



一种基于多级流表的中间件服务链的部署方法

刘释然, 兰巨龙, 胡宇翔, 段 通

(国家数字交换系统工程技术研究中心 郑州 450002)

摘 要:为了解决服务链部署中的路径循环和源地址绑定问题,首先,提出一种新的服务链部署方法,该方法利用 OpenFlow 多级流表标识服务链及服务顺序,从而使数据分组按照需求的服务顺序正确处理;其次,提出服务链部署算法以实现服务路径到流表规则的映射;最后基于 NetFPGA-10G 完成了原型系统实现。仿真结果表明,该部署方法具备可行性,并降低了流表数量和时间开销。

关键词:软件定义网络;中间件;服务链;多级流表

doi: 10.11959/j.issn.1000-0801.2015205

A Deployment Method of Middlebox Service Chain Based on Multi-Flow Table

Liu Shiran, Lan Julong, Hu Yuxiang, Duan Tong

(National Digital Switching System Engineering & Technological R&D Center, Zhengzhou 450002, China)

Abstract: Aiming at the path cycle and source address binding problem in middlebox service chain deployment, a service chain deployment approach by combining the OpenFlow mechanism and NetFPGA was proposed. First of all, the OpenFlow multi-flow table was used for marking service chains and labeling service sequences so that data packets could be processed correctly according to desired service order; secondly, the service chain deployment algorithm was designed to achieve the mapping relationship between the service path and flow table rules; thirdly, a prototype system was implemented based on the hardware platforms of NetFPGA-10G. Finally, experimental results show that the proposed method have the advantage of feasibility and outperforms compared methods in the aspect of the flow table number and time overhead.

Key words: software defined networking, middlebox, service chain, multi-flow table

收稿日期:2015-04-26;修回日期:2015-08-03

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(No.2012CB315901, No.2013CB329104),国家自然科学基金资助项目(No.61309019, No.61372121),国家高技术研究发展计划(“863”计划)基金资助项目(No.2013AA013505)

Foundation Items: The National Basic Research Program of China (973 Program) (No.2012CB315901, No.2013CB329104), The National Natural Science Foundation of China (No.61309019, No.61372121), The National High Technology Research and Development Program of China (863 Program) (No.2013AA013505)

论文引用格式:刘释然,兰巨龙,胡宇翔等.一种基于多级流表的中间件服务链的部署方法.电信科学,2015205

Liu S R, Lan J L, Hu Y X, *et al.* A deployment method of middlebox service chain based on multi-flow table. Telecommunications Science, 2015205

1 引言

传统网络越来越依托于先进的服务功能组合提供增值服务^[1],这些服务功能一般集成在物理设备上,以中间件(middlebox)^[2]的形式提供,如防火墙(firewall,FW)、入侵检测系统(intrusion detection system,IDS)、网络地址转换(network address translation,NAT)设备、代理服务器、应用网关等。现有研究表明,中间件已在网络安全保障、性能提高等方面起到至关重要的作用^[3-7]。根据现有的研究成果^[8-11],将数据分组按照一定顺序路由经过网络部署的一系列提供服务或功能的中间件的这种概念,称为“服务链”。

尽管中间件提供了至关重要的性能表现和安全益处,但构建基于中间件的服务链仍主要面临两方面的挑战:很难确保服务链策略被正确地执行,中间件一般有严格的执行顺序^[12],如本文第2节提及的路径循环问题就会使服务链策略难以被正确执行;由于中间件的状态性特点(如NAT会重写数据分组头),使得难以确保数据分组转发的正确性和有效性^[7],本文将该问题称为源地址绑定问题。

对此,StEERING^[9]利用软件定义网络(software defined networking,SDN)集中式管控的优势来实现中间件的部署和管理,其采用 OpenFlow 1.1^[13]多级流表对流表中 metadata(元数据)域进行编码,设定需要的服务类型和服务实例,从而将流量路由至相应的中间件。该方法考虑了通过改变中间件部署优化服务路径,但该方法对中间件可能会造成的数据分组头修改行为未作考虑。SIMPLE^[8]利用 OpenFlow 1.0^[14]流表中一些匹配域(VLAN、IP ToS 等),在交换机为服务链数据分组增加标签标识,以保证服务链策略正确执行,并分析数据分组前后进出中间件的相似性以解决中间件造成的数据分组头修改行为,但是这种方法需要控制器收集数据分组进行相似性比较,有较高的复杂性且需要较高的匹配精度。FlowTags^[10]打破将中间件看作黑盒的观点,通过修改中间件代码,使得中间件可以为数据分组增加标签,以更好地解决上述问题,但这种方法较为复杂,需要中间件通过专门的接口与控制器交互,在企业网的环境中,每种中间件设备的数量多达上百个^[15],面对不同厂商提供的不同种类的中间件设备,这种方法无疑使得管理员需要具备较高的专业知识且带来了较高的修改成本。

基于以上分析,为降低部署服务链的复杂度,本文提出一种新的服务链部署方法解决上述两个问题,该方法利

用 OpenFlow 多级流表中一些匹配域标识服务链并编排服务顺序,从而使数据分组按照服务链需求的服务顺序正确执行,并通过在交换机中部署 2 级流表的方式实现。其中服务路径到流表规则的映射通过本文提出的流表下发算法进行计算并部署。

2 问题描述

根据现有文献研究^[3,8-10],在实际部署服务链时,典型性地会遇到以下两个问题:一是路径循环问题;二是源地地址绑定问题。这两个问题的发生会使服务链对应的流表转发规则难以部署并且会使管理员欲部署的策略失效,以下针对这两个问题展开具体描述。

2.1 路径循环问题描述

本文将在网络拓扑中,一个服务链对应的数据分组会多次经过同一个交换机从而使该交换机存在多条不明确的转发规则的问题,简称为路径循环问题,或模糊转发问题^[9]。

以图 1 中 3 个交换机(S_1 、 S_2 、 S_3)、1 个 FW、1 个 IDS 的拓扑为例,现针对特定源地址构建一条先经过 IDS 再经过 FW 的服务链,如图 1 粗线所示。其中,src₁ 表示该服务链的源地址,粗线箭头表示数据分组的前进方向,粗线表示数据分组需要经过的路径,数字 1、2、3 表示该服务链数据分组会先后 3 次经过交换机 S_2 。交换机 S_1 、 S_2 中部署的该服务链数据分组转发规则分别见表 1、表 2,其中 IngressPort 表示数据分组进入该交换机的地址或来源,Fwd 表示转发方向。

在构建该服务链时,控制器^[18]会在 S_2 中部署 3 条流表转发规则,此时 S_2 中转发表会存在不明确的转发规则,即数据分组来源是 S_3 时,存在两个可能转发的路径,交给 IDS 或下一跳。这与 IETF 规定的服务链需要严格的执行顺序的原则相冲突。本文认为,该问题的核心是由于构建服务链时存在路径循环,从而导致流表部署时产生了不明确的转发规则,所以本文将该问题称为路径循环问题。

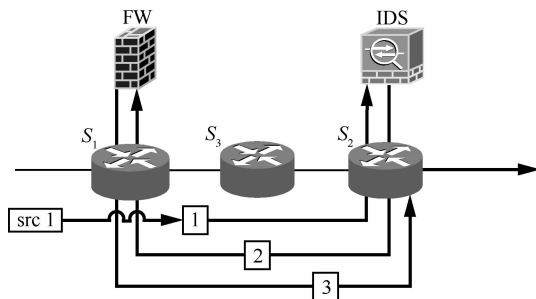


图 1 路径循环问题示意

表 1 交换机 S_1 中转发规则

IngressPort	Fwd
src ₁	S_3
S_3	FW
FW	S_3

表 2 交换机 S_2 中转发表存在不明确的转发规则

IngressPort	Fwd
S_3	IDS
IDS	S_3
S_3	下一跳

2.2 源地址绑定问题描述

源地址绑定问题源于中间件对数据分组所采取的修改行为会与 SDN 中两个关键的原则相冲突^[10]。

- 源地址绑定原则:数据分组和它的“源”应该有一个很强的绑定关系(如最初创造数据分组的网络实体)。
- 路径遵循策略原则:明确的策略应当决定数据分组的执行路径。

例如,NAT 和负载均衡等中间件设备会动态地重写数据分组头,从而违反源绑定原则,并且这种中间件动态的修改行为也可能会造成与“路径遵循策略”原则相冲突。

以如图 2 所示的拓扑为例,管理员根据主机 H_1 和 H_2 的源 IP 地址配置了交换机的转发规则和防火墙的过滤规则,从而部署了如图 2 中虚线所示的服务链,该服务链的服务顺序是先经过 NAT 再经过 FW。其中 NAT 转换私有 IP 地址到公有 IP 地址,FW 阻止主机 H_1 访问特定的公有 IP 地址,允许主机 H_2 访问特定公有 IP 地址。但现由于 NAT 会动态地修改数据分组头 IP 地址,从而会使管理员部署的 H_1 和 H_2 的服务链策略失效,即由于数据分组 IP 地址的改变, S_1 、 S_2 难以按照管理员预先部署的流表规则进行转发,而 FW 由于 IP 地址的改变也会难以进行拦截过滤,从而使得数据分组所走的路径不再遵循策略,即使部署的服务链策略失效。相较而言,子网 2 中的主机均未部署需经过中间件的服务链,则交换机 S_1 和 S_2 均能正常进行转发。

“路径遵循策略”原则不能很好地执行是由于源地址未能强绑定所致,因此本文将这种由于中间件的修改行为而造成的源地址改变,从而使部署的服务链策略失效的问题称为源地址绑定问题。

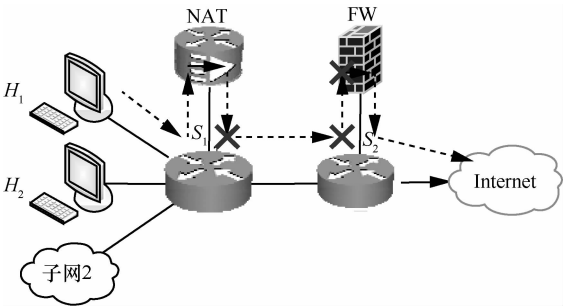


图 2 源地址绑定问题示意

3 基于多级流表的服务链部署方法

相较于传统网络中手动/半手动的静态路由策略^[4]及现有的一些基于 SDN 的解决方案,为降低部署服务链的复杂度,提出了一种基于多级流表的解决机制,该机制通过对 OpenFlow1.1 流表中 metadata(元数据)域编码,从而解决路径循环问题,并通过对数据分组增加标签标识从而标识服务链的数据分组,以便解决源地址绑定问题。

3.1 服务顺序编码

本文对 OpenFlow1.1 流表中的 metadata 域中的不同位编码,从而表示网络拓扑中不同的中间件类型和中间件实例,通过编码的方式表示特定的服务顺序,从而解决路径循环问题,本文将这种编码机制称为服务顺序编码。

为了问题说明的简洁性,假设 metadata 域是 4 bit(实际是 64 bit),初始为 0000,网络中只有 FW 和 IDS 两种中间件,每种中间件各 2 个实例,则可将 metadata 域分为高位(前两个 0 位)和低位(后两个 0 位)两个部分,高位用来表示 FW,以 01 表示第一个 FW,记为 FW₁,10 表示第二个 FW,记为 FW₂,低位同理表示两个 IDS。现针对如图 1 所示的网络拓扑中源地址 src₁ 构建经过 IDS₁ 和 FW₁ 的服务链,则 metadata 域对应的服务顺序编码可表示为 0101,如图 3 所示。

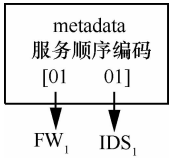


图 3 编码 metadata 域示例

交换机 S_2 在收到来自 src₁ 的数据分组后,会根据流表 metadata 域中数值的不同,决定转发顺序以及对 metadata 域的修改动作,见表 3。当数据分组被转发至相应的中间件实例时,交换机会将该中间件实例对应的 metadata 域中的服务顺序编码位重新置 0,以表示数据分组已经过该中



表 3 交换机 S2 中根据流表 metadata 域决定转发/修改行为

IngressPort	元数据	指令
S_1	0101	metadata:0101→0100; Fwd: IDS ₁
IDS ₁	0100	metadata:0100→0000; Fwd: S_1
S_1	0000	Fwd:下一跳

间件实例,从而避免数据分组再次经过 S_2 时被再次转发给此中间件实例,决定了路径循环时数据分组需要经过中间件的服务顺序。

在实际部署中,管理员可以根据中间件种类及数量合理划分 metadata 域,如取 metadata 域前 32 bit 用来编码表示中间件,并等分 4 份,每份 8 bit 用来表示 4 种不同的中间件,并通过向控制器中设置一个中间件到服务顺序编码的映射表实现,见第 4.1 节。

3.2 服务链标签

仅对 metadata 域编码仍不能解决源地址绑定问题,为此,本文提出了为服务链数据分组增加服务链标签的方法。该方法采用 OpenFlow 流表中未使用域为服务链增加标签(tag)标识,该标签所在域满足不会被中间件修改的特点,从而使得当服务链数据分组头中的匹配域被中间件修改而使交换机无法识别服务链数据分组时,交换机可以按照服务链标签识别同一服务链的数据分组,使得数据分组正确转发继而使得服务链策略得以正确执行。根据服务链标签的不同也可以识别不同的服务链数据分组,见表 4,交换机根据标签的不同,所采取的处理动作也不同。

为了使上述机制得以正确地部署和执行,本文使用了基于 OpenFlow1.1 多级流表的 2 级流表的设计,如图 4 所示,第一级流表为基本转发表,第二级流表为服务链表,该

表 4 通过标签的不同标识不同的服务链

IngressPort	标签	元数据	指令
S_1	1	0101	metadata:0101→0100; Fwd: IDS ₁
IDS ₁	1	0100	metadata:0100→0000; Fwd: S_1
S_1	1	0000	Fwd:下一跳
S_1	2	0101	Fwd:下一跳

设计可以满足需要构建服务链的数据分组和按照 IP 五元组转发的一般数据分组被交换机正常转发的需求。基本表的匹配域是 IP 五元组和交换机进口,不管是服务链的数据分组还是一般数据分组只要能在基本表中,则执行指令规定的相应动作,如转发、修改、跳到下一级流表。当数据分组因中间件动态修改分组头的行为而造成数据分组在交换机中无法命中基本表时,此时默认交由第二级流表服务链表,并根据标签和 metadata 域等字段查阅能否命中,若命中,则根据指令执行相应动作,否则上报控制器进行进一步处理。

4 服务链部署过程及算法

4.1 部署过程

整个服务链的构建过程可以看作一个由“策略—服务路径—流表规则”的 2 级映射的过程,如图 5 所示,即“策略—服务路径”映射和“服务路径—流表规则”映射,其定义如下。

定义 1 (策略)由管理员或上层应用向控制器下发的一组构建服务链的请求。 $p=\langle \text{src}, \text{mp}, \text{des} \rangle$,src 代表源地址,mp 代表预构建的服务链中的中间件类型的集合,des 代表目的地址。 P_i 代表一个策略实例,如 $P_i=\{\text{src}_k-\text{NAT}-\text{FW}-\text{des}_k \mid k \in N\}$ 。

定义 2 (服务路径)指承载策略的物理序列,该序列包

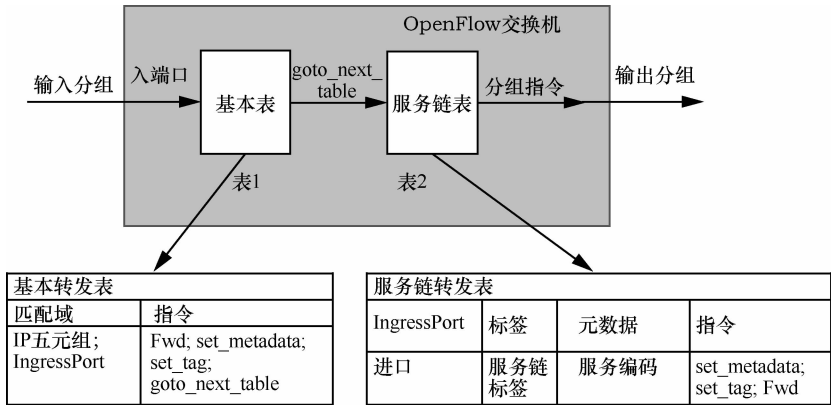


图 4 基于 2 级流表的基本表和服务链表设计

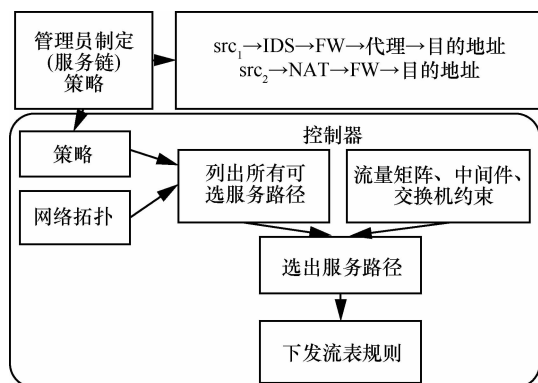


图 5 整个服务链构建过程

含了策略中所需的所有中间件类型和满足传输需求的交换机。为了和现有一些相关研究^[8-10]进行比较,本文研究的服务路径有两个限定条件:每个中间件至少与一个交换机相连;中间件之间不存在互连。本文针对满足上述两个限定条件的服务路径下发流表,所以 $sr=\{node_1, node_2 \cdots node_{j-1}, node_j, node_{j+1} \cdots node_n | node_{type} \in \{s, m\}, \forall j \in Z^+\}$ 是策略 P_i 的一条服务路径,当且仅当:

- 节点 $node_j$ 和 $node_{j+1}$ 相互连通, $1 \leq j \leq n-1$;
- 节点类型 $node_{type}$ 属于交换机节点(用 S 表示)或中间件节点(用 M 表示);
- 当 $node_{type}=M$ 时, $node_{j-1}=node_{j+1}$ 且 $node_{j-type}=S, node_{j+1-type}=S$;
- $node_1$ 与 P_i 中的源节点相连; $node_n$ 与 P_i 中的目的节点相连;

$SR \supseteq M_p$, 即 SR 中中间件节点的类型集合包含 P_i 需求的中间件类型集合。

定义 3 (“策略—服务路径”映射) 指以服务链构建策略 P 为需求, 以 SR 包含的节点资源和性能指标等为约束, 选取合适的物理链路, 作为承载服务链的服务路径。

定义 4 (“服务路径—流表规则”映射) 针对所选服务路径, 在其包含的交换机节点上下发相应的流表转发规则, 使服务链最终完成部署。

目前本文针对“服务路径—流表规则”的映射, 提出了相应的流表下发算法, 并通过本文开发的流表配置管理软件实现了服务路径到流表规则的流表下发工作, 可以完成基于基本表和服务链表的 2 级流表的服务链正确部署。

4.2 “服务路径—流表规则”的流表下发算法

当一个部署服务链请求下发控制器时, 假定控制器已根据设置选定了一条服务路径, 此时控制器需要针对该服

务路径上的交换机节点下发相应的流表转发规则。如在图 1 所示的网络拓扑中, 对于策略 $P_i=\{src_1-IDS-FW-des_1\}$, 控制器选定了一条可选服务路径, 其表示为 $SR=\{S_1-S_3-S_2-IDS_1-S_2-S_3-S_1-FW_1-S_1-S_3-S_2\}$, 此时需要控制器针对该服务路径上交换机节点 S_1, S_3, S_2 上下发相应的流表规则, 以完成服务链的部署。

在实际下发流表规则时, 根据服务路径有无循环以及包含不包含修改分组头的中间件, 共分为如下 4 种类型:

- ① 无路径循环, 无修改分组头的中间件;
- ② 无路径循环, 有修改分组头的中间件;
- ③ 有路径循环, 无修改分组头的中间件;
- ④ 有路径循环, 有修改分组头的中间件。

为了完成针对这 4 种不同类型的服务路径对应的流表下发工作, 本文提出了算法 1 节点归类算法和算法 2 流表下发算法, 从而完成“服务路径—流表规则”的映射工作, 其中算法 1 是算法 2 的前置步骤, 先将服务路径上节点归类到两种不同的序列中, 并判断服务路径是否存在路径循环及修改分组头的中间件。算法 2 根据算法 1 得出的服务路径类型及求得的两种不同序列, 从而决定如何下发流表。

为了形式化地描述服务路径对应的流表规则下发问题, 本文定义一些变量, 具体见表 5。

服务路径上除去首尾节点其余交换机节点 S_i 均属于 $S_m-S_i-S_n$ 或 $S_i-M_k-S_i$ 这两种序列, 把该服务路径上的源主机端和最终目的端也看作交换机节点, 则首尾节点也能归类于上述两种序列。之所以需将服务路径上交换机节点归类是因为针对这两种序列需要的表不一样, 如与中间件相连的交换机 S_i 至少需要两条流表规则, 处于 $S_m-S_i-S_n$ 序列中的 S_i 可能只需要一条(注: 这里仅仅考虑由源节点到目的节点的单向路径, 因为从目的节点到源节点的路径可以看作一条新的服务路径)。至此, 算法 1 共完成两件工作: 判断 SR 的类型, 并将交换机节点归类记录到上述两种序列中, 为算法 2 作前置工作。

算法 1 节点归类算法

$SR = \{node_1 - \cdots - node_i - \cdots - node_n | node_{type} \in \{S, M\}\}$,
 $judgeLoopIndex = -1$; /* $judgeLoopIndex$ 用来记录可能会出现循环的节点下标 */

Function(SR)

for $i=0:N-1$

if ($node_{i,type} == S$)

if ($node_{i,need_skip} == false \ \&\& \ node_{i+1,type} == S$) //如果



表 5 主要的符号定义及具体含义

符号	含义
SR	服务路径
S	交换机节点
M	中间件节点
$S_m-S_i-S_n$	该序列表示 SR 中 3 个相连的交换机,如图 1 中 $S_1-S_3-S_2$
$S_i-M_k-S_i$	该序列中表示 SR 与中间件 M_k 相连的交换机 S_i 与 M_k 构成的序列
$Modify(M)$	集合 $Modify(M)$ 表示该网络中所有会修改数据分组头的中间件的类型的集合,由管理员定义
$basic_table$	基本转发表
SC_table	服务链转发表
$M(SR)$	SR 中所有中间件节点
$map(M(SR))$	SR 上中间件到服务顺序编码的映射,由管理员定义

当前节点的后继节点是交换机节点,记当前节点和后继节点为 S_k-S_L 且当前节点不需要跳过,即当前节点之前的节点未出现 S_k-S_L 序列,则未出现循环,则记 $judgeLoopIndex = i$;

```

    judgeLoopIndex = i;
    if (nodei-1.type == S)
        nodei-1 → nodei → nodei+1 存入  $S_m-S_i-S_n$  序列;
    end;
end;
if (nodei+1.type == M)
    nodei-1 → nodei → nodei+1 存入  $S_m-S_i-S_n$  序列;
    nodei+1 → nodei+2 → nodei+3 存入  $S_m-S_i-S_n$  序列;
end;
if judgeLoopIndex! = -1
    for j=judgeLoopIndex+2:N-1
        if (nodej.type == S && nodej+1.type == S)
            if (nodej == nodejudgeLoopIndex && nodej+1 == nodejudgeLoopIndex+1)
                exist_loop = true; /*SR 存在循环,将该变量置为
true,为算法 2 记录 */
                nodej.need_skip = true; /* 跳过已判断出现循环
的节点 */
            end;
        end;
    end;
end;
else /*nodei.type == M */
    if (exist_modify == false)
        for k=0:M
            if (nodei ∈ Modify(M)) /*判断 M 是否 ∈ Modify(M)*/
                exist_modify = true; /*SR 存在修改分组头

```

的中间件,将该变量置为 true,为算法 2 记录 */

```

        end;
    end;
end;
judgeLoopIndex = -1;
end;

```

算法 1 通过记录已出现循环的节点的下标并跳过该节点,从而避免了对该节点所属序列是否会出现循环的重复判断,即判断该序列之后是否会出现相同序列,从而避免了重复记录该节点到 $S_m-S_i-S_n$ 序列,使得算法 2 在对 $S_m-S_i-S_n$ 序列下表时避免了重复。

算法 2 针对算法 1 判断出的 SR 属于 4 种类型中的哪一种以及将 SR 上交换机节点 S_i 归类出的 $S_m-S_i-S_n$ 和 $S_i-M_k-S_i$ 两种序列,有区分地对交换机 S_i 下发流表。对于类型①,不需要部署服务链表,只需依据策略 P_i 中源地址和目的地址下发基本转发表规则即可。对于类型②,由于存在修改分组头的中间件,所以需要部署服务链表及分配标签,由于不存在循环所以不需要编排 metadata 域。对于类型③、④存在循环,均需部署服务链表及编排 metadata 域,类型③无修改分组头的中间件,不需要分配标签,类型④则需要分配标签。类型③、④的主要不同还在于当 $S_j \in S_m-S_i-S_n$ 时,对于 S_j 下表的的不同处理:类型③不需要在 S_j 的服务链表部署对应的转发规则,相当于节省了 S_j 部分流表空间,类型④由于存在有修改分组头的中间件,需要在 S_j 的服务链表部署对应的转发规则。

算法 2 流表下发算法

```

Function()
if (exist_loop == false)
if (exist_modify == false)/* 类型①*/

```

```

for (i = 0; i < k_m; i++) /* $S_m-S_i-S_n$  中的  $S_i$ */
    distribute one rules to basic_table of  $S_i$ ;
end;
for (i = 0; i < k_s; i++) /* $S_m-S_i-S_n$  中的  $S_i$ */
    distribute one rules to basic_table of  $S_i$ ;
end;
else /* 类型②*/
    assign Tag; /* 类型②需要分配 Tag*/
    for (i = 0; i < k_m; i++) /* $S_i-M_k-S_i$  中的  $S_i$ */
        distribute onerules to basic_table and SC_tableof $S_i$ ;
    end;
    for (i = 0; i < k_s; i++) /* $S_m-S_i-S_n$  中的  $S_i$ */
        distribute onerules to basic_table and SC_tableof $S_i$ ;
    end;
end;
else /* 类型③、④均有循环,需要编排 metadata*/
    strcat(metadata, Map(M(SR))); /* 将服务顺序编码
记录到 metadata 域中 */
    if (exist_modify == false) /* 类型③*/
        for (i = 0; i < k_m; i++) /* $S_i-M_k-S_i$  中的  $S_i$ */
            distribute one rules to basic_table and SC_table
of  $S_i$ ;
        end;
        for (i = 0; i < k_s; i++) /* $S_m-S_i-S_n$  中的  $S_i$ */
            distribute one rules to basic_table of  $S_i$ ;
        end;
    else /* 类型④*/
        assign Tag; /* 类型④需要分配 Tag*/
        for (i = 0; i < k_m; i++) /* $S_i-M_k-S_i$  中的  $S_i$ */
            distribute one rules to basic_table and SC_table
of  $S_i$ ;
        end;
        for (i = 0; i < k_s; i++) /* $S_m-S_i-S_n$  中的  $S_i$ */
            distribute one rules to basic_tableand SC_table
of  $S_i$ ;
        end;
    end;
end;

```

假设一条服务路径中有 n 个节点, m 个中间件节点, 集合 $Modify(M)$ 中的数量为 M_m , 整个网络中部署的中间件

种类数为 M_i 。算法 1 首先判断服务路径中有无循环、有无修改分组头的中间件, 时间复杂度为 $O(n_2+nM_m)$; 算法 2 对 $S_i-M_k-S_i$ 和 $S_m-S_i-S_n$ 两个序列中 S_i 下表的时间复杂度 $O(n-m)$ 。当服务路径属于类型③、④时, 编排 metadata 的时间复杂度为 $O(mM_i)$, 所以最大时间复杂度为 $O(n-m+mM_i)$ 。当 $n \gg m$ 时, 算法 1 的时间复杂度近似为 $O(n^2)$; 由于中间件种类数一般已知且数量有限, 即 $M_i \ll n$, 当 $n \gg m$ 时, 算法 2 的时间复杂度可近似为 $O(n)$ 。

5 服务链部署原型实现

为了实现上述基于 2 级流表的服务链部署方法, 本文数据层使用 NetFPGA-10G^[16]仿真验证平台, 目前已在该平台实现基于 OpenFlow 1.3^[17]协议的多级流表设计, 从而为本文提出的服务链部署方法提供了实际支撑。其中 NetFPGA-10G 板卡相当于 OpenFlow 交换机, 负责数据平面转发, 采用 TCAM (ternary content addressable memory, 三态内容寻址存储器) 和 BRAM (bipolar random access memory, 双极随机存取存储器) 作为流表存储单元, 最多支持 4 级流表串联的流水线处理, 在做仿真实验时只部署前两级流表, 用以实现基本表和服务链表。

控制层选取 Ryu^[18]作为控制器进行节点的控制与管理。由于应用层 App 仍在开发之中, 目前本文提出的流表下发算法通过 C 语言代码实现, 共计 1 514 行代码, 以软件程序的方式运行在 Linux 主机内。该软件负责针对已选好的服务路径上节点归类, 判断服务路径类型, 并针对服务路径上节点的流表配置下发工作, 其中流表规则通过驱动模块下发至 NetFPGA 寄存器系统, 实现对板卡的配置。

6 实验与分析对比

从 3 方面对所提服务链部署机制进行性能评估。首先, 通过实验平台进行可行性测试; 其次, 对算法部分额外带来的时间开销进行分析; 最后, 对所提流表下发算法的流表存储资源与时间开销进行比较分析。

6.1 可行性测试

NetFPGA-10G 仿真验证平台共有 5 台支持 OpenFlow 1.1 协议的交换机, 在该试验平台搭建如图 6 所示的试验环境, 3 台交换机节点均通过本文开发的流表配置软件配置流表规则。其中 NAT₁ 和 FW₁ 的功能基于交换机 S₂ 和 S₄ 模拟, 在 S₂ 上设置 IP 地址转换的规则, 在 S₄ 上设置防火墙过滤的规则。

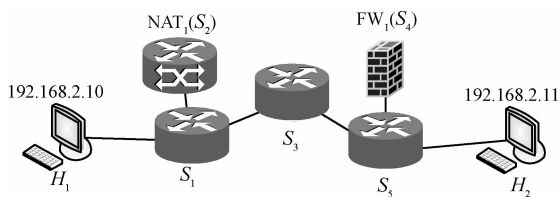


图6 仿真试验环境

以 H_1 、 H_2 表示源、目的节点, 策略 $P_1=\{H_1\text{-NAT-FW-}H_2\}$, 策略 $P_2=\{H_2\text{-}H_1\}$, 假定控制器已选取 P_1 的服务路径 $SR_1=\{S_1\text{-NAT}_1\text{-}S_1\text{-}S_3\text{-}S_5\text{-FW}_1\text{-}S_3\}$, P_2 的服务路径 $SR_2=\{S_5\text{-}S_3\text{-}S_1\}$ 。首先只基于源 IP 地址部署两条服务路径对应的转发规则, 即只在基本转发表上部署流表转发规则, 查看数据分组往返时延; 然后运行本文开发的流表配置软件, 通过本文所提算法逻辑去部署服务路径对应的转发规则, 即在基本转发表和服务链表均部署转发规则, 再次查看数据分组往返时延, 见表 6。经测试, 第一种方式由于 NAT 会改变源 IP 地址, 而 S_1 对于匹配不到的数据分组默认丢弃处理, 导致 H_1 和 H_2 之间 ping 不通, 即策略 P_1 未成功部署, 而基于本文所提方法部署, 可以使得 H_1 和 H_2 之间 ping 通, 即使服务链构建策略 P_1 和 P_2 成功部署。

其中, 为了使 FW_1 也能正确进行过滤工作, 管理员需对 FW_1 的过滤规则进行更改, 如为了过滤特定服务链的数据分组, 需要使过滤规则针对数据分组头中的服务链标签位进行过滤, 管理员可以通过流表配置软件获得所需服务链的标签。

进一步的, 本文通过实验验证了第 3.1 节所提的服务顺序编码机制可以正确工作, 见表 6 第 3 行。

6.2 算法时间开销分析

流表配置软件中算法部分所带来的时间开销使用 Visual Studio 2013 运行统计, 计算机处理器为 InterCore i7-3770 CPU 3.4 GHz, 4 GB RAM。

一条服务路径的时间复杂度主要受节点数量的影响, 当中间件数量较小、交换机数量较多时, 算法的时间开销主要受交换机数量影响。根据第 4 节服务路径类型的描述, 共设置 4 种类型的服务路径用于统计算法 1、算法 2 和

整个算法逻辑(算法 1+算法 2)所带来的处理时延。类型①~④的中间件集合分别为 $\{IDS_3, FW_2\}$ 、 $\{NAT_1, FW_2\}$ 、 $\{IDS_3, FW_2\}$ 、 $\{NAT_1, FW_2\}$, 其中第 1 个中间件与第 1 个交换机相连, 第 2 个中间件与第 5 个交换机相连, 类型①、②先访问第一个中间件后访问第二个中间件, 类型③、④先访问第二个中间件再访问第一个中间件; 拓扑中交换机节点数量 $\{S_1 \cdots S_j \cdots S_n \mid n \in [5, 10, 15, 20]\}$, 类型①、②服务路径中的交换机节点数量与拓扑中一致, 类型③、④需加上路径循环时多经过的交换机节点数量。不同类型下的处理时延如图 7 所示。

由图 7(a)可知, 对于算法 1, 当路径中有循环时, 意味着服务路径比没有循环时会多经过交换机节点, 所以类型③、④的处理时延大于类型①、②的处理时延。由图 7(b)和图 7(d)可知, 算法 2 下发流表产生的时延受服务路径需要下发的流表数量影响较大, 所以当类型②需要下发的流表数量逐渐大于类型③时, 其所耗费的处理时延也超过了类型③。

由图 7(a)、图 7(b)和图 7(c)可知, 由于算法 2 的时延远大于算法 1 判断服务类型的时延, 整个算法处理时延主要受算法 2 影响, 与服务路径中交换机节点数量呈线性关系, 符合上述理论分析。当服务路径中交换机节点数在 20 个以内时, 对同一服务路径每千次的处理时延在毫秒级, 最大不超过 1 s, 可以满足控制器对处理时延的要求。

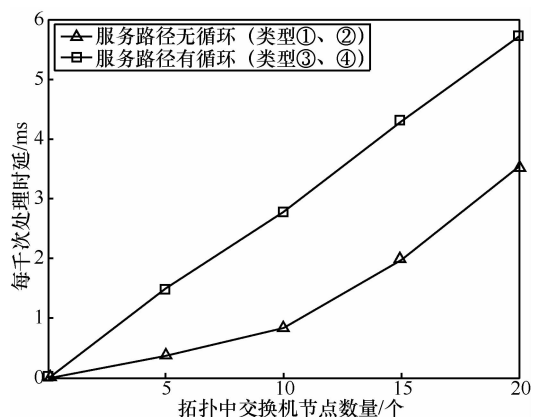
6.3 流表存储资源与时间开销分析

本文流表产生数量依据本文算法 2 得出。假设服务路径中 $S_i\text{-}M_k\text{-}S_i$ 序列中交换机 S_i 的数量为 N_M , $S_m\text{-}S_i\text{-}S_n$ 序列中交换机 S_i 的数量为 N_S , 则对于不同服务路径类型产生的流表规则数量为: 类型①为 N_M+N_S 条, 类型②为 $2N_M+2N_S$ 条, 类型③为 $2N_M+N_S$ 条, 类型④为 $2N_M+2N_S$ 条。

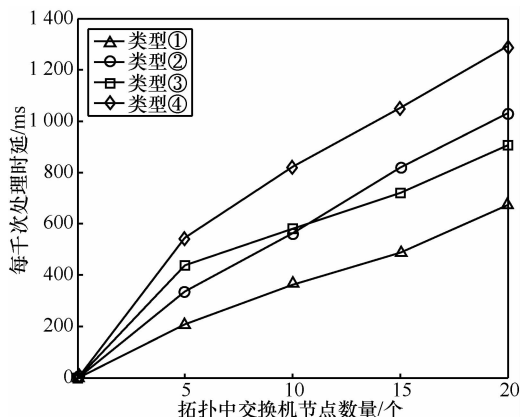
根据 4 种类型服务路径的设置, 取拓扑中交换机节点数量为 $\{S_1, \cdots, S_5, \cdots, S_{20} \mid n=20\}$, 本文与 StEERING、SIMPLE、FlowTags 流表产生数量的对比如图 8(a)所示, 相应的流表耗费存储空间对比如图 8(b)所示, 假定本文与 StEERING、SIMPLE、FlowTags 的每条流表下发时间均与第 6.2 节一致, 相应的流表下发时间开销对比如图 8(c)所示。

表 6 两种服务链部署方式比较

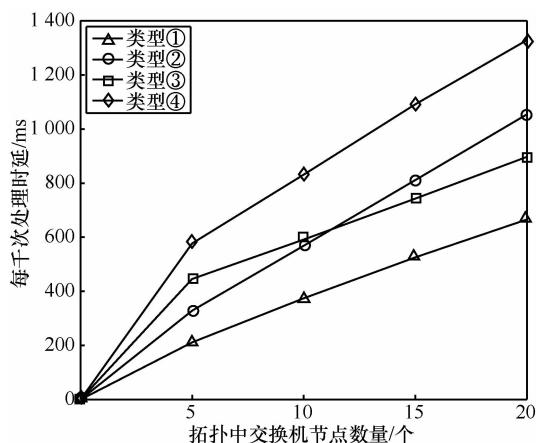
策略 P	服务路径 SR	部署方式	往返时延/ms
$H_1\text{-NAT-FW-}H_2$ $H_2\text{-}H_1$	$\{S_1\text{-NAT}_1\text{-}S_1\text{-}S_3\text{-}S_5\text{-FW}_1\text{-}S_3\}$ $\{S_5\text{-}S_3\text{-}S_1\}$	基于源 IP 地址	∞
$H_1\text{-NAT-FW-}H_2$ $H_2\text{-}H_1$	$\{S_1\text{-NAT}_1\text{-}S_1\text{-}S_3\text{-}S_5\text{-FW}_1\text{-}S_3\}$ $\{S_5\text{-}S_3\text{-}S_1\}$	基本转发表和服务链表	0.61
$H_1\text{-FW-NAT-}H_2$ $H_2\text{-}H_1$	$\{S_1\text{-}S_3\text{-}S_5\text{-FW}_1\text{-}S_5\text{-}S_3\text{-}S_1\text{-NAT}_1\text{-}S_1\text{-}S_3\text{-}S_5\}$ $\{S_5\text{-}S_3\text{-}S_1\}$	基本转发表和服务链表	0.83



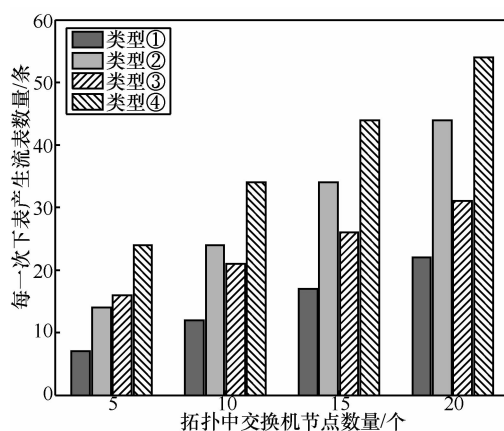
(a) 服务路径有无循环对处理时延的影响



(b) 服务路径类型对下流处理时延的影响



(c) 服务路径类型对整体处理时延的影响



(d) 相同拓扑下服务路径不同类型对流表数量的影响

图 7 不同类型下的处理时延

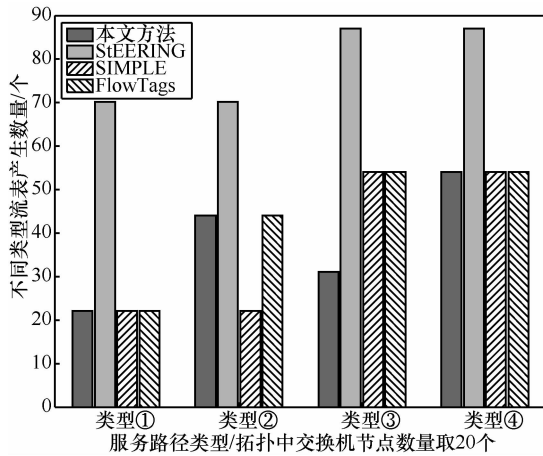
在解决路径循环问题(类型③)时,StEERING 采用 6 级流表,SIMPLE 需要每个交换机维持转发表和隧道表,FlowTags 并未解决该问题,由图 8(a)可知,本文所提方法有效降低了流表数量。在解决源地址绑定问题(类型②)时,StEERING 并未解决该问题,SIMPLE 使用单级流表并通过控制器去分析数据分组的方式解决,并未通过部署流表的方式解决该问题,FlowTags 也使用单级流表,但交换机收到被中间件添加标签的数据分组时,控制器会再次下发流表,而本文通过部署两级流表的方式解决该问题,由图 8(a)可知,本文流表产生数量多于 SIMPLE,与 FlowTags 持平。

由图 8(b)可知,在流表存储资源耗费上,在解决路径循环问题(类型③)时,本文较 SIMPLE 节省了一定的存储空间,但多于 StEERING。这是由于本文使用 2 级流表的部署方式,其匹配域较 SIMPLE 而言可以灵活设置,但灵活度不如 StEERING 使用的 6 级流表。在解决源地址绑定问题(类型②)时,由于本文不需要多次下发流表,所需存储

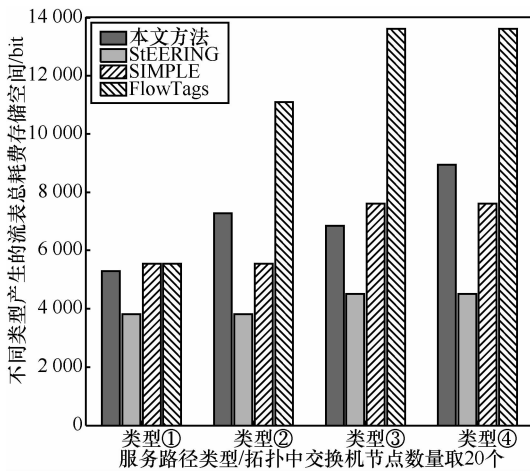
资源少于 FlowTags。

由图 8(c)可知,较现有部署流表的方案相比,本文方案降低了流表数量,时间开销相对更小。

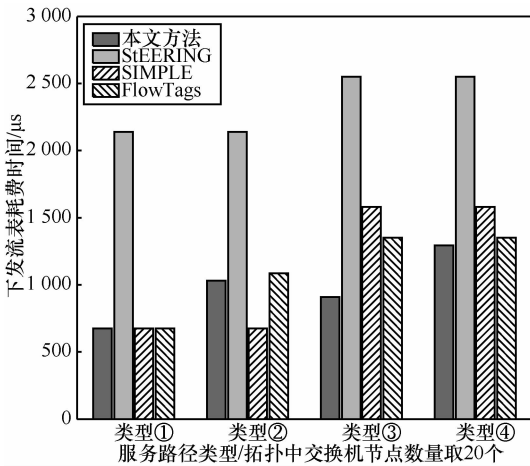
虽然 StEERING 更多的级数设计可以减少流表存储资源的开销,但需要下发更多的流表规则增加了下发流表的时间开销,而且由于多级流表流水线处理的特点,更多的级数意味着交换机的流表匹配处理时延的增加以及数据流量生成及维护的算法复杂度的提高。与 StEERING 相比,本文所提方法降低了流表产生数量与时间开销,但增加了一定的流表存储资源开销。与 SIMPLE 相比,在解决源地址绑定问题时,本文所提方法通过增加服务链表的设计,增加了一定的流表数量,但减轻了控制器的部署与管理任务。本文方法只需控制器向相应的交换机下发流表规则,而不需要在控制器上额外开发部署分析机制参与服务链数据分组后续的转发控制,从而节省出了控制器处理以及与交换机交互的时间,从而降低了控制器部署服务链的复



(a) 流表产生数量对比



(b) 流表存储资源开销对比



(c) 下发流表时间开销对比

图8 流表开销分析

7 结束语

本文针对服务链部署中的路径循环问题和源地址绑定问题提出了一种新的服务链部署方法,该方法不需要在控制器上额外部署分析机制去参与服务链数据分组后续的转发控制,并且不需要改变网络拓扑及修改中间件。首先,本文使用 OpenFlow 多级流表中一些匹配域为服务链增加标签标识服务链,并在 metadata 域上进行服务顺序编码,从而使数据分组按照需求的服务顺序正确处理。其次,本文在 NetFPGA-10G 中部署基于 2 级流表的基本表和服务链表,完成了所提方法的原型实现。实验结果表明,本文所提服务链部署方法可以解决上述两个问题,且在解决路径循环问题时较现有方法降低了流表数量与时间开销,在解决源地址绑定问题时,在少量增加流表存储资源开销的情况下,降低了服务链部署的复杂度及与控制器交互的时间。目前,关于服务路径的选取工作仍在进行之中,这将是本文的下一步工作。

参考文献

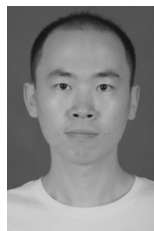
- 1 Boucadair M, Jacquenet C, Parker R, et al. Service Function Chaining: Framework & Architecture. Internet-Draft draft-boucadair-sfc-framework-00, IETF Secretariat, 2013
- 2 Srisuresh P, Kuthan J, Rosenberg J, et al. Middlebox Communication Architecture and Framework. RFC3303, 2002
- 3 Joseph D A, Tavakoli A, Stoica I. A policy-aware switching layer for data centers. Proceedings of Association for Computing Machinery's Special Interest Group on Data Communications, Chicago, USA, 2008
- 4 Gill P, Jain N, Nagappan N. Understanding network failures in data centers: measurement, analysis, and implications. Proceedings of Association for Computing Machinery's Special Interest Group on Data Communications, Toronto, Canada, 2011
- 5 Sekar V, Ratnasamy S, Reiter M, et al. The middlebox manifesto: enabling innovation in middlebox deployment. Proceedings of Association for Computing Machinery's Special Interest Group on Data Communications, Toronto, Canada, 2011
- 6 Sherry J, Hasan S, Scott C, et al. Making middleboxes someone else's problem: network processing as a cloud service. Proceedings of Association for Computing Machinery's Special Interest Group on Data Communications, Helsinki, Finland, 2012
- 7 Gember A, Krishnamurthy A, John S S, et al. Stratos: A Network-Aware Orchestration Layer for Virtual Middleboxes in Clouds. TechReport, 2014

杂度。与 FlowTags 相比,本文所提方法不需要控制器多次下发流表完成为数据分组增加标签后的转发工作,从而从总体上减少了控制器部署服务链下发流表的时间。

- 8 Qazi Z A, Tu C C, Chiang L, *et al.* SIMPLE-fying middlebox policy enforcement using SDN. Proceedings of Association for Computing Machinery's Special Interest Group on Data Communications, Hong Kong, China, 2013
- 9 Zhang Y, Behe N, Beliveau L, *et al.* StEERING: a software-defined networking for inline service chaining. Proceedings of the 21st IEEE International Conference on Network Protocols (ICNP 2013), Gottingen, Germany, 2013
- 10 Seyed K F, Chiang L, Sekar V. Enforcing network-wide policies in the presence of dynamic middlebox actions using flow tags. Proceedings of Networked System Design and Implementation, Seattle, USA, 2014
- 11 Halpern J, Pignataro C. Service Function Chaining (SFC) Architecture. Internet-Draft draft-ietf-sfc-architecture-07, IETF Secretariat, 2015
- 12 Quinn P, Nadeau T. Service Function Chaining Problem Statement. Internet-Draft draft-ietf-sfc-problem-statement-13, IETF Secretariat, 2015
- 13 OpenFlow 1.1, OpenFlow switch specification 1.1.0. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/OpenFlow/OpenFlow-spec-v1.1.0.pdf>, 2011
- 14 OpenFlow 1.0, OpenFlow switch specification 1.0.0. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/OpenFlow/OpenFlow-spec-v1.0.0.pdf>, 2009
- 15 Sekar V, Egi N, Ratnasamy S, *et al.* Design and implementation of a consolidated middlebox architecture. Proceedings of Networked System Design and Implementation, San Jose, USA, 2012
- 16 NetFPGA-10G project. <https://github.com/NetFPGA/NetFPGA-public/wiki>, 2014
- 17 OpenFlow 1.3, OpenFlow switch specification 1.3.0. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/OpenFlow/OpenFlow-spec-v1.3.0.pdf>, 2012

18 Ryu controller project. <http://osrg.github.io/Ryu>, 2014

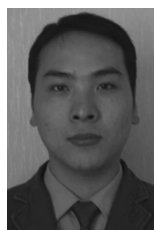
[作者简介]



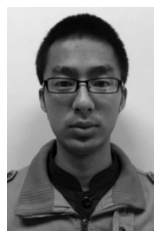
刘释然,男,国家数字交换系统工程技术研究中心硕士生,主要研究方向为新型网络体系结构、网络信息安全。



兰巨龙,男,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,主要从事新一代信息网络关键理论与技术的研究工作,目前作为首席科学家主持国家“973”计划项目“可重构信息通信基础网络体系研究”。



胡宇翔,男,博士,国家数字交换系统工程技术研究中心讲师,主要研究方向为新型网络体系结构、网络信息安全。



段通,男,国家数字交换系统工程技术研究中心硕士生,主要研究方向为新型网络体系结构、可编程网络数据平面。