中国科学: 信息科学 2018 年 第 48 卷 第 9 期: 1242–1256
SCIENTIA SINICA Informationis
论文

距离矢量路由混合 SDN 网络的一致性更新

于倡和[1], 兰巨龙[1], 胡宇翔[1*], 张彦斐[2], 王浩[2]

1. 国家数字交换系统工程技术研究中心, 郑州 450002
2. 河北特种设备监督检验研究院承德分院, 承德 067000
* 通信作者. E-mail: chxachxa@126.com

收稿日期: 2018–01–26; 接受日期: 2018–08–02; 网络出版日期: 2018–09–07

国家网络空间安全专项课题 (批准号: 2017YFB0803204)、国家高技术研究发展计划 (863 计划) (批准号: 2015AA016102) 和国家自然科学基金群体创新项目 (批准号: 61521003) 资助

摘要 混合 SDN 网络作为 SDN 网络与传统网络的过渡形态, 在保留传统网络服务的同时, 兼有部分软件定义网络易于管理维护的特点, 但混合 SDN 网络的更新问题一直没能找到有效的解决方法, 特别是当距离矢量路由参与混合的情况. 本文将分段路由技术引入混合 SDN 网络更新, 提出了一种分段路由在混合 SDN 网络更新的应用方案, 结合顺序更新与两步复制机制, 设计 LFCA 算法. LFCA 算法首先尝试用分段路由拼接最终路径并将最终路径信息封装入数据包包头. 而对于不可拼接的数据流, LFCA 算法构建了顾及更新远端效应的动态拓扑集并基于回溯原理来评估每一步更新操作, 通过不断回溯比较来最终找到最长一致性更新序列, 对于仍未完成更新的节点利用两步复制机制完成更新. 经过实验验证, 与之前算法相比, LFCA 算法可以极大地节省算法时间开销与存储资源开销.

关键词 分段路由, 距离矢量路由, 混合 SDN 网络, 一致性

1 引言

软件定义网络 (software-defined networking, SDN) 可以极大简化网络管理运维过程, 为运营商带来灵活性与细粒度控制. 但出于现实与经济考量, SDN 网络的部署是个增量过程, 所以在过渡期间, 运营商不得不维护管理混合 SDN 网络. 混合 SDN 网络是指网络中同时存在 SDN 控制器与传统路由协议如 (OSPF, IS-IS) 的网络, 并且之前的研究表明 [1], 混合 SDN 网络可以提供传统网络的服务, 如 MPLS, VPN 等, 而且还允许网络管理者通过 BGP 等协议来与其他网络进行通信, 并且实现一些 SDN 网络的优势, 如易于管理等. 然而, 因为混合 SDN 网络多控制平面 (SDN 控制器、路由协议) 共存, 而不同控制平面对于转发表 (forwarding information base, FIB) 变化的反应机制不同, 所以混合 SDN 网络的更新机制复杂. 在之前研究的技术中, 极少能直接适用于混合 SDN 网络更新. 网络更新是网络运

引用格式: 于倡和, 兰巨龙, 胡宇翔, 等. 距离矢量路由混合 SDN 网络的一致性更新. 中国科学: 信息科学, 2018, 48: 1242–1256, doi: 10.1360/N112018-00021
Yu C H, Lan J L, Hu Y X, et al. Consistent update of distance vector routing hybrid SDN networks (in Chinese). Sci Sin Inform, 2018, 48: 1242–1256, doi: 10.1360/N112018-00021

维的常见现象, 而通过网络更新, 混合 SDN 网络可以实现 SDN 节点的增量部署、流量工程、错误恢复等操作, 具有极大现实意义. 网络更新的实质在于调整节点的 FIB, 一个更新过程便是将初始的 FIB 用新 FIB 替代的过程, 而一致性, 便是保证每个流的转发要么采用初始规则, 要么采用最终规则, 不存在其他的可能. 一致性可以保证网络在更新过程中正常运行, 避免异常与错误的产生, 是网络安全更新的有力保证. 为了实现一致性更新, 前人已经做了很多研究, 一般来说, 之前研究提出的网络更新技术可以分为 3 类. 第 1 类称之为两步复制 (two-phase commit) [2,3], 这类更新技术核心在于各个节点同时维护初始规则与最终规则, 通过给流安装不同标签的方式来决定采取初始规则或者最终规则. 因为网络资源的限制, 如异常昂贵且耗电的三态内容寻址储存器 (ternary content addressable memory, TCAM) [4], 这种方法一般来说并不实际. 第 2 类方法称之为顺序更新 (ordered scheduling) [5~9], 这类方法的主要思想是按照特定顺序来更新网络节点. 通过设置一定的限制条件, 求出特定的节点更新顺序, 按照该顺序更新节点. 虽然通过这种方法实现的节点更新成功避免了网络资源的额外开销, 但这种方法一般致力于保证一种弱一致性, 如网络无环、无拥塞、无黑洞等. 这个分类中部分方法可以保证一致性 [9], 但是却有着明显缺陷, 即保证强一致性的更新序列未必存在. 第 3 类更新方法是综合利用多种机制来进行网络更新, 文献 [10] 提出了协同利用分段路由与 two-phase commit 机制来实现快速更新, 但其只适用于纯 SDN 网络. 文献 [11] 中提出的算法协同利用了 ordered scheduling 与 two-phase commit 技术, 既有效减少了网络资源的冗余消耗, 又避免了顺序更新面临的序列不存在问题, 适用于包括混合 SDN 网络在内的各种网络形态与路由协议, 有效填补了技术空白. 但其仍受限于大量的冗余资源消耗, 特别是当顺序更新机制求得的最长更新序列长度不够或不存在时, 算法仍然需要大量消耗 TCAM 以完成一致性更新, 并且, 其在距离矢量路由 (distance vector routing, DV) 参与混合的情况下所提出的算法时间代价过大, 实际情况下难以应用.

本文在分析混合 SDN 网络路由特性的基础上, 探究了距离矢量路由参与的混合 SDN 网络的更新机制, 提出了 LFCA (lightweight fast collaborative update algorithm) 算法, 将分段路由 (segment routing, SR) 技术引入其更新过程, 并提出了一种 SR 在该种混合 SDN 网络更新中的应用方案: 根据新路径是否可由现有规则来拼接成而分为可拼接流与不可拼接流, 对于可拼接流, 算法将新路由信息逐跳封装入数据包包头, 借助包头标签完成转发; 而对于不可拼接流, 算法基于节点顺序更新机制, 通过构建动态拓扑集与回溯机制来评估每一步更新操作, 最终找到最长一致性更新序列, 而对于仍未完成更新的节点利用 two-phase commit 机制完成更新. 经过实验验证, LFCA 算法比之前算法有效节省了 TCAM 资源与时间成本.

2 混合 SDN 网络建模与性质

本节主要介绍混合 SDN 网络的性质, 并深入讨论混合 SDN 网络在多控制平面共存时的路由机制, 特别是当距离矢量路由参与混合时的特性, 并分析讨论混合 SDN 网络这些性质对之前技术的影响.

2.1 混合 SDN 网络的模型

图 1 展示的是混合 SDN 网络的结构, 其中, 圆圈代表网络节点, 边代表物理连接, 并标明了 IGP 协议下的权值. 之前的研究表明 [12], 混合 SDN 网络中, 控制器可以给全网所有的节点下发流表, 包括只支持 IGP 的节点, 常见的技术包括往 IGP 协议中通告信息、安装静态路由 [13] 等. 在图 1 中, IGP

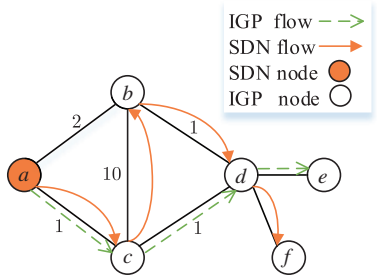

图 1 (网络版彩图) 混合 SDN 网络的实例

Figure 1 (Color online) Example of hybrid SDN networks

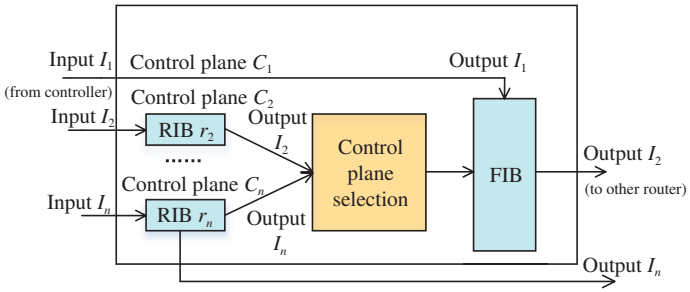

图 2 (网络版彩图) 混合 SDN 网络控制平面机制

Figure 2 (Color online) The control plane mechanism of hybrid SDN networks

协议下的数据流沿最短路径传送到节点 e, 而 SDN 控制器控制的数据流却沿着不同的路径传送到终点 f. 通过更新, 我们可以实现 SDN 节点的部署或删除、流量调度等一系列功能.

2.2 混合 SDN 网络的控制平面机制

当网络中存在多个控制平面时, 其处理机制如图 2 所示, 除 SDN 控制器外, 每个路由协议都分布式的维护各自的路由信息表, 并通过本地的控制平面选择机制来决定将哪个控制平面的路由信息写入 FIB 表. 对于 IGP 协议, 可以通过设置不同的管理位距 (administrative distance, AD) 值来确定最优的路由协议, 而对于控制器, 则可以通过流表下发来直接更改 FIB. 关于直连路由, 控制平面的选择过程跳过, 可以认为所有的控制平面同时被启用.

2.2.1 混合 SDN 网络控制平面的性质

由于不同控制平面的特性不同, 所以, 混合网络的整体控制层性质较为复杂. 文献 [12] 根据控制平面是否用 FIB 表信息作为路由转播的输入信息, 将控制平面分为 FU (FIB-unaware) 与 FA (FIB-aware). 如果 FIB 信息被用来传播路由信息, 则为 FA, 如图 2 中的控制平面 C_2, 反之为 FU, 如 C_1 与 C_n. 常见的 FU 控制平面有链路状态路由协议、SDN 控制器等, 而常见的 FA 控制平面有距离矢量路由协议, 如 EIGRP.

对于 FU 控制平面, 其路由的传播与 FIB 信息表无关, 而仅仅与 RIB 表内容相关, 所以节点的 FIB 表改变, 只有本地效果, 对于其余节点的路由信息没有影响. 如图 3 所示, 假设控制平面为 FU 控制平面, 节点 u 到 d 的路径为 $\{u, v, w, d\}$, 即使节点 v 的到 d 的 FIB 信息被其余控制平面更改, 也不

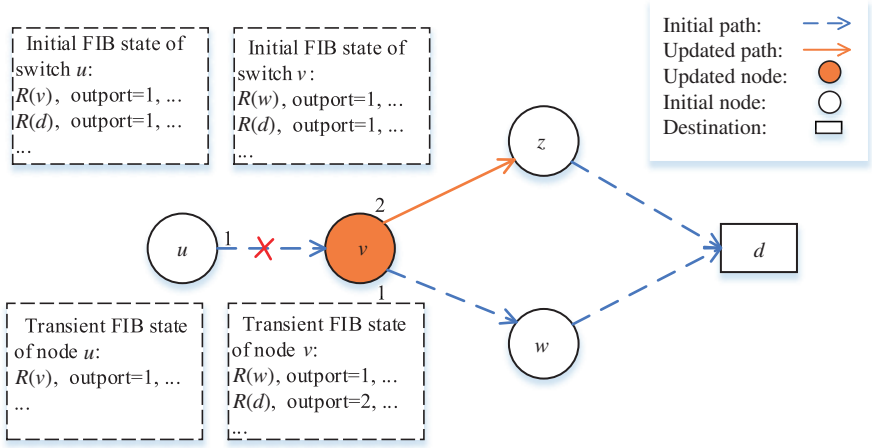

图 3 (网络版彩图) 不同控制平面性质对更新的影响

Figure 3 (Color online) The influence of different control plane properties in update

会对 u 的 FIB 产生影响. 所以有性质 1.

性质1 FU 控制平面下, 节点的 FIB 独立于其余所有共存的控制平面.

因为 FA 控制平面需要用 FIB 信息来作为路由传播时的输入信息, FIB 的变化会对节点的路由信息产生影响. 如图 3 所示, 如果网络所配置的控制平面是 FA 控制平面, 如图 2 中 C_2, 则改变节点 v 到节点 d 的 FIB 条目会导致控制平面改变节点 u 到节点 d 的路由表条目. 而在更新过程中, 如果其他控制平面到节点 d 的路由信息安装到节点 v 的 FIB 表中, 例如下一跳路径由 $\{v, w\}$ 更新为 $\{v, z\}$, 节点 v 就会停止向周围节点传播控制平面 C_2 安装的到节点 d 的路由, 则节点 u 无法获取到节点 d 的路由信息, 其到 d 的路由会被撤销. 所以说, 在 FA 控制平面下进行网络更新, 可能不仅有本地效应, 还会有远端效应 (如节点 u). 这对 FA 控制平面的混合网络更新带来了极大的困难. 概括地讲, 在 FA 控制平面下, 路由器只会传播用于计算 FIB 规则的路由, 所以对于 FA 控制平面, 有性质 2.

性质2 FA 控制平面下, 在目的节点为 d 的路径中, 所有节点到目的节点的 FIB 规则都是由同一控制平面安装的.

有兴趣的读者可以参考文献 [12] 来获得关于混合 SDN 网络控制平面性质更详细与深入的描述与讨论.

2.2.2 混合 SDN 网络性质对之前技术的影响

对于 FU 控制平面的混合 SDN 网络, 文献 [11] 已经证明之前提出的技术原理可以在 SDN, IGP 与混合 SDN 网络之间交互使用. 而对于 FA 控制平面的混合 SDN 网络, 因之前提出的技术大多没有考虑更新时候的远端效应, 故绝大部分不适用于距离矢量路由参与的混合 SDN 网络. 所以, 关于 DV 混合 SDN 网络的更新问题仍是一个挑战.

2.3 分段路由在 DV 混合 SDN 网络中的应用

分段路由是一种源路由技术, 可以将路径分割为若干路径段, 并通过包头添加的段标识符 (segment identifier, SID) 来指导数据包沿着特定路径段转发. 所以, 如果给包头安装 SID 来引导流沿着现有的路径段拼接成的新路径到达指定的目的节点, 也可以达到更新后的路径效果, 这也是利用分段路由机

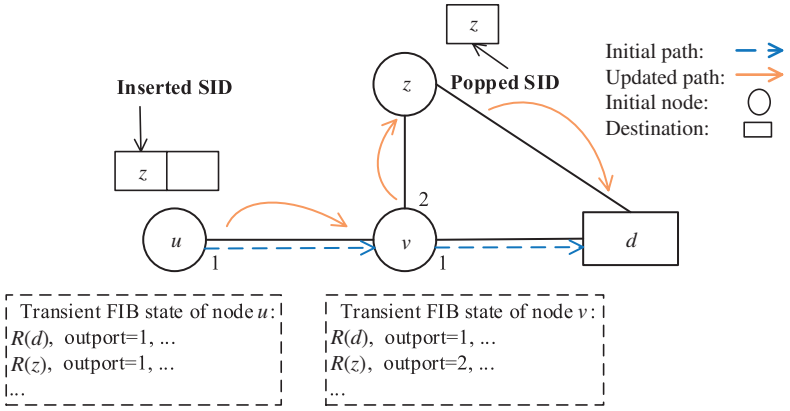

图 4 (网络版彩图) 分段路由在更新中的应用

Figure 4 (Color online) Application of the segment routing in network update

制来完成网络更新的一般原理. 如图 4 所示, 在控制平面为 FU 的条件下, 数据流 $f_{u,d}$ 初始路径为 $\{u,\ v,\ d\}$, 最终路径为 $\{u,\ v,\ z,\ d\}$, 因为路径 $P_{u,z}$ 即 $\{u,v,z\}$ 存在, 所以对于该流, 拼接路径段为 $\langle P_{u,z}, P_{z,d}\rangle$, 所以可以添加 SID $\langle z\rangle$ 到相应数据包包头, 使得数据包先沿着路径 $P_{u,z}$ 转发到 z, 然后沿着 $\{z,d\}$ 转发到终点 d. 但是在 DV 混合 SDN 网络的情景下, 由于更新时的远端效应, 在更新过程中可能会有用于拼接的现有路径被撤销, 导致数据流无法完成转发而大量丢包. 为了在 DV 混合 SDN 网络中有效使用分段路由机制, 本文提出了一种全新的分段路由应用机制.

2.3.1 应用机制

分段路由在流量工程 [14] 或网络测量 [15] 等领域内的应用中, 大多可视为一种松散源路由技术, 单个 SID 便可指导数据包沿着多跳路径转发, 如图 4 中 SID $\langle z\rangle$ 可指导数据包沿路径 $P_{u,z}$ 转发, 但由于更新时的远端效应, 多跳路由极可能在更新过程中被撤销, 所以这种机制在 DV 混合 SDN 网络中并不可靠. 为了应对更新中的远端效应, 本文将用于拼接最终路径的路径段进行分类, 如被选中的路径段在更新过程中无需变动, 则该路径可以用松散源路由的形式进行表示, 即单个 SID 封装该路径信息; 而如果路径段在更新过程中需要更新, 则该路径的信息需用严格源路由的形式逐跳封装, 利用直连路由来拼接最终路径, 如图 4 中为引导数据流沿路径 $P_{u,z}$ 转发而加装 SID $\langle u,v,z\rangle$.

2.3.2 SR 的技术优势与局限

分段路由更新网络的过程较为简单, 可以抽象为 3 个步骤: (1) 在入口节点处安装分段路由规则, 启用分段路由, 相应数据包可以在入口节点处加装对应的 SID 序列; (2) 拆除旧有规则, 安装新规则; (3) 停用分段路由, 启用新规则. 使用分段路由更新网络可以使得数据包立即沿着最终路径传播, 实时性极好, 且除入口节点外, 其余节点不需要额外的 TCAM 资源消耗. 但并不是所有的数据流的最终路径都可以用现有路径拼接而成, 并且数据包加装 SID 会为网络带来额外的流量负担. 所以, 如何判断数据流是否可拼接与找到 SID 最短的拼接方式是利用分段路由进行更新的核心与难点.

3 更新算法设计

网络更新过程的关键在于逐步更改网络中节点的配置直到网络中的所有节点都已经运行其最终

配置. 而在网络更新过程中节点可以进行的操作有: (1) 更改路由协议的管理位距, 即改变节点所采用的路由协议; (2) 安装静态路由; (3) 通过 SDN 机制安装或者删除 FIB 表中的路由规则; (4) 给数据包添加 SID、标签等一系列操作. 这些操作都可以针对特定目的节点. 然而, 由于现实中同时完成所有的更新操作是不切实际的, 所以更新过程是一个增量过程, 需要经过一系列中间态. 而如何保证更新过程的一致性, 维护网络正常功能, 避免环路、路由黑洞等异常的发生是网络安全更新的保证, 也是本文算法的关键目标. 即如何找到合适的操作顺序来避免更新过程中的不一致性是网络更新的关键, 所以我们的问题可以表述如下.

问题 在已知网络的初始配置与最终配置的前提下, 计算一个操作序列, 按照该序列更新网络不会造成任何的不一致.

值得注意的是, 这里一般假设初始配置与最终配置都是无异常的正确配置, 并且, 操作序列中的每一步中可能含有多个可以并行执行的更新操作, 但同一步中的更新操作的执行先后顺序并不影响网络一致性.

为了实现 DV 混合 SDN 网络的一致性更新, 本文提出一种协同利用分段路由、顺序更新与 two-phase commit 机制的更新算法 LFCA, 算法更新过程由 3 部分组成: LFCA 首先启用 LUSR (lightweight update algorithm with segment routing) 算法, 借助分段路由快速更新网络中的可拼接数据流; 而对于不可拼接数据流, 算法利用回溯的机制, 利用 IBUA (iteration and backtracking update algorithm) 来求得最长一致性更新序列, 按照该序列依次剩余节点; 如果仍然有节点未更新, 则利用 two-phase commit 来完成更新. 最后, 本文的更新算法发起都是集中在控制器内, 根据网络的初始配置与最终配置, 通过比较可以很容易地明确待更新的源目的节点对.

3.1 利用 SR 更新的 LUSR 算法

分段路由机制用于网络更新的性能与网络中可拼接的数据流比例有关, 算法设计的核心在于如何判断网络中待更新数据流是否可拼接与为可拼接数据流找到最短的 SID 序列, 并且, 由于现有路径段的拼接方式是指数量级的, 所以简单的枚举算法并不适用. 这里, LUSR 算法将路径段的分配组合问题转化为 0-1 整数线性规划. 并且, 因为各流的拼接路径段的分配组合彼此独立, 所以算法采用分治思想将问题转化为 N 个子问题分别求解, 即在逐流粒度上建模, 所建模型如果有解则数据流可拼接, 而所求解便是该流在当前条件下最优的路径段拼接方式. 其核心机制如算法 1.

根据初始配置与最终配置, 算法可以计算出需要更新的源目的节点对 (行 2), 随后遍历其中的待更新源目的节点对, 分别求解对应流的 SID 序列. 对于单个待更新数据流, 算法利用 extract 函数来计算其所有可能的拼接段, 函数机制在于将待更新数据流的最终路径所有可能的拼接段依次在当前路径集中通过查表查找. 如果存在, 则将其添加到候选拼接段集合, 在得到数据流的候选拼接段集合后, 算法建立 ILP 模型, 计算用 num 个路径段能否拼接出流最终路径, 若模型无解则将 num 值加 1 并重新建模计算, 直到 num 值超出范围或者拼接集已求出 (行 4∼11). 如果所求拼接集为空, 则该流不可拼接, 并记录对应源目的节点对 (行 22∼24); 若拼接集为非空, 则检测集合中的路径段是否需要在更新过程中更新, 对于需要更新的路径段, 算法逐跳提取 SID, 反之, 只需将路径段的末尾节点作为 SID, 最后更新当前网络路径信息 (行 12∼22).

对于函数 ILP, 是一个 0-1 ILP 建模并求解的过程, 建模的优化目标与约束条件见式 (1)∼(4):

$$\max \ \ \mathrm{metric}_{s,d} \tag{1}$$

Algorithm 1: LUSR: 利用分段路由机制更新网络

Input: 节点集合 N, 初始网络配置 P_i, 最终网络配置 P_f, 目的节点集 D.
Output: SID 序列集 SL, 当前路径集 P_c, 未更新源目的节点对 unsafe.

```
SL={};
nodepair=compute_changing_node(P_i, P_f, D) # 计算待更新结点对;
for (s,d) in nodepair do
    SL_{s,d}={};
    C=extract(s, d, P_i, P_f) # 候选路径段集合;
    num=2 # 所选拼接段数量;
    L=length(P_{s,d}) # 最终路径的跳数;
    while segment=∅ and num<L do
        segment=ILP(s, d, num, C, P_f) # 求解拼接段集合;
        num=num+1;
    end
    if segment≠∅ then
        for seg in segment do
            if seg in (P_i(s,d) for (s,d) in nodepair) then
                SL_{s,d}=modify(seg, SL_{s,d}) # 逐跳提取 SID 序列;
            else
                SL_{s,d}=fetch(S) # 路径段最后一跳添入 SID 序列;
            end
        end
        SL.append(SL_{s,d});
        P_c=update_network(s, d, P_i, P_f);
    else
        unsafe.append(s, d);
    end
end
return SL, P_c, unsafepair.
```

$$\text{s.t.}\ \ \forall \text{seg}_{x,y} \in C,\ \forall e \in P^f_{s,d} : \sum_{\text{seg}_{x,y} \in C} a_e^{x,y} \times w_{s,d}^{x,y} \leqslant 1, \tag{2}$$

$$\sum_{\text{seg}_{x,y} \in C} w_{s,d}^{x,y} \leqslant \text{num}, \tag{3}$$

$$L_{s,d} \times \text{metric}_{s,d} = L^*_{s,d}, \tag{4}$$

其中, 候选拼接段集为 C. 优化目标是让布尔变量 $\text{metric}_{s,d}$ 取得最大值, 该变量代表流 $f_{s,d}$ 是否可拼接. 在约束条件中, 布尔变量 $w_{s,d}^{x,y}$ 代表拼接段 $\text{seg}_{x,y}$ 是否被用于拼接流 $f_{s,d}$ 的新路径中, 布尔变量 $a_e^{x,y}$ 代表流 $f_{s,d}$ 最终路径中的任意边 e 是否在拼接段 $\text{seg}_{x,y}$ 中. 式 (2) 约束拼接路径中必须含有最终路径中的所有边. 约束 (3) 是关于拼接段的数目, 被选取的拼接段数目必须小于等于本次建模的拼接段数目上界 num. $L_{s,d}$ 与 $L^*_{s,d}$ 代表流 $f_{s,d}$ 最终路径的跳数与拼接成的流路径的跳数. 显然, 式 (4) 是为了保证拼接路径与流最终路径的长度相同.

3.2 利用回溯机制更新的 IBUA 算法

IBUA 主要用来更新 LUSR 无法一致性更新的数据流. 而由 2.2.1 小节中论述可知, FA 节点的 FIB 被其他控制平面更改, 会造成远端效应, 但这在 DV 混合 SDN 网络更新中是不可避免的, 所以

DV 混合 SDN 网络更新算法的设计关键在于如何处理更新操作所带来的远端效应. 本文建立顾及远端效应的动态路径集, 通过回溯机制来探索最长一致性更新序列. 算法在待更新节点集中随机挑选某节点, 假设该节点已经被更新, 在更新后拓扑递归调用求解函数本身, 并对返回值进行判断: 若返回的更新序列中含有所有剩余的待更新节点, 则求得当前状态下最长一致性序列, 并返回该序列; 若返回序列不能更新所有剩余待更新节点, 则将该序列与当前记录的最长一致性更新序列进行比较, 记录二者中的最长序列, 并从待更新节点中重新选取一个假设更新节点, 重新进行递归调用, 即对待更新节点进行遍历, 以确保最后返回的序列为最长一致性更新序列. 总之, 算法依靠迭代确保本次递归的返回值为其求解状态下的最长一致性更新序列, 依靠层层递归来最终求得初始状态的最长一致性更新序列. 算法 2 为 IBUA 算法的核心架构.

Algorithm 2: IBUA: 回溯的最长一致性序列更新算法

Input: 节点集合 N, 初始网络配置 P_i, 最终网络配置 P_f, 目的节点集 D, 未更新源目的节点对 unsafe.
Output: 最长一致性更新序列 seq, 当前路径集 P_c, 未更新源目的节点对 unsafe.

```
1  function calculate (N, P_i, P_f, D, unsafe);
2  U=compute_node(N, unsafe) # 已更新结点;
3  F=set(N).difference(U), C=F # 未更新结点, 即待更新节点;
4  new_candidate=set(C);
5  for x in C do
6      P_c=build_current_path(N, P_i, P_f, D, U∪{x});
7      if not check_consistency (N, P_i, P_f, D, P_c) then
8          new_candidate.remove (x);
9      end
10 end
11 C=new_candidates;
12 max_seq=[ ] # 当前最长更新序列;
13 for x in C do
14     (seq, cur_unsafe, cp)=assess(x, N, P_c, P_f, D, unsafe);
15     if len(seq)==len(F) then
16         max_seq=seq;
17         break
18     end
19     if len(seq)>len(max_seq) then
20         max_seq=seq;
21     end
22 end
23 P_c=build_current_path(x, N, P_i, P_f, D, U∪ {max_seq});
24 unsafe=store_unsafe(unsafe, max_seq);
25 return(max_seq, unsafe, cur);
26 function assess(x, N, P_i, P_f, D, unsafe);
27 new_set=remove_node(x, unsafe);
28 (tail, a, b)=calculate(N, P_i, P_f, D, new_set);
29 seq=[x]+tail;
30 return (seq).
```

变量初始化完成后 (行 2∼4), 算法首先在未更新节点集内逐节点检测更新一致性. 随后从候选节点集合中选取一个节点, 利用函数 build_current_path 来计算在该节点已被更新条件下各个节点到目

的节点的路径集, 函数从当前拓扑中提取所有的 FA 节点以及目的节点来构成一个子图即 FA 域, 剩余节点为 FU 节点. 随后, 计算 FA 节点在 FA 域到目的节点最短路径与 FU 节点在完整拓扑上到目的节点的路径. 如果 FU 节点到目的节点的路径中含有在 FA 域内的节点, 则后续路径由该路径首个 FA 节点在 FA 域内求得的到目的节点的路径替代, 得到更新后各节点到目的节点的路径 (行 5, 6). 随后, 利用函数 check_consistency 来检测更新该节点是否将造成不一致性, 即函数比较节点更新后各节点到目的节点的路径是否有既不是初始路径又不是最终路径的情况, 若产生了不一致, 则说明该节点当前无法被一致性更新, 从集合中删除该节点, 最终可以得到一个一致性更新候选节点集合 (行 6~11). 接下来, 算法利用回溯来探索最长一致性更新序列 (行 12~29). 首先从候选节点中随机挑选一个节点 x, 并调用 assess 函数, 该函数首先假设被选取节点已被更新, 并重新调用 calculate 函数来计算最长一致性更新序列 (行 27~29). 而 15~21 行是回溯的过程, 算法通过回溯, 验证在之前所选择的假设更新节点条件下, 求得的一致性更新序列是否为最优解. 如果序列包含所有未更新节点, 则之前假设节点为最优解, 则停止迭代, 递归返回所求信息, 算法再对上一轮迭代的选择进行回溯评估 (行 15~18). 而如果没有包含所有待更新节点, 则与当前记录的最长更新序列进行比较, 并从候选节点集合中选择新的假设更新节点, 再进行递归求解, 以保证算法返回的序列为当前条件下的最长一致性更新序列 (行 19~21), 通过多次递归, 最终返回的便是初始状态下的最长一致性更新序列. 值得注意的是, 对于不同 FA 控制平面之间的更新, 其机制与 FA 与 FU 之间的更新类似, 而差异在于计算当前路径时节点的划分, 网络节点被分为最终 FA 节点与初始 FA 节点而非 FA 节点与 FU 节点, 其余更新机制类似, 亦可以启用 IBUA 算法原理更新. 值得注意的是, 如果两轮更新后全网还有未更新的节点, 则算法启用 two-phase commit 机制来完成最后的更新, 因前文对该机制有过介绍, 这里不再赘叙.

3.3 算法性质

本文提出的 LFCA 算法是协同利用分段路由、顺序更新、two-phase commit 机制来更新网络, 算法具有以下 4 条性质. 且附录中给出了性质 (1)~(3) 的详细证明, 而性质 (4) 在实验评估部分有详细说明.

(1) **LUSR 算法可以保证数据流更新期间的一致性.** 即 LFCA 算法在启用分段路由更新网络期间, 可以维护一致性.

(2) **IBUA 算法可以求解出最长的一致性节点更新序列并在更新期间保证一致性.** 即 LFCA 算法可以通过顺序更新机制更新尽可能多的节点, 并且在更新期间保持网络一致性.

(3) **LFCA 算法可以完成一致性更新.** 即 LFCA 算法可以一致性地完成网络更新, 具有正确性与有效性.

(4) **与同类算法相比, LFCA 算法可以高效快速地更新网络.** 与同类算法相比, LFCA 在相同问题规模下可以极大地削减算法的运行时间.

4 实验评估

本节采取实际拓扑作为数据集来进行实验, 并与当前主流及最新技术进行比较, 从而验证算法有效性. 其中 4.1 小节简要介绍实验方法, 4.2 小节就实验结果进行分析比较.

表 1 实验拓扑信息

Table 1 The information of experimental topologies

Topology	AS1755	AS3967	AS3257	AS6164
Nodes	87	79	161	138
Links	322	297	656	418

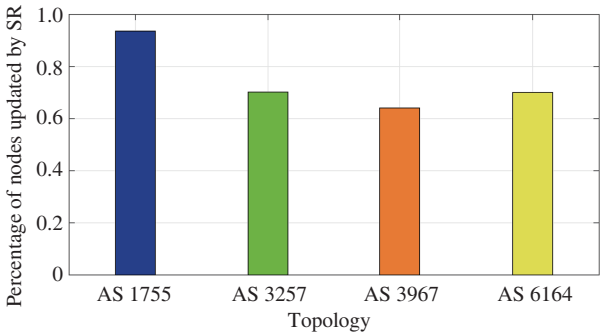

图 5 (网络版彩图) 分段路由机制完成节点更新比例

Figure 5 (Color online) Percentage of nodes updated by SR mechanism

4.1 实验方案

本文采用开源拓扑集 Rocketfuel[16] 作为实验的数据集, 所涉及的数据拓扑有 AS{3967, 1755, 3257, 6164}, 对于每个拓扑, 随机选取两个节点作为目的节点, 依次计算拓扑中所有节点到目的节点的最短路径作为初始状态. 随后, 随机选取 50% 的链路重新设置权值, 所重置权值在初始拓扑链路权值中随机选取, 并在重置后的拓扑上重新计算各节点到目的节点的最短路径, 并作为最终状态. 拓扑的具体信息见表 1.

在我们所知范围内, 只有 GPIA+FED 算法[11] 与 two-phase commit 机制[2] 可用于 DV 混合 SDN 网络更新. 为了验证算法有效性, 本文将算法在相同实验条件下与它们进行比较, 并重复实验 200 次来保证数据可靠性. 本文算法运行的环境为 Ubuntu 16.04 系统, 16 GB DDR3 内存, Intel Xeon E5-2407 CPU 处理器.

4.2 实验结果

在本文中, 进行的实验主要比较 LFCA 算法与之前算法在 TCAM 资源节省, 算法时间成本等方面的性能. 而对于 LFCA 算法本身, 分段路由机制对于网络更新的可用性, 引入的 SID 序列长度分布也是算法有效性的重要测度.

4.2.1 分段路由机制对于网络更新的可用性

本文定义 SR 节点更新比例 N_f/N 来测度分段路由机制对于网络更新的可用性, 其中 N_f 为被 LUSR 算法完成更新的节点数目, 而 N 为更新前全网待更新节点数, 这里计算各拓扑在 200 次重复实验中的 SR 节点更新比例的平均值作为实验结果并绘制图 5.

由实验结果可以看出, 分段路由机制对于各拓扑都有良好的适用性, 平均至少可以更新 64.09% 的待更新节点, 而对于拓扑 AS1755, 则可以完成 92% 的待更新节点的更新, 说明分段路由机制可以极好

地用于网络更新并缩小待更新节点规模. 值得注意的是, 规模大且复杂的拓扑 AS3257 的 SR 节点更新比例高于 AS3967, AS6164 的 SR 节点更新比例高于 AS3967, 说明分段路由更新机制的性能不随着网络规模的扩大而单调恶化, 对网络规模扩张有良好的适用性与可扩展性.

4.2.2 TCAM 资源节省

本文采用 two-phase commit 机制消耗的 TCAM 作为基准, 以其余算法相对于 two-phase commit 机制的 TCAM 资源节省比例作为算法 TCAM 资源节省性能的一个重要测度, 定义该比例为 $(T_{\text{tpc}} - T_x)/T_{\text{tpc}}$. 其中, T_{tpc} 为 two-phase commit 机制额外消耗的 TCAM 条目, 而 T_x 为相应算法额外消耗的 TCAM 条目. 实验结果如图 6 所示. 实验结果以互补累计分布函数 (complementary cumulative distribution function, CCDF) 的形式展示, 图中曲线的任意点 (x, y) 代表相应算法相对 two-phase commit 机制, 在 $y \times 100\%$ 的实验中至少节省 $x \times 100\%$ 的规则条目. 实验结果表明, 相较于其他算法, LFCA 在整体上可以实现更大比例的 TCAM 资源节省, 特别是在节省的最小值上. 如在图 6(a) 的拓扑 AS1755 中, LFCA 至少可以削减 83% 的 TCAM 冗余, 相较于 GPIA+FED 提升了近 60%. 在节省比例相同时, LFCA 的达标实验次数所占比例一般都高于 GPIA+FED, 说明在大部分情况下 LFCA 算法性能优于其余算法. 但很明显, LFCA 无法实现无冗余更新, 因为分段路由机制本身需要在入口节点加装相应规则, 这是机制本身的开销. 而 GPIA+FED 在部分重复实验中可以实现无冗余更新, 但其余实验中依旧引入较大比例 TCAM 冗余开销. 这是由于 GPIA 本身计算的一致性更新序列的长度极不稳定, 若序列长度短或者所求序列不存在, 则剩余的大量未更新节点只能通过基于 two-phase commit 机制的 FED 算法由规则复制完成, 开销较大. LFCA 算法在启用 two-phase commit 机制之前, 可以通过分段路由与顺序更新两种机制来更新网络, 两种机制可以互为补充. 若单一更新机制更新效果不佳时, 另一机制尚能有效更新剩余节点, 以保证网络中的绝大部分节点在启用 two-phase commit 机制前得到更新. 所以说, LFCA 算法的性能更加稳定, 适用性更好.

4.2.3 算法运行时间

对于一个待更新节点为 n 的场景, LFCA 算法理论上的最坏时间复杂度为 $O(p(kn)(kn)!)$, 其中, k 表征分段路由机制削减的待更新节点百分比, $p(kn)$ 为多项式时间. LFCA 算法的复杂度主要由 IBUA 算法回溯过程产生的递归树决定, 在最坏情况下, 该递归树会枚举出所有可能的更新序列, 而每个枝干的探索求解过程为简单的贪心算法, 是多项式复杂度, 所以 IBUA 算法在最坏时间复杂度为 $O(p(n)n!)$. 而 LFCA 算法在执行 IBUA 回溯求解之前, 会启用 LUSR 算法利用分段路由机制来更新数据流, 减少待更新节点的数目, 缩减问题规模. 所以对于同一个待更新节点为 n 的场景, LFCA 算法的复杂度为 $O(p(kn)(kn)!)$.

对于同一场景, 因 GPIA+FED 算法在更新 DV 混合 SDN 网络更新时, 算法极易沦为穷举算法, 固其最坏算法时间复杂度为 $O(p(n)n!)$. 在 4.2.1 小节中, 分段路由机制的可用性与性能已经得到证实, 平均至少可以削减 64.09% 的待更新节点, 有效缩小了问题求解规模. 所以与同类算法相比, LFCA 算法可以极大削减算法运行时间. 下面通过实验验证了这一点.

为了对比算法的时间消耗, 这里采用不同拓扑下 200 次算法运行的平均时间作为测度来评估算法的时间成本, 对比结果如表 2 所示. 通过对比可以发现, LFCA 算法明显削减了算法的时间成本. 这是由于 LUSR 算法有效削减了待更新节点的数量 (见 4.2.1 小节, 64.09% 以上), 从而回溯过程最坏情况下产生的递归树的规模被极大削减, 故而在相同初始条件下, LFCA 算法可以很快收敛, 有效地缩减了

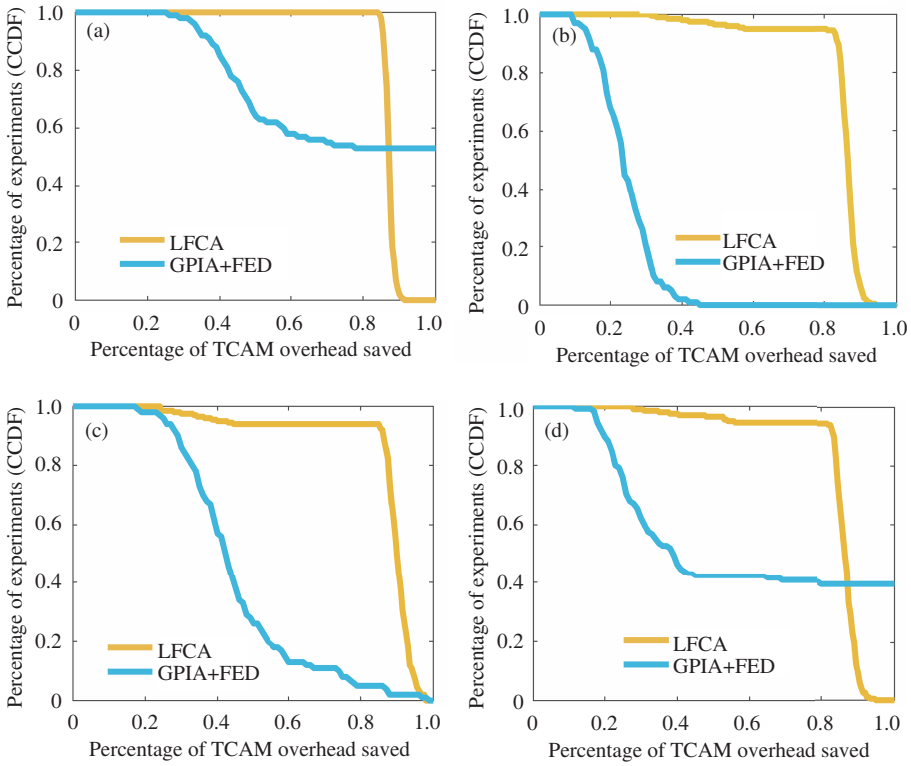

图 6 (网络版彩图) TCAM 条目节省比例

Figure 6 (Color online) Percentage of TCAM resources saved. (a) AS1755; (b) AS3257; (c) AS3967; (d) AS6164

表 2 算法运行时间

Table 2 The time cost of algorithm

Topology	AS1755	AS3967	AS3257	AS6164
LFCA (s)	1.094	4.430	15.709	6.322
GPIA+FED (s)	233.033	925.686	16597.636	8200.710

算法的时间开销, 增强了 DV 混合 SDN 网络更新算法的可用性与现实性.

4.2.4 算法 SID 的长度分布

数据包加装 SID 序列引入的额外带宽消耗也是 LFCA 算法的固有机制代价, 测度这种开销最直接的方法便是 SID 序列的长度分布. 我们计算各个拓扑的在 200 次实验中的 SID 序列长度的平均值, 统计序列长度的分布情况, 并绘制在表 3 中.

通过实验结果可以发现, 只需要一个 SID 便可实现至少 39.80% 的数据流的更新, 在序列长度小于等于 4 的情况下, 99% 的数据流已经可以完成更新, 完成全网流量更新所需 SID 序列长度最长为 6. 这表明, 在绝大多数情况下, 分段路由机制所引入的额外带宽压力较小, 对于网络来说是可以接受的.

表 3 SID 序列长度分布

Table 3 Distribution of SID list length

SL length	SL=1 (%)	SL⩽2 (%)	SL⩽3 (%)	SL⩽4 (%)	SL⩽5 (%)	SL⩽6 (%)
AS1755	39.80	77.10	93.44	99	99.94	100
AS3257	59.24	89.83	98.69	99.90	100	100
AS3967	63.28	93.40	99.44	99.98	100	100
AS6164	53.56	85.48	97.12	99.58	99.97	100

5 结论

本文在分析混合 SDN 网络的路由性质的基础上, 探究了 DV 混合 SDN 网络的更新机制, 提出了针对 DV 混合 SDN 网络更新的 LFCA 算法. 该算法首先利用分段路由迅速更新网络, 缩小待更新的数据流规模, 接下来利用回溯机制寻找最长一致性更新序列, 按照该序列更新网络节点, 如若依然有未更新节点, 则利用 two-phase commit 机制通过规则复制来完成更新. 并且, 通过实验验证, LFCA 算法有更优秀且稳定的算法性能, 与之前的混合网络更新算法相比, 有效地节省了 TCAM 资源, 极大地缩减了算法时间开销, 对于混合 SDN 网络维护管理有较大现实意义.

参考文献

1 Vissicchio S, Vanbever L, Bonaventure O. Opportunities and research challenges of hybrid software defined networks. SIGCOMM Comput Commun Rev, 2014, 44: 70–75

2 Reitblatt M, Foster N, Rexford J, et al. Abstractions for network update. SIGCOMM Comput Commun Rev, 2012, 42: 323–334

3 Katta N P, Rexford J, Walker D. Incremental consistent updates. In: Proceedings of ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, Hong Kong, 2013. 49–54

4 Sun P H, Lan J L, Wang P, et al. RFC: range feature code for TCAM-based packet classification. Comput Netw, 2017, 118: 54–61

5 Vissicchio S, Cittadini L. Safe, efficient, and robust SDN updates by combining rule replacements and additions. IEEE/ACM Trans Netw, 2017, 25: 3102–3115

6 Wang W, He W B, Su J S, et al. Cupid: congestion-free consistent data plane update in software defined networks. In: Proceedings of the 35th Annual IEEE International Conference on Computer Communications, San Francisco, 2016

7 Förster K T, Mahajan R, Wattenhofer R. Consistent updates in software defined networks: on dependencies, loop freedom, and blackholes. In: Proceedings of IFIP Networking Conference and Workshops, Vienna, 2016

8 Liu H H, Wu X, Zhang M, et al. zUpdate: updating data center networks with zero loss. SIGCOMM Comput Commun Rev, 2013, 43: 411–422

9 McClurg J, Hojjat H, Černý P, et al. Efficient synthesis of network updates. SIGPLAN Notice, 2015, 50: 196–207

10 Luo L, Yu H F, Luo S X, et al. Achieving fast and lightweight SDN updates with segment routing. In: Proceedings of IEEE Global Communications Conference, Washington, 2016

11 Vissicchio S, Vanbever L, Cittadini L, et al. Safe update of hybrid SDN networks. IEEE/ACM Trans Netw, 2017, 25: 1649–1662

12 Vissicchio S, Cittadini L, Bonaventure O, et al. On the co-existence of distributed and centralized routing controlplanes. In: Proceedings of the 34th Annual IEEE International Conference on Computer Communications, Hong Kong, 2015. 468–477

13 Vissicchio S, Tilmans O, Vanbever L, et al. Central control over distributed routing. SIGCOMM Comput Commun Rev, 2015, 45: 43–56

14 Moreno E, Beghelli A, Cugini F. Traffic engineering in segment routing networks. Comput Netw, 2017, 114: 23–31

15 Aubry F, Lebrun D, Vissicchio S, et al. SCMon: leveraging segment routing to improve network monitoring. In: Proceedings of the 35th Annual IEEE International Conference on Computer Communications, San Francisco, 2016

16 Spring N, Mahajan R, Wetherall D, et al. Measuring ISP topologies with rocketfuel. IEEE/ACM Trans Netw, 2004, 12: 2–16

附录 A

附录内容为对正文 3.3 小节中算法性质 (1)∼(3) 的证明.

符号注释. 记待更新数据流的集合为 C, 数据流源节点为 a, 目的节点为 b, 待更新数据流可以为网络中任意两点间的数据流, 记节点 x 在 t 时刻到目的节点 d 的下一跳节点为 $\mathrm{next}(x,d,t)$, 定义 $P(a,d,t)$ 为在时间 t 时由源节点 a 到目的节点 d 的路径, 其中 $P(a,d,0)$ 为初始路径, $P(a,d,f)$ 为最终路径 (0, f 分别为更新开始与完成的时间). 并假设, 网络中节点在更新前没有 TCAM 资源耗尽的情况, 即有剩余空间执行更新机制.

A.1. LUSR 算法保证数据流更新期间的一致性.

证明 LUSR 算法启用的是分段路由机制, 算法首先从待更新数据流集 C 中甄别出可拼接数据流与不可拼接数据流. 对于集合中任意不可拼接数据流, 数据包都将继续按照初始路径转发, 即在 LUSR 算法更新期间的任意时间 t_0, 都有 $P(a,d,t_0)=P(a,d,0)$, 符合一致性要求. 对于集合中任意可拼接数据流, 设在时间 T 源节点 a 开始为数据包加装 SID, 当 $0\leqslant t\leqslant T$ 时, 都有 $P(a,d,t)=P(a,d,0)$ 成立, 当 $T\leqslant t\leqslant f$ 时, 都有 $P(a,d,t)=P(a,d,f)$ 成立, 符合一致性要求. 综上, LUSR 可以保证更新期间的一致性.

A.2. IBUA 算法可以求解出最长的一致性节点更新序列, 并在更新期间保证一致性.

证明 根据定义, check_consistency 函数可以甄别出在当前更新步骤下所有可以被一致性更新的待更新节点, 而通过回溯机制 (3.2 小节中有详细论述), IBUA 算法可以探索所有可能的更新序列, 从而找到最长一致性更新序列.

对于 IBUA 算法所求的序列 S 中的任意节点 x, 在其更新后, 存在 $\mathrm{next}(x,d,t)=\mathrm{next}(x,d,f)$, 即节点完成更新, 并且按照序列 S 更新节点可以保证更新一致性; 而不在序列 S 中的节点与 S 中节点未更新前, 有 $\mathrm{next}(x,d,t)=\mathrm{next}(x,d,0)$, 即 IBUA 算法可以保证网络更新期间一致性.

综上, IBUA 算法可以求解出最长一致性更新序列, 并且在更新期间保证一致性.

A.3. 本文的 LFCA 算法可以完成一致性更新.

证明 假设 IBUA 算法求出的序列为 S, LUSR 算法可更新的数据流集合为 G, $G\in C$. 我们只需证明, 对于任意待更新数据流 $l\in C$, 源节点 a 出发的到目的节点 d 的所有数据包, 要么沿着初始路径转发要么沿着最终路径转发到目的节点 d. 在节点 a 启用 two-phase commit 机制前, 路径的一致性由 LUSR 算法或者 IBUA 算法保证, LUSR 与 IBUA 算法更新期间一致性保证已经由性质 (1) 与 (2) 证明.

如果数据流 l 为可拼接数据流, 则数据流的一致性由性质 (1) 保证, 且在 T 时刻源节点加装 SID 后, 有 $P(a,d,t)=P(a,d,f)$, $t>T$ 成立, 即数据流 l 被一致性更新.

如果数据流 l 为不可拼接数据流, 设源节点在时间 t_s 为数据流加装最终路径的标签, 则对于任意时间 $t>t_s$, 路径 $P(a,d,t)$ 中的任意节点 x, 有以下 3 种情况: (1) 节点 x 在序列 S 中, 已经被更新; (2) 节点匹配最终路径标签, 启用最终路径规则; (3) 节点 x 更新过程中无需更新. 无论哪种情况, 对于节点 x 都有 $\mathrm{next}(x,d,t)=\mathrm{next}(x,d,f)$ 成立, 即 $P(a,d,t)=P(a,d,f)$, 即数据流 l 被一致性更新.

综上, C 中任意数据流 l 都可以被一致性更新, 本文的算法机制正确性有效性得证.

Consistent update of distance vector routing hybrid SDN networks

Changhe YU[1], Julong LAN[1], Yuxiang HU[1*], Yanfei ZHANG[2] & Hao WANG[2]

1. *National Digital Switching System Engineering Technological R&D Center, Zhengzhou* 450002*, China*;
2. *Chengde branch of Hebei Special Equipment Supervision and Inspection Institute, Chengde* 067000*, China*
* Corresponding author. E-mail: chxachxa@126.com

Abstract Hybrid SDN networks are a transition between traditional networks and SDN networks. They can offer traditional services while achieving some SDN advantages such as better manageability. However, updates to hybrid SDN networks lack effective techniques, especially for the case where distance vector routing is involved. In this paper, we impose segment routing on updates to hybrid SDN networks, and formalize the application of segment routing in hybrid SDN networks. By combining ordered scheduling and two-phase commit, we propose the LFCA algorithm. The algorithm attempts to stitch the final path using segment routing and to install the routing information into the packet header first. Nevertheless, for flows that are not segmentable, the algorithm constructs the dynamic topology set that can react to the remote effect of updates, and evaluates every update operation by backtracking until it finds the longest consistent update sequence. Last, a two-phase commit mechanism is leveraged to update the remaining nodes. Our experiments verify that this algorithm can significantly save on rule overhead and time costs in comparison with prior techniques.

Keywords segment routing, distance vector routing, hybrid SDN networks, consistency

Changhe YU received his B.E. degree from Shanghai Jiao Tong University (SJTU). He is currently a master's candidate at the National Digital Switching System Engineering and Technological R&D Center (NDSC). His research interests are future networks and network security.

Julong LAN is a full professor at NDSC, China. His research interests mainly include routing and switching design, routing protocols, resource scheduling, network security, and future networks.

Yuxiang HU is an assistant professor at NDSC, China. His research interests mainly include network security, routing protocols, and future networks.