doi:10.11959/j.issn.1000-436x.2015178

基于元能力的 SDN 功能组合机制

段通，兰巨龙，程国振，胡宇翔
（国家数字交换系统工程技术研究中心，河南 郑州 450002）

摘 要：软件定义网络（SDN, software-defined network）促进了控制逻辑的快速创新，使控制逻辑模块化和模块组合机制成为 SDN 的热点研究方向之一。为了在功能模块统一定义和规范划分的基础上实现网络功能组合，首先，从可重构网络引入“元能力”作为 SDN 控制功能模块划分的原子要素，提出一种基于元能力建模的统一资源描述方法；其次，针对网络功能灵活组合问题，提出了一种基于二级映射的元能力组合模型并给出其启发式算法；最后，为实现元能力组合，设计了作为 SDN 应用层扩展结构的元能力编排层，并给出基于 NetFPGA-10G 平台的原型实现。仿真实验与结果表明所提功能组合机制提高了组合效率及节点资源利用率。
关键词：软件定义网络；元能力；功能组合；资源利用率
中图分类号：TP393 **文献标识码**：A

Functional composition in software-defined network based on atomic capacity

DUAN Tong, LAN Ju-long, CHENG Guo-zhen, HU Yu-xiang
(National Digital Switching System Engineering & Technological Research Center, Zhengzhou 450002, China)

Abstract: Software-defined network (SDN) enables frequent network functional innovation and evolution, making the control function modularity one of the most hot research fields on SDN. Aiming at network-wide functional composition based on unified module definition and function division, firstly, atomic capacity (AC) is introduced as atomic elements of module function division and an AC-based resource description method is proposed; secondly, functional composition module based on two-level mapping is put forward and a heuristic algorithm to calculate the composition is proposed; finally, an AC-orchestrating layer as extended structure of SDN application layer is devised and then a NetFPGA-10G prototype implementation is given. Experimental results show that the method can combine functional instances more effectively with better resource utilization.
Key words: software-defined network; atomic capacity; functional composition; resource utilization

1 引言

软件定义网络[1]将控制功能迁移到控制器并向上层提供应用程序编程接口（API, application programming interface），支持在控制器之上开发应用以实现对网络管控，使网络控制功能软件化、可编程化，提高了网络的管控能力和创新能力。SDN 应用可以取代传统网络中的中间件[2]，像 NAT、防火墙、负载均衡[3,4]和接入控制[5]等功能均在 SDN 中得以实现。然而，为应对网络的业务需求，现有 SDN 应用往往需要实现多个功能（比如路由、监控、接入控制和负载均衡等），同时又必须保证各个功能生成的流表项之间不会相互覆盖。这导致应用复杂难以开发[6]，且应用之间存在大量的功能重叠。对

收稿日期：2014-09-26；**修回日期**：2015-03-10
基金项目：国家重点基础研究发展计划(“973”计划)基金资助项目(2012CB315901, 2013CB329104)；国家自然科学基金资助项目(61372121)；国家高技术研究发展计划(“863”计划)基金资助项目(2013AA013505)
Foundation Items: The National Basic Research Program of China (973 Program) (2012CB315901, 2013CB329104); The National Natural Science Foundation of China(61372121); The National High Technology Research and Development Program of China (863 Program) (2013AA013505)

此，C Monsanto 等[6,7]指出模块化是减小 SDN 软件系统复杂度的关键并提出将 SDN 应用转化为多个可独立处理数据流的功能模块的组合，使控制逻辑的模块化及其组合机制成为 SDN 的热点研究方向之一。

在控制功能模块化方面，Beacon[8]控制器采用 Java 语言的 Spring 和 Equinox 编程模型将控制器分成一个个 Bundle 的组合，便于模块的添加和删除，为功能模块化提供控制层面支持。J.C.Mogul 等[9]提出名为 Corybantic 的控制平面设计，它支持不同控制器内模块化程序的组合以及上层对控制器的规划。网络功能虚拟化[10]（NFV, network functions virtualization）将软件与硬件分离，在通用化的硬件之上部署软件以完成网络功能。NFV 可为 SDN 网络功能模块化提供应用层面支持。此外，可重构网络[11,12]对网络功能模块进行具体定义和细化，提出将元能力作为网络节点内资源划分的原子要素，并通过资源和节点能力的灵活组合实现网络能力对业务的普适。

为实现 SDN 中的功能组合，文献[7]中提出并行的模块化编程语言 Frenetic，将并行的功能模块生成的流表项组合起来完成对数据分组的多重处理。随后又针对 Frenetic 的不足提出 Pyretic[6]编程语言，Pyretic 支持串行功能模块的组合，应用通过下发组合策略的方式将功能模块生成的表项组合起来，实现功能模块对数据流的串行处理。以上研究是将功能组合在单个节点内实现，其方法具有 2 方面的局限性：首先，超过 2 个功能的表项合并复杂度较高，目前尚缺乏相关研究；其次，在单个交换机内组合表项的方式面临交换机表项容量的限制，实现起来较为困难。对此，可以通过多交换节点协同的方式将不同交换节点内的功能进行组合，其研究重点在于如何驱动数据流遍历功能节点。文献[13,14]是这种研究思路的代表，其中，文献[13]提出网络功能模块开放机制，该机制允许第三方公司开发控制功能模块并向网络添加部署。网络管理员根据业务需求下发策略，控制平面根据策略计算出路径后驱动数据流遍历功能部署节点，但文中未提出功能模块划分方法及具体组合方法。文献[14~16]针对 SDN 中间件的功能组合问题提出通过下发转发表项的方式遍历所需功能节点，但其并未涉及控制器内功能模块的组合。

综上，如何在对控制功能模块进行统一定义和规范划分的基础上实现多交换节点的功能组合，以支持用户的多样化网络服务定制是一个亟待解决的问题。针对以上问题，本文从可重构网络引入元能力[11,12]（AC, atomic capacity）作为 SDN 控制功能模块划分的原子要素，并提出一种基于元能力建模的统一资源描述方法，为后续功能组合方案提供支撑。其次，提出元能力组合问题，并针对该问题提出了一种基于二级映射的元能力组合模型（ACCM, AC composition model）。该模型将元能力组合过程描述为“策略-实例链-执行路径”两级映射过程，通过全局网络视图并根据应用策略选取适当的元能力实例将其连接起来提供端到端的服务。接着提出节点优先算法求解元能力组合模型。最后，为完善 SDN 应用层以满足功能模块组合的需求，设计了作为应用层扩展结构的元能力编排层，并给出基于 NetFPGA-10G[17]平台的原型实现。

2　基于元能力建模的统一资源描述

控制功能的模块化驱使开发者致力于 SDN 应用模块的开发中，但缺乏统一的功能划分导致模块粒度不统一和模块功能不明确。针对该问题，为向 SDN 控制功能模块的开发提供统一的功能划分和描述，本文借鉴可重构网络[11,12]的思想，将基础网络控制功能分解为细粒度的功能单元——元能力。元能力作为 SDN 控制功能模块功能划分的原子要素，为开发者提供统一的功能描述，从而避免功能划分不统一导致的模块功能混乱并为网络功能的组合编排提供基础。

首先，借鉴可重构网络的元能力划分方式，将网络控制功能分解成 4 大类，分别为安全类、转发类、控制类、管理类；再对每类功能进行功能单元分解，得到细粒度的元能力集合，记为 C，包括监控、多播、路由、接入控制、排队、调度、负载均衡等，如图 1 所示。该集合是封闭的，且其中的元素是有限的。不仅可以对集合中元能力进行动态组合以实现服务的用户定制化，也可以向集合中添加新型的元能力以实现网络功能的快速扩展，由此增强网络管控能力和扩展能力。

元能力的功能描述由 ONF、IETF 等标准化组织定义并发布，开发者可针对不同元能力开发并部署自己的元能力实例。标准化组织向外发布元能力时将元能力描述为<Type: Description>，其中，Type 是元能力类型，Description 给开发者提供相应的标

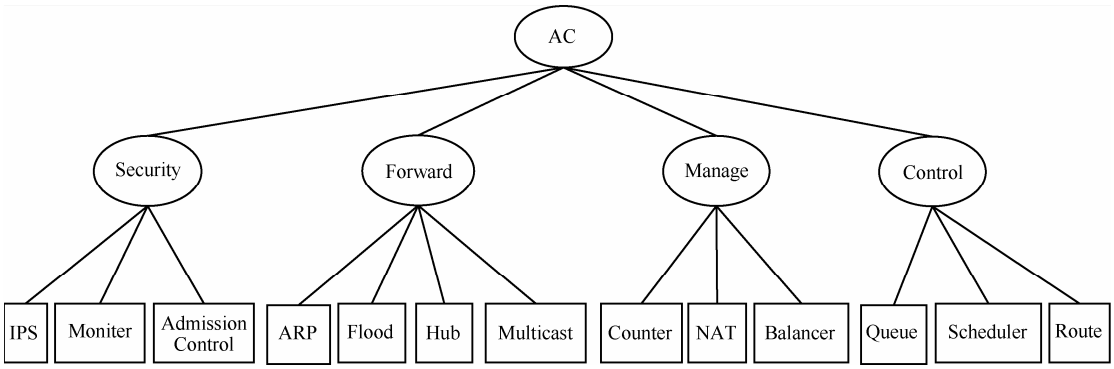


图 1 元能力集合

准功能描述。如类型为监控（Monitor）的元能力描述可用 XML 方式向外界发布，如图 2 所示。

```
<types>
  <type class= “monitor”>
    <description>
      monitor the special traffic pattern
      and prevent the match
    </description>
  </type>
  …
</types>
```

图 2 元能力功能发布示例

2.1 元能力实例

元能力实例是指由开发者开发并已经部署在网络中的元能力。元能力实例运行于控制器之上，通过生成流表项的形式部署在交换节点内完成对数据流的功能处理。本文将元能力实例描述为五元组结构<类型，标识，属性集，动作，提供者>。类型代表元能力的类别，所有具有相同类型的实例也具有相同的功能。标识是一个数字，用类型和标识全网唯一地标识一个元能力实例。属性集包括处理目标和性能参数，其中处理目标是指该元能力实例处理的对象，性能参数是指该元能力实例的参数化的性能指标，如存储代价、可靠性等。动作代表该元能力实例对处理对象执行的操作。提供者代表元能力实例的提供商。

2.2 组合链

元能力的组合机制是网络应用实现适应性的核心。元能力的组合通过定义组合链机制实现，元能力之间通过组合链关联，这种关联使组合链两端无冲突地作用于数据流，最终产生叠加处理的效果。将多个元能力的组合称作组合链，其符号化定义如下。

定义 1 组合链。C 是元能力集合，且 $C\neq\varnothing$。序列集合 C 是一个组合链，当且仅当

1) $C=\{c_1,c_2,\cdots,c_{i-1},c_i,c_{i+1},\cdots,c_n\}$，$\forall i\in Z^+, c_i\in C$；

2) 子序列 $\{c_1,c_2,\cdots,c_{i-1}\}$ 是 c_i 的前驱元能力集合，$\{c_{i+1},\cdots,c_n\}$ 是 c_i 的后继元能力集合，c_{i-1} 是 c_i 的直接前驱元能力，c_{i+1} 是 c_i 的直接后继元能力；

3) c_1 是 C 的初始节点，没有前驱元能力；c_n 是 C 的结束节点，没有后继元能力。

2.3 策略描述

针对不同的应用场景和业务需求，上层应用通过下发策略的方式管控网络。策略包含目标集和元能力集。目标集是指所要执行处理的目标集合，比如在流量工程中模式为 {srcIP=192.168.1.1/16; dstIP=192.168.2.2/16}的数据流。元能力集是用类型标识的有序元能力集合，即组合链，且对目标集中目标流的操作必须依次遍历元能力集中的全部元能力。

定义 2 策略。策略 $P=\{<O^P, C^P>|O^P\subseteq O, C^P\subseteq C\}$，$O^P\neq\varnothing$，$C^P\neq\varnothing$。其中，$O, C$ 分别代表目标集，元能力集。C^P 是一个有序的集合，它包含的元能力是有序排列的。

通过一个示例进一步说明。如图 3 所示，控制器 A 负责管控整个区域，交换机 E7 连接存储资源（比如散列表 HT），IP 地址为 10.1.2.0/8。一些元能力实例（诸如监控、均衡、NAT、整形、计数等）在相应交换节点内下发了表项。上层应用希望在用户访问 IP 为 10.1.2.0/8 的存储服务器之前，要经过计数、NAT、监控。该策略可表达为<{src= 192.168.1.2/8, dst=10.1.2.0/8}, {Counter, NAT, Monitor}>。

对于策略中的组合链，需要为每一个元能力选取一个元能力实例来执行相应的功能，记实例化后的组合链为实例链。实例链的符号定义如下，其中符号“$\mapsto$”表示实例化，如 $s\mapsto e$ 表示 e 是元能力 s 的一个实例。

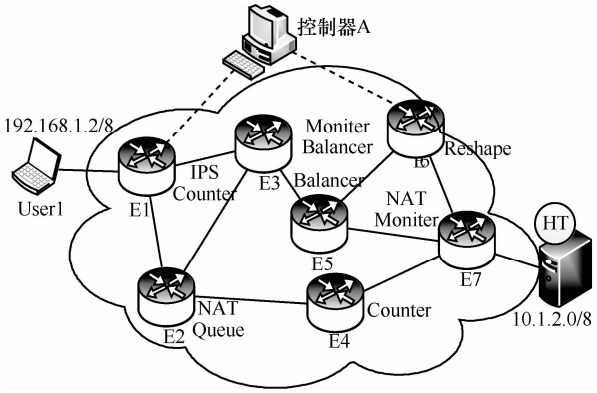


图 3　元能力应用场景示例

定义 3　实例链。C 是元能力集合，且 $C\neq\varnothing$。$E=\{e|c\mapsto e,\ c\in C\}$是所有元能力的实例集。对于 $\forall c_i\in C$，$E_i=\{e|c_i\mapsto e,\ e\subseteq E\}$是元能力 c_i 的一个实例集。假设序列 $C=\{c_1,c_2,..,c_{i-1},c_i,c_{i+1},\cdots,c_n\}$是一条组合链，则序列 $E_C=\{e_1,e_2,..,e_{i-1},e_i,e_{i+1},\cdots,e_n|\forall e_i\in E_i\}$是 C 的一条实例链。

将实例链中各个实例表项所在的节点进行连接，形成源节点到目的节点的处理路径，称作执行路径。执行路径的符号描述如下。符号“$\Rightarrow$”表示实例表项所在节点，比如 $e\Rightarrow p$ 表示 p 是实例 e 的流表项所在的节点。

定义 4　执行路径。E 是实例集合，$E\neq\varnothing$。$S=\{s|e\Rightarrow s,e\in E\}$是元能力实例表项所在的节点集。对于 $\forall e_i\in E$，$E_i=\{e|c_i\mapsto e,e\subseteq E\}$是元能力 c_i 的一个实例集。假设序列 $E_C=\{e_1,\cdots,e_i,\cdots,e_n\}$是 C 的一条实例链，则序列 $Sc=\{s_1,s_2,\cdots,s_j,\cdots,s_m\}$是 C 的一条执行路径，当且仅当

1) 节点 s_j 和 s_{j+1} 相互连通，$1\leqslant j\leqslant m-1$；

2) $m\geqslant n$；

3) 如果 $e_i\Rightarrow s_j$，则 $e_{i+1}\Rightarrow s_k$，其中 $1\leqslant i\leqslant n$，$1\leqslant j\leqslant k\leqslant m$；

4) s_1 是路径的初始节点，s_m 是路径的目的节点。

3　元能力组合问题建模

3.1　问题描述

在基于元能力的统一资源描述的基础上实现元能力的组合是实现业务的服务定制化的关键。对此，提出 SDN 中的基于全局网络视图和元能力实例节点分布视图的元能力组合问题。

定义 5　元能力组合问题。给定策略、网络拓扑、元能力实例表项分布视图，在满足节点连通的前提下，从网络节点中选取能执行组合链的节点并生成一条从源节点到目的节点的执行路径。

元能力组合过程可视为“策略–实例链–执行路径”二级映射，即“策略–实例链”（P-IC, policy-instance chain）映射和“实例链–执行路径”(IC-EP, instance chain-execution path)映射，其定义如下。

定义 6　“策略–实例链”映射是指以策略的元能力集为组合链，以需求集为约束，选取每个元能力合适的元能力实例，组成实例链。映射过程记为 $f: P\rightarrow E_C$，其中，$P=\{<O^P,C^P>|O^P\subseteq O,C^P\subseteq C\}$，$E_C\subseteq E$。

定义 7　“实例链-执行路径”映射。针对网络拓扑，将实例链中各个实例表项所在的节点进行连接，形成一条底层的端到端的数据流通路。映射过程记为 $g: E_C\rightarrow S_C$，其中 $S_C\subseteq S$。

图 4 所示为一个“策略-实例链-执行路径”映射过程示例。策略的目标集是{src=s_1, dst=s_8}，首先将策略 P 映射为 5 个元能力实例组合的实例链，然后将实例链映射到一条从节点 s_1 到节点 s_8 的执行路径，该执行路径中带阴影的节点表示是实例表项所在节点，不带阴影的节点仅起到连接作用。

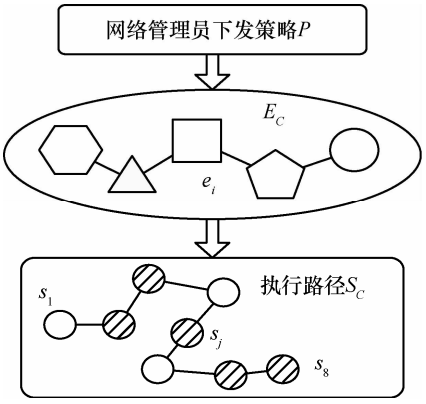


图 4　“策略–实例链–执行路径”映射过程

3.2　“P-IC”映射

用图 $G(V,L)$来表示一个 SDN 控制器管控的交换机网络，其中 V 表示交换机节点的集合，L 是网络链路的集合。$G(V,L)$中链路双向连通。假设应用安装了一个策略 P，“P-IC”映射到实例链。为形式化地描述“P-IC”映射问题，本文使用的符号如表 1 所示。

本文给出“策略–实例链”映射问题的形式化描述，即元能力组合模型（ACCM），通过选取合适的元能力实例，最大化实例链的效用。模型如式(1)~式(4)所示。

表 1 主要的符号定义及其意义

符号	含义
P	$P=\{<O^P,C^P>\mid O^P\subseteq O,C^P\subseteq C\}$ 是一个策略
O^P	策略的目标集
C^P	有 n 个元能力的组合链 $C^P=\{c_1^P,c_2^P,\cdots c_i^P\cdots c_n^P\}$，$0<i\leqslant n$，$i,n\in Z^+$
C	网络 G 中所有元能力的集合
E_i^P	元能力 c_i^P 的实例集合
E_C^P	策略 P 映射到的实例链
$J(E_i^P)$	E_i^P 所有的元素的下标
$I(C^P)$	C^P 的所有元素的下标
e_{ij}	E_i^P 中的一个实例，$i\in I(C^P)$，$j\in J(E_i^P)$
y_{ij}	表示是否选择实例 e_{ij} 作为元能力 c_i^P 的实例。1 代表选择，0 代表不选择
u_{ij}	e_{ij} 的效用
$\boldsymbol{V}_{ij}$	表示实例 e_{ij} 的 t 维性能参数向量 $\boldsymbol{V}_{ij}=\{v_{ij}^1,v_{ij}^2,\cdots,v_{ij}^t\}$
$\lvert J\rvert$	表示 J 的元素数量
S_{ij}	表示实例 e_{ij} 的表项所在的节点集
S_C^P	$S_C^P=\{s_1^P,s_2^P,\cdots,s_i^P,\cdots,s_m^P\}$ 表示策略 P 最终映射到的执行路径

优化目标

$$\max\ f(y)=\sum_{i\in I(C^P)}\sum_{j\in J(E_i^P)}u_{ij}y_{ij} \tag{1}$$

约束条件

$$a_{ij}^P\leqslant\langle E_{ij}^P\rangle,i\in I(C^P),j\in J(E_i^P) \tag{2}$$

$$\sum_{j\in J(E_i^P)}y_{ij}=1 \tag{3}$$

$$y_{ij}\in\{0,1\},i\in I(C^P),j\in J(E_i^P) \tag{4}$$

式(1)是该模型的约束目标，即最大化实例链的效用。此外，该模型还具有一系列的约束条件。式(2)中 a_{ij}^P 指实例 e_{ij} 的使用次数，$<E_{ij}^P>$指实例 e_{ij} 所能满足的请求次数，该式表明实例的使用次数有限制，因为底层节点和上层软件的资源容量有限。式(3)表明针对组合链中的每个元能力只能选择其中一个实例。式(4)表明该模型的解变量只能取值 0 或 1，也即该问题是一个 0-1 线性规划问题。

为了对元能力组合过程进行评价，本文在此给出式(1)中元能力实例的效用函数 u_{ij} 定义。考虑元能力实例的性能参数 $\boldsymbol{V}_{ij}$ 是 t 维的向量，包括节点链路带宽、流表空间耗费、QoS 指标等参数。对于带宽和流表空间等资源的耗费，不仅要考虑绝对耗费量，还要考虑其空闲比例等因素。令 v_{ij}^1 代表实例表项所在节点的链路带宽耗费，v_{ij}^2 代表节点的流表空间耗费，将其转化为[0,1]之间的比例参数

$$r_{ij}^1=\frac{v_{ij}^1}{B(i,j)},\ r_{ij}^2=\frac{v_{ij}^2}{S(i,j)} \tag{5}$$

其中，$B(i,j)$代表实例表项所在节点的总带宽，$S(i,j)$代表实例表项所在节点的流表空间容量。对于其他 QoS 指标参数，需要归一化以使各个参数的度量统一，将 v_{ij}^k 转化为[0,1]之间的参数 r_{ij}^k

$$v_m(i,k)=\max_{j\in J(E_i^P)}(v_{ij}^k) \tag{6}$$

$$r_{ij}^k=\frac{v_{ij}^k}{v_m(i,k)},\ 3\leqslant k\leqslant t \tag{7}$$

该 t 维向量中的前 m 个参数是负相关的，即取值越小越好（如流表空间耗费等）；而后 $t-m$ 个参数则是正相关的，即取值越大越好（如完备性等）。本文定义效用函数，将所有的性能参数转化为统一的效用值，使之与优化方向正相关。

$$u_{ij}=\sum_{k=1}^{m}w_k\pi^-\left(r_{ij}^k\right)+\sum_{k=m+1}^{t}w_k\pi^+\left(r_{ij}^k\right),\ 1\leqslant k\leqslant t \tag{8}$$

其中，系数 w_k 是放缩系数，用来调整各个性能参数的影响因子，且 $\sum_{1\leqslant k\leqslant t}w_k=1$，$w_k\geqslant 0$。

对于前 m 个参数，函数 $\pi^-:r\to[0,1]$是关于 r 的减函数；对于后 $t-m$ 个参数，函数 $\pi^+:r\to[0,1]$是关于 r 的增函数，从而使效用函数 u_{ij} 与优化目标正相关。为不失一般性，取 $\pi^+(r)=r$，$\pi^-(r)=-r$。

3.3 “IC-EP”映射

“IC-EP”映射仅起到连接作用，即根据网络拓扑将所选实例表项的所在节点进行连接，形成端到端的执行路径。此问题需要考虑网络拓扑以及实例的节点分布，可用输入–输出进行建模。

输入：

E_c^P：实例链，$E_c^P=\{e_{ij}\mid 0<i\leqslant n,j\in J(E_i^P)\}$

S_{ij}：实例在网络中的分布视图，$0<i\leqslant n,j\in J(E_i^P)$

$G(V,L)$：网络拓扑

输出：

S_C^P：执行路径，$S_C^P=\{s_1^P,s_2^P,\cdots,s_i^P,\cdots,s_m^P\mid s_1^P=v_1,s_m^P=v_2\}$

可以注意到，“IC-EP”映射与“P-IC”映射并不是互相独立的过程，“P-IC”映射中实例的选取依

赖于实例在各个节点上的部署视图以及网络拓扑，而“IC-EP”映射则依赖于实例链的选取。故可将“IC-EP”映射模型的输入 S_{ij} 和 $G(V,L)$作为“P-IC”映射模型的约束输入，即把“IC-EP”映射与“P-IC”映射统一起来，在求出“P-IC”映射模型解的同时，也得到对应的执行路径。

4　算法描述

元能力组合问题是 NP 问题。传统求解 NP 问题多采用贪婪算法（greedy algorithm），贪婪算法首先根据网络拓扑计算从源节点到目的节点的多路径，然后通过穷举多路径中所有满足策略需求及节点连通性的实例链，找出效用最大的实例链，继而得到其所在的路径。贪婪算法可在全网范围内搜寻实例链并选择效用最高的实例链，但其复杂度较高，在网络规模扩大或网络实例部署增多时其解空间急剧增大，由于计算机 CPU 和内存能力有限，使得计算机很难对此进行求解。

4.1　节点优先算法

为简化搜索空间，本文提出一种节点优先算法，在保证实例选取满足网络拓扑约束的前提下，最大化实例链的效用。算法执行过程中，首先寻找组合链中第一个元能力的所有实例并选取效用最大实例，记录其节点所在路径；而后在该路径中依次寻找效用最大实例，最终得到实例链。若该路径无法满足实例链存在性，则返回依次选取剩余路径。

算法 1　节点优先算法（NodeFirst）

输入：网络拓扑图 $G(V,L)$；实例 e_{ij} 所分布的节点集 S_{ij}；网络策略 P，其目标集为{src= v_1，dst= v_2}；

输出：实例选取系数 y_{ij} 及相应的执行路径 S_C^P

1) $Path=\{path_1,path_2,\cdots,path_h\}$　/*计算从节点 v_1 到节点 v_2 的多路径集合*/

2) for $i\in I\left(C^P\right)$　/* $i=1,\cdots,n$ */

3) src $=v_1, IC\_set=[]$; /* IC_set 记录实例链*/

4) for $j\in J(E_i^P|\text{src},path)$　/* $E_i^P|$src，$path$ 表示 E_i^P 中节点 src 在多路径中的所有后向节点中的实例集合*/

5) if max u_{ij} then：　/*选择效用最大实例*/

6) if max $u_{ij}=0$ then：/*如果不存在则重新选取*/

7) $E_k^P=E_k^P\setminus IC\_set(k)$, $k=k+1$

8) goto step 1

9) end if

10) $y_{ij}=0$

11) $IC\_set=[IC\_set,e_{ij}]$　/*更新实例链集合*/

12) src $=\{s\,|\,e_{ij}\Rightarrow s,s\in path\}$ /*更新搜索初始节点*/

13) end if

14) end for

15) end for

4.2　算法复杂度分析

节点优先算法在计算复杂度和实例链性能之间平衡，其搜索空间相对较小。假设拓扑中有 m 个节点，策略中组合链长度为 n，在 NodeFirst 算法的初始化阶段，先利用 Dijkstra 算法求出两节点之间多路径（路径个数为 k），时间复杂度为 $O(m^2)$；而针对一个路径选取最大效用实例的过程最多对 m 个节点搜寻 n 次，时间复杂度为 $O(mn)$；若搜寻不成功则返回重新选取，最多迭代 k 次。因此，该算法的时间复杂度为 $O(m^2+mnk)$。而贪婪算法在搜寻过程中最多对 m 个节点各搜寻 n 次，总的时间复杂度则为 $O(m^2+m^n)$。当 n 较大即组合链较长时，贪婪算法的时间复杂度比 NodeFirst 算法高很多。

此外，效用函数 u_{ij} 的计算涉及到流表空间状态等信息的收集。本文在实例表项下发时对下发节点的流表空间耗费进行记录并更新，该过程在计算执行路径之前已经完成，因此其带来的额外处理开销不会影响算法的计算复杂度。

5　基于 OpenFlow 的元能力组合实现

5.1　方案设计

为支持基于元能力的功能组合，提出 SDN 应用层的扩展结构，即元能力编排层。如图 5 所示，元能力编排层位于控制器之上，它和元能力之间有统一的 API 接口。开发者提供自己的元能力实例向元能力编排层注册，并以生成流表项的形式部署到底层交换节点，编排层根据应用下发的策略将元能力实例进行组合以提供定制化服务。组合后的实例链以执行路径的形式通过控制器下发到底层交换机。

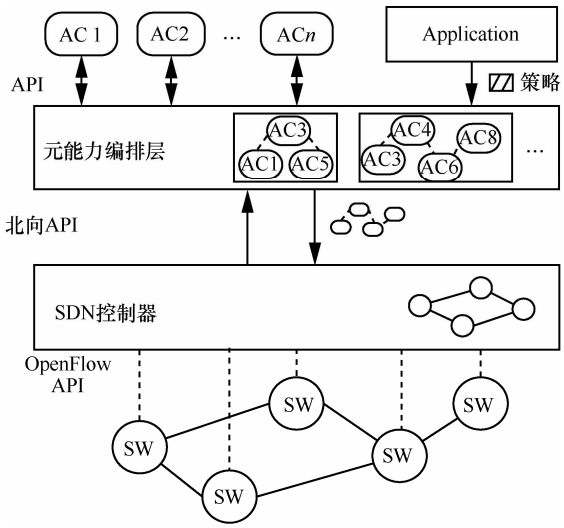


图 5 元能力编排层

当前 SDN 架构下，当数据分组到达交换机，首先和交换机内的流表项进行匹配，若匹配成功则执行相应动作；否则它将被上传至控制器，由控制器计算路由并安装到交换机中，数据流后续的数据分组会直接根据流表项被处理。

元能力编排层的引入改变了策略影响数据流的方式。首先元能力编排层会根据应用下发的策略计算得到执行路径并存储起来。数据分组到达交换机后首先与流表项进行匹配，若匹配成功则执行相应操作，否则被上传至控制器。上传至控制器的数据分组首先被元能力编排层处理：数据分组与策略的目标集匹配，若匹配成功则表明该数据流是策略关注的流。根据数据分组（src, dst）得到相应的执行路径并通过控制器安装到交换机拓扑中，该流后续的数据分组会直接被这条执行路径处理。若数据分组未匹配到策略目标集，则被 SDN 控制器处理。

5.2 基于 NetFPGA-10G 的原型系统实现

目前，本文已经在 NetFPGA-10G[17]平台上基于 Openflow1.3 协议实现了 SDN 中底层交换节点内和节点间的功能组合。原型系统支持 4 级流表串联的流水线处理，每级流表的动作模块可完成协议规定的动作，如转发、丢弃、修改分组头字段、添加 VLAN 标签等，用以实现如路由、转发、安全、接入控制、负载均衡、NAT 等元能力及其组合。原型系统实现框架如图 6 所示，NetFPGA 板卡负责数据平面转发，主机软件负责对底层硬件流表的配置管理以及与控制器的交互。底层数据通路包含 4 个硬件流表和动作处理模块，从节点网络端口进入的数据分组首先与流表进行匹配，若匹配成功则执行相应操作（支持 4 个动作的串行处理），若无匹配项则通过 DMA 上传至交换机软件；上层软件包含配置工具以及与控制器相连的安全通道模块（secure channel），支持手动配置流表以及通过上层控制器实现对交换机的配置。

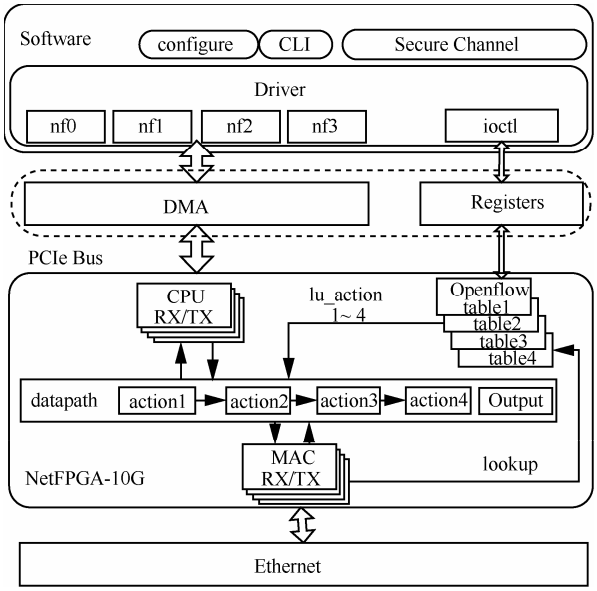


图 6 基于 NetFPGA-10G 的元能力组合实现

6 元能力组合性能仿真与分析

本节从 3 方面对元能力组合机制进行验证。首先，通过计算软件对所提元能力组合算法性能进行仿真，并与贪婪算法进行对比评价；其次，通过硬件仿真实验在 NetFPGA-10G 平台上对元能力组合机制的有效性和优势进行验证；最后通过与其他编程语言的功能组合机制做对比实验，验证元能力组合机制在组合效率上的优势。

6.1 算法性能对比

考虑如图 7 所示的网络，网络中有 6 个交换机节点，为不失一般性，假设所有的节点的硬件和软件性能相同。仿真平台采用 MATLAB 搭建，计算机处理器为 Intel Core i7-3770、4 GB RAM。

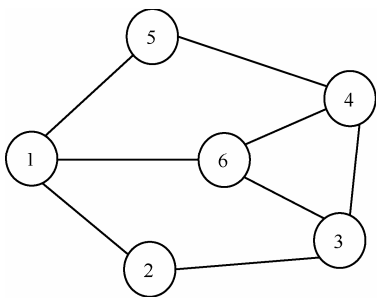


图 7 仿真网络拓扑

现假设网络中有 100 种元能力，记为 ac_1~ac_{100}。每种元能力在网络中又有 1~10 个不同开发者提供的元能力实例，实例的性能指标参数个数 t=4，且各个实例的各个性能参数分别满足高斯分布。假设这些元能力实例表项随机且均匀地下发到了各个节点上，每个节点部署有 q 个不同类型的元能力实例表项，$q\in$ [1,100]。

为了分析算法的性能，本节考虑 2 个性能指标：负载均衡度以及时间复杂度。记节点内所有实例表项的使用次数为节点负载，记从接收请求到安装执行路径的时间间隔为时间复杂度。负载均衡度体现算法在资源利用率方面的优劣，时间复杂度体现算法在计算性能方面的优劣。

1) 负载均衡度

设每个节点下发 q=50 个实例表项，随机生成 10 000 次请求，分别统计最终得到的执行路径中各个节点的使用次数。重复仿真 10 次，得到 Greedy 和 NodeFirst 算法的各个节点的平均负载，如图 8 所示。

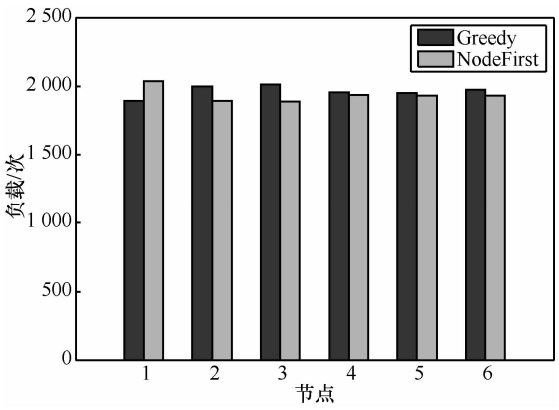


图 8　算法负载均衡度对比

从图 8 可以看出，在相同网络条件下，Greedy 和 NodeFirst 算法均可近似达到各个节点的负载均衡，说明 Greedy 和 NodeFirst 算法均可较好地平衡各个节点中的实例，消除实例分布不均匀导致的节点负载不均匀。

2) 时间复杂度

根据第 4 节的算法复杂度分析，在多路径给定的情况下，Greedy 算法的时间复杂度为 $O(m^n)$ 而 NodeFirst 算法的时间复杂度为 $O(mnk)$，即 2 种算法的时间复杂度成线性关系。实验中生成 0~7 000 次请求，分别统计 2 种算法的处理时延，如图 9 所示。

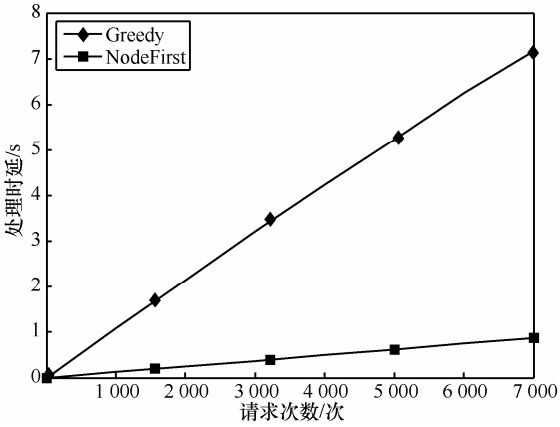


图 9　算法处理时延对比

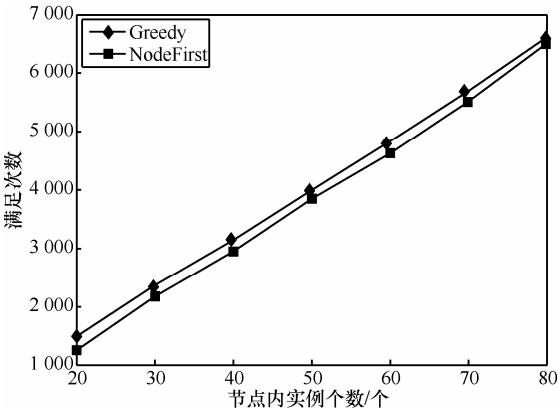


图 10　算法满足请求次数对比

实验结果表明，2 种算法的处理时延均与请求次数呈线性关系，且 Greedy 的处理时延比 NodeFirst 近似为 NodeFirst 的 9 倍，结果符合上文的理论分析。

3) 满足请求次数

相同请求次数下，Greedy 算法可搜索所有满足请求的实例链，故其找到满足请求的实例链的次数更多。图 10 对比了在不同 q 值情况下 2 种算法满足请求的次数。实验结果表明，Greedy 和 NodeFirst 算法满足请求的次数都随节点内实例个数 m 的增大而线性增大，且 Greedy 算法找到实例链的次数比 NodeFirst 算法多。这是因为 Greedy 算法的搜索空间包含所有的实例链，而 NodeFirst 的搜索空间相对较小，导致其在选择实例时可能会漏掉某些满足请求的实例链。

从 2 个算法的对比结果可以看出在相同请求下，NodeFirst 算法能在更短的处理时延下得到与 Greedy 相近的负载均衡度。综合时间复杂度和负载均衡度，NodeFirst 算法性能更优。

6.2　硬件系统仿真

Glen Gibb 等[13]在 2012 年基于 NetFPGA-1G[18]

平台实现了服务组合，但仅对组合机制进行了功能验证，并未给出具体的实验数据。本节沿用 Glen Gibb 等人在文献[13]中的仿真验证方法，在 NetFPGA-10G 平台搭建 4 个节点相连的拓扑并接入互联网，节点由 Ryu [19]控制器控制，如图 11 所示。

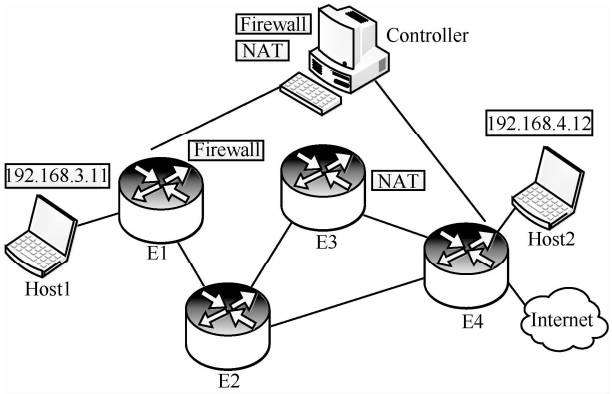


图 11 NetFPGA-10G 仿真环境

1) 元能力组合有效性验证

使用 Host1 和 Host2 作为源、目的节点，实验结果如表 2 所示。元能力组合最终的数据处理过程需通过流表项的形式在底层交换机中实现。首先，将元能力 Firewall 和 NAT 的表项分别下发到节点 E1 和 E3 上，见步骤 0，此时由于没有策略下发，2 个 Host 之间无执行路径。

下发策略<{Host1, Host2}, {Firewall}>，元能力编层根据策略及底层拓扑计算出执行路径，见步骤 1。此时将策略修改成<{Host1, Host2}, {Firewall,NAT}>，则数据流会先流入 E3 执行元能力 NAT 然后再转发至 E4，执行路径变成 E1→E2→E3→E4，见步骤 2。其中部署在节点上的实例以流表项的形式存在于数据平面，为将实例进行连接，元能力编排层在节点 E1 的 Firewall 实例表项后添加转发流表项：“dst=192.168.4.12; action=output:E2”，使数据流通过 Firewall 后直接被转发到 E2 节点；E2 是过渡节点，直接添加转发表项将数据流引到节点 E3：“dst=192.168.4.12; action=output:E3”；同样，节点 E3 的 NAT 实例表项后添加转发流表项：“dst=192.168.4.12; action=output:E4”。

至此，一条执行路径即安装完毕：“E1.Firewall → E1.output:E2 → E2.output:E3 → E3.NAT → E3.output:E4 → E4.output:Host2”，如图 12 所示。

最后，再将策略修改为<{Host1,Host2}, {NAT, Firewall}>，此时由于 E3 在 E1 节点之后，不能先到节点 E3 执行 NAT 后再返回 E1，所以 2 个 Host 之间无法生成相应执行路径，见步骤 3。表 2 中步骤 2 的往返延迟较大是因为元能力 NAT 需要修改数据分组的 IP 地址，此过程耗费一定的处理时间。实验数据表明，2 个 Host 之间的通信延迟符合与理论分析，实际数据流的转发路径也与元能力编排层所下发的执行路径相同。

2) 资源利用率对比

与不具备功能组合机制的传统网络相比，网络功能组合机制能够将功能分散到各个节点上，利用功能的灵活部署和组合实现各节点资源的充分利用。为对比元能力组合与传统网络对网络节点负载的影响，设置仿真环境，如表 3 所示。假设传统网络中的功能由交换节点完成，即所需功能在各节点中均有部署。

首先分别在 E1 和 E3 部署元能力 Firewall 和 NAT，得到执行路径 E1→E2→E3→E4；再在 E4 部署 NAT，则元能力编排层依据元能力分布重新计

表 2 **元能力组合对延迟影响**

步骤	下发策略	执行路径	往返延迟
0	installAC({E1},{Firewall}) installAC({E3},{NAT})	—	∞（Blocked）
1	installPolicy<{Host1, Host2},{Firewall}>	E1 ⟶ E2 ⟶ E4	0.19 ms
2	installPolicy<{Host1, Host2},{Firewall,NAT}>	E1 ⟶ E2 ⟶ E3 ⟶ E4	0.45 ms
3	installPolicy<{Host1, Host2},{NAT,Firewall}>	—	∞（No Path）

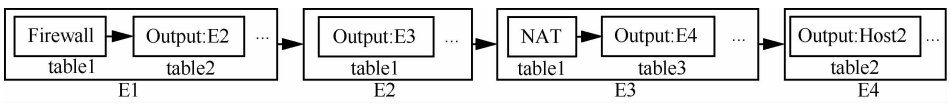


图 12 执行路径示例

表 3　　仿真环境设置

场景	下发元能力	下发策略	执行路径
元能力组合	installAC({E1},{Firewall})	installPolicy<{Host1, Host2},{Firewall,NAT }>	—
	installAC({E3},{NAT})		E1 → E2 → E3 → E4
	installAC({E4},{NAT})		E1 → E2 → E4
无功能组合	—	installPolicy<{Host1, Host2},{Firewall,NAT}>	E1 → E2 → E4

算执行路径得到：E1→E2→E4。而传统网络中节点依据最短路径寻址，因此 Host1 和 Host2 之间始终按照 E1→E2→E4 路径传递数据分组。发送 10 000 个数据分组，图 13 对比了元能力组合和无功能组合得到的各节点负载（记负载为从该节点流过的数据分组个数）。

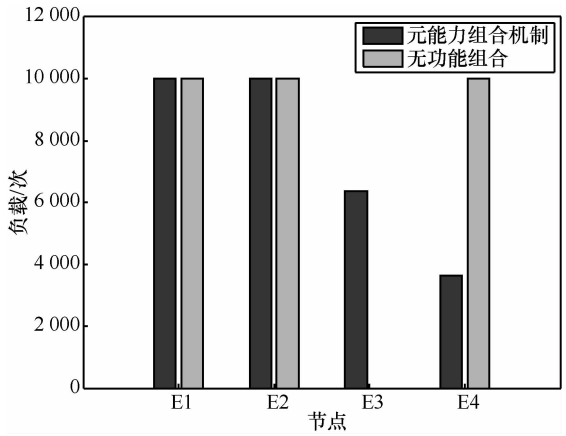


图 13　各节点负载均衡度对比

从图 13 可以看到，传统网络中的节点 E3 处于空载状态，导致该节点资源浪费。而元能力组合对节点 E3 内的实例表项有一定次数的使用。实验结果表明，相比于无功能组合，元能力组合机制能均衡地使用散落在网络中的实例表项，充分利用各节点资源。且从上节软件仿真的结果可以看出，当网络规模扩大，元能力的部署增多，各节点资源利用将更加均衡。

6.3　功能组合效率分析

现有模块组合编程语言（Frenetic[7]、Pyretic[6]等）通过将不同功能的流表项合并为单个流表项来实现功能组合。而本文所提元能力组合机制利用 OpenFlow1.3 协议的多级流表特性，将不同元能力表项下发到不同流表之中，多级流表的流水线处理使得功能组合仅需要根据流表级数与匹配域进行表项放置和连接。实验选取文献[6]中提供的 3 个功能实例表项（Monitor、Route、Load Balancer），分别随机生成 1 000~7 000 次组合策略，仿真平台同 6.1 节。图 14 对比了元能力组合机制与其他功能组合机制生成组合表项的时间复杂度。

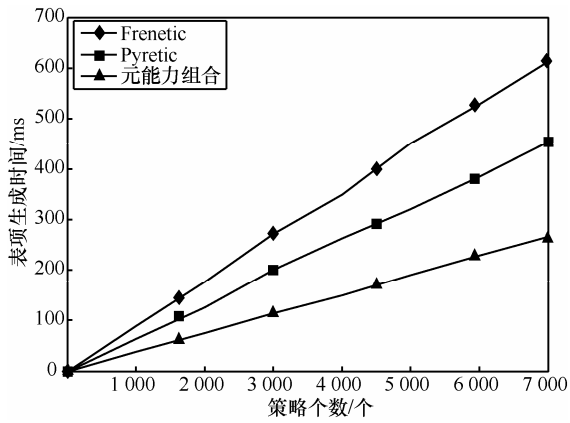


图 14　组合效率对比

Frenetic 采用并行的表项合并方法，Pyretic 则增加了串行合并方法。两者不仅要判断表项之间的可合并性，还要根据匹配域的类型进行合并，因此其生成组合表项的方法复杂度较高。仿真结果表明，相比于 Frenetic 和 Pyretic，元能力组合机制生成组合表项的时间复杂度较低，即组合效率更高。

7　结束语

本文主要研究 SDN 中的功能组合问题，通过引入元能力、组合链、策略、执行路径等概念，为 SDN 功能模块化提供标准的功能划分和描述。其次，针对元能力组合问题，建立基于二级映射的 0-1 线性规划模型，并提出节点优先算法对模型进行求解。最后，设计作为 SDN 应用层扩展结构的元能力编排层，用以对功能模块进行统一管控和组合编排，由此增强网络创新能力和管控能力。实验结果表明，文中所提功能组合机制具有有效性并达到负载均衡效果，且与其他组合机制相比具有更高的组合效率。本文所提功能模块划分及组合方法不仅在 SDN 中应用，在可重构网络等其他一些新型网络体系中同样具有适用价值。

与 NetFPGA-10G 节点相连的控制器应用层架构仍处于开发之中，这是本文的下一步工作。此外，所提节点优先算法虽优于贪婪算法，但其时间复杂度仍然较高，设计更加有效的网络功能组合算法也是本文的下一步工作。

参考文献：

[1] MCKEOWN N, ANDERSON T, BALAKRISHNAN H N, *et al*. Openflow: enabling innovation in campus networks[A]. SIGCOMM CCR[C]. Seattle, USA, 2008.69-74.

[2] WALFISH M, STRIBLING J, KROHN M, *et al*. Middleboxes no longer considered harmful[A]. Proceedings of OSDI 04[C]. Berkeley, USA, 2004.215-230.

[3] HANDIGOL N, SEETHARAMAN S, FLAJSLIK M, *et al*. Plug-*n*-serve: load-balancing web traffic using openflow[A]. ACM SIGCOMM Demo[C]. Barcelona, Spain, 2009.

[4] WANG R, BUTNARIU D, REXFORD J. Openflow-based server load balancing gone wild[A]. Workshop of HotICE'11[C]. Boston, USA, 2011.12.

[5] NAYAK A K, REIMERS A, FEAMSTER N, *et al*. Resonance: Dynamic access control for enterprise networks[A]. Proceedings of the 1st ACM workshop on Research on enterprise netwokings[C]. Barcelona, Spain, 2009.11-18.

[6] MONSANTO C, REICH J, FOSTER N, *et al*. Composing software-defined networks[A]. NSDI'13, LOMBARD[C]. IL, USA, 2013.1-14.

[7] FOSTER N, HARRISON R, FREEDMAN M J, *et al*. Frenetic: a network programming language[A]. Proceedings of ICFP[C]. Tokyo, Japan, 2011. 279-291.

[8] ERICKON D. The beacon OpenFlow controller[A]. Proceedings of 1st Workshop on HotSDN 2013[C]. Hong Kong, China, 2013. 13-18.

[9] MOGUL J C, YOUNG A, BANERJEE S, *et al*. Corybantic: towards the modular composition of SDN control programs[A]. Hotnets'13[C]. Washington D C, USA, 2013.1.

[10] Network Functions Virtualisation-Introductory White Paper[EB/OL]. http://portal.etsi.org/NFV/NFV_White_Paper.pdf, 2012.

[11] 程东年, 汪斌强, 王保进. 网络结构自调整的柔性内涵初探[J]. 通信学报, 2012,33(8):214-222.
CHENG D N,WANG B Q,WANG B J. Preliminary study on the connotation of flexibility in dynamically reconfigurable networks[J]. Journal on Communications, 2012, 33(8):214-222.

[12] 兰巨龙, 程东年, 胡宇翔. 可重构信息通信基础网络体系研究[J]. 通信学报, 2014,35(1):128-139.
LAN J L,CHENG D N,HU Y X. Research on reconfigurable information communication basal network architecture[J].Journal on Communications, 2014,35(1):128-139.

[13] GIBB G, ZENG H Y, MCKEOWN N. Outsourcing network functionality[A]. HotSDN'12[C]. Helsinki, Finland, 2012. 73-78.

[14] QAZI Z A, TU C C, CHIANG L, *et al*. SIMPLE-fying middlebox policy enforcement using SDN[A]. SIGCOMM2013[C]. Hong Kong, China, 2013.12-16.

[15] ZHANG Y, BEHESHTI N, BELIVEAU L, *et al*. StEERING：a software-defined networking for inline service chaining[A]. The 21st IEEE International Conference on Network Protocols(ICNP 2013)[C]. Gottingen, Germany, 2013.1-10.

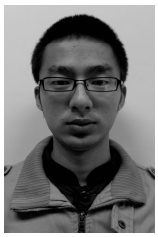
[16] SEYED K F, CHIANG L, SEKAR V. Enforcing network-wide policies in the presence of dynamic middlebox actions using FlowTags[A]. NSDI'14[C]. Seattle, WA, USA, 2014. 2-4.

[17] NetFPGA-10G project[EB/OL]. https://github.com/NetFPGA/ NetFPGA-public/wiki, 2013.

[18] NetFPGA-1G project[EB/OL]. https://netfpga.org/, 2011.

[19] Ryu controller project[EB/OL]. http://osrg.github.io/ryu, 2014.

作者简介：



段通（1992-），男，河南驻马店人，国家数字交换系统工程技术研究中心硕士生，主要研究方向为软件定义网络、可编程网络数据平面。

兰巨龙（1962-），男，河北张北人，国家数字交换系统工程技术研究中心总工程师、教授、博士生导师，主要研究方向为新一代信息网络关键理论与技术。

程国振（1986-），男，山东定陶人，国家数字交换系统工程技术研究中心博士生，主要研究方向为新型网络体系结构、软件定义网络、网络功能虚拟化。

胡宇翔（1982-），男，河南周口人，国家数字交换系统工程技术研究中心讲师，主要研究方向为新一代信息网络关键理论与技术。