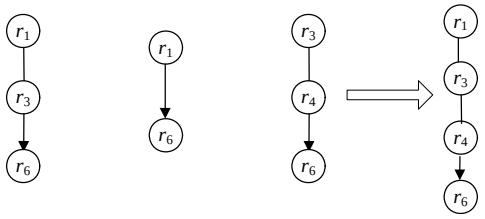
步骤 3.对于数据流 r,对其目标路径 $P_d(r)$的每条链路更新关系中涉及流 r 的部分进行合并,即得到流 r 的更新关系.如图 5 中更新 r_6需要 3 个资源节点,对这 3 个资源节点分别建立更新关系如图 5(a)~图 5(c),通过图 5(a)可知,更新 r_6依赖更新 r_1和更新 r_3执行完成;图 5(b)表明,更新 r_6依赖更新 r_1释放资源;图 5(c)中,更新 r_6需要等待更新 r_3及更新 r_4完成.合并图 5(a)~图 5(c),得到流 r_6的更新关系,从此更新关系可知,更新 r_6需依赖更新 r_1、更新 r_3和更新 r_4,即操作节点“更新 r_1”“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



(a) (b) (c)

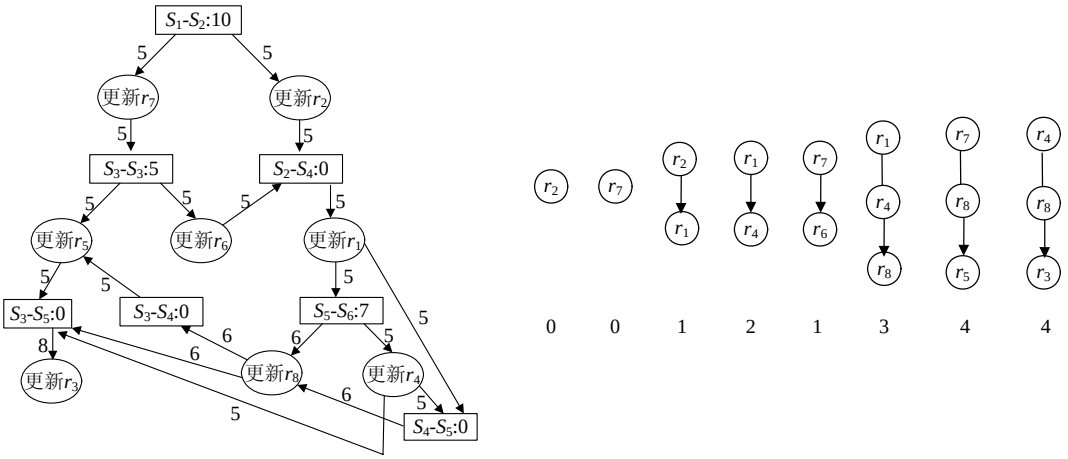
Fig.5 Example of building updater relationship for the flow

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.



(a) (b)

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

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.

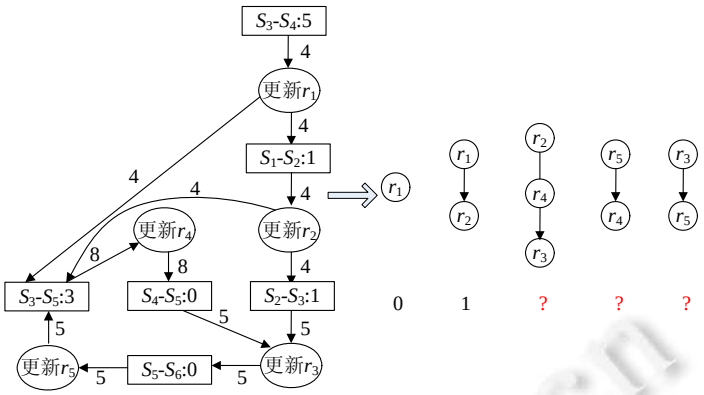


Fig.7 cycle in the update dependency graph

图 7 更新关系图中的循环

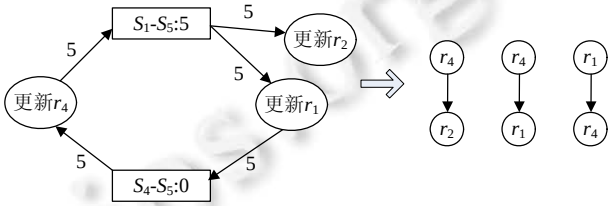


Fig.8 A deadlock example

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).

4 性能分析

4.1 仿真环境和性能评价指标

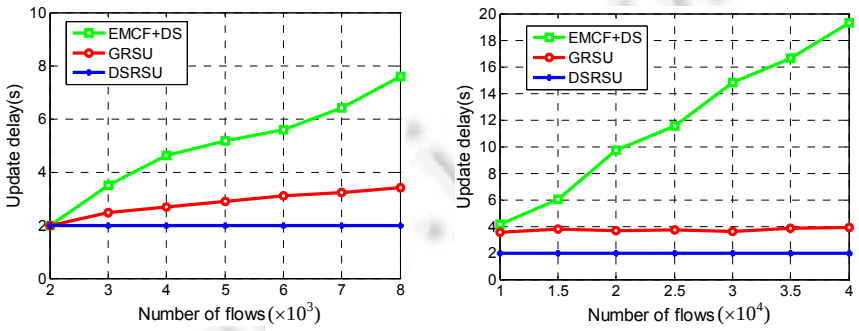
仿真实验中,我们选择具有不同网络大小的两种网络拓扑结构[16]:第 1 种网络拓扑结构(拓扑结构 *a*)包含 20 个交换机和 74 条链路,第 2 种网络拓扑(拓扑结构 *b*)包含 100 个交换机以及 397 条链路.对于两种拓扑,设每条链路的容量均为 100Mbps.文献[22]表明,通常,不足 20%的数据流占据了网络总流量的 80%.为此,仿真分析中,我们产生不同数量的数据流,每个流的大小服从上述 2-8 分布.我们基于 Mininet[23]对所提 DSRSU 策略进行性能分析,并把 DSRSU 策略与 GRSU 策略[16]、OSPF 机制、EMCF+DS 进行性能对比.GRSU 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

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.

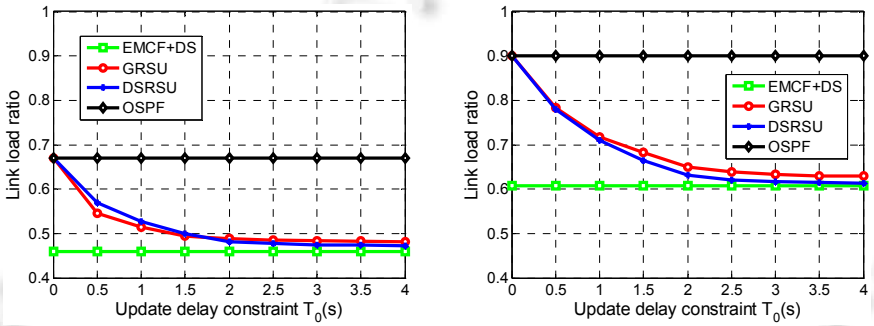
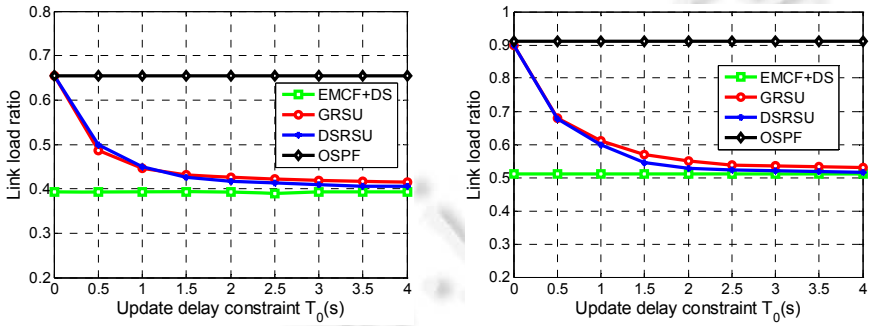


Fig.10 Route update delay constrains vs. link load ratio in topology *a*

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*

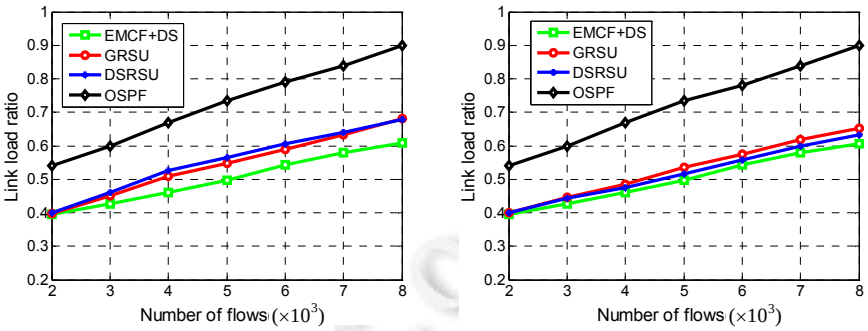
图 11 拓扑 *b* 中路由更新延迟容忍值对链路负载率

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

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

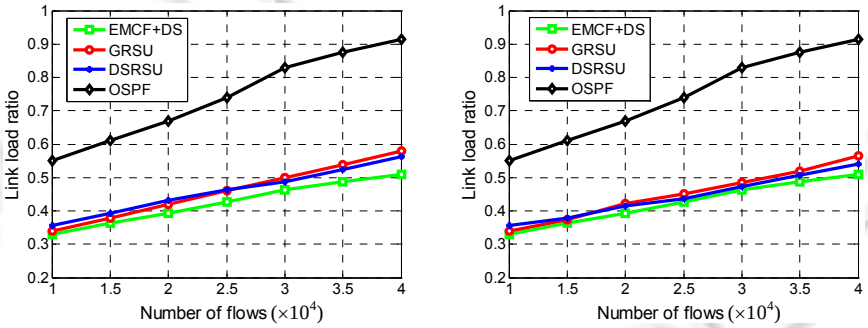
本组实验中,对于两种不同的网络拓扑结构,我们在依次设定路由更新延迟容忍值 T_0=1(s)和 T_0=2(s)的前提下,把网络中数据流的数量逐渐增大,得到的实验结果如图 12 和图 13 所示.



(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

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.

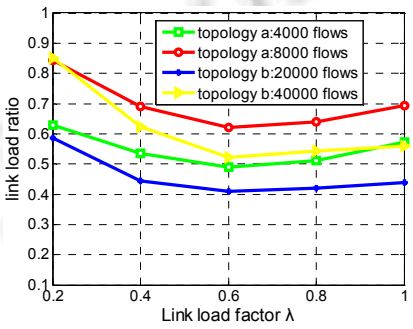


Fig.14 Link load factor vs. link load rate

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.

[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.