柯友运

柯友运

中国电子科技集团公司电子科学研究院

SDN（软件定义网络）技术将网络的控制平面和数据平面分离开来，控制器在控制平面为网络提供全局视图，为路由算法提供新的思路。目前，如 NOX、FloodLight 等均提供了用于完成数据帧转发的模块，采用 Dijkstra 最短路算法。然而，最短路算法容易导致数据流集中到同一条路径进行转发，从而导致网络拥塞。本文对面向 SDN 的路由算法进行研究，实现面向 SDN 的流量调度系统，采用蚁群算法进行路由，并根据 SDN 技术的特征进行相应优化。最后通过仿真实验来验证所提算法的优越性。

DOI：10.3969/j.issn.1001-8972.2014.22.051

面向 SDN 的路由算法研究

概述

2006 年，SDN 诞生于美国 GENI 项目资助的斯坦福大学 Clean Slate 课题，以斯坦福大学 Nick McKeown 教授为首的研究团队提出了 OpenFlow 的概念用于校园网络的试验创新，后续基于 OpenFlow 给网络带来可编程的特性，SDN 的概念应运而生。Clean Slate 项目的最终目的是要重新发明因特网，旨在改变设计已略显不合时宜，且难以进化发展的现有网络基础架构。OpenFlow 能为校园网络研究人员设计其创新网络提供真实的试验平台，目前 OpenFlow 技术的研究进展已经引起学术界和产业界的广泛关注，以开放软件模式的控制平面替代了传统的基于系统嵌入的控制平面，由软件驱动的中央控制结点来自动化控制整个网络，简化了网络的配置模式，增加了网络控制权的开放性，在某种程度上符合未来互联网的发展需求。

传统网络使用的路由算法大多基于距离，跳数或时延，且以最短路算法实现为主。这种路由算法由于其选择的路径结果单一，且没有考虑实时的链路状态参数，可能造成局部链路拥塞，整体链路利用率较低。

网络研究者也一直在对传统路由算法进行改进。研究的方向主要有：

（1）采用多路径路由算法。传统的路由算法一般是寻找一条花费最小的路径，而在实际网络中，往往存在着多条花费相同或近似的路径，多路径路由算法可为数据包提供多条可选的路径。

（2）动态调整路由计算参数。通过链路状态，例如剩余带宽、时延、丢包率等链路状态信息，动态地对路由计算中使用的参数进行改变。这种方法在实现中，也存在着一定的问题，例如网络状态变化的不稳定性和状态采集工作在分布式路由中实施困难。这种方式也可能造成路由不稳定，可能对网络的整体性能产生影响。

综上所述，本文决定采用面向 SDN 的多路径蚁群路由算法，其既运用了多路径路由，也能动态调整路由计算参数，能有效地解决链路拥塞问题，提高整体链路利用率。最后通过仿真实验来验证所提算法的优越性。

POX 控制器解析

POX 是基于 Python 实现的 SDN 控制器平台。POX 遵循 OpenFlow specification 1.0 标准进行了代码实现，采用了模块化架构，且使用了基于事件驱动的机制，并提供控制器与交换机之间的通信接口。

在 POX 控制器中，报文有两种常见的使用场景：一是在控制器端构造报文并发送给交换机，二是经由 ofp_packet_in 消息从交换机中接受报文，而控制器对网络节点的控制时机体现在多种事件的响应机制上。POX 中定义的事件都是 revent.Event 类的子类，还有一种用来发布事件的类都是 revent.EventMixin 类的子类，这个类也通常是组件中用来向 core 中注册的类。

在 POX 的事件系统中，事件处理需要有发布者和订阅者，二者缺一不可。不同的组件会触发不同的事件，具体如表 1 所示。

面向 SDN 的多路径蚁群路由算法

1991 年，DorigoM 在第一届人工生命上提出蚁群算法的基本模型。1992 年，DorigoM 在他的博士论文中提出了蚂蚁系统，并详细阐述了该算法的核心思想。在 2000 年 Gutjahr 首次对蚁群算法的收敛性进行了证明。2002 年，比安奇和她的同事提出了随机问题的最早算法。2005 年，蚁群算法首次被应用在蛋白质折叠问题上。

算法设计步骤

1）参数初始化。设置最大循环次数 NMAX，初始化 M 只蚂蚁。初始化信息素列表，将初值赋给信息素列表的每条链路。初始化可选路径列表，初始化蚂蚁禁忌表。依次将每只蚂蚁置于源节点，并将源节点加入禁忌表。

2）循环次数 N = N + 1。

表 1　事件与相应的触发模块对应表

序号	事件	触发该事件的模块
1	AggregateFlow StatsReceived	of_01.py
2	BarrierIn	of_01.py topology.py
3	ConnectionIn	__init__.py
4	ConnectionUp	of_01.py
5	ConnectionDown	of_01.py
6	ErrorIn	of_01.py
7	FlowTableModification	flow_table.py topology.py
8	FlowRemoved	of_01.py topology.py
9	FeaturesReceived	of_01.py
10	FlowStatsReceived	of_01.py
11	HostEvent	host_tracker.py
12	LinkEvent	discovery.py
13	PortStatus	of_01.py topology.py
14	PortStatsReceived	of_01.py
15	PacketIn	nicira.py of_01.py topology.py
16	QueueStatsReceived	of_01.py
17	RawStatsReply	of_01.py
18	SwitchDescReceived	of_01.py
19	SwitchJoin	topology.py
20	SwitchConnectionUp	topology.py
21	SwitchConnectionDown	topology.py
22	TableStatsReceived	of_01.py

3）蚂蚁数目 k = N+1。

4）蚂蚁个体根据概率公式（3-1）计算的概率选择下一跳链路。

5）每选择一条链路即将该链路加入蚂蚁轨迹中，将蚂蚁移动到下一节点，并将该节点加入禁忌表。

6）若该节点不是目的节点且有下一跳可用链路，则跳转到第四步，继续计算下一跳可用链路列表。

7）若该节点是目的节点，则不再计算下一跳可用链路列表，如果蚂蚁轨迹不在路径列表中，则将蚂蚁轨迹加入路径列表。

8）若 k！ = M，则跳转到第三步。

9）若满足结束条件，即如果循环次数 N = NMAX，则循环结束。否则清空禁忌表，跳转到第二步，并根据公式（3-2）和（3-3）更新每条链路上的信息素。

$$P_{ij}^{k}(t)=\begin{cases}\dfrac{[\tau_{ij}(t)]^{\alpha}*[\eta_{ij}]^{\beta}}{\sum_{k\in next}[\tau_{ik}(t)]^{\alpha}*[\eta_{ik}]^{\beta}}, 若\ j\in next\\ 0, 其\ 他\end{cases} \quad (3-1)$$

公式（3-1）中，α 为信息素因子，β 为链路参数因子，$P_{ij}^{k}(t)$ 代表当前节点为 i，选择节点 j 的概率；k 代表蚂蚁的编号；$\tau_{ij}(t)$ 代表节点 i，j 间的链路在 t 时刻的信息素浓度；η_{ij} 代表节点 i，j 间链路的可见度，其中 $\eta_{ij}=1/\cos t_{ij}$，这说明可见性与链路的花费成反比；next 代表可选下一跳链路的集合。

$$\tau_{ij}(t+n)=(1-p)*\tau_{ij}(t)+\Delta\tau_{ij}(t) \quad (3-2)$$

$$\Delta\tau_{ij}(t)=\sum_{k=1}^{m}\tau_{ij}^{k}(t),\quad \tau_{ij}^{k}(t)=Q/\cos t_{ij} \quad (3-3)$$

式中，p 表示信息素的挥发系数，$\Delta\tau_{ij}^{k}(t)$ 表示本轮循环中的蚂蚁给链路（i，j）带来的信息素增量，Q 为控制参数。

流量调度系统的设计与实现

面向 SDN 的流量调度系统的实现是基于对 POX 控制器的扩展完成的，该扩展分为 3 个模块：全局拓扑学习模块，链路状态评估模块，蚁群算法路由模块。

全局拓扑学习模块

该模块主要进行全局拓扑的学习，故需要监听 LinkEvent，ConnectionUp，ConnectionDown，HostEvent 事件，并相应的启动 discovery，of_01，host_tracker 模块。当控制器捕捉到上述事件触发时，将调用相应的函数进行事件处理，此模块中定义的相应事件处理函数的主要工作是记录全局拓扑信息，方便其他模块调用。

链路状态评估模块

该模块主要进行全局链路状态的评估，评估参数包括剩余带宽，丢包率和跳数。本文通过采取查询交换机端口状态的方法来获取链路的剩余带宽和丢包率。因此，该模块需要监听 PortStatsReceived 事件，并相应的启动 of_01 模块。在有新流到来时，通过调用二次间隔为 1s 的端口状态查询函数对连接到控制器的所有交换机的每一个端口发送请求，然后模块中自定义的端口状态处理函数将处理查询到的状态变量，并计算 1s 内每条链路的带宽使用量以及丢包率并存储以供蚁群算法路由模块使用。

蚁群算法路由模块

该模块首先读取链路状态评估模块所获取的链路状态参数并采用将状态参数相乘的评价函数计算出链路状态指数。然后调用多路径蚁群算法计算出从源节点到目的节点

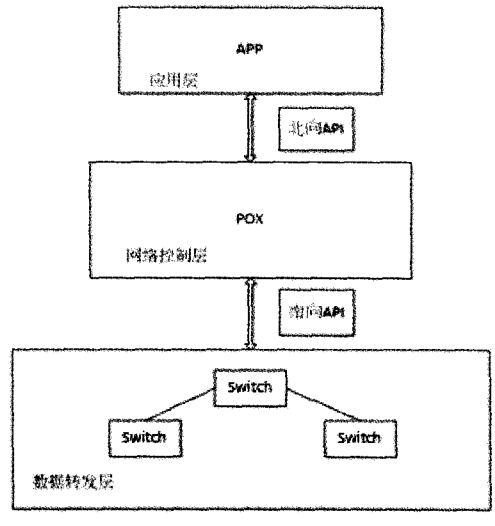

图 1　面向 SDN 的流量调度系统架构图

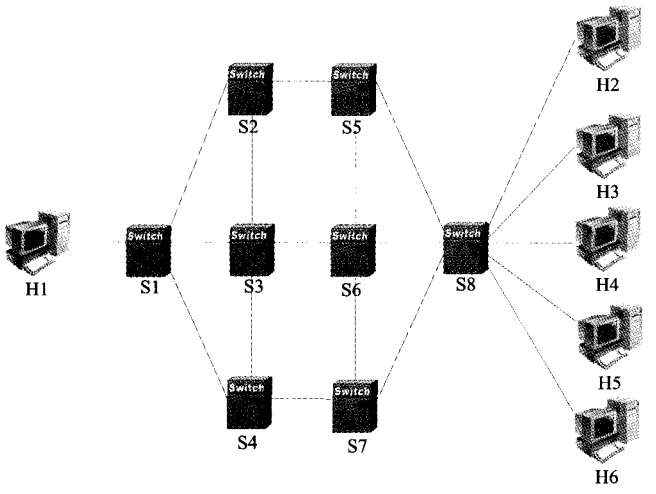

图 2　实验拓扑图一

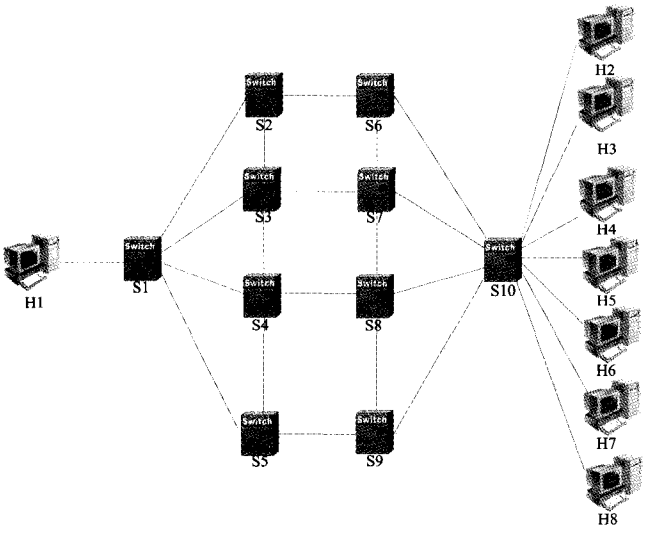

图 3　实验拓扑图二

的可用路径，然后随机选取一条路径作为该流的合适路径，选取时需要将可用路径的跳数考虑在内。多路径蚁群算法步骤见 3.1 节。随后，对所选取路径的沿途交换机下发流表。

系统实现与仿真实验

面向 SDN 的流量调度系统

该系统架构如图。

其中，数据转发层使用 Mininet 仿真平台，并使用 OVS 虚拟交换机来搭建底层网络架构。Mininet 可以方便地创建 OVS 虚拟交换机，而且消耗的系统资源也较少，能自定义拓扑并设置交换机和链路的多项参数以及模拟各类意外事件，如交换机的添加和移除，链路的添加和移除，主机的添加和移除。

仿真实验

实验环境如下：

i3 M330 @2.13GHz CPU，2G 内 存，Ubuntu 12.04 LTS，Python 2.7.3，Vim 7.3，POX 0.2.0(carp)，Mininet 2.0.0，OpenFlow 1.0

本文使用 Mininet 创建如下所示的两个拓扑用以验证面向 SDN 的流量调度系统的有效性。实验采用 iperf 软件进行打流，流速率设为 1M/s，持续时间为 100s，链路带宽设为 2M/s，蚁群算法路由模块里 α 设为 1，β 设为 5，p 设为 0.5，Q 设为 1，最大迭代次数设为 50，每次迭代蚂蚁个体数设为 30。为避免偶然性，故每个拓扑下进行三次实验，并与 ECMP 算法进行对比，实验结果记录如下。

表 2　蚁群算法拓扑一实验结果表

实验序号	发送端	接收端	路径	时延 (ms)	丢包率
1	H1	H2	S1–S2–S5–S8	0.017	0.34%
		H3	S1–S3–S6–S7–S8	0.021	0.41%
		H4	S1–S4–S7–S8	0.020	0.43%
		H5	S1–S2–S5–S6–S8	0.019	0.37%
		H6	S1–S4–S7–S6–S8	0.023	0.39%
2	H1	H2	S1–S4–S3–S6–S8	0.020	0.38%
		H3	S1–S2–S5–S6–S8	0.024	0.40%
		H4	S1–S3–S4–S7–S8	0.019	0.32%
		H5	S1–S3–S2–S5–S8	0.016	0.43%
		H6	S1–S4–S7–S8	0.023	0.41%
3	H1	H2	S1–S3–S6–S5–S8	0.018	0.41%
		H3	S1–S2–S3–S6–S8	0.023	0.39%
		H4	S1–S4–S7–S6–S8	0.020	0.37%
		H5	S1–S3–S4–S7–S8	0.019	0.40%
		H6	S1–S2–S5–S8	0.021	0.42%

表 3　ECMP 算法拓扑一实验结果表

实验序号	发送端	接收端	路径	时延 (ms)	丢包率
1	H1	H2	S1–S4–S3–S6–S5–S8	0.029	2.5%
		H3	S1–S3–S2–S5–S6–S8	15.9	26%
		H4	S1–S3–S4–S7–S6–S8	9.6	18%
		H5	S1–S2–S5–S6–S8	7.9	39%
		H6	S1–S3–S6–S8	0.026	22%
2	H1	H2	S1–S2–S3–S4–S7–S8	0.017	0.067%
		H3	S1–S3–S4–S7–S6–S8	0.79	15%
		H4	S1–S4–S3–S2–S5–S8	0.42	15%
		H5	S1–S3–S2–S5–S6–S7–S8	0.21	9.5%
		H6	S1–S3–S2–S5–S8	0.025	13%
3	H1	H2	S1–S3–S2–S5–S6–S8	0.028	0.078%
		H3	S1–S3–S4–S7–S8	0.40	9.2%
		H4	S1–S2–S3–S4–S7–S8	0.20	16%
		H5	S1–S4–S7–S8	0.029	25%
		H6	S1–S4–S3–S2–S5–S8	0.031	0.067%

实验结果分析

由表 2 和表 3 可知，蚁群算法在拓扑一的三次实验中，每条路径的时延和丢包率都处于正常范围（丢包率始终保

表 5　ECMP 算法拓扑二实验结果表

实验序号	发送端	接收端	路径	时延（ms）	丢包率
1	H1	H2	S1–S5–S9–S10	0.2	3.45%
		H3	S1–S3–S4–S8–S7–S10	6.464	13%
		H4	S1–S4–S8–S10	0.449	6.9%
		H5	S1–S3–S4–S8–S7–S10	4.243	12%
		H6	S1–S2–S6–S7–S3–S4–S5–S9–S8–S10	1.220	9%
		H7	S1–S3–S7–S10	1.216	7.68%
		H8	S1–S4–S5–S9–S10	0.224	3.62%
2	H1	H2	S1–S3–S7–S10	0.027	0.09%
		H3	S1–S5–S9–S8–S10	0.020	0.067%
		H4	S1–S2–S6–S7–S10	0.018	0.078%
		H5	S1–S2–S6–S10	0.019	0.045%
		H6	S1–S4–S8–S9–S10	0.015	0.064%
		H7	S1–S5–S9–S10	0.017	0.062%
		H8	S1–S3–S7–S6–S10	0.018	0.067%
3	H1	H2	S1–S5–S4–S3–S7–S8–S10	0.221	3.3%
		H3	S1–S2–S6–S7–S10	10.383	21%
		H4	S1–S2–S6–S10	193.247	44%
		H5	S1–S5–S4–S3–S2–S6–S10	26.5	37%
		H6	S1–S3–S2–S6–S10	30.591	34%
		H7	S1–S3–S4–S5–S9–S8–S7–S6–S10	22.130	26%
		H8	S1–S2–S3–S4–S8–S7–S6–S10	14.132	29%

表 4　蚁群算法拓扑二实验结果表

实验序号	发送端	接收端	路径	时延（ms）	丢包率
1	H1	H2	S1–S2–S6–S10	0.023	0.62%
		H3	S1–S5–S4–S8–S10	0.028	0.61%
		H4	S1–S4–S3–S7–S10	0.024	0.46%
		H5	S1–S3–S2–S6–S10	0.021	0.61%
		H6	S1–S5–S9–S10	0.027	0.68%
		H7	S1–S3–S7–S8–S10	0.024	0.52%
		H8	S1–S4–S8–S9–S10	0.025	0.46%
2	H1	H2	S1–S4–S8–S9–S10	0.027	0.39%
		H3	S1–S3–S7–S6–S10	0.015	0.48%
		H4	S1–S5–S9–S10	0.026	0.31%
		H5	S1–S2–S6–S7–S10	0.025	0.71%
		H6	S1–S3–S2–S6–S10	0.015	0.61%
		H7	S1–S2–S3–S7–S10	0.021	0.67%
		H8	S1–S4–S8–S10	0.025	0.43%
3	H1	H2	S1–S2–S6–S10	0.028	0.58%
		H3	S1–S4–S3–S7–S10	0.020	0.36%
		H4	S1–S5–S9–S10	0.023	0.47%
		H5	S1–S3–S2–S6–S10	0.026	0.63%
		H6	S1–S3–S7–S10	0.021	0.39%
		H7	S1–S4–S8–S10	0.019	0.45%
		H8	S1–S5–S9–S8–S10	0.027	0.63%

持这一范围是因为算法收敛需要一定时间，在此期间触发 PacketIn 事件的数据包将被丢弃）。而 ECMP 算法在实验一中有四条流选择 S6–S8 链路导致严重拥塞，致使时延和丢包率都大大提高，实验二中有三条流选择 S3–S2–S5 这两条链路，致使时延和丢包率都提升较大，实验三中有三条流选择 S4–S7–S8 这两条链路，致使时延和丢包率都提升较大。由此可见，蚁群算法能有效避免网络拥塞。

由表 4 和 5 可知，蚁群算法在拓扑二的三次实验中，每条路径的时延和丢包率都处于正常范围。而 ECMP 算法在实验一中有三条流选择 S1–S3，S3–S4，S3–S4，S4–S8，S5–S9，S7–S10 链路导致严重拥塞，致使时延和丢包率都大大提高，实验二偶然出现无拥塞情况，实验三中有三条流选择 S1–S2 这条链路，有 4 条流选择 S2–S6 这条链路，有五条流选择 S6–S10 这条链路，导致发送极其严重的拥塞，致使时延和丢包率都出于较高范围。

由此可见，所提算法能有效避免网络拥塞。

结束语

近年来，SDN 已经成为一项热门技术，本文尝试在 SDN 架构中运用蚁群算法更为有效地解决网络拥塞问题，提高整体的链路利用率。也为 SDN 的研究做出小小的贡献。