TFPACO: Traffic Flow Planning System for Software-Defined Networks Based on Ant Colony Optimization Algorithm

Aleksandr Safonenko and Vladimir Korkhov(✉)

Saint Petersburg State University, 7-9 Universitetskaya Embankment, St. Petersburg 199034, Russia
st107536@student.spbu.ru, v.korkhov@spbu.ru

Abstract. The rapid growth of the Internet due to the COVID pandemic and the increasing need for video content are creating pressure on existing network infrastructure. Therefore, there is a need for efficient Quality of Service routing techniques. In this context, this paper presents a new algorithm for Traffic Flow Planning system in SDN based on Ant Colony Optimization (TFPACO). In this algorithm, unlike existing optimization techniques, mice and elephant flows are considered separately. Mice flows are optimized using a Dijkstra algorithm considering risk and delay while elephant flows use a multi-objective ant colony optimization model. As a result we get multiple solutions, and only nondominated solutions are stored in the archive. Among the solutions obtained in the archive, a set of solutions is chosen for proportional routing. In this context, Dijkstra's algorithm, multi-objective Dijkstra's algorithm, TSACO, and the Motlagh algorithm are used as reference algorithms. A discrete-time simulation environment with dynamic edge parameters is used to check the performance of the proposed algorithm. The simulation results show that the proposed algorithm entirely eliminates flow and demand blocking with less latency than existing algorithms.

Keywords: software-defined networking · ant colony optimization · multi-objective optimization · flow scheduling · quality of service

1 Introduction

Software-defined networking (SDN) has become one of the main architectural approaches for improving flexibility and controllability of modern communication networks. By separating the control plane from the data plane, SDN enables centralized management of routing decisions and provides a global view of the current network state [1,2]. This property makes SDN particularly suitable for traffic engineering and traffic management tasks, where route selection must adapt to varying load conditions and heterogeneous application requirements.

However, with the increasing volume of network traffic, there are many challenges for SDN traffic management. First, route selection is no longer a simple

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027
O. Gervasi et al. (Eds.): ICCSA 2026 Workshops, LNCS 16763, pp. 447–463, 2027.
https://doi.org/10.1007/978-3-032-30530-5_28

shortest path problem. In practice, route selection should be based on various network factors, including delay, reliability, and bandwidth [3]. Second, heterogeneous traffic flows cause conflicting quality-of-service (QoS) requirements. In particular, mice flows are usually short and delay-sensitive, whereas elephant flows require a lot of available bandwidth and may create local congestion if routed improperly [4,5]. Therefore, a good SDN traffic management algorithm should be able to handle the differences between various types of traffic and multiple quality-of-service requirements.

A large number of SDN traffic management methods have been proposed in recent years. Existing researches include single-objective delay-based routing, weighted multi-criteria routing, load-balancing methods, and routing algorithms based on swarm intelligence [2,4,6]. The classical Dijkstra algorithm is simple and computationally efficient. However, this algorithm does not take into account the remaining bandwidth and reliability. The multi-objective Dijkstra algorithm improves flexibility by combining several metrics into a single scalar weight [6]. Nevertheless, the results are heavily dependent on the weights used. In the TSACO algorithm, mice traffic and elephant traffic are managed separately using ant colony optimization and edge disjoint multipath [5]. The multi-objective ACO algorithm proposed by Motlagh et al. focuses on route construction with risk and cost criteria and produces nondominated solutions [7].

Other related studies in the context QoS routing in SDN of Internet of Things based on swarm intelligence have also been conducted. For example, bio-inspired routing based on ACO has been used to analyze delay, loss, jitter, and rule capacity constraints in QoS-aware routing [8]. These studies demonstrate the potential of ant colony optimization in traffic flow planning under varying network requirements.

In this paper, a new Traffic Flow Planning system based on Ant Colony Optimization (TFPACO) algorithm is proposed. The method combines Dijkstra algorithm for mice flows with multi-objective ACO algorithm for elephant flows. The candidate paths for elephant flows are determined by path delay, path risk, and path inverse bottleneck remaining bandwidth. The nondominated paths are then stored in the archive and then used for multipath traffic allocation. Our proposed algorithm is compared to classical Dijkstra, multi-objective Dijkstra, TSACO, and the Motlagh algorithms.

The rest of the paper is outlined as follows. Section 2 presents the problem statement and mathematical model. Section 3 overviews the baseline routing algorithms. Section 4 overviews the proposed TFPACO algorithm. Section 5 presents the simulation model and experimental results. Section 6 overviews the obtained results and concludes the paper.

2 Problem Statement and Mathematical Model

The considered SDN topology is represented as an undirected graph

$$G = (V, E), \tag{1}$$

where V is the set of nodes and E is the set of edges. Each edge $(i, j) \in E$ is associated with the following parameters:

$$d_{ij} > 0, \qquad b_{ij} > 0, \qquad p_{ij} \in (0, 1), \tag{2}$$

where d_{ij} is link delay, b_{ij} is link bandwidth, and p_{ij} is the packet loss probability of the link.

A traffic flow is represented as

$$q = (s, s', \lambda, \xi, c), \tag{3}$$

where $s \in V$ is the source node, $s' \in V$ is the destination node, $\lambda > 0$ is the bandwidth demand, ξ is the flow lifetime, and $c \in \{\text{mice}, \text{elephant}\}$ is the traffic class.

A route from s_q to s'_q is represented as a path

$$\mathcal{P}_q = (v_0, v_1, \ldots, v_\ell), \tag{4}$$

where $v_0 = s_q$, $v_\ell = s'_q$, and $(v_r, v_{r+1}) \in E$ for all $r = 0, \ldots, \ell - 1$. In the optimization model, the same route may be represented as binary edge variables

$$x_{ij,q} = \begin{cases} 1, & \text{if edge } (i, j) \text{ belongs to } \mathcal{P}_q, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

For elephant flows, multipath routing is allowed. In that case, the controller selects a set of routes

$$\{\mathcal{P}_q^{(1)}, \mathcal{P}_q^{(2)}, \ldots, \mathcal{P}_q^{(m_q)}\} \tag{6}$$

together with traffic shares

$$\boldsymbol{y}_q = (y_q^{(1)}, y_q^{(2)}, \ldots, y_q^{(m_q)}), \tag{7}$$

such that

$$y_q^{(r)} \geq 0, \qquad \sum_{r=1}^{m_q} y_q^{(r)} = 1. \tag{8}$$

Hence, the amount of traffic sent over route $\mathcal{P}_q^{(r)}$ is equal to $y_q^{(r)} \lambda_q$.

Let $\mathcal{F}(t)$ be the set of active flows at time t, and let

$$b_{ij}^{\text{used}}(t) = \sum_{q \in \mathcal{F}(t)} \sum_{r=1}^{m_q} \lambda_q y_q^{(r)} x_{ij,q}^{(r)}, \tag{9}$$

where $m_q = 1$ and $y_q^{(r)} = 1$, if $c_q = \text{mice}$, be the used bandwidth of edge (i, j) at time t. Then the remaining bandwidth is

$$b_{ij}^{\text{rem}}(t) = b_{ij} - b_{ij}^{\text{used}}(t). \tag{10}$$

For a feasible route, the remaining bandwidth must be nonnegative on all its edges, and for all active flows the bandwidth constraints must satisfy

$$b_{ij}^{\text{used}} \leq b_{ij}, \qquad \forall (i,j) \in E. \tag{11}$$

The route delay is assumed additive [5]:

$$f_1(\mathcal{P}_q) = D(\mathcal{P}_q) = \sum_{(i,j)\in\mathcal{P}_q} d_{ij}. \tag{12}$$

This is the standard path delay model used in shortest-path and QoS routing.

The route risk is also assumed to be additive:

$$f_2(\mathcal{P}_q) = R(\mathcal{P}_q) = \sum_{(i,j)\in\mathcal{P}_q} -\ln(1-p_{ij}), \tag{13}$$

which is consistent with the path risk formulation used in [7]. This representation is convenient because the packet delivery probability of the path is given as

$$\text{PDR}(\mathcal{P}_q) = \prod_{(i,j)\in\mathcal{P}_q} (1-p_{ij}) = \exp(-R(\mathcal{P}_q)). \tag{14}$$

For elephant flows, an additional QoS criteria is the bottleneck remaining bandwidth [5],

$$B(\mathcal{P}_q) = \min_{(i,j)\in\mathcal{P}_q} b_{ij}^{\text{rem}}, \tag{15}$$

which characterizes the narrowest edge on the route. Since all optimization criteria are considered in minimization form, the bandwidth-related objective is defined as

$$f_3(\mathcal{P}_q) = \frac{1}{B(\mathcal{P}_q)}. \tag{16}$$

Thus, routes with larger bottleneck remaining bandwidth are preferred.

For a single route, the objective vector is defined as

$$F(\mathcal{P}_q) = \left(f_1(\mathcal{P}_q),\, f_2(\mathcal{P}_q),\, f_3(\mathcal{P}_q)\right). \tag{17}$$

The routing problem is therefore formulated as a multi-objective optimization problem

$$\min_{\mathcal{P}_q\in S_q} \left\{F(\mathcal{P}_q)\right\}, \tag{18}$$

where S_q is the set of feasible routes for flow q.

The aim is to find feasible routes, or sets of routes, that provide a good compromise between delay, reliability, and remaining bandwidth.

A route $\mathcal{P}_q^*$ is Pareto-optimal if there exists no other feasible route $\mathcal{P}_q$ such that

$$f_i(\mathcal{P}_q) \leq f_i(\mathcal{P}_q^*), \qquad \forall i, \tag{19}$$

and for at least one index j

$$f_j(\mathcal{P}_q) < f_j(\mathcal{P}_q^*). \tag{20}$$

In other words, a Pareto-optimal route cannot be improved in one objective without worsening at least one other objective.

In the proposed algorithm, mice and elephant flows are treated differently. Mice flows are routed over a single path due to their small size and high delay sensitivity. Elephant flows, in contrast, may be distributed over multiple routes. For the selected elephant route with bottleneck remaining bandwidth values $B_1, \ldots, B_{m_q}$, the traffic shares are assigned proportionally [5]:

$$y_q^{(r)} = \frac{B_r}{\sum_{j=1}^{m_q} B_j}, \qquad \sum_{r=1}^{m_q} y_q^{(r)} = 1. \tag{21}$$

This allocation rule makes the routing decision bandwidth-aware and allows the centralized controller to distribute demand of elephant flow across multiple feasible paths.

Therefore, the problem considered in this work is to construct feasible routes in SDN under multiple QoS criteria and choose the best of them.

3 Overview of Routing Methods

This section briefly overviews the routing algorithms used as baselines in the comparative study. The considered algorithms differ in the way they handle QoS criteria, mice and elephant flows, and route construction under load.

3.1 Classical Dijkstra Algorithm

The classical Dijkstra algorithm is the standard shortest-path algorithm for graphs with nonnegative edge weights. In the SDN the edge weight is chosen as the link delay:

$$w_{ij} = d_{ij}. \tag{22}$$

Then the algorithm returns a minimum-delay route between the source and the destination nodes.

The main advantages of this approach are low computational complexity and simple implementation. However, the algorithm is single-objective: it ignores remaining bandwidth, packet loss, and route diversity. As a result, the delay-based shortest path may become overloaded under high traffic demand, especially in the presence of elephant flows.

3.2 Multi-objective Dijkstra Algorithm

The multi-objective Dijkstra algorithm considered in this work follows the weighted-sum idea used in [6]. In that approach, several edge parameters are combined into a single edge cost:

$$w_{ij} = \omega_1 \widetilde{d}_{ij} + \omega_2 \widetilde{b}_{ij}^{\mathrm{rem}} + \omega_3, \tag{23}$$

where $\widetilde{d}_{ij}$ is the normalized delay, $\widetilde{b}_{ij}^{\text{rem}}$ is the normalized remaining bandwidth, and the constant term is the hop-count contribution.

Hop count is a characteristic of a path defined as the total number of traversed edges. In the algorithm each edge contributes one unit to the total hop count. As a result, this unit contribution along the path represents the total number of hops. Therefore, in the edge-weight representation, the hop-count term is written as a constant value for each edge.

In the paper [6], three weight configurations are considered:

$$\{1, -0.1, 0.2\}, \qquad \{1, -0.5, 0.2\}, \qquad \{0.5, -1, 0.2\}.$$

Compared with the classical Dijkstra algorithm, the multi-objective one is more flexible and better takes into account QoS criteria. However, all considered criteria are still reduced to a single scalar weight, and the result remains sensitive to the selected coefficients.

3.3 TSACO Algorithm

The TSACO method proposed in [5] is specifically designed for SDN traffic scheduling under mice and elephant flow coexistence. Its main idea is to handle these traffic classes separately. Mice flows are delay-sensitive and are routed using a delay-oriented ACO algorithm, while elephant flows are bandwidth-sensitive and are routed by a bandwidth-oriented ACO algorithm.

The probability of choosing the next edge is defined by the standard ACO transition rule:

$$P_{ij} = \begin{cases} \dfrac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{k\in\text{neighbours}_i}\tau_{ik}^{\alpha}\eta_{ik}^{\beta}}, & \text{if } j \in \text{neighbours}_i, \\ 0, & \text{otherwise}, \end{cases} \tag{24}$$

where τ_{ij} is the pheromone level and η_{ij} is the heuristic desirability of edge (i, j). In TSACO, the heuristic differs for the two classes of flows. For mice flows, the heuristic is delay-oriented:

$$\eta_{ij} = \frac{1}{d_{ij}}, \tag{25}$$

whereas for elephant flows the heuristic is based on remaining bandwidth:

$$\eta_{ij} = b_{ij}^{\text{rem}}. \tag{26}$$

For elephant traffic, TSACO applies a k-path ACO algorithm. Several edge-disjoint routes are constructed iteratively until the total available bandwidth becomes sufficient. The flow demand is then distributed proportionally to route bandwidth. This makes TSACO an important reference for SDN traffic scheduling, which takes into account different flow classes. At the same time, the method is focused primarily on bandwidth efficiency and does not explicitly model route risk as an optimization objective.

3.4 Motlagh Multi-objective ACO Algorithm

The routing algorithm proposed by Motlagh et al. [7] formulates SDN route restoration as a multi-objective optimization problem under two criteria: path risk and operational cost. In that method, the path risk is defined as

$$\beta(\mathcal{P}_q) = \sum_{(i,j)\in\mathcal{P}_q} -\ln(1-p_{ij}), \tag{27}$$

where p_{ij} is the packet loss probability. The operational cost reflects the route reconfiguration overhead when replacing an already installed route by a new one.

The algorithm combines ACO route construction with mechanisms taken from multi-objective evolutionary optimization. At each iteration, candidate routes are generated by ants, evaluated with respect to the two objectives, and filtered by nondominated sorting. When the number of retained solutions exceeds the desired archive size, crowding distance is used to preserve diversity among the nondominated routes. The pheromone matrix is then updated using the quality of the obtained solutions.

An additional feature of the Motlagh algorithm is the penalization of the last edge of a failed route. This discourages repeated exploration of unproductive directions and improves the exploration ability of the ACO. After the final nondominated set is obtained, a compromise route is chosen by knee-point analysis.

The Motlagh algorithm is particularly relevant to the present work because it provides a clear example of multi-objective ACO routing in SDN. However, its original formulation focuses on route restoration and does not explicitly address class-aware traffic planning or bottleneck bandwidth-driven multipath allocation.

3.5 Discussion

The considered baseline methods illustrate different approaches to SDN routing. Classical Dijkstra is computationally efficient but only delay-oriented. Multi-objective Dijkstra improves metric awareness, but still relies on scalarization. TSACO introduces class-aware routing and multipath allocation for elephant flows, but it only takes into account one criteria for each class. The Motlagh algorithm is multi-objective and archive-based, but it does not address mice and elephant flows separately. These observations motivate the design of the proposed TFPACO algorithm, which combines mice and elephant flows with multi-objective ACO algorithm in the SDN.

4 Proposed TFPACO Algorithm

The proposed Traffic Flow Planning Ant Colony Optimization (TFPACO) algorithm combines Dijkstra algorithm for mice flows with multi-objective ACO algorithm for elephant flows. The main idea is to use a lightweight routing algorithm for short delay-sensitive flows while applying a more flexible multi-objective routing algorithm for large bandwidth-sensitive flows.

4.1 General Structure of the Method

For each incoming flow, the centralized controller first determines its traffic class. Mice flows are routed over a single path, whereas elephant flows may be distributed over several routes. This separation reflects the fact that the two traffic classes have different QoS requirements and different impact on network load.

4.2 Mice-Flow Routing

Mice flows are routed using the Dijkstra algorithm based on edge delay and risk. For edge $(i, j) \in E$, the routing weight is defined as

$$w_{ij} = d_{ij} + \phi r_{ij}, \tag{28}$$

where

$$r_{ij} = -\ln(1 - p_{ij}), \tag{29}$$

and ϕ is the coefficient controlling the influence of risk.

For a mice flow, the controller computes a single feasible shortest path with respect to this edge weight. This formulation favors routes with low delay and low risk, while preserving the computational efficiency of Dijkstra-based routing. Since mice flows have relatively small demand, a single-path decision is sufficient in most cases and helps to avoid unnecessary control overhead.

4.3 Elephant-Flow Routing

Elephant flows are routed using a multi-objective ACO algorithm. Unlike mice flows, elephant flows require path delay, path risk, and path remaining bandwidth. For this reason, the algorithm constructs a set of candidate routes and evaluates them by multiple criteria before selecting a multipath routing plan.

Pheromone Initialization. Before the ACO starts, a delay-based Dijkstra algorithm is computed between the source and the destination nodes of the elephant flow. This route is used to initialize the pheromone matrix. Hence, the search does not start from a completely uniform pheromone distribution, but receives a meaningful initial direction.

This initialization improves convergence of the ACO algorithm and reduces the probability of spending many iterations on bad routes. At the same time, it does not restrict the ants to only the seeded route, since the ants can still discover another and better alternatives.

Path Construction by Ants. After initialization, ants construct candidate routes between the source and destination nodes. Let τ_{ij} be the pheromone level

of edge (i, j) and let η_{ij} be the heuristic desirability of that edge. Then the probability of moving from node i to node j is defined as

$$P_{ij} = \begin{cases} \dfrac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{k\in\mathcal{N}(i)}\tau_{ik}^{\alpha}\eta_{ik}^{\beta}}, & \text{if } j \in \mathcal{N}(i), \\ 0, & \text{otherwise}, \end{cases} \tag{30}$$

where $\mathcal{N}(i)$ is the set of feasible neighbors of node i, and α and β determine the influence of pheromone and heuristic information, respectively.

In the proposed method, the heuristic desirability of an edge is defined as

$$\eta_{ij} = \frac{1}{d_{ij}} \cdot \frac{1}{r_{ij}+\varepsilon} \cdot b_{ij}^{\mathrm{rem}} \cdot \frac{1}{(1+\mathrm{dist}(j, s'))^{\psi}}. \tag{31}$$

where b_{ij}^{rem} is the remaining bandwidth of edge (i, j), d_{ij} is the delay, r_{ij} is the edge risk, $\varepsilon > 0$ is a small positive constant preventing division by zero, and $\mathrm{dist}(j, s')$ is the hop distance from node j to the destination node. The last factor reduces the probability of long detours during path construction. Therefore, the ACO prefers edges that are fast, reliable, and still have enough available capacity.

Multi-objective Path Evaluation. For every candidate route $\mathcal{P}_q$, the following objective vector is evaluated:

$$F(\mathcal{P}_q) = (f_1(\mathcal{P}_q), f_2(\mathcal{P}_q), f_3(\mathcal{P}_q)), \tag{32}$$

where

$$f_1(\mathcal{P}_q) = D(\mathcal{P}_q), \qquad f_2(\mathcal{P}_q) = R(\mathcal{P}_q), \qquad f_3(\mathcal{P}_q) = \frac{1}{B(\mathcal{P}_q)}. \tag{33}$$

Thus, the algorithm simultaneously minimizes path delay, path risk, and inverse path bottleneck remaining bandwidth.

Archive Update and Diversity Preservation. After candidate routes are generated, they are stored in the archive. Then all routes from archive are filtered by nondominated sorting. Only Pareto-optimal routes are stored in the archive. If the number of stored solutions is larger than the archive size, crowding distance is used in order to preserve diversity among the stored routes.

Therefore, the archive contains not only high-quality routes, but also routes representing different trade-offs between delay, risk, and inverse bottleneck remaining bandwidth.

Pheromone Update. Pheromone values are updated globally according to the quality of the archived routes. The update rule is

$$\tau_{ij} = \begin{cases} (1-\rho)\tau_{ij} + \dfrac{\Delta\tau_{ij}}{|\mathcal{P}_q|}, & \text{if } (i, j) \in \mathcal{P}_q, \\ (1-\rho)\tau_{ij}, & \text{otherwise}, \end{cases} \tag{34}$$

where ρ is the pheromone evaporation parameter,

$$\Delta\tau_{ij} = \frac{1}{score}, \qquad score = \frac{\widetilde{d}_{ij} + \widetilde{r}_{ij} + \widetilde{b}_{ij}^{-1}}{3} \tag{35}$$

where $\widetilde{d}_{ij}$ is the normalized delay, $\widetilde{r}_{ij}$ is the normalized risk, $\widetilde{b}_{ij}^{-1}$ is the normalized inverse remaining bandwidth.

If the current archive contains too few distinct routes, the algorithm interprets this situation as a loss of exploration. In this case, the pheromone evaporation rate is increased. A larger value of ρ weakens the influence of over-dominant routes and encourages the ants to explore alternative directions of the search space.

As well, to prevent early convergence, pheromone values are bounded:

$$\tau_{\min} \leq \tau_{ij} \leq \tau_{\max}. \tag{36}$$

The upper bound prevents a single route from dominating the entire search, whereas the lower bound prevents potentially useful edges from being completely ignored.

Final Path Subset and Proportional Allocation. After the search phase, the algorithm does not use all archived routes. Instead, it selects a subset of routes that are both high-quality and enough different from each other. For this purpose, a simple diversity filter is applied in the final selection stage in order to avoid several nearly identical routes.

Suppose that the selected routes have bottleneck remaining bandwidth values $B_1, \ldots, B_{m_q}$. Then the traffic share assigned to route r is defined as

$$y_q^{(r)} = \frac{B_r}{\sum_{j=1}^{m_q} B_j}, \qquad \sum_{r=1}^{m_q} y_q^{(r)} = 1. \tag{37}$$

Thus, the demand of elephant flow is distributed proportionally to available path capacity. This makes the final allocation simple and feasible.

4.4 Pseudocode of TFPACO

The overall logic of the proposed method for mice is summarized in Algorithm 1, and for elephant in Algorithm 2.

4.5 Main Features of the Proposed Method

The main difference between TFPACO and the baseline methods is that it combines division into classes of mice and elephants, multi-objective ACO, and multipath allocation in the SDN. In comparison with Dijkstra and multi-objective Dijkstra algorithms, the proposed method does not reduce the routing problem to a single scalar path cost. In comparison with TSACO, it uses multi-objective optimization and takes into account three criteria instead of just one. In comparison with the Motlagh algorithm, it divides traffic into classes and use more criteria in multi-objective ACO.

Algorithm 1. TFPACO: mice

Require: flow = (src, dst, demand, lifetime, class=mice), network graph G
Ensure: route plan
  Compute shortest path P using Dijkstra with weight $w_{ij} = d_{ij} + \phi r_{ij}$
  if P is feasible **then**
    return single-path routing $\{(P, demand)\}$
  else
    return error
  end if

Algorithm 2. TFPACO: elephant

Require: flow = (src, dst, demand, lifetime, class=elephant), network graph G
Ensure: route plan
  Compute shortest path P using Dijkstra with weight $w_{ij} = d_{ij}$
  Initialize pheromone matrix τ
  Update pheromone on edges of path P
  Initialize archive $A = \emptyset$
  MOACO(flow, network graph G, pheromone matrix τ, archive A)
  Select diverse subset of routes from archive A
  Assign traffic shares y_r proportionally to bottleneck bandwidth
  return multipath routing plan $\{(P_1, y_1 \cdot demand),\ \ldots,\ (P_k, y_k \cdot demand)\}$

Algorithm 3. TFPACO: MOACO

Require: flow = (src, dst, demand, lifetime, class=elephant), network graph G, pheromone matrix τ, archive A
Ensure: archive A
  for iter = 1,. . . ,T **do**
    Generate candidate paths using ants
    for each candidate path P **do**
      Compute objectives: $f_1(P) = D(P),\ f_2(P) = R(P),\ f_3(P) = \frac{1}{B(P)}$
    end for
    Merge candidates with archive A
    Perform nondominated sorting
    Keep Pareto-optimal solutions
    if archive size is too large **then**
      Apply crowding-distance selection
    end if
    if route diversity in A is too low **then**
      Increase evaporation coefficient ρ
    end if
    Update pheromone values
    Enforce pheromone bounds: $\tau_{\min} \leq \tau_{ij} \leq \tau_{\max}$
  end for

5 Simulation Model and Experimental Setup

5.1 Experimental Setup

To evaluate the performance of the proposed TFPACO algorithm, we implemented a discrete-time simulation environment in Python [9]. The network is modeled as a graph $G = (V, E)$, consisting of $|V| = 100$ nodes and a randomly generated connected topology. Each link $(i, j) \in E$ is characterized by a set of attributes: base delay $d_{ij}^{(0)}$, bandwidth b_{ij}, and base packet loss probability $p_{ij}^{(0)}$.

To reflect real-world network dynamics and the impact of congestion on QoS, we utilize non-linear cost functions for delay and loss. As noted in recent literature [6,8], link performance degrades as utilization increases. We define the link delay $d_{ij}^{(t)}$ and loss probability $p_{ij}^{(t)}$ at step t as [10]:

$$d_{ij}^{(t)} = d_{ij}^{(0)} \left(1 + k \cdot u_{ij}(t)^z\right), \qquad p_{ij}^{(t)} = p_{ij}^{(0)} + h \cdot u_{ij}(t)^\gamma, \tag{38}$$

where

$$u_{ij}(t) = \frac{b_{ij}^{\text{used}}(t)}{b_{ij}(t)} \tag{39}$$

represents the current link utilization. We set $k = 4.0$ with $z = 2.0$ for the delay model, and $h = 0.15$ with $\gamma = 3.0$ for the loss model, ensuring a sharp QoS degradation beyond the 90% utilization threshold.

The simulation is a sequence of flows

$$q = (s, s', \lambda, \xi, c), \tag{40}$$

where s and s' are randomly sampled nodes, λ is the bandwidth demand in kbps, ξ is the flow duration in discrete steps, and c is the traffic class.

Flows are categorized into:

- **Mice Flows:** Short-lived (2–8 steps), low-bandwidth (100–800 kbps), representing 80% of the total flow count.
- **Elephant Flows:** Long-lived (20–60 steps), high-bandwidth (2000–8000 kbps), representing 20% of the count but contributing approximately 90% of the total volume.

The simulation runs for $T = 50$ steps, with 50 new flows generated at each step, creating a high-pressure scenario to test the scalability and efficiency of TFPACO algorithm.

The performance of routing algorithms is measured through a comprehensive set of QoS metrics and network indicators:

- **Blocked Flow/Demand Ratio:** The percentage of flows and total bandwidth demand of them that the controller couldn't= process due to resource constraints.
- **p95 Experienced Delay (d_{95}^{exp}):** The 95th percentile of the actual latency experienced by admitted flows, focusing on tail-latency.

- **p5 Packet Delivery Ratio (PDR_5^{exp}):** The lower 5th percentile of PDR, highlighting the reliability of the most vulnerable paths.
- **Jain's Fairness Index:** A measure of load distribution across the network, where 1.0 indicates perfectly balanced utilization.
- **Congestion:** The percentage of simulation time during which at least one edge is heavily overloaded (above 90% capacity).

5.2 Experimental Results

The comparison of the TFPACO algorithm with the Dijkstra, the Multi-Objective Dijkstra, TSACO, and the Motlagh algorithms is presented in Tables 1 and 2 and Figs. 1–5.

As illustrated in Table 1, the Dijkstra algorithm has the lowest latency for admitted flows, but at the cost of very high demand blocking rate (49.86%), since it does not take into account the available bandwidth. The MO-Dijkstra variants [6] reduce the demand blocking to some extent (approx. 30%), since they take into account the available bandwidth. However, they still fail to support high demand requirements.

In contrast, TSACO, Motlagh and TFPACO significantly reduce blocking. However, the computational cost of the TSACO and Motlagh MOACO is extremely high. The proposed TFPACO algorithm achieves a 100% admission rate (0.00% blocking), while maintaining a much smaller runtime (see Table 2) and average delay (see Fig. 1) than TSACO and Motlagh, and a much smaller mice average delay (see Fig. 2) than Motlagh, demonstrating the benefits of separating traffic classes and multi-objective ACO.

Table 1. Comparison of Routing Algorithms: QoS and Reliability Metrics

Algorithm	Flow Blk., %	Dem. Blk., %	d_{95}^{exp} (ms)	d_{95}^{mice} (ms)	PDR_5^{exp}, %
Dijkstra (Delay)	15.12	49.86	28.84	25.78	82.10
MO-Dijkstra (Cfg1)	7.88	33.30	40.25	33.36	89.23
MO-Dijkstra (Cfg2)	8.28	34.08	38.75	32.22	89.55
MO-Dijkstra (Cfg3)	6.72	29.10	46.69	37.90	91.38
TSACO	1.92	0.88	160.36	28.35	80.63
Motlagh MOACO	0.00	0.00	261.40	256.50	66.74
TFPACO	**0.00**	**0.00**	**71.49**	**27.58**	**86.28**

(See Figs. 3 and 4)

Table 2. Network State and Computational Overhead Metrics

Algorithm	Runtime (ms)	Max Util., %	Congestion, %	Jain Index
Dijkstra (Delay)	2,572.2	99.90	100.0	0.181
MO-Dijkstra (Cfg1)	4,947.0	99.88	100.0	0.211
MO-Dijkstra (Cfg2)	5,020.1	99.90	100.0	0.212
MO-Dijkstra (Cfg3)	5,086.7	99.85	100.0	0.213
TSACO	1,109,019.0	99.83	92.0	0.493
Motlagh MOACO	907,056.6	99.90	100.0	0.344
TFPACO	**103,602.4**	**99.90**	**70.0**	**0.438**

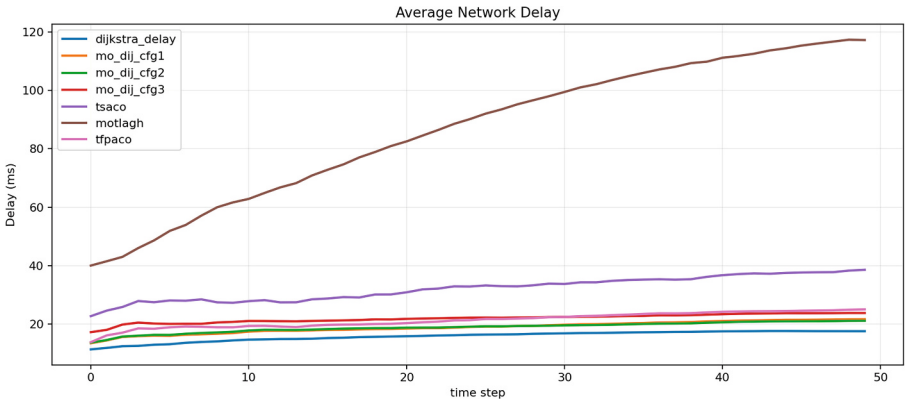

Fig. 1. Average delay over simulation time for the compared algorithms.

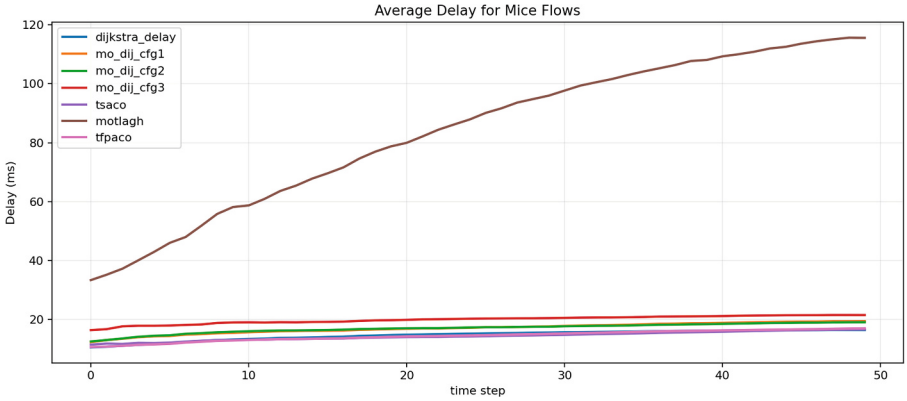

Fig. 2. Average mice delay over simulation time for the compared algorithms.



Fig. 3. Packet delivery ratio over simulation time for the compared algorithms.

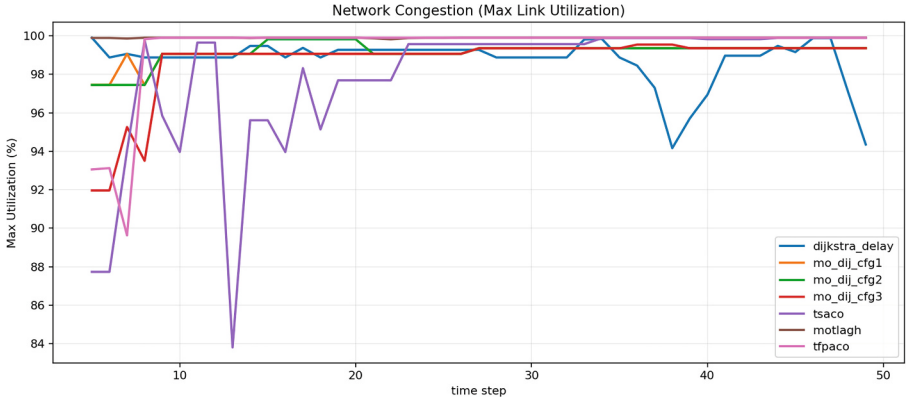

Fig. 4. Maximum link utilization over simulation time for the compared algorithms.

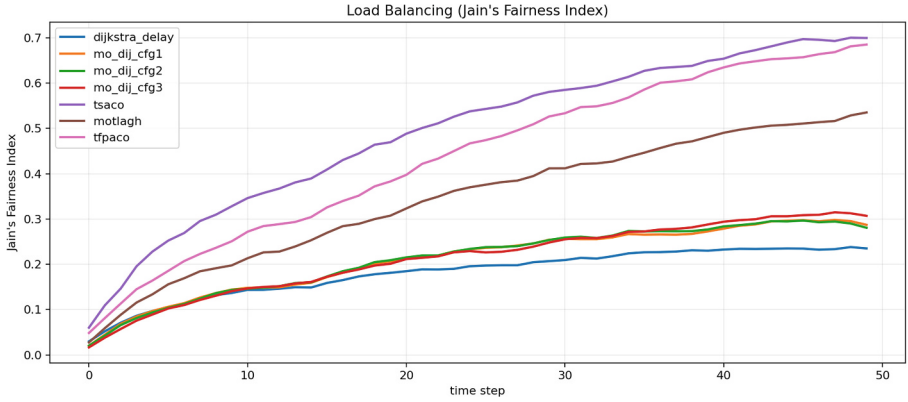

Fig. 5. Network load dynamics over simulation time for the compared algorithms.

6 Conclusion

The paper addressed the problem of traffic management in software-defined networks with QoS criteria and dividing traffic flows in two classes. Since mice flows and elephant flows have different requirements, it is not reasonable to use a common routing strategy. To deal with this problem, we proposed the TFPACO algorithm, which combines the Dijkstra algorithm for mice flows with a multi-objective ant colony optimization algorithm for elephant flows. For elephant flows, the proposed method takes into account path delay, path risk, and inverse bottleneck remaining bandwidth, where the archive of nondominated candidate paths is used for further multipath allocation of the traffic flows.

The proposed algorithm was evaluated in a discrete-time simulation environment with dynamic edge parameters and was compared with the Dijkstra algorithm, multi-objective Dijkstra algorithm, TSACO, and the Motlagh algorithm. The results show that TFPACO is effective in selecting routes for both traffic types and in improving overall routing performance. In particular, the algorithm provides lower latency and better traffic distribution, while avoiding flow and demand blocking under the considered simulation parameters. These results indicate that the proposed method is a promising solution for SDN traffic engineering in networks with different QoS requirements.

References

1. Xia, W., Wen, Y., Foh, C.H., Niyato, D., Xie, H.: A survey on software-defined networking. IEEE Commun. Surv. Tutorials **17**(1), 27–51 (2015)
2. Xu, C., Qin, D., Song, F.: A survey of SDN traffic management research. In: 11th International Conference on Communications, Circuits and Systems (ICCCAS), pp. 231–236 (2022)
3. Tache, M.D., Păscuțoiu, O., Borcoci, E.: Optimization algorithms in SDN: routing, load balancing, and delay optimization. Appl. Sci. **14**(14), 5967 (2024)
4. Wang, C., Zhang, G., Xu, H., Chen, H.: An ACO-based link load-balancing algorithm in SDN. In: 7th International Conference on Cloud Computing and Big Data (CCBD), pp. 214–218 (2016)
5. Wang, C., Zhang, G., Chen, H., Xu, H.: An ACO-based elephant and mice flow scheduling system in SDN. In: 2nd IEEE International Conference on Big Data Analysis (ICBDA), pp. 859–863 (2017)
6. Choudhary, A., Kang, S.S., Singla, S.: Multi-objective Dijkstra algorithm for enhancing QoS in SDN through balanced routing. In: 7th International Conference on Circuit Power and Computing Technologies (ICCPCT), pp. 1169–1175 (2024)
7. Taghavi Motlagh, S., Ibrahim, A., Shah Heydari, S., El-Khatib, K.: Multi-objective optimization for managing disruption risk in SDN. In: 20th International Conference on the Design of Reliable Communication Networks (DRCN), pp. 1–8 (2024)
8. Shreyas, J., Jumnal, A., Udayaprasad, P.K., Rekha, C., Askar, S.S., Abouhawwash, M.: Bio-inspired ACO-based traffic aware QoS routing in software defined Internet of Things. Appl. Artif. Intell. **38**(1), 2371739, 1–26 (2024)

9. Safonenko, A.: TFPACO algorithm LNCS. https://github.com/NillFin/TFPACO-algorithm-LNCS-26. Accessed 30 Mar 2026
10. Qiu L., Yang Y.R., Zhang Y., Shen S.: On selfish routing in internet-like environments. In: Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (ACM SIGCOMM), pp. 151–162 (2003)