MONR: Multi-Objective Optimizing Network Reconfiguration using Deep Reinforcement Learning

Yuqi Dai[†*] Hua Zhang[†*], Jingyu Wang[‡], and Jianxin Liao[‡]
[†]National Mobile Communications Research Laboratory, Southeast University, Nanjing 211111, China
E-mails: 230228198@seu.edu.cn, huazhang@seu.edu.cn
[‡]State Key Laboratory of Networking and Switching Technology,
Beijing University of Posts and Telecommunications, Beijing, China
E-mails: wangjingyu@bupt.edu.cn, liaojx@bupt.edu.cn

2024 IEEE 32nd International Conference on Network Protocols (ICNP) | 979-8-3503-5171-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICNP61940.2024.10858522

***Abstract*—Modern networks require frequent configuration updates due to dynamic events, like network expansion and evolving traffic patterns. Existing network reconfiguration tools are effective in certain scenarios, but their practical deployment still has several limitations: (i) They are restricted to specific network topologies, protocols and specifications; (ii) They can cause transient violations; (iii) Their practical deployment is limited by huge computational overheads and the specialized hardware support. To address these limitations, this paper presents a Multi-Objective Optimizing Network Reconfiguration (MONR) framework, which comprises a translator and an optimizer, to automatically generate reconfiguration command sequences. The translator represents various input types into a unified graph format based on Datalog-like facts, which are regardless of the input format. Therefore, MONR supports diverse routing protocols and network specifications. The optimizer employs deep reinforcement learning techniques to simultaneously maximize the specification satisfaction of intermediate configurations and minimize both traffic shifts and the number of command updates to reduce computational overheads. It models the reconfiguration task as a multi-objective Markov Decision Process (MOMDP) and introduces a Dueling Prioritized Experience Replay Double Deep Q-Network (DPER-DDQN) algorithm to balance multiple objectives. We compare MONR with Snowcap, AED and ConfigReco. The evaluation demonstrates that MONR is 2x, 9x, and 56x faster than Snowcap, AED and ConfigReco. Furthermore, MONR maintains 100% specification consistency while reducing the traffic shifts (< 0.1) and the number of update commands (0.8 of Snowcap's).**

***Index Terms*—Network reconfiguration, network management, multi-objective optimization, deep reinforcement learning.**

I. Introduction

MODERN network necessitates frequent router configuration updates to adapt to dynamic events, such as network expansion and evolving traffic patterns [1]. These router configurations are written in low-level configuration languages, such as standard Cisco commands. These updates involve the addition of new BGP sessions, the migration of routing protocols (e.g., from Routing Information Protocol (RIP) [2] to Open Short Path First (OSPF) [3]), and the adoption of route reflectors [4]. Network reconfiguration tools implement these updates through modifying existing configuration parameters. Existing tools are primarily categorized into in-place and parallel methods. In-place tools apply reconfiguration commands sequentially (one command at a time). In contrast, parallel tools, such as those based on Ships-in-the-Night (SITN) techniques [5]–[8], implement multiple reconfiguration commands simultaneously. Although these tools are effective in specific scenarios, but there are certain challenges in actual deployment, particularly in complex networks.

Limited Scalability: The scalability of current reconfiguration tools is limited by specific network scenarios. The reconfiguration times of these tools are prolonged as the network scale increases. Furthermore, these tools are tailored to specific routing protocols (e.g. Border Gateway Protocol (BGP) [9]) and network specifications (e.g., reachability).

Unsafety: Tools like Snowcap [4] are hard to maintain safety. In this context, safety implies the predefined network specifications must be continuously satisfied throughout any transient configuration state explored during the reconfiguration process [10]. It is important to note that "safety" here does not refer to temporal aspects such as timing or synchronization of updates. The level of this satisfaction is quantified as the specification consistency. Such tools reconfigure networks by transitioning through a series of intermediate configurations with distinct routing and forwarding states. This process can lead to unsafety because intermediate configurations may temporarily violate network specifications even if the original and target configurations are safe. These transient violations can result in downtimes and service interruptions [11].

Limited Practicality: Reconfiguration tools, particularly those based on SITN techniques, often face limitations in practical deployment due to huge computational overheads and the need for specialized hardware. These tools incur

* Corresponding authors.

This work was supported by the National Key Research and Development Program of China under Grant (2020YFB1807803), Key Research & Development Plan of Jiangsu Province (Grant No. BE2020084-2) and Nanjing 5G Network Optimization Innovation Alliance.

979-8-3503-5171-2/24/$31.00 ©2024 IEEE

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on October 30,2025 at 11:22:59 UTC from IEEE Xplore. Restrictions apply.

computational overheads by replicating routing and forwarding states across all network routers. Additionally, there may exist multiple reconfiguration command sequences satisfying the given specifications. These sequences can lead to unnecessary computational overheads [4], such as traffic shifts, which result in prolonged reconfiguration times, so that the practicality is diminished. Furthermore, not all routers support state duplication also limits the practicality of these tools [12].

In this paper, we propose a Multi-Objective Optimizing Network Reconfiguration (MONR) framework to automatically generate network reconfiguration command sequences. MONR integrates a translator to enhance its scalability and an optimizer to ensure its safety and practicality. The translator utilizes a graph embedding scheme to represent the original and target configurations, along with the given specifications, into a unified graph format. This scheme is generic with respect to the input format by using an intermediate representation based on Datalog-like facts [13]. As a result, MONR can be scaled to support diverse routing protocols and network specifications. The optimizer utilizes deep reinforcement learning (DRL) techniques to optimize the reconfiguration process to meet three optimization objectives, including maximizing the specification consistency, and minimizing the traffic shifts and the number of update commands. The reconfiguration task is modeled as a multi-objective Markov Decision Process (MOMDP), and a specifically designed Dueling Priotitized Experience Replay Double Deep Q-Network (DPER-DDQN) algorithm is proposed to balance these objectives. DRL techniques repeatedly explore different reconfiguration strategies and receive feedback on whether these strategies violate specifications to ensure safety. Their applied offline training strategy ensures practicality by reducing actual implement time without retraining. In summary, our contributions are as follows:

- We propose MONR, a scalable reconfiguration tool that automatically generate update command sequences to ensure safe and practical network reconfiguration.
- We utilizes a graph embedding scheme to represent various inputs into a unified Datalog-like facts-based graph to support any routing protocols and specifications.
- We treat the safe and practical network reconfiguration task as a multi-objective optimization problem. Furthermore, we introduce a DPER-DDQN algorithm based on DRL techniques to guide the sequence generation by balancing multiple optimization objectives.
- Our evaluation demonstrates that MONR outperforms Snowcap, AED and ConfigReco in terms of runtime, specification consistency, traffic shifts, and the number of update commands.

II. Related Work

In this section, we first review the existing literature of network reconfiguration. Then, we explore configuration tuning and optimization methods and discuss the applications of DRL techniques in network management, both of which are motivations for us to employ DRL techniques in network reconfiguration tasks.

Network Reconfiguration: Current reconfiguration tools can be divided into in-place and parallel tools. In-place tools implement configuration updates in sequence. Prior reconfiguration tools [14], [15] avoid forwarding loops during Interior Gateway Protocol (IGP) reconfiguration but are not applicable to other protocols like BGP. Snowcap [4] is the first to support reconfiguration for arbitrary network protocols and specifications by utilizing linear temporal logic (LTL). It also adopts an optimization objective to prevent unnecessary side-effects, like traffic shifts. However, it is limited by its unsafety due to transient specification violations during reconfiguration. Chameleon [10] is proposed to solve this issue. It introduces a concurrency model based on happens-before relations to model the propagation of BGP routes and leverages synchronization techniques to ensure the specification consistency during reconfiguration. However, the concurrency model is only designed for BGP routes, so Chameleon fails to reconfigure configurations for more protocols or specifications.

Parallel reconfiguration tools execute multiple configurations updates simultaneously. Typical tools are based on SITN-based techniques [5]–[8], [16], where each router runs two separate configurations in parallel. One of these two configurations is generated by duplicating routing and forwarding states of another configuration on all network devices. This duplication causes substantial computational overheads. Furthermore, not all routers support the state duplication, which restricts these tools' practical deployment.

Configuration Tuning and Optimization: The process of configuration tuning and optimization involves modifying network configurations to meet specific optimization objectives such as enhancing service performance. Traditional methods like AED [17] automate this process through a system of Satisfiability Modulo Theory (SMT) constraints, which represents the structure of configurations and their interaction with routing algorithms. However, the huge computational demands of SMT makes these methods are limited in runtime. Recent researches have increasingly focused on leveraging data-driven techniques for automating configuration tuning and optimization. For example, Configanator [18] employs a contextual multi-armed bandit method, which utilizes a historical archive of network characteristics to predict the optimal configurations for Content Distribution Networks (CDNs). Moreover, Chroma [20] utilizes the learned network contexts to recommend performance-enhancing configurations, and ConfigReco [21] utilizes Graph Neural Networks (GNNs) to learn the constructed configuration-related knowledge graph for configuration recommendation. These studies collectively provide a insight for utilizing data-driven methods to realize multi-objective optimization in network reconfiguration.

DRL in Network Management: DRL has emerged as a technique to address network optimization problems (e.g., routing and topology optimization across various large networks). For example, an application in Software-Defined Networking (SDN) integrates a GNN classifier within an Advantage Actor Critic (A2C) framework for routing optimization in large networks [22]. Furthermore, DeepConf [23] employs

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on October 30,2025 at 11:22:59 UTC from IEEE Xplore. Restrictions apply.

DRL to optimize network topologies in Data Center Networks (DCNs) by refining data flow policies and resource allocation strategies. Motivated by DRL's integration of deep learning's ability to process high-dimensional data and reinforcement learning's ability to find optimal strategies, we consider to employ DRL for safe and practical network reconfiguration.

III. Problem Formulation

In this study, network reconfiguration involves transitioning from an original configuration C to a target configuration C'. This transition must satisfy a set of network specifications Φ, which are network properties, such as reachability, should be maintained during reconfiguration to ensure safety [24]. These specifications are expressed in high-level human language, but configurations are written in low-level configuration languages. To unify the representation of network configurations and specifications, we design a translator in Section IV-A that converts them into a unified format.

Once the configurations and specifications are standardized, the process of network reconfiguration is considered as generating a sequence of update commands $U = [u_1, u_2, ..., u_n]$ to transition from C to C', while ensuring the compliance with Φ. However, there may exist multiple sequences could satisfy Φ. It is crucial to determine optimization objectives that guides the selection of the most suitable sequences. Each objective is associated with a function $O : \Delta C_i \rightarrow \mathbb{R}$ that maps a sequence of intermediate network configurations ΔC_i to a numerical value $\mathbb{R}$. This value indicates how well the sequence meets the objective and is monotonically increasing ($O([\Delta C_0, \ldots, \Delta C_{n-1}]) \leq O([\Delta C_0, \ldots, \Delta C_n])$). To deal with the limitation of existing reconfiguration tools, we focus on three common optimization objectives [4]: minimizing the total traffic shifts O_{traffic}, reducing the total number of reconfiguration commands O_{changes}, and maximizing the compliance with the network specifications O_{specs}. Network operators can extend the optimization objectives O according to their needs.

Therefore, the reconfiguration task can be formulated as a multi-objective optimization problem. Given an C, C', Φ and O, the optimal reconfiguration command sequence U^* should be generated. We assume both C and C' are correct and in compliance with Φ. Otherwise, it is meaningless to maintain safe intermediate configuration during the reconfiguration [10]. Formally, the problem is defined as follows:

$$U^* = \arg\min_U \lambda_1 \cdot O_{\text{traffic}} + \lambda_2 \cdot O_{\text{changes}} - \lambda_3 \cdot O_{\text{specs}}$$

$$\text{s.t.} \quad \begin{cases} O_{\text{specs}}(\Delta C_i, \Phi) := \frac{\sum_{i=0}^{n} \sum_{\phi \in \Phi} \mathcal{I}(\Delta C_i, \phi)}{n \cdot |\Phi|} \\ O_{\text{traffic}}([f_1, f_2, ..., f_n]) := \sum_{i=1}^{n} f_i \\ O_{\text{changes}} := |U|, \\ U = [u_1, u_2, ..., u_n] \in \{u_{\text{add}}, u_{\text{delete}}, u_{\text{modify}}\}, \\ f_i = \frac{1}{|R| \cdot |P|} \sum_{r \in R} \sum_{p \in P} \mathcal{H}(\mathcal{F}(\Delta C_{i-1}), \mathcal{F}(\Delta C_i)) \\ (R, P) = \mathcal{F}(\Delta C_i) \\ \mathcal{H}(\mathcal{F}(\Delta C_{i-1}), \mathcal{F}(\Delta C_i)) \in \{0, 1\} \\ \mathcal{I}(\Delta C_i, \phi) \in \{0, 1\}, \quad \forall \phi \in \Phi \end{cases} \tag{1}$$

where:

- $\lambda_1, \lambda_2, \lambda_3$ are weights that balance the importance of each optimization objective. The values depend on different network scenarios. For example, in data center networks, where it is more important to minimize traffic shifts [25], we should assign a larger value to λ_1 than to λ_2 and λ_3.
- ΔC_i ($1 \leq i \leq n$) represents the intermediate configuration after applying the i-th update command in the sequence U, where n is the length of entire sequence.
- O_{traffic} is the objective function of total traffic shifts $[f_1, ..., f_n]$, where f_i is the traffic shift for i-th command.
- O_{changes} is the objective function for the number of configuration changes, which is equal to the length of the reconfiguration command sequence U.
- O_{specs} ($0 \leq O_{\text{specs}} \leq 1$) is an objective function that quantifies the specification consistency of the configuration ΔC_i to the specifications Φ. $|\Phi|$ represents the number of all specifications. $O_{\text{specs}} = 0$ means no intermediate configurations can satisfy arbitrary specifications, while $O_{\text{specs}} = 1$ means all configurations during reconfiguration can satisfy any specifications.
- U is the sequence of update commands. It comprises valid operations for network configurations: additions u_{add}, deletions u_{delete}, and modifications u_{modify}.
- $\mathcal{I}(c, \phi)$ is an indicator function that returns 1 if the configuration c satisfies the specification ϕ, and 0 otherwise.
- $\mathcal{F}(\Delta C_i)$ is a forwarding state generation function to represent the next-hop for prefix $p \in P$ chosen by router $r \in R$. Here, R and P are the set of all routers and all prefixes in the configuration ΔC_i, respectively.
- $\mathcal{H}$ is a forwarding transition change function that compares the forwarding state generation function for each prefix between consecutive configurations ΔC_{i-1} and ΔC_i. It is a binary function, which outputs 1 if $\mathcal{F}(\Delta C_{i-1}) \neq \mathcal{F}(\Delta C_i)$, and 0 for other cases.

IV. Reconfiguration with MONR

In this section, we provide an overview of MONR for automated network reconfiguration, which is illustrated in Fig. 1. It takes four inputs: the original and target configurations, along with the network specifications and a set of optimization objectives. MONR is composed of a translator for unifying the languages of configurations and specifications, and an optimizer for reconfiguration optimization. The translator utilizes a graph embedding scheme to convert the input configurations and specifications into unified network state graphs based on Datalog-like facts. The optimizer treats the reconfiguration task as an optimization problem and we employ a DRL-based algorithm to solve it. Through the collaboration of these two components, MONR automatically generates a sequence of update commands for configuration transition.

Consider the network configuration illustrated in (a) of Fig. 1. This network comprises seven routers, with R serves as the root route reflector. Three routers (B1, B2 and R) receive an identical external route for a prefix p. The target configuration (b) requires to remove three iBGP sessions

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on October 30,2025 at 11:22:59 UTC from IEEE Xplore. Restrictions apply.

which are denoted as red dotted lines and add an iBGP session which is denoted as a green dotted line. For simplicity, we focus on the reachability specifications (c) during the reconfiguration, and three optimization objectives (d) defined in Eq. (1). For this reconfiguration task, MONR automatically outputs an update command sequence (e), where each command correlates with a specific reconfiguration operation. For example, $u_{\text{add}}(\texttt{ibgp}(B1, R1))$ instructs the addition of an iBGP session from B1 to R1.

Next, we introduce each component within MONR.

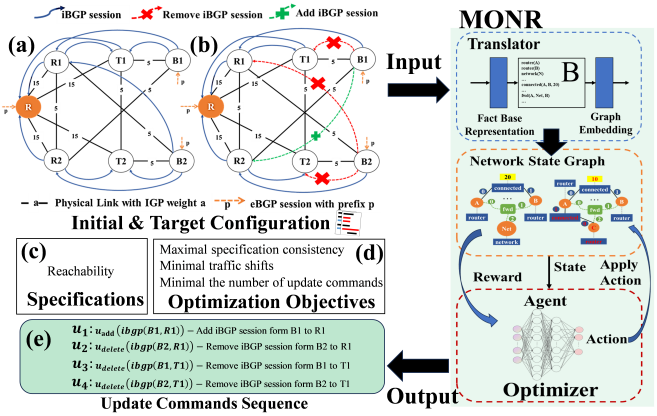

Fig. 1: The overview of MONR.

A. Translator

Routing protocols are crucial as they determine the network's forwarding state [13]. Studies on routing algebras suggest that these protocols can be modeled as message-passing graph algorithms, typically represented in forms of Bellman-Ford propagation processes [26]. To ensure our reconfiguration tool remains agnostic with routing protocols, we design a translator to standardize the input format of network configurations and specifications into a graph format. This design is inspired by the fact base graph used in a NAR-based synthesizer [13]. This protocol-agnostic translator first represents network configurations and specifications as a set of Datalog-like facts, which is similar to knowledge graphs, and then employs a structurally-defined graph embedding scheme to embed these facts into a fact base graph. The generated graph is defined as the network state graph $\mathcal{G}$. The subsequent sections provide an in-depth illustration of our translator.

1) Fact Base Representation: The translator first translates network configurations and specifications as a fact base $\mathcal{B}$. The fact base $\mathcal{B}$ is the collection of Datalog-like facts $b(a_0, \ldots, a_n)$, where parameters may be constants (e.g., A or B, which is the identification of a router) or integers (e.g., OSPF link weights or BGP attributes). Each fact comprises:

- *Identifiers*: Identifiers are utilized to distinguish routers (`router`), external BGP peers (`external`), and network prefixes (`network`) in the networks.
- *Link attributes*: Link attributes are denoted in the form of $\texttt{connected}(u, v, \omega)$, where u and v are source and destination of a link, and ω is the OSPF link weight.
- *BGP Sessions*: $\texttt{ibgp}(u, v)$ denotes an internal BGP (iBGP) session between routers u and v within the same autonomous system (AS). $\texttt{ebgp}(u, v)$ denotes an external BGP (eBGP) session, where routers u and v belong to different ASes. $\texttt{rr}(u, v)$ signifies that u is a route reflector for router v within an AS. $\texttt{br}(E, net, C)$ is used to denote various BGP route attributes, where E is the source entity (router/AS), and net is the network prefix. A set of integers C represents various attributes of the route, such as local preference lp, AS path length len, and multi-exit discriminator (MED) values med.

Based on these main components of a Datalog-like fact, arbitrary network specification can be encoded as the predicates. Common predicates are summarized as follows:

- $\texttt{fwd}(u, net, v)$ specifies the reachability that router u forwards traffic to network net via its neighbor v.
- $\texttt{reach}(u, net, v)$ denotes the waypoint that the traffic must traverse from router u to router v for reaching the destined network net.
- $\texttt{isolate}(u, v, n_1, n_2)$ dictates only one of the paths $\texttt{reach}(u, n_1, v)$ and $\texttt{reach}(u, n_2, v)$ is active at a time, where n_1 and n_2 are different destined networks.
- $\texttt{ecmp}(E_1, E_2, [p_1, p_2, ...])$ signifies the load balancing across multiple paths from E_1 to E_2, where E_1 and E_2 are the source and destination entity (router/AS), and $[\cdot]$ is a list of paths between E_1 and E_2.
- $\texttt{order}(p, v_{\text{pref}})$ denotes the path preference of path p, and v_{pref} is the value of preference. The path p can be denoted as $\texttt{fwd}(\cdot)$ or $\texttt{reach}(\cdot)$.

2) Graph Embedding: To embed the fact base into a graph with node features, the translator utilizes a structurally-defined graph embedding scheme. Given a fact base $\mathcal{B}$, the scheme embeds all configuration-related and specification-related facts into a single graph, which is called the state graph. In this graph, each fact b and constant c are represented as distinct nodes with the features h_b and h_c, respectively. The relationships between facts and constants are defined as the neighborhood functions N_i. We use multiple neighborhood functions to encode the constant position in a fact and the path orders are indicated as edge labels such as 0, 1, 2, etc. Take $\texttt{fwd}(A, Net, B)$ as an instance, we use 3 neighborhood functions to identify different positions of the source router A, the destination network Net, and the next-hop router B, respectively. The embedding rules for translating a fact base to node features relies on the learned parameters. These parameters are summarized in Table I, where v is a integer, o is a boolean value, i is the parameter index, and $\texttt{Bool}[b(a)]$ represents a truth value of a fact. Specifically, $T_b \in \mathbb{R}^D$ represents each fact type (e.g., `connected` and `ibgp`, which relies on protocols) embedding of the fact b, $V_{\text{bool}} \in \mathbb{R}^D$ denotes the embedding of boolean values, and $I_{b_i} \in \mathbb{R}^{D \times N}$ specifies the integer arguments (e.g., OSPF link weights) of the fact type b, where D and N are the number of facts and supported integer values, respectively. For example, if there exists a link between router A and B with a weight of 5, this embedded

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on October 30,2025 at 11:22:59 UTC from IEEE Xplore. Restrictions apply.

fact to the node feature is represented as: $h_{\texttt{connected}(A,B,5)} := \texttt{Gemb}(\texttt{connected}) + \texttt{Gemb}(\texttt{connected}, 2, 5)$.

B. Optimizer

To ensure safe and practical reconfiguration, we consider three optimization objectives: maximal specification consistency, minimal traffic shifts and minimal number of reconfiguration commands. To realize these objectives simultaneously, we frame the reconfiguration task as a multi-objective optimization problem. Traditional mathematical methods are not well-suited for addressing these problems due to the difficulty in balancing conflicting objectives [27]. To overcome this, we design an optimizer by employing DRL techniques, which can handle complex decision-making scenarios with multiple conflicting objectives. The optimizer first models the optimization problem as MOMDP, and then a DPER-DDQN algorithm is proposed to solve it by evaluating trade-offs between multiple objectives dynamically. Therefore, MONR can automatically generate desirable reconfiguration command sequences which simultaneously meet three mentioned optimization objectives. In the following section, we first outline the theoretical foundation of the optimizer. Then, we describe the design of MOMDP for our defined reconfiguration optimization problem, and the proposed DPER-DDQN algorithm for decision-making.

1) Preliminaries: The theoretical foundation of the optimizer commences with a review of applying reinforcement learning (RL) techniques to solve the single-objective optimization problem, which is modeled as Markov Decision Processes (MDP). After that, we introduce the principles of optimizing multiple objectives, which are utilized in MONR.

Single-Objective MDP. In RL, the decision-maker is called an agent [28]. The RL problem involves an agent learning the optimal policies in the form of MDP [29]. An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, P, \mathcal{R})$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ denotes the set of actions, P represents the state transition probabilities, and $\mathcal{R}$ signifies the reward function [30]. A complete sequence from the initial to terminal state is called an episode, while an individual step in which the agent interacts with the environment, selects an action, and moves to a new state is called a timestep. In a single-objective MDP, an agent interacts with the environment at discrete intervals $t = 0, 1, 2, ..., T$. At each interval t, the agent observes a state $s_t \in \mathcal{S}$, and selects an action $a_t \in \mathcal{A}$. After the action selection, a reward $r_{t+1} \in \mathcal{R}$ is obtained, and the agent transits into the next state s_{t+1}. A policy $\pi(a_t|s_t)$ indicates that an action is selected according to the probability distribution for any given state. The objective of the agent is to learn the optimal policy π^* that maximizes the expected discounted return (cumulative reward) [31]:

$$G_t = \sum_{i=t}^{T} \gamma^{(i-t)} r_{i+1}, \tag{2}$$

where γ $(0 \leq \gamma \leq 1)$ is the discount factor.

The state-value function is the expected discount return starting from the state s under the policy π:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t|s_t = s]. \tag{3}$$

The action-value function (also called Q-function) estimates the expected future reward of action a in a given state s under the policy π:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[G_t|s_t = s, a_t = a]. \tag{4}$$

MOMDP. MOMDP models the situations where the agent aims to optimize across multiple objectives simultaneously [32]. The crucial difference between single-objective MDP and MOMDP is that MOMDP has the vector-valued reward function $\mathbf{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}^d$, which expresses the immediate reward for each of the considered $d \geq 2$ objectives [33]. As a result, the value of state s for any time interval t and a given policy π in MOMDP is defined as:

$$\mathbf{V}^{\pi}(s) = \mathbb{E}_{\pi}[\sum_{i=t}^{T} \gamma^{(i-t)} \mathbf{r}_{i+1}|s_t = s], \tag{5}$$

where $\mathbf{V}^{\pi}(s) \in \mathbb{R}^d$ is also a vector.

In contrast to single-objective MDP, the value functions in MOMDP cannot define the natural order between different policies without any additional information about how to prioritize them. A typical method is to use a utility function (also called a scalarization function) $u : \mathbb{R}^d \rightarrow \mathbb{R}$ that maps the multi-objective value $\mathbf{V}^{\pi}$ to a scalar value. For simplicity, our paper employs the linear utility function, which computes the inner product of a weight vector $\mathbf{w}$ and a value vector $\mathbf{V}^{\pi}$:

$$\mathbf{V}_u^{\pi} = \mathbf{w}^{\top} \mathbf{V}^{\pi}, \quad \text{s.t.} \quad \sum_{i=1}^{n} w_i = 1, \tag{6}$$

where w_i are the elements of $\mathbf{w}$.

Decision-Making Algorithm. An appropriate decision-making algorithm is crucial for the reconfiguration optimization problem because exploring all possible reconfiguration command sequences is costly. Furthermore, an effective reconfiguration requires to balance multiple optimization objectives without incurring over-estimation biases, which cause the decisions that do not fully satisfy optimization objectives. As mentioned before, we use a linear utility function for MOMDP, so any single-objective DRL algorithm is applicable [33]. Double Deep Q-Networks (DDQN) [34] utilizes the experience replay to store and reuse past experiences for the policy update [35]. It is beneficial for network reconfiguration because it can reduce the search space of all possible reconfiguration command sequences. Furthermore, as an off-policy algorithm, it can reduce the practical runtime due to the off-line training. DDQN can also address the over-estimation bias problem found in traditional Q-learning [36]. Therefore, DDQN is a good algorithm to solve reconfiguration optimization problem.

The principle of DDQN commences with Deep Q-Network (DQN). DQN is a multi-layered neural network that for a given state s outputs the action-value function approximation $Q(s, a, \theta)$, where θ are the parameters of the network. Two important ingredients of the DQN algorithm are the use of experience replay and a target network. The experience replay stores tuples of the agent's experience $(s_t, a_t, \mathbf{r}_t, s_{t+1})$ in a replay buffer $\mathcal{D}$ for mini-batch training at each learning step.

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on October 30,2025 at 11:22:59 UTC from IEEE Xplore. Restrictions apply.

TABLE I: The rules of embedding a fact base $\mathcal{B}$ to node features in the state graph.

Symbol	Formula	Description
$\texttt{Gemb}(b)$	$:= \mathbf{T}_b$	fact type embedding
$\texttt{Gemb}(o)$	$:= \mathbf{V}_{\text{bool}} \cdot \text{onehot}(o)$	boolean embedding
$\texttt{Gemb}(b, i, v)$	$:= \mathbf{I}_{b_i} \cdot \text{onehot}(v)$	integer embedding
h_c	$:= \sum_{b(c)\in\mathcal{B}} \texttt{Gemb}(b)$	constant embedding
$h_{b(a_0,\ldots,a_n)}$	$:= \texttt{Gemb}(b) + \texttt{Gemb}(\texttt{Bool}[b(a_0,\ldots,a_n)]) + \sum_{(i,v)\in I(b(a_0,\ldots,a_n))} \texttt{Gemb}(b,i,v)$	fact embedding
$\mathcal{I}(b(a_0,\ldots,a_n))$	$:= \{(i,v) \mid b(a_0,\ldots,a_{i-1},v,\ldots,a_n) \in \mathcal{B} \wedge v \in \text{Int} \wedge i \in \mathbb{N}\}$	integer parameters
$N_i(h_c)$	$:= \{h_b \mid \exists i \in \mathbb{N}. b(a_0,\ldots,a_{i-1},c,a_{i+1},\ldots,a_n) \in \mathcal{B}\}$	constant node neighbors
$N_i(h_b)$	$:= \{h_c \mid \exists i \in \mathbb{N}. b(a_0,\ldots,a_{i-1},c,a_{i+1},\ldots,a_n) \in \mathcal{B}\}$	fact node neighbors

The target network with parameters θ^- periodically copies the parameters θ of the online network to reduces the variance of the updates. So that $\theta_t^- = \theta_t$, and the target value is:

$$Y_t^{\text{DQN}} = r_{t+1} + \gamma \arg\max_a \ Q(s_{t+1}, a; \theta_t^-). \tag{7}$$

The only difference between DQN and DDQN is that DDQN decouples the selection from the evaluation. In DDQN, two value functions are learned by assigning experiences randomly to update one of the two value functions. This operation results in two sets of weights, θ and θ'. For each update, one set of weights is used to determine the greedy policy and the other to determine its value. Its target value is:

$$Y_t^{\text{DDQN}} = r_{t+1} + \gamma Q(s_{t+1}, \arg\max_a \ Q(s_{t+1}, a; \theta_t); \theta_t'). \tag{8}$$

However, standard DDQN still face challenges of model accuracy and convergence speed. It is also limited in learning within large action spaces.

2) MOMDP for Reconfiguration: We now present the formulation of the MOMDP for our defined reconfiguration optimization problem, including the state space $\mathcal{S}$, the action space $\mathcal{A}$ and the reward space $\mathcal{R}$.

a) State Space $\mathcal{S}$: The state space is defined as a tuple of state graphs $\mathcal{S} = (\mathcal{G}_c, \mathcal{G}_t)$, where $\mathcal{G}_c$ represents the current state graph, and $\mathcal{G}_t$ denotes the target state graph.

b) Action Space $\mathcal{A}$: The action space $\mathcal{A}$ is defined as a finite set of update commands that can be applied for the transition from the current state graph $\mathcal{G}_c$ to the target state graph $\mathcal{G}_t$. An action $a \in \mathcal{A}$ can be an addition a_{add}, a deletion a_{delete}, or a modification a_{modify} by updating the facts within the current state graph:

$$\mathcal{A} = \{a_{\text{add}}(b_{\text{new}}), a_{\text{delete}}(b), a_{\text{modify}}(b, b_{\text{new}})\}, \tag{9}$$

where b_{new} denotes the desired fact. An addition action a_{add} involves inserting a new configuration entry to $\mathcal{G}_c$. For example, $a_{\text{add}}(\texttt{connected}(A, B, 10))$ implies adding a link with an OSPF weight of 10 between routers A and B. A deletion action a_{delete} removes an existing configuration entry. Take $a_{\text{delete}}(\texttt{br}(AS100, \texttt{192.168.0.0/16}, lp : 10))$ as an example, the BGP route entry is removed for network $\texttt{192.168.0.0/16}$ originating from $AS100$ with the local priority of 10. A modification action a_{modify} updates an existing entry to a new value. For instance, $a_{\text{modify}}(\texttt{fwd}(A, \texttt{10.0.0.0/8}, B), \texttt{fwd}(A, \texttt{10.0.0.0/8}, C))$ changes the next-hop router of traffic forwarding from router A for network $\texttt{10.0.0.0/8}$ from B to C.

c) Reward Space $\mathcal{R}$: The reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is designed to reflect the immediate feedback for each action to be taken during the network reconfiguration process. We define four different rewards, three of them for multiple optimization objectives and one for completing the transition to the target state. Therefore, at each time interval t, the reward vector is defined as $\mathbf{R}_t = \{r_t^{\text{traffic}}, r_t^{\text{changes}}, r_t^{\text{specs}}, r_t^{\text{target}}\}$.

The first component measures the traffic shifts due to the transition. It is defined as:

$$r_t^{\text{traffic}} = -f_t, \tag{10}$$

where f_t is the total traffic shifts from time t to $t-1$ respectively. They are defined in Eq. (1). The negative value means the penalty for increased traffic shifts.

The second component measures the number of reconfiguration commands at time t. It equals the negative of the number of update commands in action a_t. By using the negative value, the optimizer prioritizes solutions that minimize the number of reconfiguration commands. It is defined as:

$$r_t^{\text{changes}} = -|a_t| \tag{11}$$

The third component measures the change in the compliance with network specifications Φ. We define it as the difference between the specification consistency of network state at t and its previous state:

$$r_{\text{specs}} = \frac{\sum_{\phi\in\Phi} \mathcal{I}(s_t, \phi) - \sum_{\phi\in\Phi} \mathcal{I}(s_{t-1}, \phi)}{|\Phi|}, \tag{12}$$

where $\mathcal{I}(\cdot)$ is an indicator function that returns 1 if current network state satisfies specification ϕ and 0 otherwise.

The final component provides a binary reward to indicate if the updated configuration matches the target configuration:

$$r_{\text{target}} = \begin{cases} 1 & \text{if } s_t \supseteq \mathcal{G}_t, \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

These components are then scalarized by using a linear utility function with a normalization process according to Eq. (6). Therefore, the final reward function is:

$$\begin{aligned} \mathbf{R}_t = &\eta_1 \cdot \text{Norm}(r_t^{\text{traffic}}) + \eta_2 \cdot \text{Norm}(r_t^{\text{changes}}) \\ &+ \eta_3 \cdot \text{Norm}(r_t^{\text{specs}}) + \eta_4 \cdot \text{Norm}(r_t^{\text{target}}), \end{aligned} \tag{14}$$

where $\eta_1, \eta_2, \eta_3, \eta_4$ are the normalized weighting factors assigned to each component. They satisfies the condition

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on October 30,2025 at 11:22:59 UTC from IEEE Xplore. Restrictions apply.

$\eta_1 + \eta_2 + \eta_3 + \eta_4 = 1$. The values of the weights are set according to different requirements. The normalization function Norm$(\cdot)$ maps the raw reward values to the same scale, typically $[0, 1]$, to facilitate the aggregation of different reward components into a single utility value.

3) DPER-DDQN Algorithm: To address the limitations of standard DDQN, we propose a DPER-DDQN algorithm as our decision-making algorithm. It improves DDQN in terms of the replay strategy and the network architecture.

Priotitized Experience Replay (PER). Considering that the network reconfiguration task involves a significant number of state transitions, some of which are more critical than others. To ensure the agent quickly obtain primary states and new learning experiences for the adaptation of new environment, we adopt PER [37] to replace the uniform sampling of DDQN. DPER-DDQN assigns a higher sampling probability to transitions with higher temporal difference (TD) errors, which measures the difference between the predicted reward and the actually received reward after taking an action in a given state [38]. For a transition at time t, the TD error δ_t is:

$$\delta_t = \mathbf{r}_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-) - Q(s_t, a_t; \theta), \tag{15}$$

where $Q(s_t, a_t; \theta)$ is the current action-value function approximation with the parameters θ, $Q(s_{t+1}, a'; \theta^-)$ is the target action-value function with the parameters θ^-, and γ is the discount factor.

The priority p_i for the i-th transition is then set proportional to its TD error:

$$Pr(i) \propto p_i^{\delta_t}, \tag{16}$$

where $Pr(i)$ is the probability of sampling the i-th transition from the replay buffer. δ_t governs the degree of prioritization. If $\delta_t = 0$, it is a uniform sampling.

Dueling Network Architecture. DDQN is limited in learning within large action spaces. The dueling network architecture separates the representation of state values and action advantages to quickly identify the optimal action during policy evaluation and minimize redundant decision-making processes [39]. The action advantages are estimated by the advantage function that measures whether or not an action is better or worse than the policy's default behavior [40]. The output of the dueling network architecture is:

$$Q(s, a; \theta, \alpha_Q, \beta_Q) = V(s; \theta, \beta_Q) + \left(A(s, a; \theta, \alpha_Q) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a'; \theta, \alpha_Q) \right), \tag{17}$$

where $V(s; \theta, \beta)$ is the value function, and $A(s, a; \theta, \alpha)$ is the advantage function, with α_Q and β_Q denoting the parameters of the respective streams within the neural network.

The proposed DPER-DDQN algorithm is summarized in Algorithm. 1. The process begins with initializing two dueling Q-networks, a replay buffer $\mathcal{D}$, the priority degree δ_t, the exploration rate ϵ and the update frequency for the target network C (line 1). During each episode (from 1 to E), the network starts with an initial state (line 3). For each timestep $t = 1, \ldots, T$, the action a_t is selected according to the ϵ-greedy policy (line 5). The next state s_{t+1} and the gained reward r_t are then stored in $\mathcal{D}$ with the maximal priority (line 7). A mini-batch of experiences is sampled according to their priority $Pr(i)$ (line 8), and then TD-error are computed to update priorities (line 9). The dueling network architecture computes the target Q-values Y_t^{DDQN} (line 10), and the loss $\mathcal{L}(\theta)$ is minimized to update the network parameters θ by a gradient descent. The loss function is defined as mean square errors (MSE): $\mathcal{L}(\theta) = \frac{1}{m} \cdot \sum_{j=1}^{m} (Y_t^{\text{DDQN}} - Q(s_t, a_t; \theta))^2$, where m is size of mini-batch. Every C steps, the target network's parameters θ^- are updated to match those of the online network θ (line 12). The exploration rate ϵ is gradually reduced until $\epsilon = \epsilon_{\text{min}}$ (line 14).

Algorithm 1 Dueling PER DDQN-based Reconfiguration Optimization Algorithm (DPER-DDQN).

1: Initialize the dueling Q-networks with parameters θ and θ', the replay buffer $\mathcal{D}$ with fixed capacity N, the priority degree δ_t, the exploration rate ϵ, and the target network update frequency C
2: **for** episode $= 1$ to E **do**
3: Initialize state s
4: **for** timestep $t = 1$ to T **do**
5: Select action a_t utilizing the ϵ-greedy policy from $Q(s_t, a_t; \theta)$.
6: Execute action a_t and obtain the reward r_t and the new state s_{t+1}
7: Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$ with maximum priority
8: Sample mini-batch from $\mathcal{D}$ based on priority probability $Pr(i)$
9: Calculate TD-error δ_t using Eq. (15) for a batch and update priorities using Eq. (16)
10: Calculate target values Y_t^{DDQN} using Eq. (8)
11: Update θ by minimizing the loss $\mathcal{L}(\theta)$
12: Every C steps, update θ^- to θ
13: **end for**
14: Decay ϵ with the decay rate ϵ_{decay} until $\epsilon = \epsilon_{\text{min}}$
15: **end for**

V. Experimental setup and results

In this section, we present the experimental setup, and compare MONR with three benchmark reconfiguration methods, including Snowcap [4], AED [17], and ConfigReco [21].

A. Setup

We focus on two most widely used protocols: OSPF and BGP [41] in our evaluation. MONR supports standard Cisco commands for configurations and can easily be extended to support other languages. The optimization objective is to minimize the traffic shift and the number of configuration changes while maximizing the specification consistency. For

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on October 30,2025 at 11:22:59 UTC from IEEE Xplore. Restrictions apply.

the defined MOMDP, we use a fully-connected network with ReLU activations. By utilizing a grid search strategy [42], which is a method for parameter tuning, we find that the best performance is achieved by the parameters of DPER-DDQN algorithm outlined in Table. II. The network utilizes the Adam optimizer [43] for training. We assume that the traffic shifts and specification consistency are more significant than the number of reconfiguration commands during transition in our evaluation experiments. We set appropriate weight $\eta_2 = 0.1$ for r_t^{changes} according to the result of sensitivity analysis.

TABLE II: Parameters of DPER-DDQN.

Parameter	Value
Hidden dimension D_{hidden}	$\{64, 128, 64\}$
Learning rate lr	0.001
Batch size m	256
Discount factor γ	0.99
Episodes E	200
Timestep T	50
Initial exploration rate $\epsilon_{\text{initial}}$	1.0
Exploration decay rate ϵ_{decay}	0.1
Minimum exploration rate ϵ_{min}	0.05
Replay buffer size N	10000
Target network update frequency C	1000
Activation function	ReLU
$\eta_1, \eta_2, \eta_3, \eta_4$	0.3,0.1,0.3,0.3

We train our optimizer from a set of 106 network topologies (16-754 routers) [10] from Topology Zoo [44]. For each network topology, we randomly generate 5 types of specifications, including `fwd`, `reach`, `ecmp`, `order`, and `isolate`. Each type comprises 32 distinct specifications. Then, we apply a random strategy to generate configurations. This strategy utilizes a synthesizer NetComplete [41] to synthesize an original configuration which comply with all specifications for each topology. Furthermore, a configuration verifier Batfish [45] is used to check if all specifications are satisfied. If not, we iteratively replace configuration parameters with randomly generated alternatives until all specifications can be satisfied. After that, we generate target configurations for original configurations. We choose arbitrary actions, including adding, removing or modifying, to change 50% parameters of each original configuration. These revised configurations should fulfill all specifications.

The reconfiguration scenarios for evaluation are totally different from those used for training. Specifically, we choose a set of 80 topologies (31-196 routers) from Topology Zoo as the evaluation topology set, which excludes any topologies utilized for training. For each selected network topology, we appoint 3 route reflectors and utilize NetComplete to generate 1000 different IGP configurations. We always select the router with the highest link count as the route reflector (following best practices [46]). Each network connects to three specific neighbors e_x , e_1, and e_2, which advertise the identical FEC. However, the sessions towards e_x have the least preference. For the target configurations, we remove all eBGP sessions towards e_2. We consider 1200 waypoint specifications (`reach(·)`) that all traffic must be redirected towards either e_1 or e_2 during reconfiguration.

MONR are implemented in Python code on an Intel(R) i9-13900HX CPU @ 2.20GHz with 16GB of system memory and an NVIDIA RTX 4060 GPU with 16GB of video memory.

B. Evaluation Results

We evaluate the performance of MONR to illustrate its scalability, safety, and practicality by comparing it with three latest reconfiguration methods: Snowcap [4], AED [17], and ConfigReco [21].

1) Scalability: To assess the scalability, we record the schedule time (ST), i.e., the time needed to obtain update commands, and the reconfiguration time (RT), which is the time required to implement these update commands. The runtime is the total time taken for the reconfiguration tasks, which is computed by adding ST and RT for MONR, Snowcap, and AED. For ConfigReco, the runtime is the sum of the times for knowledge graph construction and configuration recommendation. We analyze two factors which impact the runtime: network scale, and specification complexity.

Network Scale. It is quantified by the number of routers within a topology. We utilize 6 networks (Switch (30), Arnes(34), Uninett(69), Columbus(70), Colt(153), US Carrier(158)) from the evaluation topology set. As shown in the bars of Fig 2, all methods exhibit an increasing trend in runtime with larger network scales. The time increase of MONR is reasonable. This is because MONR only takes around 25s in the largest network (158 routers) while Snowcap, AED and ConfigReco take about 53s, 233s, and 1400s, respectively. That is to say, MONR is 2x, 9x, and 56x faster than Snowcap, AED and ConfigReco, separately. Furthermore, MONR is slower than Snowcap in a small network (11 routers) due to the graph representation time required by MONR. However, this disadvantage is eliminated as the network scale increases. When the network scale is 69, MONR is faster than Snowcap. The prominent time efficiency of MONR in large networks is crucial for managing modern large networks. This efficiency attributes to the off-training strategy of DRL techniques utilized in MONR, which do not need to retrain in actual application once trained.

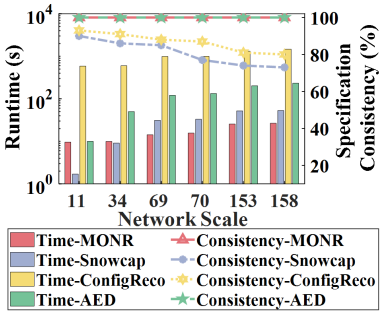

Fig. 2: The figure of comparison for runtime and specification consistency with increasing network scales.

Specification Complexity. To evaluate the effects of specification complexity on runtime performance, we record the

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on October 30,2025 at 11:22:59 UTC from IEEE Xplore. Restrictions apply.

runtimes for various specification sizes and types. These experiments are performed on the Columbus network (70 routers). For evaluating the impact of specification size, we define the size as the number of specifications (from 0 to 120). To analyze the impact of specification type, the specification size is fixed at 30 and we consider three types of specifications with different complexity levels (T1>T2>T3): T1 is reachability (`fwd(·)`), T2 is waypoint (`reach(·)`), and T3 is the path preference (`order(·)`). The results are presented in Fig. 3.

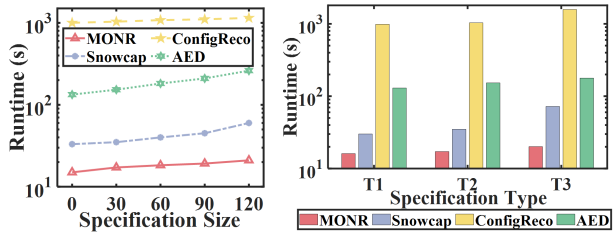

Fig. 3: The figure of runtime impacted by specification complexity, which is measured by the specification size and type.

As depicted in Fig. 3a, although the runtimes for all methods increase with the growing specification size, MONR maintains the shortest runtime (about 20s) even when processing 120 specifications. This demonstrates that MONR can scale well to handle multiple specifications without a linear decline in performance. Furthermore, Fig. 3b shows that all reconfiguration methods take the most time to process path preference specifications and the least time for reachability specifications. This is because path preference specifications involve more complex constraints and computations, thereby prolonging runtime. Regardless of the specification type, MONR consistently outperforms other reconfiguration methods. This benefits from the rapid processing of complex scenarios enabled by the DRL techniques implemented within it.

2) Safety: The safety is measured by specification consistency, which is defined as O_{specs} in Eq. (1). We conduct experiments under the same settings as those used for runtime comparisons across various network scales. The results are shown in the lines of Fig. 2. It demonstrates that both MONR and AED consistently achieve perfect specification consistency (100%) regardless of network size. In contrast, ConfigReco and Snowcap exhibit lower consistencies of approximately 93% and 90%, respectively, even in the small network (11 routers). These consistencies decrease to 80% and 75% as the network scale increases to 158. For critical (e.g., Service Level Agreements (SLAs)- or security-related) requirements, such specification inconsistency is problematic [10] because it can lead to severe disruptions in network services, including downtimes and packet loss during reconfiguration. This conclusion can be further validated in the following throughput comparison experiments.

3) Practicality: The practicality is assessed by two metrics: traffic shifts and the number of reconfiguration commands, which are denoted as O_{traffic} and O_{changes} in Eq. (1). In our evaluation, we use all topologies from the evaluation topology set, and label them in ascending order by their network scales with a unique number index. For example, the index of the smallest network topology is 1. To evaluate O_{changes}, we introduce an extra comparative ratio, $\frac{O_{\text{changes\_MONR}}}{O_{\text{changes\_benchmark}}}$, where $O_{\text{changes\_benchmark}}$ represents the number of update changes for each benchmark method. A ratio of 1 indicates that MONR and the baseline method require the same number of changes. If the ratio is less than 1, MONR requires fewer reconfiguration commands compared to the baseline; otherwise, it requires more. The results are shown in Fig. 4.

From the traffic shift comparison illustrated in Fig. (4a), MONR achieves the lowest traffic shifts in nearly 80% of evaluated topologies. This advantage is more pronounced as the network scale increases. For instance, in the largest network topology (the network index is 80), the traffic shifts of MONR is nearly 0.1 while other methods' are over 1.5. As illustrated in Figs. 4b, MONR performs better than Snowcap (the majority of ratios are around 0.8) and largely surpasses ConfigReco (all ratios are below 0.5). Although MONR achieves a comparable number of commands with AED (the ratio is fluctuated around 1), it requires slightly fewer commands than AED in about 85% of the network topologies. Furthermore, a noticeable decreasing trend in overall ratios indicates MONR's enhanced optimization capability in larger network topologies.

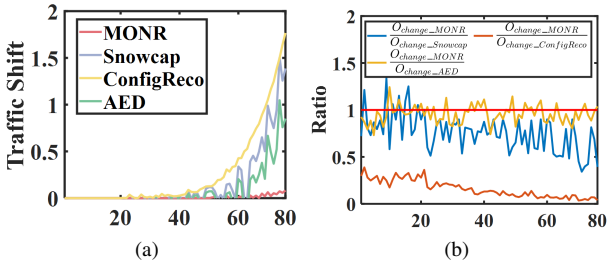

Fig. 4: The figures of comparing MONR with three baselines in terms of traffic shifts (a) and the number of reconfiguration commands (b). The red line in (b) represents comparative ratio is 1, i.e, equal number of reconfiguration commands.

4) Throughput Comparison: To further valid the effectiveness of MONR, we conduct the experiment of throughput comparison on a small Abilene network with 11 routers. We exclude ConfigReco from this experiment as it directly synthesizes desired configurations without considering the transition process during reconfiguration. The results are present in Figure 5. Although Snowcap only takes 1.7s to finish the reconfiguration task, it drops about 15.2k packets for 1s of traffic (the packet loss rate is about 92%). In contrast, both AED and MONR complete the reconfiguration task without dropping any packet, but MONR (9.58s) is faster than AED (10s) and 75% of MONR's runtime is used for graph representation.

5) Additional Experiments: We perform additional experiments to assess the adaptability of MONR to handle more routing protocols except for BGP and OSPF. We select two

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on October 30,2025 at 11:22:59 UTC from IEEE Xplore. Restrictions apply.

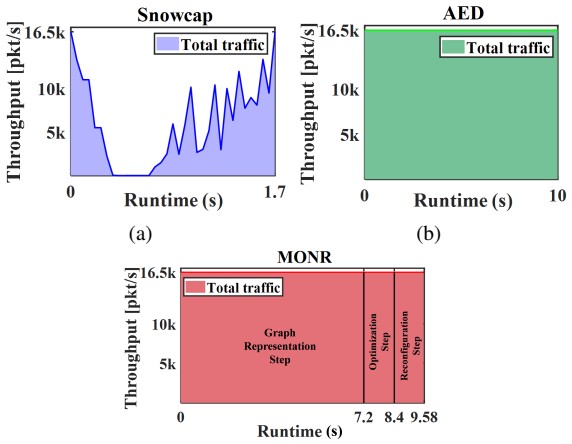

Fig. 5: The figures of throughput comparison.

additional protocols: (i) Intermediate System to Intermediate System (IS-IS) [47], which is an IGP based on a link-state routing algorithm; and (ii) Enhanced Interior Gateway Routing Protocol (EIGRP) [48], which is a hybrid routing protocol that integrates features of both distance-vector and link-state protocols. As described in Section. IV-A, these protocols are represented as follows: $\texttt{connected}(u, v, c)$ for IS-IS and $\texttt{connected}(u, v, [k])$ for EIGRP, where u and v are routers, h is the hop count, c denotes the cost metric, and $[k]$ represents the vector of combined metrics, such as delay and bandwidth.

TABLE III: The performance of MONR across IS-IS and EIGRP.

Topology	Protocol	Runtime (s)	Spec. Consistency
Arnes	IS-IS	9.82	100%
	EIGRP	10.53	100%
Columbus	IS-IS	15.02	100%
	EIGRP	15.72	100%
US Carrier	IS-IS	28.35	100%
	EIGRP	31.23	100%

We perform experiments on three network topologies: Arnes (34), Columbus (70) and US Carrier (158) for IS-IS-only and EIGRP-only reconfiguration tasks. It means we only consider a single protocol at one time. For each topology in the IS-IS-only tasks, we use the same random strategy for configuration generation described in Section V-A to generate 1000 IS-IS configurations. In each target configurations, we randomly choose 50 links to change their link costs. We consider reachability specifications that any router can reach another arbitrary router in the network. The similar setups are considered for EIGRP-only reconfiguration tasks. We evaluate the runtime and specification consistency. The results are concluded in Table. III. These results show that MONR still maintains 100% specification consistency and reasonable runtime (< 40s even in large network) for both IS-IS-only and EIGRP-only reconfiguration tasks. It highlights the adaptability of MONR in scaling to handle various routing protocols.

6) Sensitivity Analysis: We conduct a sensitivity analysis to identify the optimal weights for our multi-objective optimization. We employ a typical One-at-a-time (OAT) method [49] by modifying one weight at a time while keeping others fixed. Initially, all weights are set to 0.25, and we adjust each weight by increments of 0.05. This analysis is performed on the US Carrier network (158). We record all results of specification consistency, traffic shifts and the number of reconfiguration commands. Given the extensive data collected, we summarize the principal findings in Table IV. The results indicate that $\eta_1, \eta_2, \eta_3, \eta_4 = 0.3, 0.1, 0.3, 0.3$ achieves the highest specification consistency (100%) while maintaining competitive performance in traffic shifts and the number of commands, with a higher priority given to specification consistency and traffic shifts than to the number of update commands.

TABLE IV: Sensitivity Analysis.

η_1	η_2	η_3	η_4	O_{specs} (%)	O_{traffic}	O_{change}
0.25	0.25	0.25	0.25	97	0.072	215
0.1	0.3	0.3	0.3	81	0.061	206
0.3	**0.1**	**0.3**	**0.3**	**100**	0.062	218
0.3	0.3	0.1	0.3	**100**	0.12	**203**
0.3	0.3	0.3	0.1	85	**0.053**	208

VI. Conclusion

This paper introduces MONR, a scalable and rapid reconfiguration method to automate the generation of update command sequences. These sequences can ensure safe and practical reconfiguration by balancing three optimization objectives: specification consistency, traffic shifts and the number of update commands. This is because the implemented DPER-DDQN algorithm can explore various update command sequences and evaluate the long-term rewards of them to find the sequence with the optimal reward. By doing so, MONR ensures the safety even in transient configuration states during reconfiguration. MONR is scalable to support any routing protocols and specifications since the applied Datalog-like facts are irrelevant with input formats. It can also be applied in large networks, which is challenging for existing methods. This attributes to the ability of handling high-dimensional data for the DPER-DDQN algorithm. Furthermore, the runtime of MONR is reduced due to the utilization of an off-training strategy to avoid retraining during real application. The evaluation shows that MONR is 2x, 9x and 56x faster than Snowcap, AED and ConfigReco, respectively. Moreover, it achieves 100% specification consistency, maintains traffic shifts below 0.1 and requires relative few update commands (0.8 of Snowcap's) in large networks. Therefore, MONR provides network operators with a powerful tool for safe and practical router reconfiguration. However, the applicability of MONR for configurations of other techniques (e.g., SDN), and network functions (e.g., firewalls) should be further explored.

Acknowledgements

We thank our shepherd, Deep Medhi, and the anonymous reviewers for their valuable and thoughtful comments.

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on October 30,2025 at 11:22:59 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] H. Kim, T. Benson, A. Akella, and N. Feamster, "The evolution of network configuration: A tale of two campuses," in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, 2011, pp. 499–514. I

[2] G. Malkin, "Rfc2453: Rip version 2," 1998. I

[3] J. Moy, "Rfc2328: Ospf version 2," 1998. I

[4] T. Schneider, R. Birkner, and L. Vanbever, "Snowcap: synthesizing network-wide configuration updates," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 33–49. I, II, III, V, V-B

[5] S. Vissicchio, L. Vanbever, C. Pelsser, L. Cittadini, P. Francois, and O. Bonaventure, "Improving network agility with seamless bgp reconfigurations," *IEEE/ACM Transactions on Networking (TON)*, vol. 21, no. 3, pp. 990–1002, 2013. I, II

[6] J. P. John, E. Katz-Bassett, A. Krishnamurthy, T. Anderson, and A. Venkataramani, "Consensus routing: The internet as a distributed system," *Proc. NSDI (April 2008)*, 2008. I, II

[7] L. Vanbever, S. Vissicchio, L. Cittadini, and O. Bonaventure, "When the cure is worse than the disease: The impact of graceful igp operations on bgp," in *2013 Proceedings IEEE INFOCOM*. IEEE, 2013, pp. 2220–2228. I, II

[8] L. Vanbever, S. Vissicchio, C. Pelsser, P. Francois, and O. Bonaventure, "Seamless network-wide igp migrations," in *Proceedings of the ACM SIGCOMM 2011 Conference*, 2011, pp. 314–325. I, II

[9] Y. Rekhter, T. Li, and S. Hares, "Rfc 4271: A border gateway protocol 4 (bgp-4)," 2006. I

[10] T. Schneider, R. Schmid, S. Vissicchio, and L. Vanbever, "Taming the transient while reconfiguring bgp," in *Proceedings of the ACM SIGCOMM 2023 Conference*, 2023, pp. 77–93. I, II, III, V-A, V-B2

[11] A. Bednarz, "Global microsoft cloud-service outage traced to rapid bgp router updates," https://www.networkworld.com/article/3686531/global-microsoft-cloud-service-outage-traced-to-rapid-bgp-router-updates.html, 2023, accessed: Jan. 2023. I

[12] K.-T. Foerster, S. Schmid, and S. Vissicchio, "Survey of consistent software-defined network updates," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1435–1461, 2018. I

[13] L. Beurer-Kellner, M. Vechev, L. Vanbever, and P. Veličković, "Learning to configure computer networks with neural algorithmic reasoning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 730–742, 2022. I, IV-A, IV-A

[14] F. Clad, S. Vissicchio, P. Mérindol, P. Francois, and J.-J. Pansiot, "Computing minimal update sequences for graceful router-wide reconfigurations," *IEEE/ACM Transactions on Networking*, vol. 23, no. 5, pp. 1373–1386, 2014. II

[15] P. Francois, M. Shand, and O. Bonaventure, "Disruption free topology reconfiguration in ospf networks," in *IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications*. IEEE, 2007, pp. 89–97. II

[16] R. Alimi, Y. Wang, and Y. R. Yang, "Shadow configuration as a network management primitive," in *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2008. [Online]. Available: https://api.semanticscholar.org/CorpusID:2769850 II

[17] A. Abhashkumar, A. Gember-Jacobson, and A. Akella, "Aed: Incrementally synthesizing policy-compliant and manageable configurations," in *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, 2020, pp. 482–495. II, V, V-B

[18] U. Naseer and T. A. Benson, "Configanator: A data-driven approach to improving {CDN} performance." in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 1135–1158. II

[19] C. Zhao, T. Chugh, J. Min, M. Liu, and A. Krishnamurthy, "Dremel: Adaptive configuration tuning of rocksdb kv-store," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 6, no. 2, pp. 1–30, 2022.

[20] C. Ge, Z. Ge, X. Liu, A. Mahimkar, Y. Shaqalle, Y. Xiang, and S. Pathak, "Chroma: Learning and using network contexts to reinforce performance improving configurations," in *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, 2023, pp. 1–16. II

[21] Z. Guo, F. Li, J. Shen, T. Xie, S. Jiang, and X. Wang, "Configreco: Network configuration recommendation with graph neural networks," *IEEE Network*, 2023. II, V, V-B

[22] Z. Li, X. Wang, L. Pan, L. Zhu, Z. Wang, J. Feng, C. Deng, and L. Huang, "Network topology optimization via deep reinforcement learning," *IEEE Transactions on Communications*, 2023. II

[23] S. Salman, C. Streiffer, H. Chen, T. Benson, and A. Kadav, "Deepconf: Automating data center network topologies management with machine learning," in *Proceedings of the 2018 Workshop on Network Meets AI & ML*, 2018, pp. 8–14. II

[24] C. Huitema, *Routing in the Internet*. Prentice-Hall, Inc., 1995. III

[25] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 92–99, 2010. III

[26] T. G. Griffin and J. L. Sobrinho, "Metarouting," in *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, 2005, pp. 1–12. IV-A

[27] C.-L. Hwang and A. S. M. Masud, *Multiple objective decision making—methods and applications: a state-of-the-art survey*. Springer Science & Business Media, 2012, vol. 164. IV-B

[28] P. Vamplew, C. Foale, C. F. Hayes, P. Mannion, E. Howley, R. Dazeley, S. Johnson, J. Källström, G. Ramos, R. Rădulescu *et al.*, "Utility-based reinforcement learning: Unifying single-objective and multi-objective reinforcement learning," *arXiv preprint arXiv:2402.02665*, 2024. IV-B1

[29] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015. IV-B1

[30] Y. Yu, J. Tang, J. Huang, X. Zhang, D. K. C. So, and K.-K. Wong, "Multi-objective optimization for uav-assisted wireless powered iot networks based on extended ddpg algorithm," *IEEE Transactions on Communications*, vol. 69, no. 9, pp. 6361–6374, 2021. IV-B1

[31] A. Staffolani, V.-A. Darvariu, L. Foschini, M. Girolami, P. Bellavista, and M. Musolesi, "Prorl: Proactive resource orchestrator for open rans using deep reinforcement learning," *IEEE Transactions on Network and Service Management*, 2024. IV-B1

[32] K. Chatterjee, R. Majumdar, and T. A. Henzinger, "Markov decision processes with multiple objectives," in *STACS 2006: 23rd Annual Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 23-25, 2006. Proceedings 23*. Springer, 2006, pp. 325–336. IV-B1

[33] C. F. Hayes, R. Rădulescu, E. Bargiacchi, J. Källström, M. Macfarlane, M. Reymond, T. Verstraeten, L. M. Zintgraf, R. Dazeley, F. Heintz *et al.*, "A practical guide to multi-objective reinforcement learning and planning," *Autonomous Agents and Multi-Agent Systems*, vol. 36, no. 1, p. 26, 2022. IV-B1, IV-B1

[34] J. Fan, Z. Wang, Y. Xie, and Z. Yang, "A theoretical analysis of deep q-learning," in *Learning for dynamics and control*. PMLR, 2020, pp. 486–489. IV-B1

[35] X. Wu, R. Li, Z. He, T. Yu, and C. Cheng, "A value-based deep reinforcement learning model with human expertise in optimal treatment of sepsis," *npj Digital Medicine*, vol. 6, no. 1, p. 15, 2023. IV-B1

[36] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016. IV-B1

[37] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015. IV-B3

[38] H. Song, Y. Liu, J. Zhao, J. Liu, and G. Wu, "Prioritized replay dueling ddqn based grid-edge control of community energy storage system," *IEEE Transactions on Smart Grid*, vol. 12, no. 6, pp. 4950–4961, 2021. IV-B3

[39] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1995–2003. IV-B3

[40] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015. IV-B3

[41] A. El-Hassany, P. Tsankov, L. Vanbever, and M. Vechev, "{NetComplete}: Practical {Network-Wide} configuration synthesis with autocompletion," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 579–594. V-A, V-A

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on October 30,2025 at 11:22:59 UTC from IEEE Xplore. Restrictions apply.

[42] P. Liashchynskyi and P. Liashchynskyi, "Grid search, random search, genetic algorithm: a big comparison for nas," *arXiv preprint arXiv:1912.06059*, 2019. V-A
[43] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014. V-A
[44] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011. V-A
[45] A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan, R. Mahajan, and T. Millstein, "A general approach to network configuration analysis," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015, pp. 469–483. V-A
[46] B. R. Greene and P. Smith, *Cisco ISP essentials*. Cisco Press, 2002. V-A
[47] R. W. Callon, "RFC1195: Use of OSI IS-IS for routing in tcp/ip and dual environments," 1990. V-B5
[48] D. Savage, J. Ng, S. Moore, D. Slice, P. Paluch, and R. White, "Cisco's enhanced interior gateway routing protocol (EIGRP)," Tech. Rep., 2016. V-B5
[49] S. Razavi and H. V. Gupta, "What do we mean by sensitivity analysis? the need for comprehensive characterization of "global" sensitivity in Earth and Environmental systems models," *Water Resources Research*, vol. 51, no. 5, pp. 3070–3092, 2015. V-B6

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on October 30,2025 at 11:22:59 UTC from IEEE Xplore. Restrictions apply.