# 基于零跳传输和流规则并行下发的软件定义网络建链优化方法

赵宝康，李羿锟，杨宇
（国防科技大学计算机学院，湖南 长沙 410073）

**摘　要：**为解决软件定义网络（SDN）控制器端到端链路建立慢的问题，提出了一种基于零跳传输和流规则并行下发的新型双阶段优化方法ZHPro。该方法将端到端链路建立过程分为通过地址解析协议（ARP）解析网络地址和上层协议建链2个阶段。针对ARP地址解析阶段设计了零跳传输机制，通过已知主机信息和精准单播策略，将逐跳广播优化为目标端口直达传输，消除了控制平面与数据平面的冗余交互。针对上层协议建链阶段设计了流规则并行下发机制，通过路径规划、流规则批量构造及去重和多通道并行下发，解决了流规则串行下发所带来的性能瓶颈问题。基于真实网络拓扑数据集开展了大量各类拓扑的实验对比分析，结果表明，与ONOS、ODL、Ryu等控制器相比，ZHPro可将端到端链路建立时间平均降低90%，当网络规模较大时，较ONOS控制器效率可提升12.6倍。同时，ZHPro还具有良好的可扩展性。

**关键词：**软件定义网络；控制器；并行优化；性能优化；链路建立时间
**中图分类号：**TP393
**文献标志码：**A
**DOI：**10.11959/j.issn.1000–436x.2025226

# Optimization approach for link establishment based on zero-hop transmission and parallel flow rules deployment in SDN

ZHAO Baokang, LI Yikun, YANG Yu
School of Computer Science, National University of Defense Technology, Changsha 410073, China

**Abstract:** To address the issue of slow end-to-end link establishment in software defined network (SDN) controllers, a novel dual-phase optimization method named ZHPro, based on zero-hop transmission and parallel flow rule deployment, was proposed. In this method, the end-to-end link establishment process was divided into two phases: network address resolution via the address resolution protocol (ARP) and link setup for upper-layer protocols. For the ARP address resolution phase, a zero-hop transmission mechanism was designed. By leveraging known host information and a precise unicast strategy, hop-by-hop broadcasting was optimized into direct transmission to the destination port, whereby redundant interactions between the control plane and data plane were eliminated. For the upper-layer protocol link setup phase, a parallel flow rule deployment mechanism was designed. Through path planning, batch flow rule construction with deduplication, and multi-channel parallel deployment, the performance bottleneck caused by serial flow rule deployment was resolved. Extensive comparative experiments were conducted on various topologies based on real-world network topology datasets. The results indicated that, compared with controllers such as open network operating system (ONOS), OpenDaylight(ODL), and Ryu, the average end-to-end link establishment time was reduced by 90% with ZHPro. In larger-scale networks, the efficiency was improved by 12.6 times compared to the ONOS controller. Furthermore, ZHPro exhibits robust scalability.

**Keywords:** software-defined network, controller, parallel optimization, performance optimization, link establishment time

**收稿日期：**2025–07–01；**修回日期：**2025–12–08
**通信作者：**赵宝康，bkzhao@nudt.edu.cn
**基金项目：**国家重点研发计划基金资助项目（No.2022YFB2901204）；国家自然科学基金资助项目（No.U22B2005）
**Foundation Items:** The National Key Research and Development Program of China (No.2022YFB2901204), The National Natural Science Foundation of China (No.U22B2005)

## 0 引言

随着算力互联网（CPN, computing power network）[1]的蓬勃发展，网络架构正经历从“以连接为中心”向“以算力为中心”的范式转变。在这一新型网络形态下，计算资源与网络资源的深度融合对算网资源的高效调度提出迫切需求，传统静态僵化的网络架构难以支撑算力服务的按需、实时调度。作为实现算力互联网愿景的关键网络使能技术，基于软件定义网络（SDN, software defined network）[2]理念的软件定义广域网络（SD-WAN, software-defined wide area network）[3]凭借其全局视角和集中控制能力，为跨域网络与计算资源的智能编排与协同优化提供有力支撑。

SD-WAN的动态链路建立等核心功能，其底层依赖于SDN架构的核心组件——SDN控制器。在SDN架构中，控制器作为整个网络的“大脑”，不仅负责网络拓扑的发现和维护，还承担着流量调度、路径计算以及安全策略实施等核心功能[4-5]。随着网络规模的持续扩大和业务复杂度的急剧提升，跨域算力调度、算力任务迁移、实时推理交互等场景将产生海量、高频且动态性极强的网络控制请求，对作为SD-WAN核心枢纽的控制器动态性能提出很高要求。文献[6]表明，控制器性能已成为SDN成功部署的关键决定因素。控制器的性能瓶颈将直接导致算力调度指令延迟、算力任务执行受阻、服务等级协议违约，严重影响算力服务的用户体验和商业价值。特别是在金融、医疗和工业控制以及实时AI推理、交互式云游戏等新兴算力服务等关键领域，网络延迟和不稳定性可能带来严重的经济损失和安全风险。因此，SDN控制器性能优化研究对于保障算力互联网的高效稳定运行，推动其在实际环境中的大规模部署与应用具有至关重要的意义。

端到端链路建立时间是主导SDN的开放网络基金会（ONF, Open Networking Foundation）国际标准化组织提出的一项衡量SDN网络性能的关键指标[7]，它直接影响着用户访问体验和业务响应速度。端到端动态链路建立作为SD-WAN的核心能力之一，直接决定了算力服务的响应速度与用户体验。然而，算力互联网中频繁的虚拟机/容器迁移、服务链动态重组以及算力需求的突发性变化等场景，导致网络拓扑与流量模式持续剧烈波动，使SD-WAN赖以生存的动态链路建立机制面临端到端建立时间陡增的严峻挑战，严重制约了算力资源的实时调度效率和整体服务性能。比如，当调度远端算力执行任务或迁移虚拟机或容器实例时，控制器需要计算路径并下发流表，这一过程的耗时对于实时应用和敏感业务尤为关键[8-9]。而对于大规模多租户云数据中心而言，频繁的虚拟机迁移和弹性伸缩操作会触发大量的链路建立请求。Curtis等[10]在其研究中发现，链路建立时间的优化可以显著提升云平台的资源调度效率和服务质量，为用户提供更加敏捷和可靠的云服务体验。

现有SDN控制器端到端链路建立流程包括地址解析协议（ARP, address resolution protocol）请求处理、路径计算和流规则下发等多个步骤。这些步骤中存在着诸多冗余操作和串行处理环节，导致链路建立时间较长，支持的网络规模小，难以满足需求。以业界主流部署的ONOS控制器为例[11]，在500节点网络规模下，端到端链路建立时间长达12.04 s，且单控制器仅能稳定支持500节点以下规模，难以满足大规模网络快速建链的需求，对SDN控制器性能提升提出迫切挑战。

针对上述难题，本文提出了一种新型双阶段优化方法ZHPro，将端到端链路建立过程分为ARP地址解析和上层协议建链2个阶段，并分阶段针对性设计了零跳传输和流规则并行下发机制，从而大幅提升了控制器端到端建链效能。本文主要贡献如下。

1) 针对现有SDN控制器在ARP地址解析阶段普遍采用ARP广播机制，导致广播消息数量及跳数开销随交换机数量增长急剧攀升的问题，本文提出了一种基于精细化分类的新型ARP零跳传输机制。通过明确区分“未知源目、仅知源、仅知目、已知源目”4种情形实施差异化优化，对“未知源目、仅知源”情形，设计零跳评估函数以实现ARP广播尽早终止，减少不必要广播50%以上；对“仅知目、已知源目”情形，充分利用控制器存储的主机先验信息设计替代ARP广播的单播机制，实现交换机间ARP消息零跳传输，将时间复杂度从$O(n)$减少到$O(1)$。

2) 针对上层协议建链阶段现有SDN控制器采用逐跳串行方式下发流规则导致交互频繁性能低下问题，本文创新设计了流规则并行下发机制，通过

路径规划、流规则批量构造及去重和多通道多线程并行下发，大幅减少控制与数据平面交互次数，将时间复杂度从$O(n)$减少到$O(1)$，实现上层协议建链阶段耗时的数量级性能提升。

3) 基于 Internet Topology Zoo 真实网络拓扑数据集开展了大量各类拓扑的实验对比分析。实验结果表明：与主流控制器 ONOS、ODL、Ryu 相比，ZHPro 可将端到端链路平均建立时间减少 90%；当网络规模较大时，较 ONOS 控制器效率可提升 12.6 倍。消融实验进一步验证了流规则并行下发主导性能提升，零跳传输在抑制广播风暴中作用显著，同时，双机制叠加的组合优化效果超过单一机制。同时，大规模网络可扩展性性能验证结果表明：ZHPro 具有良好的可扩展性，可支持 2 000 台 SDN 交换机网络规模，较其他控制器提高一倍以上；且当网络规模扩展到 2 000 时，ZHPro 的端到端链路建立时间仅是 300 网络规模时 ODL 链路建立时间的一半。

## 1　相关工作

在 SDN 中，端到端链路建立时间直接影响网络性能和用户体验。近年来，研究人员从控制器放置策略优化、流表优化、控制通道优化和智能化调度等多个维度展开研究，提出了多种优化方案。

在控制器放置策略优化方面，文献[12]提出了一种基于流表安装时间的理论模型，该模型量化了控制器放置与链路响应时间的关系，为优化链路建立时间提供了理论支持。文献[13]进一步提出了多目标控制器放置优化算法，该算法结合网络延迟与控制器负载优化链路建立效率。文献[14]在低轨卫星网络环境中研究了动态控制器放置方法，为复杂网络环境的链路时间优化提供了一种解决方案。

在流表优化方面，文献[15]提出了一种基于内容表项树的SDN流表深度聚合方法，通过构建一种 SDN 大规模流表的适应性深度聚合存储架构 ADAFT，缓解了数据平面三态内容可寻址存储器资源紧张的问题。文献[16]提出了一种基于 Q-Learning 的自适应超时分配机制 HQTimer。该方法通过机器学习预测流的持续时间，动态调整流规则的空闲超时值，在数据中心场景下降低了链路建立时间。文献[17]设计了一种基于机器学习的流规则淘汰算法，通过将流分为活跃和非活跃 2 类，实现了智能化的流表管理。

在控制通道优化方面，文献[18]提出了一种动态恢复模块，用于处理带内控制通道故障。该方法通过为控制通道建立可信路径集合，实现了毫秒级的故障恢复。文献[19]基于灵活的网络虚拟化结构提出了快速故障转移机制。该机制通过在更细粒度的时间尺度上计算备份路径，仅在检测到物理网络故障时配置恢复流规则，显著降低了链路中断时间。文献[20]针对多控制器场景提出了基于 K-best 路径的控制通道优化方案。该方案通过在修改后的图中规划控制路径，确保每台交换机都能被多个控制器通过不连续路径服务，提高了系统可靠性。

在智能化调度方面，文献[21]提出了一种基于机器学习的混合 SDN 链路故障下 ACL 策略违规最小化方法 PrePass-Flow，通过机器学习方法预测链路状态，提前重新计算访问控制策略的位置，降低了策略违反带来的时延影响。文献[22]设计了一种基于深度强化学习的自适应路由机制 DRL-R，通过建模端到端时间与网络状态的关系，实现了路由策略的动态优化。文献[23]提出了安全中台服务质量（QoS, quality of service）实时优化算法，通过整合碎片化安全需求与安全基础设施至中台云模型，并结合深度强化学习与云计算提升实时匹配及动态适应能力，实现了满足 QoS 目标的安全中台资源实时调度策略。

此外，文献[24]提出了一种无线 SDN 中的链路可用容量估计方法，有效提高了链路带宽分配的精确性。文献[25]指出端到端带宽估计与链路建立时间密切相关，提出了通过网络数据实时监测优化链路性能的方法。文献[26]中提出了一种针对多厂商 SDN 控制器环境的改进方案，以提升不同厂商控制器之间的协作效率，从而优化链路建立时间。文献[27]提出了一种多目标优化框架，用于在低轨卫星（LEO, low earth orbit satellite）网络中部署 SDN 控制器，进一步提升了复杂网络环境中的链路建立效率。

综上所述，现有研究在端到端链路建立时间优化方面取得了一定进展，但仍存在一些问题：大多数方案关注单一优化维度，缺乏对流表管理、控制通道和架构设计的统筹考虑；机器学习类方法虽然效果显著，但训练开销大且实时性不足；优化方案的可扩展性和部署成本问题仍需进一步研究，这些挑战为本文的研究提供了重要参考。

## 2　问题分析

### 2.1　SDN 端到端链路建立流程

考虑如图 1 所示的简化版示例拓扑，其中 $C_0$ 表示 SDN 控制器，$S_1$ 与 $S_2$ 是 2 个存在链路连接的 OpenFlow[28] 交换机，它们下面分别接了主机 $H_1$ 和 $H_2$。

当 $H_1$ 尝试通过互联网控制报文协议（ICMP）与 $H_2$ 建立连接时，网络中完整的端到端链路建立过程时序图如图 2 所示，可分为 ARP 地址解析和上层协议建链 2 个阶段。其中，$p$ 表示设备端口，端口编号对应图 1 中的简单拓扑。

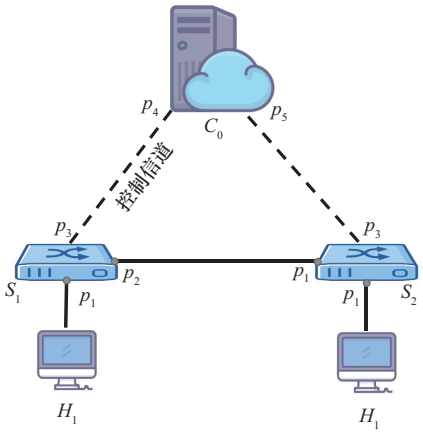


图 1　SDN 端到端链路建立流程讨论示例拓扑

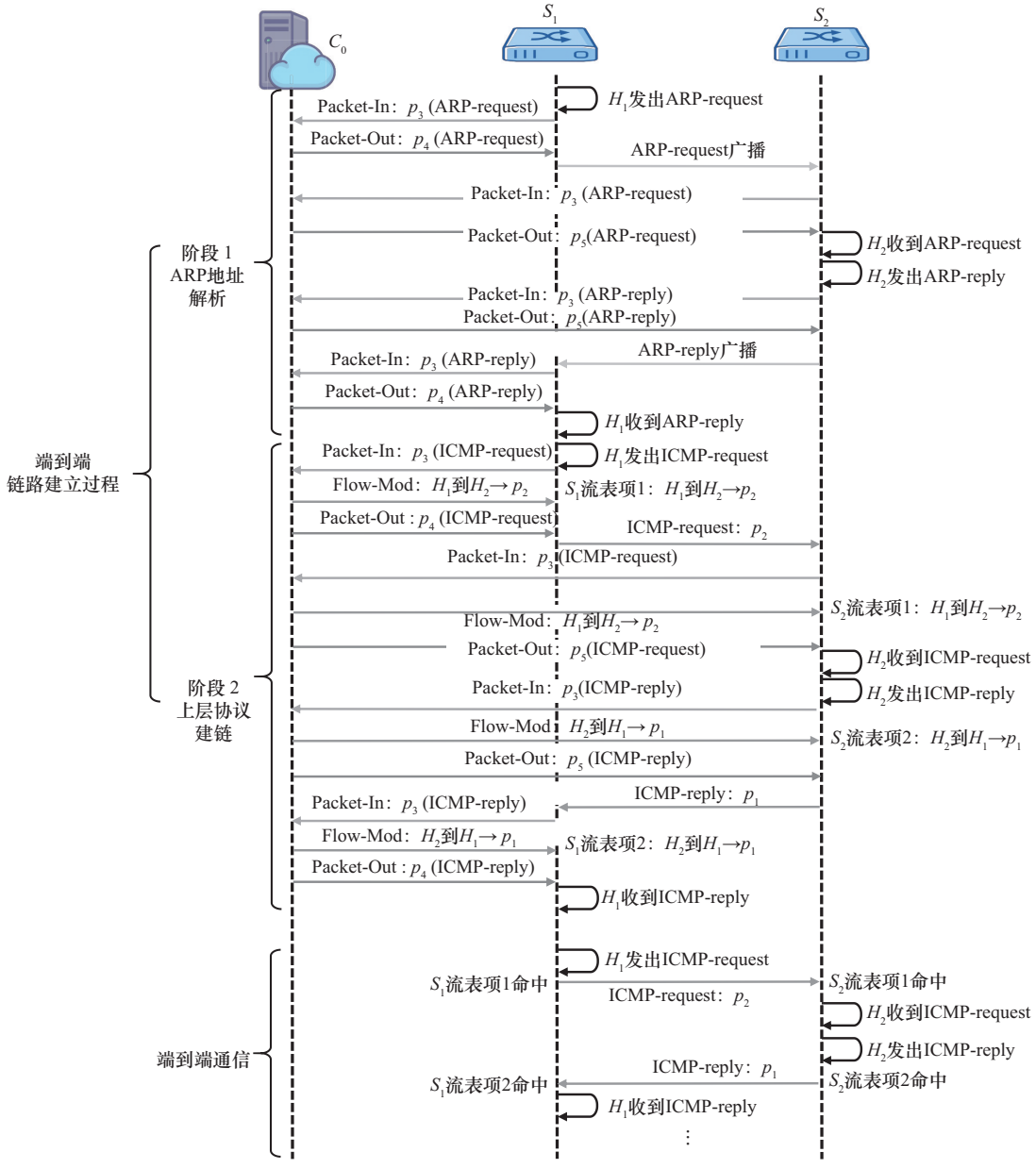


图 2　现有 SDN 端到端链路建立过程时序图与端到端通信过程时序图

阶段 1：ARP 地址解析。当 ping 命令执行后，$H_1$ 首先发出 ARP 请求（ARP-request）询问 $H_2$ 的媒体访问控制地址（MAC, media access control address），报文由 $p_1$ 端口到达 $S_1$。由于未命中流表，$S_1$ 将其封装为 Packet-In 报文后上送至 $C_0$。$C_0$ 首先根据上送的 ARP-request 更新自身维护的主机信息，然后通过 Packet-Out 报文下令 $S_1$ 将 ARP-request 从除 ARP-request 接收的端口（$p_1$）和与 $C_0$ 相连的端口（$p_3$）以外的所有端口处广播出去。同理，当 ARP-request 到达 $S_2$ 后，$C_0$ 下令 $S_2$ 对报文进行广播，使 $H_2$ 收到 ARP-request 报文。

接着，$H_2$ 回复的 ARP-reply 从 $p_2$ 端口到达 $S_2$ 后，被封装为 Packet-In 上送至 $C_0$。$C_0$ 首先根据上送的 ARP-reply 更新自身维护的主机信息，然后通过 Packet-Out 报文下令 $S_2$ 将 ARP-reply 从除 ARP-reply 接收的端口（$p_2$）和与 $C_0$ 相连的端口（$p_3$）以外的所有端口处广播出去。同理，当 ARP-reply 到达 $S_1$ 后，$C_0$ 下令 $S_1$ 对报文进行广播使得 $H_1$ 收到 ARP-reply 报文。至此，$C_0$、$H_1$、$H_2$ 均获得后续通信所需的主机信息，ARP 地址解析阶段任务完成。

阶段 2：上层协议链路建立。$H_1$ 得知 $H_2$ 的 MAC 地址后发出互联网控制报文协议请求（ICMP-request），经 $p_1$ 端口到达 $S_1$ 并被上送至 $C_0$。$C_0$ 基于全局拓扑与自身维护的主机信息进行分析后为 $S_1$ 安装流规则并回复 Packet-Out，接着 ICMP-request 从 $p_1$ 端口到达 $S_2$ 并被上送至 $C_0$。$C_0$ 基于全局拓扑与自身维护的主机信息进行分析后为 $S_2$ 安装流规则并回复 Packet-Out，随后 $H_2$ 收到 ICMP-request 报文并回复互联网控制报文协议响应（ICMP-reply）。同理，ICMP-reply 通过上述相同过程可到达 $H_1$。至此，2 台交换机上的 $H_1$、$H_2$ 双向流表（共 4 条）添加完毕，端到端链路建立完成。

### 2.2　问题模型

本节选择代表网络拓扑中的最长通信路径（网络直径）的线形拓扑进行建模分析，通过研究线形拓扑，建立时间的理论上限，为网络性能提供最坏情况保障。将图 1 的拓扑场景进行推广，考虑主机 $H_1$ 与 $H_2$ 间存在 $n$ 台交换机 $\{S_1,S_2,\cdots,S_n\}$ 的线形拓扑场景。其中交换机 $S_i$ 具有 $p_i$ 个端口。定义以下参数。

1)$T_{i,j}$：交换机 $S_i$ 到 $S_j$ 的报文传输时延。

2)$T_{i,\mathrm{ctrl}}$：交换机 $S_i$ 与控制器 $C_0$ 的通信时延。

3)$\mathrm{PI}_i/\mathrm{PO}_i$：控制器 $C_0$ 处理上送报文和下发报文的时延。

阶段 1：首先 $H_1$ 发出的 ARP-request 逐跳广播，触发每台交换机 $S_i$ 产生 1 次 Packet-In 和 1 次 Packet-Out，且每台交换机在所有端口（除输入端口和控制信道端口外）广播 ARP 报文。此时网络中的报文数量为

$$N_{\mathrm{req}}=\underbrace{2n}_{\text{控制平面报文}}+\underbrace{\sum_{i=1}^{n}(p_i-1)}_{\text{ARP广播}} \tag{1}$$

时延计算式为

$$T_{\mathrm{req}}=\underbrace{\sum_{i=1}^{n}(2T_{i,\mathrm{ctrl}}+\mathrm{PI}_i+\mathrm{PO}_i)}_{\text{控制平面耗时}}+\underbrace{\sum_{i=1}^{n-1}T_{i,i+1}}_{\text{数据平面耗时}} \tag{2}$$

接着，$H_2$ 的 ARP-reply 广播触发每台交换机 $S_i$ 产生 1 次 Packet-In、1 次 Packet-Out。网络中的报文数量 $N_{\mathrm{ans}}=N_{\mathrm{req}}$，时延也与 ARP-request 过程相同，即 $T_{\mathrm{ans}}=T_{\mathrm{req}}$。因此，阶段 1 的总时延为 $T_{\mathrm{stage}_1}=T_{\mathrm{req}}+T_{\mathrm{ans}}=2T_{\mathrm{req}}$，时间复杂度为 $O(n)$，总报文数为 $N_{\mathrm{stage}_1}=N_{\mathrm{req}}+N_{\mathrm{ans}}=2N_{\mathrm{req}}$。

阶段 2：$H_1$ 的 ICMP-request 触发正向流表下发，$H_2$ 的 ICMP-reply 触发逆向流表下发。该阶段每台交换机产生 Packet-In、Packet-Out 和 Flow-Mod 各 2 次，故网络中的报文数量为

$$N_{\mathrm{stage}_2}=\underbrace{6n}_{\text{控制平面报文}}+\underbrace{2n}_{\text{ICMP单播}}=8n \tag{3}$$

时延为 $T_{\mathrm{stage}_2}=T_{\mathrm{stage}_1}$。因此，端到端链路建立过程中产生的总报文数为

$$N_E=\sum_{k=1}^{2}N_{\mathrm{stage}_k}=12n+2\sum_{i=1}^{n}(p_i-1) \tag{4}$$

链路建立时间 $T_E$ 可表示为

$$T_E=\sum_{k=1}^{2}T_{\mathrm{stage}_k}=4\left[\sum_{i=1}^{n}(2T_{i,\mathrm{ctrl}}+\mathrm{PI}_i+\mathrm{PO}_i)+\sum_{i=1}^{n-1}T_{i,i+1}\right] \tag{5}$$

时间复杂度为 $O(n)$。

通过上述分析可知，现有 SDN 端到端链路建立过程主要存在 2 个方面问题。

1) ARP 广播导致的低效处理。随着跨交换机数量的增多，ARP 广播消息在网络中的数量急剧增

加，不仅影响自身的ARP地址解析时延，也影响网络中其余无关消息的正常传输时延，甚至产生广播风暴。

2) 数据与控制平面频繁交互。每台交换机在2个阶段都需要经历Packet-In上送、控制器处理和Packet-Out/Flow-Mod下发的完整交互流程，不仅加重了控制链路负载，也降低了网络响应速度。这种统一处理机制缺乏对已知主机信息的有效利用，未考虑不同场景下的优化可能，从而影响了网络整体通信效率。

## 3　方法

### 3.1　框架设计

基于上文对问题的建模描述，本节提出一种基于零跳传输和流规则并行下发的新型双阶段优化方法ZHPro。其框架如图3所示，主要分为3个层次：决策层、控制层和设备层，各层之间通过数据流和控制流相互协作，以实现高效的链路建立。

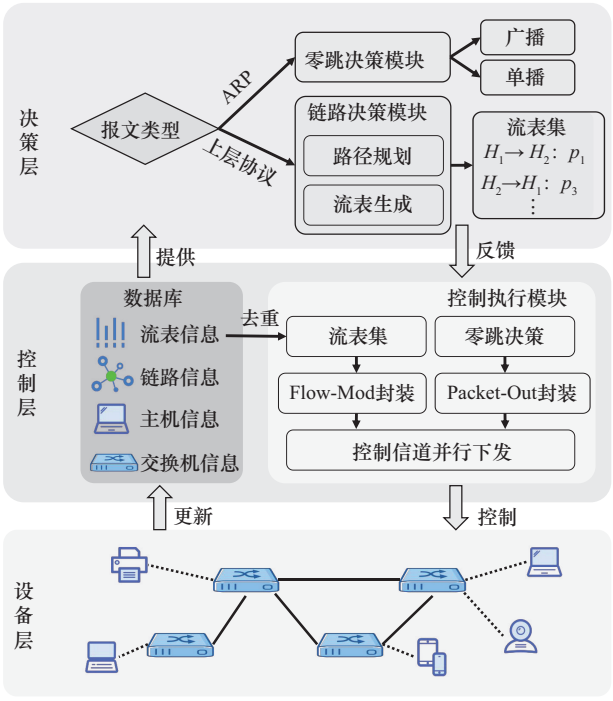


图3　ZHPro框架

设备层由网络中的物理设备组成，如交换机和路由器。这些设备接收来自控制层的指令，执行数据包的转发操作。

控制层作为决策层和设备层之间的桥梁，主要负责数据的处理、存储和转发，以及控制报文的下发。该层包括数据库和控制执行模块2个关键组件。前者用于存储流表数据、主机信息和拓扑关系，为决策层提供数据支持；后者用于接收来自决策层的流集，通过数据库去重后，封装为Flow-Mod消息下发，用于指导设备层的流量转发。同时处理零跳传输决策，并封装Packet-Out消息下发。

决策层是整个框架的核心，负责根据网络状态和应用需求做出最优的路由决策。该层包含2个主要模块。零跳决策模块通过控制层的数据库信息，决定是否需要进行ARP报文的零跳传输；链路决策模块负责规划路径以确定最佳的报文传输通路，将生成的路径信息转换为待配置的流表集，并作为控制序列发送到控制层。

在SDN中，控制平面负责网络的决策与控制，数据平面负责数据转发。因此，ZHPro中所涉及的分析、评估、处理、决策等行为均由SDN控制器负责实现。

### 3.2　零跳传输机制

零跳传输机制的核心思想是实现ARP消息在数据平面交换机间的零跳传输，基本原则是通过ARP单播代替ARP广播，尽量避免广播风暴。

具体而言，零跳传输机制充分利用控制器中的已知主机信息，在目的主机明确时，终止后续的ARP广播，通过直接将ARP消息发送至目的主机所在交换机的方式实现ARP消息在剩余链路交换机间的零跳传输。该机制将ARP地址解析阶段的工作情形分为4种：1)SDN控制器未知源和目的主机信息；2)SDN控制器仅知源主机信息；3)SDN控制器仅知目的主机信息；4)SDN控制器已知源和目的主机信息。针对情形1)和情形2)，通过机制中的零跳评估函数实现网络中的“ARP广播尽早终止”，尽量减少ARP广播的发生频率；针对情形3)和情形4)，该机制充分利用控制器中的已知目的主机信息来避免ARP广播的发生，实现完全的零跳传输。

零跳传输机制包含3个核心组件，如图4所示。首先，控制器通过主机发现组件，动态维护网络中所有主机与交换机的连接关系；其次，在收到ARP报文时快速查询目标主机的位置，并通过零跳评估组件进行决策；最后，决策执行组件切换至目标交换机信道进行ARP单播。以下是对各组件的详细描述。

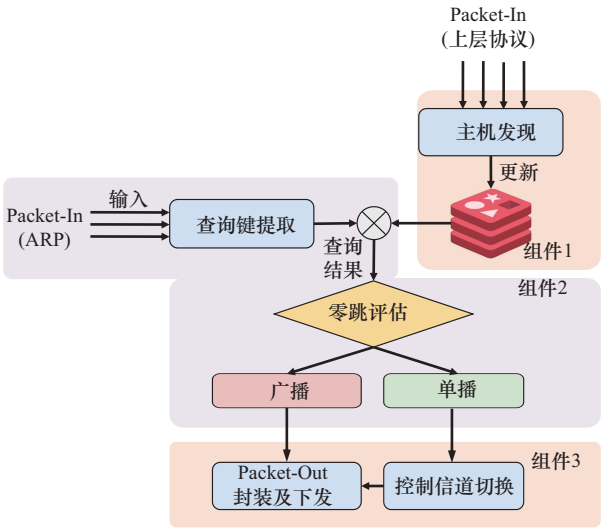


图 4　ARP 零跳传输机制工作流

首先是主机发现组件，其通过事件监听（被动）和周期探测（主动）2 种方式协作来发现和管理网络中的主机。基于事件监听的主机发现是指网络建立初期，控制器默认网络中不存在主机，拓扑视图中仅存在已连接的交换机节点，当主机实际产生网络流量时，交换机对主机报文进行 Packet-In 封装并上送至控制器。控制器通过对 Packet-In 消息头解析，可以得知报文来自哪台交换机，进一步对主机报文解析，可以得知主机的 MAC 与网络层地址（IP）。由此，控制器确定主机与交换机的位置关系，具体设计如算法 1 所示。

**算法 1**　基于事件监听的主机发现算法

**输入**　上送报文 pkt，数据库接口 DB

1) $S_{src}$ = pkt.switch

2) $P_{host}$ = PacketParse(pkt.body)

3) hostMAC = $P_{host}$.srcMAC

4) hostIP = null

5) if HasARP($P_{host}$)

6)　hostIP = $P_{host}$.ARP.srcIP

7) end if

8) if HasIP($P_{host}$)

9)　hostIP = $P_{host}$.IP.srcIP

10) end if

11) if hostIP and DB. QueryHost(hostMAC, hostIP) == null then

12)　$T_{now}$ = GetTime()

13)　DB.InsertHost((hostMAC, hostIP), $S_{src}$, $T_{now}$)

14) end if

当新的 Packet-In 报文到来时，算法 1 对报文体（即主机产生的报文）进行解析。首先从以太网帧中取主机源 MAC 地址，如果报文中包含 ARP，则取用 ARP 报文的源 IP 地址。如果是使用 IP 网络传输的上层协议报文，则直接从 IP 报文头中取 IP 地址。接着通过提取的一对地址在数据库中查询主机，若不存在则将主机与交换机的位置映射信息保存到数据库。

基于周期探测的主机信息维护是指控制器主动向网络中的子网广播 ARP 请求。当目标主机响应时，其连接的交换机会将响应报文通过 Packet-In 消息封装后上报控制器，实现主机信息的定期更新。

然而，频繁地全网主机探测会同时增加系统的报文负载和控制平面的性能消耗。但如果不进行信息更新，又会导致数据库中陈旧条目累积。因此在每次更新时，选择基于数据库中已有的主机信息集合 $H_{info}$ 进行最小化探测，该部分如算法 2 所示。

**算法 2**　基于周期探测的主机信息维护算法

**输入**　组件上下文 ctx，数据库接口 DB

1)while ctx.running

2)　$H_{info}$ = DB.QueryAllHosts()

3)　$T_{now}$ = GetTime()

4)　for each $H$ in $H_{info}$

5)　if $T_{now}$ − $H$.timestamp > ctx.timeout

6)　DB.DeleteHost($H$)

7)　ctx.ARPRequest($H$.switch, $H$.IP)

8)　end if

9)　end for

10)　Sleep(ctx.interval)

11) end while

组件启用时，主机信息维护功能作为一个独立线程持续运行。其首先取来数据库中已有的主机信息集合，遍历其中的每一个条目，判断信息是否过期。如果过期，则将该主机信息从集合中删除，再主动构造一个 ARP 请求报文来检测该主机是否仍然存活。

其次是零跳评估组件，为了更好地描述零跳评估的含义，本节在式(2)中引入系数 $\lambda_i \in \{0,1\}$ 表示交换机 $S_i$ 与控制器的报文交互是否必须。只有当 $S_i$ 与控制器交互后，控制器下发 Packet-Out，才有 $S_i$ 与 $S_{i+1}$ 的报文传输。因此补充后为

$$T_{\text{req}}^{\text{opt}}=\sum_{i=1}^{n}\lambda_i\cdot\left(2T_{i,\text{ctrl}}+\text{PI}_i+\text{PO}_i\right)+\sum_{i=1}^{n-1}\lambda_i\cdot T_{i,i+1}\tag{6}$$

式(2)相当于式(6)中$\lambda_i$均取1的最坏情况，现有SDN端到端链路建立过程因其盲目的ARP广播策略，导致$T_{\text{req}}$总是取到理论最大值。如果基于主机发现提供的信息进行$\lambda_i$值的动态评估，就可以在多数场景下尽可能减小$T_{\text{req}}$。评估函数设计为

$$\lambda_i=f(H_{\text{dst}})=\begin{cases}0, & H_{\text{dst}}\in H_{\text{info}}\text{或}\lambda_{i-1}=0\\ 1, & \text{其他}\end{cases}\tag{7}$$

其中，$H_{\text{dst}}$表示ARP请求报文的目的主机，$H_{\text{info}}$表示主机发现组件维护的主机信息集合。当交换机$S_m(m<n)$的评估值为0时，意味着$S_m$将不会对ARP报文进行传播，进而后续交换机$\{S_{m+1},S_{m+2},\cdots,S_n\}$将不再产生报文，且不对$T_{\text{stage}_1}$产生贡献。基于ARP地址解析阶段的工作情形1)和2)，图5设计了2个案例用于解释零跳评估函数的“ARP广播尽早终止”机制。其中，单实线箭头和双实线箭头分别表示案例一和案例二的报文传输方向，单/双虚线箭头表示“ARP广播尽早终止”机制终止的报文传输。

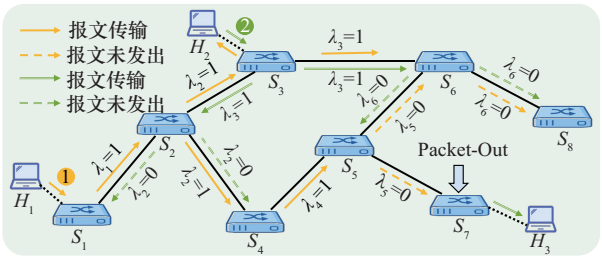


图5　目的主机未知情形下，零跳评估函数的“ARP广播尽早终止”机制示意图

案例一：$H_1$向$H_2$发起ARP-request。假设此时$H_2$从未产生任何流量，则控制器中不存在$H_2$主机信息。零跳评估函数指导流量逐跳广播，在$S_2$处分为$P_1=\langle S_2,S_3,H_2\rangle$、$P_2=\langle S_2,S_3,S_6\rangle$、$P_3=\langle S_2,S_4,S_5\rangle$3条传播路径，$P_2$和$P_3$又可继续分为多条路径在整个网络中进行传播。ARP-request通过$P_1$到达$H_2$，$H_2$发出的ARP-reply经$S_3$上送控制器后，控制器成功发现$H_2$主机信息。假设此后路径$P_2$和$P_3$上的ARP-request分别到达$S_6$和$S_5$并被上送至控制器，此时控制器已有目的主机$H_2$的主机信息。零跳评估函数分别计算得到$\lambda_6=0$和$\lambda_5=0$，因此$P_2$、$P_3$路径上的ARP-request分别在$S_6$和$S_5$处终止传播，并由控制器直接发送携带ARP-request的Packet-out至$S_3$。

案例二：$H_2$向$H_3$发起ARP-request。假设ARP-request在经过$S_3$广播后，$H_3$产生其他流量被控制器发现。此时，ARP-request到达$S_2$和$S_6$并上送控制器，经过零跳评估函数计算后，$\lambda_2$与$\lambda_6$均为0，故ARP-request分别在$S_2$和$S_6$处终止传播，并由控制器直接发送携带ARP-request的Packet-out至$S_7$。

最后是决策执行组件，其负责将零跳评估组件的决策结果实际应用到网络中。具体而言，决策执行组件根据零跳评估的结果，决定是否需要在特定交换机上执行ARP报文的零跳传输。如果评估结果为$\lambda=0$，则组件将直接向$H_{\text{dst}}$所属的交换机下发Packet-Out消息并携带ARP报文，通过目标交换机的指定端口进行单播，从而避免不必要的广播和控制平面交互。否则，组件将向上报Packet-In的交换机下发广播ARP报文的控制消息。

在图5的案例二中，$\lambda_2=\lambda_6=0$意味着决策执行组件将ARP请求报文封装为Packet-Out消息，并通过控制通道下发至目标交换机$S_7$，经过$S_7$的单播$H_3$就能直接收到报文。

综上所述，算法3展示了零跳传输的核心流程。该算法通过主机发现组件提供的主机信息，动态决定ARP报文的传输方式。

**算法3**　零跳传输算法

**输入**　上送报文pkt，数据库接口DB

1)$S_{\text{src}}$ = pkt.switch
2)$P_{\text{host}}$ = PacketParse(pkt.body)
3) if IsARP($P_{\text{host}}$)
4)　arp = $P_{\text{host}}$.ARP
5)　$H_{\text{dst}}$ = DB.QueryHost(arp.dstMAC)
7)　if $H_{\text{dst}}$ == null
8)　　$H_{\text{dst}}$ = DB.QueryHost(arp.dstIP)
9)　end if
10)　if $H_{\text{dst}}$ == null
11)　　port = OFPP_FLOOD
12)　　$P_{\text{out}}$ = PacketOut($S_{\text{src}}$, port, $P_{\text{host}}$)
13)　else
14)　　port = $H_{\text{dst}}$.port
15)　　$S_{\text{dst}}$ = $H_{\text{dst}}$.switch

16)　$P_{out}$ = PacketOut($S_{dst}$, port, $P_{host}$)

17)　end if

18)　SendPacket($P_{out}$)

19) end if

首先，算法接收来自交换机的Packet-In报文，并解析其中的主机报文。如果报文类型为ARP请求，则进一步解析ARP报文内容，并尝试通过目标MAC或IP地址查询目标主机信息。如果目标主机信息不存在于数据库中，则按照现有方式广播ARP请求。否则，算法将直接向目标主机所属的交换机下发Packet-Out消息，并通过指定端口进行单播。最后，决策执行组件将生成的Packet-Out消息发送至交换机。

通过该算法，能够在大多数情况下避免ARP请求的逐跳广播，从而显著减少网络中的冗余报文和控制平面交互次数。接下来，从理论角度分析该机制的性能优势。优化后的$T_{req}^{opt}$时延模型可以表示为式(6)，报文数量模型可以表示为

$$N_{req}^{opt} = 2\sum_{i=1}^{n}\lambda_i + \sum_{i=1}^{n}\lambda_i \cdot (p_i - 1) = \sum_{i=1}^{n}\lambda_i \cdot (p_i + 1) \quad (8)$$

考虑$H_{src}$和$H_{dst}$2台主机建立链路，零跳传输机制在ARP地址解析阶段的工作情形1)和2)下，$T_{req}^{opt} = T_{req}$，与现有方法的时间复杂度一致，均为$O(n)$；对于$T_{ans}$而言，由于此时控制器已具备主机$H_{src}$信息，使得零跳传输机制的时间复杂度为$O(1)$。因此，该机制的ARP地址解析阶段总的时间复杂度为$O(n)$。在情形1)和2)下，尽管与现有方式的时间复杂度相比仍保持在同一数量级，但减少不必要广播50%以上，实现了同一数量级内的性能提升。

考虑$H_{src}$和$H_{dst}$2台主机建立链路，在ARP地址解析阶段的工作情形3)和4)下，由于控制器已具备主机$H_{src}$信息，因此零跳传输机制的ARP地址解析阶段总的时间复杂度为$O(1)$。该情形下，与现有方式的时间复杂度$O(n)$相比，零跳传输机制实现了ARP传输耗时的数量级性能提升。

### 3.3　流规则并行下发机制

流规则并行下发机制的核心思想是减少数据与控制平面控制报文的频繁交互，基本原则是充分利用控制器的路径规划信息，通过多个线程构建多下发通道实现流规则的并行下发。

流规则并行下发机制工作流如图6所示。当携带上层协议报文（以下以ICMP为例）的Packet-In消息到达控制器时，决策层首先根据控制层提供的拓扑视图确定通信的双端，并为其规划一条可达路径。接着，决策层为这条路径上的每一台交换机节点生成一对双向流规则，形成待下发的流规则集。再根据控制层流表管理模块提供的已有流表信息，对流规则集进行去重。最后控制层将流规则集封装为一批Flow-Mod消息，通过多线程将这些消息并行下发。以下是对各流程的详细描述。

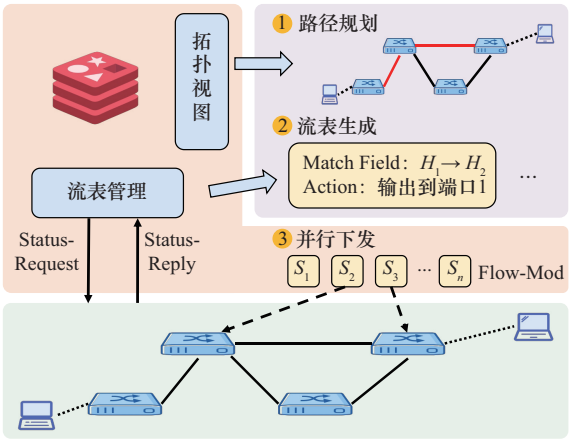


图6　流规则并行下发机制工作流

第一步是路径规划，其目标是为通信双端确定最佳转发路径。当携带ICMP请求的Packet-In报文到达控制器时，链路决策模块首先提取报文的五元组信息（源/目的IP、源/目的MAC、协议类型），并结合数据库中的拓扑视图执行路径搜索。

本机制采用改进的迪杰斯特拉（Dijkstra）算法进行路径规划。算法以源主机连接的交换机$S_{src}$为起点，目标主机连接的交换机$S_{dst}$为终点，在拓扑图中搜索满足时延约束的最短路径。与原算法不同的是，这里引入流表预存在率作为权重因子。

$$w_{i,j} = \alpha \cdot T_{i,j} + \beta \cdot (1 - r_{i,j}) \quad (9)$$

其中，$r_{i,j}$表示交换机$S_i$到$S_j$链路的流表预存在率，$\alpha$、$\beta$为可调参数。该设计通过动态感知网络状态，优先选择高流表命中率的路径，减少临时流规则安装带来的额外开销。

在路径规划算法中，流表预存率的准确性依赖于对网络中已安装流表信息的实时感知，其数据采集通过控制器的主动探测机制实现。具体而言，控制层周期性地向所有OpenFlow交换机发送Stats-Request消息。交换机收到请求后，将当前流规则

信息封装为Stats-Reply消息返回，消息体包含流规则数量、匹配条件、生存时间等元数据。控制层解析响应消息后，将流规则信息按<交换机ID，匹配域,动作集>三元组格式存入数据库，并记录最新时间戳。在此基础上，流表预存率$r_{i,j}$的计算可形式化表示为

$$r_{i,j}=\frac{|F_{\text{installed}}\cap F_{\text{required}}|}{|F_{\text{required}}|} \tag{10}$$

其中，$F_{\text{required}}$表示$S_i$到$S_j$链路所需的标准流表集合，$F_{\text{installed}}$表示当前交换机上已安装的流规则。

流规则并行下发机制工作流如图7所示，图中的表格表示各交换机上已安装的流规则。当主机$H_1$尝试与$H_2$建立通信链路时，可以选择$P_1=\langle S_1,S_3,S_4\rangle$和$P_2=\langle S_1,S_2,S_4\rangle$两条路径。对于$P_1$来说，由于单台交换机需要双向规则来保证双端正常通信，因此从$S_1$到$S_2$一共需要建立4条流规则，目前已有3条（$S_1$第一条及$S_2$后两条），故$r_{1,2}$计算为0.75。对于$P_2$而言，目前$S_1$和$S_3$上只安装了1条符合要求的流规则（$S_3$第二条），$S_1$的第一条规则输出并不指向$S_3$，所以不计入已安装条目内，故$r_{1,3}$计算为0.25。若两条候选路径的物理时延相近，算法优先选择流表预存率更高的路径$P_2$，从而减少需要新增的流规则数量。同理，当$H_1$与$H_3$通信时，算法会选择$P_1$路径来建立链路。

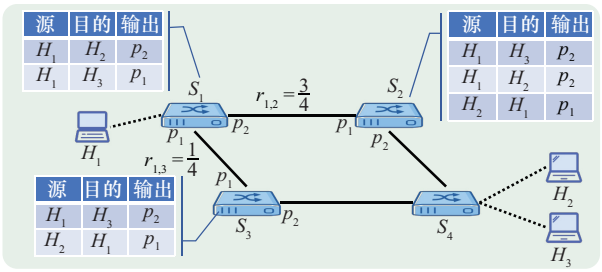


图 7 流规则并行下发机制工作流

第二步是流表生成，在完成主机$H_s$到$H_d$的路径规划后，需要为路径上的每台交换机生成双向流规则。给定规划路径$\langle S_1,S_2,\cdots,S_n\rangle$，流表生成模块将为每台中间交换机$S_i(1\leqslant i\leqslant n)$创建2个方向的流规则。

1) 正向：匹配源$H_s$到目的$H_d$的流量，指定输出端口为$S_i$到$S_{i+1}$的端口。

2) 反向：匹配源$H_d$到目的$H_s$的流量，指定输出端口为$S_i$到$S_{i-1}$的端口。

流规则的匹配字段设计采用四元组（双端MAC及IP地址）精确匹配策略，为减少重复流表的生成，本环节会查询流表信息数据库中已存在的流规则，过滤掉匹配字段完全相同的条目，仅生成必要的新规则。

在完成路径规划与流表生成后，如何高效地将流规则集部署至网络设备成为时延优化的关键环节。现有串行模式下控制器按交换机顺序逐个下发Flow-Mod消息，其总时延可表示为$\sum_{i=1}^{n}T_{i,\text{ctrl}}$。当网络直径较大时，这种线性增长特性将严重制约端到端链路建立效率。为此，本机制引入第三步多通道并行下发策略。

控制器将待下发的流规则集$F_{\text{total}}$按目标交换机进行分组，生成交换机粒度的流表子集$\{F_1,F_2,\cdots,F_m\}(m\leqslant n)$。每个子集$F_i$包含目标交换机$S_i$需要安装的所有流规则。同时，控制器维护多个独立的下发通道，每个通道对应一个线程和TCP连接。各通道线程独立执行流规则下发操作后，向对应交换机发送Barrier-Request消息。交换机在完成当前通道所有Flow-Mod消息处理后立即返回Barrier-Reply。控制器主线程通过收集所有通道的Barrier响应，来判断流规则安装是否成功。

算法4描述了流规则并行下发的核心逻辑。当携带上层协议报文的Packet-In消息到达时，算法首先提取Packet-In报文中的通信双端信息，随后调用路径规划算法确定最优路径。在流表生成阶段，算法为路径上的每台交换机构建双向流规则，并通过数据库查询去除重复条目。最终通过多通道机制并行下发流规则集，每个通道采用批量发送与Barrier同步机制确保流规则安装完成。

**算法4** 流规则并行下发算法

**输入** Packet-In报文pkt，数据库接口DB

1) $H_s$, $H_d$ = ExtractHosts(pkt)

2) $G$ = DB.GetTopology()

3) path = PathPlanning($G$, $H_s$, $H_d$)

4) $F_{\text{total}}$ = null

5) for each $S_i$ in path

6) fwd = BuildFlow($S_i$, $H_s$, $H_d$)

7) rev = BuildFlow($S_i$, $H_d$, $H_s$)

8) $F_{\text{total}}=F_{\text{total}}\cup$ {fwd, rev}

9) end for

10) $F_{new}$ = FilterExistingFlows($F_{total}$, DB)

11) $C$ = CreateChannels($F_{new}$)

12) for each $c$ in $C$

13)　SendBatchFlowMods($c$.switch, $c$.flows)

14)　SendBarrierRequest($c$.switch)

15) end for

16) WaitAllBarrierReply($C$)

与现有串行模式相比，本算法通过并行化改造实现了 $T_{stage_2}$ 的阶跃式优化。考虑包含 $n$ 台交换机的通信路径，参考式(2)，现有串行模式的总时延为

$$T_{stage_2} = 2\left[\sum_{i=1}^{n}(2T_{i,ctrl} + PI_i + PO_i) + \sum_{i=1}^{n-1} T_{i,i+1}\right] \quad (11)$$

而并行模式的总时延则由最慢通道决定

$$T_{stage_2}^{opt} = \max_{1 \leqslant j \leqslant k}\left[\sum_{S_i \in C_j}(T_{i,ctrl} + PO_i)\right] \quad (12)$$

其中，$k$表示并行通道数量($1 \leqslant k \leqslant n$)，$C_j$表示第$j$个通道分配的交换机集合。当采用全并行模式($k = n$)且控制器处理能力无约束时，时间复杂度从$O(n)$降为$O(1)$，实现与网络规模无关的恒定安装时延。

在空间复杂度方面，算法需要维护$m$个主机的信息与$|E|$条链路的流表预存率，存储开销为$O(m + |E|)$，与网络规模保持线性关系。该设计在获得显著时间优化的同时，未引入额外空间复杂度阶跃，具有良好的可扩展性。

采用上述框架与优化机制设计后，ZHPro的端到端链路建立过程时序图如图8所示。其中，$p$表

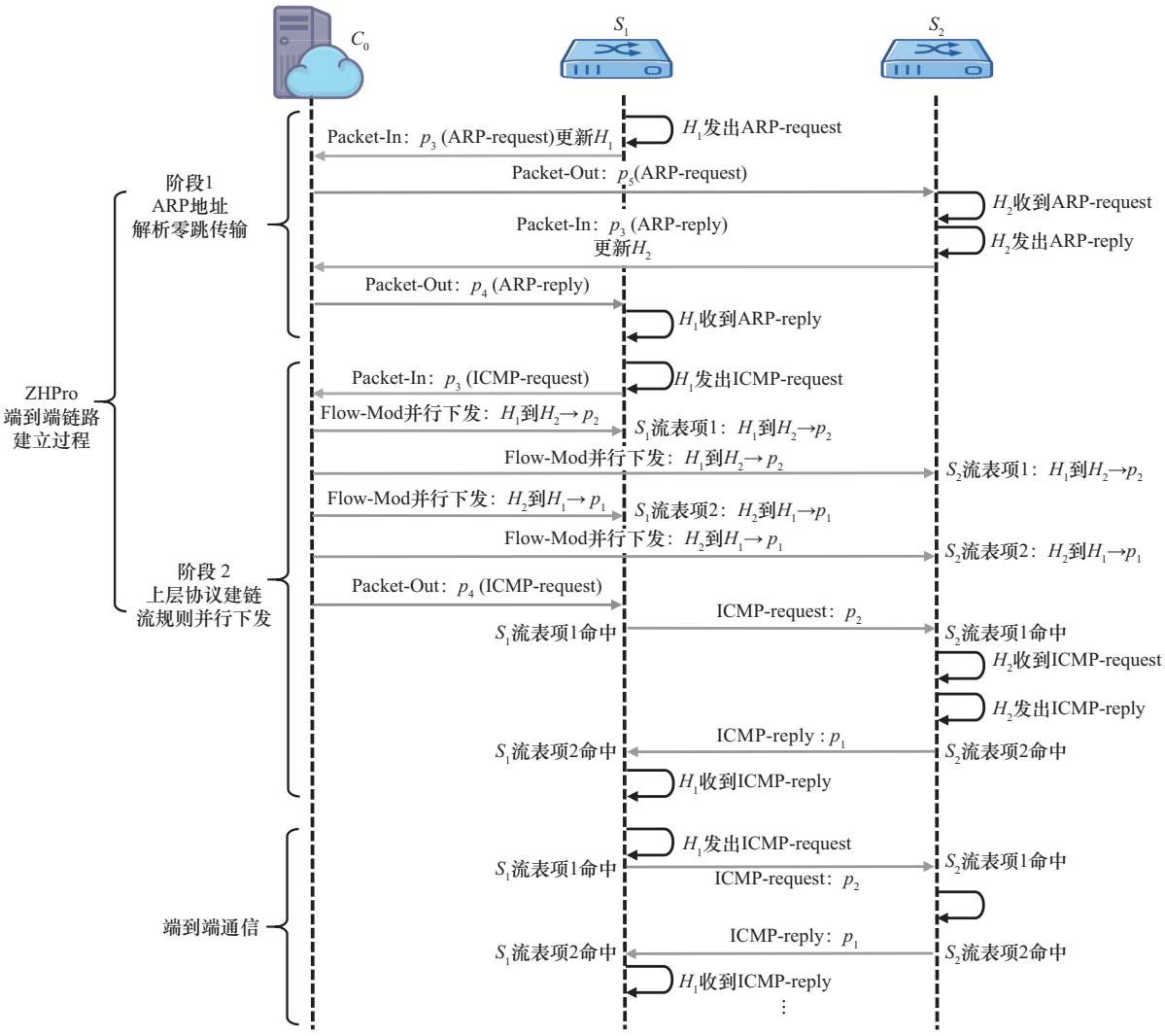


图 8　采用 ZHPro 后，SDN 网络端到端链路建立过程时序图与端到端通信过程时序图

示设备端口，端口编号对应图1中的简单拓扑。

## 4 实验评估与分析

### 4.1 实验设置与数据集

为验证ZHPro的有效性，本文选取业界主流的控制器作为对比基线，包括ONOS、Ryu及ODL，其中ODL选取的是功能支持实验测量的最新版本，Ryu与ONOS均采用开源最新版本进行实验对比。为方便对比，控制器软件均运行在当前业界主流的高性能服务器上，其配置为：处理器Intel Xeon Platinum 8368 (76核)，内存512 GB，硬盘32 TB，操作系统版本为Ubuntu 20.04。

实验采用广泛认可的公开网络拓扑数据集The Internet Topology Zoo[29]作为模型训练与测试的基础数据源。表1展示了数据中Belnet、Tinet、RedBestel、Deltacom及UsCarrier 5个典型网络拓扑结构参数。

该数据集涵盖了全球范围内真实网络的拓扑结构，包括互联网服务提供商、企业网络、研究网络等多种类型，具备以下代表性。

1) 规模多样性：Internet Topology Zoo包含从小型网络（节点数 < 10）到大型复杂网络（节点数 > 100）的拓扑结构，能够充分验证不同网络规模下方案的适应性。

2) 拓扑异构性：数据集中的网络拓扑类型丰富，包括树形、环形、网状等典型结构，以及混合型拓扑，可全面反映真实网络环境的复杂性。

3) 真实性与可扩展性：所有拓扑均基于实际网络配置生成，避免了仿真拓扑的简化假设问题，同时支持通过拓扑组合或节点扩展构建更复杂的实验场景。

### 4.2 主实验

为全面评估本文提出的端到端链路建立时间优化方法性能，本节首先在线形、树形、环形及二维环面拓扑结构下开展对比实验。

图9展示了在线形拓扑结构下，随着交换机数量增加，不同控制器的端到端链路建立时间变化趋势。从图9可以看出，ZHPro在所有规模下均表现出显著的性能优势。具体而言，当网络规模达到300台交换机时，ONOS的链路建立时间高达7 092.9 ms，Ryu为5 927.7 ms，ODL为3 904.0 ms，而ZHPro仅需103.8 ms，相较于ONOS、Ryu和ODL分别降低了98.5%、98.2%和97.3%。

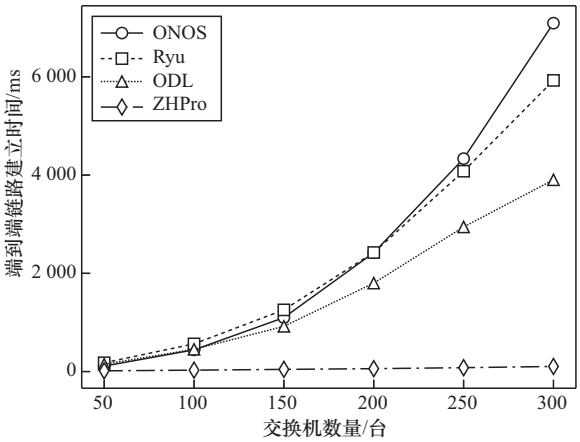


图9 线形拓扑结构下端到端链路建立时间对比

图10为树形拓扑结构下端到端链路建立时间对比。随着树深度的增加，网络节点数呈指数增长，这对链路建立机制的可扩展性提出了更高挑战。实验结果表明，在深度为9的树形拓扑中，ONOS和ODL的链路建立时间分别达到681.5 ms和653.7 ms，而ZHPro仅为47.3 ms，效率提升约93.1%。值得注意的是，Ryu在深度为9的树形拓扑中出现了运行异常，无法完成链路建立过程，这反映出该控制器在处理大规模树形拓扑时的稳定性问题。此外，从时间增长趋势来看，3种现有控制器均在深度为7~8出现了明显的拐点，时间增长率显著提高，这与树形拓扑节点数的指数增长特性相

表1 测试集网络拓扑结构参数

| 拓扑名 | 节点数/个 | 边数/条 | 平均节点度 | 密度 | 直径 | 平均路径长度 | 平均聚类系数 |
|---|---|---|---|---|---|---|---|
| Belnet | 23 | 41 | 3.565 | 0.162 | 3 | 2.146 | 0.732 |
| Tinet | 53 | 89 | 3.358 | 0.065 | 9 | 4.250 | 0.216 |
| RedBestel | 84 | 93 | 2.214 | 0.027 | 28 | 10.589 | 0.009 |
| Deltacom | 113 | 161 | 2.850 | 0.025 | 23 | 7.160 | 0.107 |
| UsCarrier | 158 | 189 | 2.392 | 0.015 | 35 | 12.090 | 0.058 |

符。相比之下，本文方法的时间曲线保持平稳增长，即使在最大规模下仍保持较低的端到端链路建立时间。

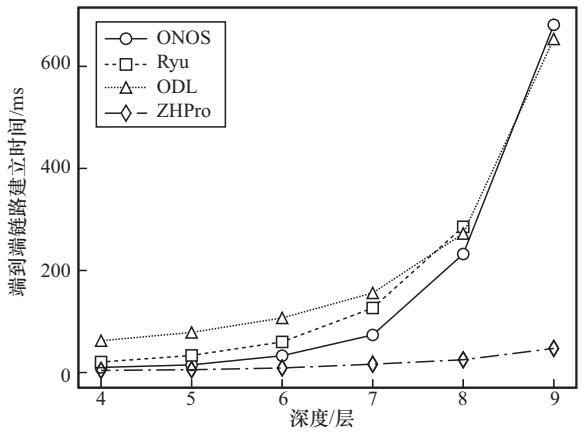


图 10　树形拓扑结构下端到端链路建立时间对比

图 11 为环形拓扑结构下端到端链路建立时间对比。环形拓扑因其特殊的闭环结构，对控制器的链路建立机制提出了更高要求。实验数据显示，Ryu 控制器在环形拓扑下完全无法工作，这主要是因为其简单的广播机制无法处理环路导致的广播风暴。ONOS 和 ODL 虽然能够在环形拓扑下正常工作，但在 300 节点规模下，链路建立时间仍高达 2 524.7 ms 和 2 013.0 ms。相比之下，ZHPro 通过零跳传输机制有效避免了广播风暴问题，即使在最大规模下时间也仅为 56.03 ms，相较于 ONOS 和 ODL 分别降低了 97.8% 和 97.2%。此外，从时间增长趋势来看，ZHPro 在环形拓扑下的性能优势随着规模增大而更加明显，这充分证明了所提机制在处理复杂拓扑结构时的有效性。

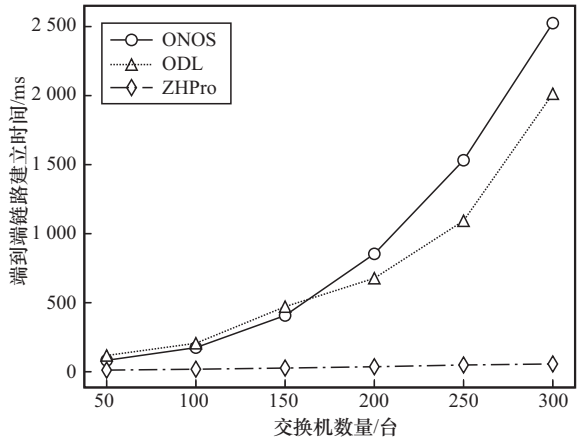


图 11　环形拓扑结构下端到端链路建立时间对比

图 12 为二维环面拓扑结构下端到端链路建立时间对比。环面拓扑结合了网格和环形的特性，具有更高的连通性和冗余路径，对链路建立机制的挑战更大。从图 12 可以看出，随着环面规模从 8×8 增加到 16×16，ONOS 的链路建立时间从 43.98 ms 激增至 1 640.0 ms，增长了约 36 倍。ODL 在 14×14 规模时达到 423.0 ms，但在 16×16 规模下已无法正常工作。本文方法虽然在环面拓扑下的优势相对减弱，但在大多数规模下仍保持明显领先，特别是在 8×8 至 12×12 规模范围内，时间优化幅度均超过 50%。值得注意的是，在最大 16×16 规模下，本文方法的时间为 1 387.25 ms，与 ONOS 的 1 640.0 ms 相差不大。

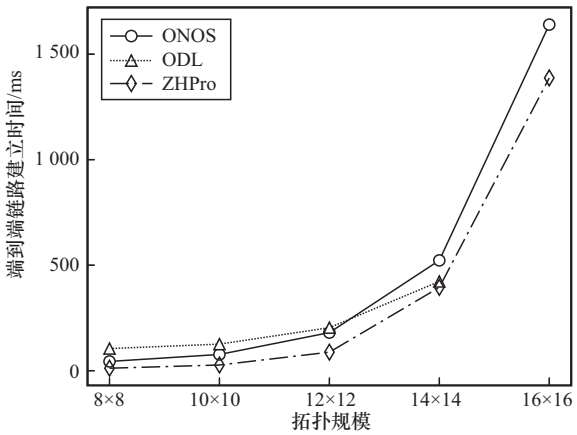


图 12　二维环面拓扑结构下端到端链路建立时间对比

导致这一现象的原因有 2 个：1)由于网络拓扑复杂度的提升，控制器的路径规划算法耗时显著增加；2)由于网络拓扑复杂度的提升，流表下发所需的并行通道需求增加，并行通道竞争控制器资源导致并行下发机制的效率受到了一定限制，当其超出控制器允许的最大并行数时，超出部分退化为串行下发。

为进一步验证本文方法在实际网络拓扑中的适用性，选取了 Internet Topology Zoo 数据集中的 5 种典型拓扑进行测试，实验结果如表 2 所示。

**表 2　　数据集拓扑链路建立时间对比**

| 方案 | 链路建立时间/ms | | | | |
|---|---|---|---|---|---|
| | Belnet | Tinet | RedBestel | Deltacom | UsCarrier |
| ONOS | 6.02 | 25.00 | 97.60 | 143.00 | 260.00 |
| ODL | **0.85** | 10.50 | 50.40 | 102.10 | 233.67 |
| ZHPro | 4.96 | **8.16** | **11.60** | **14.40** | **20.50** |

从表2可以看出，在所有测试的实际网络拓扑中，本文方法均表现出明显的性能优势。在中小规模网络Belnet中，ODL控制器表现出色，链路建立时间仅为0.85 ms，优于ONOS的6.02 ms和本文方法的4.96 ms。这一现象表明，在节点数较少且连接密度较高的网络中，ODL的轻量级处理机制具有一定优势。然而，随着网络规模和复杂度的增加，ODL和ONOS的性能迅速下降，而本文方法则保持相对稳定的性能。

特别值得关注的是，在大规模稀疏拓扑中的表现。在RedBestel拓扑中，ONOS和ODL的链路建立时间分别为97.60 ms和50.40 ms，而本文方法仅需11.60 ms，相较于两者分别降低了88.1%和77.0%。在最大规模的UsCarrier拓扑中，本文方法的优势更为显著，仅为20.50 ms，相比ONOS的260.00 ms和ODL的233.67 ms，分别降低了92.1%和91.2%。

这些结果进一步证实了本文方法在处理复杂实际网络时的有效性。实际网络拓扑通常具有不规则的结构和多样化的连接模式，这对控制器的路径规划和流表管理能力提出了更高要求。本文提出的流规则并行下发机制通过路径规划与批量下发策略，很好地适应了这种复杂性，实现了高效的链路建立。同时，本文方法在不同拓扑间的性能波动相对较小。从Belnet到UsCarrier，建链时间增长了约4倍，而同期ONOS和ODL分别增长了约43倍和275倍。这种稳定性主要得益于零跳传输机制对网络拓扑变化的适应能力，以及并行下发机制对网络规模的良好扩展性。

综上所述，无论是在经典拓扑中，还是在真实世界的复杂网络中，本文方法均表现出显著的性能优势。特别是在大规模网络场景下，现有控制器因串行处理和广播风暴等问题导致性能急剧下降，而本文方法通过创新的零跳传输和流规则并行下发机制，成功维持了稳定的高效表现，表明其具有良好的适应性和鲁棒性。

### 4.3　消融实验

为深入验证本文提出的优化机制中各模块的独立贡献，本节针对零跳传输（优化A）与流规则并行下发（优化B）2个核心机制开展消融实验。实验在线形、树形与环形3种典型拓扑下分别测试单独启用及组合启用优化机制的性能表现，结果如表3所示。

**表3　消融实验结果**

| 用例编号 | 拓扑结构 | 优化A | 优化B | 时间/ms | 降低比例 |
|---|---|---|---|---|---|
| 1 | 线形拓扑（300台交换机） | — | — | 7 070 | — |
| 2 | | √ | — | 6 294 | 10.98% |
| 3 | | — | √ | 1 104 | 84.38% |
| 4 | | √ | √ | 104 | 98.53% |
| 5 | 树形拓扑（深度9） | — | — | 641 | — |
| 6 | | √ | — | 620 | 3.28% |
| 7 | | — | √ | 226 | 64.74% |
| 8 | | √ | √ | 47 | 92.67% |
| 9 | 环形拓扑（300台交换机） | — | — | 2 525 | — |
| 10 | | √ | — | 1 997 | 20.91% |
| 11 | | — | √ | 656 | 74.02% |
| 12 | | √ | √ | 56 | 97.78% |

从表3可以看出，2种优化机制的独立贡献及其协同效应。在线形拓扑（300台交换机）场景下，单独启用零跳传输机制（用例2）可将链路建立时间从7 070 ms降至6 294 ms，降低比例为10.98%；而单独启用流规则并行下发机制（用例3）则可将时间降至1 104 ms，降低比例高达84.38%。当2种机制同时启用时（用例4），时间进一步降至104 ms，总体降低比例达到98.53%。这一结果表明，在线形拓扑中，流规则并行下发机制是性能提升的主要贡献者。

在树形拓扑（深度9）场景中，单独启用零跳传输机制（用例6）的效果相对有限，仅降低3.28%；而流规则并行下发机制（用例7）则贡献了64.74%的性能提升。这是因为在树形拓扑中，由于其层次化结构，ARP广播的影响相对较小，主要瓶颈仍在于流规则下发环节。然而，当2种机制协同工作时（用例8），时间降低比例达到92.67%，远超单一机制的贡献总和。这表明在树形拓扑中，2种机制之间存在正向的协同增强作用。

环形拓扑（300台）的实验结果展现出更为均衡的贡献分布。单独启用零跳传输机制（用例10）可降低20.91%的耗时，明显优于线形和树形拓扑中的贡献。这是因为环形结构容易导致广播风暴，零跳传输通过避免广播，有效解决了这一问题。流规则并行下发机制（用例11）仍然是主要贡献者，

降低了 74.02%。2 种机制协同工作时（用例 12），总体降低比例达到 97.78%，同样展现出显著的协同效应。

最后，从降低比例来看，3 种拓扑下的最终优化效果较为接近，这表明本文方法具有良好的通用性，能够适应不同类型的网络结构。

### 4.4　可扩展实验

为验证 ZHPro 的可扩展能力，本文在线形拓扑结构下，进一步扩展了测试网络规模。扩展后，测试网络中的最大交换机数量可达 2 000 台。图 13 展示了 ZHPro 分别在 100、300、500、1 000、2 000 台交换机下端到端链路建立时间重复测试 5 次的变化趋势，其中实线为理论分析下的线性拟合，虚线为实际测试得到的多项式拟合。

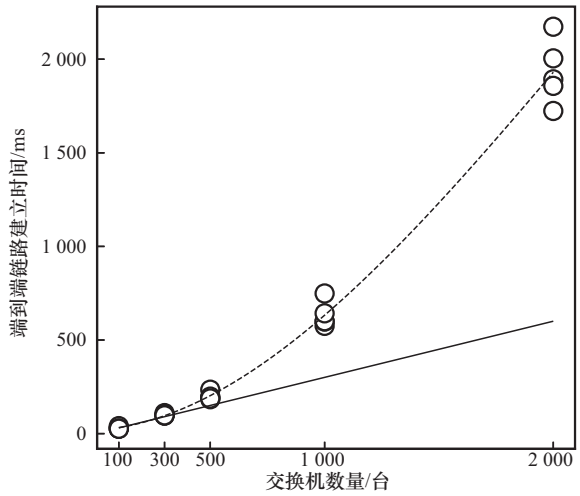


图 13　端到端链路建立时间随网络规模扩大的变化趋势

当网络规模达到 2 000 台交换机时，5 次测试的平均建立时间为 1 930.8 ms。尽管如此，与其他控制器在 300 台交换机网络规模下的建立时间相比，ZHPro 的链路建立耗时依然具有显著优势，仅约为最快的 ODL（3 904.0 ms）的一半。理论上，随着交换机数量增加，ZHPro 的端到端链路建立时间应线性增长，对应图中实线；然而，其实际的增长趋势拟合多项式曲线，对应图中虚线。一方面，与理论分析不同，ZHPro 中的路径规划算法实际所耗时间并非一成不变，其随网络规模的增一同增长；另一方面，ZHPro 的并行下发策略无法做到理论上的完全并行，当超出控制器分配的最大线程数时，超出部分随即退化为串行下发。

值得注意的是，当网络规模达到 500 台交换机时，其他控制器逐步出现无法完成端到端链路建立的情况。当网络规模达到 1 000 台交换机时，其他控制器已无法完成端到端的链路建立。

端到端链路建立过程主要发生在网络初始化及网络拓扑发生变化阶段，此时交换机中不存在完成两主机间正常通信所需要的流表信息，未命中流表的消息将被上送至控制器，由控制器分析处理后完成流表信息的下发，以保证主机间的正常通信。

图 14 展示了 ZHPro 分别在 100、300、500、1 000、2 000 台交换机下端到端通信时延的变化趋势，其中直线为实际测试结果的线性拟合。实验结果表明，端到端通信时延与通信链路长度线性正相关，跨 100 台交换机时的平均通信时延仅为 0.147 ms，跨 2 000 台交换机时的平均通信时延仅为 5.955 ms。这也证明了端到端链路建立过程完成后的端到端通信时延可以达到毫秒级，因此能够满足未来网络服务对时延性能的要求。

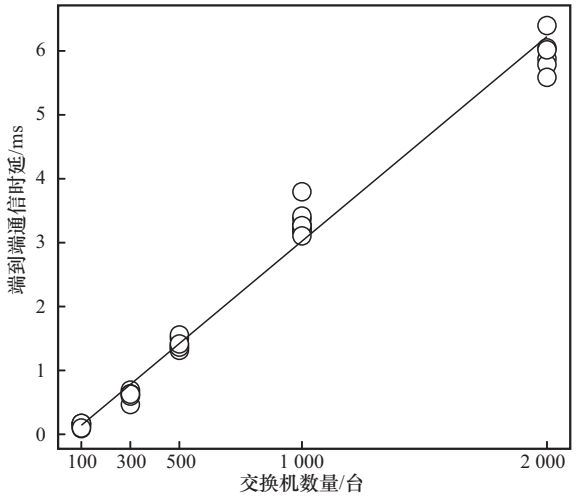


图 14　端到端通信时延随网络规模扩大的变化趋势

与此同时，本文对 ZHPro 在提升单控制器并发管控交换机网络规模方面的扩展性进行了深入的实验对比分析。在 SDN 架构中，控制器依赖控制平面维护的全局网络拓扑视图执行决策与控制操作，仅当控制平面与数据平面的拓扑视图保持一致性时，控制平面方可生成精准的控制决策，数据平面才能准确执行相应指令。随着网络规模持续扩大，控制器为维护拓扑视图一致性所需的同步开销显著增加，其中控制与数据平面间控制信道的网络带宽成为关键约束因素，直接限制了单控制器并发管控的交换机规模上限。上述带宽主要包含维持网络拓

扑所需的带宽和维持网络正常业务所需的带宽。ONOS、ODL及Ryu等主流SDN控制器的现有建链机制中，端到端链路建立过程存在控制与数据平面间的数据交互冗余问题，导致业务类消息占用较多带宽，进而影响了拓扑维护所需的带宽资源，从而限制了单控制器可管控的交换机数量。相比之下，ZHPro采用的两阶段优化机制，显著减少了端到端链路建立过程中控制与数据平面的交互次数，大幅降低了业务类消息对控制信道的带宽占用，保障了大规模网络拓扑维护所需的带宽资源，最终实现了控制信道整体带宽开销的有效降低，从而有效提升了大规模网络中控制器的可扩展性。

## 5 结束语

本文提出了ZHPro——一种针对SDN端到端链路建立过程的基于零跳传输和流规则并行下发的新型两阶段优化方法。ZHPro将端到端链路建立过程分为ARP地址解析和上层协议建链2个阶段，并分阶段针对性设计了零跳传输和流规则并行下发机制。本文通过大量的实验证明了ZHPro的性能优越性：1)主实验基于真实网络拓扑数据集开展了大量各类拓扑的实验对比分析，结果表明，与ONOS、Ryu、ODL等控制器相比，ZHPro可将端到端链路建立时间平均降低90%，在较大规模的UsCarrier拓扑场景下，较ONOS控制器性能提升12.6倍；2)消融实验揭示了双机制的协同效应，零跳传输有效抑制了广播风暴，而流规则并行下发则消除了串行累积时延，两者结合实现了端到端链路建立时间的优化；3)可扩展实验验证了ZHPro在大规模网络下的可扩展性与应用可行性，当网络规模扩展到2 000台交换机时，ZHPro的端到端链路建立时间仅是网络规模为300台交换机时ODL链路建立时间的一半，链路建立完成后的端到端通信时延为5.955 ms，满足未来网络服务对时延性能的要求。

随着网络拓扑规模的持续扩展，单服务器硬件资源受限，成为制约SDN控制器性能提升的关键因素。通过横向扩展构建分布式集群架构，实现控制器多节点硬件资源的池化调度，是突破该瓶颈的有效解决方案。在单节点性能优化方面，通过机制创新，ZHPro已在端到端链路建立时延等关键性能指标上显著优于现有主流控制器，在同等硬件资源配置下，实现了控制器单节点性能和可同时管控交换机规模的显著提升。当网络规模向更大维度扩展时，采用多服务器集群部署模式可进一步释放控制层算力，持续提升整体控制性能。未来笔者将进一步从拓扑发现时间、入包消息处理响应时延、多集群一致性更新等多个方面对控制器性能进行进一步优化，助力算力网络时代的软件定义网络性能提升。

## 参考文献：

[1] TANG X Y, CAO C, WANG Y X, et al. Computing power network: The architecture of convergence of computing and networking towards 6G requirement[J]. China Communications, 2021, 18(2): 175-185.

[2] MCKEOWN N. Software-defined networking[J]. INFOCOM keynote talk, 2009, 17(2): 30-32.

[3] YANG Z J, CUI Y, LI B C, et al. Software-defined wide area network (SD-WAN): architecture, advances and opportunities[C]//Proceedings of the 2019 28th International Conference on Computer Communication and Networks (ICCCN). Piscataway: IEEE Press, 2019: 1-9.

[4] KREUTZ D, RAMOS F M V, VERÍSSIMO P E, et al. Software-defined networking: a comprehensive survey[J]. Proceedings of the IEEE, 2015, 103(1): 14-76.

[5] 付钰, 王坤, 段雪源, 等. 面向软件定义网络的异常流量检测研究综述[J]. 通信学报, 2024, 45(3): 208-226.
FU Y, WANG K, DUAN X Y, et al. Survey of research on abnormal traffic detection for software defined networks[J]. Journal on Communications, 2024, 45(3): 208-226.

[6] ZHU L H, KARIM M M, SHARIF K, et al. SDN controllers: a comprehensive analysis and performance evaluation study[J]. ACM Computing Surveys, 2021, 53(6): 1-40.

[7] BOSI L L, MENDES A C, SALLES R M. A review on the overall performance of SDN controllers[C]//Proceedings of the 2024 11th International Conference on Software Defined Systems (SDS). Piscataway: IEEE Press, 2024: 156-163.

[8] PHEMIUS K, BOUET M, LEGUAY J. DISCO: distributed multi-domain SDN controllers[C]//Proceedings of the 2014 IEEE Network Operations and Management Symposium (NOMS). Piscataway: IEEE Press, 2014: 1-4.

[9] JARSCHEL M, WAMSER F, HOHN T, et al. SDN-based application-aware networking on the example of YouTube video streaming[C]//Proceedings of the 2013 Second European Workshop on Software Defined Networks. Piscataway: IEEE Press, 2013: 87-92.

[10] CURTIS A R, MOGUL J C, TOURRILHES J, et al. DevoFlow: scaling flow management for high-performance networks[J]. 2011, 41(4): 254-265.

[11] 全球SDN测试认证中心(SDNCTC). ONOS控制器性能测试报告 [R]. 2016.
Global SDN Certified Testing Center (SDNCTC). Performance test report of ONOS controller [R]. 2016.

[12] HE M, BASTA A, BLENK A, et al. Modeling flow setup time for controller placement in SDN: Evaluation for dynamic flows[C]//Proceedings of the 2017 IEEE International Conference on Communications

(ICC). Piscataway: IEEE Press, 2017: 1-7.

[13] KANG X Y, TAKAHASHI K, NAKASAN C, et al. Multi-objective optimization of controller placement in distributed ONOS networks[C]// Proceedings of the 2023 Eleventh International Symposium on Computing and Networking (CANDAR). Piscataway: IEEE Press, 2023: 76-85.

[14] PAPA A, COLA T D, VIZARRETA P, et al. Dynamic SDN controller placement in a LEO constellation satellite network[C]//Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM). New York: ACM Press, 2018: 206-212.

[15] 熊兵, 袁月, 赵锦元, 等. ADAFT: SDN大规模流表的适应性深度聚合存储架构[J]. 通信学报, 2024, 45(5): 226-238.
XIONG B, YUAN Y, ZHAO J Y, et al. ADAFT: an storage architecture of large-scale SDN flow tables based on adaptive deep aggregations[J]. Journal on Communications, 2024, 45(5): 226-238.

[16] LI Q, HUANG N Y, WANG D M, et al. HQTimer: a hybrid ${Q}$-learning-based timeout mechanism in software-defined networks[J]. IEEE Transactions on Network and Service Management, 2019, 16(1): 153-166.

[17] YANG H M, RILEY G F. Machine learning based flow entry eviction for OpenFlow switches[C]//Proceedings of the 2018 27th International Conference on Computer Communication and Networks (ICCCN). Piscataway: IEEE Press, 2018: 1-8.

[18] ALOWA A, FEVENS T. Towards minimum inter-controller delay time in software defined networking[J]. Procedia Computer Science, 2020, 175: 395-402.

[19] KO K, SON D, HYUN J, et al. Dynamic failover for SDN-based virtual networks[C]//Proceedings of the 2017 IEEE Conference on Network Softwarization (NetSoft). Piscataway: IEEE Press, 2017: 1-5.

[20] CHAN K Y, CHEN C H, CHEN Y H, et al. Fast failure recovery for in-band controlled multi-controller OpenFlow networks[C]//Proceedings of the 2018 International Conference on Information and Communication Technology Convergence (ICTC). Piscataway: IEEE Press, 2018: 396-401.

[21] IBRAR M, WANG L, MUNTEAN G M, et al. PrePass-Flow: a machine learning based technique to minimize ACL policy violation due to links failure in hybrid SDN[J]. Computer Networks, 2021, 184: 107706.

[22] LIU W X, CAI J, CHEN Q C, et al. DRL-R: deep reinforcement learning approach for intelligent routing in software-defined data-center networks[J]. Journal of Network and Computer Applications, 2021, 177: 102865.

[23] 李元诚, 秦永泰. 基于深度强化学习的软件定义安全中台Qo S实时优化算法[J]. 通信学报, 2023, 44(5): 181-192.
LI Y C, QIN Y T. Deep reinforcement learning based algorithm for real-time QoS optimization of software-defined security middle platform[J]. Journal on Communications, 2023, 44(5): 181-192.

[24] PAKZAD F, PORTMANN M, HAYWARD J. Link capacity estimation in wireless software defined networks[C]//Proceedings of the 2015 International Telecommunication Networks and Applications Conference (ITNAC). Piscataway: IEEE Press, 2015: 208-213.

[25] SINGH M, VARYANI N, SINGH J, et al. Estimation of end-to-end available bandwidth and link capacity in SDN[C]//Ubiquitous Communications and Network Computing. Cham: Springer, 2018: 130-141.

[26] DENEKE B B, BEYENE A M, HAILE E A. Improving Software Defined Network controllers in a multi-vendor environment[J]. Heliyon, 2024, 10(4): e26215.

[27] QIU G H, LU S, YANG L, et al. Optimization and experimental study of controller deployment for SDN-based satellite networks[C]//Proceedings of the 2024 China Automation Congress (CAC). Piscataway: IEEE Press, 2024: 1261-1266.

[28] MCKEOWN N, ANDERSON T, BALAKRISHNAN H, et al. OpenFlow: enabling innovation in campus networks[J]. 2008, 38(2): 69-74.

[29] KNIGHT S, NGUYEN H X, FALKNER N, et al. The Internet topology zoo[J]. IEEE Journal on Selected Areas in Communications, 2011, 29(9): 1765-1775.

[作者简介]

**赵宝康**（1981– ），男，湖北天门人，博士，国防科技大学副教授，主要研究方向为网络体系结构与协议、卫星互联网、高性能网络、网络安全等。

**李羿锟**（2000– ），男，贵州贵阳人，国防科技大学硕士生，主要研究方向为计算机网络、网络安全等。

**杨宇**（1997– ），男，湖南长沙人，国防科技大学工程师，主要研究方向为计算机网络、网络安全等。