基于启发式调度的 OpenFlow 网络规则一致更新方案

刘 艺[1,2],张红旗[1,2],杨英杰[1,2]
(1. 信息工程大学,河南郑州 450001;2. 河南省信息安全重点实验室,河南郑州 450001)

摘 要: 针对 OpenFlow 网络在状态转换过程中会暂时性出现转发回路、路由黑洞和违反访问控制策略等问题,提出了一种基于启发式调度的规则一致更新方案. 首先,设计基于谓词的更新分解算法,利用并行网络属性验证技术得到子更新依赖图;其次,采用任务图生成算法对子更新依赖图进行分割,降低更新调度的复杂度;之后,设计启发式更新调度算法,采用规则增删操作交替执行策略,减少交换机的规则存储开销,并通过建立更新实施和监听并发执行机制,提升更新效率. 仿真实验从更新时间开销和更新过程中交换机规则存储开销两方面验证了方案的有效性.

关键词: OpenFlow 网络; 规则一致更新; 启发式调度
中图分类号: TP393.08 **文献标识码:** A **文章编号:** 0372-2112 (2017)07-1637-09
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2017.07.013

Consistent Rule Update Scheme Based on Heuristic Scheduling for OpenFlow Networks

LIU Yi[1,2], ZHANG Hong-qi[1,2], YANG Ying-jie[1,2]
(1. *Information Engineering University, Zhengzhou, Henan* 450001, *China*;
2. *Henan Key Laboratory of Information Security, Zhengzhou, Henan* 450001, *China*)

Abstract: In view of such problems as temporary loops, blackholes, violations of access control policy and so on during state transitions in OpenFlow networks, we proposed a consistent rule update scheme based on heuristic scheduling. First, we divided operations of an update into some sub-updates and used parallel network property verification technique to construct a dependency graph for each sub-update. Second, we aggregated several update operations of the same sub-update to reduce complexity of scheduling. Then, we designed heuristic update scheduling algorithm. By adopting an alternative strategy for additions and deletions of rules, it could reduce storage cost in switches. Furthermore, by establishing a mechanism to update and monitor concurrently, it could improve efficiency of rule update. Simulation experiments on the updating time cost and the rule storage cost of switches during updating process are conducted, which verify the effectiveness of our scheme.

Key words: OpenFlow networks; consistent rule update; heuristic scheduling

1 引言

软件定义网络[1](Software-Defined Networking, SDN)是一种逻辑控制和数据转发分离的新型网络架构,基于 OpenFlow[2] 实现 SDN 已成为主流趋势[3]. 当流量均衡和 VM 迁移等网络更新事件发生时,需要以更新交换机中规则的方式来实现网络状态转换. 然而,随着网络更新事件愈加频繁出现,网络状态转换过程成为重要的不稳定因素,会引起暂时性的转发路径成环、丢包和安全脆弱性[4]等问题,因此,如何保证网络在状态变化过程中保持无转发回路、无路由黑洞和不违反访问控制策略等网络属性,即规则一致更新问题[5]亟待解决.

OpenFlow 网络的规则一致更新问题是由多交换机更新不具备原子性造成的[3],一种解决方法是降低控制器与交换机之间的时延. 其中,DIFANE[6] 和 Devo-

收稿日期:2016-01-26;修回日期:2016-06-27;责任编辑:李勇锋
基金项目:国家863高技术研究发展计划(No. 2012AA012704);郑州市科技领军人才(No. 131PLJRC644)

Flow[7]在 OpenFlow 交换机上增加部分控制功能,防止规则安装时延,但这与 SDN 控制转发分离的原则相悖;文献[8]引入平均时延和最坏时延分析控制器的部署数目和位置,但仅适用于交换机不多的网络. 另一种解决方法是协调多个交换机上规则更新操作的执行顺序. 其中,文献[9]首次提出并证明了若更新方案满足每包一致性或每流一致性,则网络能在状态转换过程中保持所有的网络属性;文献[10]基于平行配置思想提出了两阶段更新方案以满足每包一致性,但会使交换机的规则数目暂时性加倍;为此,文献[11]通过将更新过程分解为 k 次子更新,以权衡更新时间和交换机存储空间,但最优 k 值求解耗时长;文献[12]通过构建中间规则和利用控制器缓存无法识别新旧规则的数据包来保证每包一致性,但大大增加了控制器负载;文献[13]指出存在着保持无环路等单个或若干网络属性、但不满足每包一致性的更新方案,并首次提出需要考虑优化更新方案的实际执行速度;基于此,文献[14]根据交换机负载和链路流量等实际网络状况动态调度规则更新操作,但它只适用于精确匹配规则,且需要为不同的网络属性设计不同的算法.

因此,针对现有研究存在更新耗时长、交换机负载大和方案通用性差等问题,提出了基于启发式调度的 OpenFlow 网络规则一致更新方案 HS-CRU. 通过设计基于谓词的更新分解算法,将全局更新划分为多个相对独立的子更新,为灵活地调度执行规则更新操作奠定基础. 通过采用并行的网络属性验证技术生成更新依赖图,提高更新方案的通用性. 此外,在将更新依赖图转化为任务图的基础上,综合衡量更新时间和交换机规则存储开销以计算任务的优先级,并根据网络实时状况进行动态调整;同时由于调度问题本身是 NP 完全问题[15],设计启发式更新调度算法,进一步缩短更新时间.

2 基于启发式调度的规则一致更新方案

HS-CRU 的整体架构如图 1 所示.

在子更新构建阶段,更新依赖图构建器将规则更新操作划分为若干子更新,并根据期望的网络属性,采用网络属性验证技术为每个子更新生成更新依赖图;在子更新调度阶段,更新调度器首先将各子更新依赖图转化为任务图,之后通过收集网络数据平面信息,根据更新目标,计算任务图中各任务的优先级,并采用启发式更新调度算法执行任务,完成规则一致更新.

2.1 子更新构建

一般地,规则 $r:=(P,s,a,pri)$ 决定数据包的转发路径,其中 P 是谓词,包括源/目的 IP 地址等字段,代表一个数据包集合. 在本文中若 P 的各字段值范围都包含数据包 p 的相应字段值,则 $p\in P$;s 是规则所处交换机的标识;a 是动作,包括转发和修改等;pri 是优先级. 规则更新操作 $op:=(o,r)$ 是影响交换机中规则存储的实际命令,其中,o 是操作,包括增加、删除和修改规则(OpenFlow 标准[16]中"修改规则"不会改变规则谓词);r 是规则.

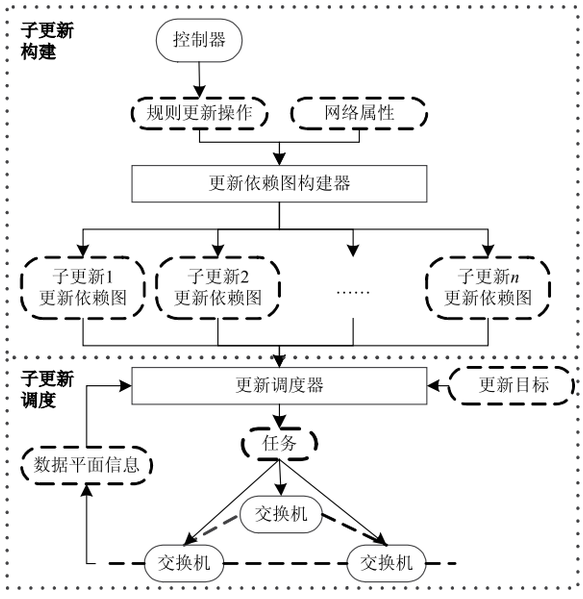

图1 基于启发式调度的规则一致更新方案

2.1.1 基于谓词的更新分解算法

依据规则更新操作对应的规则,采用基于谓词的更新分解算法(即算法 1)将它们划为若干子更新,每个子更新以谓词 P 标识,表示它会影响 P 中数据包的转发路径. 子更新 P 中的规则更新操作满足:$\forall op.r,P$ 包含的数据包在网络中转发时匹配规则 r. 在算法 1 中,P_i 是子更新标识,显然,只要集合 $\{P_1,P_2,\cdots,P_n\}$ 满足——对于任一规则更新操作 op,总能找到至少一个 $P_i\in\{P_1,P_2,\cdots,P_n\}$ 使得 P_i 中的数据包转发时匹配 $op.r$——这一条件即可,为简单起见,本文以子网为依据划分子更新,即 $P_i:=(IP_{dst}=w_i)$,w_i 是子网地址;s_{in} 是入口交换机;OP 是规则更新操作集合;T 是网络拓扑;R_{old} 和 R_{new} 分别是更新前后的规则集合. $Packetclass(P,s,R,T)$ 表示 P 中数据包从交换机 s 进入网络 T 后经规则集 R 转发得到的数据包集合 $pktset$,特别地,OpenFlow 标准定义规则的修改动作可以改变数据包包头的源/目的 IP 地址和端口等字段,因此可能 $\exists p\in pktset,p\notin P$.

算法 1 基于谓词的更新分解算法

Input:$P_i,s_{in},OP,T,R_{old},R_{new}$;
Output:$ruleset$.

ruleset←∅;
pktset←*Packetclass*(P_i,s_{in},R_{old},T) ∪ *Packetclass*(P_i,s_{in},R_{new},T) ;
foreach {s^* | $s^* \in S$}　/* S 是交换机集合 */
　temp←*pktset* ;
　foreach {r | $r.s = s^*$,$r = op.r$ 且 $op \in OP$}
/* OP 是规则更新操作集合,按优先级降序遍历 */
　　if ∃$p \in temp$,$p \in r.P$ then
　　ruleset←*ruleset* ∪ {r} ;
　　temp←*temp* − {p} ;
　　end if
　end for
end for
return *ruleset*.

2.1.2　更新依赖图构建方法

借鉴文献[5]中将计算规则更新顺序问题转换为验证问题的思想,本文开发了基于 MapReduce 的网络属性并行验证技术[17],用于为每个子更新制定满足期望网络属性的规则更新操作顺序. 同时,为便于进行子更新调度,本文以规则更新操作为节点,它们之间的依赖关系为边构造有向的更新依赖图 $UG = (UV, UE)$. 其中,UV 是更新操作节点的集合,$W(uv)$ 是 $uv \in UV$ 对应的规则更新操作所属子更新数目;UE 是更新操作节点间边的集合,$C(uv_i, uv_j)$ 表示边 $<uv_i, uv_j> \in UE$ 的权值,它等于完成 uv_i 所需时间 t_i,若 uv_j 没有后继节点,则将其用权值为 t_j 的边连到一个伪节点 uv_k 上,$W(uv_k) = 0$. 规则更新操作完成时间受网络链路时延、交换机规则安装操作耗时等影响,可在规则更新前从控制器处获得. 入度为 0 的节点(即 $uv.indegree = 0$)称为入口节点,出度为 0 的节点(即 $uv.outdegree = 0$)称为出口节点.

规则更新操作之间存在两种依赖关系:内部依赖,同一子更新的规则更新操作之间的依赖关系;外部依赖,不同子更新的规则更新操作之间的依赖关系,这是由不同子更新包含同一规则更新操作引起,称该条共有的规则更新操作为焦点操作,其对应的节点为焦点. 焦点集合 FN 满足:$\forall uv \in FN, W(uv) \geqslant 2$. 显然,若子更新依赖图中 $FN = \varnothing$,但具有多个入口(出口)节点,则可以将它们用权值为 0 的边连到一个伪入口(出口)节点 uv_{in}(uv_{out})上,$W(uv_{in}) = 0$($W(uv_{out}) = 0$),此时该图等同于 AOE 网,可以计算出完成该子更新的规则更新所需时间.

2.2　子更新调度

随着更新事件愈加频繁发生,网络状态转换过程耗时长会影响网络性能,如延长拥塞时间等. 此外,一旦 OpenFlow 交换机中规则数目超出限度,交换机会拒绝安装更多的新规则,导致网络转发出错[18]. 因此,规则一致更新方案需要对规则更新操作进行合理调度,以缩短更新时间和降低更新过程中交换机规则存储开销.

由于规则更新操作数目众多,且不同子更新间和同一子更新内都存在可并发执行的操作,以单个操作为单位进行调度十分复杂,此外,外部依赖关系使得一个子更新的更新过程会出现中断现象,以整个子更新为单位进行调度难以提高并发度. 因此,在子更新调度阶段,本文首先设计任务图生成算法,将更新依赖图中的节点聚合为若干子图,使得一个子图中或者不存在焦点,或者只存在一个节点,且该节点是焦点,并以子图为单位生成有向的任务图 $TG = (TV, TE)$,其中,TV 是任务节点的集合,$tv \in TV$ 对应一个具有唯一标识的子图,$M(tv)$ 是完成 tv 对应子图中所有规则更新操作的时间;TE 是任务节点间边的集合,$<tv_i, tv_j> \in TE$ 表示只有在完成任务 tv_i 后才能开始任务 tv_j. 易知,任务节点数目远小于规则更新操作数目,且单个任务节点中的所有规则更新操作可以完整执行. 之后,提出启发式更新调度算法,以动态可变的优先级标识任务的调度顺序,并使任务尽可能并发执行.

2.2.1　任务图生成算法

为便于叙述,在更新依赖图中定义两类节点集合:关键节点集合 KN = {uv | 或者 $uv.indegree \geqslant 2$,或者 $W(uv) \geqslant 2$},终点集合 EN = {uv | 或者 $uv.outdegree = 0$,或者 $uv \in KN$}.

在算法 2,$UG^s = (UV^s, UE^s)$ 是 $UG = (UV, UE)$ 的子图,其满足条件:$s \in UV^s$,且对于任意 $uv \in UV^s$,至少存在一条路径$(s, uv_1, \cdots, uv_m, uv)$,或者$\{uv_1, \cdots, uv_m\} = \varnothing$,或者$\{uv_1, \cdots, uv_m\} \cap KN = \varnothing$. 记 $UG(V)$ 是以 V 为节点集的 UG 的子图,*Node_gen*($UG(V)$,*flag*)表示生成与 $UG(V)$ 对应,且以 *flag* 唯一标识的任务节点 tv. *Edge_gen*(uv_i, uv_j)表示生成边 $<uv_i, uv_j>$. *Delete*(UG, V)表示从图 UG 中删除 V 中的节点和所有与其相连的边. *Sub*(UG, V)表示从图 UG 中提取包含 V 中所有节点的子图. *Conb*(UG_1, UG_2)表示合并 UG_1 和 UG_2. *DFS_Path*(UG^s, s, s^*)返回从 s 到 s^* 的路径,$\{seqn^s \mid seqn^s = \{s, uv_1, \cdots, uv_m, s^*\}, <uv_i, uv_{i+1}> \in UE^s, 1 \leqslant i \leqslant m-1\}$.

算法 2　任务图生成算法

Input:UG,KN,EN;
Output:TG.
foreach {$s \mid s \in UV$,且 $s.indegree = 0$}
　if $s \notin KN$ then *Generate*(s) ;
　else tv = *Node_gen*(UG^s({s}),{s,s}) ;
　　foreach {$uv \mid uv \in Succ(s)$,且 $uv \in UV^s$}
　　UG_{gen} = *Generate*(uv) ;
　　/* 以 UG_{gen} 代指函数 *Generate* 的返回值 */
　　Edge_gen(tv_{ss},*Sub*(UG_{gen},{uv})) ;
　　end for

```
        Delete(UG,{s});
      end if
end for
Generate(s)
{visited = ∅;
foreach {s* | s* ∈ EN∩UV^s}
  if (s* ∈ KN) then
    tv = Node_gen(UG^s({s*}),(s*,s*));
    foreach {seqn^s | seqn^s ∈ DFS_Path(UG^s,s,s*)}
      visited = visited∪seqn^s;
      if tv_ss* exists then
        temp = Node_gen(UG^s(seqn^s - {s*}),(s,s*));
        Conb(tv_ss*,temp);
      else
        tv = Node_gen(UG^s(seqn^s - {s*}),(s,s*));
        Edge_gen(tv_ss*,tv_s*s*);
      end if
    end for
    else if (s*.outdegree = 0) then
      seqn^s = DFS_Path(UG^s,s,s*);
      tv = Node_gen(UG^s(seqn^s),(s,s*));
      visited = visited∪seqn^s;
    end if
end for
Delete(UG,visited-KN);
return UG_gen.} /* 由前述算法生成的节点和边构成 */
```

为了降低任务图的存储空间,可将具有相同节点的更新依赖图子图进行合并.

2.2.2 启发式更新调度算法

文献[5]提出贪心最大并行算法提高下发规则更新操作的并行程度,但其缺少从整体上对更新过程的调度规划,并且没有考虑更新过程中交换机的规则存储开销,因此,以任务图为基础,本文提出启发式更新调度算法,其核心思想是:在确定任务优先级时,从更新时间的角度,关键路径上的节点在一定程度上决定了整个更新的执行速度[15],需要得到优先调度,从交换机规则存储开销的角度,以交换机更新前后的规则数目最大值为基准值,需要尽可能使当前存储的规则数目不大于基准值,或比基准值超出的数目小;在实际执行任务时,因为不同任务涉及的交换机可能不一样,所以通过尽可能并行执行多个任务,进一步缩短更新时间.

(1)任务优先级定义

记 $fac(tv_i)$ 为任务 tv_i 的优先级,定义如下:

$$fac(tv_i)=\alpha\cdot T^*_{tv_i}+\beta\cdot S^*_{tv_i} \tag{1}$$

$$T_{tv_i}=\frac{1}{M(tv_i)+\max\limits_{tv_j\in Succ(tv_i)}(M(tv_j))} \tag{2}$$

$$T^*_{tv_i}=\frac{T_{tv_i}-\min(T_{tv})}{\max(T_{tv})-\min(T_{tv})} \tag{3}$$

$$S_{tv_i}=\sum_s(\max(current_s+\Delta^s_{tv_i}-target_s,0))^2 \tag{4}$$

$$S^*_{tv_i}=\frac{S_{tv_i}-\min(S_{tv})}{\max(S_{tv})-\min(S_{tv})} \tag{5}$$

其中,$T^*_{tv_i}$ 是时间开销调度因子,由 T_{tv_i} 经极差标准化后得到,以 $T^*_{tv_i}$ 升序进行调度蕴含着优先调度关键路径上的节点;$S^*_{tv_i}$ 是存储开销调度因子,由 S_{tv_i} 经极差标准化后得到,$current_s$ 是交换机 s 当前存储的规则数目,$\Delta^s_{tv_i}$ 是实施任务 tv_i 后 s 的规则数目改变量,$target_s=\max(R^s_{\text{old}},R^s_{\text{new}})$ 是 s 在更新前和更新后存储的规则数目最大值,此外,取平方值是为了保证更新过程中不会出现负载过大的交换机,以 $S^*_{tv_i}$ 升序进行调度蕴含着优先调度使全网交换机的规则存储开销较低的任务;α 和 β 是调度因子权值,满足 $\alpha>0,\beta>0$,且 $\alpha+\beta=1$,由规则一致更新方案的目标侧重点确定. 易知,$fac(tv_i)$ 越小,任务越先得到调度执行,且其可以在调度过程中动态改变.

(2)启发式调度算法描述

启发式调度算法分为任务下发子算法和任务完成监听子算法,分别运行在下发模块和监听模块,二者之间通过发送交互消息和修改共享变量集合的方式,共同完成任务调度. 其中,交互消息缓存在消息池 *mess_pool* 中;共享变量集合包括三类任务节点集合,分别为:*RN* 表示就绪的任务节点集合,满足:$\forall tv\in RN, tv.indegree=0$,并且 *tv* 按 $fac(tv)$ 由小到大排列;*DN* 表示当前正在下发的任务节点集合;*EN* 表示调度过程中无法执行的任务节点集合. 此外,为保证下发模块和监听模块之间的信息交互不出现混乱,分别为消息池和共享变量集合设置互斥信号量 *m_mutex* 和 *s_mutex*,且其初始值均为 1.

算法 3 任务下发子算法

```
Initialising:
1:  DN←∅
2:  RN←{tv | tv.indegree = 0}
3:  EN←∅
Repeat:
4:  wait(s_mutex)
5:      foreach tv_i ∈ RN - DN
6:        if IS({tv_i}) ∩ IS(DN) = ∅ then
7:            DN←DN∪{tv_i}
8:            dispatching tv_i
9:        end if
10:     end for
11: signal(s_mutex)
12: wait(m_mutex)
13:     foreach messages in mess_pool
14:       if there exists Mess(upd_error(tv_k)) then
```

```
15:          Mess(Alarm)
16:          update TG   /* 更新任务图 */
17:          end if
18:       end for
19:  signal(m_mutex)
20:  until RN = DN = ∅
21:  if EN≠∅ then Mess(Incomplete)
22:  else Mess(Complete)
```

算法 3 的第 1 – 3 步是初始化下发模块和监听模块的共享变量. 在第 4 – 11 步,$IS(\{tv_1, tv_2, \cdots, tv_n\})$ 表示参与完成任务集 $\{tv_1, tv_2, \cdots, tv_n\}$ 中规则更新操作的交换机集合,对于当前就绪且未处于调度过程中的任务,按照优先级降序调度,并且如果两个任务涉及的交换机集合无交集,则可被同时调度. 在算法的第 12 – 19 步,如果收到任务 tv_k 执行出错消息,则通知管理员更新过程出错,并第 16 步删除任务图中的节点 tv_k 及其后继节点集 $Succ(tv_k)$. 在第 20 步,读取当前 *RN* 和 *DN* 的值,若当前无就绪任务且无处于调度过程中任务,则结束调度,否则继续调度. 在算法的第 21 – 22 步,在任务下发过程结束后,根据 *EN* 的值向管理员报告更新是否完全执行.

算法 4　任务完成监听子算法

```
Repeat:
1:wait(m_mutex)
2:  wait(s_mutex)
3:    foreach messages in tc_pool
4:     if tv_i is confirmed then
5:       DN←DN∪{tv_i}
6:       update RN
7:         sending Mess(upd_RN) and Mess(upd_DN) to mess_pool
8:     end if
9:     if tv_j is time_out then
10:      DN←DN∪{tv_j}
11:      sending Mess(upd_DN) to mess_pool
12:    end if
13:    if tv_k cannot be applied then
14:      EN←EN∪{tv_k}
15:      DN←DN − {tv_k}
16:      sending Mess(upd_error(tv_k)) to mess_pool
17:    end if
18:  end for
19:  signal(s_mutex)
20:signal(m_mutex)
21:until RN = DN = ∅
```

算法 4 的第 3 步,*tc_pool* 中存储着从交换机处收集到的任务完成消息. 在第 4 – 17 步,监听模块分三种情况向下发模块报告任务执行情况:若任务 tv_i 完成,则将其从当前正在调度的任务节点集合中删除,更新就绪任务集合,重新计算任务优先级,并通知下发模块 *DN* 和 *RN* 已更新;若任务 tv_j 执行超时,则只更新 *DN*,即 tv_j 仍处于就绪状态,可以被重新调度;若任务 tv_k 下发次数超过阈值,为避免浪费资源,则标识其为不可执行,并通知下发模块.

2.3　时间复杂度分析

子更新构建阶段的时间开销主要分析划分一个子更新过程,假设交换机数目为 s,每个交换机的平均规则数目为 k,规则更新操作数目为 m,网络直径(即数据包到达目的地前经过的交换机数目最大值)为 d,由于一个数据包集合在网络转发过程中可能被分割,则需要 $O(dk^2)$ 时间得到一个数据包集合从进入到离开网络过程中能生成的数据包[19],此时划分一个子更新的时间复杂度为 $O(dk^2 + sm)$. 子更新调度阶段的时间开销主要包括任务图生成和启发式调度两部分. 在任务图生成阶段,假设更新依赖图的节点总数为 v,节点的平均出边数为 e,由于生成任务节点的主要操作是对更新依赖图节点进行深度优先搜索,以生成一个任务节点为例,在最坏情况下需要搜索所有边,时间复杂度为 $O(ve)$;在启发式调度阶段,假设任务总数为 t,任务重发阈值为 l,在最坏情况下所有任务的下发次数都超过阈值,时间复杂度为 $O(tl)$.

2.4　更新依赖图存在环路情况分析

若更新依赖图 *UG* 中存在环路,则记其强连通分量集合为 $SCC(UG) = \{SCC_1, SCC_1, \cdots, SCC_n\}$,其中 $SCC_i = (SV_i, SE_i)$,可由 Kosaraju 算法等获得. 对于每个强连通分量 SCC_i,如果 $|SV_i| > 1$,那么以聚集节点 cv_i 代替,$W(cv_i) = \max(W(sv))$,$sv \in SV_i$,并且保持原有的节点依赖关系不变. 对前述规则一致更新方案做以下改变:

①在子更新构建阶段,采用暂时性去环操作将聚集节点中的有环图转换为无环图,其基本思想是以子图中心性[20]刻画节点对所在图的局部子图的参与程度,优先执行核心节点操作可以尽快消除环路. 之后,根据无环图易得到其完成时间. 此种方式不会对更新依赖图其它部分造成影响.

②在子更新调度阶段,将聚集节点转换为单个任务节点,当调度 cv_i 时,若 $W(cv_i) \leqslant k^*$,则按照①中的无环图执行,此时网络中会出现暂时性不满足期望网络属性的现象,但是由于优先执行核心节点,完成 cv_i 所需更新时间较短,并且可通过降低 cv_i 所涉及子更新的数据包发送速度,减少错误转发的数据包数,从而降低网络错误的影响程度;反之,若 $W(cv_i) > k^*$,则采用两阶段更新方法完成 cv_i 中的规则更新操作,此种方式将参与两阶段更新的规则限制在聚集节点内,减少了等待时长. 此外,k^* 的值取决于网络管理员对规则一致更新

的严格程度.

3 仿真实验与分析

3.1 实验环境

基于 Mininet 仿真平台,实验利用 Floodlight 控制器和 OpenVSwitch 交换机构建网络,其中,控制器上运行路由程序和负载均衡程序;网络拓扑信息如表 1 所示. 网络属性验证器采用 MRVeri[17]. 记文献[10][11][14]和[5]所提方案分别为方案 1、2、3 和 4. 此外,记控制器与交换机之间的时延为 CS_D,指的是从控制器下发规则更新操作到交换机完成流表更新之间的时间间隔,受到物理链路的传输时延和交换机性能等的影响,通过改变 CS_D 值模拟数据中心和广域网等网络环境[5].

表 1 交换机与主机数目表

网络编号	网络 1	网络 2
网络拓扑结构	Fattree	Waxman
核心交换机数目	4	24
聚集交换机数目	8	
边界交换机数目	8	
主机数目	16	576

3.2 实验结果分析

为评估 HS-CRU 的有效性,仿真实验从更新时间开销、交换机规则存储开销和链路过载数据量三方面展开.

3.2.1 更新时间开销实验

实验在网络 1 中通过设置 CS_D 服从均值为 6ms,方差为 4ms 的正态分布来模拟动态的数据中心网络环境,要求网络在更新过程中无路由回路,并按照方案 3 的要求配置每条规则至多匹配一个数据包集合(即流[14]). 首先仅启用 1 个核心交换机,30 秒后启用余下的 3 个核心交换机,分别采用 HS-CRU 和方案 1-4 更新规则,其中 HS-CRU1 中取 $\alpha=1$、$\beta=0$,HS-CRU2 中取 $\alpha=\beta=0.5$,二者都取 $k^*=0$,方案 2 中取 $k=3$,记录各流的更新时间,重复进行 10 次实验,图 2 为更新时间 CDF 图.

HS-CRU1 完成 50% 和 99% 更新的时间分别比方案 1 约快 56% 和 59%,HS-CRU2 完成 50% 和 99% 更新的时间分别比方案 1 约快 25% 和 33%,原因在于方案 1 需要等待最大网络时延才能删除旧规则以完成更新,而 HS-CRU 允许数据包同时被新旧规则处理,并且尽可能并行执行规则更新操作. 同时,方案 2 的更新时间开销最大,这是因为它的每轮子更新仍采用两阶段更新方法. 此外,方案 3 的时间开销没有明显降低,原因是它需要耗费比当前平均网络时延更多的时间来更新调度

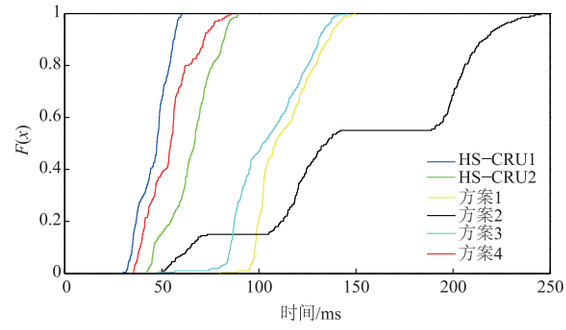

图2 更新时间CDF图

图. HS-CRU1 比方案 4 的时间开销小,这是因为前者在构建更新依赖图时采用 MapReduce 模型和增量式验证算法提高验证速度,并在调度时优先下发关键路径的规则更新操作,而后者验证缓存中操作的顺序是随机的,对验证效率有很大的影响. HS-CRU2 比 HS-CRU1 和方案 4 的时间开销略大,原因在于它额外需要平衡交换机规则存储开销,但其性能仍比较好.

为进一步比较各方案,实验改变 CS_D 值满足的正态分布参数来模拟不同网络环境,如表 2 所示,记录完成 50% 和 99% 更新的时间开销,结果如图 3 和图 4 所示.

表 2 CS_D 值服从的正态分布参数表

网络环境	DC_1		DC_2		WAN_1		WAN_2	
	μ	σ^2	μ	σ^2	μ	σ^2	μ	σ^2
CS_D/ms	6	0	6	4	100	0	100	25

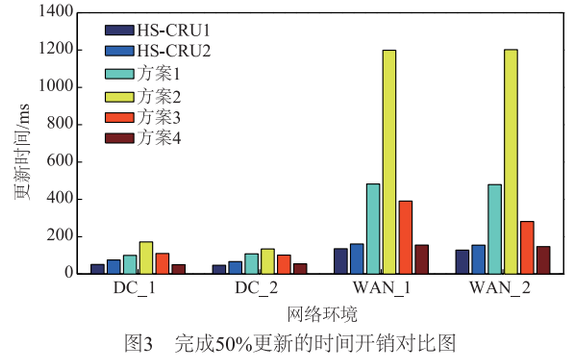

图3 完成50%更新的时间开销对比图

HS-CRU1、HS-CRU2 和方案 4 普遍优于其它方案,而且 CS_D 值的变化量对方案 3 的更新时间开销影响大,这是因为它依赖于网络环境的动态性.

此外,实验在网络 2 中配置匹配多条流的规则,比较 HS-CRU3(取 $\alpha=1$,$\beta=0$,$k^*=2$)、方案 1 和方案 4 的性能. 首先启用所有交换机,之后每隔 10 分钟随机启闭若干交换机和链路,记录规则数目的变化,选取 3 个时间窗口绘制结果图,如图 5 所示. 与方案 1 和方案 4 相比,HS-CRU3 的更新时间开销和交换机规则存储开销

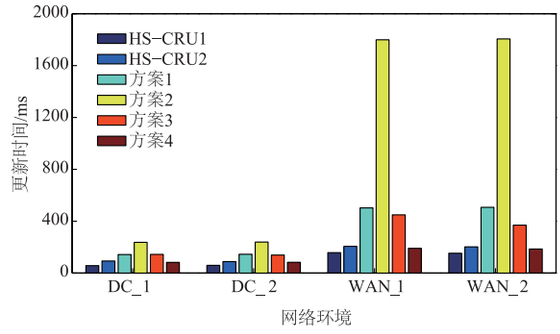

图4　完成99%更新的时间开销对比图

较低,这是因为它无需在交换机上同时存储新旧规则,而且当规则更新操作间的依赖关系可能成环时,方案 4 需要调用两阶段更新方案等,而 HS-CRU3 根据环所涉及的子更新数目灵活选择,并且即便调用两阶段更新方案,涉及的规则数目也较小,更新中的等待时间较短,此外,HS-CRU3 在每次网络更新中引起的路由回路持续时间很短,至多约为 9ms.

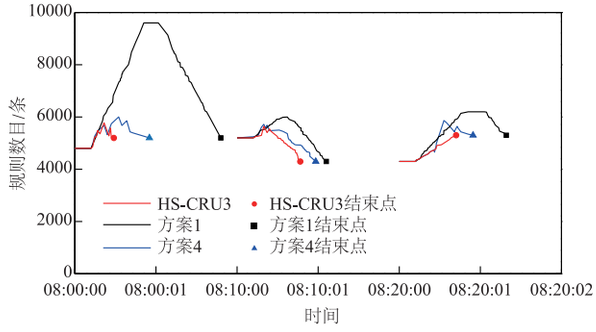

图5　更新过程中网络的规则数目变化图

3.2.2　交换机规则存储开销实验

以 r_ov 评估交换机的规则存储开销,如下:

$$r\_ov=\frac{worst-\max(R_{old},R_{new})}{\max(R_{old},R_{new})}\times 100\% \qquad (6)$$

其中,$worst$ 表示更新过程中交换机最多存储的规则数目,R_{old} 和 R_{new} 分别为更新前后交换机上的规则数目. 实验采用网络 2,要求网络在更新过程中无路由回路. 首先随机选取若干主机对,由控制器生成规则,为它们建立起连接,之后随机关闭若干交换机,由控制器重新计算路由路径,并分别采用 HS-CRU2、HS-CRU4(取 $\alpha=0,\beta=1,k^*=0$)、方案 2(取 $k=4$)和方案 4 更新,记录每个交换机的 r_ov 值,结果如表 3 所示.

HS-CRU2 的交换机最大规则存储开销在两种情况下与方案 2 相当,分别比方案 4 约少 48% 和 47.9%,而且根据图 2 可知,HS-CRU2 的时间开销远优于方案 2,且流的最长更新时间仅比方案 4 约多 4.2%. HS-CRU4 的交换机最大规则存储开销在两种情况下分别比方案 2 约少 25.3% 和 34.3%,比方案 4 约少 59.9% 和 69.7%,而且主机对越多,优势越明显. 原因是 HS-CRU2 和 HS-CRU4 根据交换机当前负载情况灵活地交替增删规则,保证规则数目尽可能保持在较低水平,而方案 4 未考虑规则存储开销,同时,方案 2 求解规则更新顺序的速度慢,拓展性差.

表 3　交换机规则存储开销对比表

主机对		500	1500
HS-CRU2	最大值	15.7%	22.7%
	平均值	12.6%	10.8%
HS-CRU4	最大值	12.1%	13.2%
	平均值	4.5%	5.6%
方案 2	最大值	16.2%	20.1%
	平均值	13.5%	12.2%
方案 4	最大值	30.2%	43.6%
	平均值	20.3%	28.1%

3.2.3　链路过载数据量实验

链路过载数据量是指到达链路的超出链路额定承载能力之外的数据量大小[14],其值越大,交换机越容易出现丢包现象. 实验采用网络 1,设置 CS_D 服从均值为 100ms,方差为 25ms 的正态分布,并与方案 3 一样采用基于隧道的转发方式,要求更新过程无路由黑洞. 因为 HS-CRU(实验采用 HS-CRU2)采用的网络属性验证技术不考虑链路资源约束等性能属性,所以实验在初始调度任务时优先开始减小链路负载或者使链路负载增加量较小的子更新所含的任务. 首先启用所有交换机,随后随机关闭某条链路,分别记录完成 50% 和 99% 更新时链路过载数据量,重复 20 次实验取均值,结果如图 6 所示. 方案 3 的链路过载数据量分别比 HS-CRU2 少约 25.2% 和 21.5%,这是因为前者在调度操作节点时考虑了链路资源约束,但 HS-CRU2 的更新时间开销明显优于方案 3,这使得链路拥塞时间不至于过长.

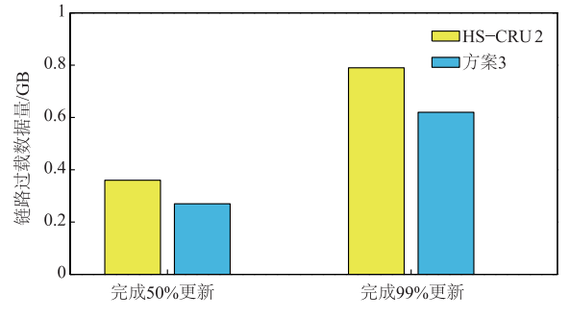

图6　链路过载数据量实验结果对比图

4　结束语

为解决 OpenFlow 网络的规则一致更新问题,本文从协调交换机的规则更新执行顺序出发,提出了基于

启发式调度的规则一致更新方案,在保证网络更新过程中满足期望的网络属性同时,缩短更新时间,降低交换机规则存储开销.仿真实验表明,在数据中心和广域网等不同网络环境下,与文献[10]、[14]和[5]中的方案相比,HS-CRU 完成50%和99%规则更新的时间能够约快48.5% ~91%,而且控制器与交换机之间的时延越大,HS-CRU 的优势越大;在不同规模的更新任务条件下,与文献[11]和[5]中的方案相比,HS-CRU 的交换机最大规则存储开销可以约少25.3% ~69.7%,且当网络中待更新的规则越多,HS-CRU 的优势越明显.但是,HS-CRU 未充分考虑链路资源约束,需要进一步研究.

参考文献

[1] McKeown N. Software-defined networking[A]. Proceedings of IEEE International Conference on Computer Communications[C]. Rio de Janeiro, Brazil: IEEE, 2009. 30 -32.

[2] McKeown N, Anderson T, Balakrishnan H, et al. OpenFlow: enabling innovation in campus networks[J]. ACM SIGCOMM Computer Communication Review, 2008, 38(2): 69 -74.

[3] 左青云,陈鸣,赵广松,等. 基于 Open Flow 的 SDN 技术研究[J]. 软件学报,2013,24(5):1078 -1097.
Zuo Qingyun, Chen Ming, Zhao Guangsong, et al. Research on OpenFlow-based SDN technologies[J]. Journal of Software, 2013, 24(5): 1078 -1097. (in Chinese)

[4] Peresini P, Kuzniar M, Canini M, et al. ESPRES: easy scheduling and prioritization for SDN[A]. Proceedings of USENIX Open Networking Summit[C]. Santa Clara, CA, USA: USENIX Association, 2014.

[5] Zhou W, Jin D, Croft J, et al. Enforcing customizable consistency properties in software-defined networks[A]. Proceedings of USENIX Symposium on Networked Systems Design and Implementation[C]. Oakland, CA, USA: USENIX Association, 2015. 73 -85.

[6] Yu M, Rexford J, Freedman MJ, et al. Scalable flow-based networking with DIFANE[J]. ACM Sigcomm Computer Communication Review, 2010, 40(4): 351 -362.

[7] Curtis AR, Mogul JC, Tourrilhes J, et al. DevoFlow: scaling flow management for high-performance networks[J]. ACM Sigcomm Computer Communication Review, 2011, 41(4): 254 -265.

[8] Heller B, Sherwood R, McKeown N. The controller placement problem[J]. ACM Sigcomm Computer Communication Review, 2012, 42(4): 7 -12.

[9] Reitblatt M, Foster N, Rexford J, et al. Abstractions for network update[J]. ACM Sigcomm Computer Communication Review, 42(4), 2012: 323 -334.

[10] Reitblatt M, Foster N, Rexford J, et al. Consistent updates for software-defined networks: change you can believe in![A]. Proceedings of ACM Special Interest Group on Data Communication Workshop on Hot Topics in Networks[C]. Cambridge, MA: ACM, 2011. 1 -6.

[11] Katta NP, Rexford J, Walker D. Incremental consistent updates[A]. Proceedings of ACM Special Interest Group on Data Communication Workshop on Hot Topics in Software Defined Networking[C]. Hong Kong, China: ACM, 2013. 49 -54.

[12] McGeer R. A safe, efficient update protocol for openflow networks[A]. Proceedings of ACM Special Interest Group on Data Communication Workshop on Hot Topics in Software Defined Networking[C]. Helsinki, Finland: ACM, 2012. 61 -66.

[13] Mahajan R, Wattenhofer R. On consistent updates in software defined networks[A]. Proceedings of ACM Special Interest Group on Data Communication Workshop on Hot Topics in Networks[C]. College Park, MD, USA: ACM, 2013. 29 -31.

[14] Jin X, Liu H H, Gandhi R, et al. Dynamic scheduling of network updates[A]. Proceedings of ACM Special Interest Group on Data Communication[C]. Chicago, IL, USA: ACM, 2014. 539 -550.

[15] 杜晓丽,蒋昌俊,徐国荣,等. 一种基于模糊聚类的网格 DAG 任务图调度算法[J]. 软件学报,2006,17(11):2277 -2288.
Du Xiaoli, Jiang Changjun, Xu Guorong, et al. A grid DAG scheduling algorithm based on fuzzy clustering[J]. Journal of Software, 2006, 17(11): 2277 -2288. (in Chinese)

[16] Openflow Switch Consortium. Openflow switch specification/version 1.0.0[OL]. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf, 2015 -03 -09.

[17] 刘艺,雷程,张红旗,等. 基于 MapReduce 的 OpenFlow 网络属性验证技术[J]. 计算机研究与发展.(已录用)
Liu Yi, Lei Cheng, Zhang Hongqi, et al. MapReduce-based network property verification technique for OpenFlow network[J]. Journal of Computer Research and Development. (accepted) (in Chinese)

[18] Liu Yujie, Li Yong, Wang Yue. Optimal scheduling for multi-flow update in Software-defined networks[J]. Journal of Network and Computer Applications, 2015, 54: 11 -19.

[19] Kazemian P, Varghese G, McKeown N. Header space analysis: static checking for networks[A]. Proceedings of

USENIX Symposium on Networked System Design and Implementation[C]. San Jose, CA, USA: USENIX Association, 2012. 113 – 126.

[20] Estrada E, Rodriguez V, Lazquez J A. Subgraph centrality in complex networks[J]. Physical Review E, 2005, 71(5): 1 – 9.

作者简介

刘　艺　女,1991 年出生,江西崇义人,现为解放军信息工程大学硕士研究生,研究方向为 SDN 安全
E-mail:liuyi9582@126.com

张红旗　男,1962 年出生,河北唐山人,博士、教授、博士生导师,研究方向为网络安全和等级保护等