点更改数据包头。

1.2 相关工作

目前 IoV 中的大多数服务，例如，道路安全、车辆管理、车辆导航等，依赖于车与车（V2V）之间的数据传输以及车与设备（V2I）之间的数据传输 [?]。已有一些研究工作验证了通过路旁 AP 以及 WiFi 提供数据的连通性 [?, ?]。

Hare 等人 [?] 设计一个集中式的策略框架来管理车辆网络的光谱资源来保证用户需要的数据传输性能。Abadi 等人（XXX）根据有限的交通信息，对交通流量进行预测以及交通拥

塞控制。Wu 等人 [?] 结合 GPS 信息提高了对车辆姿态估计的准确度。Ahn 等人 [?] 第一个提出了面向车辆网络的分布式路面状况探测以及数据分发的架构。Calafate 等人 [?] 研究通过计算最优数据包大小以及决定最优前向纠错，实现广播下的可靠数据传送机制。

Grassi 等人 [?] 通过将命名数据网络应用于车辆网络中，实现了所有计算设备上的互通。并且设备的互通独立于它们以何种方式连在一起，如有线网路，ad-hoc 网络，或延迟容忍网络。Yap 等人 [?] 通过将网络服务与底层设施的分离，提出 OpenFlow 无线网络以实现任何无线设施之间的互通。

Kim 等人 [?] 考虑在校园网络以及家庭网络中利用 SDN 优化网络的管理。Kanizo 等人 [?] 提出一个分布式的框架对较大的交换机流表进行分解来应对交换机流表容量受限问题。Voellmy 等人 [?] 设计 Maple 系统来发现可重用的转发策略并通过一个踪迹树的结构记录对数据包的操作来减少下发的流规则数量。

从以上讨论可以看出，虽然目前有很多 IoV 的研究工作以及相关的 SDN 的应用，但并没有考虑新服务的部署以及通过集中式的控制器优化 IoV 中数据传输。因此，我们提出 SDIV 的架构。

1.3 SDIV 架构以及工作流程

本节中，我们首先给出 SDIV 架构，并通过实时查询服务来说明其工作流程。

1.3.1 SDIV 架构

如图 1.3所示，我们提出的 SDIV 网络是一个具有三层的架构。从下至上分别为，物理层，控制层，以及应用层。

物理层

物理层包括车辆，接入点（Access Point，AP），路边电子设备，交换机，以及服务器（区别于 SDN 中的控制器）。车辆作为移动的节点通过附近 AP 与服务器进行连接。这里我们假设 AP 的数量足够多并可以覆盖所有道路。路边电子设备如摄像头收集完路况信息后需要将数据传送到服务器或车辆。交换机连接着 AP，摄像头以及服务器。当收到车辆发送的请求时，服务器会将相应的信息传送给车辆。一辆车可以同时连接多个服务器。

关于交换机数据通路设计，如图 1.2所示。我们提出将每一个交换机的数据通路分为两个部分：面向静态转发路径部分和面向动态转发路径部分。其中静态部分则由高级 SDN 程序编译生成。该部分具体的应用可以是：IoV 中设施之间的通信（如摄像头到服务器）或实现对数据包的非全网路由的操作（如 ACL）。对于面向动态转发路径部分，由于车辆的不断移动，提前生成流水线结构以及流表项并不是一个好的设计。无法提前生成流表项可以很好理解，因为节点的移动性，流表项应该不断变化。对于流水线方面，由于车辆的不断移动，若对动态转发路径部分使用流水线结构，则要求网络中所有节点都为相同的流水线结构。该做法既浪费网

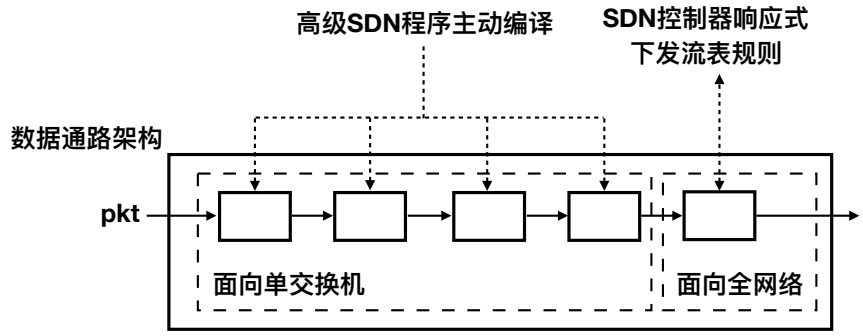


图 1.2: SDIV 的数据通路架构。

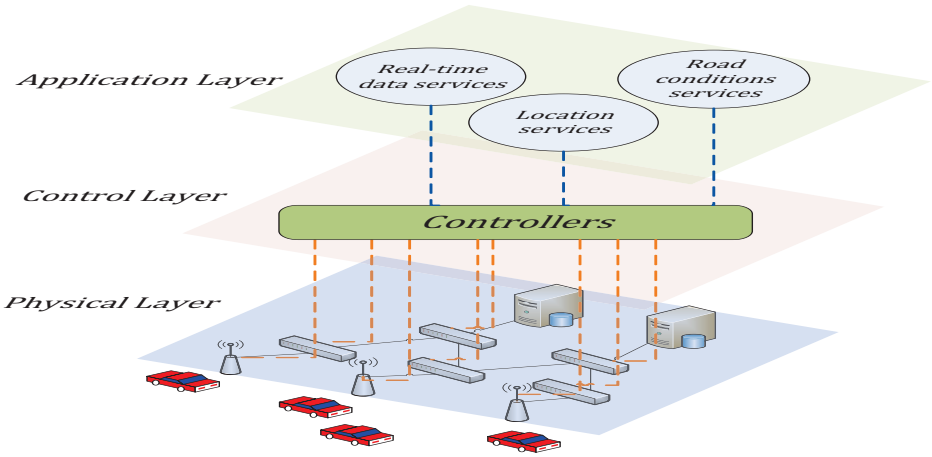


图 1.3: SDIV 的三层架构模型。

络资源，也会增加数据包时延，因为数据包经过流水线的时间与流表数量成正比。因此 SDIV 将面向动态转发路径部分设计为单流表的响应式下发流规则。

控制层

利用 SDN 技术，控制层中的控制器连接着网络中的每一个交换机（包括 AP）。通过下发 OpenFlow 流规则，控制器可以控制 IoV 中数据的传输。当网络状态发生改变时，交换机会发送相应的消息至控制器，使其获取最新的全局网络状态。通过全局网络状态信息，控制器可以将上层应用的策略，如路径选择或接入控制，转化为特定交换机上的 OpenFlow 流规则。

应用层

所有的 IoV 应用程序组成了 SDIV 应用层。应用程序复杂给 IoV 中的车辆提供服务，如实时查询服务，位置服务等。每个应用从控制层的控制器中获取网络信息，并根据自身策略进行决策。

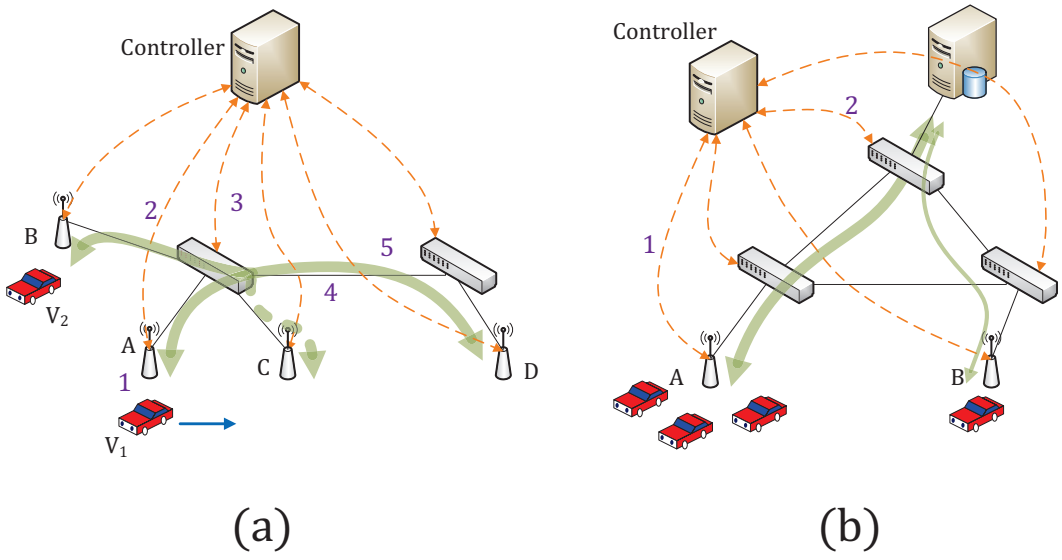


图 1.4: （a）：通过分析车辆状况，控制器可以提前下发流规则（虚线）以降低额外请求；（b）：控制器通过全局信息，实现智能带宽分配。

1.3.2 工作流程

给出 SDIV 架构后，我们这里通过两个场景来描述 SDIV 的工作流程（如图 1.4）。

图 1.4(a) 描述的是一个车辆试图接受从路况摄像头发过来的路况信息的典型场景。

第一步，车辆 V_1 发送请求至附近的路旁 AP；第二步，因为当前没有规则匹配该数据包，交换机（AP）A 转发该数据包至控制器；第三步，控制器识别数据包头，并根据车辆的请求、车辆状况（例如，位置，速度、方向等）以及网络状况下发流规则；第四步，根据下发的规则，沿途的交换机转发数据包至目的节点 D（即路况摄像头）；第五步，从摄像头 D 传送过来的数据经过上述二至四的步骤最终到达车辆 V_1。当车辆的数量增加时（即车辆 V_2 出现在了 B 附近），我们需要一个可拓展的方法去下发流规则，而非简单地重复上述步骤。同时，为了避免由于车辆移动性带来的数据传输中断，我们需要事先在 C 下发流规则。

图 1.4(b) 描述的是车辆通过附近 AP 与服务器进行连接并上传车辆信息，如速度和位置。在该数据上传场景中，控制器通过分析哪一个 AP 收到的数据包比较多，可以判断哪一个 AP 附近车辆较多，进而可以尽心更优化的资源调度。第一步，控制器分别从两个 AP 节点 A 和 B 收集数据包。当交换机收到数据包但没有相应匹配流规则时，则发送数据包至控制器。第二步，根据目的地址，控制器下发流规则到交换机中。此时，控制器可以发现 A 附近的车辆数量较多，因此可以分配更高的带宽给 A。

除此之外，在考虑大规模的多播和车辆的移动性时，SDIV 有着更大的优势。随着连接一个摄像头的车辆增多，可以想到用多播进行数据传输达到更高效的数据传输。虽然密集模式的多播在小规模网络有很好的效果，随着数据传输范围的变大，AP 上不断增多的 (S, G) 项（S：源 IP 地址，G：组播 IP 地址）以及为了建立 SPT（Shortest Path Tree）而周期性发送的广播消息限制了数据传输的进一步拓展。为了实现快速的嫁接，由于控制层和数据层的强

表 1.1: Pros and cons of traditional network technologies (multicast) and SDIV

传统技术	SDIV
劣势: 周期性广播消息	优势: 响应式方式
劣势: 路由器上保存 (S,G) 项	优势: 当需要时设置
劣势: SPT 不符合车辆行驶路线	优势: 根据车辆状态预测行驶路线
优势: 不需要控制器	劣势: 需要控制器

耦合，(S,G) 项需要被每一个路由器所保留（即使当前没有被该路由器使用）。如图 1.4(a) 所示，AP 节点 B 也需要保留 (S,G) 项为了使新来的车辆 V_2 快速获取数据。车辆的移动性也给传统的网络技术带来了困难。如图 1.4(a) 所示，当车辆 V_1 从 A 来到 C 时，车辆需要重新发送请求从而导致数据接收的中断，同时也给 D 节点的摄像头带来了困难。在我们的设计中，我们通过预测车辆可能行驶的路径解决移动性问题。我们会在预测出的路径沿途的 AP 上实现下发实现多播的流规则。而沿着 SPT 的数据传输路径进行多播无法解决移动性问题，这是因为数据传输的最短路径不一定符合车辆行驶的路径。

以下表格总结了 SDIV 和传统网络技术（多播）的优劣势比较。相比传统多播方法需要周期性广播消息以搭建 SPT，SDIV 通过利用 OpenFlow，即只有当数据包没有匹配流规则时才需要下发所需规则，实现了响应式的流规则下发方式，进而去除了周期性广播消息的需求。该响应式方式也让交换机可以保留最少数量的流规则，而非传统方式中的大量的 (S,G) 项。同时，基于全局的网络状态信息，控制器上的应用可以选择车辆可能行驶的路线并提前设置流规则进而实现数据传输的连续性。

1.4 优化的流规则下发

在描述完 SDIV 的架构以及如何工作后，我们接下来会给出流规则下发问题的解决方案，并说明简单的下发流规则会导致的后果。

在实时查询服务中,如果控制器简单地根据车辆的请求下发流规则,则由于 Ternary Content-Addressable Memory （TCAM）有限的存储容量，流表的容量大小会成为系统性能的瓶颈。因此，我们需要建立一个紧凑的流表以放入更多的决策策略。对于每个数据流的建立，都需要两条流规则来分别处理发送请求和接受数据。即使我们只考虑第二个数据流，因为请求的数据流仅在开始时使用，并且可以通过 OpenFlow 交换机中的超时机制删除，我们仍然需要为每个流至少分配一个规则。似乎每个流使用一个规则对于流表来说足够紧凑，但实际上车辆会试图同时连接多个监控摄像头。因此，我们需要在每个交换机上对每个此类数据流设置规则，即使这些数据流都具有相同的目标。总而言之，我们需要减少转发到控制器的数据包数量并同时建立紧凑的流表。

图 1.5描述了一个实时查询服务的基本工作流程。假设 V_1 想要获得道路（或十字路口）E 的状况。这里，状况可以是视频或存储在路边设备中的任何其他种类的数据。第一步，V_1 向 AP 节点 A 发送请求，然后数据流转发到 E。在确认来自 V_1 的请求后，E 将数据发送到 AP

节点 A。最后，A 将数据传输到 V_1 并完成数据传输。整个过程很简单，但可能会带来显着的性能损失。如图 1.5所示，考虑另一车辆 V_2 想要获得 E 信息的情况。同时，考虑 V_2 在 V_1 的前方。当 V_2 重复与 V_1 相同的过程时，会建立另一条从 E 到 B 的路径，这使得交换机 D 对于相同的路径安装两条规则。结果，随着连接到 E 的车辆数量的增加，沿途的交换机的流表大小可能达到最大值，从而降低了实时查询服务的性能。此外，当 V_1 移动到另一个地方时，它也需要再次发送请求。

接下来，我们考虑另一个场景来说明优化的流规则下发的重要性。V_1 想要同时从 E 和 F 获取数据。交换机 B 和 D 需要设置更多规则以使数据流传输到 A，即使两个流（$E \rightarrow A$ 和 $F \rightarrow A$）具有相同的目的地。如果 V_3 请求相同的服务，这种情况会变得更加复杂。为了使 C 下面的 V_3 接收数据，需要构建一条新路径（即，相同的数据流将在三辆车辆所连接的交换机处分开，因而该交换机成为分支节点），这再次增加了流表的大小。车辆的移动性也增加了复杂性并降低了实时查询服务的性能。

为了解决规则下发问题并减缓流表大小的增长，我们考虑将无线数据平面（用于车辆和 AP 之间的通信）与有线数据平面（用于交换机之间的通信）分开，并提出目的地驱动模型用于有线数据平面。当车辆从附近的 AP 接收数据时，它们不关心数据如何在有线数据平面中传输。相反，它们只想要一个持久的连接，即使它们的位置随着时间而变化。为了有效地进行分离，我们引入了三种技术：1. 使用多播地址作为摄像头到车辆的最后一跳地址；2. 根据车辆的状况预先在最可能的路径中安装规则；3. 当数据包到达分支节点时修改数据包头。

1.4.1 使用多播地址作为最后一跳

我们利用多播地址来解决移动性问题并增强 SDIV 的可扩展性。每个监控摄像头都有一个唯一的组播地址，可以通过传统网络中的相同方法从 MAC 地址生成。多播地址可以在从摄像头到车辆的最后一跳中用于无线数据平面中的数据传输。车辆 V_1 接收目的地址为 E 多播地址的数据包，如图 1.5所示。使用多播地址的优点是显而易见的，即在已配备规则的 AP 范围内的每个车辆可以在不发送新请求的情况下获得实时数据。此外，在实时查询服务中，必须将第一个用户请求包转发到控制器以安装规则。如果每个请求都需要联系控制器进行规则安装，则会导致控制器成为严重的瓶颈。多播地址可以有效地解决这个问题。在第一辆车通过控制器的参与并连接摄像头后，数据流开始进行多播传输（沿着到达目的地的路径），并且对同一摄像头请求的每辆新车都会直接接收数据而不用发送请求，这降低了联系控制器的频率。例如，在图 1.5中，如果有另一车辆想要查看 E 的状况，它可以在 V_1 发送请求后不用发送请求而直接接收数据。

由于 OpenFlow 交换机中 TCAM 的大小有限，我们需要从流表中删除无用的规则。在我们的设计中，我们使用 OpenFlow 中的超时机制来删除规则。对于每一流规则，交换机都会维护一个变量，该变量表示是否在一段时间 T 秒内，没有数据包到达匹配该流规则。实际上，OpenFlow 中，通过向流表条目添加 “空闲时间” 值来实现超时机制。与相等的超时时间相比，我们根据离目的地的距离在交换机中选择不同的超时值。现实情况中，随着离目的地越近，对目的地感兴趣的车辆数量会越大。因此我们在离目的地更近的交换机设置更大的超时值。例

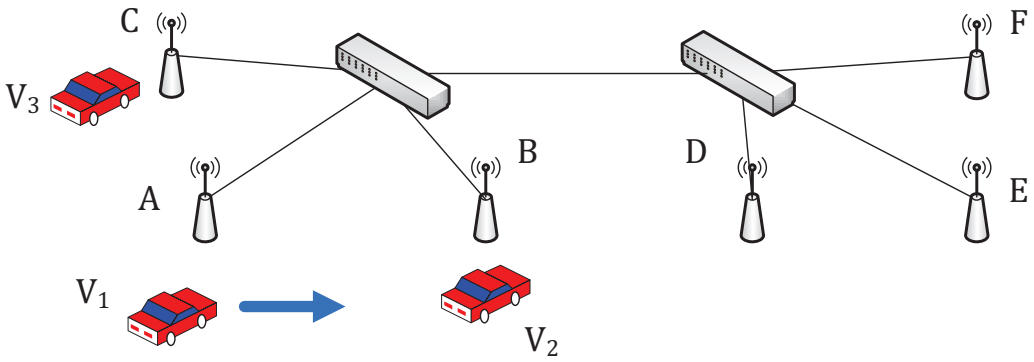


图 1.5: V_1 同时连接 E 和 F。V_2 在 V_1 到 E 的路径上并向 E 请求数据。V_3 也同时向 E 请求数据但在不同路径。

如图 1.5所示，超时值 T_i 在交换机 i 符合约束条件：$T_A < T_B < T_D < T_E$。离信息源越近，数据流的变化越小因为这些流会合并成稳定的数据流。而稳定的数据流需要比不稳定数据流具有更长的超时。

1.4.2 路线预测并提前下发流规则

为了处理 SDIV 中车辆的移动性，有必要预测车辆选择的最可能路线，然后沿路线预先安装规则，以便在车辆移动时保持数据传输不中断。预测这种路线的一种简单方法是计算最短路径，但实际上并不总是正确。我们设计算法 $PathFind(s, d, v)$（算法 1.4.2），来找到拓扑中最可能的路径。这里 s 代表当前位置，d 表示目的地，v 表示车辆的方向。$A(s, c)$ 表示 s 和 c 的角度，$D(n, d)$ 表示 n 和 d 之间的欧几里德距离。

作为准备工作（第三至五行），$PathFind$ 根据车辆的状况（例如，方向）过滤并得到的路径的可能的第一节点，这是因为车辆很少在街道中掉头。如图所示在十七至二十行，我们通过选择具有 $D(p, n) + D(n, d)$ 最小值（即所选位置和目的地之间的距离）的节点来递归地应用 $Find$ 来计算结果路径。最后，$Find$ 将目标标识为下一个节点并返回结果（第十二至十四行）。

我们在图 1.6中应用 $PathFind$。考虑一个位于 A 的车辆计划找到 F 的路线。首先，它根据当前位置的车辆方向设置 B 为 O 的值。然后发现 D 与 C 相比有更短的路径到 F。因此，它选择 D 重新开始，发现 D 直接连接到 F 并完成。结果 $A \to B \to D \to F$ 比最短路径 $A \to E \to F$ 更合理，因为实际上很少看到倒车行为。虽然路径 $A \to B \to D \to F$ 的时延比路径 $A \to E \to F$ 长，但是当车辆发现没有收到数据时，会发出更多请求。在这种情况下，车辆可能选择 B 作为其下一个位置，因此得到的路径优于最短路径，因为车辆可以在不发送另一个请求的情况下接收数据。在图 1.6中，在预测路线上的交换机上安装规则（$A \to B \to D \to E$）后，多播地址也会使 V_2 直接接收数据而不用发送一个新的要求。

Algorithm 1 PathFind(s, d, v)

```
1: Algorithm PathFind(s, d, v)
2:   put s into set N, R;
3:   for each child node c of s
4:     if the angle between v and A(s, c) is less than 90 degree then
5:       put c into set O;
6:   Find(s, O, d);
7:   return R;
8: Procedure Find(p, O, d)
9:   put p into set N;
10:  for each n in O
11:    if n is d then
12:      put n into set R;
13:      finish;
14:    put n into set N;
15:    remove n from set O;
16:    calculate D(n, d), D(p, n);
17:    if D(p, n) + D(n, d) < max then
18:      max = D(p, n) + D(n, d);
19:      r = n;
20:  put r into set R;
21:  for each child node c of r
22:    if c not in set N then
23:      put c into set O;
24:  Find(r, O, d);
25:  return;
```

1.4.3 分支节点修改数据包地址

在本小节中，我们将展示如何在交换机上安装规则。如算法 XXX 所示，其主要思想是我们通过利用 OpenFlow 特性在分支节点（第八至九行）添加数据包头修改操作。输入 s 表示数据源，d 表示车辆的当前位置，$path$ 由 $PathFind$ 计算生成。当数据包匹配规则并修改匹配字段时，它将在转发到特定端口之前应用修改。修改后的地址使数据包转发到直接连接到车辆的交换机。通过使用地址修改算法，我们让具有不同源地址但相同目的地地址的数据包匹配相同的规则，从而减少规则数量并减少流表大小。我们将此方法称为目标驱动模型，因为它只需要匹配目标地址。

如图 **??**所示，数据流 F_1 和 F_2 均来自监控摄像头 S_1，而 F_3 来自 S_2。F_1 由 V_1 请求。F_2 和 F_3 的目标是 V_2。假设 V_1 的当前位置是 C，V_2 的位置是 D。在单播情况，F_1 和 F_2 需

Algorithm 2 ModifyAddress$(s, d, path)$

```
1:  Algorithm ModifyAddress(s, d, path)
2:    for each node n in path do
3:      nx = the next node of n;
4:      setRule = false;
5:      for each rule r in n do
6:        no = the node connecting the output port of r;
7:        if r matching s and no != nx then
8:          act = forward to nx and modify the destination address to d;
9:          emitRule(matchFor(d), act);
10:         setRule = true;
11:       else if r matching s and no = nx then
12:         setRule = true;
13:     if setRule == false then
14:       act = forward to nx;
15:       emitRule(matchFor(d), act);
16:   return;
```

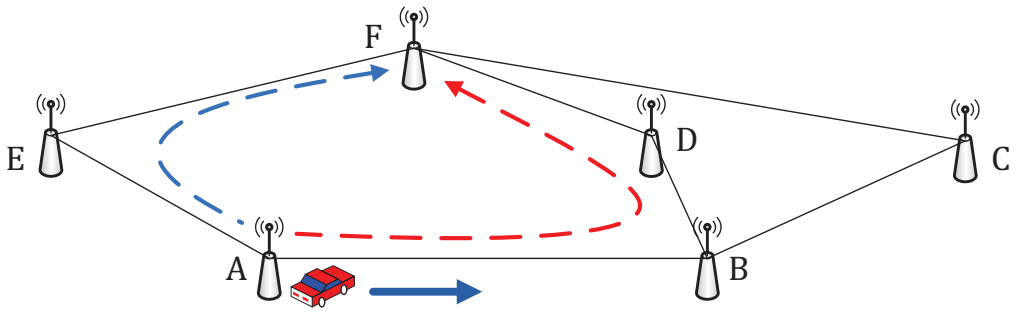


图 1.6: 蓝色虚线代表当前位置到目的位置的最短路径。红色虚线代表车辆可能选择的路径。

要至少两个规则在交换机 A 和 B；在多播情况下，F_2 和 F_3 也需要至少两条规则在交换机 A 和 B。然而这是低效的，因为 F_1 和 F_2 来自同一节点，而 F_2 和 F_3 具有相同的目的地。我们在交换机 B（作为分支节点）上修改 F_1 和 F_2 数据包头中的目标地址。通过设置 F_1 和 F_2 的目标地址（即 $F_1 \rightarrow C$，$F_2 \rightarrow D$），减少了交换机中的规则数量并可以提供更多服务。再假设 V_2 在开始时向 S_1 发送请求。每个交换机只需要一个规则来转发 F_2。而当 V_2 从 S_2 获取数据时，交换机的规则仍适用。最后，当 V_1 需要来自 S_1 的数据时，它只需在交换机 B 上额外增加一个规则并匹配 (S_1, D) 以及一个修改操作。这里我们使用两个参数来表示包头，即（源地址，目的地址），以及流表中的匹配条件。该目标驱动模式将目标地址设置为流表中的匹配条件，合并同一目标的数据流的流规则，以减小交换机中流表的大小。在图 1.5中，当 V_3 加入时，它需要在分支交换机处进行地址修改以优化规则安装。

接下来，我们通过两种情况来分析优化的规则下发：1. 一台服务器连接多台车辆；2. 一

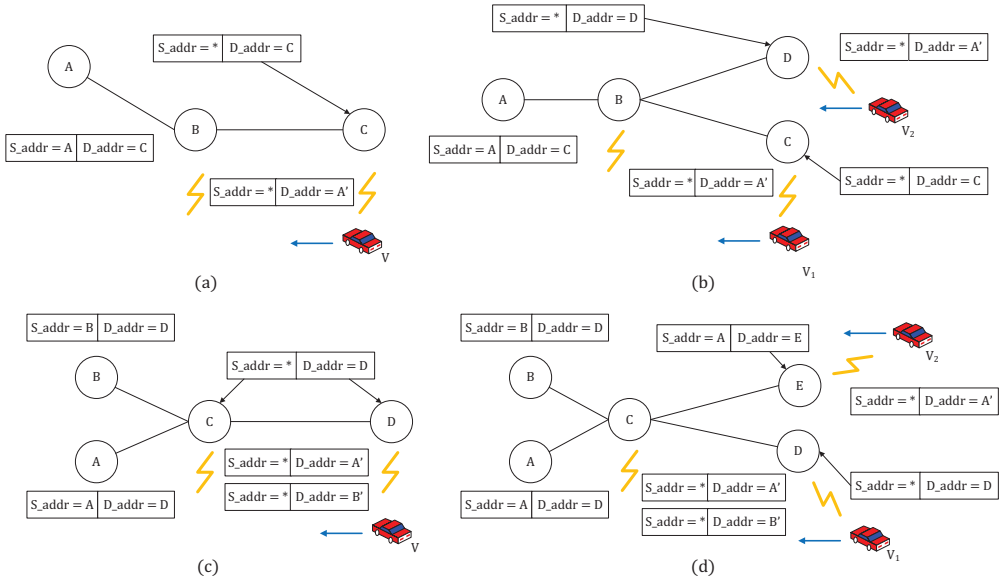


图 1.7: （a）：1 对 1 场景即 V 接受 A 的数据；（b）：N 对 1 场景即 V_1 和 V_2 同时接受 A 的数据；（c）：1 对 N 场景即 V 同时接受 A 和 B 的数据；（d）：N 对 N 场景即 V_1 接受 A 和 B 数据，而 V_2 接受 A 数据。

台车辆连接多台服务器。对于第一种情况，传统的规则安装仅根据源地址将数据包转发到给定端口，而在路径的每个交换机上需要一个规则。在我们提出的规则下发方法中，我们在每个交换机上需要一个规则用于转发，并且需要在分支节点处修改数据头的操作。由于修改地址操作可以与 OpenFlow 中的转发操作组合作为一个规则操作，因此与传统规则下相比，规则的数量不会改变。对于第二种情况，我们假设有一辆车辆（v）连接到多个服务器（$s_1, s_2, ...s_n$），并且我们规定 s_i 离 v 的距离为 h_i（跳数）。设 L_i 表示用于 v 和 s_i 之间数据传输的一组链路。因此，我们有 $|L_i| = h_i$。因此传统方法需要 $\sum_{i=1}^{n} h_i$ 条规则，而通过我们优化的规则安装，我们需要 $|L|$ 条规则，其中 $L = \bigcup(L_1, L_2, ...L_n)$。考虑一个极端情况，任何两组链路 L_i 和 L_j 都没有共享链路，则我们有 $|L| = \sum_{i=1}^{n} h_i$。在这种情况下，我们提出的方法所需的规则数量与传统方法相同。对于其他大多数场景，$|L| < \sum_{i=1}^{n} h_i$，这意味着我们提出的方法减少了规则的数量。总而言之，我们优化的规则的最坏情况包括上述第一种情况和第二种的极端情况不会改变规则的数量，而最好的情况可以在一辆车连接到多个服务器时减少很多规则。

1.4.4 示例

这里分为四种场景来给出我们方法是如何工作。如图 1.7所示，通过分析实时查询服务中的数据传输过程，我们给出交换机上的实际流规则。

1 对 1 场景

在最简单的情况下，只有一辆车向一台监控摄像头请求实时查询服务。如图 1.7(a) 所示，V 向摄像头 A 发送请求，然后从 A 接收数据流。来自 A 的数据包是 (A, C)，B 和 C 的匹配条件都是 $(*, C)$。从 C 多播的数据包是 $(*, A^{'})$，其中 $A^{'}$ 表示 A 的多播地址。在这种情况下，$PathFind(s, d, v)$ 计算出的路径是 $C \rightarrow B \rightarrow A$。然后我们在 B 下发匹配 $(*, A^{'})$ 的多播规则。当 V 到达 B 时，它可以不间断地接收数据。

N 对 1 场景

当摄像机连接到多个车辆时，需要修改分支节点处的目的地地址，如图 1.7(b) 中的 B 所示。对于 OpenFlow 交换机，它在一个规则上可以支持一组动作（例如，包头修改和转发动作）。因此，修改操作不需要额外的规则。开始，A 发送数据包 (A, C) 因为 C 处的 V_1 发送的请求遭遇 D 处的 V_2。当 V_2 发送请求时，控制器（图中未显示）下发新的具有修改操作的 OpenFlow 规则。最终，交换机 B 将两个包 (A, C) 和 (A, D) 转发到不同的端口。由 B，C 和 D 的转发的数据流均为 $(*, A^{'})$。

1 对 N 场景

在该场景中车辆需要同时连接到多个摄像头。如图 1.7(c) 所示，V 想要同时从 A 和 B 接收数据。A 和 B 生成的数据包是 (A, D) 和 (B, D)。交换机 C 和 D 的匹配条件都是 $(*, D)$。而这里只需要一条规则来满足不同数据包的要求。当数据包 (A, D) 达到 C 时，规则 $(*, D)$ 与之匹配，然后转发到 D。交换机 D 遵循与 C 相同的过程。同时 C 和 D 还需要更改其数据包的目标地址以进行多播转发。C 和 D 的多播数据包是分别为 $(*, A^{'})$ 和 $(*, B^{'})$，其中 $A^{'}$ 和 $B^{'}$ 是多播地址。

N 对 N 场景

我们将 1 对 N 和 N 对 1 场景组合并形成一个更常见的 N 对 N 场景。如图 1.7(d) 所示，V_1 需要来自 A 和 B 的实时数据，V_2 需要来自 A 的数据。对于匹配 $(A, *)$ 的数据包，在 C 处进行数据包的修改，因为当数据源 A 同时将数据包传输到 D 和 E 时，C 是一个分支节点。来自 B 的数据包无需修改即可转发到给定端口。

N 对 N 场景作为一种普遍的情况，给出了有关如何实现优化的规则下发的详细信息。我们可以看到，我们提出的优化规则下发方法的流表大小的理论上限与车辆同时连接的设备数量有关。N_d 表示设备数量，R_0 表示传统方法的规则数量，R 表示优化的规则下发方法的规则数量。我们可以得到 $R = R_0/N_d$ 作为流表大小的上限。在这种情况下，所有设备都连接到相同的交换机。

1.5 实验评估

我们使用 Floodlight [?] 作为控制器以及 Mininet [?] 来构建 SDN 环境，以便评估 SDIV 中优化的流规则下发。我们在服务器上运行 Floodlight，配置为 16 AMD Opteron(tm) 处理器 6172 和 16GB 内存。服务器安装了 Linux 内核版本 2.6.32。我们在另一个单独的服务器上运行 Mininet。服务器之间通过 10Gbps 以太网连接。

1.5.1 优化的规则下发

我们使用上海市车辆行驶线路的真实数据 [?] 作为我们的实验场景，以显示优化的规则下发的优势。图 1.8(a) 显示了上海人民广场周围的场景。车辆的行驶痕迹由不同时间的 GPS 数据组成。由于我们使用上海市的实际数据，车辆数量有限，但足以显示所提方法的效果。在 t_1 时，该区域只有五辆车。在 t_2，还有两辆车加入。t_3 时，V_6 出现在该区域内。车辆的出现时间和消失时间如图 1.8(b) 所示。任何两个时刻之间的间隔时间是 120 秒。由于这些车辆无法在确切时间出现，例如 t_1 和 t_2 间隔时间为 120 秒，我们会将车辆显示在 t_i 如果 $|t_i - t_{real}| < 30$ 其中 t_{real} 是车辆的实际出现时间。虽然这些 GPS 标记在同一时刻可能没有完全相同的时间戳，但通过限制 GPS 标记的时间戳和所标识的时刻为一定范围之间，可以说这些车辆非常接近这些位置。在这种情况下，我们将时间范围设置为 30 秒，以确保每个 GPS 的标记都能反映当时的真实位置。

图 1.8(c) 是图 Fig. 1.8(a) 的简化版本。如图 1.8(a) 所示，具有不同特征的箭头根据 GPS 标记中这些时间戳的顺序来说明车辆的方向。我们假设在 GPS 标记周围总有一个路边 AP（图 1.8(c) 中的交换机），以便随时可以随时传输数据。我们使简化版本中的每辆车在出现的立刻都有数据传输需求。因此，地铁 8 号线和地铁 2 号线交叉点附近的 V_1 和 V_2 想要从 B 和 C 接收数据，并且具有相同路径即沿着地铁 8 号线。V_3 位于两条道路汇合点并需要 B 和 C 的数据。V_4-V_5-V_6 具有相同的目标 A，但选择不同的路径。V_7 询问 D 的数据。V_8 只有目标 B。所有这些车辆都需要持久的数据流服务，直到他们移动到目的地。我们需要在每个必要的交换机上安装规则以满足他们的要求。

我们的对比方法是一种简单地基于源地址将数据包转发到给定端口的通用方法。通过优化的规则下发，我们合并具有相同目的（目的地址驱动模式）的流规则。结果如图 1.9(a) 所示。在 t_4 上的下降是由于我们设定的规则超时。图 1.9(b) 显示了不同时刻的车辆数量。图 1.9(c) 显示了两种方法中不同车辆的延迟时间。如果车辆同时连接两个摄像头，我们在此评估的延迟时间是最长的。如我们选择 C 而不是 B 来评估 V_1 的延迟时间。

另外，我们可以从图 1.9(c) 中看到我们优化的规则下发方法虽然需要在数据包报头中进行地址修改，但几乎不会影响数据传输的性能。我们减少了规则数量并压缩了流表中的空间，以实现更好的可管理性和可拓展性。特别是，从图 1.9(b) 看出，与对比方法相比，规则数减少了 60‰

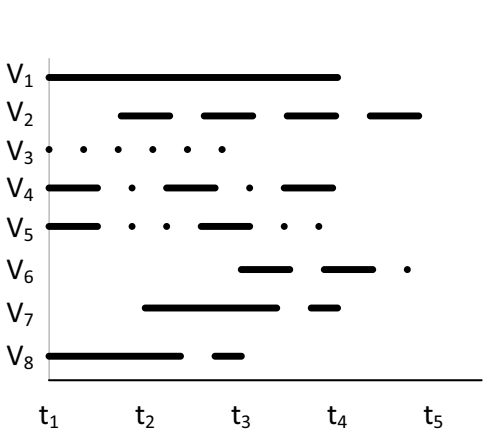
1.5.2 时延分析

我们对两种模式（1 对 N 和 N 对 1）进行研究分析，并说明它们如何以不同方式影响结果。我们将规则数和延迟时间与对比方法进行比较，以表明它不会影响数据传输的性能。图 1.10(a) 显示 N 对 1 模式，即有多个车辆连接到一个监控摄像头（B）。我们选择具有最长路径的 A 来评估延迟时间。对比方法需要在每个交换机上安装一个规则，并匹配源地址。对于所提的方法，从 B 传输到 A 的每个数据包都会修改包头（目标 IP 地址），然后转发到下一个交换机。图 1.10(b) 表明，随着交换机数量的增加，两种方法的延迟时间都变大，但没有显着差异。我们得出结论，优化的规则下发方法中的数据包修改过程不会影响性能。图 1.10(c) 显示了交换机的规则数量。红色（右侧）行表示节点 B 同时通过网络将数据流传输到每个节点的对比方法。每个交换机只需要一个规则基于 $(*, B')$ （B' 是多播地址）。黑色（左侧）线描述了优化的规则下发方法，该方法修改每个分支节点处的数据包目标地址，然后将数据包转发到给定端口。每个交换机只需要一个规则。

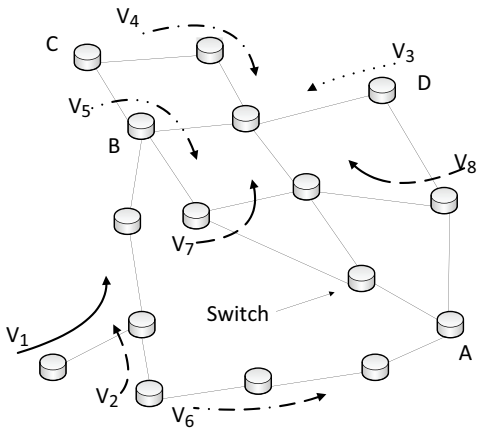
图 1.10(d) 显示 1 对 N 模式。其中车辆 V 同时连接多个摄像机（B, C, D）。图 1.10(e) 显示了根据不同交换机数量的延迟时间。如果没有优化的规则下发，每个交换机的每个源节点都需要三个规则，$(B, *), (C, *), (D, *)$，这是源地址驱动模式（基于数据源转发数据包）。通过优化的规则下发，我们只需要一个规则，即每个交换机上的 $(*, A)$，它仍然可以支持 N 对 1 模式。图 1.10(f) 显示了两种方法的规则数量。

我们从结果中得出两个结论。首先，分支节点处的数据包修改对数据传输的性能没有影响。其次，对于优化的规则下发的最佳情况（1 对 N），压缩后规则的数量与数据源的数量成比例。对于优化的规则下发的最坏情况（N 对 1），交换机的规则数量等于对比方法的规则数量。

(a)



(b)



(c)

图 1.8: （a）：上海市人民广场附近的场景图；（b）：车辆的出现和离开时间；（c）：（a）的简化图并保护车辆方向信息。

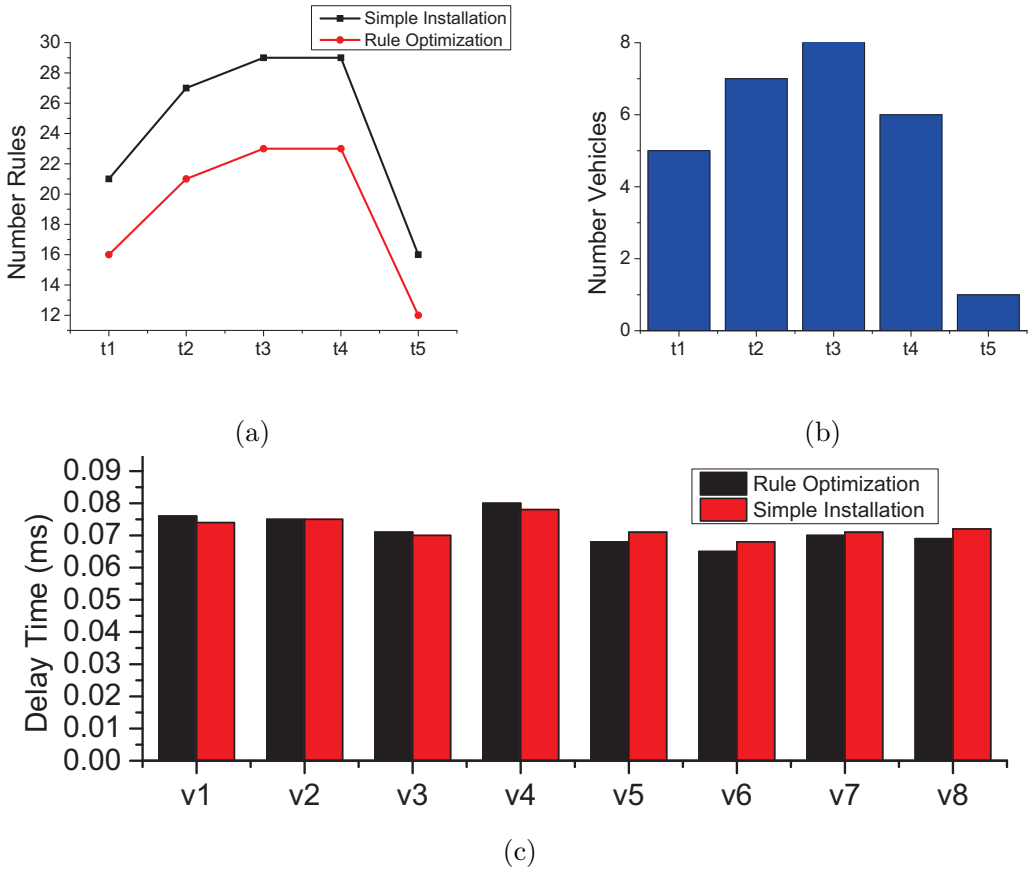


图 1.9: （a）：两个策略的规则数量差异；（b）：不同时刻的车辆数量；（c）两个策略的车辆延迟时间。

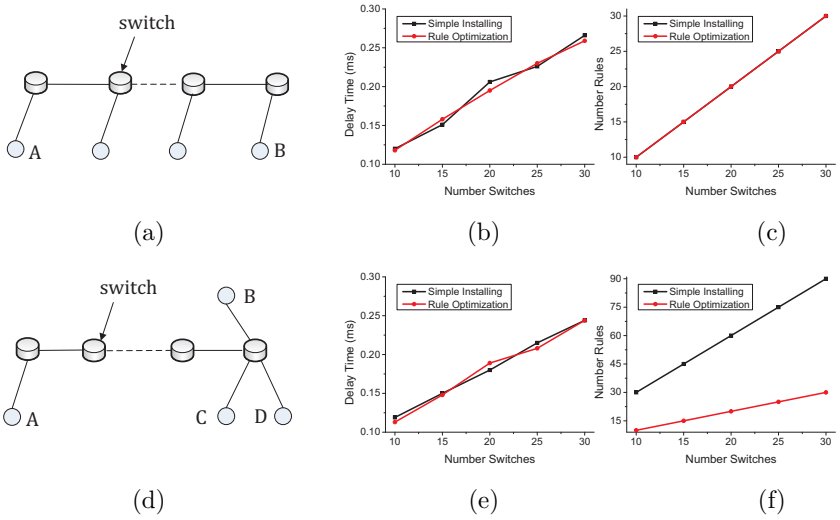


图 1.10: （a）：N 对 1 模式的拓扑；（b）：对于（a）中不同交换机数量的延迟时间；（c）：对于（a）中不同交换机数量的规则数量；（d）：1 对 N 模式的拓扑；（e）：对于（d）中不同交换机数量的延迟时间；（c）：对于（d）中不同交换机数量的规则数量。