Neural Packet Routing

Shihan Xiao, Haiyan Mao, Bo Wu, Wenjie Liu, Fenglin Li
Network Technology Lab, Huawei Technologies Co., Ltd, Beijing, China
{xiaoshihan,maohaiyan1,wubo.net,liuwenjie15,lifenglin}@huawei.com

ABSTRACT

Deep learning has shown great potential in automatically generating routing protocols for different optimization objectives. Although it may bring superior performance gains, there exists a fundamental obstacle to prevent existing network operators from deploying it into a real-world network, i.e., the uncertainty of statistical nature in deep learning can not provide the certainty of basic connectivity guarantee required in real-world routing.

In this paper, we propose the first deep-learning-based distributed routing system (named NGR) that can achieve the connectivity guarantee while still attaining the routing optimality. NGR provides a novel packet routing framework based on the link reversal theory. Specially-designed neural network structures are further proposed to seamlessly incorporate into the framework. We apply NGR in the tasks of shortest-path routing and load balancing. The evaluation results validate that NGR can achieve 100% connectivity guarantee despite the uncertainty of deep learning and gain performance close to the optimal solution.

CCS CONCEPTS

• **Networks → Network protocols**; • **Computing methodologies → Artificial intelligence**.

KEYWORDS

Distributed routing, reliability, deep learning

ACM Reference Format:
Shihan Xiao, Haiyan Mao, Bo Wu, Wenjie Liu, Fenglin Li. 2020. Neural Packet Routing. In *Workshop on Network Meets AI & ML (NetAI '20), August 14, 2020, Virtual Event, NY, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3405671.3405813

1 INTRODUCTION

Achieving *scalable, reliable and optimal* routing in IP networks has become a fundamental research topic during the past tens of years. Despite of many historical proposals, distributed routing protocols like OSPF/IS-IS that support *hop-by-hop packet forwarding* have become the communication basis of today's Internet [13]. They have the advantages of simplicity, high scalability and reliability. However, they only support shortest-path routing, which is known to suffer from low resource utilization and difficult to extend to implement the *optimality* with respect to diverse objectives [14].

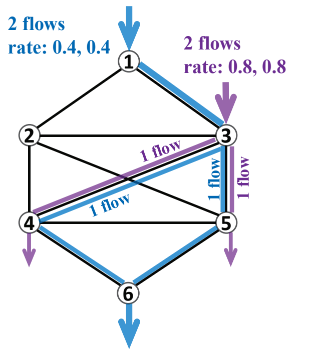
In recent years, deep learning has shown great potential for automatic control with custom optimization goals [11]. Recent research work shows the potential to implement line-rate neural network inference on the data plane of switches [17]. Despite the optimality and potential hardware feasibility, it remains challenging and potentially risky for network operators to apply learning-based hop-by-hop packet routing in practice. As the earliest attempt, Q-routing is the first well-known reinforcement-learning-based routing algorithm for QoS optimization [3] and various variants are followed [5, 10]. Early machine learning methods suffer from the instability and non-convergence issues [5]. With the advances of deep learning, recent work begins to explore the use of neural networks to perform hop-by-hop routing [7]. It successfully exhibits better performance on convergence time and robustness against link failures. However, it fails to satisfy the routing reliability and may generate persistent routing loops (see later sections). The fundamental challenge is that deep learning is not reliable due to the inevitable errors of neural networks [20], while the routing in production networks requires the complete *reliability*[1], i.e., any routing loops or black-holes are unacceptable.

Recent research show that centralized routing can avoid the reliability issues and achieve near-optimal performance with deep learning methods [19]. However, they suffer from scalability issues in today's large-scale networks. To address the scalability problem, recent proposal of semi-oblivious routing [9] is to pre-compute some loop-free paths in the offline and then distribute the traffic onto the paths online in real time. However, such methods fail to achieve routing optimality since not all the paths are available during the online selection.

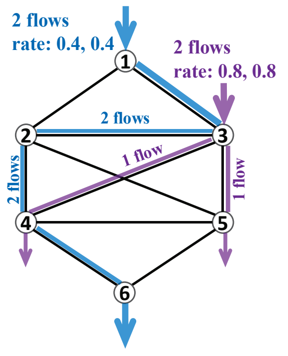
In this paper, we propose a deep-learning-based distributed routing system, named Neural-Guided Routing (NGR), to address the challenges on routing scalability, reliability and optimality. Rather than direct end-to-end deep learning in traditional machine learning domains (e.g., image processing), the main contribution of this work is combining the deep learning and theoretically-guaranteed deterministic method to address the safety of deep learning applications in the network domain. NGR provides a scalable hop-by-hop packet forwarding framework with specially-designed neural

[1]In this paper, the routing reliability is defined as there exists no routing loops and blackholes when the topology is connected.

NetAI '20, August 14, 2020, Virtual Event, NY, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8043-0/20/08...$15.00
https://doi.org/10.1145/3405671.3405813



(a) Existing routing solution with loop-free restrictions on next hop selection (max link utilization is 1.2)



(b) Optimal routing solution (max link utilization is 0.8)

Figure 1: Example of trade-off between reliability and optimality in hop-by-hop routing. Link bandwidth is set as 1. The left figure is existing routing solution where the next hops of node 3 is restricted to node 4 and 5 to avoid routing loops. This restriction leads to the loss of optimality on minimizing the maximum link utilization compared to the optimal solution on the right.

networks to achieve both the reliability and optimality. We first prove the reliability of NGR based on the link reversal theory. We then prove the power of NGR to attain the routing optimality, i.e., any loop-free optimal routing solutions can be equivalently implemented by setting a specific set of neural network weights in NGR. We evaluate the performance of NGR over different topologies, and validate its reliability guarantee and high performance close to the optimal solution in the tasks of the shortest path and load balancing.

2 MOTIVATION

Challenges in achieving scalable, reliable and optimal routing. Conventional hop-by-hop routing has advantages of high scalability and reliability, but lacks the support for optimality with respect to custom objectives. We identify two key challenges in achieving optimality in hop-by-hop routing. The first challenge comes from a fundamental trade-off between achieving the optimality and routing reliability. Specifically, achieving routing optimality requires a solution space large enough to include the optimal routing solution with respect to various custom objectives. Allowing each hop to select any its neighbor node as the next hop will trivially satisfy this requirement but easily lead to routing loops. Previous research work generally propose to restrict the selection of next hops to only the nodes that are closer to the destination than the current node (termed the *triangle constraint*) [18]. However, this restriction will shrink the solution space and is possible to exclude the optimal routing solution [14]. We give a specific example in Fig. 1.

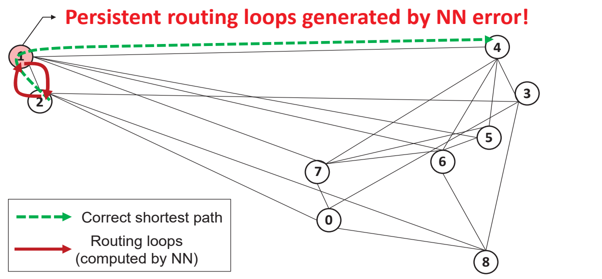

Persistent routing loops generated by NN error!
Correct shortest path
Routing loops (computed by NN)
0
1
2
3
4
5
6
7
8

Figure 2: Example of loop generated by neural networks after training with 97.5% accuracy.

This challenge requires the design of a new routing framework to construct a solution space that exactly captures all the loop-free routing solutions while excluding any routing solution with loops or blackholes.

The second challenge comes from the design of optimal algorithms for custom objectives. Previous work shows some heuristic algorithms in minimizing the total costs of transmitting the traffic [14]. However, such case-by-case heuristic design is difficult to meet diverse new demands (e.g., low latency) of fast-increasing emerging applications [15]. Hence it calls for a unified routing framework that can support high scalability, reliability and also optimality with respect to custom objectives. To support these desired functions, in this paper we present a novel hop-by-hop routing framework to maintain the routing reliability without shrinking the solution space. Then we employ deep learning under the framework to support the optimization of diverse custom objectives.

The unreliability issue introduced by deep learning. Despite the merit of flexibility, the uncertainty of deep learning will lead to unreliable routing. In practice, the successful training of a properly-structured neural network will generate *small but non-zero* errors in both the training phase and test phase.

To illustrate the unreliability issues by applying deep learning into hop-by-hop routing, we present a simple example in Fig. 2. We train a GNN (Graph Neural Network)-based routing model for a shortest-path task following the same design in [7], and apply a real network topology from the Internet topology zoo [1]. In Fig. 2, we show the routing results when the neural network model is properly trained with a high accuracy that 97.5% pairs of nodes are routing over the shortest paths. We can see that even a small error ($< 3\%$) in training is possible to generate persistent loops. We further evaluate the routing model when the training of neural network is overfitting with zero training error. In the testing phase of random link failure, we find that there exists persistent blackholes even after 200 iterations of GNN due to the generalization errors of neural networks [20].

The above example shows that the small but inevitable errors of neural networks in deep learning can lead to unreliable routing results in the real network topology. In real production networks, it would be unacceptable if any pair of nodes happens to have no connected path (e.g., persistent

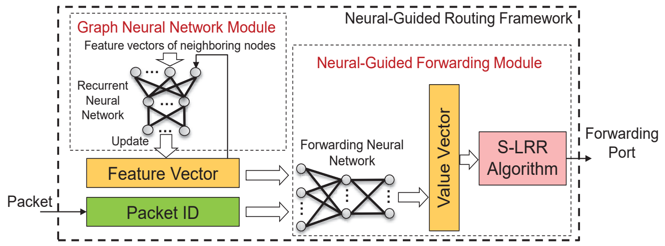

Neural-Guided Routing Framework
Graph Neural Network Module
Feature vectors of neighboring nodes
Recurrent Neural Network
Update
Feature Vector
Packet ID
Packet
Neural-Guided Forwarding Module
Forwarding Neural Network
Value Vector
S-LRR Algorithm
Forwarding Port

Figure 3: Framework of NGR.

loops or blackholes). In latter sections, we will show that a careful design of neural network structures under the new hop-by-hop routing framework can address this issue despite of neural network errors.

3 SYSTEM DESIGN

To solve the above challenges, in this section we will present the new routing framework NGR and its detailed design of each component.

3.1 Basic routing framework

Conventional routing protocols consist of two parts, i.e., the communication part to propagate the topology information and the forwarding part to select the specific forwarding port for packet routing. As Fig. 3 shows, the basic routing framework of NGR consists of two modules to implement these protocol functions.

First is the GNN (*Graph Neural Network*) module. It is responsible for the communication among routers and keeps updating the feature vector of topology information at each router. Second is the NGF (*Neural-Guided Forwarding*) module. It is responsible for selecting the forwarding port when packets arrive. One input of NGF is the feature vector of topology information. The other input of NGF is the *packet ID*, i.e., the packet destination that is used to query for routing[2]. The output of NGF is the port to forward the packet. Specifically, there are two components inside this module, i.e., the forwarding neural network (FNN) and the S-LRR algorithm. FNN is a multi-layer neural network and computes a *value vector* as an intermediate result to avoid routing loops (detailed in Section 3.3). Then the S-LRR algorithm is applied to select the final forwarding port based on the value vector. We will show that the specially-designed cooperation structure of NGF can provide both the routing reliability and full flexibility to attain the optimality despite the potential errors of neural networks.

In the following, we will present the detailed design of GNN and NGF module respectively.

[2]The packet ID can be extended based on the operator's intent to achieve different levels of routing, e.g., setting it as the 5-tuples will enable the flow-based routing.

3.2 Graph neural network module

Let $G(V, E)$ denote the network topology, where V is the node set and E is the link set. The node id $nid(v_i)$ of each node $v_i \in V$ is a one-hot vector. In GNN, each node $v_i \in V$ stores a vector $h(v_i)$ (termed the *feature vector*), which is initialized as $nid(v_i)$. Next, each node updates the value of its feature vector through aggregating the newest feature vectors of its neighboring nodes. The updating process continues for several iterations to converge to a fixed point [16], which resembles that of traditional distance-vector routing protocols. The updating step is $h_t(v) = \mathcal{F}(\{h_{t-1}(u) \mid u \in \mathcal{N}_v\})$, where $h_t(v)$ denotes the feature vector of node v at time t, $\mathcal{N}_v$ is the set of neighboring nodes of v, $\mathcal{F}(\cdot)$ is a function to aggregate the feature vectors from the neighboring nodes. Typically, the $\mathcal{F}(\cdot)$ function is implemented as a recurrent neural network [16]. The GNN module is used to update the feature vector when topology changes and then input the feature vector of current topology information into the NGF module for packet routing.

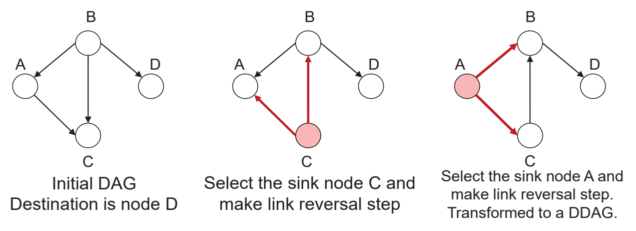
3.3 Neural-guided forwarding module

The NGF module is designed based on the *link reversal* (LR) theory. We first present the basic concepts in LR theory and then introduce how we extend it to address the neural network errors and achieve routing reliability.

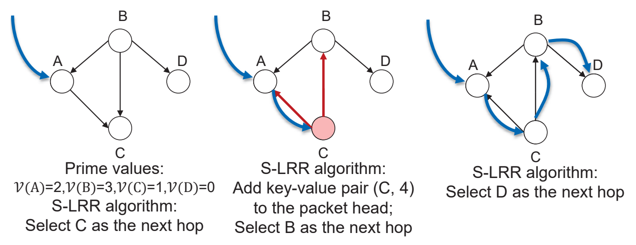
Link Reversal Theory. A graph $G(V, E)$ is a DAG (Directed Acyclic Graph) if each link in G is associated with a direction without forming directed circles. A *sink node* is denoted as the node without *out-links*, i.e., its link directions are all pointed from its neighboring nodes to itself. We denote G as a DDAG (Destination-oriented DAG) for some destination d if G is a DAG satisfying that any node v other than d is not a sink node. The property of DDAG is that any node can reach d by simply following the link directions without any loops.

One step of *link reversal* in G is denoted as the operation to randomly select one sink node s other than destination d and then reverse all the link directions of s. The LR theory proves that given any connected DAG, if we repeatedly perform the above step of link reversal, the DAG is guaranteed to be transformed into a DDAG for destination d within finite steps [6]. To illustrate the LR theory, we present an example in Fig. 4(a).

Assume a packet p with destination d enters the network. Intuitively, we can first define the link directions in the original network topology to make it a DAG. Then we can directly apply the LR theory to transform it into a DDAG for destination d, after which we can easily route packet p from any node to destination d by simply following the link directions. However, there are three challenges to perform such operations. First, the application of LR theory calls for a global network view to select the sink node at each step. It requires either a centralized controller or highly-complex distributed protocols to perform the link reversal steps [12]. Second, the packet p cannot be routed until such process is



(a) Example of basic link reversal theory. Transform any given DAG to a DDAG by making link reversal steps at sink nodes.



(b) Example of S-LRR algorithm. Make link reversal step when packet arrives at node C. The routing path is $A \to C \to B \to D$.

Figure 4: Examples of LR theory and S-LRR.

finished, which leads to unacceptable waiting delay for routing in real networks. Third, the LR theory only provides the routing reliability and has no ability to optimize the generated routing paths [12]. In the following, we propose the S-LRR (Sequential Link-Reversal Routing) algorithm combined with neural networks to address all the above challenges.

S-LRR Algorithm. S-LLR defines a *value vector* that contains two types of values, termed the *prime values* and *secondary values* to guide the routing of packets at each node. The prime values are used to define the link directions and provide the routing reliability following the LR theory, while the secondary values are used to provide the full flexibility to control the generation of optimal routing paths.

Consider the case that a packet p arrives at any node v in topology $G(V, E)$. The value vector is computed by a function $\bar{\mathcal{F}}(\cdot)$, which is implemented as a multi-layer neural network, i.e., the FNN in the NGF module. From the view of packet p, each node v is assigned a prime value $\mathcal{V}_p(v)$, which can be computed as

$$\mathcal{V}_p(v) = \bar{\mathcal{F}}(h(v), nid(v), nid(v), p.id) \tag{1}$$

where $p.id$ is the packet ID of p queried for routing [3]. We denote the link direction of $e_{ij} \in E$ as node v_i pointing to node v_j if $\mathcal{V}_p(v_i) > \mathcal{V}_p(v_j)$. Specially, if $\mathcal{V}_p(v_i) = \mathcal{V}_p(v_j)$, the link direction of e_{ij} is denoted as pointing from v_i to v_j if $i > j$. Moreover, each neighboring node $\hat{v} \in \mathcal{N}_v$ of node v is assigned a secondary value $\mathcal{C}_p^v(\hat{v})$ as

$$\mathcal{C}_p^v(\hat{v}) = \bar{\mathcal{F}}(h(v), nid(v), nid(\hat{v}), p.id) \tag{2}$$

where $\mathcal{N}_v$ is the neighboring node set of v.

[3]Note that $nid(v)$ repeats twice in Equ. (1). This is due to the consistent use of function $\bar{\mathcal{F}}(\cdot)$ in Equ. (2).

Algorithm 1 S-LRR Algorithm

```
Input: A packet $p$ arrives at current node $Cur$ with destination $d$. The neighboring node set $\mathcal{N}_{Cur}$ of node $Cur$. A value vector computed by neural network at node $Cur$ that contains the prime values $\mathcal{V}_p(v)$ where $v \in \mathcal{N}_{Cur} \bigcup Cur$ and secondary values $\mathcal{C}_p^{Cur}(\hat{v})$ where $\hat{v} \in \mathcal{N}_{Cur}$.
Output: Next node $\mathcal{Q}$ to forward packet $p$
1: if $Cur \neq d$ then
2:   Extract the key-value set $\mathcal{S}$ from the packet head
3:   for $v \in \mathcal{N}_{Cur} \bigcup Cur$ do
4:     $\mathcal{V}_p(v) \leftarrow \mathcal{S}(v)$ if $v$ is one key in $\mathcal{S}$
5:   end for
6:   $\Omega \leftarrow \{v \in \mathcal{N}_{Cur} | \mathcal{V}_p(v) < \mathcal{V}_p(Cur)\}$
7:   if $\Omega = \emptyset$ then
8:     Add the below key-value pair to the packet head: $(key = Cur,\ value = \max_{k \in \mathcal{N}_{Cur}}\{\mathcal{V}_p(k)\} + 1)$
9:     $\Omega \leftarrow \mathcal{N}_{Cur}$
10:  end if
11:  $\mathcal{Q} \leftarrow \arg\min_{u \in \Omega}\{\mathcal{C}_p^{Cur}(u)\}$
12:  Forward packet $p$ to node $\mathcal{Q}$
13: else
14:  Remove the key-value set in the packet head if any
15: end if
```

The first feature of S-LRR is that when any packet p arrives at node v, it does not wait for transforming G to a DDAG. Instead, it is forwarded immediately following the link directions and performs the link reversal step only when it arrives at any sink node.

The second feature of S-LRR is that the step of link reversal does not require a controller or other protocols. When packet p arrives at a sink node v^*, we simply increase the prime value of node v^* to be the maximum prime value of neighboring nodes plus 1, thus equivalent to reversing all its link directions. In order to synchronize this information to the next hop, we add a specific key-value pair to its packet head where the key is v^* and the value is the newly-increased value. In this way, when p arrives at the next hop, it can know the newest prime value of v^* by simply extracting the packet head.

The third feature of S-LRR is that it selects the final forwarding port based on the secondary values computed by FNN. By training the FNN, we can generate routing paths optimized for different optimization objectives following the network operator's intents.

In the following, we introduce the detailed steps for the S-LRR algorithm to select the forwarding port. As Algorithm 1 shows, when a packet arrives at node Cur, it first checks the packet head to extract the key-value set $\mathcal{S}$ if any (line 1-2). Then the prime values are updated based on $\mathcal{S}$ (line 3-5). Next, S-LRR computes a *feasible* set Ω of next-hop nodes, i.e., the neighboring nodes that have a prime value smaller than that of the current node (line 6). If Ω is empty, S-LRR performs the step of link reversal by adding the key-value pair to the packet head where the *key* is the current node and the *value* is the maximum prime value of neighboring nodes plus 1 (line 7-8), and Ω is set to be $\mathcal{N}_{Cur}$ (line 9). After obtaining set Ω, S-LLR selects the forwarding port that connects to the neighboring node v with the smallest secondary value in Ω (line 11-12). We present a simple example to illustrate the algorithm in Fig. 4(b).

Reliability and Optimality Theorem. A reliable routing path for packet p with source s and destination d is denoted as a path from s to reach d without persistent loops and blackholes. A reliable routing solution $\mathbb{S}$ is a set of reliable routing paths for each packet. Based on the LR theory, we generate the following theorems to illustrate the properties of S-LRR algorithm:

Theorem 1. *(Reliability guarantee) Given any set of neural network weights $\mathcal{W}$ and a connected topology G, Algorithm 1 with the value vector computed by $\mathcal{W}$ can always generate a reliable routing solution $\mathbb{S}$.*

Proof. We first show that given any value vector computed by $\mathcal{W}$, Algorithm 1 will generate a reliable routing path in G for any packet p with destination d. Assume packet p arrives at any node v. If v is not a sink node, Algorithm 1 selects the next-hop node following the link directions. Otherwise, it performs the link reversal step and then selects the next-hop node following the link directions. Based on the LR theory, any sequence of link reversal steps for sink nodes is guaranteed to transform the original DAG into a DDAG [6]. Therefore, in the worst case of DDAG, p is guaranteed to reach d. In the other cases, before it becomes DDAG, p already arrives at d. In both cases, Algorithm 1 generates a reliable routing path for packet p. This completes the proof. □

Theorem 2. *(Capacity to attain optimality) Given any reliable routing solution $\mathbb{S}$ in topology G, there exists a set of neural network weights $\mathcal{W}$ to compute the value vector that make Algorithm 1 generate $\mathbb{S}$ exactly.*

Proof. Consider any packet p with source s and destination d. Let $\mathcal{P}$ denote the specific path defined in $\mathbb{S}$ for routing p, and $|\mathcal{P}|$ denote the path length. We first present a construction of value vectors that can generate the same path $\mathcal{P}$ following Algorithm 1. For any links $e \in \mathcal{P}$, we set the link cost to be 1. For the other links, we set the link cost to be $|\mathcal{P}|$+1. Let $l_c(v_i, v_j)$ denote the link cost between node v_i and v_j. Then we can use a shortest-path algorithm to compute the minimum cost $n_c(v)$ of each node v to reach d. We set the prime values $\mathcal{V}_p(v)$=$n_c(v)$ and the secondary values $\mathcal{C}_p^v(\hat{v}) = l_c(v, \hat{v}) + n_c(\hat{v})$ where $\hat{v} \in \mathcal{N}_v$. Let $h_p(v)$ denote the next-hop node defined in $\mathcal{P}$ for routing p at node v. When p arrives at node v, $h_p(v)$ has a smaller cost to reach d than that of v, i.e., a smaller prime value. Moreover, $h_p(v)$ must be the neighboring node that has the smallest cost from v to reach d, i.e., the smallest secondary value. Therefore, Algorithm 1 will select the same next-hop as $h_p(v)$. Finally, since the value vectors can be explicitly obtained as above based on $\mathbb{S}$, a neural network with enough capacity can be trained in a supervised way to generate the same value vectors [8]. This completes the proof. □

Theorem 3. *(Overhead Theorem) Given any reliable routing solution $\mathbb{S}$ in topology G, the overhead of inserting the key-value pairs in the packet head by Algorithm 1 can only occur for the packets that are forwarding to the wrong port (i.e., different from the forwarding port defined in $\mathbb{S}$) due to the errors of neural networks.*

Proof. Based on the proof of Theorem 2, if the value vectors derived from $\mathbb{S}$ are precisely computed by the neural network, packet p will not encounter any sink node before arriving at the destination, thus no overhead in the packet head occurs. Moreover, for small value errors that do not generate sink nodes, there is no overhead in the packet head. Otherwise, if the computation errors of value vectors are large enough to generate some sink nodes for packet p. Then the link reversal steps are performed on the sink nodes with the packet head overhead. This completes the proof. □

Theorem 1 provides the reliability guarantee of routing despite the errors of neural networks. Theorem 2 ensures the full flexibility of NGR to implement any given reliable routing solution, thus obtaining the power to attain the optimality for any custom objectives. Theorem 3 ensures that Algorithm 1 will generate additional overheads in the packet head only when the neural networks generate forwarding errors in the computation of value vectors. This error can be minimized during the training, thus minimizing the packet head overhead at the same time. For comparison, other alternative solutions that naively use the packet head to record all the visited nodes during routing can also address the routing loops. However, they will introduce high overhead on the packet head, which linearly increases with the number of routing hops.

4 EVALUATION

Simulation Setup. We implement NGR in the Tensorflow platform [2] and run the training of neural networks on a server with two GPUs [4]. The recurrent neural network used in GNN module is implemented as gated recurrent unit (GRU) [4], and the forwarding neural network used in NGF module is implemented as a fully-connected neural network with three hidden layers. We use the real topologies in the Internet topology zoo [1]. To evaluate the optimality performance, we test the algorithms in solving the load balancing task, i.e., minimizing the maximum link utilization.

For comparison, we implement an existing solution named PEFT, i.e., the optimal solution under the triangle constraints that demonstrated in [18] (introduced in Section 2). We also implement the optimal solution named OPT, which is computed by CPLEX python library. Since the solution PEFT has been proved to be optimal (i.e., the same as the solution OPT) if the topology exactly fits the triangle constraint [18], to avoid redundant evaluation results, we

[4] In the experiment, we focus on the supervised learning setting by giving the optimal routing solutions as the labels. Since the GNN and FNN are concatenated and trained together, we only need to set the training labels for the output value vectors of FNN. Specifically, the labels of value vectors are obtained following the link cost setting in proof of Theorem 2. We leave the extension to reinforcement learning setting as a future work.

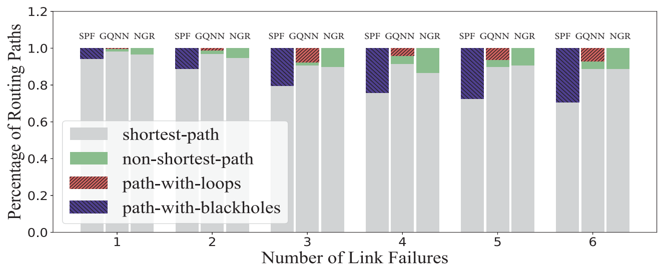


Figure 5: Reliability performance in the task of shortest-path routing.

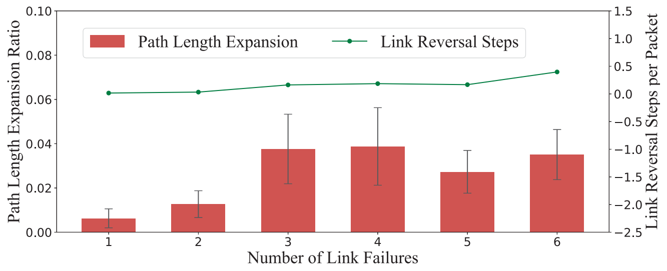


Figure 6: Overhead performance in the task of shortest-path routing.

use the real topology from the Internet topology zoo that does not fit the triangle constraints. This helps illustrate the potential large optimality gap of solution PEFT in practice and how our solution NGR can help close this gap.

Next, in order to analyze the potential reliability issues and overhead of our NGR solution, we test the algorithms in a new task of shortest path routing. We select this task because it is simple, and thus it is much easier to check the reliability and overheads of the algorithms. For comparison, we implement the distributed routing learning model (named GQNN) in [7], which uses a neural network with GNN to directly compute the forwarding port [5] without handling the neural network errors. The conventional shortest-path routing (named SPF) is also evaluated as a baseline. The metric of accuracy in the two tasks is defined as the percentage of pairs of nodes that have the optimal routing paths.

Optimality Performance. In Table. 1, we show the performance of NGR in the task of load balancing over different traffic demand matrixes (termed DM). We randomly generate the traffic demand matrixes DM1-DM4 within the link bandwidth and solve the optimal routing solution by integer programming. We can see that NGR is able to learn the behavior of the optimal solution and achieves near-optimal link utilization under different traffic demands, which has the maximum link utilization about 30% that of PEFT. We find that the performance of PEFT is limited due to the strict restrictions on the selection of next hops, which limits its entire solution space to exclude the global optimal solution. On the other hand, NGR does not give any restrictions on its solution space, and achieve the optimality for DM2 and DM3. We can also see that NGR achieves 100% routing reliability without any loops or blackholes while the learning accuracy keeps above 90%. It demonstrates the power of NGR to learn specific objectives with near-optimal performance.

Reliability Performance. In Fig. 5, we show the reliability results over different link failures. The GQNN and NGR are trained over topologies with random link failures and then test on a set of new different topologies with increasing link failures. The average training accuracy of GQNN and NGR are about 95%. To serve as a baseline, we evaluate the performance of SPF that does not change the shortest routing paths when link failure happens. We can see that the ratio of unreliable paths (i.e., paths with loops or blackholes) of SPF is more than twice that of GQNN and NGR. It validates the generalization ability of GNN applied in GQNN and NGR to handle the topology changes. However, the neural network errors in GQNN lead to a number of routing loops when more link failures happen. NGR does not generate any loops or blackholes in all the cases despite the same training accuracy as GQNN. Instead, NGR transforms the neural network errors into non-shortest routing paths and thus ensures the reliability guarantee of distributed routing.

Overhead Performance. Since NGR relies on the LR theory for routing, it will generate additional overhead on the link reversal steps, which may enlarge the path length when transforming the neural network errors into non-shortest routing paths. In Fig. 6, we show the overhead results of NGR over different link failures. We evaluate two overhead metrics. The first metric is the path length expansion ratio. It is computed as the average value $\frac{l_{NGR}-l_{OPT}}{l_{OPT}}$ for all the packets, where l_{NGR} is the routing path length of NGR and l_{OPT} is the shortest path length after the link failures. The second metric is the required number of link reversal steps per packet. We can see that NGR achieves a small path length expansion ratio below 6% under all the cases, while the number of link reversal steps keeps small and increases slowly with more link failures. This illustrates the low overhead of NGR to handle the neural network errors in packet forwarding.

5 CONCLUSION

In this paper we propose NGR, a deep-learning-based hop-by-hop routing system to achieve both the reliability guarantee and full flexibility to attain the optimality despite the uncertainty of deep learning. Specifically, we carefully combine the deep learning and link reversal theory to address the safety of deep learning application in the network routing problem. Our evaluations validate that NGR can achieve 100% routing reliability and gain performance close to the optimal solutions in the tasks of shortest-path routing and load balancing.

[5]During random link failures, GQNN may select the forwarding port connected to the failed links. To avoid such trivial blackholes, we improve the original GQNN to select another port with the second-highest score if the port with highest score is failed.

Table 1: Load Balancing Task: Maximum Link Utilization of PEFT, OPT and NGR

	PEFT	OPT	NGR	Accuracy of NGR	Loop/Blackholes
DM 1	114%	29.4%	36.4%	92%	0%
DM 2	219%	51%	51%	97%	0%
DM 3	235%	59.3%	59.3%	94%	0%
DM 4	220%	50%	78%	92%	0%

REFERENCES

[1] 2020. *Internet Topology Zoo.* http://www.topology-zoo.org/.

[2] 2020. *Tensorflow.* https://www.tensorflow.org/.

[3] Justin A. Boyan and Michael L. Littman. 1993. Packet routing in dynamically changing networks: a reinforcement learning approach. In *International Conference on Neural Information Processing Systems (NIPS).*

[4] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[5] Samuel P. M. Choi and Dit-Yan Yeung. 1995. Predictive Q-routing: A Memory-based Reinforcement Learning Approach to Adaptive Traffic Control. In *Proceedings of the 8th International Conference on Neural Information Processing Systems (NIPS).*

[6] Eli M. Gafni and Dimitri P. Bertsekas. 1981. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Transactions on Communications* 29 (1981), 11–18.

[7] Fabien Geyer and Georg Carle. 2018. Learning and generating distributed routing protocols using graph-based deep learning. In *ACM SIGCOMM 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks.*

[8] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks* 2, 5 (1989), 359–366.

[9] Praveen Kumar, Yang Yuan, Chris Yu, Nate Foster, Robert Kleinberg, Petr Lapukhov, Chiun Lin Lim, and Robert Soulé. 2018. Semi-Oblivious Traffic Engineering: The Road Not Taken. In *NSDI 2018.*

[10] Shailesh Kumar and Risto Miikkulainen. 1998. Confidence-Based Q-Routing: An On-Line Adaptive Network Routing Algorithm. In *International Joint Conference on Artificial Intelligence.*

[11] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436.

[12] Junda Liu, Aurojit Panda, Ankit Singla, Brighten Godfrey, Michael Schapira, and Scott Shenker. 2013. Ensuring connectivity via data plane mechanisms. In *NSDI 2013.*

[13] Deepankar Medhi and Karthikeyan Ramasamy. 2007. *Network Routing: Algorithms, Protocols, and Architectures.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[14] Nithin Michael and Ao Tang. 2014. HALO: Hop-by-Hop Adaptive Link-State Optimal Routing. *IEEE/ACM Transactions on Networking* 23, 6 (2014), 1862–1875.

[15] Imtiaz Parvez, Ali Rahmati, Ismail Guvenc, Arif I. Sarwat, and Huaiyu Dai. 2018. A Survey on Low Latency Towards 5G: RAN, Core Network and Caching Solutions. *IEEE Communications Surveys & Tutorials* (2018).

[16] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2008), 61–80.

[17] Tushar Swamy, Alexander Rucker, Muhammad Shahbaz, and Kunle Olukotun. 2020. Taurus: An Intelligent Data Plane. *arXiv preprint arXiv:2002.08987* (2020).

[18] Dahai Xu, Mung Chiang, and Jennifer Rexford. 2011. Link-State Routing With Hop-by-Hop Forwarding Can Achieve Optimal Traffic Engineering. *IEEE/ACM Transactions on Networking* 19, 6 (2011), 1717–1730.

[19] Zhiyuan Xu, Jian Tang, Jingsong Meng, Weiyi Zhang, Yanzhi Wang, Chi Harold Liu, and Dejun Yang. 2018. Experience-driven Networking: A Deep Reinforcement Learning based Approach. In *INFOCOM 2018.*

[20] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. 2017. Understanding deep learning requires rethinking generalization. In *ICLR 2017.*