Learning and Generating Distributed Routing Protocols Using Graph-Based Deep Learning

Fabien Geyer
Technical University of Munich
Garching b. München, Germany
fgeyer@net.in.tum.de

Georg Carle
Technical University of Munich
Garching b. München, Germany
carle@net.in.tum.de

ABSTRACT

Automated network control and management has been a long standing target of network protocols. We address in this paper the question of automated protocol design, where distributed networked nodes have to cooperate to achieve a common goal without a priori knowledge on which information to exchange or the network topology. While reinforcement learning has often been proposed for this task, we propose here to apply recent methods from semi-supervised deep neural networks which are focused on graphs. Our main contribution is an approach for applying graph-based deep learning on distributed routing protocols via a novel neural network architecture named Graph-Query Neural Network. We apply our approach to the tasks of shortest path and max-min routing. We evaluate the learned protocols in cold-start and also in case of topology changes. Numerical results show that our approach is able to automatically develop efficient routing protocols for those two use-cases with accuracies larger than 95 %. We also show that specific properties of network protocols, such as resilience to packet loss, can be explicitly included in the learned protocol.

CCS CONCEPTS

• **Networks → Network protocol design**; • **Computing methodologies → Distributed artificial intelligence**; **Neural networks**;

KEYWORDS

Routing, Graph Neural Network, Deep Learning

ACM Reference Format:
Fabien Geyer and Georg Carle. 2018. Learning and Generating Distributed Routing Protocols Using Graph-Based Deep Learning. In *Big-DAMA'18: ACM SIGCOMM 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks , August 20, 2018, Budapest, Hungary*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3229607.3229610

1 INTRODUCTION

The current improvements in computational power in packet processing devices and the ability to instrument always more measurements about network performance and behavior have resulted in the emergence of a new paradigm in networking, namely *data-driven networks* and *data-driven protocols* [6, 12]. This new paradigm aims at bringing lessons learned from measurements and data in protocol behavior and design. A concrete application is the concept of *self-driving network* proposed by Feamster and Rexford [6], where network control is tightly coupled with measurements and performance objectives. In this context, machine learning has often been proposed and applied to various tasks such as routing [3, 25, 28], computing resource management [16] or packet scheduling [4].

We propose in this paper to investigate the question of automatic network protocol design using recent methods from semi-supervised deep learning. Our contribution is a novel approach for training a network of independent agents such that they cooperatively exchange information and solve a common goal in a fully distributed manner without central control. We address more specifically the question of distributed routing protocols. From a network protocol perspective, the routing agents should autonomously develop a network protocol akin to RIP or OSPF, *i.e.* exchange topology information and perform local path computations based on the exchanged information. Traditional properties from routing protocols are also considered, namely handling topology changes and packet losses.

Our approach is based on a novel extension of Graph Neural Network (GNN) [10, 21], which we name here *Graph-Query Neural Network* (GQNN). GNNs are neural network architectures able to process graphs as input using the concept of message passing between nodes in the graph. We evaluate our approach on various topologies from real networks [13] and show that our approach leads to the creation of communication protocols able to exchange data about topology information as well as topology changes. Using two different path calculation strategies – namely shortest path routing and max-min fair routing – we show that various routing objectives can be achieved using the same neural network architecture. The results of our approach are also compared against bounds on information propagation in topologies and we show that the routing protocols are efficient in term of number of iterations necessary to reach convergence. We also demonstrate that resilience to packet loss can be explicitly trained for.

This work is structured as follows. We describe in Sections 2 and 3 our modeling approach and the neural network architecture, with an introduction on Graph Neural Networks and Graph Gated Neural Networks, followed by the application of those concepts to network topologies and distributed routing protocols. We numerically evaluate our approach in Section 4 with the evaluation on real network topologies. In Section 5, we present similar research studies. Finally, Section 6 concludes our work.

Big-DAMA'18, August 20, 2018, Budapest, Hungary

ACM ISBN 978-1-4503-5904-7/18/08…$15.00
https://doi.org/10.1145/3229607.3229610

2 NEURAL NETWORKS FOR GRAPHS

The main intuition behind our approach is to map network topologies to graphs, with nodes representing routers and additional nodes for each physical interface of the router. Those graph representations are then used as input for a neural network architecture able to process general graphs. The transformation from a network topology to its graph representation is presented in Section 3.

In this section, we detail the neural network architecture used for learning on graphs, namely the family of architectures based on Graph Neural Networks [10, 21], and introduce a new extension of this architecture.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph with nodes $v \in \mathcal{V}$ and edges $e \in \mathcal{E}$. Let $\mathbf{i}_v$ and $\mathbf{o}_v$ represent respectively the input features and target values of node v. The concept behind Graph Neural Networks is called *message passing*, where hidden representations of nodes are based on the hidden representations of their neighboring nodes. Those hidden representations are propagated through the graph using multiple iterations until a fixed point is found. The final hidden representation is then used for predicting properties about nodes. This concept can be expressed as:

$$\mathbf{h}_v^{(t)} = f\left(\left\{\mathbf{h}_u^{(t-1)} \;\middle|\; u \in \textsc{Nbr}(v)\right\}\right) \tag{1}$$

$$\mathbf{o}_v = g\left(\mathbf{h}_v^{(t\to\infty)}\right) \tag{2}$$

$$\mathbf{h}_v^{(t=0)} = \mathit{init}\left(\mathbf{i}_v\right) \tag{3}$$

with $\mathbf{h}_v^{(t)}$ representing the hidden representation of node v at time t, $f(\cdot)$ a function which aggregates the different hidden representations, $\textsc{Nbr}(v)$ the set of neighboring nodes of v, $g(\cdot)$ a function for transforming the final hidden representation to the target values, and $\mathit{init}(\cdot)$ a function for initializing the hidden representations based on the input features.

The concrete implementations of the $f(\cdot)$ and $g(\cdot)$ functions are feed-forward neural networks, with the special case that $f(\cdot)$ in Equation (1) is the sum of per-edge terms (as recommended by [21]) such that:

$$\mathbf{h}_v^{(t)} = f\left(\left\{\mathbf{h}_{\textsc{Nbr}(v)}^{(t-1)}\right\}\right) = \sum_{u\in\textsc{Nbr}(v)} f^*\left(\mathbf{h}_u^{(t-1)}\right) \tag{4}$$

with $f^*(\cdot)$ a feed-forward neural network. For $\mathit{init}(\cdot)$, the initial node features are used and zero-padded to fit the dimensions of the hidden representations.

In [10, 21], training the neural network architecture, namely the parameters of $f(\cdot)$, $g(\cdot)$ and $h(\cdot)$, is done via the Almeida-Pineda algorithm [2, 20] which works by running the propagation of the hidden representation to convergence, and then computing gradients based upon the converged solution.

2.1 Extensions of Graph Neural Networks

Various extensions of GNNs have been proposed in the literature in recent years in order to improve accuracy and applicability. Those extensions build on the principle of message passing with more recent development from deep learning. We give here an overview over the extensions which were used for the final architecture used in this paper. For easier notation, we define $\mathbf{H}^{(t)}$ as the vector of all hidden representations at iteration t: $\left[\mathbf{h}_1^{(t)} \cdots \mathbf{h}_{|\mathcal{V}|}^{(t)}\right]$.

2.1.1 Gated Graph Neural Network. In order to improve the training of Graph Neural Networks, Li et al. proposed Gated Graph Neural Networks (GG-NNs) in [14]. This extension implements the function $f(\cdot)$ using a memory unit called Gated Recurrent Unit (GRU) [5] and unrolls Equation (1) for a fixed number of iterations.

Formally, the propagation of the hidden representations among neighboring nodes for one time-step is formulated as:

$$\mathbf{x}^{(t)} = \mathbf{H}^{(t-1)}\mathbf{A} + \mathbf{b}_a \tag{5}$$

$$\mathbf{z}^{(t)} = \sigma\left(\mathbf{W}_z x^{(t)} + \mathbf{U}_z \mathbf{H}^{(t-1)} + \mathbf{b}_z\right) \tag{6}$$

$$\mathbf{r}^{(t)} = \sigma\left(\mathbf{W}_r x^{(t)} + \mathbf{U}_r \mathbf{H}^{(t-1)} + \mathbf{b}_r\right) \tag{7}$$

$$\widetilde{\mathbf{H}}^{(t)} = \tanh\left(\mathbf{W} x^{(t)} + \mathbf{U}\left(\mathbf{r}^{(t)} \odot \mathbf{H}^{(t-1)}\right) + \mathbf{b}\right) \tag{8}$$

$$\mathbf{H}^{(t)} = \left(1 - \mathbf{z}^{(t)}\right) \odot \mathbf{H}^{(t-1)} + \mathbf{z}_v^{(t)} \odot \widetilde{\mathbf{H}}^{(t)} \tag{9}$$

where $\sigma(x) = 1/(1+e^{-x})$ is the logistic sigmoid function and $\odot$ is the element-wise matrix multiplication. $\{\mathbf{W}_z, \mathbf{W}_r, \mathbf{W}\}$ and $\{\mathbf{U}_z, \mathbf{U}_r, \mathbf{U}\}$ are trainable weight matrices, and $\{\mathbf{b}_a, \mathbf{b}_r, \mathbf{b}_z, \mathbf{b}\}$ are trainable bias vectors. $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ is the graph adjacency matrix, determining how nodes in the graph $\mathcal{G}$ communicate with each other.

2.1.2 Edge attention. A recent advance in neural networks has been the concept of *attention*, which provides the ability to a neural network to focus on a subset of its inputs. This mechanism has been used in a variety of applications such as computer vision or natural language processing (eg. [26]). For the scope of GNNs, we introduce here so-called *edge attention*, namely we wish to give the ability to each node to focus only on a subset of its neighborhood. Formally, let $a_{(v,u)}^{(t)} \in [0, 1]$ be the attention between node v and u. Equation (4) is then extended as:

$$\mathbf{h}_v^{(t)} = \sum_{u\in\textsc{Nbr}(v)} a_{(v,u)}^{(t)} \cdot f^*\left(\mathbf{h}_u^{(t-1)}\right) \tag{10}$$

$$\mathbf{a}_{(v,u)}^{(t)} = f_A\left(\mathbf{h}_v^{(t-1)}, \mathbf{h}_u^{(t-1)}\right) \tag{11}$$

To make the computation of $a_{(v,u)}^{(t)}$ for all edges more efficient, we use the modified adjacency matrix $\widetilde{A}^{(t)}$ defined as:

$$\widetilde{A}^{(t)} = A \odot \sigma\left(f_{A_1}\left(\mathbf{H}^{(t)}\right)^T \odot f_{A_2}\left(\mathbf{H}^{(t)}\right)\right) \tag{12}$$

with $f_{A_1}(\cdot)$ and $f_{A_2}(\cdot)$ feed-forward neural networks. Similar concepts of edge attention have already been proposed in the literature with various implementations (eg. [11, 27]).

3 APPLICATION TO ROUTING

We describe in this section the application of the graph-based deep learning architectures presented in Section 2 to the task of distributed routing protocols. We are interested in training neural networks on two important aspects of those network protocols. The first one is the network protocol itself, namely how to distribute topology information among different nodes, and the second one is how to compute routes given a topology and link weights.

3.1 Graph representation

For the first aspect, we wish to train routing agents such that they autonomously exchange data about a given topology without explicitly specifying which information about the topology to transmit. Based on this exchanged information, the routing agent can populate routing tables. In comparison to traditional distributed routing protocols, we essentially wish to train neural networks to transmit information akin to link-state updates (as used in OSPF for example) or router distances (as used in RIP for example). As for standard routing protocols, the learned protocol should also deal with changes in the topology (i.e. link failure, node addition).

The main intuition behind the input feature modeling is to use the topology as input graph $\mathcal{G}$, with additional nodes representing the network interfaces as illustrated in Figures 1a and 1b. In order to enforce communication between nodes according to the physical network topology, no additional edge is added to the graph. As for traditional routing protocols, each router in the topology is assigned an integer identifier, noted R_i. Nodes representing routers in the graph use this identifier encoded as a one-hot vector for their initial representation $\mathbf{i}_v$. Nodes representing interfaces use a weight parameter (eg. based on the link bandwidth) for $\mathbf{i}_v$.

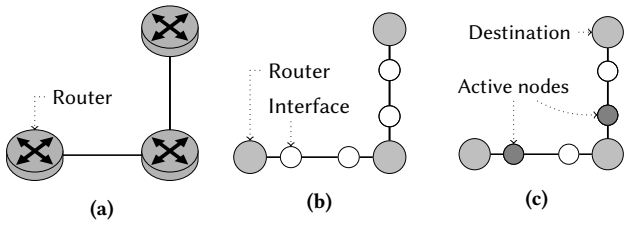

Figure 1: (a) Example network topology. (b) Its associated graph used for training. (c) The output feature of the interfaces according to a selected destination.

3.2 Graph Query Neural Networks

We are interested in this section in the local computation of the routing table based on the topology information which was distributed by the different nodes in the graph. Given a destination router identifier R_d, each router must locally decide which output interface should be used. Based on the graph representation from the previous section and a given algorithm for path calculation, this is modeled by labeling the interfaces with $\mathbf{o}_i = [1]$ if they are used for transmitting packets to router R_d, and $[0]$ otherwise, as illustrated in Figure 1c.

In order to build a routing protocol with local path computation, we introduce here a new extension to GNNs, called *Graph Query Neural Network* (GQNN). The neural network architecture is illustrated in Figure 2. The hidden node representations $\mathbf{h}_v^{(t)}$ correspond to the messages which are transfered between nodes. Once the message passing is finished, each node in the graph has a local representation of the network topology $\mathbf{h}_v^{(T)}$. For determining which interface to use for a given destination router R_d, each router applies the following procedure on each of its interface nodes:

$$\mathbf{q}_d = Q(R_d) \qquad \text{query vector computation} \tag{13}$$

$$\mathbf{o}_v = g\left(\mathbf{q}_d \odot \mathbf{h}_v^{(T)}\right) \qquad \text{output label as in Equation (2)} \tag{14}$$

with $Q(\cdot)$ and $g(\cdot)$ feed-forward neural networks.

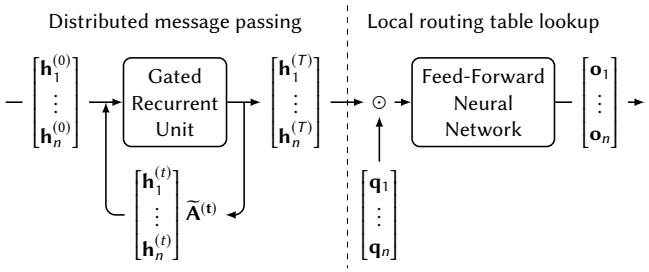

Figure 2: Overview of the Graph Query Neural Network architecture used in this paper

3.3 Learned routing strategies

For the scope of this paper, we evaluate two algorithms for route calculation: *shortest path* and *max-min routing*.

In the case of shortest path routing, the neural network is trained against path calculations based on Dijkstra's algorithm, where each link is associated with a weight. In case multiple shortest paths are available, we need to discriminate between them in order to have stable routing strategies for easier training of the neural network. This is performed by using the router identifiers as discriminant between paths.

In the case of max-min fair routing [18], we aim at maximizing the minimum allocated bandwidth between all possible source-destination pairs in the network. Such routing strategy should lead to network topologies with less link overload than shortest path routing. For our evaluations, we assign an equal demand for all possible source-destination pairs. The route computation is done using linear programing. As for shortest path routing, we also give priority to paths which minimize the identifiers of the traversed routers. This is performed by defining multiple objectives and solving them in a hierarchical way.

3.4 Packet losses and topology changes

An important requirement of routing protocols is the ability to cope with packet losses and dynamic topology changes. In the case of packet losses, various strategies have been used in existing routing protocols: either leverage transport protocols (*e.g.* BGP over TCP), or design an own transport layer (*e.g.* OSPF). For this paper, we are interested in designing network protocols which do not leverage other transport protocol functionalities.

In order to train the neural network to handle packets loss, the adjacency matrix $\mathbf{A}$ is randomly modified during training such that some edges are temporarily disabled according to a Bernoulli distribution with parameter p. We implemented this by using a dropout layer [23] in the neural network architecture without the traditional normalization factor used in standard dropout:

$$\hat{\mathbf{A}}^{(t)} = \mathbf{r}(t) \odot \mathbf{A} \quad \text{with } \mathbf{r}(t) \sim \textit{Bernoulli}(p),\ \forall t \in [0, T] \tag{15}$$

Since routing protocols are designed to run continuously and handle topology changes, we define here two phases of the protocol: *cold-start* when the routing protocol is first initialized on the active routers, and *warm-start* when a node fails or a new node joins a network where the routing protocol already ran for some iterations. More formally, we define graphs pairs $\{\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1), \mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)\}$ such that some routers are added or removed between $\mathcal{G}_1$ and $\mathcal{G}_2$. The neural network is first trained on $\mathcal{G}_1$, and the final hidden representations from this first phase are then reused as initial hidden representations for the second training phase on $\mathcal{G}_2$ for the nodes which did not change between $\mathcal{G}_1$ and $\mathcal{G}_2$:

$$\forall v \in \{\mathcal{V}_1 \cap \mathcal{V}_2\}, \quad \mathbf{h}_{v,\mathcal{G}_1}^{(t=T)} = \mathbf{h}_{v,\mathcal{G}_2}^{(t=0)} \tag{16}$$

3.5 Implementation in routers

Regarding implementation of the resulting network protocol in real routers, each router has an internal subgraph, with one central node representing the router connected to multiple nodes representing its interfaces. Each router then periodically broadcasts its hidden interface representations to its neighboring routers, and recomputes its routing table based on the received messages.

4 NUMERICAL EVALUATION

We evaluate in this section the approach presented in Sections 2 and 3 on real network topologies from the *Internet Topology Zoo* [13], which is a collection of topologies from Internet providers around the world. We selected topologies such that the number of nodes is limited to 20 and the maximum hop count between any two nodes in the network is less than 10.

In order to generate more topologies for training our neural network, each topology is modified by either randomly adding a router and connecting it randomly to other routers, or randomly deleting one router in the topology. Router identifiers are randomly assigned to the routers as described in Section 3. Random router failure or addition are generated as described in Section 3. This resulted in a dataset with 40 000 graphs in total.

The two routing algorithms presented in Section 3.3 are then applied on each generated topology to build the datasets used for this evaluation.

4.1 Implementation

The GG-NN architecture presented in Sections 2 and 3.2 was implemented using Tensorflow [1] and trained using Nvidia GPUs. Additional dropout layers [23] were added according to standard practices for neural network and recurrent neural networks [22] in order to avoid over-fitting. Hidden node representations were chosen with a size of 160. The same parameters were used for training the neural network for both routing use-cases.

4.2 Accuracy

We evaluate in Figure 3 the accuracy of the computed routes according to the two use-cases and routing phases. For a given topology, we define the accuracy as 1 if the route for a given destination is correct for all routers in the topology, and 0 otherwise. The learned protocol is better able to predict shortest path routing, where a perfect accuracy is reached for than 50 % of the evaluated topologies.

In average, accuracies of 98 %, respectively 95 %, could be reached for shortest path routing, respectively min-max routing.

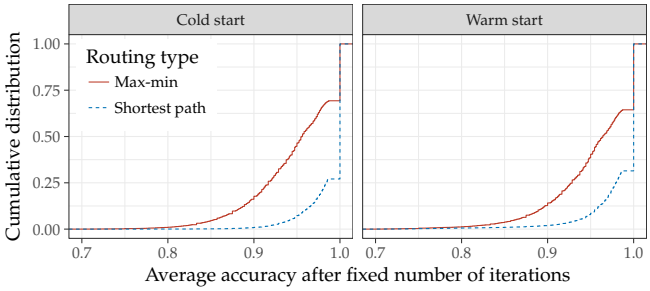

Figure 3: Overview over the accuracy of the predicted routes

4.3 Convergence time

In order to assess the convergence time of the developed protocols, we first evaluate the accuracy of the routing at different iterations of the fixed point evaluation presented in Equation (1) in cold-start and warm-start phases. The numerical results are presented in Figure 4. In case of topology changes (ie. warm start), better accuracies are reached faster as routes only need partial reconfiguration. This shows that the protocol is indeed able to efficiently cope with and react to topology changes.

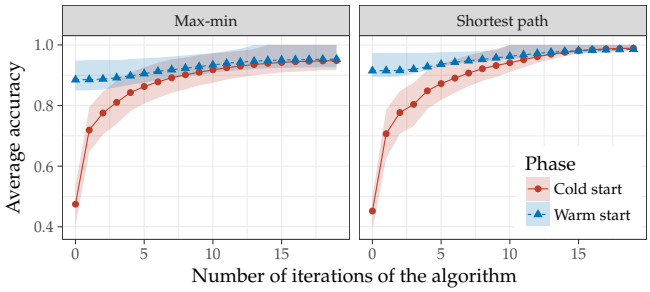

Figure 4: Accuracy according to the iterations of the protocols. Areas indicate the 25 and 75 percentile.

We then evaluate the developed protocols against minimum bounds of the number of iterations required to achieve convergence in the computed routes. According to the message passing principles described in Section 2, each node broadcasts learned information at each iteration of the routing protocol to its neighbors. In order to achieve a correct computation of the routes, each node needs to have received at least some piece of information from all the other nodes in the topology. Hence, the minimum number of iterations corresponds to the diameter d of the graph of the network topology. We call this minimum bound the *one-way bound*. In case any pair of nodes in the topology needs to have exchanged information in both directions, the minimum number of iterations needed is $2d$, called here *both-way bound*.

We define T_C as the iteration when convergence occurs, i.e. the time when the computed routes for each node in the topology are stable. Figure 5 compares T_C against the two bounds previously

defined, namely for each graph we compute $T_C - d$ and $T_C - 2d$. Negative values indicate that the learned protocol converged faster than the theoretical bound.

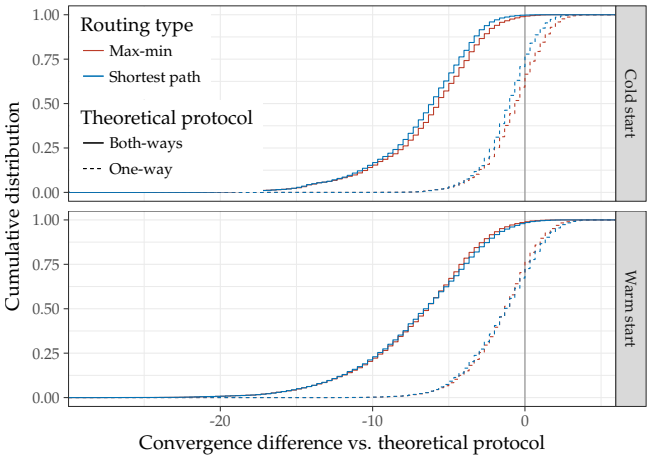

Figure 5: Evaluation of cold-start and warm-start convergence time of the learned protocols against theoretical bounds. Negative values indicate fast convergence.

4.4 Resilience to packet loss

We described in Section 3.4 that a crucial property of routing protocol is resilience to packets loss. Figure 6 illustrates the accuracy of the protocols in case of different packet loss probabilities in the network. We also compare in Figure 6 two different variants for training the neural network, namely explicit or unspecific training for handling packet loss. Both variants of the protocol are run for the same number of iterations. Explicit training for packet loss was done by using a dropout layer as described in Equation (15).

We notice a clear difference between the two training variants. By explicitly training for packet loss, the learned protocol is able to reach better accuracy in case of packet loss.

4.5 Visualization

In order to better understand the working of the generated routing protocol, we propose in this section to visualize information propagation in a topology. Figure 7 illustrates the accuracy of a given route on a small topology at different iterations of the protocol. Such visualization can be used to determine protocol convergence for the different interfaces in the network.

5 RELATED WORK

The question of distributed routing protocols based on machine learning has already attracted various researchers. Early work on this topic include *Q-Routing* from Boyan and Littman [3], *COllective INtelligence* (COIN) from Wolpert et al. [28], or *distributed Gradient Ascent Policy Search* (GAPS) from Peshkin and Savova [19]. Their general approach is to use multi-agent reinforcement learning in combination with a network-wide utility function. More recently, Valadarsky et al. [25] also proposed to use reinforcement learning, with the goal of using past traffic matrices in order to guide route calculations. Compared to those works, we use here semi-supervised

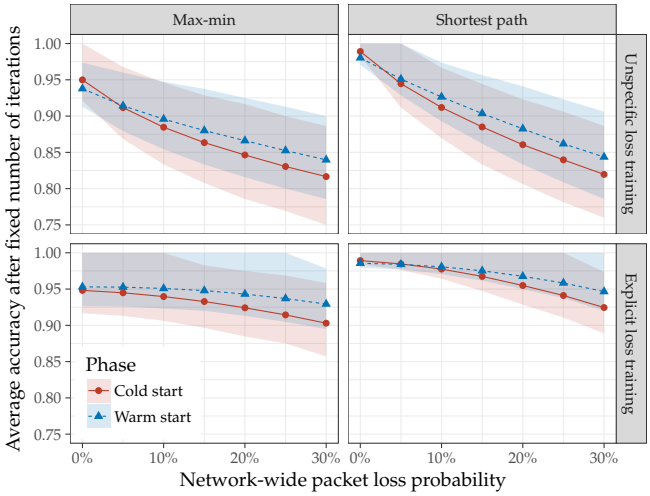

Figure 6: Accuracy of the protocols in case of packet loss in the network with explicit or unspecific training for packet loss. Areas indicate the 25 and 75 percentile.

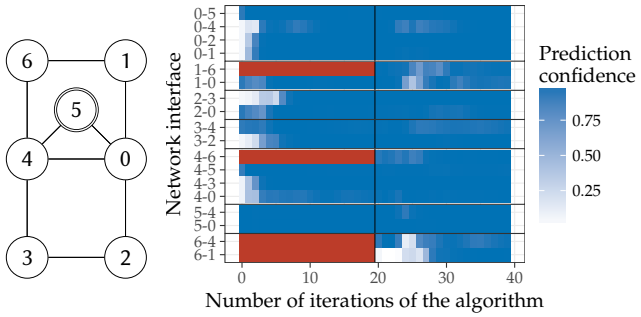

Figure 7: Visualization of the protocol evolution on a small network topology. Each network interface is queried for node 5. Node 6 is first offline and started at iteration 20.

learning in order to more easily specify the routing policy which is expected. Previous work also often predetermined or constrained the specification and format of the communication, whereas our approach leaves the content or format of the exchanged information as a parameter to be learned. Our work also evaluates key aspects of routing protocols, namely resilience against packet loss and inclusion of network dynamics.

A supervised learning approach was recently proposed by Mao et al. [15] using Supervised Deep Belief Architectures, with a focus on speed of route computation. Compared to their approach, our method can be applied to a wider range of network topologies since it is independent of the underlying structure of the topology.

The challenge of training agents to communicate and realize a common goal has attracted work in other domains. Foerster et al. [7] applied *Deep Distributed Recurrent Q-Networks* (DDRQN) for solving logic riddles. Sukhbaatar et al. [24] proposed a deep neural network architecture called *CommNet* for developing communication between agents on the task of multi-turn games, traffic junction or

logic riddles. In both approaches, no constraint on communication structure is enforced as a broadcast channel is used.

Neural networks for graphs have recently attracted a larger interest, and are generally based on the concept of message passing presented in Section 2. In the context of communication networks, they have successfully been applied to performance evaluation of TCP flows in [8]. The model presented here is based on [8] with novel extensions for edge attention, query as presented in Section 3.2, and support training for topology changes. They have also been used in a variety of other domains such as basic logical reasoning tasks and program verification [14], semantic role labeling in natural language processing [17], prediction of chemical properties of molecules [9]. To the best of our knowledge, this is the first work applying GNNs to distributed routing protocols.

6 CONCLUSION

We contributed in this paper a novel approach for automatic network protocol design using graph-based deep learning. Our method is based on an extension of Graph Neural Networks called Graph Query Neural Network and a mapping from network topologies to graphs with special nodes representing network interfaces.

We applied our approach to distributed routing protocols, where routing nodes need to exchange information about the network topology in order to reach efficient routes without specifying which information to exchange. Shortest path and max-min routing were evaluated as routing strategies. In our numerical evaluation, we showed that our approach is able to reach good accuracies. We illustrated that specific properties of network protocols such as resilience to packet loss can be explicitly included in the learned protocols by training the neural network with appropriate dropout.

As our approach is not specific to routing protocols, future work may include evaluations and extensions of our approach to other network protocols and applications.

Acknowledgments This work was supported by the German Federal Ministry of Education and Research (grant 16KIS0538, project DecADe), by the German-French Academy for the Industry of the Future, and the High-Performance Center for Secure Networked Systems.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). http://tensorflow.org/ Software available from tensorflow.org.

[2] Luis B. Almeida. 1990. Artificial Neural Networks. IEEE Press, Piscataway, NJ, USA, Chapter A Learning Rule for Asynchronous Perceptrons with Feedback in a Combinatorial Environment, 102–111.

[3] Justin A. Boyan and Michael L. Littman. 1994. Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach. In *Advances in Neural Information Processing Systems 6*, J. D. Cowan, G. Tesauro, and J. Alspector (Eds.). Morgan-Kaufmann, 671–678.

[4] Timothy X. Brown. 2002. Switch Packet Arbitration via Queue-Learning. In *Advances in Neural Information Processing Systems 14*, T. G. Dietterich, S. Becker, and Z. Ghahramani (Eds.). MIT Press, 1337–1344.

[5] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. (June 2014). arXiv:1406.1078

[6] Nick Feamster and Jennifer Rexford. 2017. Why (and How) Networks Should Run Themselves. (Oct. 2017). arXiv:cs.NI/1710.11583v1

[7] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. 2016. Learning to Communicate to Solve Riddles with Deep Distributed Recurrent Q-Networks. (Feb. 2016). arXiv:cs.AI/1602.02672v1

[8] Fabien Geyer. 2017. Performance Evaluation of Network Topologies using Graph-Based Deep Learning. In *Proceedings of the 11th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2017)*. https://doi.org/10.1145/3150928.3150941

[9] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Doina Precup and Yee Whye Teh (Eds.), Vol. 70. PMLR, International Convention Centre, Sydney, Australia, 1263–1272.

[10] Marco Gori, Gabriele Monfardini, and Franco Scarselli. 2005. A New Model for Learning in Graph Domains. In *Proceedings of the 2005 IEEE International Joint Conference on Neural Networks (IJCNN'05)*, Vol. 2. IEEE, 729–734. https://doi.org/10.1109/IJCNN.2005.1555942

[11] Yedid Hoshen. 2017. VAIN: Attentional Multi-agent Predictive Modeling. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 2698–2708.

[12] Junchen Jiang, Vyas Sekar, Ion Stoica, and Hui Zhang. 2017. Unleashing the Potential of Data-Driven Networking. In *Proceedings of 9th International Conference on COMmunication Systems & NETworkS (COMSNET)*.

[13] Simon Knight, Hung X. Nguyen, Nick Falkner, Rhys Bowden, and Matthew Roughan. 2011. The Internet Topology Zoo. *IEEE J. Sel. Areas Commun.* 29, 9 (Oct. 2011), 1765–1775. https://doi.org/10.1109/JSAC.2011.111002

[14] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2016. Gated Graph Sequence Neural Networks. In *Proceedings of the 4th International Conference on Learning Representations (ICLR'2016)*.

[15] Bomin Mao, Zubair Md. Fadlullah, Fengxiao Tang, Nei Kato, Osamu Akashi, Takeru Inoue, and Kimihiro Mizutani. 2017. Routing or Computing? The Paradigm Shift Towards Intelligent Computer Network Packet Transmission Based on Deep Learning. *IEEE Trans. Comput.* 66, 11 (Nov. 2017), 1946–1960. https://doi.org/10.1109/TC.2017.2709742

[16] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource Management with Deep Reinforcement Learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets '16)*. 50–56. https://doi.org/10.1145/3005745.3005750

[17] Diego Marcheggiani and Ivan Titov. 2017. Encoding Sentences with Graph Convolutional Networks for Semantic Role Labeling. (March 2017). arXiv:cs.CL/1703.04826v4

[18] Dritan Nace and Michał Pióro. 2008. Max-Min Fairness and Its Applications to Routing and Load-Balancing in Communication Networks: A Tutorial. *IEEE Commun. Surveys Tuts.* 10, 4 (2008), 5–17. https://doi.org/10.1109/SURV.2008.080403

[19] Leonid Peshkin and Virginia Savova. 2002. Reinforcement Learning for Adaptive Routing. In *Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1825–1830. https://doi.org/10.1109/IJCNN.2002.1007796

[20] Fernando J. Pineda. 1987. Generalization of back-propagation to recurrent neural networks. *Phys. Rev. Lett.* 59 (Nov. 1987), 2229–2232. Issue 19. https://doi.org/10.1103/PhysRevLett.59.2229

[21] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The Graph Neural Network Model. *IEEE Trans. Neural Netw.* 20, 1 (Jan. 2009), 61–80. https://doi.org/10.1109/TNN.2008.2005605

[22] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. 2016. Recurrent Dropout without Memory Loss. (March 2016). arXiv:1603.05118

[23] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15, 1 (Jan. 2014), 1929–1958.

[24] Sainbayar Sukhbaatar, Rob Fergus, et al. 2016. Learning Multiagent Communication with Backpropagation. In *Advances in Neural Information Processing Systems*. 2244–2252.

[25] Asaf Valadarsky, Michael Schapira, Dafna Shahaf, and Aviv Tamar. 2017. Learning to Route. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks (HotNets-XVI)*. ACM, New York, NY, USA, 185–191. https://doi.org/10.1145/3152434.3152441

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 6000–6010.

[27] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2017. Graph Attention Networks. (Oct. 2017). arXiv:stat.ML/1710.10903v1

[28] David Wolpert, Kagan Tumer, and Jeremy Frank. 1999. Using Collective Intelligence to Route Internet Traffic. In *Advances in Neural Information Processing Systems 11*, M. J. Kearns, S. A. Solla, and D. A. Cohn (Eds.). MIT Press, 952–960.