计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(10):128−137 [doi: 10.15888/j.cnki.csa.008187]
©中国科学院软件研究所版权所有.
E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于分类搜索的 SDN 流表无环一致性更新方案①

杨荣宽[1,2], 张奇支[1,2], 赵淦森[1,2], 郑伟平[1,2]

[1](华南师范大学 计算机学院, 广州 510631)
[2](广州市云计算安全与测评技术重点实验室, 广州 510631)
通讯作者: 张奇支, E-mail: zhangqizhi@m.scnu.edu.cn

摘　要: 在软件定义网络 (Software-Defined Networking, SDN) 中, 由于配置策略的改变导致控制器需要对多个交换机中的流表项进行更新时, 会出现更新不一致的情况. 其内在原因是控制器无法同时对所有交换机完成更新, 不同的更新时延会导致网络状态在逻辑上的不一致, 从而影响数据报文的正确转发. 针对分类时序更新方案应用场景适用性差和更新时延长, 最优化更新方案计算复杂度高等问题, 本文在两者的基础上, 提出基于分类搜索的无环更新一致性方案 (Categorical Search based loop-free Consistent Update scheme, CSCU). 方案通过设计交换机分类模型, 并在分类的基础上, 结合节点依赖思想设计环路搜索优化模型, 实现更新时延短, 更新效率高的一致性更新. 仿真结果表明, 本方案有更好的场景适用性和更低的节点操作复杂度, 也有更少的更新轮次和更低的计算复杂度, 可有效提升更新性能.

关键词: SDN; 流表; 一致性更新; 无环; 更新轮次

引用格式: 杨荣宽,张奇支,赵淦森,郑伟平.基于分类搜索的 SDN 流表无环一致性更新方案.计算机系统应用,2021,30(10):128–137. http://www.c-s-a.org.cn/1003-3254/8187.html

Categorical Search Based Loop-Free Consistent Update Scheme for Flow Tables in SDN

YANG Rong-Kuan[1,2], ZHANG Qi-Zhi[1,2], ZHAO Gan-Sen[1,2], ZHENG Wei-Ping[1,2]

[1](School of Computer Science, South China Normal University, Guangzhou 510631, China)
[2](Key Laboratory on Cloud Security and Assessment Technology of Guangzhou, Guangzhou 510631, China)

Abstract: In Software-Defined Networking (SDN), inconsistent updates are frequent when the controller needs to update the flow table entries in multiple switches due to the change of configuration policy. The internal reason of this phenomenon is that the controller updates all switches asynchronously. Update delay leads to the logical inconsistency of the network state, affecting the correct forwarding of data messages. With regard to poor generality of application scenarios and prolonged update time of the classification and sequence based update scheme and high computational complexity of the optimal update scheme, this study proposes a Categorical Search based loop-free Consistent Update scheme (CSCU) for flow tables. In this scheme, a switch classification model is developed on the basis of classification, and a loop search optimization model is built according to the idea of node dependence. Those contributions achieve the consistent update with short update delay and high update efficiency. The simulation results show that the proposed scheme has promising applicability with lower node operation complexity as well as fewer update rounds, which can markedly improve the update performance in terms of lower computational complexity.

Key words: Software-Defined Networking (SDN); flow table; consistent update; loop-free; update rounds

① 基金项目: 国家重点领域研发计划 (2019YFB1804003, 2018YFB1404402); 广州市科技计划 (201802030004, 201804010314); 广东省重点领域研发计划 (2019B010137003, 2018A07071702, 2016B030305006)
Foundation item: National Key Field Research Program of China (2019YFB1804003, 2018YFB1404402); Science and Technology Plan of Guangzhou Municipality (201802030004, 201804010314); Key Field Research Program of Guangdong Province (2019B010137003, 2018A07071702, 2016B030305006)
收稿时间: 2021-01-12; 修改时间: 2021-02-02; 采用时间: 2021-03-19

© 中国科学院软件研究所 http://www.c-s-a.org.cn

1 引言

1.1 研究背景及介绍

SDN (Software-Defined Networking) 是一种将网络控制平面和数据平面分离、具备可编程性的新型网络架构[1]. 基于 OpenFlow 协议的 SDN 架构将底层基础设施抽象出来, 使用户能够通过编程来集中控制网络转发行为. 当网络状态发生变化时, SDN 控制器可通过下发新的转发规则更新交换机流表项, 从而管理整个网络状态. 尽管 SDN 实现了网络的集中控制, 但数据平面设备的分布式特性也带来了许多问题, 如交换机自身安装流表项的速度、交换机间的传输延迟等. 这使得流表项生效时间难以一致, 进而导致更新不一致, 增加了集中管理的难度[2].

目前, 我们正在参与一项国家重点研发计划子课题的项目研究. 该项目的一个主要研究目标是在政务通信网络中提出一种能保证数据转发高可靠性、提供细粒度 QoS 保障的机制. 而数据转发的可靠性和细粒度 QoS 直接依赖于 SDN 转发策略的更新一致性. 因此, 保证 SDN 流表更新一致, 对保证项目中的数据转发高可靠性和细粒度 QoS 具有重大意义. 本文在此基础上, 研究能保证 SDN 流表更新一致、数据可靠转发、更新性能好的更新方案.

1.2 SDN 流表一致性更新

在 SDN 网络中, 当交换机收到一个数据报后, 它首先查找转发流表. 若有匹配的流表项, 则执行相应的动作. 否则交换机会向控制器发送 packetin 消息, 将报文上报至控制器进行处理. 控制器处理后, 会向交换机下发新的转发流表项指导后续转发. 此外, 当网络状态发生变化时, 控制器会主动或被动向交换发送新的转发流表项, 以形成新的转发路径[3].

由于数据平面的交换机呈分布式排列, 控制器向每台交换机下发新流表的时延会有不同, 交换机内部的更新速度也有差异, 这会导致新规则的所有流表项无法同时生效, 使数据包出现错误的转发处理, 引发数据转发中断、环路等问题, 影响网络流量传输的可靠性和细粒度 QoS 的保证.

因此, 在新旧规则的更新过程中, 必须要保持流表的一致性更新. 即控制器在更新交换机中的流表项时, 那些正在传输的数据包在每个交换机上要么匹配旧规则流表项, 要么匹配新规则流表项, 不能出现在交换机 A 上匹配旧规则, 而在交换机 B 上又匹配新规则的情况. 为了更好地说明 SDN 流表更新不一致性引起的问题, 下面以图 1 进行举例说明. 其中节点 N_i ($1 \le i \le 8$) 代表交换机, 实线代表旧转发路径, 虚线代表新转发路径.

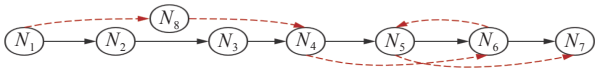


图 1 路由更新实例

假设某时刻 SDN 控制器同时向各个交换机下发更新命令: 将转发规则更新至新路径. 由于控制器和不同交换机的传输时延差异, 交换机无法同时完成更新. 当 N_1 完成更新而 N_8 尚未更新时, 即 N_8 没有匹配数据流的转发规则, 则转发到 N_8 的数据流将不知作何处理. 如果默认动作是丢弃报文, 则会引起流中断. 此外, 如果交换机 N_6 更新了而 N_5 尚未更新, 就会形成环路 $N_6 \rightarrow N_5 \rightarrow N_6$. 这会长时间占用链路直至报文的 TTL 超时, 从而降低链路利用率和转发性能.

通常来说, SDN 流表更新出现不一致时, 可能会导致网络中断、环路、安全漏洞等问题. 为确保从旧规则过渡到新规则期间, 网络能正确转发报文, 则必须要考虑所有可能出现的中间状态及其属性[4]. 很明显, 流表的一致性更新, 关系到网络配置能否正确实现.

目前, 针对 SDN 流表更新一致性问题, 研究者已经提出了多种更新方案, 如两阶段更新[5]、反向更新[6]、基于分类时序更新[7]、时间触发更新[8]、分段更新[9] 等.

1.3 相关工作

为了确保更新无黑洞和无循环的一致性属性, Reitblatt 等首先提出每流/每包一致性的定义[5], 并基于此定义提出了两阶段更新方案进行策略转发更新.

周晔等[10]、齐婵等[7] 在两阶段更新的基础上分别提出了基于分类和基于分类时序的流表更新一致性方案, 核心均在于根据更新操作类别对交换机进行分类, 并按照一定的顺序更新规则. 不同的是, 分类时序更新增加了对流表项的细分操作. 分类时序更新虽算法简单, 但其根据严格设定的顺序来更新新旧转发路径上的节点, 忽略了同时是新旧路径上的交换机的情况, 存在应用场景局限性. 此外, 不管更新拓扑复杂与否, 都需要等待一定的端到端最长时延才能进行下一阶段, 方案更新轮次固定, 更新时延过长.

基于节点依赖关系, Diogo 等[6] 提出了反向更新方

© 中国科学院软件研究所 http://www.c-s-a.org.cn

案. 该方案虽算法简单, 但其节点依赖复杂, 也没有考虑节点规则无须更新的情况, 严格依据新规则转发路径从目的节点递归向前更新以避免环路. 因此, 更新轮次多, 致使更新时延长. 另外, Fu 等[11] 提出双向更新方案. 通过寻找满足当前更新状态下其子节点都已更新至新规则, 或者在最终状态下其所有父节点都已经更新至新规则这两个条件的节点依次更新. 但每次最多更新 2 个节点, 更新轮数依然过多. 针对依赖问题, Mahajan 等[12] 提出了最优化更新. 方案设定每个节点都具有 3 个状态, 在保证无环的情况下, 让尽可能多的节点并行更新, 从而最大程度减少更新轮次. 但此方案每一轮都要全局遍历以更新节点状态, 节点搜索复杂度高.

1.4 基于分类搜索的无环更新

针对分类时序更新应用场景适用性差和更新时延长、最优化更新计算复杂度高等问题, 本文在两者基础上, 提出基于分类搜索的无环更新一致性方案 (Categorical Search based loop-free Consistent Update scheme, CSCU). 在 CSCU 方案中, 我们设计了一种交换机分类模型和一种环路搜索优化模型, 其中优化模型包括环路检测模块和搜索模块. 两种模型结合的更新流程可概括为: 分类模型先分析所有节点的更新前后状态并初始化; 再调用环路搜索优化模型的环路检测模块获取对应节点集的可更新节点进行更新. 最后, 在未更新节点集中循环调用优化模型的搜索模块查找满足条件的可更新节点, 直至更新完成.

本文的主要贡献在于:

(1) 改进分类时序更新算法中的分类方式, 构建了一种新的交换机分类模型, 它可以增大应用场景的适用性和减少更新等待时延.

(2) 在分类和节点依赖的基础上, 通过分析更新前后转发规则的差异, 构建相应的环路搜索优化模型, 优化了更新的搜索计算复杂度和更新效率.

(3) 在分类模型和环路搜索优化模型的基础上, 提出一种基于分类搜索的更新方案, 它具有更新时延短, 更新效率高的优点.

2 SDN 系统模型

在本节中, 我们通过构建网络模型和更新模型对整个 SDN 网络的更新过程进行分析.

2.1 网络模型

我们将网络建模抽象为一个图 $G=(N, L)$, 其中, N 是网络的节点集, L 是节点间的链路集. 为了简化, 本节假设网络模型为单一控制器 C_0 和多 OpenFlow 交换机 $S=\{s_1, \cdots, s_{n-1}\}$的模式, 图 2 展示了本节所述的网络模型. 其中, 实线是交换机间转发数据包的数据链路, 虚线是控制器和 OpenFlow 交换机间的控制链路 (也称 OpenFlow 通道).

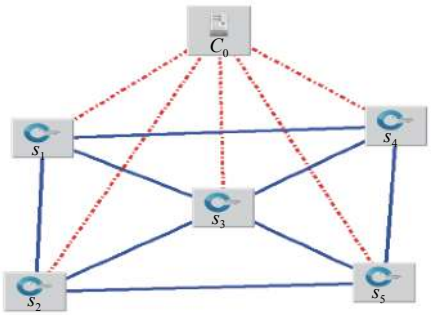


图 2 SDN 网络示意图

在 SDN 网络中, 节点集 N 可用数学表示为: $N = C \cup S$, 其中, C 是控制器集, S 是交换机集. 链路集 L 可定义为$L = L_{cc} \cup L_{cs} \cup L_{ss}$, 其中, L_{cc} 指代控制器与控制器之间的链路集, L_{cs} 指代控制器和交换机间的控制链路集, L_{ss} 指代交换机和交换机间的传输链路. 由于模型简化为单控制器模式, 因此集合 L_{cc} 为空, 集合 L_{cs} 的元素个数为$|S|=n-1$.

交换机流表规则 R_i 可定义为一个三元组$<Pr_i, M_i, A_i>$. 其中, Pr_i 指代规则的优先级, M_i 指代规则的匹配域集 (OpenFlow 协议中, 流规则使用多个字段来匹配数据包, 包括以太网源地址、以太网目的地址、VLAN ID 等), A_i 指代规则的动作集. 如果进入网络的数据包同时匹配多条规则, 则选择优先级最高的规则, 并执行相应的动作.

2.2 更新模型

网络更新是对网络配置的更新操作: SDN 控制器生成更新消息并将其转发给交换机, 然后交换机根据此消息改变它的规则表. 很明显, 更新前后涉及交换机状态的变化. 因此可将 t 时刻的交换机状态定义为[2]:

$$ST_i(t) = \{R_i(t), L_{csi}(t), L_{ssi}(t)\} \quad (1)$$

其中, $R_i(t)$ 指在 t 时刻的交换机 S_i 的规则集, $L_{csi}(t)$ 代表 t 时刻与 S_i 相关的控制链路集, $L_{ssi}(t)$ 代表 t 时刻与 S_i 相关的数据传输链路集.

网络配置, 即网络中所有交换机的状态并集, 则 t 时刻的网络配置可定义为:

© 中国科学院软件研究所 http://www.c-s-a.org.cn

$$E(t)=\bigcup_{i=1}^{|S|} ST_i(t) \tag{2}$$

网络更新是对网络配置的更新操作, 即不同时刻的网络配置的迁移. 因此, 网络更新可定义为:

$$E_{\text{src}}(t_i) \xrightarrow{U} E_{\text{dst}}(t_j),\ t_i < t_j \tag{3}$$

其中, $E_{\text{src}}(t_i)$ 代表更新前的网络配置, $E_{\text{dst}}(t_j)$ 代表更新后的网络配置, U 指代网络配置迁移过程的更新操作集.

基于 OpenFlow 协议中流表的组成部分, 可以将数据包在交换机中的处理定义为布尔函数:

$$f_s(p)=X\rightarrow Y \tag{4}$$

其中, s 代表交换机, p 代表数据包; X 代表匹配域, Y 代表处理结果 (包含动作: 将数据包转发到其他交换机或将数据包上传至控制器).

数据包从进入网络到离开网络, 是数据包在交换机间的相互转发过程. 由式 (4) 可知, 数据包在网络转发是个迭代处理过程, 因此, 可将数据包在网络中的操作以布尔函数的形式定义为网络转发函数 $F(p)$:

$$F(p)=f_k(f_j(f_i(p))) \tag{5}$$

其中, p 指代数据包, i, j, k 指代交换机编号 (假设数据包在网络中的传输路径为 $i\rightarrow j\rightarrow k$, 即数据包 p 先在交换机 S_i 被处理, 然后根据定义 (4) 得到的结果, 将数据包转发到交换机 S_j 中进行处理, 以此类推, 直至转发完成).

使用π_{old}表示旧数据包, 则旧数据包可定义为: 如果一个数据包 p 被尚未更新的交换机处理, 那么它就被标记为π_{old}. 同理, 使用π_{new}表示新数据包, 则可定义为: 如果一个数据包被已更新了的交换机处理, 它将被标记为π_{new}. 因此可将网络转发函数拓展为:

$$\left\{\begin{array}{c} F(p)=F_{\text{new}}(p) \\ \text{or} \\ F(p)=F_{\text{old}}(p) \end{array}\right\} \tag{6}$$

其中, $F_{\text{new}}(p)$ 和 $F_{\text{old}}(p)$ 代表π_{new}和π_{old}由新/旧规则集处理.

包一致性. 由网络转发函数定义可得: 每个数据包的传输函数应该只涉及一个规则集: $F(p)=F_{\text{new}}(p)$ 或 $F(p)=F_{\text{old}}(p)$, 而不能是两者的混合.

更新持续时间 (U_d). U_d 指控制器发送第一个更新消息和最后一个交换机更新完成之间的时间间隔.

更新轮次 (U_{round}). 本节假设在单个交换机上安装的规则之间没有冲突或者依赖关系, 从而可以在单轮消息传递中更新交换机而不影响网络一致属性. U_{round} 指更新过程所需的阶段更新次数, 在一定条件下, U_{round} 直接影响 U_d, 两者成正相关. 每轮中, 控制器更新一个交换机子集, 并保证以任何顺序更新该子集的交换机都能确保无环路产生. 控制器在确认前一轮中选择的交换机都已经在内存中安装了新的规则后, 再开始新一轮的更新.

节点操作复杂度 (N_{cp}). 指更新过程对交换机流表操作的复杂程度. 流表规则更新操作类型和操作次数是 N_{cp} 的关键影响因素, 会直接影响 U_d 和更新性能.

本节模型构建主要为了优化更新调度算法, 提升更新性能. 因此, 我们可将本模型的优化目标和约束问题描述为:

$$\left\{\begin{array}{c} \min(U_{\text{round}}) \\ \min(N_{\text{cp}}) \end{array}\right\} \rightarrow \min(U_{\text{d}}) \tag{7}$$

$$loop(E(t_i) \xrightarrow{U} E(t_j)) = \text{false} \tag{8}$$

$$\left\{\begin{array}{c} F(p)=F_{\text{new}}(p) \\ \text{or} \\ F(p)=F_{\text{old}}(p) \end{array}\right\} \tag{9}$$

其中, 式 (7) 为优化目标, 即通过最大程度的减少更新轮次 U_{round} 和降低节点操作复杂度 N_{cp} 来减少更新持续时间 U_d. 式 (8) 和式 (9) 为约束问题, 其中, 式 (8) 代表更新过程需保证不会引起转发环路、式 (9) 代表更新过程需保持包级一致性.

本模型的难点在于如何设计更新调度算法, 在保证更新过程无环路产生、数据包转发保证一致性的前提下, 尽可能的减少更新所需时间, 提升更新性能. 由于单轮更新不会影响网络的一致属性, 因此, SDN 控制器可协调交换机间的规则更新, 通过设计更新算法, 使更新轮次最小化, 减少完成更新的所需时间. 已经证明, 最小化更新轮数是 NP 完全问题[13].

3 基于分类搜索的无环一致性更新方案

3.1 问题描述

定义 P_{old} 和 P_{new} 分别代表 $E_{\text{src}}(t_i)$ 下的数据原始传输路径和 $E_{\text{dst}}(t_j)$ 下的数据最终传输路径, $nexthop(s, P_{\text{old}}/P_{\text{new}})$代表节点 s 在路径 P_{old} 或 P_{new} 的下一跳. 更新过程中, 配置是否更新成功受设备之间的实时交互时延、设备的实时更新操作所影响, 可能会因为上述因素导致网络中出现环路等问题. 基于以上定义, 可将方

© 中国科学院软件研究所 http://www.c-s-a.org.cn

案的问题描述为: 已知 $E_{src}(t_i)$ 和 $E_{dst}(t_j)$, 通过分析 P_{old} 和 P_{new} 上的节点, 在满足式 (8) 和式 (9) 的前提下, 设计更新算法, 寻找更新轮数较少、更新性能较优的操作集 U, 求得式 (7) 的最优解.

3.2 基于分类搜索的更新方案

3.2.1 CSCU 方案的节点集合定义

在构建模型过程中, 本方案基于分类和节点依赖搜索思想, 需要创建和更新以下相关集合, 如表 1. 其中, $P_{new}[length-1]$ 指代最终传输路径上的最后一个节点、S_{subi} 代表每一轮更新后剩余的待更新节点集、r_{max} 代表完成整个更新过程所需要的最大更新轮数、$check_loop(s,P_{old},P_{new})$用于判断更新节点 s 是否引入环路、$judge_allfather(s,P_{old},P_{new})$用于判断节点 s 的父节点是否都已经更新至新规则.

表 1 相关节点集

节点集	公式
S_{node}: 存储待更新节点的集合	$S_{node}=\{s\mid((s\in P_{old}\cap P_{new}, nexthop(s,P_{old})\neq nexthop(s,P_{new}))\cup (s\in P_{new}\cap s\notin P_{old}))\}$
S_{un}: 存储无须更新节点的集合	$S_{un}=\{s\mid((s\in P_{old}\cap P_{new}, nexthop(s,p_{old})=nexthop(s,p_{new}))\cup (s\in P_{new}[length-1]))\}$
S_{del_old}: 存储只需删除旧规则节点的集合	$S_{del\_old}=\{s\mid s\in P_{old}\cap s\notin P_{new}\}$
T_i: 存储第i轮可更新节点的集合	$T_i=\begin{cases}\{s\mid s\in S_{node}, check\_loop(s,P_{old},P_{new})=\text{True}\} & i=1\\ \{s\mid s\in S_{subi}, judge\_allfather(s,P_{old},P_{new})=\text{True}\} & 1<i<r_{max}\end{cases}$
$S_{updated}$: 存储已更新节点的集合	$S_{updated}=\left\{s\mid s\in\sum_{i=1}^{r_{max}}T_i\right\}$

3.2.2 CSCU 方案的流程

先给出节点依赖关系的定义: 当节点 N_a 必须在节点 N_b 更新完成之后才能更新, 则节点 N_a 依赖于节点 N_b. 另外, 由于目的节点不涉及规则更新, 定义为不需要更新节点. CSCU 方案的更新流程如图 3 所示.

首先, CSCU 方案利用节点分类模型对 $E_{src}(t_i)$ 和 $E_{dst}(t_j)$ 进行分析, 通过对比 P_{old} 和 P_{new} 节点状态, 初始化表 1 中的节点集.

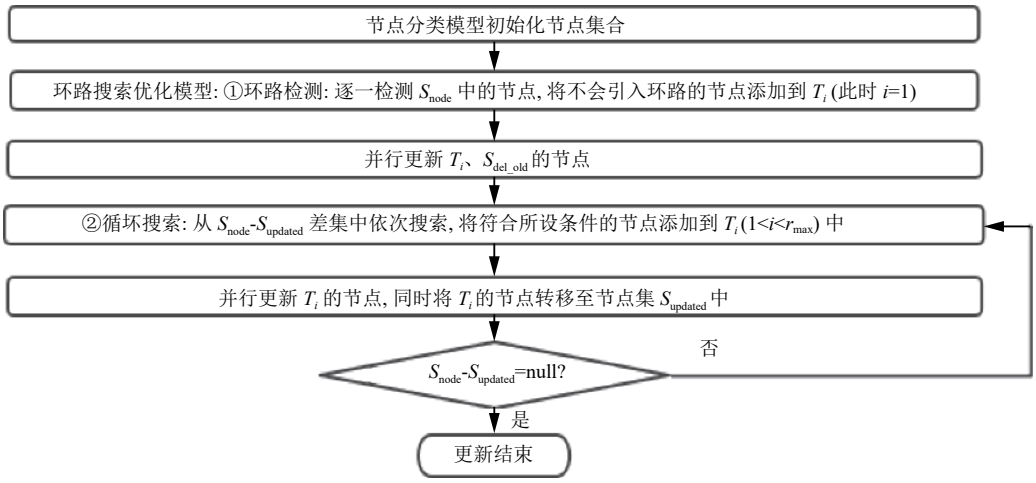


图 3 CSCU 更新流程

分类模型初始化后, 执行环路搜索优化模型中的第一个模块. 调用基于拓扑排序算法的环路检测模块逐一检测 S_{node} 中的全部节点—直接给节点增添新的转发规则. 如果$check_loop(N_i,P_{old},P_{new})=\text{True}$, 即不会引入环路, 则将 N_i 添加到节点集 T_i 中 (此时 i=1). 检测完毕后, 并行更新节点集 T_i 和 S_{del_old} 中的节点. 其中, 对 T_i 中的节点实施以下操作: 下发新规则替换旧规则, 同时将节点转移至节点集 $S_{updated}$ 中. 对于 S_{del_old} 中的节点, 直接删除对应的旧规则后, 移除该节点. 旧规则是否删除, 同样是判断一个节点是否已经更新的依据. S_{del_old} 存储只需删除旧规则的节点, 由于节点规则失效时间不确定, 如果没有及时删除 S_{del_old} 中节点

© 中国科学院软件研究所 http://www.c-s-a.org.cn

的旧规则, 会影响方案接下来根据依赖关系搜索可更新节点的算法. 当以上节点集更新完成, 算法继续.

接着, 调用环路搜索优化模型中的第二个模块. 对 $N_i \in S_{node} - S_{updated}$ 依次搜索可更新节点, 将满足条件 $Find\_father(N_i) = null$ 或 $judge\_allfather(N_i, P_{old}, P_{new}) = True$ 的节点 N_i 加入 T_i 中, 前面两个条件表示当前状态下的节点 N_i 无父节点或者父节点都已更新. 搜索完后, 并行更新 T_i 的节点, 同时将 T_i 的节点转移至 $S_{updated}$ 中. 如果 $S_{node} - S_{updated} \neq null$, 则循环执行搜索. 直至 $S_{node} - S_{updated} = null$, 方案结束.

后文我们将证明, 结合分类模型和环路搜索优化模型双模型, CSCU 方案能有效提高更新效率, 且具有更低的计算复杂度.

3.2.3 CSCU 方案的更新示例

现以图 4 所示拓扑为例, 将网络抽象为一个点对点有向图来演示 CSCU 方案的更新过程. 其中, 实线代表网络配置更新前数据转发路径, 虚线代表更新后数据转发路径. 更新步骤如图 4 所示.

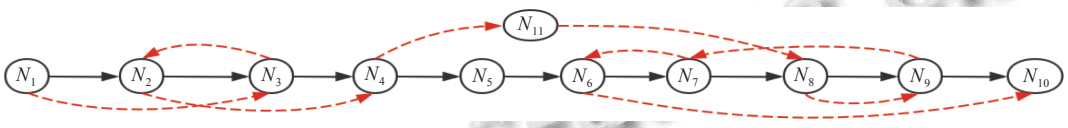


图 4 更新例子

(1) 执行节点分类模型初始化集合得: $S_{node}=\{N_1, N_2, N_3, N_4, N_6, N_7, N_9, N_{11}\}$, $S_{del\_old}=\{N_5\}$, $S_{un}=\{N_8, N_{10}\}$, $S_{updated}=null$, $T_i=null$.

(2) 调用环路搜索优化模型的环路检测模块检测得: $T_1=\{N_1, N_2, N_4, N_6, N_{11}\}$. 并行更新节点集 S_{del_old} 和 T_1 中的节点, 对 $s \in T_1$ 执行新旧规则替换并将 s 转移至 $S_{updated}$ 中. 对 $s \in S_{del\_old}$ 执行删除旧规则操作, 并移除节点 s. 此时得到 $S_{updated}=\{N_1, N_2, N_4, N_6, N_{11}\}$, $S_{del\_old}=null$.

(3) 然后, 调用搜索模块从 $S_{node}-S_{updated}$ 依次搜索符合条件的节点得 $T_2=\{N_3, N_7\}$, 执行更新后, 此时得到 $S_{updated}=\{N_1, N_2, N_3, N_4, N_6, N_7, N_{11}\}$.

(4) 循环搜索得 $T_3=\{N_9\}$, 执行更新后得 $S_{node}=\{N_1, N_2, N_3, N_4, N_6, N_7, N_9, N_{11}\}$. 很明显, 此时 $S_{node}-S_{updated}=null$ 且 $S_{del\_old}=null$, 更新完毕.

3.2.4 与其他方案对比

以图 4 为例将 CSCU 方案与其他方案做比较:

(1) 基于分类时序更新[7]: ① 控制器将交换机初始化, 分成 4 类: 入口交换机、新路径交换机、旧路径交换机、出口交换机. ② 更新出口交换机 $\{N_{10}\}$ 和新路径交换机 $S_{new}=\{N_1, N_3, N_2, N_4, N_{11}, N_8, N_9, N_7, N_6\}$, 执行插入待新增流表项操作. ③ 等待一个最长端到端时延后, 更新入口交换机 $\{N_1\}$, 执行修改流表项操作. ④ 等待最长端到端时延, 更新旧路径交换机 $S_{old}=\{N_1, N_2, N_3, N_4, N_5, N_6, N_7, N_8, N_9\}$ 和出口交换机 $\{N_{10}\}$, 执行删除流表项操作, 全部更新完毕. 显然, 在新旧规则所涉及的交换机存在复杂关联时, 方案需要在不同阶段对大量共同交换机、不需更新的交换机进行不同的流表操作, 增加了交换机更新操作次数, 这会增加更新的等待时延.

(2) 反向更新[6]: 该方案所有的节点严格依据新规则从目的节点递归向前更新. 由于默认目的节点不需更新, 因此, 所需的更新轮次=新规则涉及的节点数−1. 由图 4 示例可知, 新规则转发路径涉及的节点数为 10, 即需要 9 轮才能完成更新.

(3) 双向更新[11]: 依据两个搜索条件得到 $\{(N_1, N_6)\rightarrow(N_2, N_7)\rightarrow(N_3, N_9)\rightarrow(N_4, N_8)\rightarrow(N_5, N_{11})\}$, 共需要 5 轮才能完成更新.

(4) 最优化更新[12]: 依次加入新规则对所有节点进行无环检测, 找到的依赖森林根节点有 $\{N_1, N_2, N_4, N_5, N_6, N_{11}\}$, 分别删除其旧规则, 再通过无环检测寻找其子节点, 得到依赖树有: $N_2\rightarrow N_3$, $N_6\rightarrow N_7\rightarrow N_9$, 故整个依赖森林为 $\{N_1, N_2\rightarrow N_3, N_4, N_5, N_6\rightarrow N_7\rightarrow N_9, N_{11}\}$. 其中, 依赖森林中依赖树的最大高度为所需更新轮次, 即完成更新共需要 3 轮.

本文提出的 CSCU 方案: 执行分类模型后, 通过执行环路搜索优化模型的无环检测模块, 得到的 $T_1=\{N_1, N_2, N_4, N_6, N_{11}\}$, 再循环执行搜索模块获得 $T_2=\{N_3, N_7\}$, $T_3=\{N_9\}$. 所以, 经过 3 轮即可完成更新.

3.3 CSCU 无环性验证

无环性是方案的主要研究点, 证明如下:

(1) 首先, 环路搜索优化模型中的无环检测模块不

© 中国科学院软件研究所 http://www.c-s-a.org.cn

会引入环路. 事实上, 在更新过程中, 基于拓扑排序的无环检测模块对节点集 S_{node} 中的节点进行检测, 是根据直接增加新规则是否会产生环路来决定. 若产生环路则保持状态不变, 否则就将它逐一添加进节点集 T_i 中. 然后并行更新 T_i 和 S_{del_old} 中的节点: 对 $s \in T_1$ 执行新旧规则替换并加入 $S_{updated}$ 中, 对 $s \in S_{del\_old}$ 执行旧规则删除. 因为节点只有通过无环检测才确定是否更新, 并行更新 T_i 中的节点的过程显然不会引入环路. 另外, 对 $s \in S_{del\_old}$ 执行删除旧规则的操作, 只是释放相应的交换机内存空间, 不会引起无环. 所以, 无环检测更新过程不会引入环路.

(2) 其次, 环路搜索优化模型中的循环搜索模块也不会引入环路. 事实上, 按默认方法更新至最终状态的路径是不存在环路的. 现假设当前状态下有局部链路 $N_1 \rightarrow N_2 \rightarrow N_{cur}$, N_{cur} 尚未更新, 其父节点 N_1, N_2 都已更新. 更新 N_{cur}, 形成环路 $N_{cur} \rightarrow N_1 \rightarrow N_2 \rightarrow N_{cur}$, 而根据假设, 其父节点都已更新, 即整个环路链路涉及的节点都已更新至新规则. 该环路是整个更新至最终状态的一个局部链路, 则与最终状态下的新规则中无环路的矛盾, 所以更新 N_{cur} 不会引入环路. 循环搜索更新, 直至 $S_{node} - S_{updated} = \text{null}$ 时更新完毕, 整个更新过程都不会出现环路.

3.4 相关研究比较

本节将从适用性、更新持续时间 U_d、更新节点操作复杂度 N_{cp}、更新算法计算复杂度 4 个方面进行比较. 适用性, 指方案场景适用范围. U_d 为整个更新过程每个阶段所用时间总和, 集中体现于更新过程的所需轮次 U_{round}, 更新所需轮次越多, 等待更新的时间越长, U_d 越大. N_{cp} 指更新过程对节点操作的复杂程度, 集中体现于: 需在不同更新阶段执行不同更新操作的节点越多、对同一节点更新操作的次数越多, 更新等待时延就越长, U_d 越大. 更新算法计算复杂度, 指切换更新规则过程中相应逻辑功能计算的复杂性.

本节将本文方案与第 2 节的相关研究进行比较, 结果如表 2 所示.

表 2 本文与相关方案比较

方案	适用性	更新持续时间	节点操作复杂度	算法计算复杂度
分类时序更新	较差	一般	高	简单
反向更新	一般	较长	一般	一般
双向更新	一般	较长	一般	一般
最优化更新	良好	短	一般	高
本文(CSCU)	良好	短	一般	一般

(1) 适用性. 不管网络拓扑复杂与否, 分类时序更新都依据新旧转发路径上的节点严格设定阶段进行更新. 若网络配置简单, 使用该方案反而降低更新效率. 若配置更新存在大量的共同交换机, 则需要分别在不同的更新阶段, 对同一交换机进行不同的流表操作, 更新操作量增加. 因此, 分类时序存在一定的应用场景局限性, 适用性较差. 本方案在分类初始化基础上, 设计能减少更新轮数的环路搜索优化模型, 无论拓扑结构复杂与否, 本方案更新效率都有保证.

(2) 更新持续时间 U_d. 分类时序更新设定固定阶段更新, 下一个阶段的节点更新必须等待上一个阶段的节点更新完毕才能进行, 更新轮次固定. 若更新配置复杂, 还会因为等待旧包流出网络等待端到端的最长时延, 增加更新时间. 反向更新和双向更新节点依赖复杂, 导致更新轮次多, 更新时延高. 本方案结合分类模型和环路搜索优化模型, 平均更新轮次接近最优化更新, 更新持续时间短.

(3) 更新节点操作复杂度 N_{cp}. 如 3.2.4 节所述, 更新配置复杂时, 分类时序更新需在不同阶段对大量节点进行不同的流表更新操作, 大量节点更新操作次数多. 因此, 方案的更新节点操作复杂程度高, 更新等待的时延长. 虽说细化流表项操作在更新某一阶段能使新旧数据包并行传输, 一定程度上降低控制器负载. 但与其产生的等待时延相比, 时间开销更为重要. 本方案只需针对相应节点集执行相应规则操作, 操作复杂度低, 等待时延少, 能进一步减少 U_d.

(4) 算法计算复杂度. 假设使用邻接矩阵存储节点信息. 最优化更新中, 初始化所有节点状态、加入新规则检测是否引入环路都为 $O(n)$, 在状态转换过程中, 更新节点状态和下一轮节点的判环操作也都为 $O(n)$. 因此, 最优化更新的时间复杂度为 $O(n^4)$. CSCU 方案中, 分类初始化和对待更新的节点判环操作的总时间复杂度为 $O(n^2)$, 搜索可更新的节点为 $O(n)$, 因此总的时间复杂度为 $O(n^3)$. 而基于分类时序更新只需遍历矩阵进行分类, 然后在不同节点进行更新, 时间复杂度为 $O(n^2)$.

4 仿真分析

为了分析比较相关方案在不同情况下更新的效果, 本文基于自主搭建的 SDN 仿真平台进行模拟实验. 平台所用设备为 Ubuntu 16.04 操作系统, RYU 开源控制器和 mininet 网络仿真器. 仿真实验设定实验拓扑节点

© 中国科学院软件研究所 http://www.c-s-a.org.cn

数, 设定源点和目的节点. 每次实验通过随机更改相邻节点的边权值来模拟网络拓扑结构的变化, 相邻节点的边权值服从 (1–1000) 的均匀分布. 权值确定后, 再通过 OSPF 算法获取源节点到目的节点的路径.

① 关于更新轮次和更新持续时间 U_d 的关系, 在上一节已经有详细解释. 在此, 通过 300 次仿真实验测试, 对相关研究方案每一次实验的更新轮次进行分析对比, 结果如图 5 所示.

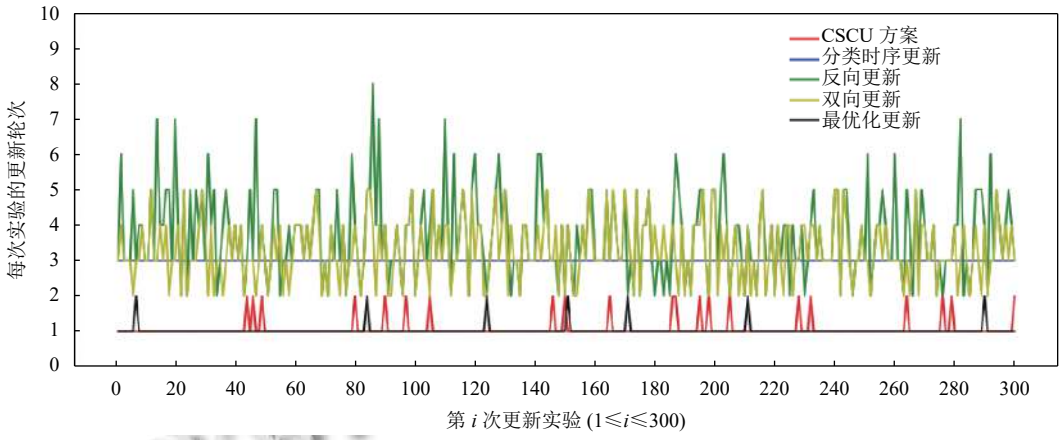
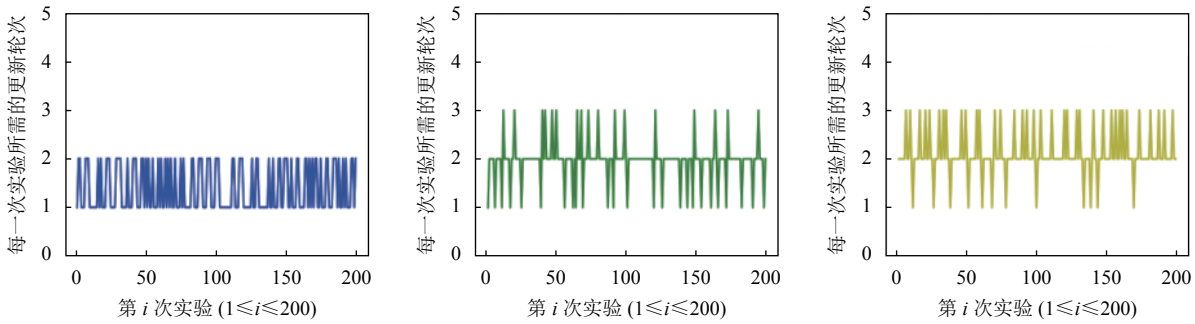


图 5 不同方案下的更新轮次情况

实验结果显示, 反向更新的所需的更新轮次普遍多于其它方案, 这是因为反向更新只与新的转发规则相关, 更新轮次=新规则转发相关节点−1. 由于分类时序更新严格设计更新阶段, 因此更新轮次固定. 本方案的更新轮次接近最优化更新, 比其他方案都少.

② 配置更新交叉程度. 指配置更新过程中, 同时在新、旧路径上的交换机占与更新相关的交换机总数的比例. 配置更新交叉程度的大小影响 CSCU 方案的更新性能. 更新过程中, 在新路径上而不在旧路径上的交换机数越多, 涉及旧规则的交换机就越少, 就越难形成回路. 因此, 更新轮次就越少, 更新性能越好. 相反则需要更多的轮数来完成更新.

至此, 就配置更新交叉程度对本方案更新轮次的影响进行实验分析. 由于交叉程度低的时候形成环路的可能性低, 所以交叉程度为 0 和 25% 的结果参考价值不大. 因此本实验只对交叉程度为 50%、75%、100% 的情况进行实验, 结果如图 6 所示.



(a) 交叉程度为 50% (b) 交叉程度为 75% (c) 交叉程度为 100%

图 6 CSCU 方案在不同配置交叉程度下的更新轮次情况

为了更好的与分类时序更新对比分析, 综合 CSCU 方案在不同交叉程度更新情况的实验结果, 求得配置更新不同交叉程度下的平均更新轮次. 实验结果表明, 本方案在配置更新交叉程度为 0、50%、75%、100% 下的平均更新轮次都比分类时序更新少, 结果如图 7 所示, 其中, 0、50%、75%、100% 分别代表

© 中国科学院软件研究所 http://www.c-s-a.org.cn

CSCU 方案的配置更新交叉程度, timing 代表分类时序更新.

③ 更新过程的节点操作复杂度直接体现于对同一交换机的流表项操作次数大于 1 的交换机数. 用 num_large_1 表示对同一交换机的流表项操作次数大于 1 的交换机数. 显然 num_large_1 越大, 更新的节点总操作次数越多, 更新过程的节点操作复杂度就越高. 针对节点操作复杂度, 本节设计多次实验, 假设实验的节点数固定为 20、新旧路径跳数相同. 对分类时序更新进行测试分析, 得出 num_large_1 的平均值随路径跳数变化的情况, 实验结果如图 8 所示. 实验设定新旧路径源、目的节点, 所以 num_large_1≥2.

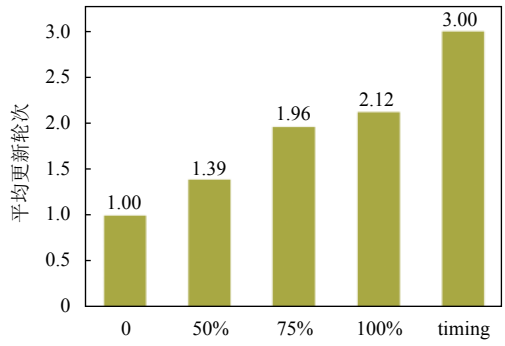


图 7 不同交叉程度下与分类时序更新的比较

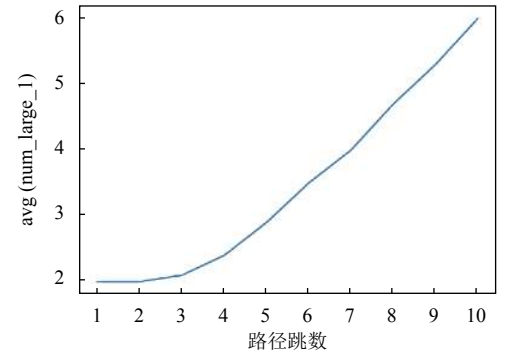


图 8 不同路径跳数的更新节点操作复杂度

分析实验可得, 网络节点数一定的情况下, num_large_1 随着路径跳数的增大而增大. 即随着路径跳数的增多, 同是新旧路径上的节点的节点数就越多, 对所有节点的总操作次数就越多. 因此, 更新过程的节点操作复杂度就越高. 而本方案经分类初始化后, 与更新配置复杂度无关, 更新节点操作复杂度低.

综上所述, 相较于分类时序更新, 基于分类搜索的无环更新一致性方案有更好的场景适用性、更低的节点操作复杂度 N_{cp} 和更少的更新轮次 U_{round}, 能够在满足模型约束条件 (8) 和 (9) 的前提下有效减少更新持续时间 U_d、提升更新效率, 实现模型的优化目标. 另外, 在更新轮次方面, 本方案相对于其他相关研究方案都有着不同方面的性能提升.

5 结语

本文根据在研项目的研究目标, 对 SDN 流表更新一致性的相关研究成果进行了详细的研究. 并针对基于分类时序更新方案存在应用场景适用性差、更新时延长和节点更新复杂程度高等问题, 提出了一种基于分类搜索的无环更新方案. 随后, 本文通过更新示例的展示, 在适用性、更新时间等性能方面对该方案和相关研究方案进行了对比分析. 此外, 本文对该方案的无环一致性更新进行了理论分析, 证明了方案的可行性. 最后, 通过搭建 SDN 仿真平台进行了仿真实验. 仿真实验针对更新轮次对该方案与相关研究方案进行了分析对比, 也针对不同配置更新交叉程度和节点操作复杂度单独跟分类时序方案进行了对比. 仿真结果表明, CSCU 方案具备应用场景适用性高、节点更新操作复杂度低的优点, 是一种能够有效的减少更新所需轮次、更新时延和降低计算时间复杂度的高效率更新方案.

参考文献

1 Kreutz D, Ramos FMV, Verissimo PE, *et al*. Software-defined networking: A comprehensive survey. Proceedings of the IEEE, 2015, 103(1): 14–76. [doi: 10.1109/JPROC.2014.2371999]

2 Maity I, Mondal A, Misra S, *et al*. CURE: Consistent update with redundancy reduction in SDN. IEEE Transactions on Communications, 2018, 66(9): 3974–3981. [doi: 10.1109/TCOMM.2018.2825425]

3 刘江, 胡晓露, 黄韬, 等. 基于多向搜索的 SDN 流表更新一致性方案. 北京邮电大学学报, 2016, 39(3): 54–59.

4 Li Q, Wang L, Jiang Y, *et al*. A fast and incremental update scheme for SDN based on a relation graph. Computer Networks, 2017, 125: 41–52. [doi: 10.1016/j.comnet.2017.05.005]

5 Foerster KT, Schmid S, Vissicchio S. Survey of consistent software-defined network updates. IEEE Communications Surveys & Tutorials, 2019, 21(2): 1435–1461.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

6 Mattos DMF, Duarte OCMB, Pujolle G. Reverse update: A consistent policy update scheme for software-defined networking. IEEE Communications Letters, 2016, 20(5): 886–889. [doi: 10.1109/LCOMM.2016.2546240]

7 齐婵, 刘建伟, 毛剑, 等. 基于分类和时序的 SDN 流表更新一致性方案. 计算机应用研究, 2018, 35(11): 3405–3408, 3412. [doi: 10.3969/j.issn.1001-3695.2018.11.049]

8 Zheng JQ, Chen GH, Schmid S, *et al*. Scheduling congestion- and loop-free network update in timed SDNs. IEEE Journal on Selected Areas in Communications, 2017, 35(11): 2542–2552. [doi: 10.1109/JSAC.2017.2760146]

9 Li P, Guo ST, Pan CS, *et al*. Fast congestion-free consistent flow forwarding rules update in software defined networking. Future Generation Computer Systems, 2019, 97: 743–754. [doi: 10.1016/j.future.2019.03.030]

10 周烨, 杨旭, 李勇, 等. 基于分类的软件定义网络流表更新一致性方案. 电子与信息学报, 2013, 35(7): 1746–1752.

11 Fu J, Sjodin P, Karlsson G. Loop-free updates of forwarding tables. IEEE Transactions on Network and Service Management, 2008, 5(1): 22–35. [doi: 10.1109/TNSM.2008.080103]

12 Mahajan R, Wattenhofer R. On consistent updates in software defined networks. Proceedings of the 12th ACM Workshop on Hot Topics in Networks. College Park: ACM, 2013. 1–7.

13 Dolati M, Khonsari A, Ghaderi M. Consistent SDN rule update with reduced number of scheduling rounds. Proceedings of the 2018 14th International Conference on Network and Service Management. Rome: IEEE, 2018. 254–260.

www.c-s-a.org.cn

© 中国科学院软件研究所 http://www.c-s-a.org.cn