面向蜻蜓拓扑的在线自适应路由及故障自愈技术

赖明澈 王正浩 徐佳庆 高 蕾 欧 洋
吴利舟 王 强
（国防科技大学计算机学院 长沙 410073）

摘 要 面向后E级超智算通信系统采取蜻蜓拓扑具有可扩展性强、延迟小、成本低优势。蜻蜓拓扑中自适应路由和容错路由对网络性能和稳定性至关重要。传统路由方法结合局部流量选择多样化路径对路径跳步影响考虑不足，性能优化效果有限。本文提出一种分布式在线自适应路由DOAR算法，传输过程中根据组内局部通道和组间全局通道状态动态选择路径，缓解拥塞同时降低跳步数，提升网络性能；提出路由计算部件结构及可重构路由配置方法，具有计算延迟低、配置灵活、无死锁特点。针对蜻蜓拓扑网络故障敏感特点提出动态路由自愈及恢复技术，主动产生诊断或恢复报文更新分布式路由表，在保证全局目的可达同时最大化可达路径数量，实现网络故障快速自动诊断与恢复，并减少网络性能损失和抖动。实验表明DOAR与PAR_{PH}、TPR、UGAL_LE相比，在合成流量下吞吐量分别提升约15.2%、12.1%和23.7%；在混合流量下吞吐量分别提升约15.1%、14.4%和41.2%；在实际应用负载下通信延迟分别降低约8.2%、30.4%和26.0%。针对典型故障场景，标准合成流量下40和200处链路故障时吞吐量最大损失仅1.7%和7.4%；10和30处路由器故障时吞吐量最大损失仅5.3%和10.5%；混合合成流量100处链路故障时吞吐量平均损失仅2.9%。高负载随机流量50处链路故障下故障自动诊断和恢复过程耗时分别约5.2和10.3 μs，带宽损失和延迟增幅仅约1.1%和5.7%，体现较强的性能优势与自动故障容错能力。

关键词 高性能网络；蜻蜓拓扑；自适应路由；容错机制
中图法分类号 TP393 **DOI号** 10.11897/SP.J.1016.2026.00721

An On-the-Fly Adaptive Routing and Fault Self-Healing Technology for Dragonfly Topology in Post-Exascale Supercomputer

LAI Ming-Che WANG Zheng-Hao XU Jia-Qing GAO Lei OU Yang
WU Li-Zhou WANG Qiang
（College of Computer Science and Technology, National University of Defense Technology, Changsha 410073）

Abstract For post-exascale supercomputing and intelligent computing communication systems, the Dragonfly topology exhibits advantages of strong scalability, low latency, and cost-effectiveness. Adaptive routing and fault-tolerant routing are critical to network performance and stability in the Dragonfly topology. Traditional routing methods, when selecting diverse paths through local traffic, insufficiently consider the impacts of path hop counts, resulting in limited performance optimization. This paper proposes a Distributed On-the-Fly Adaptive Routing (DOAR) algorithm, which dynamically selects paths based on the status of intra-group local channels and inter-group global channels during data transmission. This approach alleviates

收稿日期：2025-03-01；在线发布日期：2025-11-07。本课题得到国家重点研发计划项目（2023YFB4403400）资助. **赖明澈**，博士，教授，主要研究领域为计算机系统结构、高性能网络、集成电路。E-mail：mingchelai@nudt.edu.cn。**王正浩**，博士研究生，主要研究领域为高性能网络、芯粒互连。**徐佳庆**，博士，副研究员，研究方向为高性能计算和高性能互连网络。**高 蕾**，博士，副研究员，研究方向为计算机体系结构、人工智能和高性能互连网络。**欧 洋**，博士，副研究员，研究方向为计算机体系结构和高性能互连网络。**吴利舟**，博士，助理研究员，研究方向为异构存储池、计算机体系结构和高性能互连网络。**王 强**（通信作者），博士，助理研究员，主要研究领域为高性能网络、人工智能。E-mail：qiangwang@nudt.edu.cn。

congestion, reduces hop counts, and enhances network performance. Additionally, we present the architectural design of routing computation units and a reconfigurable routing configuration method, characterized by low computational latency, flexible configuration, and deadlock-free operation. To address the fault sensitivity of Dragonfly topology, dynamic routing self-healing and recovery techniques are proposed. These techniques proactively generate diagnostic or recovery packets to update distributed routing tables, ensuring global destination reachability while maximizing the number of available paths. This enables rapid automatic diagnosis and recovery from network faults, minimizing performance degradation and jitter. Experimental results demonstrate that compared to PAR_{PH}, TPR, and UGAL_LE under synthetic traffic, DOAR achieves throughput improvements of approximately 15.2%, 12.1%, and 23.7%, respectively; under mixed traffic, throughput improvements reach 15.1%, 14.4%, and 41.2%, respectively; and under real application workloads, communication latency is reduced by approximately 8.2%, 30.4%, 26.0%, respectively. For typical fault scenarios, under standard synthetic traffic with 40 and 200 link failures, maximum throughput losses are only 1.7% and 7.4%, respectively; with 10 and 30 router failures, maximum throughput losses are limited to 5.3% and 10.5%, respectively. Under mixed synthetic traffic with 100 link failures, average throughput loss is only 2.9%. For automatic diagnosis and recovery processes under high-load random traffic with 50 link failures, the required time is approximately 5.2 μs and 10.3 μs, respectively. Meanwhile, the network bandwidth loss and latency increase are only about 1.1% and 5.7%, respectively, showcasing strong performance advantages and automated fault tolerance capabilities.

Keywords high-performance interconnection network; dragonfly topology; adaptive routing; fault-tolerance

1 引 言

高性能网络属于超级计算和智能计算中心重要基础设施。近十年高性能计算机发展迅猛，系统规模持续增加[1]，给网络成本、功耗和稳定性带来巨大挑战。最常见的“胖树”[2]拓扑流量均衡且对分带宽高，但可扩展性差[3]，天河[4-5]和神威[6]最大仅数万节点规模。“胖树”依靠增加层级扩展规模，但四层以上树形拓扑功耗和成本指数级增长，难以支撑未来10E级高性能计算机数十万点规模。最近基于超高阶路由器[7]的新型拓扑不断涌现，如Flattened Butterfly[8]、HyperX[9]、Dragonfly[10-12]、Slimfly[13]、Megafly[14]、Galaxyfly[15]等。其中蜻蜓(Dragonfly)拓扑由于扩展性和高性价比优势在部分系统中逐渐被采用，如Frontier、Aurora等[16]。

蜻蜓拓扑任意两点之间采取最小路径(Min)通信跳步数少，但仅靠少量Min路径难以适应复杂流量。通常考虑利用大量非最小路径(Non-Min)，通过增加多样化路径以缓解拥塞。关于Non-Min路径选择十分关键，通过局部流量选择无法感知远程拥塞导致次优决策；随机路径选择较大程度实现负载平衡，但路径长度并非最优，过多资源消耗会制约吞吐量性能。大规模蜻蜓拓扑如何适应各种复杂流量，动态选择理想路径缓解拥塞并减少网络资源消耗，成为通信系统的关键性问题。同时，故障对蜻蜓拓扑敏感，较大影响整个系统稳定性和使用效率。由于蜻蜓拓扑复杂，其他容错算法难以直接适用。理想算法不仅需要考虑故障发生时的鲁棒通信，而且还需要最小化性能损失，拓扑自动快速收敛，减少网络性能波动。

本文面向蜻蜓拓扑提出一种高性能高可靠路由框架，自适应路由技术能适应各种复杂流量，缓解拥塞，优化通信性能；故障自愈与恢复技术能保持通信稳定且性能损失小。本文主要贡献包括：(1)提出一种分布式在线自适应路由(DOAR)算法，根据局部和全局通道检测，在线动态选择路由缓解拥塞并减少跳步数；提出路由计算结构和可重构路由配置方法，具有延迟低、配置灵活、无死锁特点；(2)提出动态路由自愈算法及硬件结构，通过监听链路故障，主

动产生诊断或恢复报文更新路由表，支持故障自动诊断与恢复，无需用户干预对上层透明；(3)实验表明，DOAR算法在标准合成流量下相对PAR_{PH}、TPR、UGAL_LE算法吞吐量提高约15.2%、12.1%和23.7%；在混合流量下吞吐量提高约15.1%、14.4%和41.2%；在实际应用负载下通信延迟降低约8.2%、30.4%和26.0%；(4)DOAR算法故障发生时性能损失小。标准合成流量40和200处链路故障时吞吐量最大损失仅1.7%和7.4%；10和30处路由器故障时吞吐量最大损失仅5.3%和10.5%。与SIM算法比较，40处链路故障时吞吐量提高19%，延迟降低14%；10处路由器故障时吞吐量提高23.2%，延迟降低14.8%。混合合成流量100处链路故障时吞吐量平均损失仅2.9%；(5)在接近饱和高负载流量典型链路故障场景下故障自动诊断和恢复过程耗时分别约5.2和10.3 μs，网络性能波动小，带宽损失和延迟增幅分别约1.1%和5.7%，体现良好的故障处置能力。

2　相关工作

2.1　蜻蜓拓扑结构

蜻蜓拓扑路由器在不同维度上分配端口，低维度端口组内连接形成虚拟交换，高维度端口在虚拟交换之间连接构成网络[10]。如图1，路由器S_i(i=0，…，a−1)含三组端口，p个端口连接本地节点，a-1个端口连接组内路由器，h个端口负责组间互连。每个虚拟交换对外端口数为ah，最多连接$g=ah+1$个分组，系统可扩展至$(ah+1)ap$个节点。本文采取$dfly(p,a,h,g)$表示蜻蜓拓扑，假设p=16，a=32，h=16，系统可扩展至26万点。

蜻蜓拓扑Min路径从源分组G_s直达目标分组G_d不经中间分组G_i，报文经过2个局部跳L和1个全局跳G，一般表示为L-G-L。采取Min路径可满足随机流量，但在对抗流量中带宽收缩比仅$1/ap$。为适应复杂流量模式，Non-Min路径在源、中间和目标分组内分别经过1、2和1个局部跳L，路径最大为LG-LL-GL。这种策略能增加极端流量下路径数量，但会伴随延迟全面增加。为减少路径跳步数也可采取：1)源组内选择与当前路由器相邻的中间分组，路径为G-LL-GL；2)中间组内直接经出口路由器输出，路径为LG-L-GL。这两种策略也无法适应所有流量。G-LL-GL策略不适应间距小于h的对抗流量(组偏移流量)ADV_c[17]，源分组和中间分组流量在全局链路处冲突，中间分组优先级高会抑制源分组流量；LG-L-GL策略不适应间距为h的对抗流量(组对传流量)$ADV+h$[18]，中间分组两个路由器h个全局端口通过唯一组内链路对传，带宽收缩至$1/h$。因此，固定策略难以适应流量变化，理想自适应路由应主动适应复杂流量缓解拥塞并提供最少跳步，降低延迟并提高带宽性能。

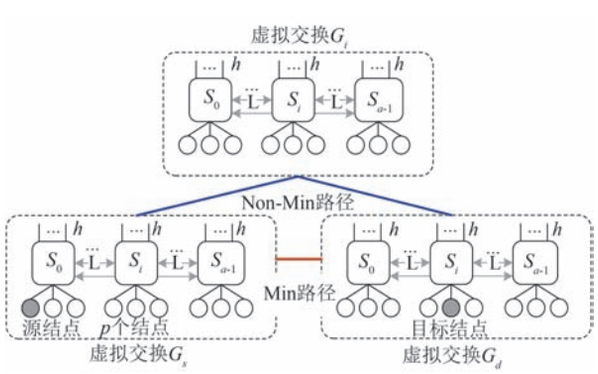


图1　蜻蜓拓扑分层结构

2.2　自适应路由与容错路由

UGAL算法[19]在最小路径和VAL路径之间权衡，通过本地队列长度估计全局拥塞，缺乏比较准确的全局信息。部分工作采取各种策略估计全局拥塞用于路由决策。TPR[20]设置流量统计寄存器推断组内组间流量模式，但合理的偏置值依赖不同流量，不存在唯一偏置值满足所有流量需求。DGB算法[21]提出了基于梯度下降的路由算法来动态设置偏置值，但基于UGAL-L局部通道的延迟近似公式不准确。UGAL-LE[22]通过本地流量预测全局通道冲突，更准确估计Min路径延迟趋势，有利于选择合理路径。PAR算法[23]在源组中重新估计最小路由决策，部分流量下比UGAL性能有较大提升，但存在拥塞幻象问题。PAR_{PH}[24]提出基于历史窗口方法优化解决了幻象问题。这些工作集中在源路由，根据本地通道状态预测全局拥塞，效果比较有限。关于组内交互全局信息，Piggyback算法[23]采取显式拥塞机制在分组内交互全局链路信息；ECtN算法[25]利用计数器记录端口争用情况在组中交互全局端口争用信息。这些都采取源路由，难以感知到组间拥塞，并且未区分大量Non-Min路径跳步数影响，资源浪费导致网络性能损失。还有工作属于在线自适应路由，传输途中动态调整路径实现更精确路由，缓解了全局信息获取难度大问题。OFAR算法[18]在动态传输中就近检测拥塞选择Non-Min路径；RLM/OLM算法[26]提出两种自适应路由算法在线

动态选择路径，避免局部缓冲依赖并采取逃逸通道解决死锁，但它们也未区分Non-Min路径跳步数，并且最大跳步数可能增至8，优化效果有限。在考虑跳步数方面，T-UGAL算法[27]优化Non-Min路径平均跳步数提升网络性能，但采取离线流量模拟难以实时适应复杂流量需求。ACOR算法[28]在源节点区分不同跳步路径，但回传跨芯片全局通道信息和组间拥塞信息延迟较高，影响路由决策准确性。AMLR算法[29]对三种跳步数路径优先级选择，存在不同跳步路径分类不全问题，并且借助CNP拥塞报文反馈指导二次路由决策，存在拥塞信息获取不及时难题。Q-adaptive算法[30]提出了分布式强化学习方法训练Q表格倾向于较短路径，但更新维护Q表格参数损失带宽且成本过高，学习参数依赖于流量模式，学习收敛过程较长。

在网络高可靠方面，文献[31-32]采取有连接协议实现端端硬件重传容忍少量丢包，但重传报文难以规避故障，每次重传性能损失大。文献[33]提出嵌套多层拓扑容错路由算法，故障重传时查找路由表并选择冗余链路经中间代理避开故障链路，算法复杂度高且容错能力差。Immunet算法[34]在故障发生时切换至安全路由表紧急配置状态重构拓扑，不足在于算法开销大且仅支持torus拓扑和数百点规模。Immucube容错路由算法[35]利用动态可重构路由表支持故障场景下所有节点可达且性能损失小，但仅适应k元n立方体网络且未给出重构细节。文献[36]面向“胖树”拓扑提出动态折射路由机制，通过叶交换多次折射容忍k-1次故障，但只适合“胖树”拓扑，容错开销大性能下降明显。文献[37]改进Immunet算法适应蜻蜓拓扑，在组内和组间分别建立超立方体安全路径保证任意点可达，但切换超立方体拓扑的网络性能损失较大。文献[38-39]对树形网络故障集中进行容错路由计算并配置路由，64K网络节点规模下故障处理和恢复算法耗时约数百毫秒，耗时长难以实现上层透明。文献[40]中SHIELD技术在故障发生时采取快速链路故障恢复（FLFR）技术让本地路由器选择其它端口，并提出快速链路故障通知技术（FLFN）在FLFR无本地可用端口时通知邻居选择其它路径避开本地路由器。SHIELD未给出技术细节，主要限于“胖树”拓扑。

2.3 天河E级原型系统网络分层路由表

天河E级原型系统网络分布式分层路由表主要原理如图2所示，根据内部、组内和组间地址范围，将所有表项分为N、B、G三个区间。每个报文自带*DestID*目的标识，每个路由器自带*RouterID*地址。通过比较*DestID*和*RouterID*中*gid*和*bid*域信息，查找各路由表项。如果两者*gid*和*bid*域都不相等，以*DestID*中*gid*为索引查询G区间，表项指向其他分组；如果*gid*相等而*bid*域不等，以*DestID*中*bid*为索引查询B区间，表项指向组内路由器；如果*gid*和*bid*域信息均一致，以*DestID*中*nid*为索引查询N区间，指向当前路由器内部节点。路由表table A和table B分别记录Min和Non-Min路径。table A条目少，采取编码方式记录本地端口，table B条目多，采取位图方式记录。

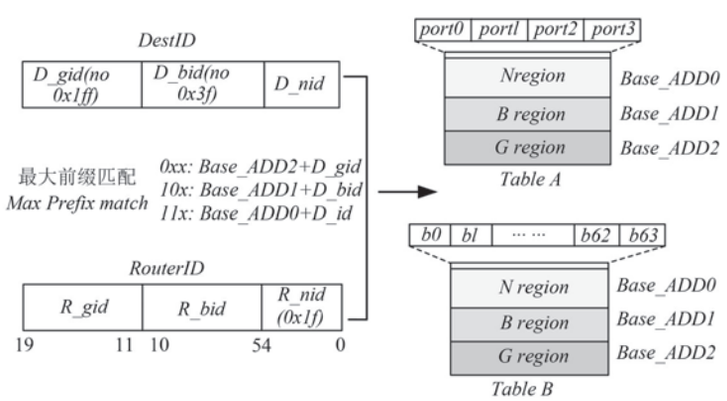


图2 天河E级原型系统网络分层路由表寻址

天河E级原型系统采取树型拓扑支持数万点规模，本文主要面向后E级超大规模超智算网络考虑蜻蜓拓扑，在有限成本功耗约束下大幅改善系统可扩展性。但如何充分发挥蜻蜓拓扑性能优势并实现高效且稳定传输非常重要。因此，本文将提出一种分布式路由框架技术解决高性能和高可靠难题，具体第3节提出在线自适应路由技术提升性能，第4节提出动态路由自愈及恢复技术增强通信鲁棒性。

3　分布式在线自适应路由技术

本节提出分布式在线自适应路由(DOAR)算法，快速收集本地和全局通道拥塞信息，缓解拥塞同时区分跳步数路径，优先少跳步路径传输。然后介绍路由计算部件和可重构配置方法。

3.1　基于通道检测的在线自适应路由算法

任意拓扑的理想路由算法应该在避免拥塞和流量均衡基础上降低跳步数。假设网络中通信带宽和平均利用率分别为T和U，节点数和平均跳步数分别为N和H，节点带宽正比于$(T\times U)/(N\times H)$。本节思路主要是适应复杂流量挖掘Non-Min路径多样性避免拥塞，提高平均通道利用率U；同时优先选择较短路径减少跳步H，优化延迟提高吞吐量。

在蜻蜓拓扑中，源节点$C_s=<G_s, R_s>$向目标节点$C_d=<G_d, R_d>$传输时，源分组选择不同全局端口，中间组选择不同本地端口，分别对应不同跳步数。图3显示源分组4种全局端口选择策略，实箭头分别代表路径，虚箭头代表全局通道状态。本文拓扑中任意分组第G_i个全局端口连接第G_i组。首先，情形①中Min路径经路由器R_m直达目标分组G_d，跳步最少，路径为L-G-L。情形④中经过组内R_{n2}和中间组G_i到达G_d，G_i内含一次转发，跳步数最多，路径为LG-L-GL。其次，在源分组和目标组内分别考虑如何减少组内跳。情形②中经与源路由器R_s直连的中间组G_i转向G_d，G_i内含一次局部转发。源组内省略一次组内跳，路径为G-L-GL。情形③中当$G_i\in\{hR_d, \cdots, hR_d+h-1\}$时，目标路由器$R_d$上全局端口直连中间组$G_i$，在目标组内省略一次组内跳，路径为LG-L-G。假设情形②和③同时发生($R_s=R_d$)，那么在源和目标组内各省略一次组内跳，路径为G-L-G。另外，针对情形②③④，如果在中间组G_i组内出现折射，那么跳步数加1，路径分别为G-LL-GL、LG-LL-G和LG-LL-GL。

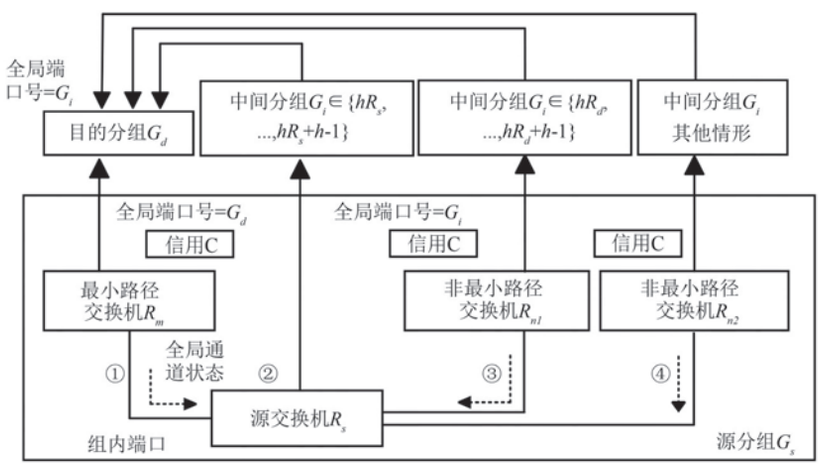


图3　蜻蜓拓扑源分组多路径选择策略

算法1.　DOAR自适应路由算法伪代码

输入：源$C_s=<R_s, G_s>$、目标$C_d=<R_d, G_d>$、当前$C_i=<R_i, G_i>$

输出：当前输出端口

```
1. if ((Ci=Cs)&(Gi≠Gd)) {    // 源路由器组间路由，查询全局端口
2.     GA=Sel_min (tableA, Cd);    // 情形①
3.     G2= Sel_nm (tableB, Cd)∩{hRs, …, hRs +h−1};
4.     G3= Sel_nm (tableB, Cd)∩{hRd, …, hRd +h−1};
5.     GB = G2∩G3;                 // 情形②③
6.     GC = G2∪G3 − GB;            // 情形②或者情形③
7.     GD = Sel_nm (tableB, Cd)−G2∪G3;   // 情形④
8.     for (g∈GA) {l= Lport(tableC, g);    // 跳步数3
9.         if(~CongG(g) &~CongL(l)) {route with {l}; exit;} //min
10.    for (g∈GB) {l= Lport (tableC, g);    // 跳步数3
11.        if(~CongG(g)&~CongL(l)) {route with {l}; exit;} //nmin
12.    for (g∈GC) {l= Lport (tableC, g);    // 跳步数4
13.        if(~CongG(g)&~CongL(l)) {route with {l}; exit;} //nmin
14.    for (g∈GD) {l= Lport (tableC, g);    // 跳步数5
15.        if(~CongG(g)&~CongL(l)) {route with {l}; exit;} //nmin
16.    route with {l} (l∈GA∪GB∪GC∪GD) randomly;}
17. else {                          // 其它情况，查询本地端口
```

18. LE = **Sel_min** (tableA, C_d);
19. LF = **Sel_nm** (tableB, C_d);
20. **for** (l∈LE) {if (∼**CongL**(l)) {route with {l}; **exit**; }} //min
21. **for** (l∈LF) {if (∼**CongL**(l)) {route with {l}; **exit**; }} //nmin
22. route with {l} (l∈LE∪LF) randomly; } //min & nmin

本节参考上述策略提出DOAR自适应路由算法如算法1所示，g表示全局端口，l表示当前路由器本地端口，算法通过在源路由器和中间分组入口分别选择合适的全局端口和本地端口，避免拥塞减少路径跳步。首先在源路由器处，第2～7行针对情形①查找Min路由表A产生集合GA，对应最小路径的全局端口，路径跳步为3。针对情形②，查找Non-Min路由表B。源路由器R_s上全局端口为{hR_s，…，hR_s+h-1}，通过它们直达G_i，能够减少源分组内组内跳。针对情形③，目的路由器R_d全局端口为{hR_d，…，hR_d+h-1}，经源分组的这些全局端口和中间分组到达R_d，能够减少目标分组组内跳。如果情形②③中全局端口重叠，跳步数进一步减少为3。针对其它情形④，查找表B得到其他Non-Min路径的跳步数均为5。算法1在R_s上需要进一步判断各路径全局端口和本地端口状态。表A和表B分别记录Min路径和Non-Min路径可用的全局端口，表C记录了每个全局端口关联的本地端口号。第8～16行通过g查找表C获取本地端口l，在g没有全局拥塞且l没有本地拥塞时优先选择跳步最少路径。其次，在其它情况下第18～22行查找表A和表B产生集合LE和LF，分别包含直达路径和折射路径对应本地端口，在本地端口没有拥塞时优先选择直达路径。例如，中间分组入口或源路由器组内路由时，在表A中端口拥塞时选择表B中本地端口折射，路径长度加1。

图4显示了DOAR算法效果图，G_s=1组c节点正向G_d=4组e节点传输，其中p=2，a=3，h=2。如果G_s选择全局端口4可直达G_d目标组。源和目标组分别含一次组内跳，路径为L-G-L。如果选择全局端口2或3经G_i=2或G_i=3中间组转发，适应情形②，减少源组组内跳，路径为G-L-GL。如果选择全局端口5经G_i=5中间组转发，适应情形③，减少目标组内跳，路径为LG-L-G。如果选择全局端口0经G_i=0中间组转发，适应于情形④，路径为LG-L-GL。如果目标路由器改为G_d=4内d节点，G_s选择全局端口2经G_i=2中间组转发，适应情形②③，减少源分组和目标组内跳，路径为G-L-G。

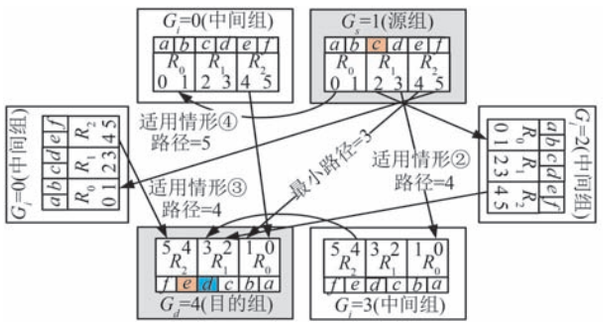


图4 DOAR算法效果示意图

路由器依靠信用计数记录下游端口缓冲情况判断拥塞。信用计数和拥塞判断均在交换部件出口处，减少拥塞判断延迟；产生的全局端口拥塞位图搭载在物理编码层广播给组内所有路由器，减少全局拥塞的传播延迟。针对本地或全局端口，交换部件出口分别配置本地拥塞和全局拥塞阈值。针对本地端口，路由器中函数CongL将信用和本地拥塞阈值比较，判断是否本地端口拥塞。针对全局端口，将它和全局拥塞阈值比较，并采取历史窗口方法[24]判定是否全局端口拥塞。由于全局端口采取光纤传输，实际全局拥塞阈值通常大于本地拥塞阈值。紧接着，全局端口状态采取快速通道广播给组内邻居。为节省开销，路由器只产生自己所属h个全局端口的拥塞状态(以64口路由器为例，h=16)，采取位图嵌入至报文广播给组内邻居。以天河报文格式为例，10个256位切片报文搭载16位位图，额外负载约6‰；由于有效利用尾切片部分保留位，几乎无性能损失。广播间隔等于最大报文大小。最后，路由器将接收组内广播报文并形成全局通道拥塞位图。如图5(a)所示，由于提前配置了各邻居编号R_bid，本地端口收集邻居h位拥塞位图后根据$R_bid\times h$计算出全局端口偏移，快速形成全组全局端口拥塞位图。为加速h位位图传递，路由器采取交换部件硬连线收集及物理编码层搭载方式。物理编码层位于链路层下，穿透延迟t_{pcs}≈30个周期，明显小于信用返回延迟t_{crt}，能快速获取全局拥塞信息。路由器从多个组内邻居收到h位位图后，分时复用加载形成512位全局通道拥塞位图，采取环路广播给各端口路由计算部件。

3.2 路由计算部件与可重构配置方法

DOAR路由计算部件结构如图6所示，根据目

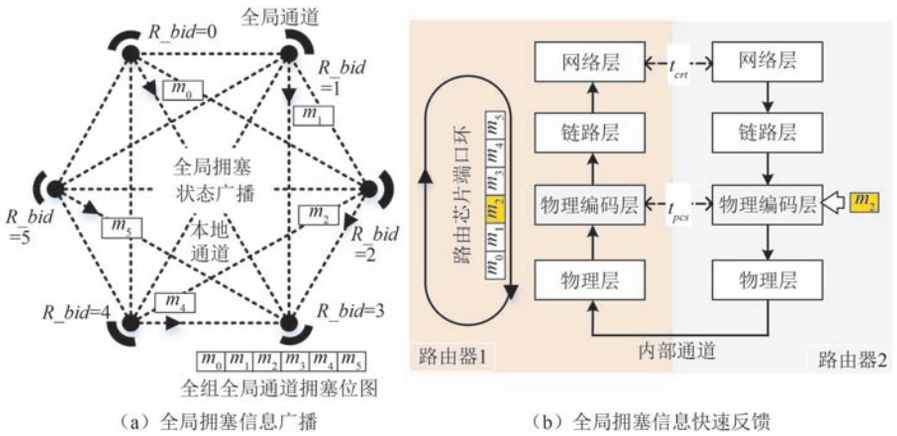


图5　全局通道拥塞位图快速形成机制

标节点*DestID*和当前路由器*RouterID*查找两表，表A和B在天河E级原型系统基础上更新。首先，表A为最小路由表，表项缺省配置为64位位图LP，lp_j（j=0，…，63）对应64个本地端口，分别表示是否允许采用第j本地端口传输。但在内部端口G区，表A采用全局配置，表项为本地端口lpc_0及16个全局端口位图GP，用于计算组间路由Min路径的全局端口。其次，表B为非最小路由表，只包含B和G区，表项缺省配置也采取位图LP。在内部端口G区间也采取全局配置，表项为8个本地端口lpc_k（k=1，…，8），用于计算组间路由Non-Min路径的全局端口。由于每个本地端口对应相邻路由器16个全局端口，最多可支持128个全局端口。最后，表C为端口映射表，32个表项对应32个相邻路由器，每个表项含6个路由项分别记录6个本地端口。表C主要用于计算全组512个全局端口对应的本地拥塞情况。上述表A和B表项大小为8 B，N、B、G区的表项数分别为16、32和512；表C表项大小为6 B，数量为32。每个端口路由表体积约9 KB。

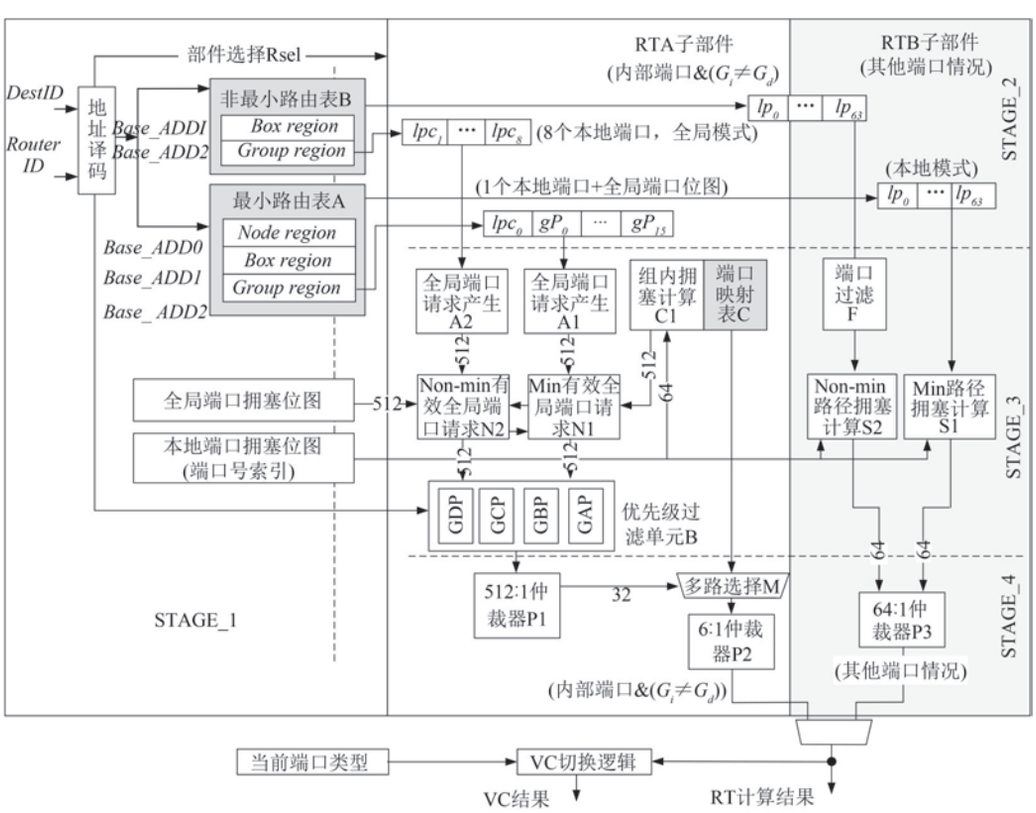
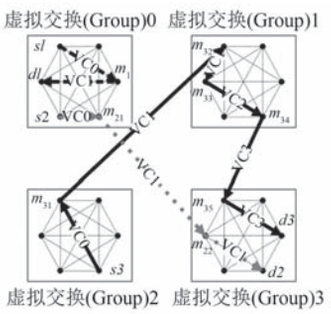


图6　DOAR自适应路由计算部件结构图

图6中计算部件包含两个计算单元，采取4级流水线结构。两个计算单元通过第3级和第4级流水产生仲裁结果，路由计算吞吐量高。第一，如果当前属于内部端口且目标指向其他分组，RTA单元查找G区路由表项，结合全局端口和本地端口拥塞选择路径。在流水线第3级A1和A2单元读取表A和表B路由表项，扩展为512位全局端口请求发送给N1和N2单元。C1单元根据本地端口拥塞位图计算512个全局端口的本地端口拥塞情况发给N1和N2单元。N1单元参考全局端口拥塞位图和本地端口拥塞，将没有全局和本地拥塞的Min路径全局端口请求发给B单元。N2将没有拥塞的Non-Min路径全局端口请求也发给B单元。B单元接收地址译码结果，参考算法1第2～15行描述，设置GAP，GBP，GCP和GDP模块，过滤跳步数3、3、4、5的全局端口请求给P1单元。在流水线第4级，P1单元优先选择跳步低的全局端口请求，选择唯一获胜者（32位位图指明具体组内路由器）给M单元。M单元和P2单元从表C中选择唯一获胜本地端口输出。第二，在其他情况下RTB单元工作。流水线第3级S1单元接收路由表项LP位图和本地端口拥塞位图，将没有拥塞的Min路径本地端口请求给P3；S2单元将没有拥塞的Non-Min路径本地端口请求给P3。表B中G区路由表项LP位图可能同时含组内端口和全局端口，端口过滤单元F判断属于中间分组折射时过滤掉全局端口，否则过滤掉组内端口。流水线第4级中P3单元优先S1请求，选择唯一获胜本地端口输出。如图所示，除了在内部端口读取表项G区间路由计算，其它情况下都采取RTB结果。最后根据路由结果产生通道切换信息。由于本地端口等级低于全局端口，路由计算比较当前和输出端口等级相等或升序，虚通道序号加1。虚通道数量为4可避免全局死锁。

		配置表格
内部端口	N区	表A
	B区	表A+表B
	G区	表A+表B
组内端口	N区	表A
	B区	表A
	G区	表A/表B
全局端口	N区	表A
	B区	表A
	G区	表A+表B

（a） 路由配置方法



（b） 蜻蜓拓扑路由示意图

目标	场景	跳步	VC切换
组内路由器	直达路径	L	0
	折射路径	L-L	1
组间路由器	最小路径	L-G-L	1
	非最小路径2	G-L-GL	1
		LG-GL	2
		G-LL-GL	2
		LG-L-GL	2
		LG-LL-GL	3

（c） 自适应路由跳步数和通道切换数

图7 蜻蜓拓扑路由配置及通信效果

根据上述结构，蜻蜓拓扑典型路由配置如图7(a)，本地端口再分为内部端口和组内端口。第一，针对内部端口，N区指向内部节点，仅配置表A。B区组内路由配置表A指向组内目标路由器，配置表B指向组内其他路由器折射。G区组间路由配置表A，最小路径指向与目标分组直连的路由器；配置表B，非最小路径指向其它路由器。第二，针对组内端口，N区指向内部节点，B区指向组内目标路由器，均仅配置表A。G区在不同表项配置表A或表B，相同表项A和B不同时配置。针对当前路由器直连分组的表项只配置表A，针对其它分组只配置表B，均采取64位位图配置。第三，针对全局端口，N区和B区与组内端口类似。G区意味着当前属于中间分组，配置表A指向中间分组出口路由器，配置表B指向组内其他路由器进行折射。

图7(b)举例说明查表路由情况。黑色虚线显示组内$s1$至目标$d1$。在$s1$内部端口查找B区，表A中所有端口发生拥塞时在表B中选择端口折射；到达m_1组内端口后，仅有表A中B区的表项指向$d1$。路径为L-L。其次，灰色点线显示分组间$s2$至$d2$采取最小路径。在$s2$内部端口查找G区，结合全局端口和本地端口拥塞情况从表A中选择与目标分组直连路由器m_{21}；在m_{21}组内端口查找G区，仅有表A表项指向m_{22}；在m_{22}全局端口查找B区，仅有表A中表项指向$d2$。路径为L-G-L。最后，黑色实线显示分组间$s3$至$d3$采取非最小路径。在$s3$内部端口查找G区，表A中路径上出现本地或全局端口拥塞，选择表B中表项指向组内其他路由器m_{31}；由于m_{31}与目标分组不相连，在m_{31}组内端口查找G区，表B中表项过滤后指向m_{32}；在m_{32}全局端口查找G区，表A中端口出现拥塞，选择表B中端口折射至m_{33}；在m_{33}组内端口查找G区，表B中表项过滤后指向m_{34}；由于m_{34}与目标分组相连，在m_{34}组内端口查找G区，仅有表A中表项指向m_{35}；在m_{35}全局端口查找B区，

仅有表A中表项指向*d3*。路径为LG-LL-GL。图7(c)显示了各种场景下跳步数和通道切换次数。在m_{32}全局端口，如果选中表A直接指向m_{34}的路径没有拥塞，跳步减至LG-L-GL；在*s3*内部端口，如果直接从*s3*全局端口离开当前组，跳步减至G-L-GL。

4　动态路由自愈及恢复技术

DOAR算法提供了较多路径选择，但任意路径上的故障都可能导致端到端重传。本节提出路由动态自愈与恢复技术。产生故障诊断报文，失效表A和表B特定路由项，避开故障保证稳定通信，保留更多路径优化性能；产生恢复报文还原路由项，快速恢复所有路径。

4.1　基于主动报告的故障诊断及恢复算法

本节在故障发生或恢复时主动产生诊断或恢复报文，在表A和表B中匹配表项，作废或还原特定路由项。针对本地故障或故障诊断报文，故障诊断过程需要对本地每个端口操作。对于本地故障需要遍历所有路由表项，对于故障诊断报文只要定位故障区域的关联表项。当表项中所有路由项失效时将产生新的故障诊断报文告知邻居路由器。故障诊断流程如算法2所示，函数Travel_FD处理本地端口故障，遍历每个表项；函数Match_FD处理诊断报文，仅匹配唯一表项。

首先，本地*Fp*端口故障后调用函数Travel_FD遍历所有表项。第1～3行根据*Fp*端口、故障地址*et_addr*信息，访问每个表项*et*，调用函数FD_act进行处理：(1)*et*中除*Fp*外还存在其他有效路由项，意味着还可从其它端口到达*et_addr*区间，此时只失效含*Fp*的路由项；(2)*et*中*Fp*为仅剩唯一有效路由项，意味着从当前端口无法到达*et_addr*区间。除失效路由项外，还将产生新的FD报文告知上游该区间不可达。其次，对于FD报文调用Match_FD函数处理。第6行函数Aim将*ip_addr*故障地址和路由器*RouterID*比较，寻找覆盖故障地址的唯一表项为关联表项；第7行函数FD_act对关联表项作上述类似处置。从系统上看，*Fp*端口故障后，当前路由器在每个端口先调用Travel_FD产生FD报文给各邻居；各邻居再在每个端口调用Match_FD产生FD报文，直至不产生FD报文为止。该过程通过函数Aim仅选中覆盖故障区域的关联表项，不影响其他表项，可将合法路径最大化。

算法2.　故障诊断算法伪代码

输入：(1)故障端口号*Fp*

(2)故障诊断报文*FD[<ip_g, ip_b, ip_n>, ip_prefix]*

输出：1)故障诊断报文*FD[<op_g, op_b, op_n>, op_prefix]*

```
1.  Procedure Travel_FD (Fp) {
2.      for each entry et in table {
3.          et_addr = [<et_g, et_b, et_n>, et_prefix];
4.          FD_act (Fp, et_addr , et) ; }}
5.  Procedure Match_FD (Fp , [<ip_g, ip_b, ip_n>,
    ip_prefix]) {
6.      ip_addr = [<ip_g, ip_b, ip_n>,ip_prefix];
7.      m_et = Aim (ip_addr) ;
8.      FD_act (Fp , ip_addr, m_et) ; }
9.  Procedure FD_act(Fp, [<op_g,op_b,op_n >, op_
    prefix], et){
10.      if Fp does not exist in et of Table A+B {Do
         NOTHING};
11.     else if et of Table A+B exists other valid ports
        besides Fp{
12.         Only invalidate route option of Fp;
13.     else if Fp is the only valid route in et of Table
        A+B {
14.         Invalidate route option of Fp;
15.         accessibility[et] = 0;
16.         Propagate FD[<op_g, op_b, op_n >, op_
            prefix]; }}
17.  Procedure Aim ([<ip_g, ip_b, ip_n>,ip_prefix]) {
18.     if (ip_prefix == 1) m_et = ip_g +Base_
        ADD2;
19.     else if (ip_prefix == 2) {
20.         if (ip_g != R_gid)   m_et = ip_g
            +Base_ADD2;
21.         else m_et = ip_b +Base_ADD1; }
22.     else if (ip_prefix == 3)  {
23.         if (ip_g != R_gid)   m_et = ip_g
            +Base_ADD2;
24.         else if (ip_b != R_bid) m_et = ip_b
            +Base_ADD1;
25.         else m_et = ip_n +Base_ADD0;}
26.         return m_et;}
```

故障恢复指故障维护后路由器自动还原表项恢复初始状态。如算法3所示，监测本地故障恢复后产生初始FR恢复报文；路由器接收初始FR报文或上游FR报文后，在每个端口调用函数FR_act还原各表项，在所有表项变为有效后激活端口产生FR

报文。第3～10行端口 Cp 调用FR_act遍历表项：(1)如果表项 et 中已有其他有效路由项，那么仅还原含 Fp 的路由项；(2)如果表项 et 中没有其他有效路由项，除还原含 Fp 路由项之外，还需打开 *check_tag* 检查。第10～12行发现所有表项都有效时激活 Cp 端口并产生FR报文给上游。如果遍历过程中没打开 *check_tag*，那么表示所有表项没变化或所有表项均已有效，此时无需产生FR报文。

算法3. 故障恢复算法伪代码

输入：本地故障端口 Fp，上游故障恢复报文 $FR[Fp]$

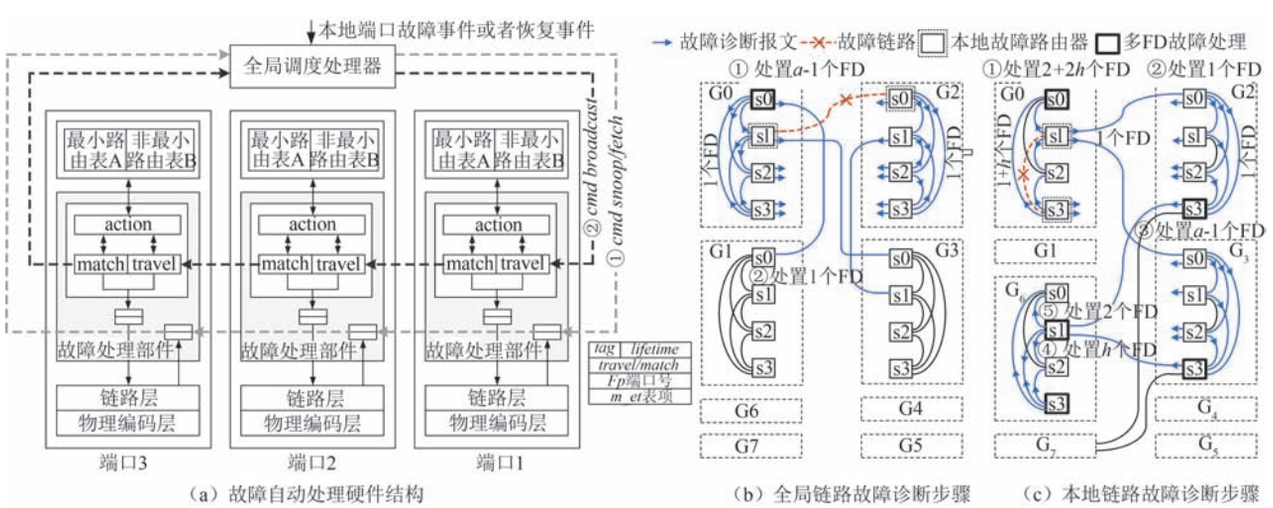
输出：故障恢复报文 $FR[C]$

```
1.  Procedure FR_src (Fp) {Propagate FR[Fp]; }
2.  Procedure FR_act (Fp, C) {
3.      for each entry et in Table A+B of C {
4.          check_tag = 0;
5.          if Fp does not exist in et {Do NOTHING}
6.          else if et exist other valid ports besides Fp
7.                  Validate route option of Fp;
8.          else { Validate route option of Fp;
9.                  accessibility[et] = 1;
10.                 check_tag = 1; }}
11.     if (check_tag == 1) {
12.         if (||accessibility == 0) {check_tag = 0;}}
13.     if (check_tag == 1) Propagate FR[C] ;}
```

4.2 故障自动处理部件硬件结构

故障处理硬件结构如图8(a)所示，路由器设置了全局调度处理器单元，各端口设置了故障处理部件。调度处理单元主要负责从各端口读取具有较高优先级的FD和FR报文，并监听记录各端口故障事件或恢复事件。调度处理器单元工作主要有两个步骤。步骤①查询各输入端口诊断或恢复报文，以循环优先级从不同端口输入缓冲中读取唯一报文，同时监听故障事件或恢复事件；步骤②解析报文或本地事件，广播诊断或恢复命令给所有端口故障处理部件。调度处理单元对路由器其它功能没有干扰且每个任意时刻仅调度唯一命令，主要包含 *tag* 遍历标识、Fp 端口及关联表项等信息。针对故障诊断命令，当 *tag* 有效时travel单元执行Travel_FD函数遍历所有路由表项；当 *tag* 无效时match单元执行Match_FD函数只对关联表项操作。Travel和match单元每次通过action单元对单个路由表项具体操作，产生的FD报文发给输出缓冲。action单元对表A和表B的读写操作在路由计算访问路由表间隔期间完成，因此不会对路由器数据通路产生任何影响。针对故障恢复命令，travel单元根据 Fp 端口遍历路由表，试图激活每个路由表项，当所有表项变有效时产生FR报文输出。

图8(a)硬件结构有较高执行效率。首先针对诊断命令，多端口travel/match单元并行工作。假设调度处理单元取回FD报文需要 t_1 个周期，广播诊断命令需要 t_2 个周期，action单元执行FD_act函数耗时两个周期，那么调度处理单元诊断本地故障端口耗时 $t_{ld}=t_1+t_2+2\times et\_num$ 个周期。诊断邻居FD报文只操作关联表项，耗时约 $t_{gd}=t_1+t_2+2$ 个周期，其中 *et_num* 表示路由表项数量。其次对于恢复命令，多个端口travel单元并行执行FR_act。由于两次遍历每个表项耗时2和1个周期，单个FR报文处理耗时 $t_r=t_1+t_2+3\times et\_num$ 个周期。以1.0 G主频和560条表项为例，路由器处理本地故障和FD诊断报文分别约1.3 μs和100 ns，处理FR恢复报文约2 μs。



(a) 故障自动处理硬件结构　(b) 全局链路故障诊断步骤　(c) 本地链路故障诊断步骤

图8 故障自动处理硬件结构及诊断开销

整个网络故障诊断与恢复效率至关重要，尤其是故障诊断中报文丢失容易引起端到端重传。根据上述算法，单个本地和全局链路故障诊断耗时分别为 $t_{ld}+a\times t_{gd}$ 和 $t_{ld}+(a+h+2)\times t_{gd}$ 个周期。以 $dfly(2,4,2,8)$ 举例，如果图8(b)中G0与G2间全局链路故障，G0s1(G0组s1路由器)处理本地故障耗时 t_{ld}，给每个组内端口发送1个FD报文；G0s2和G0s3诊断后发送FD给G0s0，G0s0共处理3个报文，然后发送FD给G1s0，告知G0无法作为中间分组。G1s0处理FD完毕后整个诊断过程结束，耗时 $t_{ld}+4\times t_{gd}$ 个周期。其次，如果图8(c)中G0s1和G0s3间发生本地故障，G0s1/s3向每条本地链路发送3个FD，向每条全局链路发送1个FD。在组内，G0s0从G0s1和G0s3分别接收3个报文，处理耗时 $6\times t_{gd}$ 个周期。在组间，G2s0接收FD经诊断后向组内各端口发送FD，G2s1和G2s2诊断后传递FD给G2s3。G2s3收集3个FD报文，处置完毕后向G6发送FD报文，通知G2无法作为中间分组向G0通信。G2组内诊断耗时 $4\times t_{gd}$ 个周期。G3也向G6发送FD报文。G6s1从G2和G3接收两个FD，诊断后发送FD给组内其它节点，告知无法再经G6s1到达G0。同理，G0s3本地故障也会导致G6s3向组内其它节点发送FD。G6s0共需处理2个FD后诊断结束。G6组诊断共需 $4\times t_{gd}$ 个周期，整个诊断耗时 $t_{ld}+8\times t_{gd}$ 个周期。

4.3　蜻蜓拓扑故障诊断及恢复效果

本节假设G2s0与G2s2间链路故障，分析故障诊断效果和路径变化情况，如图9所示。以s0的ES端口故障为例，故障诊断组内处理分三种情形。首先，G2作为源分组的效果如图9(a)。s0遍历端口时失效含ES端口的路由项，具体包括：失效N端口B区间的路由项(黑方格)，表明s0无法直达s2，s0经s2无法到达s1和s3；失效N端口G区间路由项，表明s0无法再经s2到其他组；失效E和S端口B区间表项中唯一路由项(空方格)，并发送FD报文给s1和s3，告知无法再经s0到达s2。G2作为源分组通信的可选路径减少，但不影响目的可达性。其次，G2作为中间组的效果如图9(b)。s0失效W端口G区间部分路由项，表明经s0无法直达s2，需折射才能到s2离开G2；失效E和S端口G区间表项中唯一路由项，发送FD报文给s1和s3，s1和s3分别失效E和W端口G区间路由项，表明s1和s3报文无法再经s0到s2。G2中间组不影响交换功能但影响交换吞吐量。最后，G2作为目标组的效果如图9(c)。s0失效W端口B区间部分表项中唯一路由项，发送FD报文给G0s1和G1s1，表明报文经过G0s1/G1s1再无法到达目的G2。G0s0/G1s1接收FD报文后继续在组间传播，G0/G1将无法作为中间分组达到G2，最终影响其他分组到G2的路径数量。

接下来分析故障诊断的组间处理过程。G2s0发送FD给G0、G1，G2s2发送FD给G4、G5，通知G2已不可达。图9(d)以G0处理FD为例，G1、G4、G5类似。G0s1收到FD后匹配关联表项，在N端口失效含E0端口的路由项，表示本地节点无法直到G2；同时在其它端口失效含E0的唯一路由项，产生FD给s0、s2和s3。s0、s2和s3失效组内端口相关路由项并发送FD；全局端口处理3个FD报文过程中失效关联表项所有路由项，然后发送FD给G1、G3～G7，通知再无法经G0到达G2。最后，G6接收来自G0、G1、G4、G5的FD报文，如图9(e)所示，G3

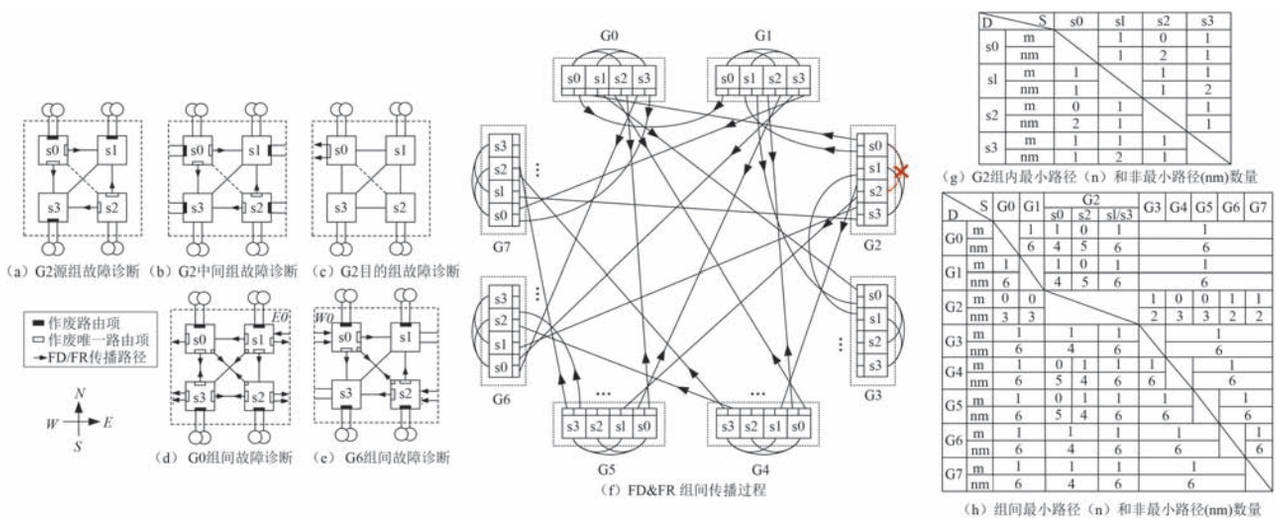


图9　蜻蜓拓扑链路故障诊断效果图

和G7类似。G6s0接收G0/G1的FD报文，在3个组内端口失效关联表项中路由项，并给其他组内节点发送FD，表示经G0、G1无法到达G2组。同理经G4、G5也无法到达G2组。

上述故障诊断后全局连通性保持但路径减少。G2组内通信效果如图9(g)。s1与s3间路径不受影响保持为3，s0与s2间损失最小路径，其它节点之间损失一条非最小路径。组间通信效果如图9(h)。如第4列G2作为源分组，G2s0到G4、G5及G2s2到G0、G1分别损失1条非最小路径和1条最短路径；G2s0和G2s2到达其它组各损失2条非最小路径。如第4行G2作为目的组，其它组至G2路径损失至3条，G0、G1和G4、G5不可直达G2。

上述场景故障恢复比较直观，从故障链路产生初始FR开始，路由器在每个端口所有表项变为有效时(空方格处)产生FR报文传播。G2s0产生FR报文后在各端口还原含ES端口的路由项。首先G2s0激活E和S端口，发送FR报文给s1和s3。s1和s3还原含W和N端口的路由项，由于各端口表项原来有效，将停止FR在G2内传播。第二，G2s0激活W端口，发送FR至G0。G0s1相继激活3个组内端口发送FR给s0、s2、s3。G0s0、G0s2、G0s3先后激活组内和全局端口后，发送FR给G1、G3～G7，表示经G0可达G2。最后，G6s0接收FR报文在各端口还原含W0端口的路由项，激活3个组内端口，发送FR给G6s1～G6s3。由于G6s1～G6s3端口所有表项原本有效，停止FR传播。同理，G6s2也接收FR报文，激活3个组内端口并还原路由项，直到G6路由表恢复初始状态。

5 实验结果

5.1 仿真实验环境

本节基于Booksim2.0[41]构建周期精确模拟环境，配置输入队列交换模型和2.0倍路由器内部加速度。模拟器通过伯努利过程注入包，采样周期为10 K，注入标记报文经30 K个周期流量预热。当平均包延迟超500个周期时网络饱和，饱和前稳定注入速率为饱和吞吐量。环境配置如表1，蜻蜓拓扑配置p=6，a=12，h=6，g=73，简写$dfly$(6，12，6，73)，规模约5 K节点。为充分比较，除了MIN和VAL算法外，还添加UGAL_L、UGAL_G、TPR、PAR_{PH}、UGAL_LE算法。MIN采取2个VC，PAR_{PH}采取5个VC，其他算法采取4个VC保证无死锁。DOAR算法设置本地和全局端口拥塞阈值分别为10和30；本地和全局通道延迟分别为10和100个周期；本地和全局端口缓冲区分别容纳32和256个报文。

表1 周期精度互连网络仿真环境配置

参数	配置
路由算法	MIN、VAL、UGAL_L、UGAL_G、TPR、PAR_{PH}、UGAL_LE、DOAR
虚通道VC数量	5 VC：PAR_{PH} 4 VC：UGAL_L、UGAL_G、TPR、UGAL_LE、DOAR、VAL 2 VC：MIN
缓冲区大小	全局端口缓冲区：256pkts 本地端口缓冲区：32pkts
链路延迟	全局链路：100cycles 本地链路：10cycles
拥塞阈值	本地端口拥塞阈值：10 全局端口拥塞阈值：30
路由器内部加速	2.0倍

本节先采用四种不同合成流量模式，包括均匀随机流量(UR)、对抗移位流量(ADV+i)、随机排列流量(RP)和混合流量(Mixed)。在均匀随机流量下，源节点向所有目的节点发送包概率相等。在对抗流量下，组n中所有节点将所有流量发给组($n+i \bmod g$)。在随机排列流量模式中，每个节点随机分配唯一的目标节点。混合流量按粒度区分路由器(R)和节点(N)级，R表示路由器上节点采取相同流量模式，N表示节点采取不同混合流量。针对路由器混合流量，又区分UR和ADV混合、UR和RP混合，分别用R_MIXED_UA(UR%，ADV%)和R_MIXED_UR(UR%，RP%)表示，其中UR%、ADV%、RP%表示三种流量各占百分比。针对节点混合流量，以N_MIXED_UA(UR%，ADV%)和N_MIXED_UR(UR%，RP%)表示。

为评估实际应用通信性能，还选取了六种典型实际应用负载[42]，覆盖科学计算与智能计算中多种通信模式：LU、FFT3D、Halo3D、Stencil5D、DL和LULESH。LU具有扫描通信特征，沿正方形对角采用波阵面方式逐级扫描，每个节点从上游收到消息然后发给下游，通信延迟较高；FFT3D采取多步环形交换方式实现大量Alltoall操作；Halo3D与Stencil5D体现模板计算通信特征，每个进程在所有维度正反向与最近进程通信，具有典型一对多通信模式且吞吐量较高；DL代表分布式深度学习中高

负载全归约应用，包含大量Allreduce操作；LULESH代表混合模式，结合MPI非阻塞通信与集合通信，包含大量三维模板计算和三维扫描通信。所有负载流量由SST[43]生成并注入至模拟器，可反映实际通信行为。

5.2　典型流量网络性能分析

本节给出蜻蜓拓扑下与MIN、VAL、UGAL_L、UGAL_G、PAR_{PH}、TPR、UGAL_LE算法的性能对比分析。

5.2.1　标准合成流量模式

本节针对各种算法在接近各自饱和注入率时对数据包进行采样，统计不同跳步路径百分比。三种标准合成流量模式结果如图10(a)(b)(c)。在UR模式下，各种算法约80%报文选择Min路径，集中在3跳。对于大跳步路径，UGAL_LE算法6和5跳路径各占7.7%和2.1%；PAR_{PH}算法6跳路径约占1.7%；TPR和DOAR基本没有6和5跳路径。在ADV模式下，TRP算法5和6跳路径分别约15.6%和73.7%；UGAL_LE算法5和6跳路径分别约15.0%和73.6%；PAR_{PH}算法由于二次路径选择，5和6跳路径达15.1%和74.5%，大量6跳路径消耗较多资源造成网络迅速饱和。比较而言，DOAR算法主要包含3和4跳路径，分别约占76.0%和14.9%。在RP模式下，TPR、UGAL_LE和PAR_{PH}趋于饱和时5跳路径各占约18.9%、9.2%和7.9%，6跳路径各占66.9%、33.2%和35.2%，而DOAR中5和6跳路径分别仅占比13.7%和0.7%。

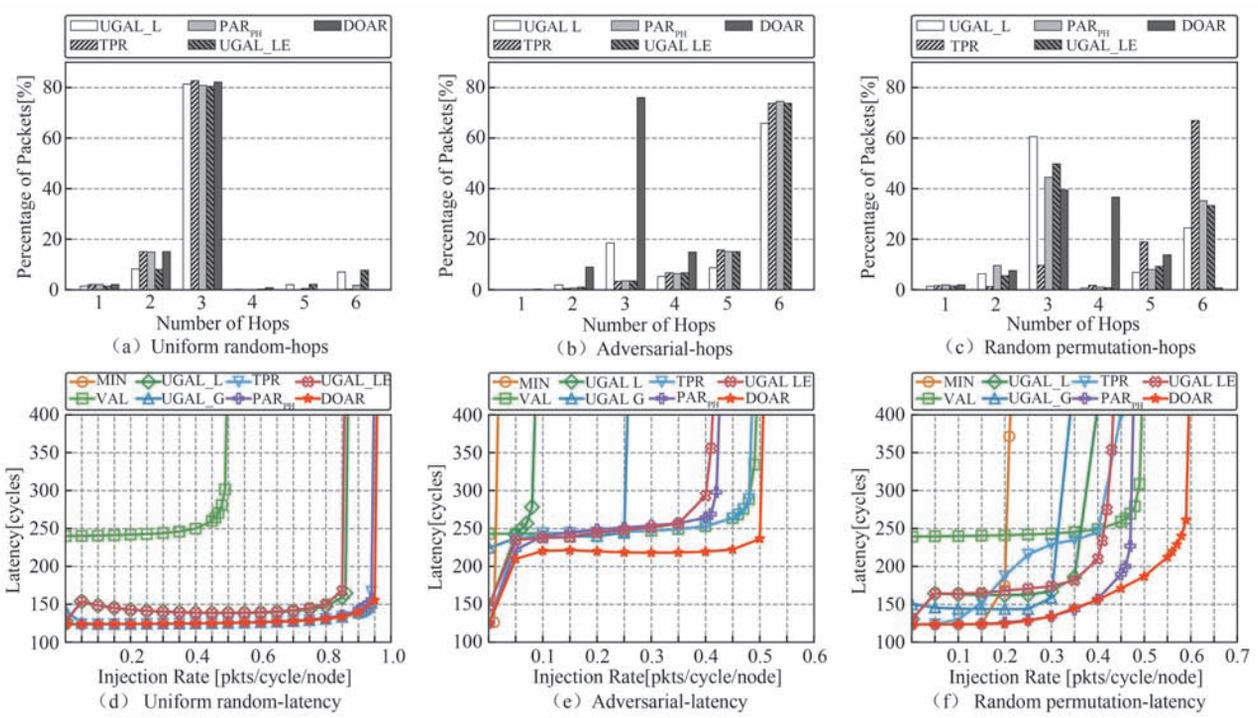


图10　三种标准合成流量下的网络性能评估

图10(d)(e)(f)显示UR、ADV、RP三种流量下不同算法性能，X和Y轴分别表示注入率和包延迟。图10(d)中MIN算法代表最佳性能，UGAL_G拥有全局拥塞信息，属于UGAL类算法理想情况。DOAR在UR流量下全局通道缓冲利用率相当，大多数报文采取Min路径跳步数为3，曲线性能基本接近MIN，饱和吞吐量约0.95。PAR_{PH}、TPR、UGAL_LE吞吐量分别为0.94、0.94和0.85，DOAR相比它们分别提高约1.1%、1.1%和11.8%。

图10(e)显示ADV流量下性能。DOAR在低负载下大多选择MIN路径。MIN算法全采取MIN路径，存在分组间直连带宽瓶颈问题。随着流量增加，其他算法选择Non-Min路径导致报文延迟开始便增加。DOAR算法在避免拥塞同时优先选择更少跳步路径，在相同注入率下具有延迟小优势。在注入率0.40时，DOAR延迟约219.3周期，VAL、PAR_{PH}、TPR和UGAL_LE分别为253.0、264.1、253.0和293.3周期，相比降低约13.3%、17.0%、13.3%和25.2%。报文传输占用通道资源少，表明

着网络可容纳更多报文。DOAR饱和吞吐量约0.50，VAL、PAR_{PH}、TPR和UGAL_LE分别约0.49、0.42、0.48和0.41，分别提高约2.0%、19.0%、4.2%和22.0%。DOAR在低负载下与MIN类似具有理想低延迟；在较高负载下比其他算法具有较低延迟；在高负载下具有更高吞吐量。

图10(f)显示RP流量下性能。由于流量目标分组分散均匀，DOAR在注入率小于0.3左右时大多采取3跳路径，类似PAR_{PH}延迟增长较缓。随着流量增加，DOAR监听局部全局通道拥塞时倾向选择更短路径，资源消耗较小，吞吐量更高；PAR_{PH}和VAL不区分路径长短，每次传输消耗通道资源多，吞吐量较低；TPR依赖于计数器统计流量信息，无法准确判断端口拥塞；UGAL_LE选择Non-Min路径灵活度不够，导致部分全局通道利用率低，吞吐量更低。VAL、PAR_{PH}、TPR、UGAL_LE吞吐量分别为0.49、0.47、0.45和0.43，而DOAR约为0.59，相比分别提高约20.4%、25.5%、31.1%和37.2%。DOAR算法在三种典型合成流量下相对PAR_{PH}、TPR和UGAL_LE算法平均吞吐量分别提高约15.2%、12.1%和23.7%。

图11进一步显示DOAR在各种ADV+i(i=1,…,8)流量下的性能趋势，区别在源分组与目标组间偏移量不同。以VAL算法在ADV+1流量模式下的性能参考。DOAR在ADV+1模式中性能最佳，在ADV+1至ADV+6(h=6)模式中性能略微下降。在ADV+1模式中大多流量只需要经过特定路由器上转发，中间分组内部直连通道的压力小。在其他模式下中间分组内部直连通道的压力逐渐变大且不均衡，多数报文需要额外的内部跳步进行转发，导致大量折射发生。ADV+6流量性能最低，主要表现为吞吐量大致不变，报文延迟相较于ADV+1增加约20个周期。针对所有ADV+i流量，DOAR在饱和吞吐量及各种速率下报文延迟方面均优于VAL算法，体现极端条件的性能优势。

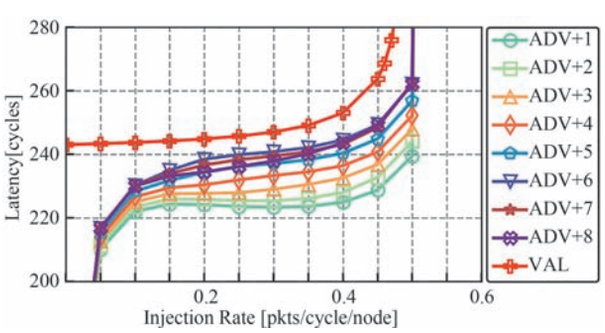


图11 不同ADV+i流量下的网络性能评估

DOAR算法在分组全局通道上体现出较好的负载均衡能力。随机选取交换分组，在接近饱和时统计全局通道使用次数并进行归一化处理。图12显示UR_0.8与ADV_0.4(后缀为注入率)两种流量下负载均衡情况，其中横坐标为全局通道号，纵坐标为使用次数归一化值。在UR流量下均匀使用全局通道进行转发。在ADV流量下通道4为最短路径全局通道，使用率最高；本地路由器全局通道次之；其余通道使用率略低但保持均匀调度，展现良好负载均衡能力。

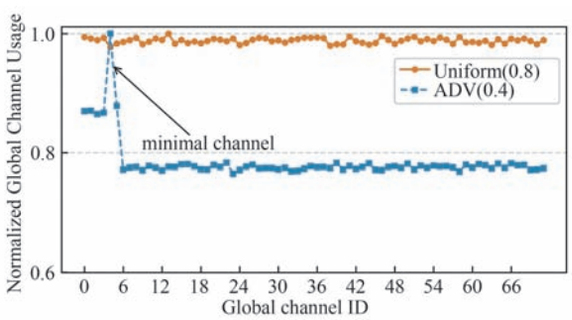


图12 通道负载均衡性能评估

5.2.2 混合合成流量模式

图13(a)(b)(c)显示了R_MIXED_UA混合流量性能，ADV流量比例分别为25%、50%和75%。首先在性能曲线开始，随着ADV流量比增加，延迟增长速度逐渐加快。图13(a)中ADV流量比较低时大量报文选择Min路径，组间流量比较均匀。随着ADV流量增加，继续采取Min路径使组间直连带宽出现瓶颈，更多报文选择Non-Min路径，延迟增长开始加速。图13(a)中DOAR于0～0.2区间的延迟由124.0增至145.8周期；图13(c)中于0～0.2区间的延迟由124.8快速增至195.5周期。其次性能曲线中间较稳定，DOAR尽可能选择跳步数少的路径，相对其他算法优势明显。在三种情形且注入率为0.4时，DOAR延迟分别为149.2、174.7和197.8周期，相较PAR_{PH}分别下降约7.6%、9.2%和11.8%，较TPR分别下降约3.8%、6.3%和9.6%。最后在性能曲线饱和区，DOAR路径跳步少，资源消耗低导致吞吐量高，吞吐量分别为0.72、0.61和0.54。VAL算法在各ADV占比下，随机选择路径的平均跳步数较多，饱和吞吐量偏低分别约0.45、0.43和0.44。PAR_{PH}优先选择Min路径但随机采取Non-Min路径，吞吐量分别约0.61、0.55和0.50。TPR依赖计数器估计组内和组间流量，但随机采取Non-Min路径进行路由，吞

吐量分别约0.67、0.55和0.51。UGAL_LE提高拥塞估计准确性，同样随机选择Non-Min路径，吞吐量分别约0.40、0.38和0.40。DOAR吞吐量相对PAR_{PH}提高约18.0%、10.9%和8.0%，相对TPR提高约7.5%、10.9%和5.9%，相对UGAL_LE提高约80.0%、60.5%和35.0%。

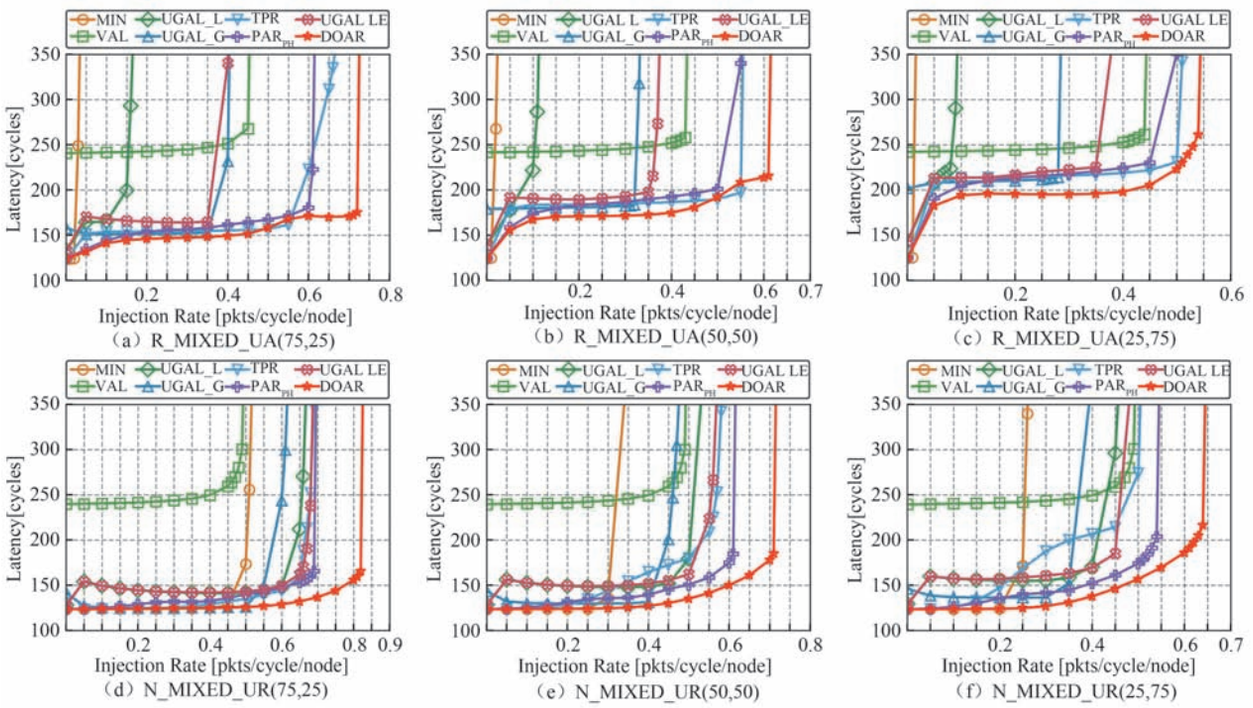


图13　混合流量模式下的网络性能评估

图13(d)(e)(f)显示了N_MIXED_UR混合流量性能，RP流量比分别为25%、50%和75%。在性能曲线开始，RP流量呈现对抗性不明显，多数报文选择Min路径，延迟没有明显增长。如图13(e)中RP流量占50%且注入率0.1时，UGAL_G、PAR_{PH}和DOAR延迟分别为130.4、125.3和123.7周期。在性能曲线中间，RP流量增加呈现一定的对抗性，全部采取Min路径会出现拥塞，部分报文开始采取Non-Min路径。相比而言，DOAR选择跳步数较小路径。当RP流量分别为25%、50%和75%且注入率为0.4时，DOAR延迟分别为124.9、127.2和137.7周期，较PAR_{PH}分别下降5.7%、8.8%和9.6%，较TPR分别下降3.3%、22.7%和33.4%，较UGAL_LE分别下降12.1%、16.1%和18.7%。在性能饱和区，随着RP占比增加更多报文采取Non-Min路径，PAR_{PH}吞吐量分别为0.69、0.61和0.54，TPR吞吐量分别为0.69、0.59和0.52，UGAL_LE吞吐量分别为0.69、0.57和0.50。DOAR路由算法跳步数少，具有更高吞吐量，分别为0.82、0.71和0.64。相对PAR_{PH}提高约18.8%、16.4%、18.5%；相对TRP提高约18.8%、20.3%、23.1%；相对UGAL_LE提高约18.8%、24.6%和28.0%。总之，DOAR在6种混合流量下相对PAR_{PH}、TPR和UGAL_LE吞吐量分别平均提高约15.1%、14.4%和41.2%。

5.2.3　实际应用负载

图14显示六个典型场景下延迟情况。LU程序具有一对多和多对一通信模式，全局流量多且负载重。PAR_{PH}、TPR和UGAL_LE延迟分别为658.8、734.2和714.0周期；DOAR约654.8周期，相比下降约0.6%、10.8%和8.3%。FFT3D程序Alltoall通信主要采取点对点方式。PAR_{PH}、TPR和UGAL_LE延迟分别约146.6、190.1和186.1周期；DOAR约142.3周期，相比下降2.9%、25.1%和23.5%。Halo3D与Stencil5D程序主要采取模板计算通信，典型的一对多通信且流量局部性较强。PAR_{PH}、TPR和UGAL_LE分别约292.3、362.5、332.1和134.2、209.9、198.7个周期，DOAR延迟约249.2和132.2周期，相比下降约14.7%、31.3%、25.0%和1.5%、37.0%、33.5%。DL程序含大量Allreduce操作，具有一对多和多对一通信且消息间隔适中。PAR_{PH}、TPR和UGAL_LE分别约139.6、

214.9和205.1周期，DOAR约136.7周期，相比下降约2.1%、36.4%和33.4%。LULESH属于混合通信模式。PAR_{PH}、TPR和UGAL_LE分别约675.9、840.8和722.9周期；DOAR约489.2周期，相比下降约27.6%、41.8%和32.3%。综上，DOAR在六种实际应用负载下延迟表现最优，与PAR_{PH}、TPR和UGAL_LE比较延迟分别降低约8.2%、30.4%和26.0%。

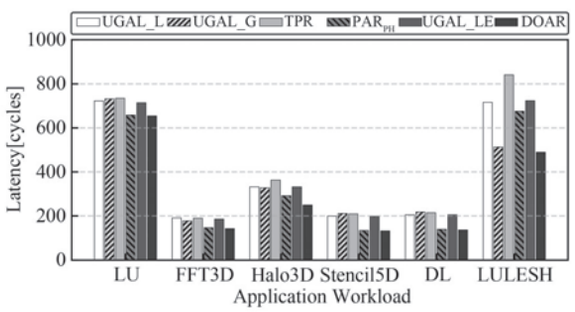


图14 不同应用负载的延迟性能评估

5.3 故障场景网络性能分析

本节先在路由器模型中增加故障处理部件和全局调度单元，采取特定标识报文用于故障诊断和恢复；然后在网络中随机插入特定数量的链路或路由器故障，复现故障自愈及恢复过程，统计容错场景下DOAR算法（简称DFT）性能。链路故障数量分别为10、40和200个，各占7446条链路数量的0.1%、0.5%和2.7%；路由器故障数量分别为10和30个，各占876个路由器的1.1%和3.4%。由于超智算中心通常按月巡检，上述比例可覆盖典型故障场景。

5.3.1 标准合成流量模式

图15(a)(b)(c)显示在链路故障下各种算法的跳步路径分布情况。在UR模式下，当注入链路故障时，DFT由于部分路径无法使用开始采取Non-Min路径传输，优先采取4跳路径。DFT_10、DFT_40和DFT_200采取少量短路径便绕开故障，较无故障时DOAR算法，4跳路径分别增至1.1%，3.1%和6.7%，5跳路径分别增至0.1%，0.4%和0.9%，几乎没有出现6跳路径，网络性能损失较小。在ADV模式下DOAR几乎未使用5和6跳路径，发生链路故障时仅采取低跳步Non-Min路径替代，未明显增加5和6跳路径。DFT_10、DFT_40和DFT_200中4跳路径略增加，网络性能未见明显影响。在RP流量模式下，当链路出现故障时DFT_10、DFT_40、DFT_200中5和6跳路径略微增加。

图15(d)将DFT与无故障时VAL、PAR_{PH}和UGAL_LE算法进行比较，显示三种流量下不同链路数量场景下的性能。在UR流量下，DFT_10于0.95处达到饱和，与无故障情况下DOAR和MIN的性能相当；DFT_40吞吐量约为0.94，优于PAR_{PH}；DFT_200吞吐量降至0.88，较UGAL_LE高0.02。相对DOAR，DFT_40和DFT_200饱和吞吐量分别仅下降约1.1%和7.4%。在ADV流量下，DFT_10于0.50处达到饱和，性能与图10(e)中DOAR相近且较TPR高0.02；DFT_40在0.50处饱和；DFT_200在0.47处饱和，吞吐量与图10(e)中TPR相当。相对DOAR，DFT_10和DFT_40饱和吞吐量保持不变，DFT_200饱和吞吐量仅下降约6.0%。RP流量情况类似，在三种故障场景下的吞吐量分别约0.59、0.58、0.56，优于图10(f)中PAR_{PH}(0.47)。相对DOAR，DFT_10饱和吞吐量保持不变，DFT_40和DFT_200饱和吞吐量分别仅下降约1.7%和5.1%。由此可见，DFT在合成流量少数链路故障时性能优于PAR_{PH}、TPR、UGAL_LE算法，在链路故障较多时仍保持较高性能。

图15(e)显示了40条典型链路故障场景下DFT与SIM（Safety Information Model）算法[37]延迟性能对比。SIM采取重建拓扑方式实现全局结点可达，网络性能损失大，而DFT只需作废路由表中部分路由项，维持网络路径最大化，具有较高吞吐性能和较低延迟。在UR、ADV和RP三种模式下SIM饱和吞吐量分别为0.85、0.41和0.43，DFT饱和吞吐量为0.94、0.50和0.58，分别提高10.6%，22.0%和34.9%。选取UR_0.75、ADV_0.3、RP_0.4负载流量对比延迟性能，SIM延迟分别为143.0、249.3和199.8周期，DFT延迟为130.7、218.1和157.8周期，分别降低8.6%、12.5%和21.0%。

图15(f)显示DFT在三种流量接近饱和时报文延迟随链路故障数的变化趋势。随着故障增加，DFT在UR和ADV流量下只需额外增加少量4跳步Non-Min路径能绕开故障链路，基本无需5~6跳路径，在RP流量下仅增加跳步数较少的4~5跳路径，由此可见增加链路故障对报文延迟影响较小。图15(g)显示饱和吞吐量随链路故障数变化趋势。UR流量下使用Non-Min路径替换Min路径，4跳路径增幅相对较大，相比其他流量的资源消耗增幅较多，导致吞吐量由0.95逐步降至0.88。在其他流量下无故障时已选择部分Non-Min路径，故障发生时高

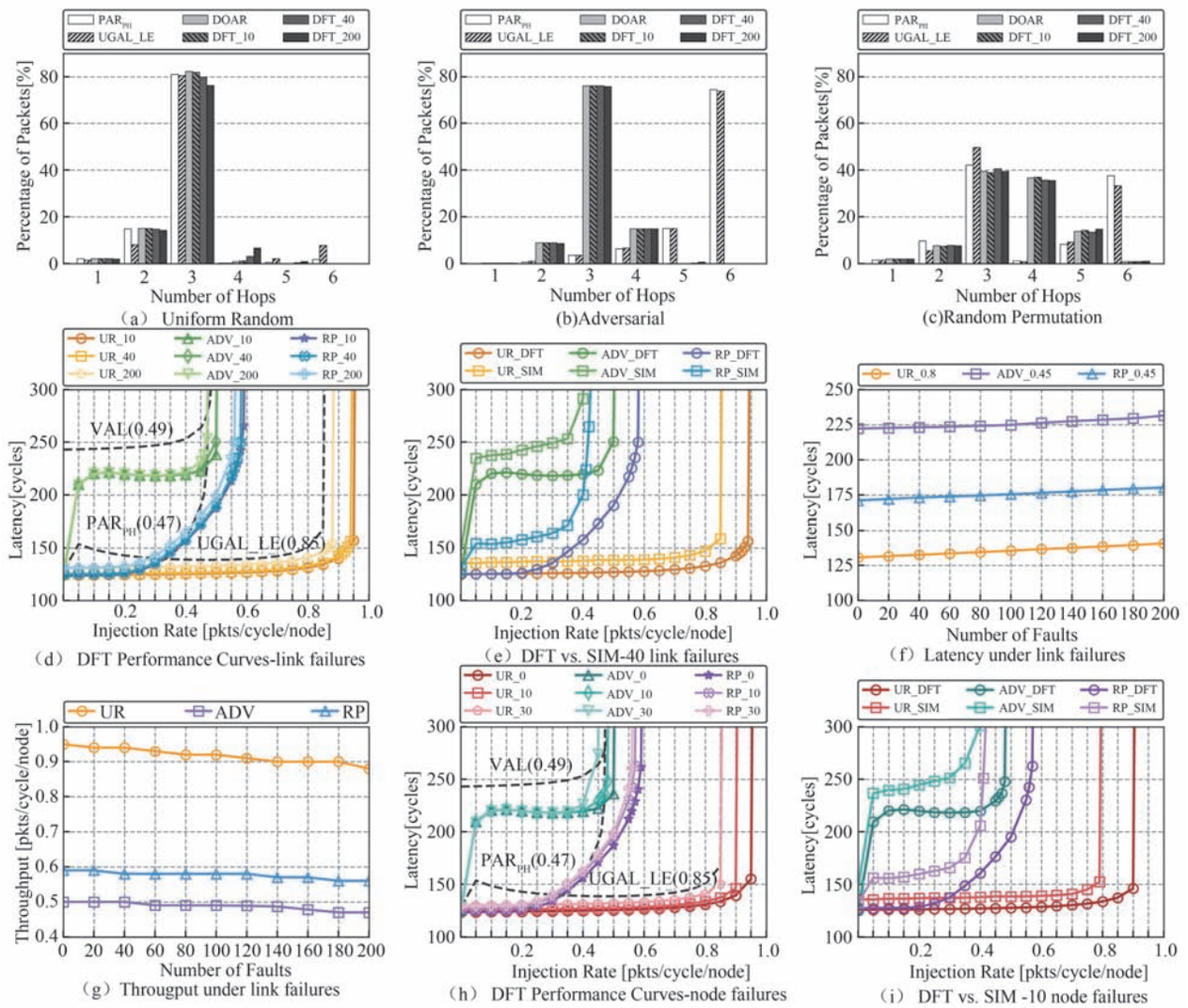


图15　不同故障场景下合成流量性能评估

跳步数路径增幅很小，吞吐量基本不变。针对链路故障，DFT利用多样性路径绕开故障并主动减少跳步数，性能损失较小，体现良好容错能力和稳定性。

图15(h)进一步显示路由器故障DFT性能评估，随机失效10和30个路由器，分别记为DFT_N10和DFT_N30。在UR流量下，DFT_N10于0.90处达到饱和；DFT_N30吞吐量约为0.85，与UGAL_LE性能相当。相对DOAR，DFT_N10和DFT_N30饱和吞吐量分别下降5.3%和10.5%。在ADV流量下，DFT_N10于0.48处饱和，DFT_N30在0.45处饱和。其性能仍优于无故障情况下PAR_{PH}和TPR。相对DOAR，DFT_N10和DFT_N30饱和吞吐量分别下降约4.0%和10.0%。在RP流量下，DFT_N10和DFT_N30吞吐量约0.57和0.56，性能优于图10(f)中的PAR_{PH}。相对DOAR，DFT_N10和DFT_N30饱和吞吐量分别下降约3.4%和5.1%。由此可见，DFT在路由器故障场景下保持较高性能。

图15(i)显示10个路由器故障场景下DFT与SIM性能对比。在UR、ADV和RP三种流量模式下SIM饱和吞吐量分别为0.79、0.40和0.42。DFT饱和吞吐量为0.90、0.48和0.57，分别提高13.9%，20.0%和35.7%。同时DFT延迟优势明显，在UR_0.75、ADV_0.3、RP_0.4流量下SIM延迟分别为145.0、251.3和205.8周期，DFT延迟分别131.6、218.1和160.6周期，各自减少9.2%、13.2%和22.0%。DFT在路由器故障时仍有高吞吐量与低延迟特征。

5.3.2　混合合成流量模式

图16显示了三种节点混合流量下100条链路故障发生后的路径分布和性能曲线变化情况。图16(a)中VAL算法在流量趋于饱和时5和6跳路径各占

22.5%和69.2%，PAR_{PH}、UGAL_LE和TPR算法的6跳路径各占33.8%、24.9%和16.6%。较高比例的大跳步数路径使得饱和前延迟较大，曲线缓慢增长，吞吐量偏低。DOAR算法优先采取较小跳步Non-Min路径，趋于饱和前大多采取2～3跳路径，4跳路径约4.3%，饱和前延迟约154周期。当故障出现后DFT_100优先选择跳步较少的Non-Min路径，4跳增至6.4%，5～6跳路径占比极低。如图16(d)所示性能曲线饱和前延迟也较低，DOAR在0.78处饱和，DFT_100在0.75处饱和，吞吐量下降约3.8%。但DFT_100性能仍优于PAR_{PH}和UGAL_LE，与TPR算法相当。

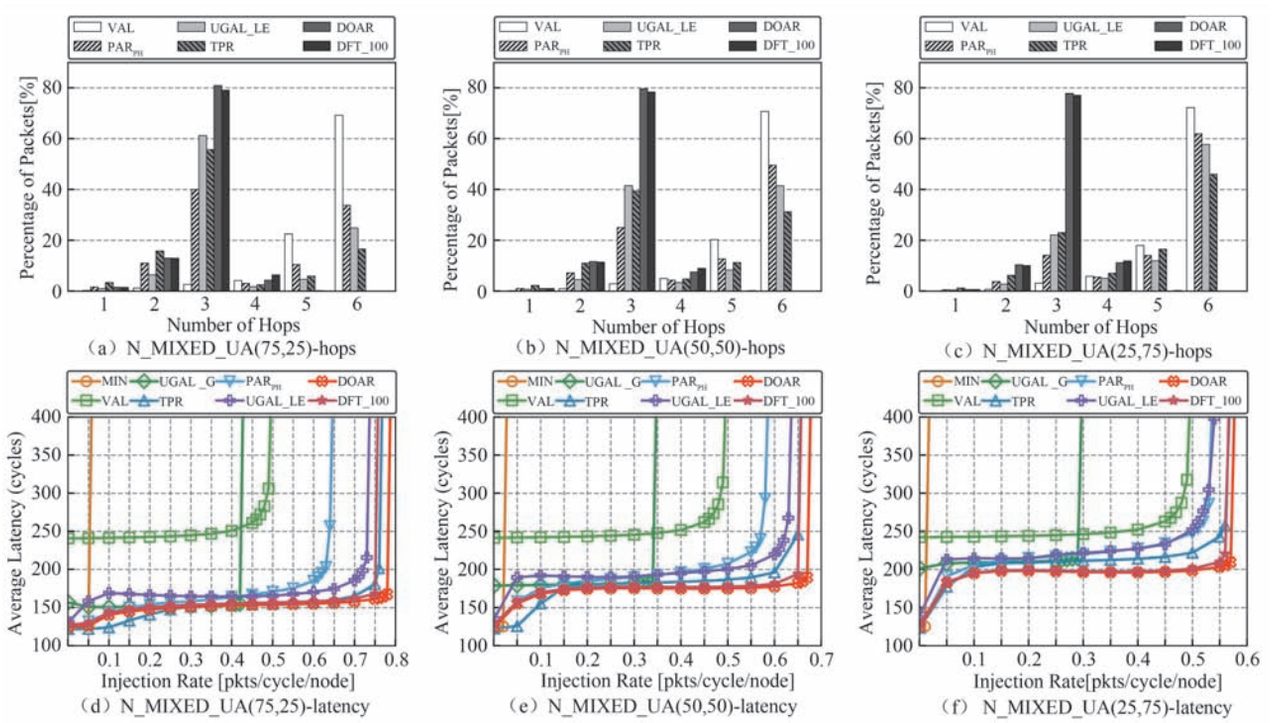


图16 100个链路故障场景下混合流量性能评估

图16(b)中VAL和PAR_{PH}在趋于饱和时高跳步路径占比大致相同，但PAR_{PH}、UGAL_LE和TPR算法的6跳路径增加，分别占比49.6%、41.5%和31.3%，导致饱和前延迟明显变大。DOAR饱和前4跳路径约7.5%，故障出现后4跳路径增至8.9%。DOAR在0.67处饱和，DFT_100在0.65处饱和，下降约3.0%。DFT_100仍优于PAR_{PH}和UGAL_LE算法，与TPR算法相当。图16(c)结果类似，PAR_{PH}饱和前延迟继续增加，DOAR饱和前延迟变化比较小；DOAR在0.57处饱和，DFT_100在0.56处饱和，下降约1.8%。DFT_100性能仍优于PAR_{PH}和UGAL_LE算法，与TPR算法相当。由此可见，DFT_100在混合流量模式下优先选择较少跳步路径导致传输延迟较小，性能饱和时曲线陡峭，与PAR_{PH}和UGAL_LE对比优势显著。

5.3.3 故障处理效率分析

在UR流量下对自动故障诊断及恢复过程进行仿真，保持高注入率约0.95。故障诊断中DFT对故障迅速响应并重新计算可用路径，诊断处理本地和远程故障分别耗时约1300和100个周期；故障恢复过程中遍历端口路由表约2000个周期。

图17显示故障诊断恢复前后网络性能的变化过程。在正常无故障状态，网络第一次稳定时吞吐量约0.95，延迟约149.2个周期。在第10 K个周期处随机插入50个链路故障，网络吞吐量立即下降。故障链路造成一部分包丢失，节点接收端流量降至0.93。此时部分路由器端口缓冲排队压力减小，包延迟略降至147.4周期。从10 K周期自动故障诊断开始，本地故障处理约在11.3 K周期完成，远程故障处理约在13.5 K周期完成，路由表更新基本完毕，丢包现象得以解决，如A区间所示。从13.5 K至15.2 K周期(B区间)，网络性能逐步收敛至第二次稳定阶段，吞吐量约0.94，相比第一阶段降低约1.1%；延迟约157.7周期，相比增加约5.7%。由于路由表更新后部分Min路径失效，路由算法选择跳步数较高Non-Min路径替代，跳步数增加，排队压

力上升，导致包延迟增加。在15.2 K至20 K周期网络均处于第二次稳定阶段。

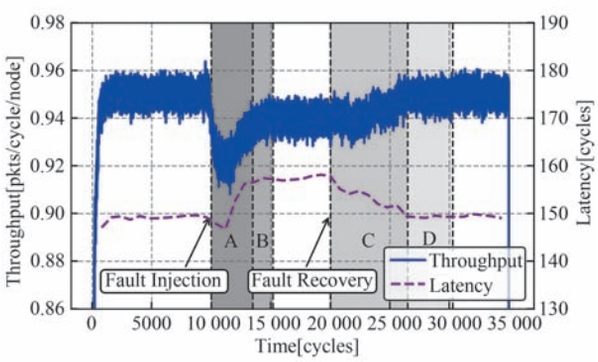


图17　故障诊断与恢复阶段网络性能变化曲线

其次，在20 K周期恢复所有故障链路。在20 K至26.5 K周期（C区间）内大多数路径逐步恢复尤其是Min路径，网络性能趋于收敛，吞吐量升至0.95，延迟降至149.2周期。从26.5 K至30.3 K周期（D区间），故障恢复阶段路由表更新完毕，性能基本维持不变。在第30.3 K至35 K周期处于第三次稳定阶段，与第一次稳定阶段类似。

实验结果显示，多链路故障时DFT快速找到替代路径确保网络通信稳定可靠。假设路由器工作主频为1.0 GHz。在故障发生后，故障处理过程（A区）耗时约3.5 μs网络停止丢包，故障诊断（A区+B区）耗时约5.2 μs网络性能收敛。在故障恢复后，网络未出现丢包情况，网络性能收敛耗时约6.5 μs，整个故障恢复过程（C区+D区）耗时约10.3 μs，表现出优异的故障处置实时性。

6　总　结

本文结合天河E级原型系统网络分布式分层路由表结构和蜻蜓拓扑，提出DOAR路由算法有效提升了各种流量下的自适应路由性能；面向常见故障场景提出了动态路由自愈及恢复技术，在自动处置故障、维持通信稳定和减少性能损失方面表现出色。

与蜻蜓拓扑传统路由算法靠局部流量信息随机选择多样化路径不同，DOAR算法通过动态评估组内与组间通道状态，优先选择有助于均衡负载且跳步数少的路径，减少了随机选择非最小路径导致的性能损耗。首先，提出了路由计算部件及可重构路由配置方法，具有计算延迟低、配置灵活、无死锁特点。其次，针对蜻蜓拓扑中故障影响大特点，提出了动态路由自愈恢复机制及相应硬件结构，通过链路故障检测、逆向报文传播、快速故障诊断及恢复等过程，支持了高效的故障自动诊断与故障恢复。实验表明，该机制在各种流量下均表现出优异性能，在典型故障场景下不仅故障自动诊断和恢复在十微秒量级，有效保持了系统通信稳定性，而且性能损失小。下一步工作主要考虑在更大规模网络仿真中的DOAR算法性能评估。

参　考　文　献

[1] Yamamura S, Akizuki Y, et al. A64FX: 52-core processor designed for the 442petaflops supercomputer fugaku// Proceedings of 2022 IEEE International Solid-State Circuits Conference (ISSCC). San Francisco, USA, 2022: 352-354

[2] Al-Fares M, Loukissas A, Vahdat A. A scalable, commodity data center network architecture. ACM SIGCOMM computer communication review, 2008, 38(4): 63-74

[3] De Sensi D, Di Girolamo S, et al. An in-depth analysis of the slingshot interconnect//Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis. Atlanta, USA, 2020: 1-14

[4] Cao J, Xiao L, Pang Z, et al. The efficient in-band management for interconnect network in Tianhe-2 system// Proceedings of 24th International Conference on Parallel, Distributed, and Network-Based Processing (PDP). Heraklion, Greece, 2016: 18-26

[5] Xie M, Lu Y, Wang K, et al. Tianhe-1a interconnect and message-passing services. IEEE Micro, 2012, 32(1): 8-20

[6] Gao J, Zheng F, Qi F, et al. Sunway supercomputer architecture towards exascale computing: analysis and practice. Science China Information Sciences, 2021, 64(4): 141101

[7] Kim J, Dally W, Towles B, et al. Microarchitecture of a high radix router//Proceedings of the 32nd International Symposium on Computer Architecture. Madison, USA, 2005: 420-431

[8] Kim J, Dally W J, Abts D. Flattened butterfly: a cost-efficient topology for high-radix networks//Proceedings of the 34th annual international symposium on Computer architecture. San Diego, USA, 2007: 126-137

[9] Ahn J H, Binkert N, Davis A, et al. HyperX: topology, routing, and packaging of efficient large-scale networks// Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis. Portland, USA: 2009: 1-11

[10] Kim J, Dally W J, Scott S, et al. Technology-driven, highly-scalable dragonfly topology//Proceedings of the 2008 International Symposium on Computer Architecture. Beijing, China, 2008: 77-88

[11] Shpiner A, Haramaty Z, Eliad S, et al. Dragonfly+: Low cost topology for scaling datacenters//Proceedings of International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB). Austin, USA,

2017: 1-8

[12] Faanes G, Bataineh A, Roweth D, et al. Cray Cascade: A scalable HPC system based on a Dragonfly network// Proceedings of the 2012 International Conference for High Performance Computing, Networking, Storage and Analysis. Salt Lake City, USA, 2012: 1-9

[13] Besta M, Hoefler T. Slim fly: A cost effective low-diameter network topology//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. New Orleans, USA, 2014: 348-359

[14] Flajslik M, Borch E, Parker M A. Megafly: A topology for exascale systems//Proceedings of International Conference on High Performance Computing. Frankfurt, Germany, 2018: 289-310

[15] Lei F, Dong D, Liao X, et al. Galaxyfly: A novel family of flexible-radix low-diameter topologies for large-scales interconnection networks//Proceedings of the 2016 International Conference on Supercomputing. Istanbul .Turkey, 2016: 1-12

[16] TOP500, https://www. top500. org/lists/top500/2024/06/ 2024,6,18

[17] Fuentes P, Vallejo E, Camarero C, et al. Network unfairness in dragonfly topologies. The Journal of Supercomputing, 2016, 72(12): 4468-4496

[18] García M, Vallejo E, Beivide R, et al. On-the-fly adaptive routing for dragonfly interconnection networks. The Journal of Supercomputing, 2015, 71(3): 1116-1142

[19] Singh A. Load-balanced routing in interconnection networks. Stanford University, USA, 2005

[20] Faizian P, Alfaro J F, Rahman M S, et al. TPR: traffic pattern-based adaptive routing for dragonfly networks. IEEE Transactions on Multi-Scale Computing Systems, 2018, 4(4): 931-943

[21] Kasan H, Kim G, Yi Y, et al. Dynamic global adaptive routing in high-radix networks//Proceedings of the 49th Annual International Symposium on Computer Architecture. New York, USA, 2022: 771-783

[22] Chaulagain R S, Yuan X. Enhanced ugal routing schemes for dragonfly networks//Proceedings of the 38th ACM International Conference on Supercomputing. Kyoto, Japan, 2024: 449-459

[23] Jiang N, Kim J, Dally W J. Indirect adaptive routing on large scale interconnection networks//Proceedings of the 36th annual international symposium on Computer architecture. Austin, USA, 2009: 220-231

[24] Won J, Kim G, Kim J, et al. Overcoming far-end congestion in large-scale networks//Proceedings of the International Symposium on High Performance Computer Architecture (HPCA). Burlingame, USA, 2015: 415-427

[25] Fuentes P, Vallejo E, García M, et al. Contention-based nonminimal adaptive routing in high-radix networks//Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium. Hyderabad, India, 2015:103-112

[26] Garcia M, Vallejo E, Beivide R, et al. Efficient routing mechanisms for dragonfly networks//Proceedings of the 42nd International Conference on Parallel Processing. Lyon, France, 2013: 582-592

[27] Rahman M S, Bhowmik S, Ryasnianskiy Y, et al. Topology-custom UGAL routing on dragonfly// Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis. Denver, USA, 2019: 1-15

[28] Benito M, Fuentes P, Vallejo E, et al. ACOR: Adaptive congestion-oblivious routing in dragonfly networks. Journal of Parallel and Distributed Computing, 2019, 131: 173-188

[29] Zhu L, Gu H, Yu X, et al. AMLR: An adaptive multi-level routing algorithm for dragonfly network. IEEE Communication Letters, 2021, 25(11): 3533-3536

[30] Kang Y, Wang X, Lan Z. Q-adaptive: A multi-agent reinforcement learning based routing on dragonfly network// Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing. Virtual, Sweden, 2021: 189-200

[31] JungHyungsoo, HanHyuck, Yeom H Y, et al. Athanasia: A user-transparent and fault-tolerant system for parallel applications. IEEE Transactions on Parallel and Distributed Systems, 2011, 22(10): 1653-1668

[32] Birrittella M S, Debbage M, et al. Intel® omni-path architecture: Enabling Scalable, High Performance Fabrics// Proceedings of the IEEE 23rd Annual Symposium on High-Performance Interconnects. Santa Clara, USA, 2015: 1-9

[33] Underwood K D, Borch E. Exploiting communication and packaging locality for cost-effective large scale networks// Proceedings of the 26th ACM international conference on Supercomputing. Venice, Italy, 2012: 291-300

[34] Puente V, Gregorio J A, Vallejo F, et al. Immunet: Dependable routing for interconnection networks with arbitrary topology. IEEE Transactions on Computers, 2008, 57(12): 1676-1689

[35] Puente V, Gregorio J A. Immucube: Scalable fault-tolerant routing for k-ary n-cube networks. IEEE Transactions on Parallel and Distributed Systems, 2007, 18(6): 776-788

[36] Sem-Jacobsen F O, Skeie T, Lysne O, et al. Dynamic fault tolerance in fat trees. IEEE Transactions on Computers, 2011, 60(4): 508-525

[37] Xiang D, Li B, Fu Y. Fault-tolerant adaptive routing in dragonfly networks. IEEE Transactions on Dependable and Secure Computing, 2017, 16(2): 259-271

[38] Vigneras P, Quintin J N. Fault-tolerant routing for exascale supercomputer: The bxi routing architecture//Proceedings of the 2015 IEEE International Conference on Cluster Computing. Chicago, USA, 2015: 793-800

[39] Gliksberg J, Capra A, Louvet A, et al. High-quality fault resiliency in fat trees. IEEE Micro, 2019, 40(1): 44-49

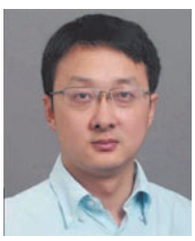
[40] NVIDIA Networking, https://network.nvidia.com/pdf/whitepa (pers/WP_Mellanox_SHIELD.pdf, 2020,6,1)

[41] Jiang N, Balfour J, Becker D U, et al. A detailed and flexible cycle-accurate network-on-chip simulator//Proceedings of the 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). Austin, USA, 2013: 86-96

[42] Kang Y, Wang X, Lan Z. Study of workload interference with

intelligent routing on dragonfly//Proceedings of the SC22: International Conference for High Performance Computing, Networking, Storage and Analysis. Dallas, USA, 2022: 1-14

[43] Rodrigues A F, Hemmert K S, Barrett B W, et al. The structural simulation toolkit. ACM SIGMETRICS Performance Evaluation Review, 2011, 38(4): 37-42



LAI Ming-Che, Ph. D., professor. His research interests include computer architecture, interconnection network, FPGA-based hardware accelerators, and hybrid integrated optoelectronic chip.

WANG Zheng-Hao, Ph. D. candidate. His main research interests include computer architecture, chiplet interconnection and high performance interconnection network.

XU Jia-Qing, Ph. D., associate research fellow. His research interests include high-performance computing and high performance interconnection network.

GAO Lei, Ph. D., associate research fellow. Her research interests include computer architecture, artificial intelligence and high performance interconnection network.

OU Yang, Ph. D., associate research fellow. His research interests include computer architecture and high performance interconnection network.

WU Li-Zhou, Ph. D., assistant research fellow. His research interests include heterogeneous memory pooling, computer architecture and high performance interconnection network.

WANG Qiang, Ph. D., assistant research fellow. His research interests include computer architecture, high performance interconnection network, and artificial intelligence.

Background

High-performance networks are important infrastructure for supercomputing and intelligent computing centers. The most common fat tree topology has balanced traffic and high bipartite bandwidth, but the scalability is poor, and the maximum node size of Tianhe and Sunway is only tens of thousands. Fat Tree relies on increasing the scale of tiers, resulting in exponential increases in power consumption and cost, and it is difficult to support hundreds of thousands of points of 10E-level high-performance computers in the future. The dragonfly topology of high-performance computer systems has the advantages of strong scalability, low latency and low cost, and foreign systems such as Frontier and Aurora adopt this topology. However, dragonfly topology is sensitive to faults, and network faults have a great impact on system stability. Therefore, it is crucial to study adaptive routing and fault-tolerant routing in dragonfly topologies for network performance and stability.

The minimum path communication is adopted at any two nodes of the dragonfly topology, and the number of hops is small, but it is difficult to adapt to various complex traffic by relying only on a small number of shortest paths. It is often considered to take advantage of a large number of non-minimum paths to alleviate congestion by adding diverse paths. However, the inability to perceive remote congestion through local traffic selection leads to suboptimal decision-making. Random path selection achieves load balancing to a large extent, but the path length is not optimal, and excessive resource consumption will limit the throughput performance. The traditional routing method combined with local traffic to select diversified paths has insufficient consideration on the impact of path hopping, and the performance optimization effect is limited.

In this paper, a Distributed On-the-Fly Adaptive Routing (DOAR) algorithm is proposed based on the distributed hierarchical routing table structure and dragonfly topology of the Tianhe Express network, which dynamically selects paths according to the local channel status within the group and the global channel state between groups in packet transmission, so as to alleviate congestion and reduce the number of hops. The structure of route computing components and the reconfigurable route configuration method are proposed, which have the characteristics of low computing delay, flexible configuration and no deadlock. On this basis, the dynamic routing self-healing and recovery technology is proposed, which actively generates diagnostic or recovery packets to update the distributed routing table, and proposes the structure of automatic fault handling components to achieve the effect of fast fault automatic diagnosis and recovery.