Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.085191

ARTICLE

Congestion-Aware Load Balancing with Flowlet Switching Based on Data and Control Plane Cooperation

Ziyong Li[1,*], Yusheng Xia[1], Junfei Li[2], Le Tian[2] and Xinglong Pei[2]

[1]Military Intelligence Research Institute, Military Academy of Sciences, Beijing, China
[2]National Digital Switching System Engineering and Technological Research Center, Information Engineering University, Zhengzhou, China

*Corresponding Author: Ziyong Li. Email: 17629352940@163.com

Received: 06 May 2026; Accepted: 17 August 2026; Published: 15 September 2026

ABSTRACT: Multipath load balancing can effectively improve network throughput and reliability by aggregating the available bandwidth of multiple paths. However, existing load balancing schemes including Equal-Cost Multi-Path forwarding (ECMP), Weighted-Cost Multi-Path forwarding (WCMP) or LetFlow may lead to significant performance degradation due to hash conflicts and only target fixed symmetric topologies (e.g., Fattree). Flowlet switching has been proven to be a fine-grained load balancing technique, but remains elusive for widespread deployment. The emergence of network programmability including the control plane and data plane provides a new insight for the management of multipath load balancing. To achieve more effective load balancing and guarantee Quality of Service (QoS) on any network topology, we present ConFlet, a congestion-aware load balancing with Flowlet switching based on the cooperation of the data and control plane, where the control plane monitors the global network status to perform the optimized routing decision, and the data plane implements the flow splitting and path switching. Specifically, the centralized controller evaluates the reliability of nodes and links to perform multi-path routing calculations, ensuring that flows always travel along the most reliable multiple paths. The data plane can dynamically set the flowlet timeout to split flows into flowlets according to the path quality difference, and then implement path switching and congestion avoidance based on real-time congestion feedback. Experimental results show that compared to WCMP/LetFlow, ConFlet can significantly improve average service reliability and network throughput, and reduce average packet delay. Meanwhile, ConFlet can react quickly to congestion and maintain high resilience to network asymmetry.

KEYWORDS: Load balancing; flowlet switching; network programmability; congestion avoidance

1 Introduction

With the rapid development of network technologies including 6G, satellite Internet and edge computing, modern Internet infrastructures are required to accommodate a growing variety of differentiated applications. These emerging applications such as video broadcasting, telemedicine, and big-data analytics are profoundly affecting people's lives, which also poses great challenges to the Quality of Service (QoS) of the Internet. Multipath routing can effectively aggregate available bandwidth to provide high-quality network services, and also improve the reliability of communication [1]. However, the current Internet lacks fine-grained forwarding control for multipath routing management, which is difficult to support more applications with higher requirements for low delay, high throughput, and high reliability [2].

Nowadays, the Internet is characterized by a topology with high connectivity and a large number of equivalent paths. Equal-Cost Multi-Path forwarding (ECMP) remains the dominant load balancing

approach deployed across a majority of traditional IP routing devices. As a classic flow-level multipath routing scheme, ECMP adopts a dedicated hash calculation module to conduct random mapping to allocate every flow onto a single forwarding link among all available paths. However, due to hash conflicts and uneven allocation of large and small flows, ECMP may cause significant performance degradation [3]. WCMP [4] and Niagara [5] are weighted ECMP algorithms that set different weights for each path according to the available bandwidth, and implement routing according to the path weight. However, these schemes only work under specific symmetric network topologies (e.g., Fattree) in data centers, and perform poorly in other asymmetric topologies due to a lack of visibility into path congestion.

Due to the scale, heterogeneity, and high performance required by many emerging applications, the QoS and reliability of the Internet have become more challenging, which increases the need and complexity of managing multiple paths in the network. Software-Defined Networking decouples the control plane and data plane to enable unified and centralized global control over the entire network system, significantly simplifying automated network maintenance, orchestration and daily operational management tasks [6]. However, due to the non-negligible controller-switch delay and communication overhead (e.g., notification of events and installation of flow rules), the SDN controller is too slow to react quickly to sudden network events (e.g., network congestion and link failure). The programmable data plane not only empowers network operators to independently customize the underlying packet processing rules running on forwarding hardware, but also to offload some network functions (e.g., large flow monitoring, load balancing and in-band telemetry) to the data plane, which can effectively improve the QoS of the network [7]. Hence, the next generation network architecture may adopt SDN to implement the global optimization decision at the flow level, and combine P4 to sustain precise, fine-grained forwarding management at the packet level.

To overcome the limitations of existing load balancing schemes, we present ConFlet, a congestion-aware load balancing scheme with flowlet switching. ConFlet combines SDN and P4 to implement centralized and distributed collaborative management, retains the control plane to achieve centralized multipath routing control, and exploits the data plane to perform flowlet-level load balancing. The main contributions of our work can be concluded as follows:

- We identify the problem that existing load balancing schemes may lead to significant performance degradation due to hash conflicts and only target fixed symmetric topologies (e.g., Fattree). To overcome these limitations, we propose a general load balancing scheme named ConFlet, which implements centralized and distributed collaborative control to perform flowlet-level load balancing and congestion avoidance.
- We use a centralized controller to evaluate the reliability of nodes and links, and implement global multipath routing decisions to ensure reliable transmission of flows. On the other hand, we exploit a programmable data plane to perform flowlet-level load balancing based on adaptive timeouts and implement distributed congestion-awareness path switching, which realizes more fine-grained load balancing.
- We construct comprehensive simulations to verify the network performance of ConFlet. The experimental results show that compared with WCMP/LetFlow, ConFlet significantly improves network performance and performs quite well when encountering link congestion and network asymmetry.

The rest of this paper is organized as follows. The related works are presented in Section 2. Section 3 gives our research motivation about multipath load balancing. Section 4 details the design principles of ConFlet. Section 5 evaluates the performance of ConFlet. Finally, we give our conclusion in Section 6.

2 Related Works

Multipath routing includes multipath calculation and multipath forwarding control. Multipath calculation uses efficient algorithms to calculate multiple node/link-disjoint paths based on the global network view for meeting the QoS requirements of a given flow. Multipath forwarding control splits individual flows and distributes packets onto a set of alternative transmission paths. For multipath calculation, some centralized mechanisms (e.g., Hedera [8], ARION [9], Flowcut [10] and EPIC [11]) exploited centralized controllers to monitor network link status, and then rescheduled large flows or congested flows in real time to alleviate network congestion. Farhan et al. [12] proposed a DRL-based approach for improving SDN traffic engineering named RDG-TE. RDG-TE adopts deep reinforcement learning (DRL) and GNN, considering multiple parameters such as link reliability, available link bandwidth, and node betweenness, to predict the optimal path of data flow in model training and reward calculation for achieving load balancing. Cheng and Jia [13] abstracted the co-optimization problem of multi-path routing and flow table matching constraints as a biconvex integer programming model and put forward a dedicated multi-path transmission strategy named NAMP. NAMP designed a new LP-based alternating convex search algorithm to minimize the transmission time of flow groups when meeting the flow table capacity constraints. Yan et al. [14] distinguished different types of services to calculate multiple paths that meet specific QoS constraints between source and destination nodes, and real-time monitored link utilization to select the optimal path for each flow, which can significantly boost overall network throughput while cutting down the end-to-end transmission latency of packets. However, the centralized control schemes cannot quickly capture the instantaneous state of the network, which was difficult to adapt to the fluctuations of traffic.

Multipath forwarding control can be divided into flow, packet and flowlet granularity. Flow granularity (e.g., ECMP and Hedera) uses the same path to forward all packets of each flow. However, due to hash conflicts, flow granularity will cause congestion or underutilization. Packet granularity transmits all packets along different paths. Although achieving high load balancing, packet granularity will cause serious packet out-of-order. Some other schemes (e.g., CONGA [15], HULA [16]) divided the flow into smaller units "flowlets," according to the packet arrival interval from the same flow, and assigned these flowlets to different paths. Compared with flow and packet granularity, flowlet granularity achieves higher load balancing and avoids packet out-of-order. Vanini et al. [17] proposed a flowlet-based load balancing mechanism (LetFlow). LetFlow can control flowlet granularity by setting different flowlet timeouts and randomly selecting paths for flowlets to realize high load balancing. However, LetFlow cannot dynamically change the flowlet timeout with the traffic load of the network. Besides, these multipath routing schemes, including ECMP, CONGA, HULA, and LetFlow, are only designed for specific data center topologies. The comparison of typical multipath routing schemes is shown in Table 1.

Table 1: Comparison of typical multipath routing schemes.

Scheme	Granularity	Centralized or Distributed	Control Overhead	Topological Dependency
ECMP [3]	Flow	Distributed	None	Yes
WCMP [4]	Flow	Distributed	Moderate	Partially
Hedera [8]	Flow (large or small)	Centralized	High	Yes
HULA [16]	Flowlet (fixed timeout)	Distributed	High	Yes
LetFlow [17]	Flowlet (fixed timeout)	Distributed	Low	Yes
ConFlet	Flowlet (adaptive timeout)	Combining centralized and distributed	High	No

In summary, existing multipath routing schemes either use a centralized controller to make decisions, leading to slow response to instant congestion, or implement local routing strategies, only working in specific symmetric topologies. ConFlet is a general load balancing scheme that can work seamlessly on any network topology. It takes full advantage of centralized routing decisions and distributed forwarding control to implement flowlet-level load balancing and congestion-aware path switching under strict reliability requirements for enhancing the QoS and reliability of the network.

3 Research Motivation

ECMP and WCMP are typical load balancing schemes in data center networks. ECMP randomly selects one path from multiple equivalent paths for forwarding. WCMP implements a set of flow splitting based on the ideal weights initially calculated by the SDN controller. Let's provide an example to show that ECMP and WCMP may cause serious load imbalance and performance degradation. Consider a simple asymmetric topology in Fig. 1. There are three paths with available bandwidth ratios of 1:2:3 for forwarding each flow. ECMP lacks awareness of path congestion and distributes flows equally across multiple paths. Two or more elephant flows are transmitted on the bottleneck link, resulting in serious load imbalance. WCMP uses the controller to realize global congestion awareness, and assigns different weights for paths according to the available bandwidth of different paths, effectively avoiding load imbalance. However, WCMP still has hash conflicts. Once link congestion occurs due to hash conflicts or traffic bursts, WCMP can only request the controller to update the path weight to alleviate congestion, easily damaging latency-sensitive flows.

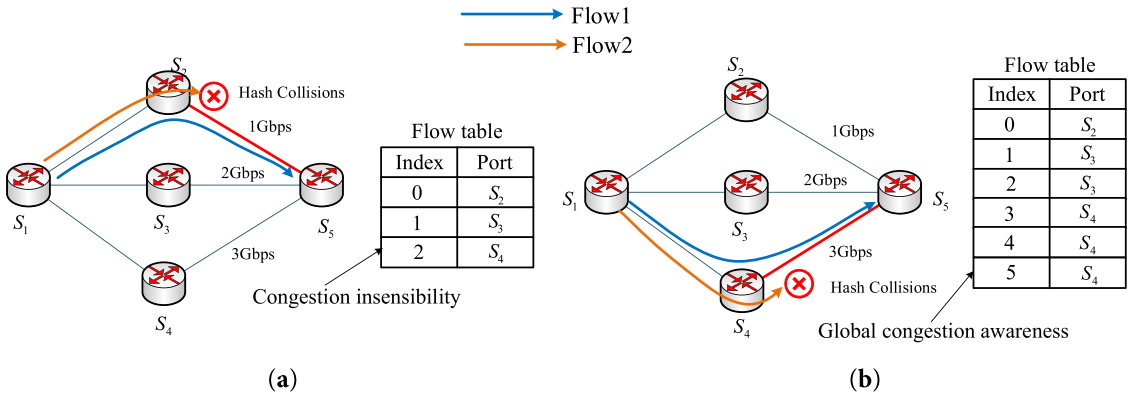

Figure 1: An example of ECMP and WCMP load balancing. (**a**) ECMP (congestion insensibility); (**b**) WCMP (global congestion awareness).

To overcome the limitations of ECMP and WCMP, ConFlet combines centralized global decision-making and distributed congestion awareness to implement flowlet-level load balancing. Specifically, as shown in Fig. 2a, ConFlet inherits the advantages of WCMP global congestion awareness to set different weights for paths. To avoid hash collisions between flows in the network, ConFlet can track the packet arrival interval of each flow and the load difference of each path in real time, and then dynamically split flows into multiple flowlets along different paths to achieve more fine-grained load balancing. Besides, suppose the worst thing is that link congestion still occurs due to traffic bursts or hash conflict, as shown in Fig. 2b, even if the controller re-plans the path, it will greatly increase the transmission delay of packets, which is unacceptable for flows that are sensitive to delay and jitter. ConFlet designs a distributed congestion-awareness mechanism on the data plane. The switch can locally detect congestion and quickly switch paths, which can react to congestion in microseconds.

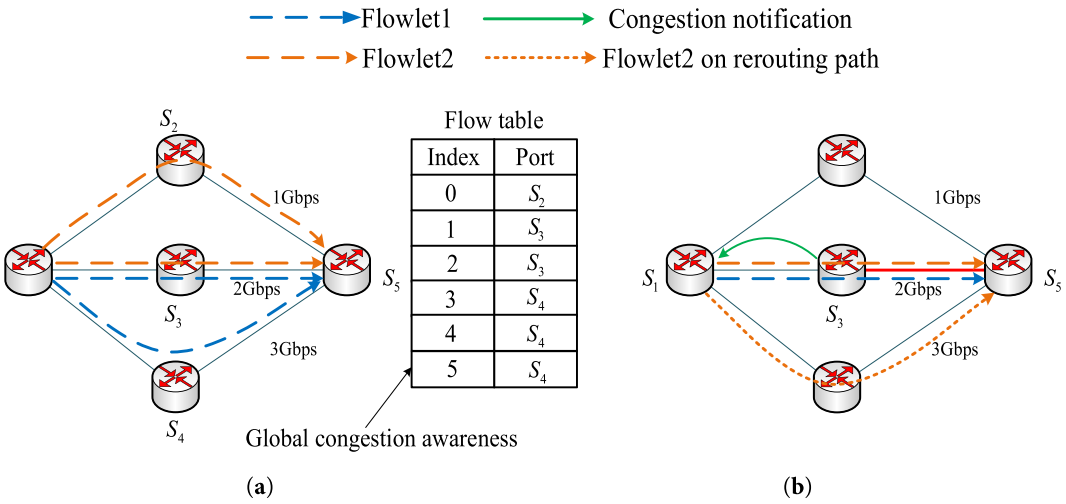

Figure 2: Core idea of ConFlet. (**a**) Flowlet-level load balancing; (**b**) Distributed congestion avoidance.

4 System Design

4.1 Global Routing Decision in the Control Plane

4.1.1 Network Model

The whole network uses a centralized SDN controller to maintain a global network view and implement centralized multipath routing planning. The network topology can be described as an undirected graph $G(N, E, B_e)$, where N represents the network node set, E represents the link set, and B_e indicates the bandwidth of each link. To better aggregate traffic demand, we introduce the concept of a *flow group*. A flow represents a continuous sequence of packets carrying matching quintuple attributes, including source IP address, destination IP address, transport protocol, source port, and destination port. A flow group is a collection of flows with the same source switch and destination switch. Therefore, a flow group contains a large class of flows, which often share the same network path. For a given flow group ψ, it can be expressed as $\{s_\psi, d_\psi, \beta_\psi\}$. s_ψ, d_ψ, and β_ψ are the source switch, destination switch, and aggregate bandwidth. Table 2 shows the key mathematical notations of the network.

Table 2: Mathematical notations and descriptions.

Notation	Description
$G(N, E, B_e)$	Network topology
N	Set of nodes in the network
E	Set of links in the network
B_e	Total bandwidth of link e
t_e	Delay of link e
d_e	Length of link e
D	The maximum distance between any two nodes in the network
R_i	Reliability of node i
R_e	Reliability of link e
$\psi: \{s_\psi, d_\psi, \beta_\psi\}$	A flow group
s_ψ	Source switch of flow group ψ

(Continued)

Table 2 (continued)

Notation	Description
d_ψ	Destination switch of flow group ψ
β_ψ	The bandwidth requirement of flow group ψ
δ	The delay threshold
P_i^{lost}	Packet loss rate of node i
C_i	Processing capacity of node i
l_i	Load of node i
U_i	Number of forwarding errors on node i per unit time
ϑ_e	Residual bandwidth of link e
P_ψ	Set of paths of flow group ψ
$R_p, p \in P_\Psi$	Reliability of path p
x_p^e	Binary variable: $x_p^e = 1$ if link e on path p, $x_p^e = 0$ otherwise
$y_{p,\Psi}$	Path weight of flow group ψ on path p

We define that the reliability of a node R_i is related to the packet loss rate P_{lost}^i, load l_i and the number of forwarding errors per unit time U_i of the node i, as shown in Eq. (1). The higher the load of the node, the more likely it is to become the target of a saturation attack. The number of forwarding errors can be obtained by performing the forwarding verification mechanism to audit the actual forwarding behavior of the switch [18]. A node with more forwarding errors is more likely to become a faulty or malicious node.

$$R_i = \left(1 - P_{lost}^i\right) \cdot \left(\frac{C_i - l_i}{C_i}\right)^{U+1} \tag{1}$$

The reliability of a link R_e is defined as a function of the available bandwidth ϑ_e, delay t_e, and length d_e of the link e. The available bandwidth and delay of the link directly determine the QoS of the network. Besides, according to the availability model of a bidirectional line proposed in [19], for optical fiber cables, owing to the fact that the cable is cut frequently and the maintenance time is very long, the availability of a line depends on its length. Therefore, we normalize the available bandwidth, delay and length, respectively, and then calculate the link reliability by Eq. (2). The three parameters α, β and k are defined to tune the impact of the factors on the reliability.

$$R_e = \left(\frac{\vartheta_e}{B_e}\right)^\alpha \left(1 - \frac{t_e}{\delta}\right)^\beta \left(1 - \frac{d_e}{D}\right)^k \tag{2}$$

Assuming that the forwarding path of a flow can be expressed as $p = \left\{\cdots, s_i, e_{ij}, s_j, \cdots\right\}$, the reliability of a path is calculated based on the reliability of nodes and links on this path, which is calculated as shown in Eq. (3).

$$R_p = \prod_{i \in p} R_i \prod_{e \in p} R_e \tag{3}$$

Our objective function is a measure of the reliability of the paths chosen for routing the flows belonging to a flow group. For the whole network, the goal is to maximize the service reliability of the entire network while satisfying the QoS requirements of all flow groups. Given a flow group ψ with bandwidth requirement

β_ψ, the objective function is to calculate a set of paths P_ψ and determine the path weights $y_{\rho,\psi}$ of each path to route these flows along the most reliable paths. Hence, the objective function is to maximize the weighted path reliability, which is defined as the reliability of the paths chosen for routing, weighted by the bandwidth of the flow group routed along these paths, i.e.,

$$Maximize \qquad \sum_{\forall \psi} \sum_{p \in P_\psi} R_p y_{p,\psi} \beta_\psi \tag{4}$$

where R_p is the reliability of path p and $y_{p,\psi}\beta_\psi$ is the bandwidth of the flow group routed on path p.

(1) Delay constraint: for any path P_ψ to be chosen for routing flow group, the sum of transmission delay of all links is less than the upper bound delay threshold δ.

$$\sum_{p \in P_\psi} t_e x_p^e \le \delta \qquad \forall e \in E \tag{5}$$

(2) Bandwidth constraint: the bandwidth constraint guarantees that all forwarding links selected for path calculation retain adequate transmission capacity. Assume the total bandwidth demand of a specific flow group ψ is defined as a fixed variable β_ψ. For any physical link e assigned to forward packets belonging to this flow group, its remaining available bandwidth resource must be enough to satisfy the transmission demands brought by the newly scheduled flow group.

$$\sum_{\forall \Psi} \sum_{\forall p} \beta_\psi \odot y_{p,\psi} x_\rho^e \le B_e, \qquad \forall e \in E, \forall p \in P_f \tag{6}$$

(3) Reliability constraint: the reliability of all nodes and links constructing the path should not be lower than the given lower threshold R_{lower}^v and R_{lower}^e, that is, when the reliability of nodes or links is lower than R_{lower}, it should be considered as malicious nodes or links, which should be isolated from the network.

$$R_i \ge R_{lower}^v \qquad \forall i \in P_\psi \tag{7}$$

$$R_e \ge R_{lower}^e \qquad \forall i \in P_\psi \tag{8}$$

4.1.2 Algorithm Design

To quickly solve this multi-path routing problem, the SDN controller pre-calculates a set of paths P_ψ between each source-destination pair and runs a solving algorithm for each flow request. As a result, the high-performance SDN controller installs forwarding rules on SDN enabled nodes through secure channels based on the optimal solution for the flow group ψ with the parameters $\left(P_\psi, y_{\rho,\psi}\right)$. We assume that graph G is m-connected (m ≥ 2); otherwise, it cannot satisfy the reliability constraints. We note that the nonlinear objective function in (4) can be linearized using the standard linearization technique described in [20]. The linear optimization program is solved using the Linux GLPK solver on the SageMath platform. The sub-flows must be routed along path p such that the nodes $i \in N$ and links $e \in p$ on the path p have maximum reliability (R_p). Consequently, the time complexity of this linear programming is given by the order $\mathrm{O}\left(E^2 \cdot P_\psi\right)$, where E is the total number of edges in the network and P_ψ is the total number of paths between given source–destination pairs. However, as the network size increases, it is computationally infeasible to obtain the optimal solution within a reasonable time, as each source-destination pair has an exponential number of simple paths between them. Therefore, we developed a fast heuristic algorithm to solve the multi-path routing problem. Algorithm 1 gives the multipath routing algorithm based on reliability evaluation.

Algorithm 1: Multipath routing algorithm based on reliability evaluation

Input: Network topology $G(N, E, B_e)$, Flow group set $\psi : \{s_\psi, d_\psi, \beta_\psi\}$
Output: Multipath set P_ψ^K and the path weight of each flow group on each path $y_{p,\psi}$
Initialize multipath set P_ψ^K
for $i \in N$ **do**
    $R_i = \left(1 - P_{lost}^i\right) \cdot \left(C_i - l_i/C_i\right)^{U+1}$
end for
for $e \in E$ **do**
    $R_e = \left(\vartheta_e/B_e\right)^\alpha \left(1 - t_e/\delta\right)^\beta \left(1 - d_e/D\right)^k$
end for
$G'(V', E') = Isolate((G'(V', E'))$
$\Omega = SPF(G'(V', E'))$
for $path \in P_{\psi_i}$ **do**:
    if $\vartheta_{path} < \beta_{\psi_i}$ and $D_{path} < \delta$ **then**
        $P_{\psi_i} \leftarrow path$;
    end if
end for
for $\psi_i \in \Psi$ **do**:
    for $p \in P_{\psi_i}$ **do**:
        $R_p = \prod_{i \in p} R_i \prod_{e \in p} R_e$,
    end for
end for
$P_{\psi_i} = sort\left(P_{\psi_i} \leftarrow R_p\right)$
$P_\psi^K = min\_link\_disjoint\left(P_\psi, K\right)$
$y_{p,\psi} = R_p / \sum_1^K R_P$

When implementing multipath routing, the controller uses an in-band telemetry technique to inject probes into the network regularly, collects network status information including topology changes, packet loss rate and load of nodes, available bandwidth and delay of links, and also audits the actual forwarding behavior of switches. Then the controller will evaluate the node reliability and link reliability to calculate the K most reliable paths to ensure that the flows are always transmitted along the K most reliable paths. The core idea of the multipath routing algorithm is to evaluate the reliability of nodes and links to remove malicious nodes and links from the network topology $G = (V, E)$, and then form a new network topology $G' = (V', E')$. On this basis, for each flow group request, we improve the Dijkstra algorithm to calculate all end-to-end paths Ω. The algorithm filters the paths that meet the QoS from these reachable paths Ω to form a multipath set P_ψ according to the delay and bandwidth constraints. Then the algorithm will sort the paths according to the path reliability to select K paths with link-disjoint paths P_ψ^K. Finally, the algorithm arranges the flow group in descending order according to bandwidth requirement, and gives priority to the flow group with high bandwidth requirement to implement multipath weighted forwarding, where the weight is proportional to the reliability of the path.

4.1.3 Algorithm Overhead Analysis

The overhead of the multipath calculation algorithm comes from two parts: one is the complexity of the algorithm itself, and the other is additional control overhead incurred to implement the path calculation algorithm.

For the complexity of the algorithm itself, we adopt an improved Dijkstra algorithm. Assuming there are n nodes and m edges in the network, the initial number of paths calculated by the improved Dijkstra algorithm is L, the number of flows in the network is F during peak traffic, the complexity of the improved Dijkstra algorithm is $\mathrm{O}\left(FLn\left(m+nlogn\right)\right)$. After the initial path calculation is completed, the algorithm needs to traverse L paths for each flow and filter out K paths that meet the Quality of Service, the complexity of the process is $\mathrm{O}\left(FL\right)+\mathrm{O}\left(FK\right)$. Therefore, the total algorithm complexity is $\mathrm{O}\left(FLn\left(m+nlogn\right)\right)$ $+\mathrm{O}\left(FL\right)+\mathrm{O}\left(FK\right)$. Due to the fact that path calculation algorithms mainly run on remote controllers, they can be fully applied to real-time computing.

For additional control overhead, it mainly includes the collection of full dimensional network state information, such as packet loss rate, node load, the number of forwarding errors, link bandwidth, link delay, link length, etc. To achieve better network optimization decisions, we need to evaluate the reliability of nodes and links to find the most reliable K path forwarding, the additional control overhead is necessary. To minimize the network control overhead as much as possible, we adopt an in-band telemetry mechanism to encapsulate the network state information inside the packet, avoiding a large amount of network probe.

4.2 Distributed Control in the Data Plane

4.2.1 Adaptive Flowlet-Level Forwarding Mechanism

In Section 4.1, we use a flow group to count network traffic demand and calculate the path weight of each flow group on multiple paths to maximize service reliability. In the actual forwarding of the data plane, the switch needs to deal with packet forwarding and flow splitting across multiple paths from one flow group.

Flowlet switching has been proven to be a fine-grained load balancing. A smaller timeout will easily lead to packet out-of order, while a larger timeout will affect the load balancing rate. Therefore, an appropriate flowlet timeout should be slightly greater than the maximum delay difference of multiple paths to ensure that the subsequent packets will not arrive at the destination earlier than the previous packets even if they are forwarded along the optimal path. However, the transmission delay of each path is not static, it varies greatly due to differences in link quality and traffic load. Hence, we propose an adaptive flowlet-level forwarding mechanism. It tracks the delay change of each path in real time, then performs dynamic flow splitting to achieve a better load balancing rate and minimize packet out-of-order. ConFlet is cost-efficient and topology-independent, which adopts an active probing technique to obtain path delay accurately in a distributed way.

Fig. 3 shows the adaptive flowlet-level forwarding mechanism. It dynamically updates the flowlet timeout by real-time monitoring the delay difference between paths, and changes the flow splitting granularity to perform path switching. The implementation process is as follows:

(1) According to the multipath set P_{ψ}^{K} and path weight $y_{p,\psi}$ calculated in Section 4.1 for routing each flow group, the controller realizes the path weight by creating multiple rules for the same output port in the flow table.

(2) For each arriving flow, the ingress switch maintains a separate timeout threshold for each flow, records each packet arrival time of each flow, and evaluates the packet arrival interval. When the packet arrival interval exceeds the timeout threshold T_{th}, it will be considered as a new flowlet, and updates the *flowlet_id* for flow path switching.

(3) During the flow forwarding process, the ingress switch monitors the delay difference of multiple paths in real time, records the maximum delay $T_{\max}$ and the minimum delay $T_{\min}$, and sets the flowlet timeout threshold to $T_{th} = T_{\max} - T_{\min} + \varepsilon$. ε is the burst tolerance factor, the flowlet timeout threshold should be greater than or equal to ε.

(4) Finally, the switch maps different flowlets to multiple paths for forwarding by hashing over header fields (e.g., source IP, destination IP, source port, destination port, protocol) and the flow splitting flag *flowlet_id*.

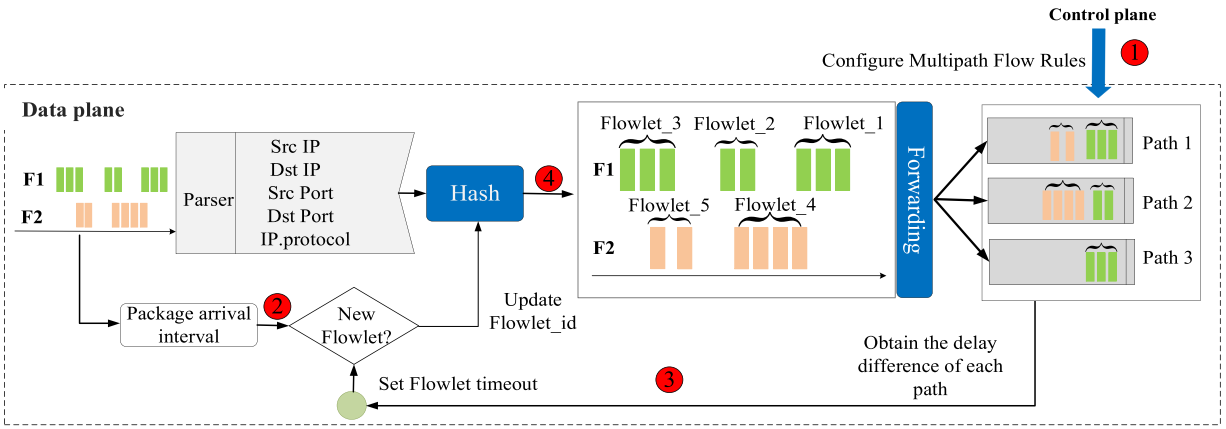

Figure 3: Adaptive flowlet-level forwarding mechanism.

As shown in Fig. 4, to obtain the delay difference on multiple paths synchronously, for multiple paths of each flow, the ingress switch will periodically clone the packet digest of the corresponding flow as the probe packet and forward it from the output port of multiple paths in multicast mode. The probe header is shown in Table 3, including the field *flow_ID* to identify the flow, two timestamp fields *start_time* and *finish_time* to record the start time and the finish time of the probe for calculating the path delay. The flag field *bos* is used to distinguish the ingress switch, intermediate switch, or egress switch. The switch processing logic is shown in Algorithm 2. For each flow, the ingress switch will record the arrival interval of each packet online and compare it with the *flow_timeout* to determine whether to mark the flowlet and switch paths (steps 3~6). At the same time, it maintains two registers *Max_delay_register* and *Min_delay_register* to record the maximum and minimum delay of multiple paths. The ingress switch periodically clones the packet digest as the probe at the egress stage and sends the probe back to the ingress stage through the *recirculate(.)* function (step 24~28). Then the ingress switch first writes the current timestamp to *start_time* and forwards along different paths in multicast mode (step 7~10). The intermediate switch only performs normal forwarding operations. When the probe is forwarded to the egress switch (step 29~33), the egress switch sends the probe back to the ingress stage through *recirculate(.)*, writes the current timestamp to *finish_time*, exchanges the source and destination IP addresses of the probe, and returns the probe along the original path (step 19~22). After receiving the returned probes, the ingress switch will compare the path delay from each probe $T_{path} = finish\_time - start\_time$ with the value $T_{\max}$ and $T_{\min}$ in registers *Max_delay_register* and *Min_delay_register*. If $T_{path} > T_{\max}$, the value T_{path} will be updated to register *Max_delay_register*. If $T_{path} < T_{\min}$, the value T_{path} will be updated to register *Min_delay_register*. Otherwise, the value in the register is not updated (step 11~18). When collecting the probes of all paths, the ingress switch will calculate the delay difference between the best path and the worst path, and set the timeout threshold to $T_{th} = T_{\max} - T_{\min} + \varepsilon$ (step 18).

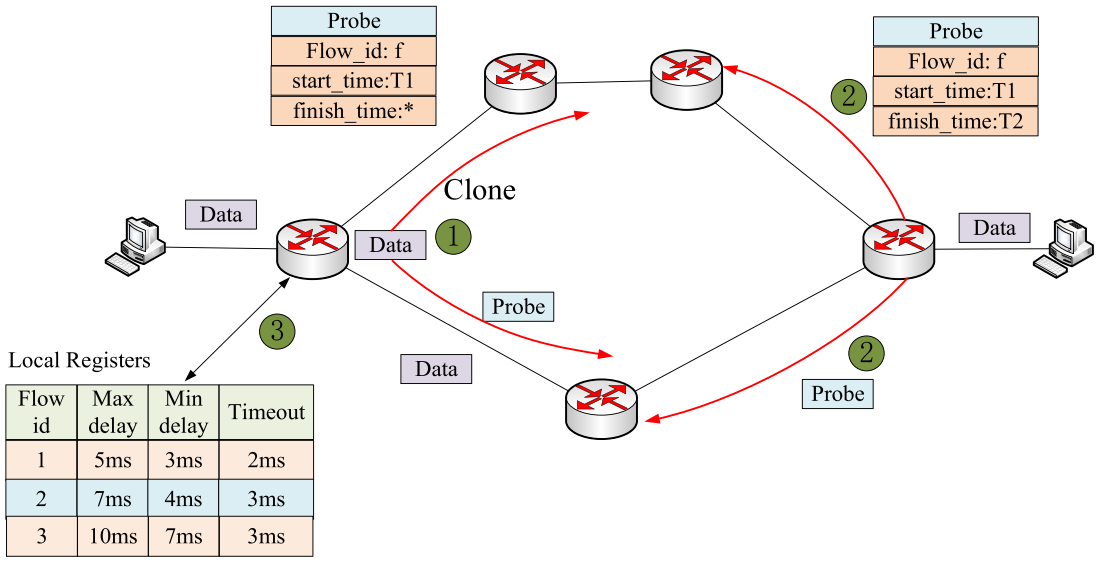

Figure 4: The process of obtaining delay difference on multiple paths.

Table 3: The header format of the probe packet.

Notation	Description
bit<16> flow_id	Identifier of the flow
bit<48> start_time_stamp	Time stamp to start forwarding
bit<48> finish_time_stamp	Time stamp to finish forwarding
bit<8> bos	Mark of probe round-trip process
bit<16> protocol_id	Internal protocol type

According to reference [21], assuming two paths P_1 and P_2 with bottleneck capacities B_1 and B_2, consider n flows transmitted on these two paths, with n_1 flows on path P_1 and n_2 flows on path P_2, packet arrivals from flow i occur as a Poisson process with rate λ_i, independent of all other flows. Let $P^j_{n_1,n_2}$ be the transition probabilities from (n_1, n_2) to $(n_1 - j, n_2 + j)$ and $(n_1 + j, n_2 - j)$, the transition probabilities $P^j_{n_1,n_2}$ can be further approximated as

$$P^j_{n_1,n_2} \approx \frac{B_j}{2\left(B_1 + B_2\right)} e^{-\left(B_j/n_j\right)\varepsilon} \tag{9}$$

According to Eq. (9), for $B_1 = 40$ Gbps, $B_2 = 10$ Gbps and $n = 100$, the ideal operating point in this case is (80, 20). For small ε (30 and 50 μs) and large ε (900 μs), the state distribution is far from ideal. However, for moderate values of ε (300 and 500 μs), the distribution concentrates around the ideal point. Hence, we set a flowlet timeout around 500 μs to support the largest range of flow rates.

Algorithm 2: Multipath load balancing forwarding

```
Input: pkt from flow
Function Ingress(pkt):
    if pkt is data or probe packet then
        if (current_time-flowlet_last_stamp[flow_hash]) > flowlet_timeout then
            update_flowlet_id();
        forwarding action (set_nhop, ecmp_group);
        node_type.apply();
    if pkt is recir_packet && ! hdr.probe.isValid() then
        set hdr.probe.setValid();
        probe.start_stamp←current_timestam;
        meta.mcast_grp←meta.ecmp_group_id
    if pkt is normal_probe && pkt.egress_type == host && pkt.probe.bos == 1 then
        path_delay ←pkt.start_time_stamp − pkt.finish_time_stamp;
        if path_delay > T_max then
            T_max ← path_delay;
        if path_delay < T_min then
            T_min ← path_delay;
        if probe_count == path_count then
            flowlet_timeout←T_max − T_min;
    if pkt is recir_probe && pkt.egress_type == host && pkt.probe.bos == 1 then
        hdr.ipv4.srcAddr⇔hdr.ipv4.dstAddr;
        probe.finish_stamp←current_timestam;
        egress_spec = ingress_port;
Function Egress (pkt):
    if ecmp_flag == 1 then:
        if (current_stamp-probe_generate_stamp[flow_hash]) > Update_interval then
            clone.pkt.digest;
        if pkt.type == egress_clone then
            recirculate(pkt);
        if pkt is probe then
            if pkt.egress_type == host && pkt.probe.bos == 0 then
                set pkt.probe.bos = 1;
                recirculate(probe);
```

4.2.2 Distributed Congestion Avoidance Mechanism

When the congestion control mechanism of traditional Internet such as TCP encounters link congestion, the switch usually notifies the terminal of congestion information in the form of packet loss, and the terminal eventually reduces the sending rate to alleviate congestion. This congestion feedback mechanism needs at least one round-trip time to work, which is not suitable for delay-sensitive flows. To prevent the network connection from being interrupted due to the packet transmission timeout or packet loss caused by the instantaneous link congestion, ConFlet uses a distributed congestion avoidance mechanism to perform local path switching when encountering link congestion.

To realize local rerouting of the switch, for each flow group ψ, according to the multipath set P_{ψ}^{K} calculated in Section 4.1, the controller will form a multipath decision tree to calculate the intersection nodes

of multiple paths V^i, and select these intersection nodes V^i as rerouting nodes to make multiple paths become backup paths for each other. As shown in Fig. 5, the multipath decision tree has three reachable paths, where $V^i = \{S_1, S_2\}$. Distributed congestion avoidance logic in the switch is shown in Algorithm 3. The local detection module of the switch will monitor the queue status information in real time, when some flows are experiencing congestion, it will clone the packet digest and form a congestion notification message to the upstream switch (step 12~22). After receiving the congestion notification message, the rerouting node will automatically switch the matching rules to adjust paths in advance to avoid congestion (step 6,7). Finally, the congestion notification message will be fed back to the ingress switch, and the ingress switch will perform flowlet-level path switching by updating *flowlet_id* (step 8~10). As shown in Fig. 6, when switch s_3 detects link congestion, it will notify the upstream reroute node s_2 of the congestion message (i.e., first-level congestion notification), then switch s_2 will switch the flow to the path $s_2 - s_6 - s_4$. If the link $s_6 - s_4$ is also congested, the switch s_6 will generate the congestion message (i.e., second-level congestion notification) to the upstream switch s_2, and feed back to the upstream reroute node s_1. Finally, switch s_1 will adjust the flow to path $s_1 - s_7 - s_8 - s_4$. To avoid too many congestion messages generated by a link congestion event, the congestion switch generates a congestion message every 100 μs until the link congestion is relieved.

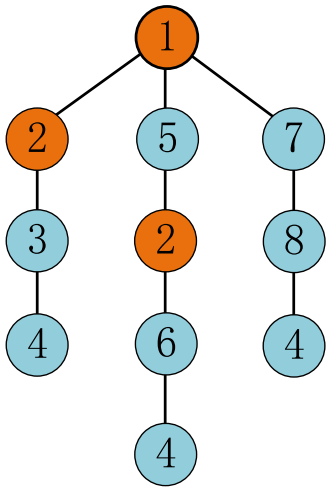

Figure 5: The multipath decision tree.

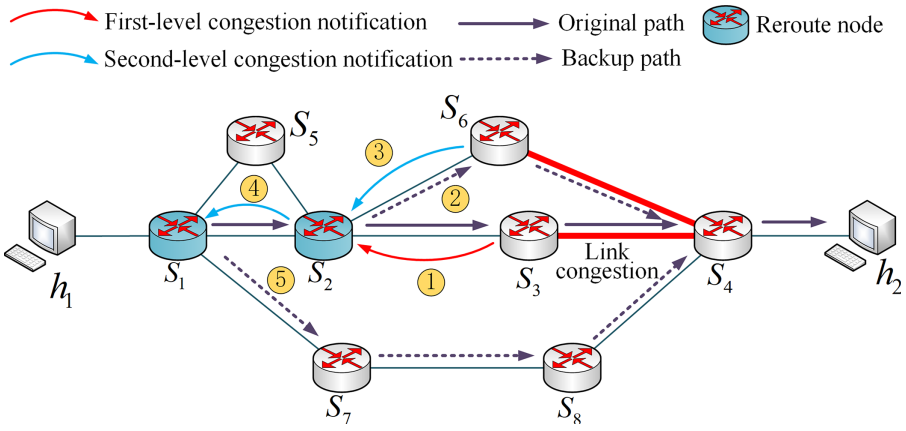

Figure 6: Distributed congestion avoidance process.

The following describes the conditions to generate congestion notification messages. Assuming that C is the total queue depth and c is the current queue depth, the switch maintains a tolerance factor k, that is, when the queue length $c > kC$ is satisfied, the switch will be considered as queue congestion. To avoid excessive congestion avoidance operations caused by micro-burst, the switch will not immediately trigger congestion notification, but set a time interval $T_{interval}$ for temporary queue congestion, if the queue depth converges to $c < kC$ in the future $T_{interval}$ time units, the queue congestion will be relieved, if the queue length is maintained $c > kC$ by more than $T_{interval}$ time units, the switch will trigger the congestion notification message. Besides, to prevent rerouting all flows to cause congestion on new paths, the rerouting node will selectively route some flows with probability δ, where the probability δ is related to the congestion degree of the link, which is calculated by Formula (10). The more serious the queue congestion, the greater the probability δ. To quickly respond to link congestion, we set the threshold k to 0.8. On the one hand, $k = 0.8$ can leave room for queue overflow, triggering congestion avoidance before queue overflow and preventing packet loss. On the other hand, $k = 0.8$ can avoid congestion and excessively increase probe overhead.

$$\delta = \frac{c - kC}{(1-k)\,C} \tag{10}$$

Algorithm 3: Distributed congestion avoidance

```
Input: pkt from flow
Function Ingress (pkt):
    if (pkt.type == INGRESS_RECIRC) then
        hdr.ipv4.srcAddr ⇔ hdr.ipv4.dstAddr;
        hdr.ethernet.etherType = TYPE_FEEDBACK;
    apply[forwarding←action (set_nhop, ecmp_group)];
    if type_feedback && meta.node_type == Reroute_switch then
        Start backup path;
    if type_feedback && meta.node _type == Ingress_switch then
        update_flow_seed();
        drop();
Function Egress(pkt):
    if pkt.enq_qdepth
        register timer.write()←last_stamp = 0;
    if pkt.enq_qdepth ≥ kC then
        last_stamp ←register timer.read();
            if last_stamp then
                if (current_stamp-last-stamp)> T_interval then
                    probability = (pkt.enq_qdepth − kC) ∗ 100/ (1 − kC);
                    backoff = random(0, 100);
                if backoff ≤ probability then
                    generate feedback-probe ← clone.pkt.digest;
                    recirculate(probe);
    else:
        register timer.write()←last_stamp = current_stamp;
```

5 Performance Evaluation

We evaluate ConFlet compared with the state-of-the-art load balancing schemes in terms of average service reliability, average network throughput, and average packet delay. Furthermore, we conduct experiments to validate the stable performance of ConFlet over various network architectures, which consist of both symmetric and asymmetric topologies.

5.1 Simulation Setup

Experiment Platform: The test environment is built on Mininet [22] with BMv2 software switches [23]. The simulation uses the Fattree (symmetric) topology (80 nodes and 256 links) and Interllifiber (asymmetric) topology (73 nodes and 93 links) taken from the Internet Topology Zoo [24]. In terms of the Interllifiber topology, link distances are evaluated using the geographic positions of interconnected nodes. For Fat-tree, link lengths are randomly generated within a certain range (10, 100) and the value D is set to 1000. We randomly select multiple host pairs to generate flows. The packet size range of the flow is 10 Kb ~10 Mb. All flows from the same source switch to the same target switch belong to a flow group. For every flow, the time interval between incoming packets complies with exponential distribution characteristics. Shorter packet intervals indicate heavier traffic loads. We used Python scripting tools to build controller entities on the SDN control plane. The controller implements the multipath forwarding of flow in real time through P4 Runtime API [25].

Scheme Comparison: (1) ECMP [3]: the flow-granularity multipath routing scheme, which distributes different flows evenly to multiple paths through a hash function, but the load effect is limited in networks with unevenly large and small flows. (2) WCMP [4]: weighted multipath routing scheme, which sets different weights for each path according to the available bandwidth of each path. (3) LetFlow [17]: the flowlet-granularity load balancing scheme, which sets a fixed timeout threshold of 50 μs to split flows into flowlets. (4) ConFlet: our proposed multipath routing scheme based on reliability evaluation and congestion awareness.

5.2 Network Performance Evaluation

5.2.1 Average Service Reliability

To quantify the average service reliability of ConFlet, given a flow f with a bandwidth requirement of β_f, the calculated path set is P_f, the reliability of each path is R_p, and the path weight of the flow on the path p is $y_{p,f}$, then the average service reliability is defined as $\sum_{\forall f,\forall p\in P_f} R_p \beta_f y_{p,f} / \sum_{\forall f} \beta_f$. During the network experiment process, we randomly select host pairs to generate the traffic as the background traffic, and also inject invalid traffic and error rules into some nodes to simulate nodes with different reliability values. Fig. 7 presents a horizontal comparison of average service reliability metrics corresponding to the four tested schemes. Observations from the plotted curves reveal that ECMP yields the minimum average service reliability value, succeeded by WCMP and LetFlow in descending order, whereas ConFlet obtains the optimal reliability performance. When the network load grows higher, the performance disparity between ConFlet and the other two benchmark schemes (WCMP and LetFlow) becomes increasingly significant. Compared with LetFlow, ConFlet can improve the average service reliability by about 32%~47% under high load (90%). ECMP randomly distributes the flows to any path without sensing the node and link congestion status, which aggravates the reliability risk. WCMP and LetFlow improve the load balancing effect and service reliability by dynamically setting the link weights and adjusting flowlet timeout, respectively. The proposed ConFlet mechanism conducts comprehensive reliability assessment for all network nodes and transmission links within the system. Once faulty or malicious network entities are detected, ConFlet will isolate these abnormal nodes and links from the forwarding topology to achieve superior average service reliability compared with alternative scheduling strategies.

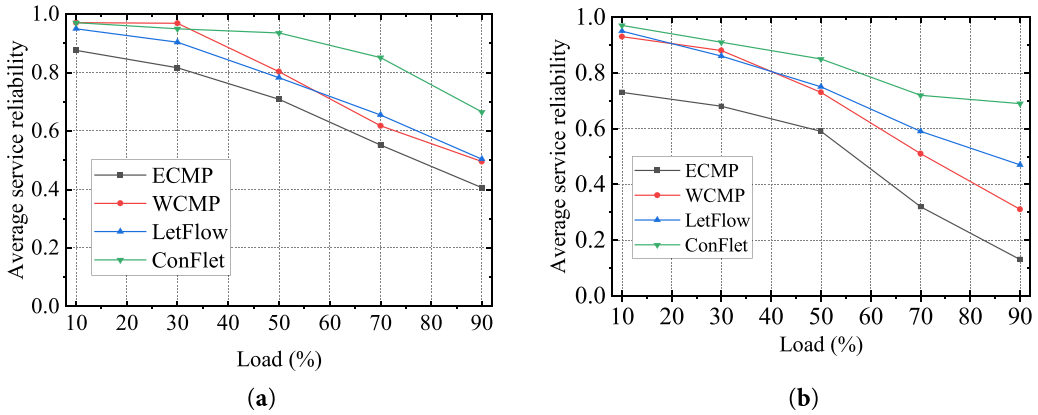

Figure 7: Average service reliability. (**a**) Fat tree topology; (**b**) Interllifiber topology.

5.2.2 The Average Throughput and Packet Delay

Fig. 8 shows the average throughput comparison under different loads in Fattree (symmetric) and Interllifiber (asymmetric) topology. To better display the performance comparison, we normalized the throughput to ConFlet. It can be seen that ConFlet has higher throughput than WCMP and LetFlow, while ECMP obtains the worst performance. WCMP sets different weights for the path according to the available bandwidth of the link and LetFlow can divide the large flows into small flows by monitoring the arrival intervals of packets, which has a certain effect of load balancing. However, WCMP still has the problem of hash conflict. LetFlow sets a fixed timeout threshold, which still leads to partial path congestion. As the load increases, the gap between ConFlet and LetFlow becomes more obvious. Compared with LetFlow, under high load (90%), ConFlet improves throughput by about 15% and 45% in symmetric and asymmetric topologies, respectively. ConFlet, on the one hand, realizes weighted forwarding according to the reliability of each path, implying the effect of load balancing. On the other hand, it implements flowlet-level load balancing based on adaptive flowlet timeout, which achieves the highest throughput.

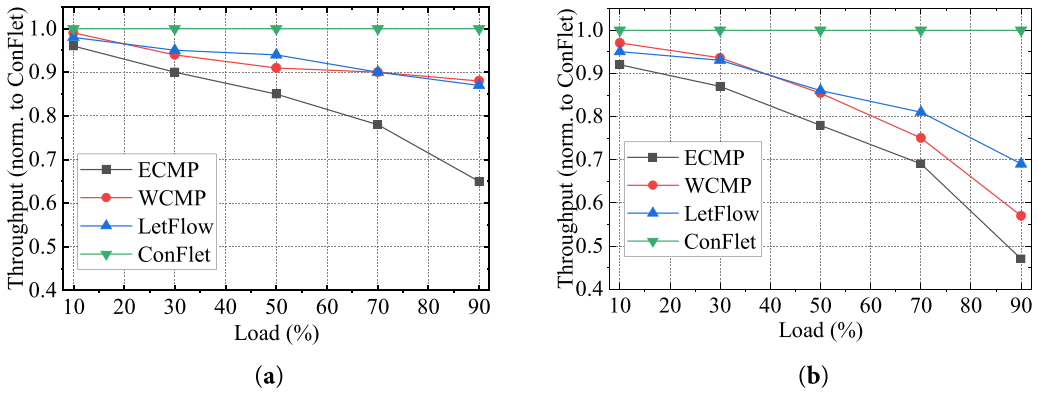

Figure 8: Average throughput comparison. (**a**) Fat tree topology; (**b**) Interllifiber topology.

Fig. 9 shows the average packet delay comparison under different loads. Average packet delay increases with the increase of traffic load. Since ConFlet, WCMP and LetFlow have finer granularity and fewer hash conflicts, their performance is also significantly better than ECMP. Compared with LetFlow, ConFlet reduces the average packet delay by about 25% and 53% in symmetric and asymmetric topologies, respectively, under high load (90%). In terms of throughput and delay, ConFlet has more advantages in asymmetric topology.

This is because the asymmetric topology has fewer equivalent multipaths than the symmetric topology, which can better reflect the advantages of ConFlet's load balancing and congestion avoidance mechanisms.

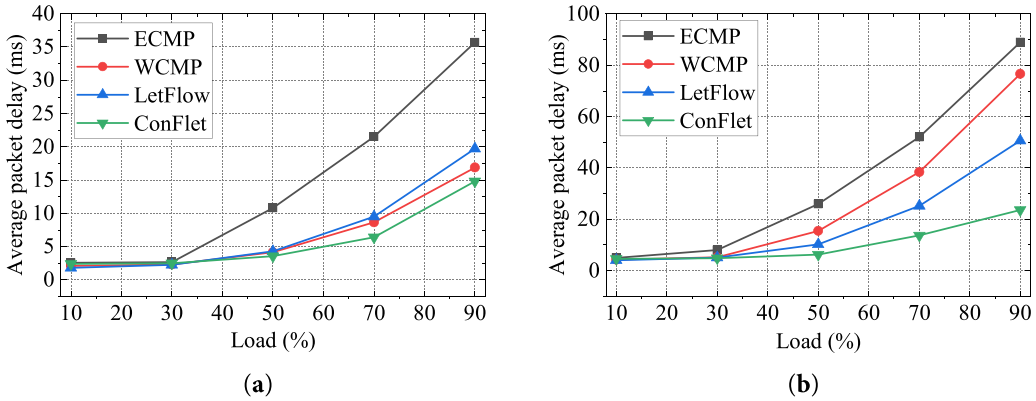

Figure 9: Average packet delay comparison. (**a**) Fat tree topology; (**b**) Interllifiber topology.

6 Conclusion

In this paper, we design ConFlet, a general load balancing scheme that can work seamlessly on any network topology. ConFlet takes full advantage of centralized and distributed control to achieve efficient and fine-grained load balancing. Specifically, in the control plane, ConFlet exploits the global network view to evaluate the reliability of nodes and links, and performs multipath routing decisions to guarantee that flows are always transmitted on a set of reliable paths. In the data plane, ConFlet can track the delay difference of each path and the packet arrival interval of each flow in real time, and set the adaptive timeout to split flows into flowlets along multiple paths. In addition, ConFlet adopts a distributed congestion avoidance mechanism to flexibly switch paths according to the congestion feedback, which can react to congestion in microseconds. Finally, we conduct simulation results to evaluate ConFlet against existing load balancing schemes. The experimental results show that, compared to LetFlow, ConFlet improves average service reliability by 47% and network throughput by 45%, reduce average packet delay by 53% under high load (90%) on asymmetry topology. For future research, we will deploy ConFlet in hardware devices and verify its performance in realistic large-scale networks.

Acknowledgement: Not applicable.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: Ziyong Li: data curation; formal analysis; investigation; methodology; software; validation; writing—original draft. Yusheng Xia: formal analysis; validation; project administration. Junfei Li: data curation; formal analysis; investigation. Le Tian: conceptualization; supervision. Xinglong Pei: methodology; software; validation; visualization. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The performance data generated or analyzed in support of this study are available from the corresponding author upon reasonable request.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Prabha R, G. A S, Bharathi GP, Sridevi S. Hybrid multipath routing cluster head prediction based on SDN-enabled IoT and heterogeneous context-aware graph convolution network. Peer Peer Netw Appl. 2024;17(4):2016–30. doi:10.1007/s12083-024-01685-z.
2. Li H, Yang X, Wei G, Li K, Zeng W, Huang R, et al. An evolutionary network layer for future network MIN architecture. IEEE Trans Netw Sci Eng. 2026;13(2):360–75. doi:10.1109/tnse.2025.3583568.
3. Rhamdani F, Suwastika NA, Nugroho MA. Equal-cost multipath routing in data center network based on software defined network. In: Proceedings of the 2018 6th International Conference on Information and Communication Technology (ICoICT); 2018 May 3–5; Bandung, Indonesia. p. 222–6. doi:10.1109/ICoICT.2018.8528730.
4. Zhou J, Tewari M, Zhu M, Kabbani A, Poutievski L, Singh A, et al. WCMP: weighted cost multipathing for improved fairness in data centers. In: Proceedings of the Ninth European Conference on Computer Systems. Amsterdam, The Netherlands: ACM; 2014. p. 1–14. doi:10.1145/2592798.2592803.
5. Kang N, Ghobadi M, Reumann J, Shraer A, Rexford J. Efficient traffic splitting on commodity switches. In: Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies. Heidelberg Germany: ACM; 2015. p. 1–13. doi:10.1145/2716281.2836091.
6. Kaur A, Rama Krishna C, Patil NV. A comprehensive review on Software-defined Networking (SDN) and DDoS attacks: ecosystem, taxonomy, traffic engineering, challenges and research directions. Comput Sci Rev. 2025;55(9):100692. doi:10.1016/j.cosrev.2024.100692.
7. Spina MG, De Rango F, Scalzo E, Guerriero F, Iera A. Distributing intelligence in 6G programmable data planes for effective in-network intrusion prevention. IEEE Netw. 2025;39(3):319–25. doi:10.1109/MNET.2025.3544828.
8. Al-Fares M, Radhakrishnan S, Raghavan B, Huang N, Vahdat A. Hedera: dynamic flow scheduling for data center networks. Nsdi. 2010;10(8):89–92.
9. Korenfeld E, Kampeas J, Gurewitz O. *ARION*: aggregated routing for in-order optimized network load balancing in data centers. IEEE Trans Netw. 2026;34:4496–509. doi:10.1109/TON.2026.3678315.
10. Bonato T, De Sensi D, Di Girolamo S, Bataineh A, Hewson D, Roweth D, et al. Flowcut switching: high-performance adaptive routing with in-order delivery guarantees. IEEE Trans Netw. 2026;34(2):1974–87. doi:10.1109/ton.2025.3636209.
11. Dou S, Qi L, Wang J, Guo Z. EPIC: traffic engineering-centric path programmability recovery under controller failures in SD-WANs. IEEE/ACM Trans Netw. 2024;32(6):4871–84. doi:10.1109/TNET.2024.3438292.
12. Farhan M, Shah N, Wang L, Muntean GM, Song HH. RDG-TE: link reliability-aware DRL-GNN-based traffic engineering in SDN. Expert Syst Appl. 2025;265(4):125963. doi:10.1016/j.eswa.2024.125963.
13. Cheng Y, Jia X. NAMP: network-aware multipathing in software-defined data center networks. IEEE/ACM Trans Netw. 2020;28(2):846–59. doi:10.1109/TNET.2020.2971587.
14. Yan J, Zhang H, Shuai Q, Liu B, HiQoS G X. An SDN-based multipath QoS solution. China Commun. 2015;12(5):123–33. doi:10.1109/CC.2015.7112035.
15. Alizadeh M, Edsall T, Dharmapurikar S, Vaidyanathan R, Chu K, Fingerhut A, et al. *CONGA*: distributed congestion-aware load balancing for datacenters. In: Proceedings of the 2014 ACM conference on SIGCOMM; 2014 Aug 17–22; Chicago, IL, USA. p. 503–14. doi:10.1145/2619239.2626316.
16. Katta N, Hira M, Kim C, Sivaraman A, Rexford J. HULA: scalable load balancing using programmable data planes. In: Proceedings of the Symposium on SDN Research; 2016 Mar 14–15; Santa Clara, CA, USA. p. 1–12. doi:10.1145/2890955.2890968.
17. Vanini E, Pan R, Alizadeh M, Taheri P, Edsall T. Let it flow: resilient asymmetric load balancing with flowlet switching. In: Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation; 2017 Mar 27–29; Boston, MA, USA.
18. Wu P, Shang Y, Bai S, Cheng L, Tang H. A lightweight path consistency verification based on INT in SDN. Math Biosci Eng. 2023;20(11):19468–84. doi:10.3934/mbe.2023862.
19. Zhao C, Li X, Qi S, Zhao Z, Qiao K, Li D, et al. A survey on reliability evaluation of optical networks. IEEE Commun Surv Tutor. 2026;28(3):3443–77. doi:10.1109/comst.2025.3621165.

20. Murali Mohan P, Gurusamy M, Lim TJ. Dynamic attack-resilient routing in software defined networks. IEEE Trans Netw Serv Manag. 2018;15(3):1146–60. doi:10.1109/TNSM.2018.2846294.
21. Liu WX, Cai J, Ling S, Zhang JY, Chen Q. QALL: distributed queue-behavior-aware load balancing using programmable data planes. IEEE Trans Netw Serv Manag. 2024;21(2):2303–22. doi:10.1109/TNSM.2023.3345862.
22. Sheikh MNA, Hwang IS, Raza MS, Ab-Rahman MS. A qualitative and comparative performance assessment of logically centralized SDN controllers via mininet emulator. Computers. 2024;13(4):85. doi:10.3390/computers13040085.
23. Elbediwy M, Pontikakis B, David JP, Savaria Y. Enabling rank-based P4 programmable schedulers: requirements, implementation, and evaluation on BMv2 switches. IEEE Trans Netw. 2025;33(1):299–310. doi:10.1109/TNET.2024.3481152.
24. Knight S, Nguyen HX, Falkner N, Bowden R, Roughan M. The internet topology zoo. IEEE J Select Areas Commun. 2011;29(9):1765–75. doi:10.1109/jsac.2011.111002.
25. Stubbe H, Gallenmüller S, Simon M, Hauser E, Scholz D, Carle G. Exploring data plane updates on P4 switches with P4Runtime. Comput Commun. 2024;225(C):44–53. doi:10.1016/j.comcom.2024.06.020.