第一章　绪论

1.1　引言

软件定义网络（Software Defined Network，SDN）将网络控制平面和数据平面进行分离，并通过逻辑上集中式的 SDN 控制器向上层应用提供网络全局视图，使灵活、高效的网络管理变得可能 [?, ?, ?]。对于底层数据平面中的网络设备，通过南向接口（例如 OpenFlow 协议 [?]），SDN 控制器上的应用可以对其进行配置，进而控制网络中数据包的传输。然而，让上层应用直接调用底层网络设备的低级配置接口对其进行配置，则会导致系统非常容易出错，影响可拓展性。因此，研究人员在 SDN 中提出高级编程模型的概念 [?, ?, ?]，对上层应用提供对网络配置的高级抽象接口，并自动地将上层应用表达的高级抽象网络策略转化为底层网络设备的低级配置。然而，随着 SDN 概念的普及，以及底层网络设备技术的迅速发展，SDN 高级编程模型不断迎接着新的挑战。

一方面，随着 SDN 概念的普及，上层应用希望 SDN 高级编程模型提供的高级抽象接口可以不断地支持新的特性。从最原始的对单个交换机的配置，到可以对网络中多个交换机的配置，再到可以对网络中交换机和特定网络功能节点（如防火墙）的统一配置。从对网络管理的意图 [?]，到高级 SDN 程序（即逻辑上可以应用于网络中所有数据包的 onPacket 函数），再不断地丰富高级 SDN 程序的语法（如支持循环结构，支持路由代数 [?] 等）。这些上层应用迫切希望的新的特性，给高级编程模型所提的自动地生成底层网络设备的低级配置带来了巨大挑战。

另一方面，随着底层网络设备技术的迅速发展，底层网络设备在不断变得高效、灵活的同时，其架构也变得越来越复杂。从 OpenFlow1.0 [?] 的单流表架构，到 OpenFlow1.1 [?] 的多流表流水线架构，再到协议无关并可自定义的多流表流水线架构的 P4 [?]。最初的单流表架构虽然简单，但由于多

个匹配域的叉乘（cross-product），单流表架构并不高效。多流表流水线架构解决了匹配域的叉乘问题，但其架构也变得复杂从而影响流表规则的下发：要考虑给哪一个流表下发。P4 的协议无关并可自定义的多流表流水线架构实现了非常灵活的网络配置，但使用时不光要考虑流表规则的下发，而且（实际上要先于流表规则下发）要考虑生成多流表流水线结构。支持有状态操作的可编程交换机 [?, ?] 其强大的灵活性也伴随着复杂的架构。这些新诞生的底层网络架构同样给高级编程模型带来了巨大挑战。

随着高级抽象接口支持的特性越来越多，底层网络设备架构变得越来越复杂，两者之间结构的差异也会变得越来越大。而 SDN 高级编程模型始终需要考虑的问题则是，如何高效地填充两者之间不断变大的鸿沟，即尽可能地将上层应用表达的高级抽象网络策略转化为符合底层网络设备架构的高效的低级配置。

在这章中，我们先从面向单交换机和面向全网络两个大方向进行研究背景介绍。其中对于面向单交换机，我们先介绍可编程数据通路（datapath）的研究，再介绍面向单交换机编程模型的研究；对于面向全网络，我们先介绍面向一般网络编程模型的研究，再介绍针对特定网络编程模型的研究。再结合介绍内容，给出论文中主要的研究挑战以及相关解决方案。最后给出论文整体架构。

1.2 研究背景：面向单交换机

1.2.1 可编程数据通路的研究背景

可编程数据通路的研究可以分为两大类：基于硬件的可编程数据通路，基于软件的可编程数据通路。

基于硬件的可编程数据通路：灵活的数据包分类（即可以利用通配符对数据包匹配域的任意组合进行匹配）是实现可编程数据通路的基础。基于 TCAM 的三元匹配功能（即 0，1，x），可以非常高效地实现数据包分类 [?, ?, ?]。然而，由于其的电力消耗，以及占用的电路空间，TCAM 不具备很好的可拓展性。Naous 等人 [?] 将数据通路中的匹配表分为准确匹配和通匹配，并分别利用 SRAM 和 TCAM 进行实现。最终他们在 NetFPGA 平台上实现了 OpenFlow 交换机。Jiang 等人 [?] 基于决策树的数据包分类算法，在单个 FPGA 上放置了 10K 条可以支持 5 个匹配域的规则或 1K 条可以支持 12 个匹配域的规则。

由于单纯利用交换机芯片无法实现灵活的数据包处理，研究者开始考虑利用交换机中的其他资源（如 CPU）来处理一部分数据包操作。Luo 等人 [?] 利用基于网络处理器的加速卡实现 OpenFlow 交换机，并降低了百分之二十的网络数据包延迟。Lu 等人 [?] 利用 CPU 来辅助 ASIC 交换机对数据流的处理。具体来说，他们利用 CPU 实现基于流转发的大容量匹配表以及利用 DRAM 存储网络中突发数据包。Mogul [?] 等人利用 CPU 实现交换机中的软件定义计数器，以达到对计数器相关信息的灵活操作。类似的，他们利用 DRAM 存储计数器，并通过 CPU 对其进行更新操作。数据包匹配后会在 buffer 中存储匹配记录，然后 CPU 根据 buffer 记录更新 DRAM 中数据。

对于单流表匹配结构对流表容量要求较大（由于匹配域叉乘），使用起来并不高效等问题，研究人员考虑用多流表流水线作为可编程数据通路的基本结构。OF-DPA [?] 是 Broadcom 提出的 OpenFlow 数据平面抽象。OF-DPA 与 OpenFlow v1.3.4 兼容，其底层为多个具有固定匹配域流表的流水线结构。PicOS [?] 与 OF-DPA 类似，在底层为多个具有固定匹配域流表的流水线情况下，向上兼容 OpenFlow 协议。FlowAdapter [?] 设计了具有三层的交换机架构。其中上层为 OpenFlow 软件数据平面，并通过中间的 Flow Adapter 转化为底层的 OpenFlow 硬件数据平面。

对于上面所述的多流表流水线架构，用户只可以下发流规则，不可以更改流水线结构。因此，为了追求更灵活的可编程数据通路，研究人员开始考虑自定义的多流表流水线架构。Forwarding Metamorphosis [?] 提出 RMT 架构，表示 Reconfigurable Match Tables。RMT 架构包含 ingress processing、egress processing、以及中间的队列结构。其中 processing 由 parser、deparser、以及中间的多个 stage 组成。一个流水线上的逻辑流表可以由一个或多个 stage 共同实现，因此 RMT 可以灵活地实现具有不同结构的多流表流水线。同时其 parser 也可以自定义。最终，RMT 实现了协议无关可自定义的多流表流水线架构。dRMT [?] 删除了 RMT 中 stage 与流表的绑定关系，即任何 stage 可以访问任何流表；删除了 stage 与 stage 的依赖关系，即数据包可以进入任意 stage 而无需经过前面的 stage，实现了更灵活更高效的数据包处理。为了考虑更丰富的数据包操作，PISA [?] 概括了 RMT 架构并实现了更灵活的数据包操作指令（atoms）。OpenState [?] 考虑了有状态的操作。

基于软件的可编程数据通路：Linux 内核 [?]，DPDK [?]，Netmap [?]，Click [?]

等工作需要对底层实现具有相当的了解才能在上面构造软件交换机，这对网络程序员需要快速适应这些平台并开发新特性增加了困难。Open vSwitch（OVS）[?] 考虑了向其添加流表规则的接口，但不可以自定义地配置协议以及操作。Oko [?] 通过 Berkeley Packet Filter (BPF) 对 Open vSwitch 进行拓展，使其可以支持有状态过滤器。Pisces [?] 是一个可编程的协议无关的软件交换机，程序员可以在上面通过 P4 语言进行配置，并可以输出以 OVS 为目标的底层代码。

1.2.2 面向单交换机编程模型的研究背景

2006 年，IETF 网络配置工作组提出了 NETCONF [?] 作为用于修改网络设备配置的管理协议。NETCONF 允许网络设备发布可通过其发送以及接受可扩展配置数据的 API。另一种广泛部署的网络设备管理协议是 SNMP [?]。SNMP 使用结构化管理接口（SMI），以获取存储于管理信息库（MIB）中的数据。通过 SNMP，网络管理员可以改变 MIB 中的变量以修改配置设置。

OpenFlow [?] 是目前 SDN 中所使用最广的南向接口标准。OpenFlow 对于支持 OpenFlow 转发设备提供了一个共同说明，以及对于 SDN 数据平面和控制平面之间的通信通道提供了标准。从 OpenFlow1.1 [?] 开始，OpenFlow 标准增加了多表、组表以及流水线处理技术。对于支持 OpenFlow 多流表交换机，当数据包到达时，和第一个流表中的流项进行匹配。匹配成功时执行相应操作，可以是应用指令或者跳转至指定其他流表，实现数据包在多个流表之间的转移。组表可以用来实现多播。从 OpenFlow1.3 [?] 开始增加 Meter 表，用来实现简单的 QoS 操作。

OVSDB [?] 的设计是向 Open vSwitch 提供一个高级的管理接口。除了 OpenFlow 的基本配置流表的能力外，通过 OVSDB 可以创建多个虚拟交换机的实例，以及向交换机接口设置 QoS 策略。

目前面向多流表流水线的交换机配置语言包括 Concurrent Netcore [?] 以及 P4 [?]。Concurrent Netcore 可以用来指定路由策略以及多流表中数据包处理的顺序。P4 作为目前占有统治地位的配置自定义多流表流水线的语言，P4 不光可以支持自定义的流水线结构，而且可以自定义数据包的 parser 以实现协议无关的流水线。

与 P4 需要手动地指定每一个流表的格式以及流水线的结构，Packet transactions [?] 向用户提供了一个高级编程模型，可以用来自动地生成底层

数据通路的结构。通过该高级编程模型，用户可以编写 onPacket 函数并提交给 Packet transactions，后者会自动生成底层数据通路结构以及配置。在生成底层数据通路结构时，Packet transactions 会合并用户 onPacket 函数中的语句。

1.3 研究背景：面向全网络

1.3.1 面向一般网络编程模型的研究背景

一般网络这里是相对特定网络（如车联网等）而言的概念。我们考虑面向一般网络的编程模型不会对网络实现的功能以及应用场景进行限制。

在 1990 年代中期，Active Networking [?, ?] 提出了通过可编程网络基础架构，可以实现定制化的服务。主要有两种方法被考虑：1. 通过用户可编程交换机并使用 in-band 负责数据传输，out-of-band 负责管理信道；2. 将需要执行的程序附带在数据包上，并在交换机或路由器上执行对应的程序。

基本 SDN 控制器如 [?, ?, ?, ?]，向下通过 OpenFlow 协议与底层交换机通信，获取网络状态信息并传给控制器上的应用程序。通过响应式模式，上层应用收到底层数据包后，进行相关处理，再通过控制器提供的接口对相关网络设备进行配置。ONOS [?] 提供意图（intent）的方式对网络中数据流的传输进行控制。

具有 SDN 编程抽象的系统如 FML [?]，一种基于流的管理语言提供了高级编程模式来指定网络安全相关配置。Onix [?] 引入了 NIB（Network Information Base）抽象，以便应用程序通过读写存储在 NIB 中的键值对来修改流表。Frenetic [?]，Pyretic [?]，Nettle [?]，提供了 SDN 编程语言。其中 Frenetic 的 NetCore 支持特定的策略组合形式，如信息收集和数据流控制。Pyretic 拓展了 Frenetic 提供了基于模块化的 SDN 编程语言。Nettle 则是基于函数式相应式编程（FRP）对网络进行控制。Maple [?] 提出了基于算法式的 SDN 编程语言，并提供对数据包的读、测试等基本 API。通过记录数据包执行的踪迹，构建踪迹树，然后基于踪迹树自动生成底层流表。McClurg 等人 [?] 提出网络事件结构的模型，保证了在事件到达时对网络状态的更新过程中也可以满足的指定的不变性质。

SNAP [?] 考虑将高级 SDN 程序转化为多个有状态交换机的配置。但是其编程模型为 One-Big-Switch 模型，即用户无法指定数据包在网络传输的具体路径。

1.3.2 面向特定网络编程模型的研究背景

我们这里对特定网络分为三种情况：1. 交换机和中间盒（或 NFV）的混合网络；2. 实现特定功能的网络（如 Paxos）；3. 针对特定场景的网络（如车联网，物联网等）。

混合网络：Qazi 等人 [?] 通过 SDN 保证数据流可以正确地传过事先规定好的中间盒序列。Fayazbakhsh 等人 [?] 解决了当中间盒对数据包头进行修改后，数据包仍然可以按照指定路径进行转发。Trident [?] 考虑利用网络中间盒可以处理数据包更高层信息的功能，设计统一的高级编程模型，使网络数据包的传输可以依赖更高层信息（如数据包对应流的 http 信息）。为了实现类似的功能（即数据包根据高层信息进行转发），Mekky 等人 [?] 采取拓展 Open vSwitch 的方法，使交换机存储状态信息。

实现特定功能的网络：通过利用可编程交换机，一些传统的分布式算法可以高效地在网络中实现。NetPaxos [?, ?] 利用 P4 语言对可编程交换机进行配置，使其分别扮演 Paxos [?] 的 acceptor，coordinator 等角色，通过订制包头的数据包在交换机之间的传输，实现分布式一致性算法。基于类似的想法，Netcache [?] 和 Netchain [?] 则利用可编程交换机流表的性质，将键值对的存储在网络中，实现分布式键值对存储。Typhoon [?] 利用底层 DPDK 数据平面，实现基于 SDN 的实时大数据流处理框架。Beckett 等人 [?] 针对 BGP 场景，将 BGP 路由的约束转化为底层交换机的配置。

特定场景网络：Zhu 等人 [?] 考虑对利用 SDN 技术实现车联网中紧急消息的快速分发功能。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。Mobile fog [?] 提出面向物联网的编程模型。Mobile fog 中的一个应用由多个进程组成，每个进程映射到网络中的一个节点，如核心节点，fog 节点，边缘节点。进而这些进程根据节点之间的层级关系，也组成一个逻辑上的层级关系（如父子节点关系）。通过层级关系，进程之间可以发送数据，共同完成任务。

1.4 论文研究挑战

根据上节对研究背景的叙述，我们把本论文研究的挑战主要分为两部分: 面向单交换机编程模型的研究和面向全网络编程模型的研究。而每一部分都有两个基本问题：SDN 程序数据平面实现问题和针对特定场景情况的优化问题。其中实现问题是指：给定一个 SDN 程序，一个数据平面是否可

以正确表达该 SDN 程序。正确是指：对于任意数据包，SDN 程序对数据包返回的结果和数据平面对数据包返回的结果一致。下面我们针对上述的两部分来具体说明论文研究挑战。

第一，在面向单交换机编程模型的研究中，已有的相关工作可以分为如下：1. 通过低级配置接口生成具有固定结构（如单流表或 OF-DPA）数据通路的配置；2. 将高级程序转化为具有可定制结构（如 RMT）数据通路的配置。但是缺少将高级程序转化为具有固定结构数据通路的配置的相关工作。而在转化之前，需要考虑的是该固定结构数据通路是否可以实现高级程序。因此，如何判断一个高级 SDN 程序是否可以在具有固定结构数据通路上实现，是一个挑战。除此之外，当 SDN 程序具有循环结构时，如何将该 SDN 程序实现在可定制结构数据通路上并没有相关工作。因此高级 SDN 程序中循环结构在可定制结构数据平面的高效实现，是一个挑战。

第二，在面向全网络编程模型的研究中，对于一般网络，大量工作是以响应式模式（即不会主动生成配置）对网络进行管理，因此效率并不高。SNAP 实现了主动编译生成数据平面配置，但其程序并不灵活。因此，给定一个高级灵活的 SDN 程序，如何生成面向全网络的数据平面配置是一个挑战。除此之外，针对不同场景，如软件定义联合 [**?**] 网络的低时延要求，以及车联网中车的移动性要求，如何优化数据平面配置是一个挑战。

1.5 论文研究内容

针对上一节中探讨的两方面的挑战，本文进行有针对性的研究，主要分为以下五个部分：

第一，针对判断高级 SDN 程序是否可以在具有固定结构数据通路实现问题，提出了将高级 SDN 程序和底层数据通路统一的特征空间。并将实现问题转化为空间中的比较问题，提出数据通路编程容量理论来系统地解决问题。

第二，针对高级 SDN 程序中循环结构在可定制结构数据通路的高效实现问题，提出了重复软件流水线转换。通过转换，在计算循环结构的最佳数据通路结构时显示出更高的效率。

第三，针对高级灵活的 SDN 程序面向全网络数据平面实现问题，当网络节点全部为可编程交换机时，提出将程序拆分并部署到不同交换机的方案，并算出给定目标下的最优部署；当网络节点存在固定功能节点时（如防

火墙），提出程序正确性的定义，并通过系统路径约束的概念保证程序的正确。

第四，针对软件定义联合网络的具体场景，设计高级编程系统，并基于共享本地状态的方法，优化系统性能：降低数据传输时延，提高传输带宽。

第五，针对车联网的具体场景，提出软件定义车联网的架构以及编程框架。考虑车联网中车移动性特点，提出优化的规则下发方法，减少生成的规则数量。

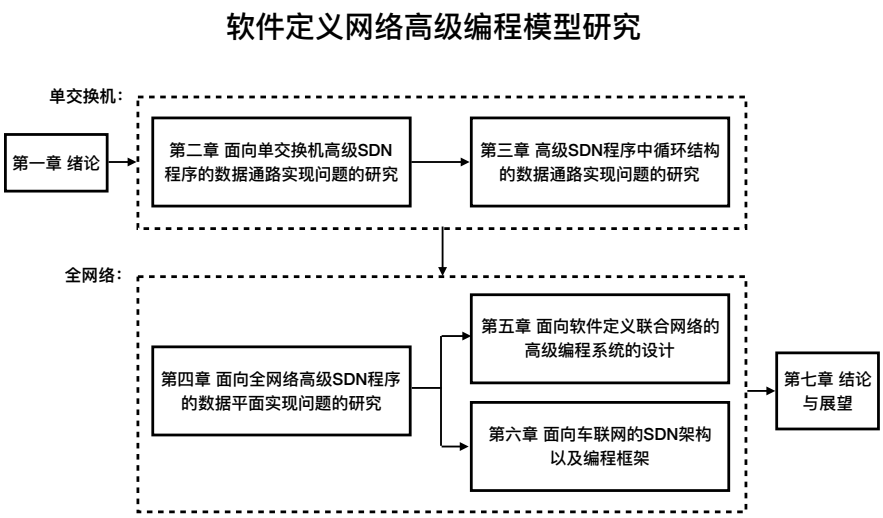

图 1.1: 论文框架

1.6 论文结构

如图 1.1所示，本论文总共由七部分组成：第一章为绪论，先介绍了目前软件定义网络高级编程模型研究的背景。并根据背景给出论文挑战，最后列出论文内容。

第二章，第三章均为面向单交换机情况。其中，第二章研究面向单交换机的高级 SDN 程序数据通路实现问题，并给出特征空间以及数据通路编程容量理论。待第二章的基本实现问题讨论完后，第三章考虑高级 SDN 程序中循环结构，并给出循环结构在可定制结构数据通路的高效实现方法。

第四章，第五章，第六章均为面向全网络情况。其中，第四章研究面向全网络的高级 SDN 程序数据通路实现问题。待第四章的基本实现问题讨论

完后，第五章考虑针对软件定义联合网络的数据平面优化，第六章考虑软件定义车联网架构以及编程框架。

最后，第七章总结了研究内容并对未来工作进行展望。在正文之后，附上参考文献，博士期间个人论文发表情况。