软件定义网络中快速和一致的流更新策略

史久根 杨 旭* 刘雅丽 孙 立
(合肥工业大学计算机与信息学院 合肥 230009)

摘 要：在软件定义网络中，为了实现各种网络性能优化目标，控制面需要频繁的对数据面进行更新。然而，由于数据面的异步性，不合理的更新将严重降低网络性能。针对此问题，该文提出一种快速和一致的流更新策略(FCFU)。该策略通过流分段减弱其原有的强依赖关系，使能并行更新，通过分析子流段与多个资源间的依赖关系得到总更新轮数较少的更新安排，最后基于延时队列完成一致性流更新。实验结果表明，与现有的流更新算法相比，该策略能够缩短流更新总时间达20.6%，同时保证了更新期间无拥塞和包乱序等问题的发生。

关键词：软件定义网络；流更新；一致性更新；无拥塞；包乱序
中图分类号：TP393 **文献标识码**：A **文章编号**：1009-5896(2021)09-2617-07
DOI: 10.11999/JEIT200231

Fast and Consistent Flow Update in Software Defined Network

SHI Jiugen YANG Xu LIU Yali SUN Li
(*School of Computer and Information*, *Hefei University of Technology*, *Hefei* 230009, *China*)

Abstract: In Software Defined Networks (SDN), in order to meet various network performance optimization goals, the control plane needs to frequently update the data plane. However, due to the asynchronous nature of the data plane, unreasonable updates will severely degrade network performance. To address this issue, a Fast and Consistent Flow Update (FCFU) strategy is proposed, which weakens the original strong dependency through flow segmentations and enables parallel updates. By analyzing the dependency relationship between sub-flow segments and multiple resources, the update schedule with less numbers of rounds is obtained. Finally, consistent flow update is achieved based on the delay queue. Experimental results show that, compared with the existing flow update algorithms, this strategy can shorten the total completion time of flow update by 20.6%, while ensuring that no congestion and packet reordering occur during the update period.

Key words: Software Defined Network (SDN); Flow update; Consistent update; Congestion-free; Packet reordering

1 引言

近年来，软件定义网络(SDN)[1]因拥有全局的网络视图和灵活的管理能力，在流量工程[2]、故障恢复和负载均衡等方面发挥着独特的优势。SDN将网络解耦为控制面和数据面，并通过OpenFlow等面向协议进行交互管理。流更新在链路故障、网络拥塞、虚拟机迁移[3]和设备维护等情况下频繁发生，然而不合理的更新将严重降低网络性能。

流更新要求将一些流从旧路径切换到新路径上，在数据面表现为分别在一些交换机上添加新规则和删除旧规则。然而，由于链路传播时延、交换机性能和规则安装时间不定等因素[4,5]，使得更新期间产生了不一致的行为，如转发黑洞、转发回环、拥塞和包乱序等。Reitblatt等人[6]首次分析了在SDN中进行网络更新的问题，并提出了每包一致性概念，即在流更新期间，数据包要么按照旧规则转发要么按照新规则转发，不能同时按照混合规则转发。接着提出了两阶段更新方法，即通过在交换机中同时安装新旧两套规则来分别匹配头部设置了新旧标签的数据包，以此完成更新。尽管该方法可以保证无转发黑洞和无转发回环，但没有考虑多流更新时的拥塞问题，而且两套规则极大占用了交换机中昂贵的TCAM存储空间。

基于两阶段更新方法，文献[7,8]提出了基于线性规划(LP)的无拥塞更新方案。前者思想是如果每条链路都预留一定的空闲带宽，通过多路径变换则总存在更新计划。后者通过找到满足约束条件的中间路径，从旧路径过渡到中间路径后最终完成更新

收稿日期：2020-04-03；改回日期：2020-09-22；网络出版：2021-08-09
*通信作者： 杨旭 2018110930@mail.hfut.edu.cn
基金项目：国家重大科学仪器设备开发专项(2013YQ030595)
Foundation Item: The National Major Scientific Instruments Development Project (2013YQ030595)

到新路径。为了在更新期间同时满足用户的各种特定需求，如设定最长更新时间和最大丢包率等，文献[9]还提出了可定制的无拥塞更新方案。然而，这些方法计算出更新安排的速度太慢，而且无法扩展到大规模网络中。

文献[10–12]提出基于构造依赖图的方法安排网络更新。文献[10]以规则更新操作、链路资源和交换机资源为结点构造全局依赖图。为了减小依赖图的规模和加快更新速度，文献[11]将流分割为许多子流段并提出了关键结点，并基于关键结点构造局部依赖图。文献[12]将流的子段和链路资源作为结点构造全局依赖图。然而，尽管这些方法保证了网络更新期间的一致性，构造和解析依赖图仍将耗费大量的时间，而且以上方法均没有考虑网络更新期间出现的包乱序问题。

综上，本文针对流更新期间出现的耗时和更新不一致等问题，提出了在软件定义网络中快速和一致的流更新策略(FCFU)。该策略首先通过流分段将单条流分割为多个子流段，这不仅减弱了原有的强依赖关系而且能加速更新过程。然后通过分析子流段与多个资源间的依赖关系得到总更新轮数较少的更新安排。最后，为了避免包乱序，利用延时队列完成一致性流更新。

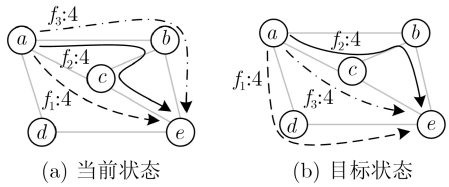
2 问题提出与系统模型

本节将首先描述网络更新问题，随后提出系统模型。

2.1 问题描述

在SDN中，流规则包括匹配域、动作域和统计信息域。当数据包到达交换机后，交换机通过匹配流规则来执行对应的动作，如转发到控制器、某端口或丢弃数据包。流更新的本质是各交换机中规则的更新。规则更新分为添加规则、替换规则和删除规则。在更新期间，网络中的数据传输不终止，且依旧按原来的状态转发。因此必须避免更新期间数据包的异常行为。然而，数据面的异步性将导致转发黑洞、转发回环和拥塞问题。由于链路时延的存在，也将出现包乱序问题。下面以图1为例加以说明。

图1中各链路带宽为10 Mbps，3条流的传输速率均为4 Mbps。当链路$a\to b$发生故障后，控制器将流f_1, f_2, f_3从图1(a)所示当前状态更新到图1(b)所示目标状态。



(a) 当前状态 (b) 目标状态

图 1 流更新示例

(1)转发黑洞问题。

为了将流f_1从旧路径$a\to c\to e$更新到新路径$a\to d\to e$，当控制器同时下发规则更新命令后，若交换机a先于交换机d完成更新，则到达交换机d的数据包将被丢弃。

(2)转发回环问题。

为了将流f_2从旧路径$a\to b\to c\to e$更新到新路径$a\to c\to b\to e$，若交换机c先于交换机a和b完成更新，则将形成回环$a\to b\to c\to b$。

(3)拥塞问题。

当同时更新这3条流时，若f_3先于f_1完成更新，则链路$c\to e$或链路$a\to c$将同时流经3条流，总流量为12 Mbps，超过了链路最大带宽10 Mbps，将造成链路拥塞。

(4)包乱序问题。

设链路$c\to e$的传播时延为10 ms，其余链路均为5 ms。当流f_1从旧路径切换到新路径后，由于新路径时延小于旧路径，若沿新路径转发的数据包已到达交换机e而沿旧路径转发的数据包未完全到达，则将造成部分数据包的乱序。对于TCP协议，若因包乱序而触发了超时重传机制，将导致发送方降低发送速率[13]。对于UDP协议，传输的视频流对乱序异常敏感，乱序不仅会影响视频质量，严重时还会发生解码错误[14]。

2.2 系统模型

图$G=(V,E)$表示SDN网络模型，其中V表示交换机的集合，$|V|=n$。E表示链路的集合，$|E|=m$。对交换机$\forall v\in V$, L_v表示其流表的最大容量。对链路$\forall(u,v)\in E$，$C_{u,v}$和$d_{u,v}$分别表示链路(u,v)的最大带宽和传播时延。而F表示欲更新的流集合，$|F|=r$。对流$\forall f\in F$, b_f表示流f的传输速率，P_f和P'_f表示流f更新前和更新后的转发路径。流更新问题在于将集合F中的所有流从旧路径快速、一致地更新到新路径上，同时在更新期间保证无转发黑洞、转发回环、拥塞和包乱序等问题的发生。

在有链路容量约束的情况下，流更新过程至少需要1轮完成，最多需要r轮完成。令$U=\{u_1,u_2,\cdots,u_r\}$表示最终的更新安排，其中u_j表示第j $(1\le j\le r)$轮需更新的流集合。

为了增加更新的并行性，采用文献[11]的方法将单个流分割为多个子流段，每个子流段可独立更新，各子流段间可并行更新。以图1中流f_2为例，流f_2从旧路径更新到新路径可表示为$(a\to b\to c\to e)\to(a\to c\to b\to e)$，经过流分段后可得两个子流段$(a\to b\to c)\to(a\to c)$和$(c\to e)\to(c\to b\to e)$。以$R_f$表

示对流f分割后子流段的集合，对$\forall s(f,i)\in R_f$，$s(f,i)$表示流f的第i个子流段，其传输速率与流f一致，以$p_{f,i}$和$p'_{f,i}$表示$s(f,i)$更新前和更新后的转发路径。则流f的旧(新)路径与其子流段的旧(新)路径满足关系：

$$P_f=\bigcup_{i=1}^{|R_f|}p_{f,i},\forall f\in F \tag{1}$$

$$P'_f=\bigcup_{i=1}^{|R_f|}p'_{f,i},\forall f\in F \tag{2}$$

对F中所有的流分割后总的子流段集合为

$$S=\bigcup_{\forall f\in F}R_f \tag{3}$$

对流分段后，流更新问题转化为将集合S中的所有子流段从旧路径快速、一致地更新到新路径上，同时在更新期间保证无转发黑洞、转发回环、拥塞和包乱序等问题的发生。

设$x_{(f,i,j)},j\in\{1,2,\cdots,r\}$为二值变量，表示子流段$s(f,i)$是否在第$j$轮更新。

$$x_{(f,i,j)}=\begin{cases}1,\ s(f,i)\text{在第}j\text{轮更新}\\0,\ \text{其他}\end{cases} \tag{4}$$

其满足式(5)的约束，表示$s(f,i)$在r轮内完成了更新。

$$\sum_{j=1}^{r}x_{(f,i,j)}=1,i\in(1,2,\cdots,|R_f|),\forall f\in F \tag{5}$$

设$\hat{u}_j,j\in\{1,2,\cdots,r\}$为二值变量，表示是否需进行第$j$轮更新。若集合$u_j$不为空，则表明需进行第$j$轮更新。易得$|u_j|=\sum x_{(f,i,j)},\forall s(f,i)\in S$。

$$\widehat{u}_j=\begin{cases}1,|u_j|\geq 1\\0,\ \text{其他}\end{cases} \tag{6}$$

以$h_j(u,v),\forall(u,v)\in E,j\in\{1,2,\cdots,r\}$表示在进行第$j$轮更新时，链路$(u,v)$的初始总负载。$h_1(u,v)$表示链路$(u,v)$最初的总负载。以$k_{f,i}(u,v)\in\{0,1\}$表示子流段$s(f,i)$的旧路径是否包含链路$(u,v)$，以$k'_{f,i}(u,v)\in\{0,1\}$表示子流段$s(f,i)$的新路径是否包含链路$(u,v)$。

$$k'_{f,i}(u,v)=\begin{cases}1,\ p'_{f,i}\text{包含链路}(u,v)\\0,\ \text{其他}\end{cases} \tag{7}$$

则在更新期间，为了避免链路拥塞，流经各个链路的总流量不能超过其最大带宽。需满足式(8)约束：

$$\begin{aligned}&h_j(u,v)+\sum_{\forall f\in F}\sum_{i=1}^{|R_f|}b_f k'_{f,i}(u,v)x_{(f,i,j)}\leq C_{u,v},\\&j\in\{1,2,\cdots,r\},\forall(u,v)\in E\end{aligned} \tag{8}$$

当第j轮更新完成后，移入链路(u,v)的流将占用其带宽，移出链路(u,v)的流将释放已占用的带宽，若子流段$s(f,i)$的新旧路径均包含链路(u,v)，则对应带宽不作变化。第j+1轮各链路初始总负载可按式(9)得到。

$$\begin{aligned}h_{j+1}(u,v)=&h_j(u,v)+\sum_{\forall f\in F}\sum_{i=1}^{|R_f|}b_f k'_{f,i}(u,v)x_{(f,i,j)}\\&-\sum_{\forall f\in F}\sum_{i=1}^{|R_f|}b_f k_{f,i}(u,v)x_{(f,i,j)},\\&j\in\{1,2,\cdots,r-1\},\forall(u,v)\in E\end{aligned} \tag{9}$$

以$w_j(v),\forall v\in V,j\in\{1,2,\cdots,r\}$表示在进行第$j$轮更新时，交换机$v$流表的初始已用容量。$w_1(v)$表示交换机$v$流表的最初已用容量。对子流段$s(f,i)$，$p_{f,i}$和$p'_{f,i}$有共同的源和目的交换机，而更新期间源交换机只需替换规则，目的交换机无需更新规则，因此对应流表不作变化。以$z_{f,i}(v)\in\{0,1\}$表示子流段$s(f,i)$的旧路径$p_{f,i}[1:-1]$是否包含交换机v，以$z'_{f,i}(v)\in\{0,1\}$表示子流段$s(f,i)$的新路径$p'_{f,i}[1:-1]$是否包含交换机v。

$$z'_{f,i}(v)=\begin{cases}1,\ p'_{f,i}[1:-1]\text{包含交换机}v\\0,\ \text{其他}\end{cases} \tag{10}$$

则在更新期间，各交换机安装的流规则数量不能超过其最大容量。假设在单个交换机上，单条流对应单个流规则。需满足式(11)约束：

$$\begin{aligned}&w_j(v)+\sum_{\forall f\in F}\sum_{i=1}^{|R_f|}z'_{f,i}(v)x_{(f,i,j)}\leq L_v,\\&j\in\{1,2,\cdots,r\},\forall v\in V\end{aligned} \tag{11}$$

与式(9)类似，当第j轮更新完成后，对交换机v添加和删除规则都将改变其流表已用容量。则第j+1轮各交换机流表初始已用容量可按式(12)得到。

$$\begin{aligned}w_{j+1}(v)=&w_j(v)+\sum_{\forall f\in F}\sum_{i=1}^{|R_f|}z'_{f,i}(v)x_{(f,i,j)}\\&-\sum_{\forall f\in F}\sum_{i=1}^{|R_f|}z_{f,i}(v)x_{(f,i,j)},\\&j\in\{1,2,\cdots,r-1\},\forall v\in V\end{aligned} \tag{12}$$

最后，为了使得流更新尽可能快地完成，该模型以最小化更新轮数为优化目标。

$$\left.\begin{aligned}&\min\sum_{j=1}^{r}\widehat{u}_j\\&\text{s.t.}\quad\text{式(4)}-\text{式(8)},\text{式(10)},\text{式(11)}\end{aligned}\right\} \tag{13}$$

3 流更新算法

由于在链路容量约束下，文献[10]证明了找出最优的更新安排是NP-完全问题，因此本文提出基于贪心思想的启发式算法解决此问题。该算法首先对流分段以增加更新并行性，然后通过多轮迭代找出较快的更新安排并解决出现的死锁问题，最后在子流段更新期间避免包乱序问题。下面详细描述算法过程。表1为流分割算法。

表 1 流分割算法(算法1)

输入：P_f, P'_f
输出：R_f

(1) $R_f \leftarrow \emptyset, d=\emptyset$, s=first(P'_f)
(2) **while** $d \neq$ last(P'_f) **do**
(3) d =common(P'_f, P_f, s)
(4) $p \leftarrow P_f[s,d], p' \leftarrow P'_f[s,d]$
(5) **if** $p \neq p'$ **then**
(6) $R_f \leftarrow R_f \cup (p, p')$
(7) **end if**
(8) $s=d$
(9) **end while**
(10) **return** R_f

算法1 将单个流f分割为多个子流段。first(P'_f)表示新路径的第1个交换机，last(P'_f)表示最后一个交换机。第(3)行common(P'_f, P_f, s)表示从交换机s开始沿着新路径查找P'_f和P_f的下个共同的交换机d。第(4)行分别将以s和d为源和目的的子路径作为子流段的新旧路径。第(5)～(7)行表示若子流段的新旧路径相同，则不必更新，这将进一步加快更新过程。对F中所有流执行算法1后得到子流段集合S。

表2为流更新规划算法。

表 2 流更新规划算法(算法2)

输入：S //子流段集合
输出：U //更新规划

(1) $U \leftarrow \varnothing, \bar{S}=S, j=1$
(2) sort $\bar{S}$ by b_f of each $\forall s(f,i) \in \bar{S}$
(3) **while** $\bar{S} \neq \varnothing$ **do**
(4) $u_j \leftarrow \varnothing$
(5) **for** each $s(f,i)$ **in** $\bar{S}$ **do**
(6) **if** $\forall (u,v) \in p'_{f,i}$, $h_j(u,v)+b_f \le C_{u,v}$ **and** $\forall v \in p'_{f,i}[1:-1]$, $z_j(v)+1 \le L_v$ **then**
(7) $u_j \leftarrow u_j \cup s(f,i)$
(8) $\forall (u,v) \in p'_{f,i}, h_j(u,v) = h_j(u,v) + b_f$
(9) $\forall v \in p'_{f,i}[1:-1]$, $z_j(v) = z_j(v) + 1$
(10) **end if**
(11) **end for**
(12) **if** $u_j = \varnothing$ **and** $\bar{S} \neq \varnothing$ **then**
(13) calculate $\pi_{f,i}$ for each $s(f,i)$ in $\bar{S}$
(14) $u_j \leftarrow$ select the $s(f,i)$ with minimal $\pi_{f,i}$
(15) **end if**
(16) $U \leftarrow U \cup \{u_j\}$, update (u_j)
(17) **for** each $s(f,i)$ **in** u_j **do**
(18) $\forall (u,v) \in p_{f,i}$, $h_j(u,v) = h_j(u,v) - b_f$
19: $\forall v \in p_{f,i}[1:-1]$, $z_j(v) = z_j(v) - 1$
(20) **end for**
(21) remove all elements of u_j from $\bar{S}$
(22) $j = j + 1$
(23) **end while**
(24) **return** U

算法2 表示流更新规划的具体过程。第(2)行首先将算法1得到的集合$\bar{S}$按照流的传输速率从大到小排序。基于贪心的思想，将总能以较少的轮数完成更新。第(5)～(11)行判断集合$\bar{S}$中每一个流子段的更新是否满足链路容量和流表容量的约束，若满足则将该子流段加入集合u_j中，同时调整相关链路的负载量和相关交换机的流表占用量。第(12)～(15)行表示若没有找到可更新的子流段而且未完成所有更新，则表明出现了死锁问题。例如，若流f_1与f_2互换路径，但f_1和f_2的原路径都没有足够的可用带宽容纳对方，则不可能完成更新，这就是更新死锁问题。要解决该问题，必须降低f_1或f_2的传输速率再进行更新。第(13)～(14)行采用文献[12]的方法，找出发生死锁时对流速率影响最小的子流段并更新，具体是首先计算每个子流段的屈服率$\pi_{f,i}$，见式(14)，该值越小表明更新对应子流段造成的丢包越少，然后选择屈服率最小的子流段更新。

$$\pi_{f,i} = 1 - \frac{C_{u,v} - H}{b_f}, H = \max\{h_j(u,v)\}, \forall (u,v) \in p'_{f,i} \tag{14}$$

第(16)行更新集合u_j中的所有子流段。当更新完之后，相关链路可用带宽和相关交换机流表可用空间相应增加，第(17)～(20)行对变量作相应调整。算法2最后返回更新规划U。

因在更新期间会出现数据包的乱序，这将降低网络性能，本文采用基于延时队列的方法来避免包乱序问题。SDN控制器拥有全局的网络视图，因此可控制单个流流经特定交换机时的行为，这为解决包乱序问题提供了便利。问题描述部分已经说明了产生包乱序的根源在于新旧路径存在时延差，而且

只有当流从旧路径切换到新路径后才真正造成了包乱序。因此，可在子流段新路径的第一个交换机添加延时队列来修正时延差。

表3为子流段更新算法。

表 3　子流段更新算法(算法3)

输入：u //子流段集合

(1) **for** each $s(f,i)$ **in** u **do**
(2) d_o← delay of old path
(3) d_n← delay of new path
(4) add rules in reverse order
(5) **if** d_o>d_n then
(6) set delay queue
(7) **end if**
(8) modify rule
(9) **if** d_o>d_n **then**
(10) wait for $(d_o - d_n)$ ms
(11) unset delay queue
(12) **end if**
(13) wait for d_n ms
(14) delete rules in forward order
(15) **end for**

算法3　表示更新单个子流段的具体过程。第(2)～(3)行分别计算子流段新旧路径的总时延，第(4)行对新路径上的交换机以逆序的方式添加流规则，这将避免转发黑洞和转发回环。第(5)行判断新旧路径是否存在时延差，若存在则表明可能造成包乱序问题，然后为子流段的新路径添加延时队列，延时队列中数据包的等待时间为$d_o - d_n$，此时流的数据包依旧沿旧路径转发。第(8)行修改新路径上第1个交换机的流规则，此后流将沿着新路径首先进入延时队列中。第(9)～(12)行表示当设置了延时队列且等待了延时差时间间隔后取消延时队列，此后将不会造成包乱序问题。第(13)～(14)行表明等待旧路径端到端总时延后，旧路径上的数据包已为空，则沿着旧路径正序删除各交换机上的规则。

对算法的时间复杂度进行分析，设流的转发路径长度最大为N，算法1对单条流进行分段，则其时间复杂度为$O(N)$；算法2首先对子流段集合进行排序，设$|S| = M$，最坏情况下排序复杂度为$O(M^2)$，第(5)～(11)行循环遍历所有剩余子流段找出可更新的子流段，时间复杂度为$O(M)$，而while循环最坏情况下循环M次，算法中为了实时调整资源，所需时间复杂度为$O(2MN)$，因此算法2总时间复杂度为$O(2M^2+2MN)=O(M^2+MN)$；算法3对集合中所有子流段更新，循环最多M次，每次统计各子流段新旧路径总时延，则算法3时间复杂度为$O(2MN)=O(MN)$。

4　实验与分析

为了验证算法的可行性和有效性，本文首先在真实的小型网络环境中进行实验，随后在真实的大型网络拓扑中进行仿真实验。实验环境为：Ubuntu16.04 64位操作系统，i7处理器，8 GB内存。每组实验重复5次，结果取平均值。实验代码基于Python实现。

4.1　小型拓扑实验

在该小型网络中，SDN数据面基于Mininet[15]实现，因该平台能够在基于Linux的系统上搭建真实的网络环境，因此在SDN实验评估中被广泛采用。SDN控制器基于Ryu[16]实现，通过OpenFlow协议与数据面交互。将流更新算法作为控制器的子模块独立运行。网络拓扑来自Internet2，如图2所示，共16个交换机和26条链路，设定链路带宽为100 Mbps，链路时延基于两端点间的地理距离和光纤传播速率确定。

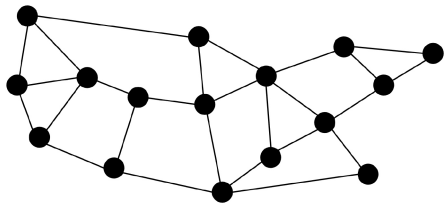


图 2 网络拓扑

随机选取两个不相邻交换机作为流的入口和出口，并为其分配转发路径和基于重力模型[17]设定流的传输速率，介于1～20 Mbps之间。实验比较了流数量分别为150，200，250时的情形，并保证更新前后的网络均处于正常状态。利用iperf工具生成真实的流数据并分析其传输报告统计丢包、乱序等信息。

本实验令所有流的运行周期为20 s，实验从第0 s时运行，在第10 s时发起更新。在流更新期间，使用OpenFlow协议的barrier消息作为每个规则更新完成时的确认消息。因OpenFlow协议对队列的支持不完善，为了避免包乱序，延时队列基于tc-netem工具实现。实验评估指标有总更新时间，丢包率和乱序率。

本实验对比的算法有OneShot方法，Wu等人的方法[12]和本文方法FCFU。其中OneShot更新方法直接将所有流从当前路径更新到目标路径，而Wu等人的方法是基于构造和解析全局依赖图完成更新。

实验首先比较了各算法完成流更新所需的总时

间，结果如图3所示。总更新时间包括计算时间和规则更新时间。从图中可以看出，随着需更新流数量的增多，各算法总更新时间也相应增加。由于OneShot无更新规划过程，其计算时间为0 ms，而相比于Wu等人的方法，FCFU总能以较少的计算时间得到更新方案，这是因为Wu等人的方法需要首先构造全局依赖图，随后通过遍历和解析依赖图得到更新方案，而这将消耗一定的时间。当流数量为150时，FCFU的总更新时间少于Wu等人的，但接近，这是因为流数量较少且网络总负载较低，使得总更新轮数较少。当流数量增多时，FCFU与Wu等人的差距逐渐加大，比如流数量为250时，FCFU和Wu等人平均分别需要185 ms和233 ms完成更新，相比后者，FCFU减少总更新时间达20.6%。这是因为随着网络负载加重，避免拥塞需要更多轮完成，而FCFU基于贪心思想总能得到较少更新轮数的安排。

实验对比了各算法在更新期间的丢包情况，如表4所示。从表中可以看到，无论流的数量为多少，OneShot总会造成较高的丢包率，而FCFU和Wu等人的通过合理安排各流之间的更新顺序，几乎完全避免了可能因为转发黑洞和链路拥塞而造成的丢包，这表明合理的安排流更新是非常必要的。当流数量为250时，Wu等人的方法造成了少量的丢包，这是因为在更新期间出现了死锁，而FCFU未出现。

最后，实验对比了各算法在更新期间包的乱序情况，如图4所示。从图中可以看到，由于OneShot和Wu等人的方法均没有考虑包乱序问题，都造成了较高的乱序率，而在同样的网络环境中，前者低于后者的原因在于OneShot更新期间部分乱序的数据包最终发生了丢失。由于FCFU考虑了包乱序问题，因此几乎完全避免了包乱序的发生。

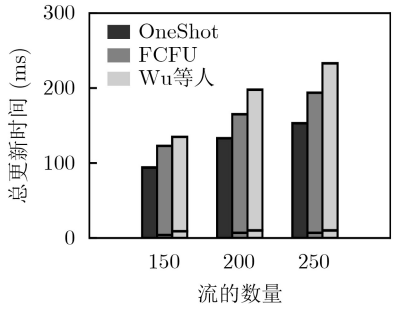


图 3 总更新时间

表 4 丢包率(%)

流数量	OneShot	FCFU	Wu等人
150	0.91	0	0
200	0.85	0	0
250	0.89	0	0.02

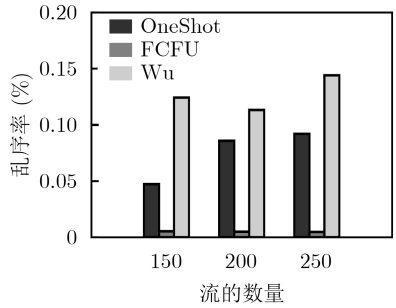


图 4 乱序率

4.2 大型拓扑实验

在该大型网络中，没有真实的流传输，实验目的是为了进一步验证在大型网络拓扑和大规模流更新场景下算法的性能。网络拓扑来自RocketFuel[17]工程中标号为1239的真实运营商网络，共315个交换机和1944条链路，设定链路带宽为100 Mbps。

与上小节实验类似，随机选取两个不相邻交换机作为流的入口和出口，并为其分配转发路径和设定传输速率。实验比较了流数量分别为4000, 5000, 6000, 7000和8000时的情形，同时保证更新前后的网络均处于正常状态。实验将所有流初始状态和目标状态作为输入，将更新安排U作为输出，统计得到U所需的计算时间和更新所需轮数。实验评估指标有计算时间和更新轮数。因为OneShot方法无需计算过程直接更新，因此仅对比Wu等人的方法和本文方法FCFU。

图5表示控制器得到更新安排所需的计算时间。对于Wu等人的方法，随着需更新流数量的增多，其构建的依赖图也相应增大，解析依赖图将耗费更多的时间。而FCFU是直接通过分析比较子流段与资源之间的定量关系来获得更新安排，因此在各种更新情形下均快于Wu等人的方法。表5表示得到的

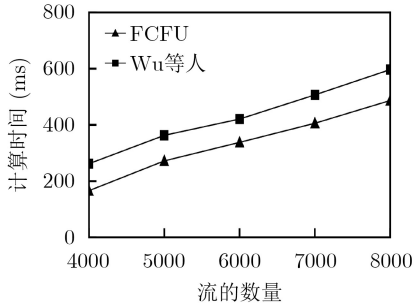


图 5 计算时间

表 5 更新轮数

流数量	4000	5000	6000	7000	8000
FCFU	24	35	35	27	35
Wu等人	63	57	82	64	53

更新安排所需的更新轮数。从表中可得，相比于Wu等人的方法，FCFU所获取的更新安排极大的减少了更新轮数，这得益于算法中贪心的思想，而这将进一步加速后续的规则更新过程。

5　结束语

合理的安排流更新对于提升网络性能具有重要意义。本文提出一种在软件定义网络中快速和一致的流更新策略，在有效地减少总更新时间的同时，还避免了链路拥塞和包乱序的发生。实验结果验证了该策略是可行且有效的。在后续工作中，将继续研究复杂网络环境中策略一致性相关的问题。

参 考 文 献

[1] FOERSTER K T, SCHMID S, and VISSICCHIO S. Survey of consistent software-defined network updates[J]. *IEEE Communications Surveys & Tutorials*, 2019, 21(2): 1435–1461. doi: 10.1109/COMST.2018.2876749.

[2] 胡宇翔, 李子勇, 胡宗魁, 等. 基于流量工程的软件定义网络控制资源优化机制[J]. 电子与信息学报, 2020, 42(3): 661–668. doi: 10.11999/JEIT190276.
HU Yuxiang, LI Ziyong, HU Zongkui, *et al*. Control resource optimization mechanism of SDN based on traffic engineering[J]. *Journal of Electronics & Information Technology*, 2020, 42(3): 661–668. doi: 10.11999/JEIT190276.

[3] 史久根, 许辉亮, 陆立鹏. 软件定义网络中数据中心虚拟机迁移序列问题的研究[J]. 电子与信息学报, 2017, 39(5): 1193–1199. doi: 10.11999/JEIT160792.
SHI Jiugen, XU Huiliang, and LU Lipeng. Research on the migration queue of data center's virtual machine in software defined networks[J]. *Journal of Electronics & Information Technology*, 2017, 39(5): 1193–1199. doi: 10.11999/JEIT160792.

[4] KHALILI R, DESPOTOVIC Z, and HECKER A. Flow setup latency in SDN networks[J]. *IEEE Journal on Selected Areas in Communications*, 2018, 36(12): 2631–2639. doi: 10.1109/JSAC.2018.2871291.

[5] KUŹNIAR M, PEREŠÍNI P, KOSTIĆ D, *et al*. Methodology, measurement and analysis of flow table update characteristics in hardware openflow switches[J]. *Computer Networks*, 2018, 136: 22–36. doi: 10.1016/j.comnet.2018.02.014.

[6] REITBLATT M, FOSTER N, REXFORD J, *et al*. Abstractions for network update[J]. *ACM SIGCOMM Computer Communication Review*, 2012, 42(4): 323–334. doi: 10.1145/2377677.2377748.

[7] HONG Chiyao, KANDULA S, MAHAJAN R, *et al*. Achieving high utilization with software-driven WAN[C]. Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, Hong Kong, China, 2013: 15–26. doi: 10.1145/2486001.2486012.

[8] LIU H H, WU Xin, ZHANG Ming, *et al*. zUpdate: Updating data center networks with zero loss[C]. Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, Hong Kong, China, 2013: 411–422. doi: 10.1145/2486001.2486005.

[9] LUO Shouxi, YU Hongfang, LUO Long, *et al*. Customizable network update planning in SDN[J]. *Journal of Network and Computer Applications*, 2019, 141: 104–115. doi: 10.1016/j.jnca.2019.05.007.

[10] JIN Xin, LIU H H, GANDHI R, *et al*. Dynamic scheduling of network updates[J]. *ACM SIGCOMM Computer Communication Review*, 2014, 44(4): 539–550. doi: 10.1145/2740070.2626307.

[11] WANG Wen, HE Wenbo, SU Jinshu, *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, San Francisco, USA, 2016: 1–9. doi: 10.1109/INFOCOM.2016.7524420.

[12] WU Kunru, LIANG Jiaming, LEE S C, *et al*. Efficient and consistent flow update for software defined networks[J]. *IEEE Journal on Selected Areas in Communications*, 2018, 36(3): 411–421. doi: 10.1109/JSAC.2018.2815458.

[13] FENG Jie, OUYANG Zhipeng, XU Lisong, *et al*. Packet reordering in high-speed networks and its impact on high-speed TCP variants[J]. *Computer Communications*, 2009, 32(1): 62–68. doi: 10.1016/j.comcom.2008.09.022.

[14] ARTHUR C M, GIRMA D, HARLE D, *et al*. The effects of packet reordering in a wireless multimedia environment[C]. Proceedings of the 1st International Symposium on Wireless Communication Systems, Mauritius, 2004: 453–457. doi: 10.1109/ISWCS.2004.1407288.

[15] DE OLIVEIRA R L S, SCHWEITZER C M, SHINODA A A, *et al*. Using mininet for emulation and prototyping software-defined networks[C]. Proceedings of 2014 IEEE Colombian Conference on Communications and Computing, Bogota, Colombia, 2014: 1–6. doi: 10.1109/ColComCon.2014.6860404.

[16] Ryu SDN Framework Community[EB/OL]. http://ryu-sdn.org/.

[17] NGUYEN T D, CHIESA M, and CANINI M. Decentralized consistent updates in SDN[C]. Proceedings of the Symposium on SDN Research, Santa Clara, USA, 2017: 21–33. doi: 10.1145/3050220.3050224.

史久根：男，1963年生，副教授，研究方向为嵌入式系统、计算机网络和软件定义网络.

杨　旭：男，1994年生，硕士生，研究方向为嵌入式系统和软件定义网络.

刘雅丽：女，1996年生，硕士生，研究方向为软件定义网络和网络功能虚拟化.

孙　立：男，1993年生，硕士生，研究方向为软件定义网络和网络功能虚拟化.

责任编辑：陈　倩