软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(11):3440−3456 [doi: 10.13328/j.cnki.jos.005655]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

软件定义网络中延迟满足的路由选择与实时调度更新*

朱金奇[1,2], 孙华志[1], 黄永鑫[1], 刘 明[2]

[1](天津师范大学 计算机与信息工程学院,天津 300387)

[2](电子科技大学 计算机科学与工程学院,四川 成都 611731)

通讯作者: 孙华志, E-mail: sunhuazhi@eyou.com; 朱金奇, E-mai: zhujinqi1016@163.com

摘 要: 由于数据流的动态性和流量负载转移,软件定义网络(software defined networking,简称 SDN)需要频繁更新数据平面以优化网络性能.大多数已有路由更新策略首先根据网络当前流量状态确定目标路由配置,然后更新数据流的路由.然而,由于交换机基于 TCAM(ternary content addressable memory)进行流表更新的速度较慢,导致路由更新的延迟通常较大.当网络规模大或网络拓扑结构经常变化时,路由更新的延迟可能更大.研究发现,大多数数据流的持续时间很短且整个网络的流量强度在一段时间后会发生变化.如果路由更新延迟过长,更新后的路由配置可能不再有效.为此,研究了 SDN 的实时路由更新问题,提出了延迟满足的路由选择和调度更新策略(delay satisfied route selection and updating scheme,简称 DSRSU).与大多数现有研究不同,DSRSU 同时从控制平面路径选择和数据平面的更新调度两方面来联合优化,降低路由更新的延迟.路径选择阶段只选择部分数据流进行路由更新;更新调度阶段通过建立更新关系图挖掘数据流的更新先后顺序,进一步加快路由更新速度.仿真分析结果表明,与现有几种路由更新策略相比,DSRSU 能够在大幅度降低路由更新延迟的同时,达到与现有策略相似的网络性能.

关键词: 软件定义网络;路由更新;实时更新;交换机

中图法分类号: TP393

中文引用格式: 朱金奇,孙华志,黄永鑫,刘明.软件定义网络中延迟满足的路由选择与实时调度更新.软件学报,2019,30(11):3440−3456. http://www.jos.org.cn/1000-9825/5655.htm

英文引用格式: Zhu JQ, Sun HZ, Huang YX, Liu M. Delay satisfied route selection and real-time update scheduling in software defined networking. Ruan Jian Xue Bao/Journal of Software, 2019,30(11):3440−3456 (in Chinese). http://www.jos.org.cn/1000-9825/5655.htm

Delay Satisfied Route Selection and Real-time Update Scheduling in Software Defined Networking

ZHU Jin-Qi[1,2], SUN Hua-Zhi[1], HUANG Yong-Xin[1], LIU Ming[2]

[1](School of Computer and Information Engineering, Tianjin Normal University, Tianjin 300387, China)

[2](School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China)

Abstract: Software defined networking may need to frequently update their data planes to optimize network performance due to flow dynamics or traffic load transfer. Most existing strategies first determine a target route configuration based on the current network flow status. Then, flows in the network are updated to the target route configuration. However, since the low operation speed of ternary content addressable memory (TCAM) for flow tables update, route updates usually gets long delay. Moreover, route updates delay will get longer in a large or topology frequently changed network. According to recent works, most flows have short durations and the total flows

* 基金项目: 国家自然科学基金(61472068, 61572113); 天津市自然科学基金(17JCYBJC16400); 天津市科技发展战略研究计划(17ZLZXZF00530); 天津师范大学博士基金(043/135202XB1615)

Foundation item: National Natural Science Foundation of China (61472068, 61572113); Natural Science Foundation of Tianjin Municipality (17JCYBJC16400); Science and Technology Development Strategy Research Plan of Tianjin Municipality (17ZLZXZF00530); Doctoral Fund of Tianjin Normal University (043/135202XB1615)

收稿时间: 2018-03-26; 修改时间: 2018-05-27, 2018-07-05; 采用时间: 2018-08-18

© 中国科学院软件研究所 http://www.jos.org.cn

intensity may vary after a certain time period. Hence, the new route configuration may be inefficient if the route update delay takes too long. In this study, the real time route update for SDN is addressed and a delay satisfied route selection and updating scheme (DSRSU) is proposed. Different from most existing studies, DSRSU jointly considers the flow route selection in the control plane and route update scheduling in the date plane to reduce the route update delay. More specially, only a subset of flows is chosen for route updates in route selection. To further improve the update speed, an update dependency graph is established to explore the scheduling order of the flows during update scheduling. Simulation results demonstrate DSRSU can largely reduce the route update delay compared with previous route update strategies while maintaining a similar route performance.

Key words: software defined network; route update; real time update; switch

软件定义网络(software defined networking,简称 SDN)是一种把控制平面和数据平面分离在不同设备上实现的新型网络架构[1].在控制层,控制器通过在数据平面上建立路由转发规则,对数据平面提供集中和灵活的控制,掌握全局网络信息;在数据层,交换机根据转发规则进行数据流转发.由于网络中数据流的动态性[2]、流量负载转移、网络维护[3]等原因,数据平面的状态需要不断更新来避免次优路径可能导致网络拥塞.这里,数据平面的状态指决定交换机如何转发数据包的转发规则集合.控制器则以推送转发规则到交换机流表的形式来响应诸如流量强度变化、来自主机的新连接等事件,以达到各种网络性能需求,如负载均衡和资源的高效利用.因此,路由更新(网络更新)能够极大地改善网络的性能,提高网络资源的利用率[4].

路由更新过程包括控制平面的路由选择和数据平面的交换机流表更新两部分.由于路由更新的速度决定了控制器的灵活性,因而成为众多网络应用的重要指标.已有大多数路由更新相关研究[4,5]根据网络当前数据流状况(如流的数量、网络中流的总大小)为数据流计算新路由,接着利用各种路由更新调度算法,把数据平面从当前路由配置更新到新配置.如文献[5]把网络中所有数据流分成最小数量的集合,并在每轮更新一个集合中的数据流来避免网络拥塞.Jin 等人[4]则把各交换机流表更新调度操作之间的依赖关系建立一个全网的更新调度图,接着动态调度不同交换机上的更新操作,以降低路由更新延迟.然而,由于现有相关工作没有关注控制平面的路由选择对路由更新速度的影响,导致路由更新延迟较长.当网络规模较大时,更新延迟增加现象尤为明显.例如,在一个中等规模的数据中心网络中,对于由 40 个服务器组成的机架,到达交换机的流量达到 75KB~100KB 流量/min[6].假设某交换机需要更新 8KB 数据流的路由(仅仅是一小部分),其更新延迟取决于需要更新的总的转发规则数量和 TCAM 更新操作(插入或修改)的速度两部分.通过对当今商用交换机进行测试,每个插入和修改 TCAM 流表的操作大约需要 5ms 和 11ms[4].对于上述实例,如果插入和修改操作分别为 4KB,则交换机转发规则更新所需时间至少为 60s.文献[2,6]通过对不同的数据中心进行测试,得到网络中的数据流具有如下特性:1) 超高 80%的数据流持续时间不超过 10s;2) 仅有低于 0.1%的数据流持续时长超过 200s;3) 数据流中超过 50%的字节持续时间低于 25s.根据这些特性可知,如果全网路由更新所需时间过长,由于部分数据流已经终止且新的数据流产生,更新后的传输路径可能变得无效.为此得出:实时快速的路由更新是提升 SDN 性能的关键,是网络必须的.路由更新速度快,将有助于提高网络性能.然而,实时路由更新策略的设计具有如下挑战:1) 网络拓扑结构可能动态变化,并且网络中不断有来自主机新的传输请求,造成网络中的数据流动态变化;2) 各交换机执行更新操作的速度存在差别,另外,各交换机开始执行更新的时间不同,这可能造成链路拥塞;3) 满足实时更新的需求,在更新延迟容忍值内完成路由更新操作.

事实上,路由选择和交换机流表调度更新均对更新延迟具有重要影响.当控制器更新的数据流数量较多时,尽管路由已经是最佳了,更新延迟仍较大;相反地,若需要更新的数据流少,更新延迟则大为降低.也就是说,网络路由更新延迟和数据流路径优化之间存在折中.为此,与现有研究不同,本文提出延迟满足的路由选择和调度更新策略(delay satisfied route selection and updating scheme,简称 DSRSU),同时从路径选择优化和流表调度更新两方面考虑来满足 SDN 实时路由更新的要求.路由选择阶段,只选择部分数据流进行路由更新,根据链路剩余容量和网络路由更新的延迟容忍值 T_0 共同决定选择哪部分数据流进行路径更新,并确定这些数据流的目标路径.路由更新阶段,首先建立全网更新调度图;接着,根据更新调度图挖掘数据流更新之间的依赖关系,从而建立更新关系图;最后,根据更新关系图确定交换机执行数据流更新的先后顺序,以在满足路由更新延迟容忍值基础

© 中国科学院软件研究所 http://www.jos.org.cn

上进一步加快路由更新的速度.大量实验仿真结果表明,与现有的几种路由更新策略相比,DSRSU 不仅能够极大地降低路由更新的延迟,同时能够实现与对比策略类似的网络性能.本文创新性如下:

- 提出 SDN 路由更新延迟和数据流路由优化之间存在折中,因此仅更新部分数据流的路由来满足数据流实时路由更新要求的思想;
- 提出路由更新过程要保证拥塞避免、丢包避免、数据包一致性和实时路由更新的要求;
- 在路由选择阶段,根据链路剩余容量和路由更新的延迟容忍值共同决定更新数据流集,在实现实时路由更新的同时,保证链路负载均衡;
- 路由更新过程通过建立更新关系图挖掘数据流更新间的依赖关系,使无依赖数据流的更新操作尽量并行执行,以进一步降低路由更新的延迟,保证实时路由更新.

1 相关工作

SDN 中,由于网络的动态性,控制器需要频繁更新数据平面以提高网络性能.近些年,很多研究者提出了不同的路由更新策略以实现各种网络性能优化目标.根据优化目标的不同,已有路由更新策略可以分为两类.

- 第 1 类策略以路由更新过程数据包一致性保证、丢包避免、更新过程拥塞避免为主要目的.

如 Wang 等人[7]对 SDN 网络更新可能引起的若干问题如数据包不一致、链路拥塞等进行了综述,并给出了针对各问题的几种现有解决方案.在保证数据包一致性方面,文献[8]介绍了一致性网络更新的概念,并确定了两个不同的一致性等级:每个包和每个流.接着提出两阶段提交协议以保证数据包的一致性,每个数据包或者按照更新前的路径传输,或者根据更新后的路径在网络中传输直到目的地.然而这种方式中,旧转发规则的数据包处理完毕后新规则的数据包才开始处理,引起交换机的空间被占用,产生较高的处理成本.文献[9]给出一种 openflow 协议来保证路由更新过程中数据包一致性,该协议通过确保任何时间交换机上只有一组路由转发规则集合存在来节省交换机的资源,尤其是 TCAM 的空间.然而,由于该协议执行完一组转发规则后再开始执行下一组转发规则,造成路由更新速度慢.Katta 等人[10]提出一种遗传算法来降低路由更新中对内存的需求,保持数据包的一致性.该算法把新的路由转发规则分成一个个的分片,将转发规则更新分成多轮进行,每一轮更新一个分片.通过增加分片的数量,可以降低交换机的存储开销,但却增加了路由更新延迟.文献[11]讨论了能够实现并行和稳健转发规则实施的分布式控制平面.作者首先引入描述数据平面和控制平面交互的形式化模型;接着,系统地阐述并发网络转发规则更新的一致化问题.另外,在保证拥塞避免、丢包避免等方面,文献[12]认为,路由更新应保证链路不拥塞和回路避免,即数据包在网络传输中不应该出现回路.该文研究了虚拟机场景下的转发规则更新算法,该算法可以确保在更新过程中拥有足够的带宽,同时不会影响到其他流的正常转发.Liu 等人[13]提出了异步交换机和网络流量动态变化下的拥塞避免网络更新机制 zUpdate,以避免更新过程中链路拥塞和数据包丢失.Mizrahi 等人[14]介绍了利用准确时间协调网络更新的方法 Time4,文中定义了一组称为流交换的更新场景,在这些场景中,Time4 具有比已有路由更新方法更低的数据包丢失率.

- 第 2 类策略的主要目标是降低路由更新的延迟.

Hong 等人[5]通过分流来最大限度地减小无拥塞更新的延迟.他们把路由更新问题转化为线性规划问题,提出在多项式时间内解决该问题,并分析了路由更新的延迟上限.文献[4]描述了 SDN 中快速、一致性更新的 Dionysus 系统,该系统把各交换机流表更新调度操作之间的依赖关系建立一个全网的更新调度图;然后,为每个更新操作节点计算关键路径长度;然后,根据各更新操作节点关键路径长度值的大小动态调度不同交换机上的路由更新操作,以降低路由更新延迟.文献[15]介绍了 TIMEFLIPs 方法实现准确的基于时间的路由更新.TIMEFLIPs 利用 TCAM 条目中的时间戳区域实现,不仅可以用来完成 Atomic Bundle 更新,而且能够以较高的精度实现网络更新.然而上述工作仅仅依据网络当前工作量,把网络从当前路由更新到目标路由配置.尽管试图提高路由更新的速度,但由于交换机基于 TCMA 进行流表更新的操作速度慢、更新数据流数量大等原因,仍可能造成路由更新的延迟较长.此外,Xu 等人在文献[16]中首次提出了更新部分数据流以保证路由更新延迟时间约束的思想.文中介绍了两种算法以实现实时路由更新.然而他们把数据流实时路由更新问题转化为线性规划

© 中国科学院软件研究所 http://www.jos.org.cn

问题,这将导致所提算法更新速度慢且不适用于具有数据流数量大的大型网络.此外,为防止链路陷入拥塞,该文在为数据流分配新的路由后,并不回收原来路径上该数据流占用的链路资源,造成链路容量的浪费和信道利用率低下.

2 网络模型和问题描述

2.1 网络模型

一个典型的 SDN 由两种类型的设备组成:一台控制器和若干台交换机.整个网络的拓扑为 $G=(V,E)$.其中,$V=\{v_1,v_2,\ldots,v_n\}$是交换机的集合,v_i 代表第 i 个交换机,$n=|V|$;E 是连接交换机之间链路的集合.令当前网络中所有数据流的集合为 $T=\{r_1,r_2,\ldots,r_m\}$,$m=|T|$是数据流的总数量.假设任一数据流 r 从入口交换机(ingress switch)开始直到被传输给出口交换机(egress switch).假设当交换机为数据流 r 建立流表项后,交换机就能统计流 r 的大小.通过收集流量统计信息,控制器了解数据流 r 的大小 $S(r)$.与已有研究类似[5,8],为简单起见,令每个流在网络中只选择一条路径进行数据传输.对于数据流 r,设当前路由为 $P_c(r)$,更新的目标路径为 $P_d(r)$.此外,当数据流到达交换机时,如果有匹配的流表项则按流表项进行转发;否则,交换机把数据包头报告给控制器,再由控制器决定此数据流的路由,并在该路径的所有交换机上为此数据流建立流表项.

2.2 TCAM更新时间模型

文献[4]表明,若所有流表项具有相同优先级,则插入或修改流表项的延迟与需要插入或修改流表项的数量几乎成线性关系.然而很多实际应用中,大多数数据流具有不同的优先级[17].另外,某些应用把数据流分为微流和宏流规则方案[18],并对宏流(又称为大象流)赋予较高优先级,对微流(老鼠流)赋予较低优先级.由于本文只选择一些大象流进行路由更新,我们假设这些大象流具有独特的优先级.此时,对交换机流表项的插入/修改操作认为一个常数[16].令 t_i 和 t_m 分别表示插入和修改一个流表项操作所需的时间,通过对实际交换机进行测试,得出 $t_m\approx 11\text{ms}$,$t_m>t_i$[4],因为修改流表项涉及插入新流表项和删除旧流表项两种操作.用 $t(s,P_c(r),P_d(r))$表示为了把流 r 的路径更新到目标路径 $P_d(r)$,对交换机 s 流表项的操作延迟.我们有:

$$t(s,P_c(r),P_d(r))=\begin{cases}t_m, & s \text{ is on both } P_c(r) \text{ and } P_d(r)\\ t_i, & s \text{ is only on path } P_d(r)\\ 0, & \text{otherwise}\end{cases} \tag{1}$$

2.3 问题描述

SDN 中,路由更新的速度通常决定了控制器的灵活性,因而成为众多网络应用的重要指标.路由更新速度慢,意味着发生网络丢包和链路拥塞的时间较长,这严重影响网络性能.现有大多数路由更新方法[5,10,13]是静态的,即提前计算好各数据流路径更新的顺序,路由更新过程中按照此顺序进行转发规则更新.然而此方法很难适应网络中数据流的动态变化和不同交换机执行转发规则更新的时间差异,造成路由更新的速度慢和更新的不一致性.由于交换机硬件上的差异、CPU 负载的不同以及控制器进行远程过程调度 RPC 的时间可变性,网络中不同交换机执行转发规则更新的时间差异是普遍存在的.为此,某些系统[4,19]根据当前工作量进行动态路由更新,以快速适应网络中不断变化的数据强度.如文献[19]提出了频繁的网络更新,以实现快速路由更新并获得高效的网络利用率.文献[4]的 Dionysus 方法根据流表更新调度操作间的依赖关系动态调度不同交换机上的路由更新操作,加快路由更新速度.然而由于需要更新的数据量可能较大,上述路由更新的延迟仍然可能较长.Xu 等人[16]提出,当需要更新的数据流总量为 40KB 时,利用文献[4]的方法进行路由更新的延迟接近 65s.这对于许多网络应用是不能容忍的.为进一步降低路由更新的延迟,Xu 等人提出仅更新部分数据流以保证路由更新延迟时间的约束.文中介绍了两种算法以保证路由更新延迟不超过设定的约束值.然而所提算法主要问题在于,首先,更新过程中为避免拥塞,为数据流分配新路径后,并不回收原来路由上该数据流占用的链路资源,造成链路容量的严重浪费;其次,哪些数据流能够并行执行并不明确,影响路由更新的速度.

© 中国科学院软件研究所 http://www.jos.org.cn

为此,本文提出延迟满足的路由选择和调度更新策略 DSRSU,从路由选择和路由更新调度两方面联合进行优化,争取在满足更新延迟容忍值前提下尽可能降低路由更新的延迟,保证实时路由更新.除实时路由更新的要求外,路由更新过程还需满足以下条件.

1) 链路拥塞避免:路由更新要合理安排数据流调度更新的执行顺序,以保证到达链路的总流量低于链路容量.如图 1 所示,设每条链路的容量是 10 单位,且每条数据流的大小在图中进行了标记.控制器想把网络从图 1(a)状态更新为图 1(b)状态.如果控制器同时发送所有更新操作命令给交换机,由于不同交换机执行更新的时间不同及快慢不同,此策略可能造成某些链路拥塞.比如,当交换机 s_4 执行更新 r_2 之前,s_3 就执行了更新 r_1,那么链路 s_1s_4 将拥塞.
2) 丢包避免:交换机在转发数据流过程中要尽量避免丢失数据包.
3) 满足一致性:数据流或者遵循旧的路由,或者遵循新路由进行转发,而不能根据两者的混合进行数据转发,否则会引起数据转发的混乱.

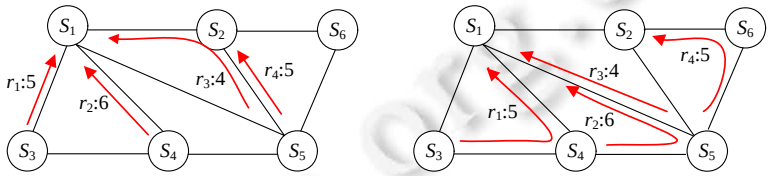


Fig.1 Link congestion in route update

图 1 路由更新中的链路拥塞

在路由更新发生时,为实现丢包避免和数据包一致性,本文采取了与文献[4,8,17]类似的两阶段提交方法来更新规则:第 1 阶段,当需要更新某数据流的转发规则时,只有对相应原数据流处理完毕的交换机才能进行转发规则更新或删除;第 2 阶段,在数据流新路径完全建立起来之后(新路径上所有交换机均更新完毕后),才把数据流放在新路径上传输.两阶段转发的具体细节,可详见文献[4,8].对于路由更新过程中如何避免拥塞和保证实时路由更新,是本文要研究的重要内容,具体见下文.

3 DSRSU 的实现

为了达到网络性能优化和路由更新延迟间的折中,DSRSU 策略力图从路径选择和交换机流表调度更新两个方面来联合优化,实现实时路由更新.通常,路由更新中更新的数据流数量越多、更新的数据总量越大,路由更新调度的效果就越好.但更新的数据流数量大会造成非常大的更新延迟,难以满足网络实时路由更新要求.为实现实时路由更新,DSRSU 就需要合理设计路径选择和分配算法,确定更新的数据流集,通过更新部分数据流尽量达到较好的路由更新效果.DSRSU 路径选择对路由更新调度的影响还表现在路径选择阶段选择的数据流数量越少,路由更新完成的速度越快.因此,在保证实时路由更新和较低网络负载的同时,路径选择和分配应尽量减小更新数据流的数量,以进一步加快路由更新调度的速度,降低路由更新调度延迟.下面首先描述如何通过路径选择和分配算法确定更新数据流集;接着,在路由选择基础上设计路由更新调度策略,把更新数据流集中的数据流更新到新路由的同时进一步降低路由更新的延迟;然后描述路由更新调度算法;最后介绍更新调度中的循环问题,并给出了处理方法.

3.1 路径的选择和分配

路由选择是 DSRSU 的重要组成部分,在路由选择过程中,为了实现实时路由更新和链路负载均衡,我们同时从链路的剩余容量和交换机的延迟容忍值 T_0 两个方面选择网络中数据流 T 的子集 T^U 作为更新路径集,并为其中每个数据流选择一条可行路径作为目标路由.与已有工作[5]类似,设控制器预先为 T 中的每个数据流计算

© 中国科学院软件研究所 http://www.jos.org.cn

了一个可行路径集,并始终从中选择一条路径作为该数据流的目标传输路径.如数据流 r 的可行路径集为 P_r,我们首先把集合 T 中所有流按由大到小顺序排列成队列 Z;然后,T^U 的选择和 T^U 中数据流的目标路径分配算法伪代码见算法 1.具体过程描述如下.

步骤 1:从队头中选择数据流 r,对 P_r 的每条路径 p,若用 p 上各链路 e 剩余容量的最小值代表其可用容量,则把 P_r 中的路径按照可用容量由大到小进行排列.

步骤 2:从 P_r 中选择可用容量最大的路径(设为 p_m),判断若为 r 分配 p_m 作为目标路径 $P_d(r)$时,p_m 的可用容量是否满足公式(2):若满足,说明 p_m 的所有链路都能包含流 r,则转入步骤 3;若不满足,说明 P_r 中其余路径的可用容量也不能包含 r,则 r 的路径更新失败,此时把 r 从 Z 中移入集合 T^N 后,继续选择 Z 头数据流重复步骤 1.

$$s(r)\leqslant\lambda c(p_m) \tag{2}$$

其中,$c(P_m)$表示路径 p_m 的可用容量,$s(r)$是流 r 的大小,λ为链路负载因子.

步骤 3:若步骤 2 中 p_m 的剩余容量能够包含数据流 r,则继续判断 p_m 上每个交换机的更新延迟是否不大于更新延迟容忍值 T_0:若成立,则分配 p_m 为 r 的目标路径 $P_d(r)$.如 p_m 上交换机 s 更新延迟的判断公式为

$$T_s+t(s,P_c(r),P_d(r))\leqslant T_0 \tag{3}$$

其中,T_s 表示交换机 s 的更新延迟值;$t(s,P_c(r),P_d(r))$为交换机 s 更新 $P_c(r)$为 $P_d(r)$需要执行的流表项操作所需时间,根据公式(1)选择相应的值.若 r 的路径更新成功,则把数据流 r 放入集合 T^U,从集合 Z 中删除流 r 并转入步骤 4;否则,认为 p_m 不可用,忽略 P_r 中可用容量最大的路径,转入执行步骤 2.

步骤 4:更新路径 $P_d(r)$上各链路 e 的剩余容量和 $P_d(r)$上所有交换机的更新延迟.若令 $l(e)$表示 $P_d(r)$上链路 e 的剩余容量,T_s 为 $P_d(r)$上交换机 s 的更新延迟,则 $l(e)$和 T_s 分别更新为

$$l(e)=l(e)-s(r) \tag{4}$$

$$T_s=T_s+t(s,P_c(r),P_d(r)) \tag{5}$$

步骤 5:对于当前路径 $P_c(r)$的每条链路,回收已分配给流 r 的容量,更新 $P_c(r)$上各链路 e'的剩余容量值为

$$l(e')=l(e')+s(r) \tag{6}$$

步骤 6:重复步骤 1 到步骤 5,直到网络中各交换机的更新延迟均大于 T_0.

算法 1. T^U 的选择和数据流的目标路径分配.

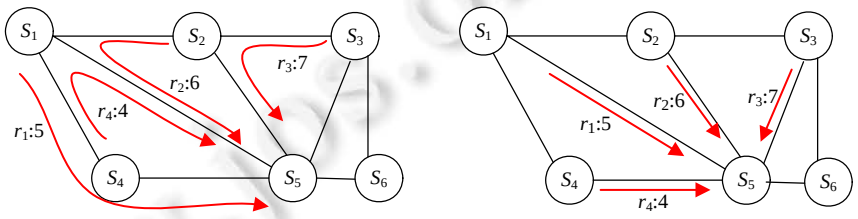
```
1.  for each flow r in Z do
2.    for each path p∈P_r in the decreasing order of its available capacity do
3.      if the available capacity of path p times λ≥s(r) then
4.        if Eq.(3) is satisfied for each switch s on path p then
5.          assign p as its target route and T^U=T^U∪{r};
6.          update the remaining capacity of each link e on p using Eq.(4);
7.          update the total update delay of each switch s on p using Eq.(5);
8.          for each link e on P_c(r) do
9.            update its remaining capacity as Eq.(6);
10.       else
11.         delete p from P_r
12.         continue
13.     else
14.       T^N=T^N∪{r};
15.       Remove flow r from queue Z;
16.       break
```

© 中国科学院软件研究所 http://www.jos.org.cn

3.2 动态路由更新调度

3.2.1 为什么需要动态更新调度

根据第 3.1 节确定数据流的目标路径后,我们需要更新交换机的流表项,把 T^U 中的数据流从当前路由更新为新路由.更新过程要保证链路不拥塞、丢包避免、数据包一致性,并尽量降低路由更新的延迟.第 3.1 节的目标路径分配算法给定了路径分配的顺序,如果按照此顺序更新数据流的路径,即上一数据流的路径更新完毕后控制器再给下一数据流目标路径上的交换机发送更新命令进行流表调度更新,虽然可以保证更新过程中链路不拥塞,但却导致路由更新的速度慢.例如,如图 2 所示,设每条链路的容量是 10 单位,数据流 r_1 的大小为 5 单位,r_2 大小是 6 单位,r_3 和 r_4 的大小分别为 7 单位和 4 单位.假设按第 3.1 节算法的顺序进行路由更新,更新顺序为更新 r_3→更新 r_2→更新 r_1→更新 r_4.这里,更新路径包括建立新路径并删除旧路径.控制器按照该顺序发送更新指令,仅当交换机完成上一路径更新后,控制器才发送下一路径更新命令.但从图 2 可知,交换机完成 r_3 的更新后,控制器可发送操作指令进行流 r_2 的更新,并且在完成 r_2 的路径更新后,流 r_1 的更新操作和 r_4 的更新操作可并行进行,并不会造成链路拥塞.相对于目标路径分配算法规定的更新顺序,在保证链路不拥塞的前提下,交换机同时进行多个数据流的更新可进一步降低路由更新的延迟.为此,在满足更新延迟容忍值基础上进一步降低路由更新的延迟,我们需要研究更加高效的更新调度方法.



(a) 当前路由 (b) 目标路由

Fig.2 Example of route update

图 2 路由更新实例

3.2.2 建立更新调度图

为了在路由更新延迟容忍值 T_0 内把 T^U 中数据流的路径更新为目标路径,同时尽量降低路由更新的延迟,就需要根据网络状况动态地调度数据流的更新操作.我们用更新调度图反映更新之间的依赖关系,保证路径更新过程中拥塞避免.更新调度图有两种类型的节点:资源节点和操作节点.资源节点用方框表示,方框内的数字代表该链路的可用剩余容量;操作节点用椭圆表示,代表数据流的更新操作(这里的更新操作包括建立新路由和删除旧路由,涉及到数据流新路径和旧路径上所有交换机,包括建立和修改转发规则).更新调度图中从某资源节点指向某操作节点的有向边代表执行该更新操作需要该链路资源,边上的数字代表所需链路资源的数量.此时,该操作节点称为此资源节点的子节点.而一条从操作节点指向资源节点的有向边表示完成该更新操作能够释放的链路资源,有向边上的数字为释放的链路资源数量.此时,该操作节点称为此资源节点的父节点.

我们通过图 3 来说明更新调度图的建立过程.图 3(a)的状态为当前路由状态,图 3(b)的状态是目标路由状态.设每条链路的容量是 10 单位,各数据流的大小在图中进行了标记.根据图 3 中各数据流当前占有链路资源和更新各流需要链路资源的情况,建立更新调度图如图 3(c)所示.以其中流 r_1 为例,把 r_1 的路径更新为目标路径需要链路 S_1-S_5 这 5 单位的资源;此外,r_1 的路径更新为目标路径后能够分别释放链路 S_4-S_5 及链路 S_1-S_4 各 5 单位的链路资源.根据该更新调度图容易看出:更新 r_3 或更新 r_6 执行完成后才能执行更新 r_2.另外,更新 r_1 受限于流 r_2 或 r_4 更新执行完成.如果交换机并行执行更新 r_3、更新 r_6 和更新 r_4,在流 r_3 或 r_6 之一更新完成后立即执行更新 r_2,并且更新 r_2 和更新 r_4 的任意一个执行完后马上执行更新 r_1.此时,无论网络中是否有交换机执行流表更新的速度较慢,按照该顺序调度路由更新操作,相对于第 3.1 节的路径更新顺序,网络路由更新的延迟降低了.在满足更新延迟容忍值前提下,为任意类型的网络拓扑执行这样的路由更新调度,进一步加快路由更新的速度,是下

© 中国科学院软件研究所 http://www.jos.org.cn

面我们工作的目标.

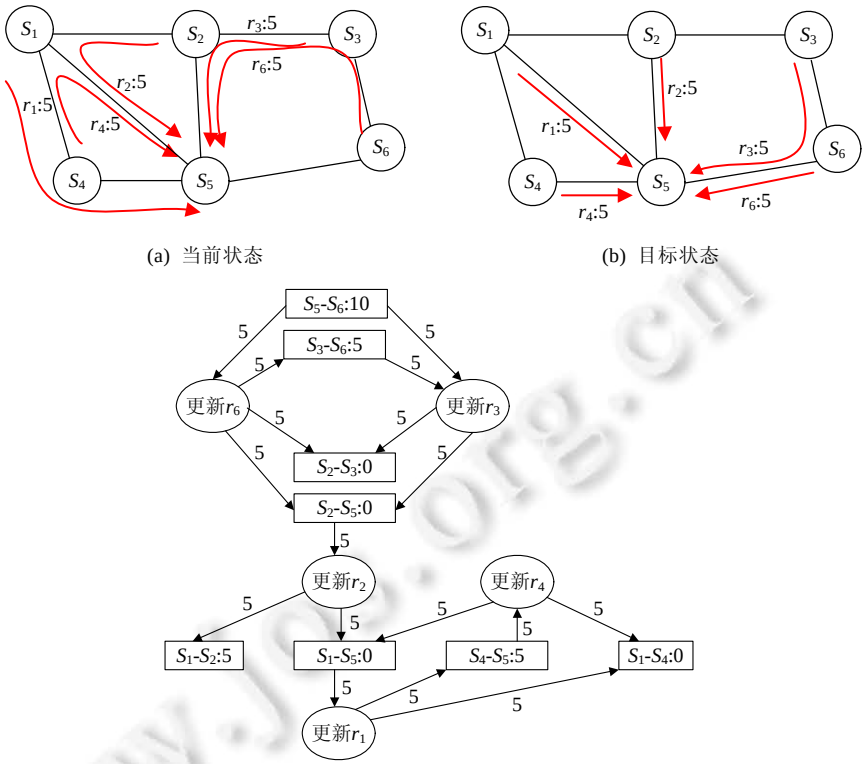


Fig.3 Building update scheduling graph for updating from current state to target state

图 3 从当前状态到目标状态更新调度图的建立

3.2.3 建立更新关系图

我们用更新关系图反映数据流更新间的依赖关系.对更新调度图中每个资源节点,若该资源节点存在子节点,我们就对该资源节点进行分析,挖掘其父节点与子节点间的更新依赖关系;然后,再对任一数据流进行路由更新需要的所有资源节点进行综合,最终建立网络中各数据流间的更新关系.以链路 S_a-S_b 和流 r 为例,设更新流 r 的路由需要链路资源 S_a-S_b,则流 r 的更新依赖关系建立过程描述如下.

步骤 1.

1) 若 S_a-S_b 的剩余容量能够满足其所有子节点所需资源(大于所有子节点请求资源的总和),说明子节点的更新操作无需等待该资源,则此时建立 S_a-S_b 的父节点和子节点的更新关系如图 4(a)所示.

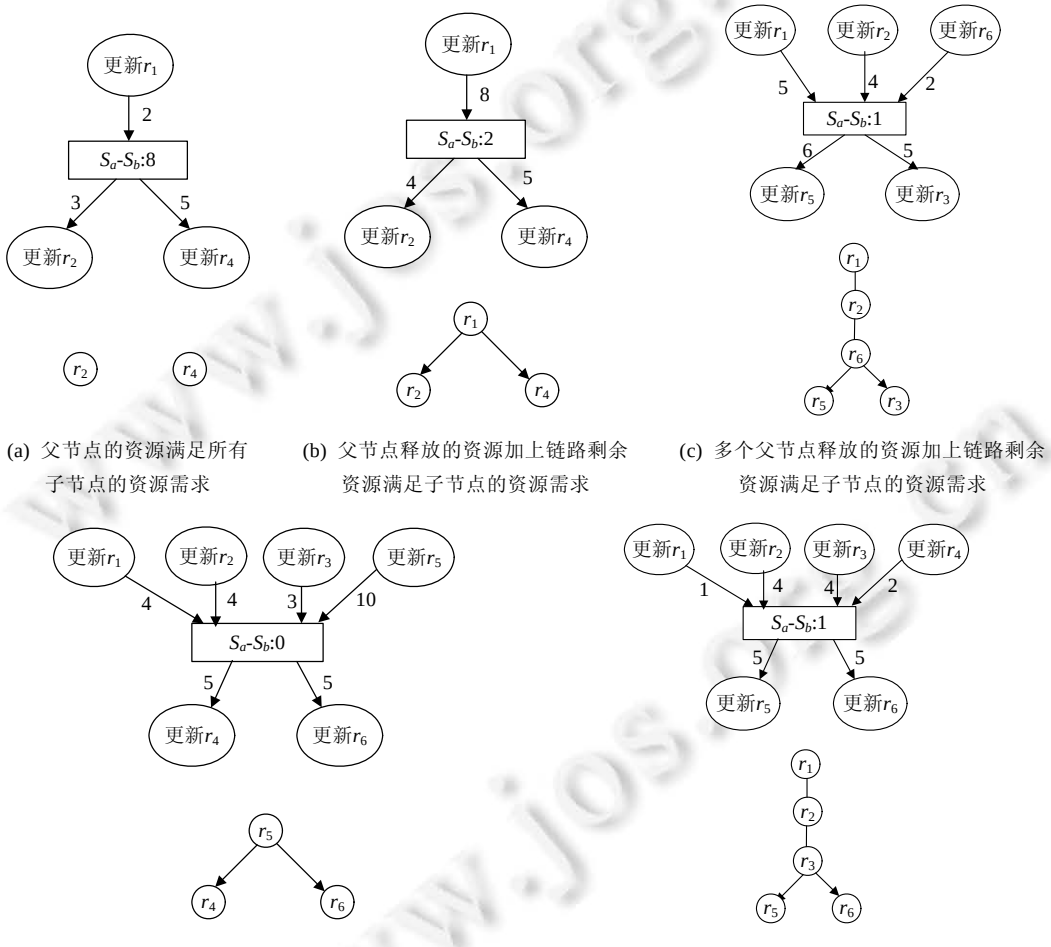
2) 若 S_a-S_b 的剩余容量不能满足所有子节点的资源需求,而存在链路 S_a-S_b 的某个父节点释放的资源加上链路 S_a-S_b 的剩余资源大于等于该链路所有子节点请求资源的总和,表明资源节点 S_a-S_b 的子节点需要等待 S_a-S_b 的这个父节点执行完成,则建立数据流之间的更新关系如图 4(b)所示.

3) 若链路 S_a-S_b 的剩余容量不能满足所有子节点的资源需求,但 S_a-S_b 的多个父操作节点释放的资源总和加上该链路的剩余资源大于等于其所有子节点所需此链路资源的总和,则首先如图 4(c)所示把这些父节点连接起来,再引入从连接之后的任一端节点分别指向此资源节点各子操作节点的有向边.

4) 若链路 S_a-S_b 的剩余容量不能满足其所有子节点的资源需求,但存在 S_a-S_b 的多个父节点的组合,并且每个组合释放的资源总和加上该链路的剩余资源大于等于其所有子节点请求该链路资源的总和,则此时首先计算各组合中父节点的总个数与该组合中每个父节点需求资源节点个数之和,然后选择其

© 中国科学院软件研究所 http://www.jos.org.cn

中和最小的组合中的父节点,把这些父节点连接起来后,引入端节点分别指向此资源节点各子节点的有向边.选择上述父节点的组合作为资源节点 S_a-S_b 所有子操作节点的父节点是因为和最小说明子节点进行更新操作需要依赖的操作节点数量越少,因此子节点就越容易尽早实现路径更新.若有多个组合的上述和相同,则查看不同组合的父节点个数,并直接选择父节点个数最少的那个组合中的父节点为资源节点 S_a-S_b 所有子操作节点的父节点.若父节点的个数仍相同,则查看不同组合父节点之间有无依赖关系,并选择不依赖其他组合中节点的组合为资源节点 S_a-S_b 所有子操作节点的父节点.如图 4(d)所示,流 r_1,r_2 和 r_3 更新完成释放的资源总和或更新 r_5 完成释放的链路资源均能满足更新 r_4 和更新 r_6 所需资源,由于等待"更新 r_5"只需等待一个父节点执行完成即可进行流 r_4 和 r_6 的更新,因此最终选择操作节点"更新 r_5"作为节点"更新 r_4"和"更新 r_6"的父节点.同理,图 4(e)中,由于更新 r_1、更新 r_2 和更新 r_3 相比更新 r_2、更新 r_3 和更新 r_4 需要依赖的节点数量少(设更新 r_4 需要的资源比更新 r_1 的多),选择"更新 r_1"、"更新 r_2"和"更新 r_3"为"更新 r_5"和"更新 r_6"的父节点,把更新 r_1,r_2 和 r_3 连接起来后,分别引入更新 r_3 指向更新 r_5 和更新 r_6 的有向边.



(a) 父节点的资源满足所有子节点的资源需求

(b) 父节点释放的资源加上链路剩余资源满足子节点的资源需求

(c) 多个父节点释放的资源加上链路剩余资源满足子节点的资源需求

(d) 多个父节点组合释放的资源满足子节点的资源需求

(e) 多个父节点组合释放的资源满足子节点的资源需求

Fig.4 Build the relationship between the parent operation and the child operation for a resource node

图 4 建立资源节点的父子操作间的关系

步骤 2.对更新流 r 需要的所有其他链路资源进行步骤 1,建立其父节点和子节点之间的依赖关系.

© 中国科学院软件研究所 http://www.jos.org.cn

步骤 3.对于数据流 r,对其目标路径 $P_d(r)$的每条链路更新关系中涉及流 r 的部分进行合并,即得到流 r 的更新关系.如图 5 中更新 r_6需要 3 个资源节点,对这 3 个资源节点分别建立更新关系如图 5(a)~图 5(c),通过图 5(a)可知,更新 r_6依赖更新 r_1和更新 r_3执行完成;图 5(b)表明,更新 r_6依赖更新 r_1释放资源;图 5(c)中,更新 r_6需要等待更新 r_3及更新 r_4完成.合并图 5(a)~图 5(c),得到流 r_6的更新关系,从此更新关系可知,更新 r_6需依赖更新 r_1、更新 r_3和更新 r_4,即操作节点“更新 r_1”“更新 r_3”和“更新 r_4”是操作节点“更新 r_6”的父节点.

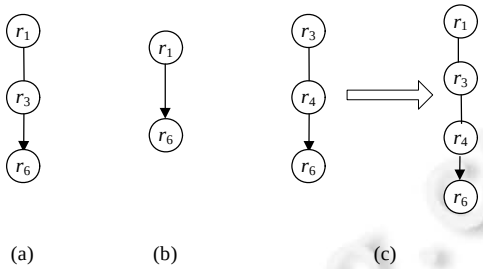


Fig.5 Example of building updater relationship for the flow

图 5 为流建立更新关系示例

对网络中的每个流,按上述描述依次建立更新关系,即得全网数据流的更新关系图.

3.2.4 实例描述

图 6 给出了根据更新调度图为网络中所有数据流建立更新关系图的实例.根据第 3.2.3 节描述的更新关系图建立方法,根据图 6(a)的网络更新调度图,最终为流 r_1~r_7建立数据流之间的更新关系如图 6(b)所示.对于数据流 r_3,从 6(a)可见,更新 r_3既可依赖于流 r_4且流 r_8更新完成,又可依赖于 r_4且 r_5或 r_8且 r_5更新执行完成.利用第 3.2.3 节步骤 1 的描述,由于更新 r_5依赖于更新 r_8,我们最终选择更新 r_4和更新 r_8为操作节点更新 r_3的父节点.

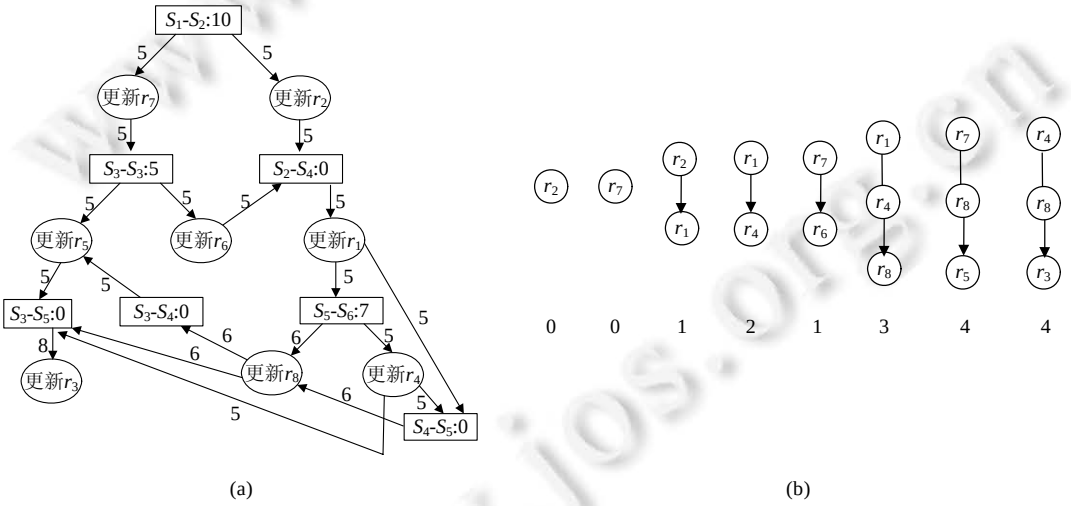


Fig.6 Example of building update dependency graph based on the update scheduling graph

图 6 根据更新调度图建立更新关系图示例

3.3 更新调度过程

为了进行路由更新,控制器首先根据数据流的更新关系图,为每个需要更新的数据流建立更新操作优先级.在具体更新调度过程中,控制器根据各流的优先级高低发送更新调度指令给交换机.优先级越高,相应数据流路由更新操作执行就越优先.具体来说,控制器首先给优先级最高的数据流目标路径上的所有交换机发送更新指

© 中国科学院软件研究所 http://www.jos.org.cn

令,让相应交换机执行流表更新操作.若有多个数据流的更新优先级相同,则控制器同时向它们的目标路径上的交换机发送流表更新指令.只有当上一优先级的流路径更新操作全部完成后,控制器才发送下一优先级数据流的更新指令.数据流更新优先级的确定过程如下.

首先,没有任何父节点的数据流更新操作优先级为 0(最高优先级,数字越大,优先级就越低),仅依赖于优先级 0 的数据流更新操作的优先级设为 1.若要确定操作节点“更新 r”的优先级,首先要确定其所有父节点的优先级.在所有父节点的优先级确定后,依次查看其父节点的优先级,操作节点“更新 r”的优先级在其优先级最低的父节点的优先级的数字基础上加 1.例如,对于图 6(b),“更新 r_2”和“更新 r_7”不依赖于任何更新操作,因此,“更新 r_2”和“更新 r_7”的优先级为 0.由于“更新 r_1”仅依赖于“更新 r_2”执行完成,可以确定“更新 r_1”的优先级为 1.同理,“更新 r_6”的优先级也为 1.由于“更新 r_1”的优先级为 1 且“更新 r_1”是“更新 r_4”的父节点,因此,“更新 r_4”的优先级是 2.同理,“更新 r_8”的优先级在“更新 r_4”的基础上加 1 是 3,“更新 r_5”的优先级为 4,“更新 r_3”的优先级是 4.各数据流的更新操作优先级见图 6(b).更新调度过程的伪代码如算法 2 所示.

算法 2. 更新调度.

```
1.  M=NULL;
2.  for each operation node O_i in UpReGraph(G) do
3.    calculate PRIORITY for each node O_i;
4.  sort nodes by PRIORITY in decreasing order and add them in set Z;
5.  while true do
6.    a:=PRIORITY(The head node in Z);
7.    for each node i(i=1; i≤size(Z), i++) do
8.      if PRI(i)=a then
9.        add i in set M;
10.     else
11.       break;
12.   Schedule nodes in M;
13.   Wait for all scheduled operations to finish;
14.   Delete M from Z;
```

3.4 处理循环

我们发现,有些情况下,按照第 3.2.3 节所述的方法建立更新关系图,会造成因最终的数据流更新关系图处理不当而存在循环.若更新关系图存在循环,就无法利用第 3.3 节所述的方法确定数据流更新操作的优先级.甚至有些情况下,更新关系图中有多个循环纠缠在一起,造成问题更加复杂.如图 7 中,按照上述建立更新关系图的方法,操作节点“更新 r_4”选择“更新 r_5”作为父节点,造成操作节点“更新 r_3”“更新 r_4”和“更新 r_5”出现循环,引起死锁.

对于图 7 的死锁问题,我们可以采取如下方法解除死锁.

对循环中的每个操作节点,按照路径选择和分配的优先顺序依次查看这些节点是否可以选择更新调度图中其他节点作为父节点.若成立,则查看选择其他节点作为父节点后,循环是否解除:如果能够解除,则死锁问题得到解决;否则,按此方法继续查看循环中下一操作节点,直到循环解除为止.如图 7 所示,操作节点“更新 r_3”、“更新 r_4”和“更新 r_5”出现循环,首先查看流 r_3 和 r_5,发现 r_3 和 r_5 无其他父节点可以选择,则继续查看流 r_4,发现操作“更新 r_4”除选择操作节点“更新 r_5”外,还能选择“更新 r_1”和“更新 r_2”作为父节点.由于“更新 r_1”和“更新 r_2”的优先级均可确定,此时死锁解除.

我们还发现,在一些特殊情况下,采用上述选择其他父节点的方法并不能解除死锁.如图 8 中,若更新 r_2 先于更新 r_1 执行完毕,则更新 r_4 和更新 r_1 之间出现死锁.对于这种情况,我们采取改进的 Dionysus 方法[4]解除死锁.具体如下.

© 中国科学院软件研究所 http://www.jos.org.cn

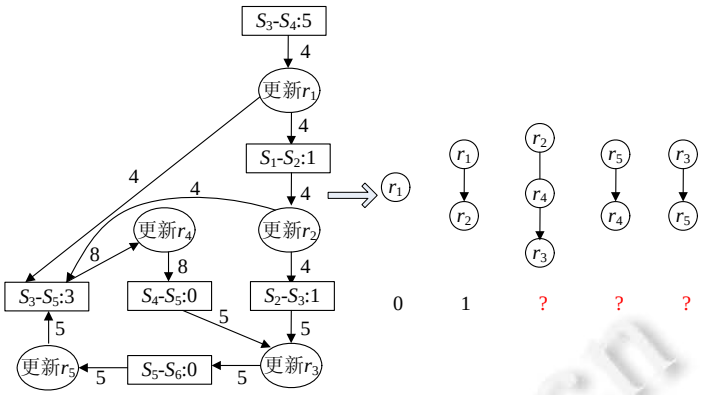


Fig.7 cycle in the update dependency graph

图 7 更新关系图中的循环

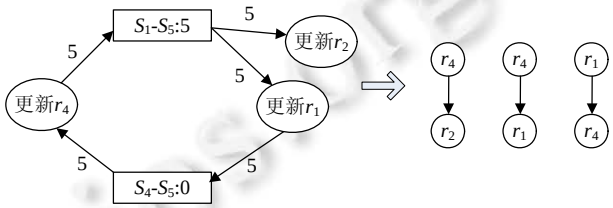


Fig.8 A deadlock example

图 8 死锁示意

- 首先,把更新关系图中所有流的更新关系组合成一幅图,并把组合后的图转换为虚拟 DAG 图.这里用到强连通分量(strongly connected component,简称 SCC)[20]的概念,一个 SCC 被看作 DAG 图中的一个虚拟节点.如果将每个 SCC 视为图中的虚拟节点,那么组合后的更新关系图就成为虚拟 DAG 图.可以通过 tarjan 算法[21]寻找更新关系图中所有的 SCC.当每个 SCC 成为一个虚拟节点后,我们首先利用第 3.3 节所述方法确定虚拟 DAG 图中优先级最高的虚拟节点的更新优先级.然后,对于该虚拟节点内的独立节点,依据更新调度图查看其中哪个节点可以相对最先执行(剩余链路资源大于对该链路的请求资源数量),最先执行的节点优先级等于其所在虚拟节点的优先级,其他内部节点优先级的值在虚拟节点的优先级别基础上按照执行顺序依次加 1.例如,在图 8 的更新关系图中,操作节点“更新 r_4”和“更新 r_1”构成一个 SCC 节点,在确定了该 SCC 的更新优先级之后,通过分析图 8 了解到,“更新 r_1”可先于“更新 r_4”执行(更新 r_1 能够获得链路 S_1-S_5 的剩余资源),那么“更新 r_1”的优先级等于其所在 SCC 的优先级,“更新 r_4”优先级的值等于该 SCC 优先级加 1.
- 其次,用该 SCC 内部优先级最低节点的优点级替换该 SCC 的更新优先级,继续计算 DAG 内其他节点的优先级.此后,每当获得 DAG 内一个虚拟节点的优先级后,就依据上述确定虚拟节点内独立节点优先级的方法确定其所有内部独立节点的优先级;然后,用此 SCC 内部优先级最低节点的优点级替换该 SCC 的更新优先级,继续计算 DAG 内其他节点的优先级,直到所有操作节点优先级确定完毕.例如,按照上述确定优先级的方法,图 8 中更新 r_2 的优先级在更新 r_4 的优先级基础上加 1.

此外,为了防止循环发生,当位于 SCC 内的操作节点和位于此 SCC 外的节点都请求同一资源时,若所请求的资源也位于 SCC 循环之内,则规定此资源优先分配给位于该 SCC 内的操作节点.若不能满足,则此 SCC 内的操作节点只能从独立节点请求资源.这种方法可以防止在满足 SCC 内部操作节点的资源需求前,把 SCC 内的资源分配给 SCC 外的节点引起死锁(如更新 r_2).

© 中国科学院软件研究所 http://www.jos.org.cn

4 性能分析

4.1 仿真环境和性能评价指标

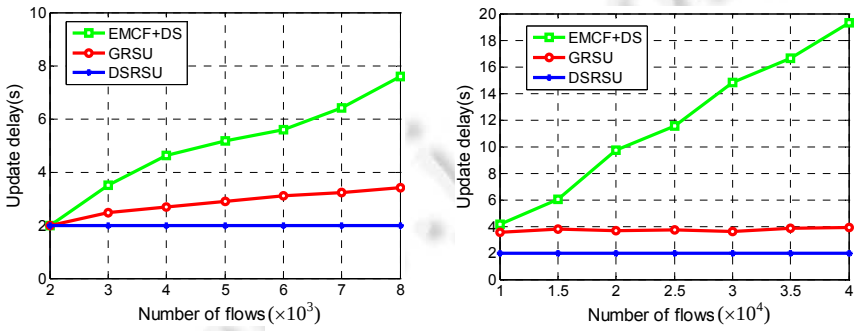
仿真实验中,我们选择具有不同网络大小的两种网络拓扑结构[16]:第 1 种网络拓扑结构(拓扑结构 *a*)包含 20 个交换机和 74 条链路,第 2 种网络拓扑(拓扑结构 *b*)包含 100 个交换机以及 397 条链路.对于两种拓扑,设每条链路的容量均为 100Mbps.文献[22]表明,通常,不足 20%的数据流占据了网络总流量的 80%.为此,仿真分析中,我们产生不同数量的数据流,每个流的大小服从上述 2-8 分布.我们基于 Mininet[23]对所提 DSRSU 策略进行性能分析,并把 DSRSU 策略与 GRSU 策略[16]、OSPF 机制、EMCF+DS 进行性能对比.GRSU 是文献[16]提出的针对 SDN 路由更新延迟时间保证的思想.OSPF 是典型的 TCP/IP 网络路由选择协议,始终选择网络中的最短路径充当数据流的路由且并不进行路由更新.对于 EMCF+DS 策略,众多文献提出 SDN 网络控制器首先利用不同的路由算法,如 MCF 算法[24],根据网络当前工作量为各数据流计算目标路由配置;然后,再利用更新调度算法,如 Dionysus[4],把网络从当前路由更新为目标路由配置.此时,由于控制器需要更新所有的流,包括大象流和老鼠流,造成路由更新的延迟大[16].一种改进的策略是只更新大象流的路由[18],表示成 EMCF,并同样利用 Dionysus 方法进行路由调度更新,这种联合策略称为 EMCF+DS.由于本文关注于实时路由更新,我们采用以下两个指标来分析各路由更新调度策略的更新效率和性能.

- 路由更新延迟:通过各路由更新策略,把网络从当前路由配置更新到目标路由配置所需时长.
- 链路负载率(link load ratio,简称 LLR):链路负载率定义为 $LLR=\max\{n(e)/c(e),e\in E\}$,其中,$e$ 代表链路,$c(e)$是链路 e 的容量,$n(e)$为链路 e 的流量负载.

实验中,设更新延迟容忍值 T_0 的默认值是 2s,λ初始值是 0.65.每个仿真执行 100 次,最后结果取每次执行结果的平均值.

4.2 不同流的数量对更新延迟的影响

本组实验讨论数据流数量的变化对路由更新延迟的影响.实验中,我们把两种网络拓扑中的数据流数量逐渐增加,得到的实验结果如图 9(a)和图 9(b)所示.两幅图均显示,当数据流数量增加时,由于需要进行路由更新的数据流数量变大,因此 EMCF+DS 策略的路由更新延迟逐渐增加.图 9(b)表明,在规模较大的拓扑结构 *b* 中,当数据流的数量增加到 40KB 时,使用 EMCF+DS 进行路由更新的延迟超过了 19s.这表明,尽管在 EMCF+DS 策略中控制器仅更新大象流,路由更新的延迟仍远远大于延迟容忍值 T_0.由于 DSRSU 设置了路由更新延迟容忍值,令路由更新的延迟不超过该值,所以随着网络中数据流数量变化,该策略的路由更新延迟保持 T_0 默认值不变.DSRSU 策略在满足更新延迟容忍值前提下,通过路由更新调度进一步加快了路由更新的速度,所以 DSRSU 的更新延迟容忍值比 GRSU 策略要低.



(a) 拓扑结构 *a* (b) 拓扑结构 *b*

Fig.9 Number of flows vs. route update delay

图 9 流的数量对路由更新延迟

© 中国科学院软件研究所 http://www.jos.org.cn

4.3 延迟容忍值对网络性能的影响

本组实验讨论延迟容忍值的大小对网络性能的影响.当网络中流总数量恒定时,我们把GRSU和DSRSU策略的延迟容忍值分别从0变化到4s,得到的实验结果如图10和图11所示.图10和图11都表明,在拓扑结构*a*和*b*中,DSRSU和GRSU的链路负载率均随着路由更新延迟容忍值T_0的增大而降低.这是因为,T_0越大,两种策略在延迟容忍值内更新的数据流总数量就越大,也就越有利于实现网络负载均衡.由于OSPF始终选择网络中源到目的的最短路径为数据流的路由,并不进行路由更新,因此当数据流数量不变时,OSPF策略的链路负载率不变.当数据流总数量增多时,从图10(a)到图10(b)和图11(a)到图11(b)显示,因为此时需要更新的数据流总量多了,在T_0内能够被更新的数据流占总数据流的比重减小,当T_0相同时,DSRSU和GRSU策略的链路负载率也增大了.无论数据流数量如何变化,在同一数据流数量的前提下,两种拓扑结构中,OSPF策略的网络性能始终低于其他策略(链路负载率最高).这主要是由于OSPF不进行路由更新,致使网络中某些链路的负载率大且某些链路可能拥塞的原因.此外,因为EMCF+DS策略始终需要为网络中所有大象流更新路由,完全更新这些大象流需要的时间大于4s,所以,T_0的变化对EMCF+DS策略的链路负载率无影响.两幅图中,随着T_0的变化,我们的策略开始链路负载率略高于GRSU;而随着更新延迟容忍值的变大,DSRSU链路负载率变得低于GRSU.原因主要在于,当更新延迟容忍值逐渐变大时,更新的数据流数量变大,我们的策略路由选择阶段在确定数据流新路径后回收期旧路径的资源,相对于GRSU对旧路径资源不回收以防止拥塞,本策略相对于GRSU更加高效地利用链路资源,使得实际能够进行路由更新的数据流数量增多,因此更容易实现负载均衡.

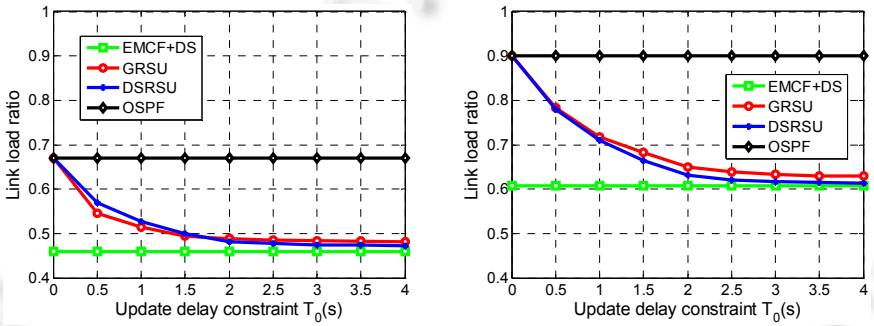
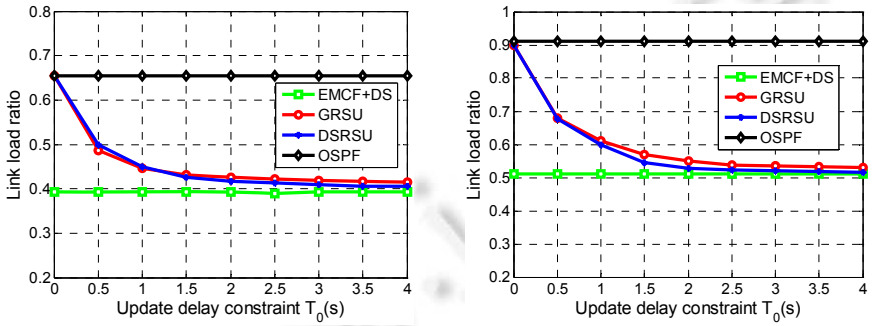


Fig.10 Route update delay constrains vs. link load ratio in topology *a*

图10 拓扑*a*中路由更新延迟容忍值对链路负载率



(a) 20 000个流 (b) 40 000个流

Fig.11 Route update delay constrains vs. link load ratio in topology *b*

图11 拓扑*b*中路由更新延迟容忍值对链路负载率

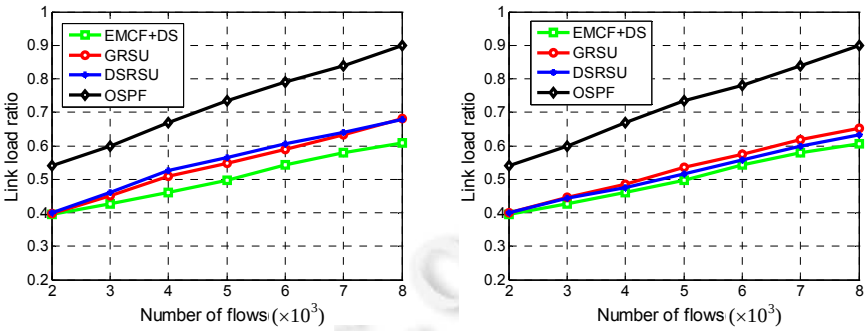
图10显示,当网络规模较小时,相对于EMCF+DS,本文的DSRSU策略能够降低EMCF+DS策略约69%的路由更新延迟,却达到与EMCF+DS策略相近的网络性能(链路负载率比EMCF+DS低2.1%).例如,当网络中数

© 中国科学院软件研究所 http://www.jos.org.cn

据流数量是 4 000 时,图 9(a)表明,EMCF+DS 策略的路由更新延迟约为 4.3s;而图 10(a)显示,DSRSU 仅需要 2s,就能达到与 EMCF+DS 相似的网络链路负载率.在图 11 中,当网络规模较大时,我们发现,利用 DSRSU 策略进行流路由更新,相对于 EMCF+DS,能够在降低 EMCF+DS 策略约 74.5%路由更新延迟的前提下,具有和 EMCF+DS 策略相近的链路负载率.

4.4 数据流总数量对网络性能的影响

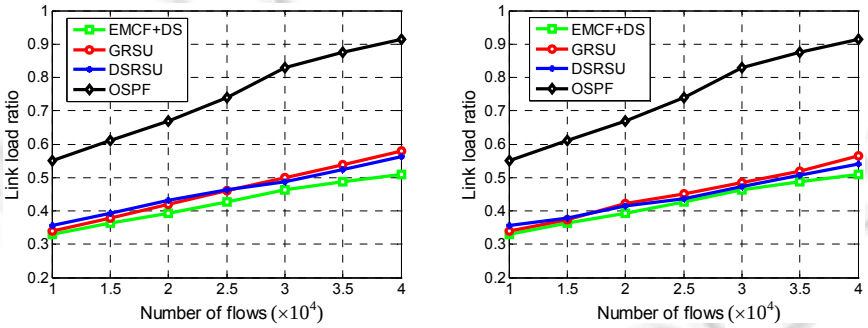
本组实验中,对于两种不同的网络拓扑结构,我们在依次设定路由更新延迟容忍值 T_0=1(s)和 T_0=2(s)的前提下,把网络中数据流的数量逐渐增大,得到的实验结果如图 12 和图 13 所示.



(a) T_0=1s (b) T_0=2s

Fig.12 Link load ratio vs. number of flows in topology *a*

图 12 拓扑 *a* 中流数量的变化对链路负载率



(a) T_0=1s (b) T_0=2s

Fig.13 Link load ratio vs. number of flows in topology *b*

图 13 拓扑 *b* 中流数量的变化对链路负载率

图 12 和图 13 均表明,在路由更新延迟容忍值一定的情况下,4 种策略的链路负载率均随着网络中数据流总数量的变大呈逐渐增大的趋势,且 OSPF 的链路负载率远高于其他 3 种策略.其原因是,对于 OSPF,当网络中数据流逐渐增多时,由于有更多的数据需要放在网络上传输,网络的负载逐渐加重.并且,OSPF 不进行路由更新,因此链路负载率高的那些链路始终保持高链路负载率居高不下,造成 OSPF 的链路负载率较高.对于 EMCF+DS、GRSU 和 DSRSU 策略,随着数据流数量增大,实现路由更新的数据流数量占总数据流总量的比率降低了,路由更新的效果下降,因此链路负载率升高.从图 12(a)到图 12(b)以及图 13(a)到图 13(b)的变化可见,在数据流数量相同的条件下,T_0=2 相对于 T_0=1 时 GRSU 和 DSRSU 策略的链路负载率均下降.这是因为 T_0 增大时,路由更新延迟容忍值内可以更新更多数据流的结果.此外,图 12 和图 13 显示,在路由更新延迟容忍值恒定不变的前提下,随着网络中数据流数量的增多,EMCF+DS、GRSU 和 DSRSU 策略的网络性能比较接近.当路由更新延迟容忍值 T_0=2 时,本文 DSRSU 策略的链路负载率略始终低于 GRSU.这是因为,我们的策略不仅在路径选择时强调链路负载均

© 中国科学院软件研究所 http://www.jos.org.cn

衡,同时相对于 GRSU 强调链路资源的回收,使得实际由于链路资源不满足要求致使无法进行路由更新的情况减少了.另外,具体来讲,对于拓扑结构 *a*,DSRSU 策略在 T_0=2 时就能达到与 EMCF+DS 策略类似的网络性能;而在拓扑结构 *a* 中,EMCF+DS 的更新延迟却需要 5s~8s.图 13(b)表明,本文的 DSRSU 策略在 T_0=2 时达到与 EMCF+DS 策略花费 11s~16s 的路由更新延迟类似的链路负载率.其主要原因在于,DSRSU 不仅在选路时考虑了链路负载均衡,而且通过建立更新关系图的方式挖掘各数据流路由更新调度的顺序来尽量降低路由更新延迟,使得路由更新能够尽快完成.

4.5 链路负载因子对网络性能的影响

本实验讨论路径分配中的链路负载因子λ对网络性能的影响.在其他参数不变的情况下,把链路负载因子λ从 0.2 逐渐变化到 1,得到的实验结果如图 14 所示.由图 14 可见,在拓扑结构 *a* 和拓扑结构 *b* 中,当λ很小时,网络链路负载率高.这是因为λ较小,导致数据流在路由更新的选路阶段,由于链路负载的限制,能够被选择充当新路径的链路很少,造成很多数据流由于找不到满足条件的新路由而无法完成路径的更新.此时,几乎无法通过路由更新来降低链路负载率.当λ逐渐增大时,链路负载大为降低,因为此时很多数据流能够完成新路径的选择,从而路由更新可以进行了.图中当λ达到 0.6 时,网络链路负载率最小,而之后,随着链路负载因子的增大,链路负载率提高.这是由于链路负载因子较大会导致某些链路在路由更新中负载较重的结果.

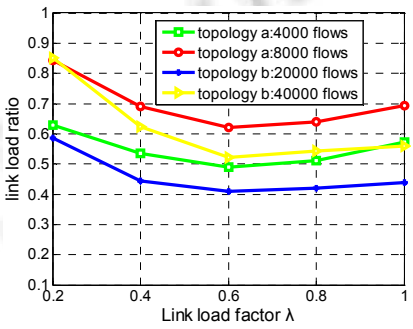


Fig.14 Link load factor vs. link load rate

图 14 链路负载因子对链路负载率

5 结 论

本文研究软件定义网络中实时路由更新问题,在同时考虑 TCMA 的更新速度、当前网络工作负载和各个交换机流表更新速度差异的前提下,提出延迟满足的路由选择和调度更新策略(delay satisfied route selection and updating scheme,简称 DSRSU),从路径选择和路由更新调度两个方面联合进行优化,以降低路由更新延迟,实现实时路由更新.大量仿真结果表明,DSRSU 策略具有高效性,能够在保证网络性能的前提下,极大地降低路由更新的延迟.

References:

[1] McKeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L, Rexford J, Shenker S, Turner J. Openflow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 2008,38(2):69−74.

[2] Kandula S, Sengupta S, Greenberg A, Patel P, Chaiken R. The nature of data center traffic: Measurements & analysis. In: Proc. of the 9th ACM SIGCOMM Conf. on Internet Measurement Conf. New York: ACM Press, 2009. 202−208.

[3] Zhang CK, Cui Y, Tang HY, Wu JP. State-of-the-art survey on software-defined networking (SDN). Ruan Jian Xue Bao/Journal of Software, 2015,26(1):62−81 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4701.htm [doi: 10.13328/j.cnki.jos.004701]

[4] Jin X, Liu HH, Gandhi R, Kandula S, Mahajan R, Zhang M, Rexford J, Wattenhofer R. Dynamic scheduling of network updates. In: Proc. of the 2014 ACM Conf. on SIGCOMM. New York: ACM Press, 2014. 539−550.

[5] Hong CY, Kandula S, Mahajan R, Zhang M, Gill V, Nanduri M, Wattenhofer R. Achieving high utilization with software-driven wan. In: Proc. of the Process of ACM SIGCOMM. New York: ACM Press, 2013. 15−26.

© 中国科学院软件研究所 http://www.jos.org.cn

[6] Kannan K, Banerjee S. Compact tcam: Flow entry compaction in tcam for power aware SDN. In: Proc. of the Process of Distributed Computing and Networking. Springer-Verlag, 2013. 439−444.
[7] Wang ST, Li D, Xia ST. The problems and solutions of network update in SDN: A survey. In: Proc. of the Int'l Workshop of Software-defined Data Communications and Storage (SDDCS). IEEE Press, 2015. 474−479.
[8] Reitblatt M, Foster N, Rexford J, Schlesinger C, Walker D. Abstractions for network update. In: Proc. of the ACM SIGCOMM 2012 Conf. on Applications, Technologies, Architectures. New York: ACM Press, 2012. 323−334.
[9] McGeer R. Safe A. Efficient update protocol for OpenFlow networks. In: Proc. of the ACM SIGCOMM HotSDN Workshop. New York: ACM Press, 2012. 61−66.
[10] Katta NP, Rexford J, Walker D. Incremental consistent updates. In: Proc. of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking. New York: ACM Press, 2013. 49−54.
[11] Canini M, Kuznetsov P, Levin D, Schmid S. A distributed and robust sdn control plane for transactional network updates. In: Proc. of the IEEE Infocom. New York: IEEE Press, 2015. 1−9.
[12] Ghorbani S, Caesar M. Walk the line: Consistent network updates with bandwidth guarantees. In: Proc. of the ACM SIGCOMM HotSDN Workshop. New York: ACM Press, 2012. 67−72.
[13] Liu HH, Wu X, Zhang M, Yuan L, Wattenhofer R, Maltz DA. zUpdate: Updating data center networks with zero loss. In: Proc. of the ACM SIGCOMM. New York: ACM Press, 2013. 1−12
[14] Mizrahi T, Moses Y. Software defined networks: It's about time. In: Proc. of the IEEE Infocom. New York: IEEE Press, 2016. 1−9.
[15] Mizrahi T, Rottenstreich O, Moses Y. Timeflip: Scheduling network updates with timestamp-based tcam ranges. In: Proc. of the IEEE Infocom. New York: IEEE Press, 2015. 1−10.
[16] Xu HL, Yu ZL, Li XY, *et al*. Real-time update with joint optimization of route selection and update scheduling for SDNs. In: Proc. of the IEEE ICNP Conf. New York: IEEE Press, 2016. 1−10.
[17] Al-Fares M, Radhakrishnan S, Raghavan B, Huang N, Vahdat A. Hedera: Dynamic flow scheduling for data center networks. In: Proc. of the NSDI. ACM Press, 2010. 19−29.
[18] Narayanan R, Kotha S, Lin G, Khan A, Rizvi S, Javed W, Khan H, Khayam SA. Macroflows and microflows: Enabling rapid network innovation through a split sdn data plane. In: Proc. of the IEEE Software Defined Networking Worksop. New York: IEEE Press, 2012. 79−84.
[19] Jain S, Kumar A, Mandal S, Ong J, Poutievski L, Singh A, Venkata S, Wanderer J, Zhou J, Zhu M. B4: Experience with a globally-deployed software defined WAN. In: Proc. of the ACM SIGCOMM. New York: ACM Press, 2013. 1−12.
[20] 2018. https://en.m.wikipedia.org/wiki/Strongly_connected_component
[21] Lower G. Concurrent depth-first search algorithms based on Tarjan's algorithm. Int'l Journal on Software Tools for Technology Transfer, 2016,18(2):129−147.
[22] Curtis AR, Mogul JC, Tourrilhes J, Yalagandula P, Sharma P, Banerjee S. Devoflow: Scaling flow management for highperformance networks. ACM SIGCOMM Computer Communication Review, 2011,41(4):254−265.
[23] 2018. The mininet platform. http://mininet.org/
[24] Even S, Itai A, Shamir A. On the complexity of time table and multi-commodity flow problems. In: Proc. of the 16th Annual Symp. on Foundations of Computer Science. IEEE, 1975. 184−193.

附中文参考文献:

[3] 张朝昆,崔勇,唐騫祎,吴建平.软件定义网络(SDN)研究进展.软件学报,2015,26(1):62−81. http://www.jos.org.cn/1000-9825/4701.htm [doi: 10.13328/j.cnki.jos.004701]

朱金奇(1980－),女,天津人,博士,副教授,主要研究领域为无线传感器网络,车联网,智能感知,软件定义网络.

黄永鑫(1996－),男,硕士生,主要研究领域为软件定义网络.

孙华志(1961－),男,博士,教授,主要研究领域为分布式计算.

刘明(1972－),男,博士,教授,博士生导师,CCF 专业会员,主要研究领域为无线传感器网络,智能感知,深度学习.

© 中国科学院软件研究所 http://www.jos.org.cn