软件定义网络中一种快速无循环路径迁移策略

燕昺昊[1]，刘勤让[1,2]，沈剑良[1]，汤先拓[1]，梁栋[1]
（1. 信息工程大学信息技术研究所，河南 郑州 450001；2. 国家数字交换系统工程技术研究中心，河南 郑州 450001）

摘 要：针对软件定义网络中数据平面转发设备的分布式部署及异步操作导致的路径迁移缓慢及故障等问题，提出了一种快速无循环路径迁移策略。首先，提出了一种基于节点排序的快速循环检测算法。该算法通过对比流的新旧路径上相邻节点的位置差异，可快速判定路径迁移过程中是否存在转发循环以及检测循环发生位置。然后，提出了一种基于节点松弛依赖关系的贪婪更新机制。该机制利用快速循环检测算法发掘出新旧路径上公有交换机之间存在的松弛依赖关系，进而保证了迁移过程每轮更新的交换机数量最大化。仿真实验结果表明，相比于现有迁移方案，所提策略在不同网络状态下均可有效避免迁移循环且获得最佳更新时间开销。

关键词：软件定义网络；路径迁移；循环避免；松弛依赖；贪婪机制
中图分类号：TP393
文献标志码：A
DOI: 10.11959/j.issn.1000−436x.2022088

Fast loop-free path migration strategy in software defined network

YAN Binghao[1], LIU Qinrang[1,2], SHEN Jianliang[1], TANG Xiantuo[1], LIANG Dong[1]
1. Institute of Information Technology, Information Engineering University, Zhengzhou 450001, China
2. National Digital Switching System Engineering & Technological R&D Center, Zhengzhou 450001, China

Abstract: Aiming at the problems of slow and faulty path migration caused by distributed deployment and asynchronous operation of data plane forwarding devices in software defined network, a fast loop-free path migration strategy was proposed. First, a fast loop detection algorithm based on node ranking was proposed. By comparing the position differences of adjacent nodes on the old and new paths of the flow, whether there was a forwarding loop in the path migration process and the location detection where the loop occurs could be quickly determined. Then, a greedy update mechanism based on node relaxation dependency was proposed. The fast loop detection algorithm was used to uncover the relaxation dependency between the common switches on the old and new paths, and the number of switches updated in each round of the migration process was ensured to be maximized. Simulation results show that the proposed strategy can effectively avoid migration loops and obtain the optimal update time overhead under different network states compared with existing migration schemes.

Keywords: software defined network, path migration, loop avoidance, relaxation dependency, greedy mechanism

0 引言

软件定义网络（SDN, software defined network）通过集中式的控制逻辑实现了对流量的细粒度管理[1]。运营商可利用 SDN 根据自身需求或网络状态随时调整策略部署而不需要考虑转发设备之间存在的硬件差异。因此，传统网络中需要复杂操作才可实现的路由优化、故障恢复等功能均可灵活实现。

路径迁移作为一种有效的性能优化机制及故

收稿日期：2021−12−17；**修回日期：**2022−03−17
基金项目：国家科技重大专项基金资助项目（No.2017ZX01030301）；工业互联网创新发展工程基金资助项目（No.TC190A446-2）
Foundation Items: The National Science and Technology Major Project of China (No.2017ZX01030301), Industrial Internet Innovation Development Foundation Project (No.TC190A446-2)

障恢复手段，在 SDN 管理中发挥了关键性作用[2]。网络中流的转发路径受突发情况影响无法时刻保持最佳传输状态。因此，为保证数据流正确传输，路径迁移技术通常为数据流同时规划多条转发路径。当原始转发路径发生拥塞或故障时，控制器将同时下发新的转发规则至新旧路径上的交换机，以引导数据流迁移至新路径传输。

然而，由于链路及设备状态差异，发往不同交换机的指令将经历不同的传输时延以及生效时间[3-4]。这种异步更新机制造成了流转发行为的不一致，导致包括更新循环、黑洞以及拥塞在内的一系列问题[5-6]，严重影响端到端服务质量，特别是在大规模网络环境下。其中，更新循环是路径迁移过程中频繁出现且易造成网络故障的关键问题之一，已经成为当前关注重点[2,7-8]。更新循环是指当迁移过程的新旧路径存在反向重叠时，不正确的节点更新顺序将导致数据包在节点之间往返，更新循环示例如图 1 所示。图 1 中，如果交换机 v_4 在交换机 v_3 之前更新，则将产生 $v_3 \rightarrow v_4 \rightarrow v_6 \rightarrow v_3$ 的循环。对于交换机(v_2,v_3)同理。这种循环行为将极大地消耗链路带宽资源，进而影响正常数据流的传输，严重时可导致链路瘫痪。

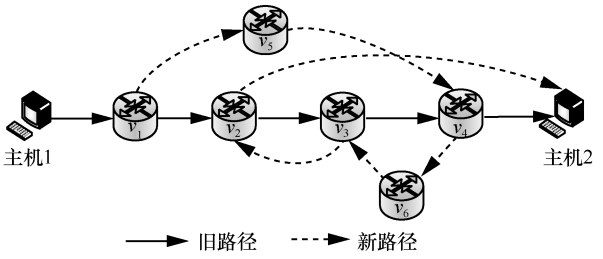


图 1　更新循环示例

为避免路径迁移过程出现转发循环，研究人员深入探究了循环产生的原因并提出了许多代表性的工作。最早提出的代表性方法称为两阶段提交法[6,9]。在流更新期间，具有不同匹配标识的新转发规则被发送到交换机来替代旧规则。新到的数据包因为在头字段中具有不同于旧数据包的标签，如虚拟局域网（VLAN, virtual local area network）可被新规则识别并转发。旧规则最终将由于超时被删除。然而，同时保持新旧规则在交换机中会降低存储空间的利用率，且需要付出一个无关的包头字段的代价。

为解决利用率的问题，Zhou 等[10]提出了倒序更新机制。当新旧路径存在反向重叠时，一个节点的更新需要保证其在旧路径的上游节点先被更新。然而，倒序更新本质上给网络施加了强一致性约束，可能会导致不必要的更新时间。同样考虑图 1 所示的拓扑，交换机 (v_2,v_3,v_4) 在新旧路径上处于反向重叠状态。因此，为满足循环更新依赖，倒序更新机制使交换机按照 $v_2 \rightarrow v_3 \rightarrow v_4$ 的顺序进行更新，更新时间为 $t(v_2+v_3+v_4)$。然而，本文通过观察发现，如果 v_2 先被更新，则 v_3 和 v_4 处于“孤立”状态，因为此时没有任何流通过 v_3 和 v_4。显然，在下一时刻，v_3 和 v_4 可被同时更新而不会造成循环。此时，更新时间可缩减为 $t(v_2+\max(v_3,v_4))$。

为加快更新过程，Wang 等[7]提出了基于分段的并行更新机制 Cupid。通过将新路径上的节点分割为不存在依赖关系的更新片段并在段内施行倒序更新，可在保证无转发循环的同时实现多节点并行更新。在此基础上，Li 等[11]进一步改进了 Cupid 并提出了新的更新方式 RU。RU 的优势在于其实现了对段内节点的同步更新，即保证段内的第一个循环节点最后更新，且多段可以同时并行更新。然而，尽管 2 种方法都使用了不同的约束机制加速了更新过程，但对于路径循环的检测依赖于求解有向无环图的强连通分量，时间复杂度为 $O(E+V)$。V 和 E 分别为新旧路径构成的有向无环图中节点和连边的数量。当待更新流的数量急剧增加时，时间开销将显著增加。

上述大部分工作只关注具体的实施方案及策略，而忽略了时间开销对路径迁移的重要性程度。快速的网络状态转移可有效减少数据包在交换机内的等待时长，进而节省交换机内部存储空间且避免数据包长时间等待而被超时丢弃。此外，即使部分工作涉及快速更新机制[7,11]，但并没有充分探索解空间，例如如何快速检测迁移过程中可能出现的循环问题。因此，在已有研究基础上，本文关注快速实现无循环路径迁移，包括检测和执行，以应对现有方案在实时性上的不足。具体地，本文提出了一种快速无循环路径迁移策略。该策略可同时缩短路径迁移中检测阶段和更新阶段的完成时间，进而实现整体迁移性能优化，提升网络收敛速度以及抗振荡能力。本文主要贡献如下。

1) 提出一种基于节点排序的快速循环检测（NRLD, node ranking-based loop detection）算法。NRLD 通过对新旧路径上的公共交换机进行编号并对比相邻交换机序号位置差异，实现了对路径

迁移过程中是否发生循环及循环发生位置的快速检测。

2) 提出并分析了路径迁移过程中待更新交换机之间存在的松弛依赖关系，并基于NRLD实现了松弛依赖关系的快速检测。

3) 提出一种基于节点松弛依赖关系的贪婪更新（RDGU, relaxation dependency-based greedy update）机制。基于获得的松弛依赖关系，RDGU机制构建贪婪更新机制以保证每轮中可同时更新的节点数量最大化。

4) 通过仿真实验验证了提出的快速无循环路径迁移策略。相比于已有路径迁移中的循环避免方案，所提策略在不同网络环境下均可显著降低时间开销，有效提升路径迁移效率。

1 网络模型

本节首先对网络进行建模并给出路径迁移的相关形式化定义，然后建立优化目标函数并分析约束条件。

本文提出了流更新方案基于SDN实现。在SDN中，位于控制平面的控制器C负责为每个新到达的流f计算转发路径p并发送相关的规则r至交换机v。对于数据平面，使用无向图$G=(V,E)$表示由交换机集合$V=\{v_i|i=1,\cdots,n\}$和交换机之间的链路集合$E=\{e_j|j=1,\cdots,m\}$构成的网络拓扑。给出如下定义来描述路径迁移。

定义 1 路径迁移。对于任意流f，其转发路径从具有相同起始点s_f和终点d_f的旧路径p_o迁移到新路径p_n。

本文分别使用有向序列$p_n=(v_1^n,v_2^n,\cdots,v_p^n)$和$p_o=(v_1^o,v_2^o,\cdots,v_q^o)$表示流$f$的新旧转发路径。显然，对于任意节点$v_i\in V$，如果其在新旧路径上满足$v_{i+1}^n=v_{i+1}^o$，即具有相同的相邻下一跳节点，则不会在该节点处产生循环。因此，将流f从旧路径p_o迁移至新路径p_n，只需要对新旧路径上具有不同下一跳的公共交换机$v_c\in\{v_p^n\bigcap v_q^o, v_{p+1}^n\neq v_{q+1}^o\}$上的规则进行更新。在本文中，更新指交换机上的旧规则替换为新规则。进一步，本文给出无循环更新的定义。

定义 2 无循环更新。对于p_o和p_n中所有公共节点构成的有向图G_d，任意节点$v_c\in G_d$的更新不得使流f经过的路径形成有向环，即流f不能经过同一节点两次。

设$x(f,v_c),v_c\in G_d$为二值变量，表示流f是否经过节点v_c。

$$x(f,v_c)=\begin{cases}1, & \text{流}f\text{经过节点}v_c\\ 0, & \text{流}f\text{未经过节点}v_c\end{cases} \tag{1}$$

因此，定义2可表示为如下约束

$$\mathrm{C}_1:\sum_f\sum_{v_c}x(f,v_c)=1,\ \forall v_c\in G_d \tag{2}$$

假设流f从旧路径p_o迁移到新路径p_n在$k(k\geqslant 1)$轮之内完成。因此期望找到一组迁移调度序列$U=\{u_1,u_2,\cdots,u_k\}$可以在满足定义2的前提下使k最小化。因此，首先给出如下优化目标函数

$$\min\sum_{i=1}^{k}\bar{u}_i \tag{3}$$

其中，$\bar{u}_i$为0-1变量，表示是否存在第i轮更新。本文假设控制器可同时更新多个交换机且控制信道时延和规则安装时延均为常数。显然，更少的更新轮数意味着更短的更新时间。除约束式(2)外，优化目标函数仍需要满足如下约束。

1) 对每轮中更新的交换机数量$|v_i|$至少为1且不超过待更新交换机的总数$|v_c|$。

$$\mathrm{C}_2:1\leqslant\bar{u}_i|v_i|\leqslant|v_c|,\ \forall v_i\in v_c,\ i\in[1,\cdots,k] \tag{4}$$

2) 对于任意节点v_i，进入交换机的流f只能匹配新规则r_n或旧规则r_o二者中的一个。

$$\mathrm{C}_3:m_{v_i}(f,r_n)+m_{v_i}(f,r_o)=1,\ \forall v_i\in v_c,\ i\in[1,\cdots,k] \tag{5}$$

3) 对于任意节点v_i，所有待更新流f所需新规则总数不得超过交换机剩余可用容量R_{v_i}。

$$\mathrm{C}_4:\sum_{\forall f\in F}m_{v_i}(f,r_n)\leqslant R_{v_i},\ \forall v_i\in v_c,\ i\in[1,\cdots,k] \tag{6}$$

4) 对于任意节点v_i，满足流守恒约束，即流入的流数量与流出的流数量相等。

$$\mathrm{C}_5:\sum_{\forall v_i\in v_c}f_{\text{in}}-\sum_{\forall v_i\in v_c}f_{\text{out}}=0 \tag{7}$$

此外，本文中假设所有交换机$v_i\in V$均由同一个控制器C进行管理。交换机之间的任意链路$e_j\in E$容量充足，不会导致拥塞。表1给出了网络模型相关参数及含义。

表 1　网络模型相关参数及含义

参数	含义
$G=(V,E)$	网络拓扑
$V=\{v_i \mid i=1,\cdots,n\}$	网络中交换机集合
$E=\{e_j \mid j=1,\cdots,m\}$	网络中链路集合
C	控制器
f	流
$p_o=(v_1^o,v_2^o,\cdots,v_q^o)$	旧路径
$p_n=(v_1^n,v_2^n,\cdots,v_p^n)$	新路径
$v_c \in \{v_p^n \cap v_q^o, v_{p+1}^n \neq v_{q+1}^o\}$	新旧路径公共交换机
$U=\{u_1,u_2,\cdots,u_k\}$	更新轮数
R_{v_i}	交换机剩余可用容量

2　算法设计

2.1　概述

本文提出的快速无循环路径迁移策略主要由基于节点排序的快速循环检测算法以及基于节点松弛依赖关系的贪婪更新机制构成，且均部署于控制器端。对于每个需要进行路径迁移的流，控制器首先调用循环检测算法判断其新旧路径是否可能发生循环。若不满足发生循环的条件，则直接将新规则下发至相应的交换机来实施路径迁移；否则，再次调用循环检测算法来发掘节点之间存在的松弛依赖关系。基于松弛依赖关系，控制器调用贪婪更新模块获得路径迁移更新序列并分步下发规则至相应的交换机。快速无循环路径迁移流程如图 2 所示。

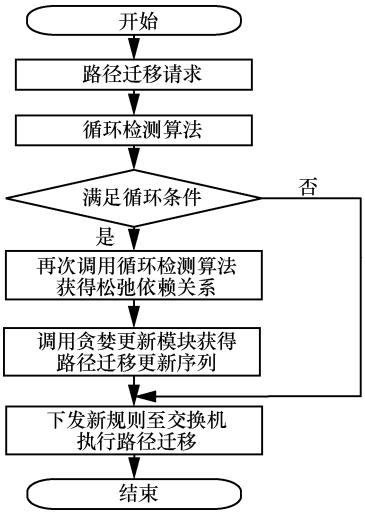


图 2　快速无循环路径迁移流程

2.2　基于节点排序的快速循环检测算法

在实施路径迁移时，需要以有效的方式来检测迁移过程是否会发生循环，以避免数据包对链路带宽的无效重复占用。现有方案[7,11]均通过构建新旧路径之间的有向图并求解强连通分量来实现循环检测，多流处理时间开销较高。而本文所提快速循环检测算法 NRLD 仅需通过对比相邻节点在新旧路径上的位置关系即可实现对循环的检测。检测步骤如下。

步骤 1　NRLD 按照流在旧路径上的传输顺序对各节点依次编号，获得节点编号集合 $\mathrm{seq\_}p_o$，即

$$\mathrm{seq\_}p_o \leftarrow \left\{ v_c^o = k \middle| v_c \in \left\{ v_p^n \cap v_q^o, v_{p+1}^n \neq v_{q+1}^o \right\} \right\} \quad (8)$$

步骤 2　NRLD 根据新路径上流传输顺序获得新的编号顺序集合 $\mathrm{seq\_}p_n$，即

$$\mathrm{seq\_}p_n \leftarrow \left\{ v_c^n[k] = v_c^o[k] \right\} \quad (9)$$

步骤 3　最后通过逐项比较新路径编号序列 $\mathrm{seq\_}p_n$ 中相邻节点编号值的大小即可在完成检测的同时输出循环位置 $\mathrm{loop\_loc}$，即

$$\begin{cases} v_c^n[k] > v_c^n[k+1], \text{节点之间存在循环} \\ v_c^n[k] < v_c^n[k+1], \text{节点之间不存在循环} \end{cases} \quad (10)$$

$$\mathrm{loop\_loc} \leftarrow \left\{ \left(v_c^n[k], v_c^n[k+1] \right) \middle| \exists v_c^n[k] > v_c^n[k+1] \right\} \quad (11)$$

基于节点排序的快速循环检测算法示例如图 3 所示。图 3 中，新旧路径拓扑一共包含 7 个公共节点，起始节点为 v_1，终止节点为 v_7。首先根据步骤 1 获得旧路径节点编号 $\mathrm{seq\_}p_o$ 为[1-7]。然后根据步骤 2 获得新路径相对应的节点编号 $\mathrm{seq\_}p_n$ 为[1,4,3,2,6,5,7]。最后，通过对比 $\mathrm{seq\_}p_n$ 中相邻节点的编号大小，可知新旧路径存在循环且可能发生循环的位置为[4,3,2]和[6,5]。

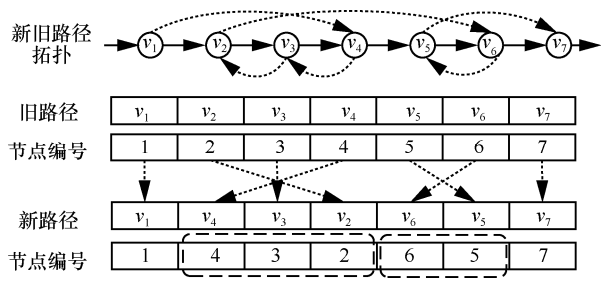


图 3　基于节点排序的快速循环检测算法示例

NRLD 伪代码如算法 1 所示。

算法 1　基于节点排序的快速循环检测算法

输入　新路径 p_n，旧路径 p_o

输出　节点循环关系 $\mathrm{loop\_loc}$

1) for each $v_i \in p_n$:
2) if $v_i \in p_o$:
3) $v_c \leftarrow v_i$ // 获得新旧路径公共节点
4) end if
5) end for
6) for each $v_i \in v_c \bigcap p_o$:
7) $\text{seq}\_p_o \leftarrow \text{num}(v_i)$ // 计算旧路径节点编号
8) $\text{num}(v_{i+1}) = \text{num}(v_i) + 1$
9) end for
10) for each $v_j \in v_c \bigcap p_n$:
11) if $p_n(v_j) = p_o(v_i)$
12) $\text{seq}\_p_n(v_j) \leftarrow \text{seq}\_p_o(v_i)$ // 获得新路径节点编号
13) end if
14) end for
15) for each $v_j \in \text{seq}\_p_n(v_j)$
16) if $\text{seq}\_p_n(v_{j+1}) < \text{seq}\_p_n(v_j)$
17) $\text{loop}\_\text{loc} \leftarrow \left(v_c^n[k], v_c^n[k+1]\right)$ // 获得可能发生循环的节点位置
18) end if
19) end for
20) 返回 loop_loc

2.3 基于节点排序的松弛依赖关系检测

为便于理解，在介绍具体算法之前先给出节点依赖关系相关定义并证明相关结论。

定义 3 节点依赖性。如果节点 $v_n \in v_c$ 在节点 $v_m \in v_c$ 之前更新，将导致 (v_n, v_m) 之间产生循环，称节点 v_n 依赖于节点 v_m，表示为 $v_n < v_m$。

已有方法通过搜索有向图中的强连通分量可找到节点之间满足的依赖关系。需要说明的是，求解强连通分量获得的是节点之间的强依赖关系。强依赖关系定义如下。

定义 4 强依赖关系。在任意更新轮 u_i 中，节点 v_n 必须在节点 v_m 之前完成更新以避免循环，称 v_n 和 v_m 之间满足强依赖关系。

强依赖关系表现在有向图中即存在从节点 v_n 指向节点 v_m 的与旧路径 p_o 相反的有向边。以图 1 为例，通过求解可知，$[v_2, v_3, v_4]$ 为新旧路径组成的有向图中的一个强连通分量。因此，根据定义 4，节点集 $[v_2, v_3, v_4]$ 须按照顺序 $v_4 \to v_3 \to v_2$ 逐个进行更新。现有工作已经证明了根据强依赖关系进行节点更新可以避免循环[7]。

事实上，尽管基于强连通分量以及强依赖关系进行节点更新保证了严格避免循环，但更新时间因此将同时受到有向图规模以及依赖链长度的严重影响。同时，通过观察发现，在一定条件下，满足强依赖关系的节点可同时更新而不造成循环。如图 1 所示，如果交换机 v_1 先更新将消除 $[v_2, v_3]$ 之间存在的强依赖关系而使之可以实现同时更新。本文称这种更新关系为松弛依赖关系，定义如下。

定义 5 松弛依赖关系。在新旧路径组成的有向图 $\boldsymbol{G}_d$ 中，节点 v_n 的更新将使节点集 $V_r = \{v_i \mid i = n+1, \cdots, m\}$ 中任意节点所具有的强依赖关系消失，称 V_r 中节点之间满足松弛依赖关系。其中，v_n 称为松弛依赖触发点，v_{m+1} 称为松弛依赖终止点。根据定义 5 可得以下结论。

结论 1 松弛依赖触发点 v_n 与其在新路径 p_n 和旧路径 p_o 上的后继节点具有相同排列顺序。

证明 假设松弛依赖触发点 v_n 与其在新路径 p_n 和旧路径 p_o 上的后继节点具有不同排列顺序，即在有向图中存在从节点 v_n 出发的具有相反方向的 2 条有向边，如图 4 所示。此时，节点 v_n 与其在旧路径上的直接祖先节点 v_{n-1}^{op} 形成强连通分量。根据定义 3，此时 v_n 的更新依赖于 v_{n-1}^{op}，即 v_{n-1}^{op} 将取代 v_n 成为新的松弛依赖触发点。可知与前提假设矛盾，因此结论 1 成立。证毕。

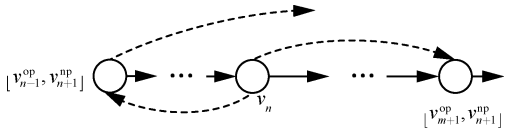


图 4 结论 1 参考示意

结论 2 由 v_n 触发松弛依赖的节点集 V_r 必包含在 v_n 与其在新路径上的下一跳节点 v_{n+1}^{np} 之间。

证明 对于依赖触发点 v_n，根据结论 1 可知其不依赖于任何祖先节点 $v_i^{\text{op}} (0 \leqslant i \leqslant n-1)$，即 v_n 的更新并不影响 v_i^{op}。因此，节点集 V_r 中任意节点均为 v_n 后继节点 $v_i^{\text{op}} (n < i)$。对于 v_n 在新路径上的下一跳节点 v_{n+1}^{np}，其后继节点 v_{n+2}^{np} 存在 2 种情况，如图 5 所示。情况 1，v_{n+2}^{np} 为 v_{n+1}^{np} 在旧路径上的祖先节点。显然，v_{n+2}^{np} 只有为 v_n 在旧路径上的后继节点才可被包括在节点集 V_r 中。情况 2，v_{n+2}^{np} 为 v_{n+1}^{np} 在旧路径上

的后继节点。此时，根据结论 1 可知，v_{n+1}^{np} 将成为新的触发点。综上所述，由 v_n 触发松弛依赖的节点集 V_r 必包含在 v_n 与其在新路径上的下一跳节点 v_{n+1}^{np} 之间，即结论 2 成立。证毕。

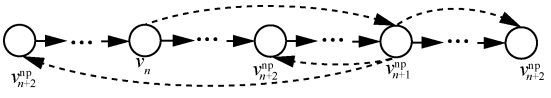


图 5　结论 2 参考示意

结论 3　松弛依赖节点集 V_r 内节点可同时更新且避免循环。

证明　由转发唯一性（式(5)约束 3）可知，松弛依赖触发点 v_n 在同一时刻只能保留一条输出边用于流量转发。因此，v_n 在 t 时刻的更新将导致 v_n 与其在旧路径 p_o 上下一跳节点 v_{n+1}^{op} 之间不再存在有向边，如图 6 所示。再由流守恒约束（式(7)约束 5）可得，在 v_n 更新后，v_n 与其在新路径上的下一跳节点 v_{n+1}^{np} 之间的所有节点之间均不存在组成旧路径 p_o 的有向边。显然，此时 v_n 与 v_{n+1}^{np} 之间的节点可以同时更新而不违反循环依赖。同时根据结论 2 可知，v_n 与 v_{n+1}^{np} 之间的节点即满足松弛依赖的节点集 V_r，故结论 3 成立。证毕。

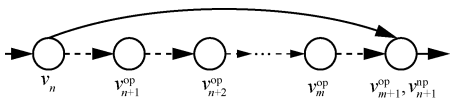


图 6　结论 3 参考示意

根据上述结论可知，并非所有待更新节点均可同时满足松弛依赖更新条件。因此，如何快速找到满足松弛依赖的节点集 V_r 成为进一步所关注的内容。通过分析发现，基于 NRLD 同样可实现对松弛依赖关系的快速检测。不同之处在于，NRLD 步骤 3 中的相邻节点大小关系需进一步约束。

$$\begin{cases} v_c^n[k+1]-v_c^n[k] \geqslant 2, \text{存在松弛依赖关系} \\ v_c^n[k+1]-v_c^n[k] < 2, \text{不存在松弛依赖关系} \end{cases} \tag{12}$$

此时，满足松弛依赖的节点集为

$$V_r \leftarrow \left\{ v_c^o[k+i] \middle| i=1,\cdots,\ v_c^n[k+1]-v_c^n[k] \right\} \tag{13}$$

对于图 3，新路径 p_n 相邻节点编号[1, 4]和[2, 6]满足式(12)给出顺序递增关系。因此可知旧路径 p_o 节点集$[v_2,v_3]$和$[v_3,v_4,v_5]$即满足松弛依赖的节点集。

结合已有结论，给出并证明如下定理来说明算法有效性。

定理 1　基于 NRLD 获得的松弛依赖关系进行路径迁移不产生任何循环。

证明　由 NRLD 的查找方式可得，新路径 p_n 上可导出松弛依赖关系的相邻节点对 $\left[v_c^n[k],v_c^n[k+1]\right]$ 必满足 $v_c^n[k]<v_c^n[k+1]$。因此，$v_c[k]$ 和 $v_c[k+1]$ 在新旧路径中具有相同的排列顺序。由结论 1 可知，$v_c[k]$ 为松弛依赖触发点。同时，由于有向图 G_d 中只包括新旧路径公共节点，因此在旧路径上，$v_c[k]$ 和 $v_c[k+1]$ 之间必包含其他公共节点 $v_c^o[i]\in \left[v_c^o[k],v_c^o[k+1]\right]$。因此，根据结论 2 和结论 3 可知，由公共节点 $v_c^o[i]$ 组成的节点集即满足松弛依赖的节点集 V_r，且在路径迁移过程中同时更新 V_r 中节点不会产生任何循环。证毕。

由于松弛依赖关系检测与 NRLD 仅在步骤 3 的判别条件上存在差异，因此不再重复给出 NRLD 伪代码。

2.4　贪婪更新算法

节点之间的松弛依赖关系本质上表征了节点实施并行更新的局部约束条件。因此，对松弛依赖集内的节点进行同步更新只能满足部分优化需求。为了进一步减少整体更新时间，需要提出一种更高效的更新调度机制，在每轮操作中更新更多的节点以减少控制器与交换机交互的次数，从而符合式(1)给出的目标函数。考虑 2 种不同的更新方式。

方式 1　在每轮更新中选择可并行更新的最大数量松弛节点集 $\sum_{\forall i\neq j, V_i\cap V_j=0} V_r$。

方式 2　在每轮更新中选择具有最多松弛节点的集合 $\max(|V_r|)$。

然而，通过分析网络拓扑结构发现，单独执行二者中任意一个都无法满足优化目标。对于方式 1，可能并不存在可并行更新的松弛节点集。如图 7(a)所示，节点对 (v_1,v_4) 和 (v_2,v_5) 可分别导出松弛依赖集$[v_2,v_3]$和$[v_3,v_4]$。显然，二者存在重叠部分，即公共更新节点 v_3。因此，触发点 v_1 和 v_2 无法同时进行更新以导出松弛依赖集。原因在于，如果 v_2 先完成更新，此时流的转发路径为 $v_1\rightarrow v_2\rightarrow v_5$。节点 v_3 和 v_4 上的转发规则可能会因为在有效时间内没有匹配数据包被超时删除。因此，当 v_1 更新后，流的转发路径 $v_1\rightarrow v_4\rightarrow v_5$ 就会成为断路而导致丢包。

对于方式 2，最长松弛依赖链 $\max(|V_r|)$ 则无法

保证在单一轮中更新最多的节点。如图 8(a)所示，节点 v_4 可导出松弛依赖集 $[v_5,v_6,v_7,v_8]$。此时，可同时更新的节点数为 4。然而，由方式 1 可知，节点 v_1 和 v_7 导出的松弛依赖集不存在重叠，因此可同时更新。此时，可同时更新的松弛依赖集为 $[v_2,v_3,v_4]$ 和 $[v_8,v_9]$，且可同时更新的节点数为 5。

因此，结合上述示例，本文提出了贪婪更新机制 RDGU。在每轮更新开始之前，RDGU 首先检测是否存在无重叠松弛依赖集 $V_i \bigcap V_j=\varnothing$。若存在，则根据可并行更新的松弛依赖集 $\sum_{\forall i\neq j, V_i\cap V_j=0} |V_r|$ 中节点数与 $\max(|V_r|)$ 中节点数的比较结果选择不同方式进行更新。

$$\begin{cases} \sum\limits_{\forall i\neq j, V_i\cap V_j=0} |V_r| > \max(|V_r|), \text{ 按照方式1更新} \\ \sum\limits_{\forall i\neq j, V_i\cap V_j=0} |V_r| \leqslant \max(|V_r|), \text{ 按照方式2更新} \end{cases} \tag{14}$$

其中，方式 1 更新流程如图 7 所示，方式 2 更新流程如图 8 所示。当存在多个包含相同数量节点的 V_r 时，任意选择其中一个进行更新。

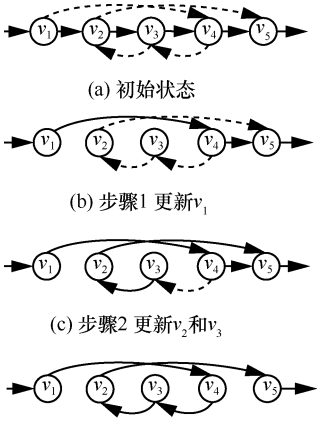


图 7 方式 1 的更新流程

本文给出并证明如下定理来说明RDGU的有效性。

定理 2 无重叠松弛依赖集中节点更新是相互独立的。

证明 假设松弛依赖集 V_r^1 中交换机 v_i 的更新依赖于松弛依赖集 V_r^2 中的交换机 v_j。根据定义 3 可知，v_i 在 v_j 之前更新将导致循环。根据定义 1 可知，v_i 更新将产生 $\left\{v_i^{\text{np}} \to v_{i+1}^{\text{np}}\right\} \in V_r^1$ 之间的新路径。显然，$v_{i+1}^{\text{np}} \notin V_r^2$ 且 $\left\{v_i^{\text{np}} \to v_{i+1}^{\text{np}}\right\} \notin V_r^2$。因此，$V_r^2$ 中任意节点的转发路径没有发生改变且循环没有发生在 V_r^2 中。同时，根据结论 3 可知，松弛依赖集 V_r^1 中的节点更新不产生循环。综上所述，假设不成立，即定理 2 成立。证毕。

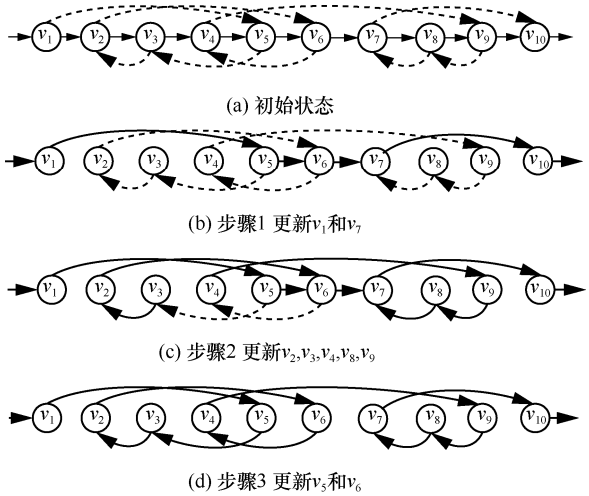


图 8 方式 2 的更新流程

定理 3 基于 RDGU 算法进行节点更新不产生任何循环。

证明 在每轮更新中，无论是对于方式 1 还是方式 2，RDGU 所选择的更新节点可分为松弛依赖触发点 v_n、松弛依赖节点集 V_r 以及松弛依赖终止点 v_{m+1} 三类，且更新顺序 $v_{m+1} < V_r < v_n$。对于松弛依赖触发点 v_n，根据结论 1 可知其更新不产生任何循环；对于松弛依赖节点集 V_r，根据结论 3 可知其更新同样不产生任何循环；对于松弛依赖终止点 v_{m+1}，其更新依赖于在新路径 p_n 上的下一跳节点 v_{m+2}^{np}。当 v_{m+1} 与 v_{m+2}^{np} 在新路径上的顺序关系与旧路径相同时，根据结论 1 可得，此时 v_{m+1} 同时为松弛依赖触发点，因此其更新不产生循环；当 v_{m+1} 与 v_{m+2}^{np} 在新路径上的顺序关系与旧路径相反时，此时根据定义 3，有 $v_{m+1} < v_{m+2}^{\text{np}}$。由于 v_{m+1} 为在最终轮中完成更新的节点，因此此时 v_{m+2}^{np} 必已完成更新。显然，v_{m+1} 更新不再导致循环。综上所述各种情况，可得定理 3 成立。证毕。

RDGU 伪代码如算法 2 所示。

算法 2 基于松弛依赖关系的贪婪更新机制

输入 松弛依赖关系节点集 $V_R = \bigcup_{i=1}^{|V_r|} V_{r+1}$

输出 节点更新序列 V_u

初始化 $V_u=\varnothing$，$V_r^{\max}=\max|V_r|$，$r=1$

1) while $V_R \neq \varnothing$ & $r < |V_R|$

2) if $V_r \bigcap V_{r+1}=\varnothing$ & $|V_r|+|V_{r+1}| \geqslant V_r^{\max}$ //判断

是否存在满足更新条件的并行松弛依赖集

3)　$V_u \leftarrow \{V_r, V_{r+1}\}$ // 生成更新序列

4)　$V_R = V_R - \{V_r, V_{r+1}\}$ //更新 V_R

5)　else

6)　$V_r^{\max} = \max\{V_r, V_{r+1}, \cdots, V_{r+|V_r|}\}$ //选择具有最多节点的松弛依赖集

7)　$V_u \leftarrow V_r^{\max}$

8)　$V_r = V_r - V_r^{\max}$

9)　$r = r + 1$

10)　end if

11)　end while

12)　返回 V_u

2.5　时间复杂度分析

由于循环检测算法以及松弛依赖关系检测算法具有相同的执行过程，因此二者具有相同的时间复杂度。其中，节点编号可以 $O(1)$的时间复杂度完成，相邻节点编号对比的时间复杂度为 $O(n)$，n 为新旧路径公共节点数量。因此，检测过程总的时间复杂度为 $O(n)$。相比于求解强连通分量，检测过程不再需要遍历边。

对于贪婪更新机制，由于一个具有 n 个公共节点的有向图至少包括 2 个松弛依赖集合，因此判断松弛依赖节点集中是否存在无重叠集合的时间复杂度为 $O(n^2)$。对于节点更新过程，最坏情况下需要 n 轮完成更新，即每次只更新单一节点。因此其时间复杂度为 $O(n)$。综上可知，提出的贪婪算法的总体时间复杂度为 $O(n^2)$，且整体路径迁移算法的时间复杂度为 $O(n^2+n) \sim O(n^2)$ 。

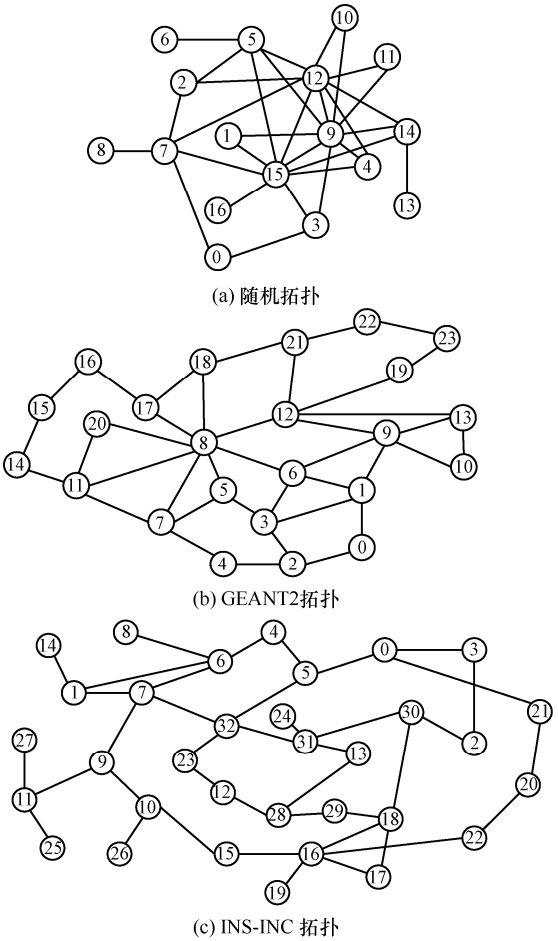
3　实验及分析

本节通过仿真实验验证了所提路径迁移机制的有效性并与已有的代表性方案进行了对比。

3.1　实验设置

实验环境。本文所有算法均在 Ubuntu 18.04 LTS 环境中使用 Python 3.8 编程实现。硬件配置为 AMD Ryzen 7-4800H 处理器，16 GB 内存。

网络拓扑。使用了一个随机拓扑和取自 The Internet Topology Zoo[12]2 个不同规模的真实网络拓扑来评估提出的算法：1) 随机拓扑，包括 17 个节点和 31 条边；2) GEANT2 拓扑，包括 24 个节点和 38 条边；3) INS-INC 拓扑，包括 33 个节点和 40 条边。实验拓扑如图 9 所示。



(a) 随机拓扑

(b) GEANT2拓扑

(c) INS-INC 拓扑

图 9　实验拓扑

路径生成。从拓扑中随机选择节点来模拟流转发的源节点和目的节点。然后计算并选择节点之间的任意一条简单路径（不存在重复节点的路径）作为流转发的旧路径。进一步，随机排列旧路径上除源节点和目的节点之外的其他节点来构造出存在转发循环的新路径。

实验参数。考虑到典型的真实网络拓扑平均直径约为 $\log(n)$ [13]，结合计算开销及实际情况（如生成树协议计时器的默认值将最大网络直径限制为 7），实验中最大公共节点数量设为 $\lfloor 2\log(n) \rfloor$，规则基准更新时间设为 0.5 ms。同时考虑到规则更新时间受交换机性能如流表占用率、队列长度等指标影响，因此将该时间乘以[0,1)内的随机数来模拟真实规则更新时间。控制器到交换机的信道时延为 0.5 ms。每组实验运行 10 次取平均值保证无偏性。

3.2　循环检测算法性能对比及分析

首先验证所提 NRLD 算法与已有算法在不同条件下对于循环的检测时间差异。为保证公平性，所有算法结果被转换为相同输出形式。对比算法如下。

深度优先搜索（DFS, depth-first search）。从源节点开始对新旧路径形成的有向图进行遍历。

Tarjan[7,11]。通过寻找强连通分量，发现有向图中存在的环路。

不同的循环检测算法在不同更新路径数量下的循环检测时间对比如图 10 所示，其中新旧路径的公共节点数设置为 6，包括源节点和目的节点。

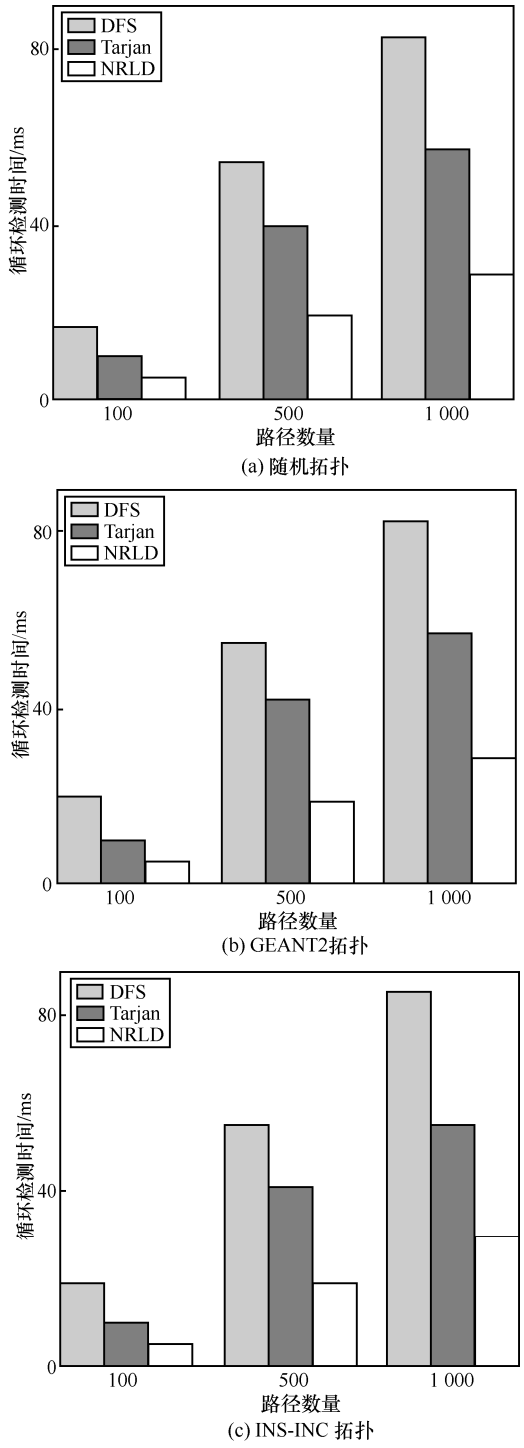


图 10　不同更新路径数量下的循环检测时间对比

从图 10 中可以看出，NRLD 在不同拓扑上均取得了最佳性能。这里以 GEANT2 拓扑上路径数为 500 时的结果为例来进行分析。对于 DFS 和 Tarjan，NRLD 可将检测时间分别减少 64.58%和 55.81%。原因在于，一方面，NRLD 是通过对比相邻节点的序号来进行循环检测的，避免了 DFS 和 Tarjan 在循环检测过程中基于堆栈的节点逐项对比操作。另一方面，DFS 对有向图遍历的输出结果为有向边的形式，需要更多时间将输出结果处理为节点之间的依赖关系。而 Tarjan 所涉及的退栈以及修改时间戳操作影响了其检测性能。

不同的循环检测算法在不同公共节点数量下的循环检测时间对比如图 11 所示，其中路径数量设为 500。根据前文所述，实验中随机拓扑、GEANT2 拓扑和 INS-INC 拓扑的最大公共节点分别为 7、9、10。从图 11 中可以看出，在给定的节点数量范围内，NRLD 在 3 个不同的拓扑上均获得最小的检测时间。特别地，当公共节点数量越多时，NRLD 优势更加明显。例如，在 GEANT2 拓扑上，相比于 DFS 和 Tarjan，当节点数量为 6 时，NRLD 将检测时间分别减少了 34.41 ms 和 19.35 ms；当节点数量为 9 时，NRLD 将检测时间分别减少了 82.98 ms 和 29.89 ms。产生这一现象的主要原因是当公共节点数量增多时，新旧路径之间形成循环的数量将同步增加，这将导致 DFS 和 Tarjan 产生更多的入栈以及退栈操作。而每次入栈后的节点逐项对比又进一步增加了时间开销。注意到，3 种算法的检测时间增长均呈现非线性趋势。这也对具有低时延需求的相关路径优化方案（如多路径转发）以及故障恢复策略带来了一定的启发，即新旧路径应尽可能地避免重叠。

3.3　整体方案性能对比及分析

本节进一步验证提出的路径迁移策略整体性能与已有迁移更新方案在多项指标上的差异。对比方案如下所示。

CCG[10]。对满足循环依赖的节点逐个进行倒序更新。

Cupid[7]/FCFU[2]。将节点集分割为不存在依赖关系的更新片段并在段内施行倒序更新。

RU[11]。在 Cupid/ FCFU 基础上，段内的第一个循环节点最后更新，其余节点同步更新。

首先在不考虑循环检测时间的情况下，验证了提出的基于贪婪算法的更新机制相比于上述方案的优势。各方案在不同公共节点数量下的平均归一

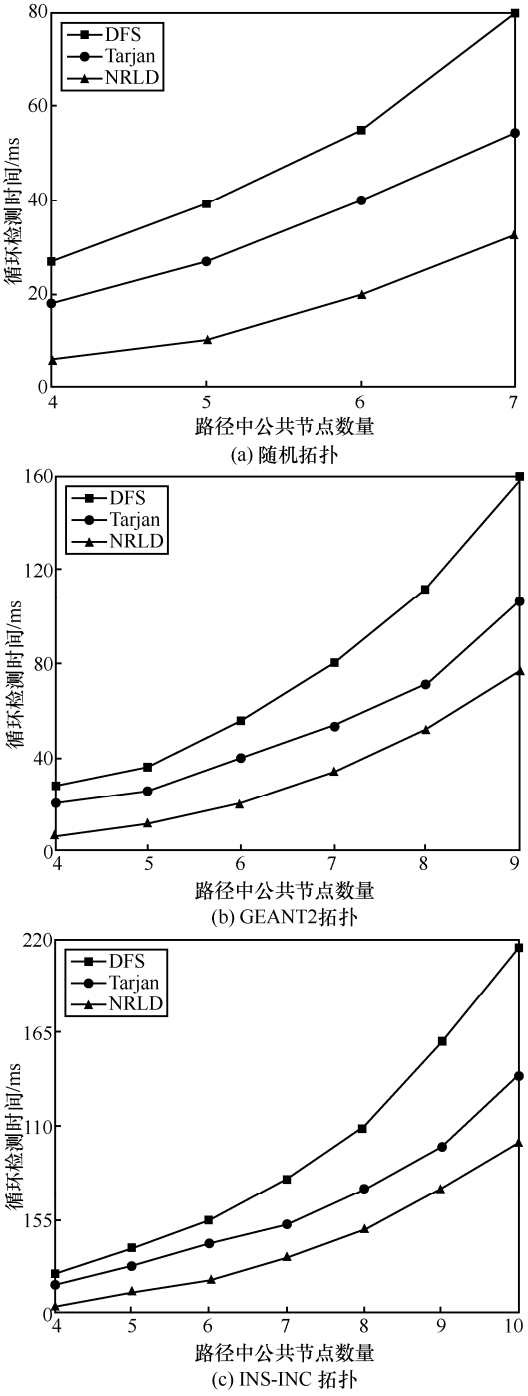


图 11　不同公共节点数量下的循环检测时间对比

化路径迁移更新时间对比如图 12 所示，其中更新路径数量设为 500。为更好地体现方案之间的性能差异，将实验结果以 CCG 为基准进行归一化。当公共节点数量为 4 时，所有算法的更新时间相同。这是因为此时新旧路径只存在单一形式的循环，并不满足松弛依赖以及并行更新执行的条件。而当公共节点数量逐渐增加时，所提更新机制的优势开始显现，而且随着节点数量的增加，其性能优势更加明显。例如，在 GEANT2 拓扑上，相比于其他方案，所提贪婪更新机制在公共节点数为 6 时分别降低了 17.5%和 7.5%（Cupid/FCFU 和 RU 在此时的时间开销是相同的）的时间开销。而当公共节点数为 9 时分别降低了 54.8%、17.8%和 14.7%。这是因为当公共节点数量较多时，新旧路径之间开始出现较长的松弛依赖节点集以及并行集。这些集合使所提算法可以在更少的轮数内完成节点更新。

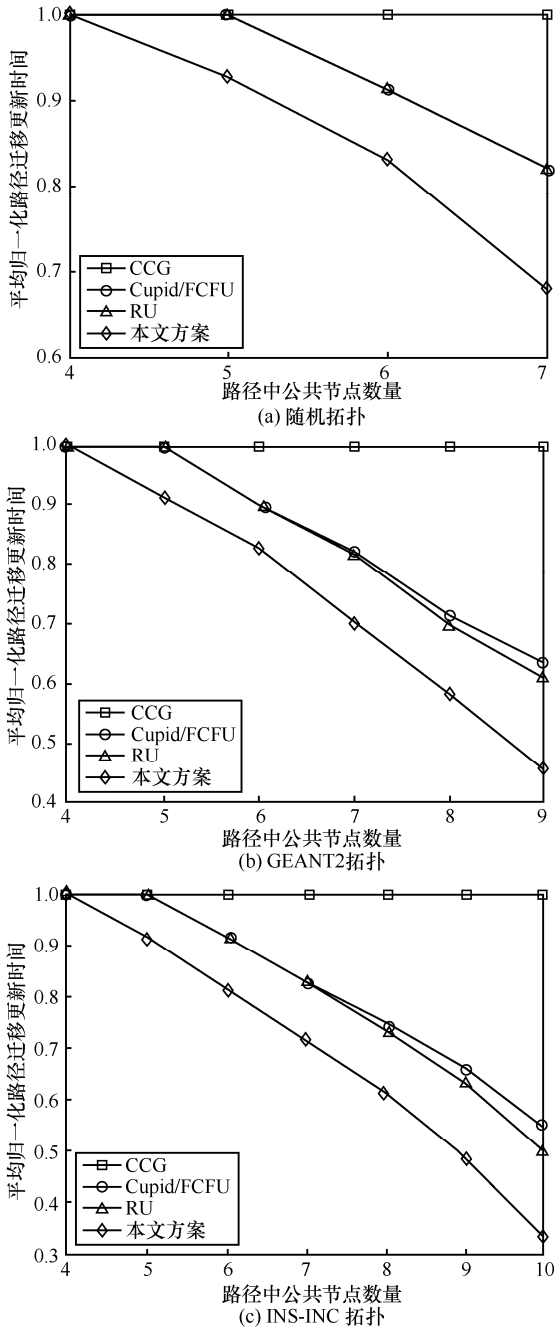


图 12　不同公共节点数量下的平均归一化路径迁移更新时间对比（更新路径数为 500）

需要说明的是，当公共节点数量不超过 7 时（图 12(a)所示），Cupid/FCFU 和 RU 两类算法的更新时间是一致的。这是因为此时新旧路径形成的循环中的更新片段并不存在可同步更新的节点。因此 RU 算法退化为与 Cupid/FCFU 算法一致。而当公共节点数量超过 7 时，RU 算法展现出了一定的优势。

进一步，各方案在不同更新完成时间下的路径迁移更新完成率如图 13 所示。

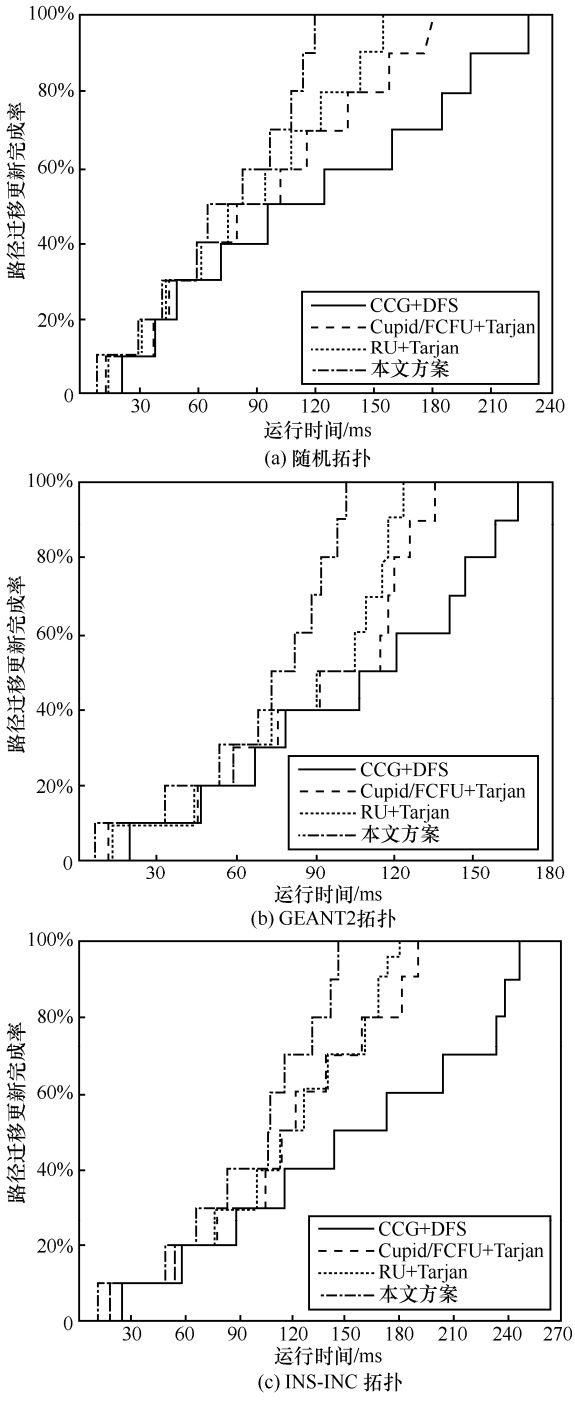


图 13　不同更新完成时间下的路径迁移更新完成率（更新路径数为 100）

对于 CCG 算法，使用了 DFS 来检测其循环。而 Cupid/FCFU 和 RU 则按照原文献描述均使用了 Tarjan 来求解强连通分量。对于本文所提出的贪婪更新机制，则使用 NRLD 来检测节点松弛依赖关系。实验中更新路径数量设定为 100 且其中公共节点数的比例如表 2 所示。同时，规定单一交换机上规则更新是逐个进行的。从图 13 中可以看出，算法运行前期 4 种方案在不同拓扑上的路径迁移完成率相差不大。随着运行时间的增加，本文方案展现出了明显的优势。这主要是因为在运行前期完成更新的均为较短路径，即公共节点数较少的路径。因此 4 种方案的性能差异仅表现在循环检测时间上。在运行后期，长路径的迁移需要 CCG、Cupid/FCFU 和 RU 运行更多的轮数。而本文方案得益于长路径节点之间普遍存在的松弛依赖关系以及贪婪更新机制，因此可在同一轮中更新更多的节点。更少的更新轮数意味着更少的更新时间。同时，该实验结果也表明，当网络面临由于链路故障而导致的大量路径并发迁移需求时，本文更新机制同样也可具有快速收敛性能。这是由于一方面，本文机制并不依赖于网络状态，而只与网络规模有关；另一方面，不同流之间的迁移过程在循环检测和更新策略的计算中并不存在依赖关系。因此迁移过程类似于流水线操作，先完成迁移计算的流可以先执行迁移。

表 2　　3 种拓扑中不同公共节点的路径比例

公共节点数	随机拓扑	GEANT2 拓扑	INS-INC 拓扑
4	10%	5%	5%
5	30%	10%	10%
6	40%	20%	15%
7	20%	25%	20%
8	—	25%	25%
9	—	15%	20%
10	—	—	5%

4　结束语

本文对软件定义网络架构下的流路径迁移缓慢及故障等问题进行了研究，提出了一种快速无循环路径迁移策略。该策略首先基于节点排序实现了路径循环的快速检测。在此基础上，通过发掘节点之间的松弛依赖关系并构建贪婪更新机制，保证了

节点更新过程的快速实施。仿真结果表明，相比于现有路径迁移机制，本文所提策略在保证路径迁移无循环的前提下，实现了更快的检测及迁移速度，且在不同拓扑及网络条件下均具有良好的稳定性。即使面临网络链路故障导致的大量并发路径迁移操作，同样具有较好的收敛性。

参考文献：

[1] YAN B H, LIU Q R, SHEN J L, et al. A survey of low-latency transmission strategies in software defined networking[J]. Computer Science Review, 2021, 40: 100386.

[2] 史久根, 杨旭, 刘雅丽, 等. 软件定义网络中快速和一致的流更新策略[J]. 电子与信息学报, 2021, 43(9): 2617-2623.
SHI J G, YANG X, LIU Y L, et al. Fast and consistent flow update in software defined network[J]. Journal of Electronics & Information Technology, 2021, 43(9): 2617-2623.

[3] 苏建忠, 张华宇, 朱海龙. 结合 SDN 控制器的 TSN 周期性带宽预留值计算方法[J]. 通信学报, 2021, 42(10): 23-31.
SU J Z, ZHANG H Y, ZHU H L. Computing method for periodic stream reservation in TSN combined with SDN controller[J]. Journal on Communications, 2021, 42(10): 23-31.

[4] 汪硕, 黄玉栋, 黄韬, 等. 基于软件定义的时间敏感网络跨域调度机制[J]. 通信学报, 2021, 42(10): 1-9.
WANG S, HUANG Y D, HUANG T, et al. Software-defined cross-domain scheduling mechanism for time-sensitive networking[J]. Journal on Communications, 2021, 42(10): 1-9.

[5] FOERSTER K T, SCHMID S, VISSICCHIO S. Survey of consistent software-defined network updates[J]. IEEE Communications Surveys & Tutorials, 2019, 21(2): 1435-1461.

[6] WU G H, GAO X F, ZHENG J Q, et al. Achieving fast loop-free updates with ingress port in software-defined networks[J]. IEEE/ACM Transactions on Networking, 2021, 29(4): 1527-1539.

[7] WANG W, HE W B, SU J S, et al. Cupid: congestion-free consistent data plane update in software defined networks[C]//Proceedings of the 35th Annual IEEE International Conference on Computer Communications. Piscataway: IEEE Press, 2016: 1-9.

[8] SONG H F, GUO S T, LI P, et al. FCNR: fast and consistent network reconfiguration with low latency for SDN[J]. Computer Networks, 2021, 193: 108113.

[9] VISSICCHIO S, CITTADINI L. FLIP the (flow) table: fast lightweight policy-preserving SDN updates[C]//Proceedings of IEEE INFOCOM 2016-the 35th Annual IEEE International Conference on Computer Communications. Piscataway: IEEE Press, 2016: 1-9.

[10] ZHOU W, JIN D, CROFT J, et al. Enforcing customizable consistency properties in software-defined networks[C]//Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI'15). Berkeley: USENIX Association, 2015: 73-85.

[11] LI P, GUO S T, PAN C S, et al. Fast congestion-free consistent flow forwarding rules update in software defined networking[J]. Future Generation Computer Systems, 2019, 97: 743-754.

[12] KNIGHT S, NGUYEN H X, FALKNER N, et al. The Internet topology zoo[J]. IEEE Journal on Selected Areas in Communications, 2011, 29(9): 1765-1775.

[13] RUSEK K, SUÁREZ-VARELA J, MESTRES A, et al. Unveiling the potential of graph neural networks for network modeling and optimization in SDN[C]//Proceedings of the 2019 ACM Symposium on SDN Research. New York: ACM Press, 2019: 140-151.

[作者简介]

燕昺昊（1994− ），男，山西吕梁人，信息工程大学信息技术研究所博士生，主要研究方向为软件定义网络、低时延通信等。

刘勤让（1975− ），男，河南商丘人，博士，信息工程大学信息技术研究所研究员、博士生导师，主要研究方向为新一代网络体系结构、时间敏感网络等。

沈剑良（1982− ），男，浙江德清人，博士，信息工程大学信息技术研究所副研究员，主要研究方向为新一代网络信息系统架构设计、大规模集成电路设计等。

汤先拓（1984− ），男，湖南长沙人，博士，信息工程大学信息技术研究所副研究员，主要研究方向为新型网络架构、高性能片上网络设计等。

梁栋（1992− ），男，河南郑州人，信息工程大学信息技术研究所博士生，主要研究方向为软件定义网络、网络故障恢复等。