Steering Traffic via Recurrent Neural Networks in Challenged Edge Scenarios

Alessandro Gaballo*† Matteo Flocco* Flavio Esposito* Guido Marchetto†
*Saint Louis University, USA †Politecnico di Torino, Italy

***Abstract*—With edge computing, it is possible to offload computationally intensive tasks to closer and more powerful servers, passing through an edge network. This practice aims to reduce both response time and energy consumption of data-intensive applications, crucial constraints in mobile and IoT devices. In challenged networked scenarios, such as those deployed by first responders after a natural or human-made disaster, it is particularly challenging to achieve high levels of throughput due to scarce network conditions.**

In this paper, we present an algorithm for traffic management that takes advantage of a deep learning model to implement the forwarding mechanism during task offloading in these challenging scenarios. In particular, our work explores if and when it is worth using deep learning on a switch to route traffic generated by microservices and offloading requests. Our approach differs from classical ones in the design: we do not train centralized routing decisions. Instead, we let each router learn how to adapt to a lossy path without coordination, by merely using signals from standard performance-unaware protocols such as OSPF. Our results, obtained with a prototype and with simulations are encouraging, and uncover a few surprising results.

I. Introduction

In recent years, an increasing number of IoT and mobile devices became available, producing a massive amount of data, and hence exacerbating network orchestration challenges, and creating research and business opportunities. The majority of these IoT devices do not have or cannot handle the computational requirements to process the data they capture. For this reason, solutions that require outsourcing the responsibility to perform all, or a part, of the computations to the edge cloud are gaining popularity [1]–[5]. The process of transferring or delegating computational tasks is called offloading [2] or onloading [6]. Offloading or onloading operations are crucial for mobile devices as they commence to lower response time, lower processing time, and smaller device energy consumption.

Computation offloading is strictly necessary for critical scenarios, such as natural or human-made disasters [7], where the physical network infrastructure is scarce or likely to be temporarily unavailable. As latency requirements become stringent, network alternatives are scarce, and data needs fast delivery. In this or similar scenarios, responsive path management solutions to direct offloading requests, $e.g.$, mobile-generated traffic steering, may become an essential application requirement.

Traffic engineering solutions used in production today ($e.g.$, OSPF, ECMP), are performance-unaware, that is, they react only when losses or delay impact the cost assigned to a path; we argue that those are hence unsuitable for unstable or unreliable networks; moreover, in the presence of dynamic traffic and network conditions, these solutions are known to lead to sub-optimal performance [3], [5], [8]–[11]. To fill the performance-unaware gap of many (edge) network decision problems, the community has revived the decade old [12] idea of Data-driven networking [4], [13]–[15]. Despite the extensive use of machine learning to solve networking problems [16], $e.g.$, traffic classification [17], latency prediction [18] and video streaming bitrate optimization [19], most of these approaches follow into two categories: either a model is trained in a centralized fashion, as a Software-Defined Network controller application [16], [20], or distributed machine learning is used to train learning models faster [21]. While traffic engineering solutions have been devised using deep learning, see for example a chapter of this recent survey [16], to the best of our knowledge, approaches that support deep learning at every switch, and that provide performance-aware forwarding decisions learning from performance-unaware protocols are still at their infancy. While it may be challenging to apply our approach to wide-area or data center networks, despite the recent advances in high-performance switches, we believe that our approach can be ideal for the task offloading problem during critical networked scenarios, such as those of a data collection for situation awareness in disaster scenarios. We test our algorithm for task offloading over MiniNeXT [22], a network emulation environment based on container, and we evaluate the performance tradeoff within several policies using different network conditions. As a result of our study, we find a few expected and a few surprising results. One surprising result is that deep learning-based traffic offloading policies may not always help improving network performance (when each router runs a separate supervised learning model), so the training overhead time may not be justified.

Another message from our study lies in the poorly [23] explored use of our performance-agnostic traffic engineering policies to generate performance-aware policies. We release the code of our prototype [24] to allow the community to exploit it and explore other (deep learning-based) traffic

The work of A.Gaballo was performed as visiting scholar in the Computer Science Department at Saint Louis University, USA.

978-3-903176-24-9

offloading policies.

II. Offloading Path Prediction via Deep Learning

Low-latency is a crucial aspect of task offloading systems, especially when it comes to computationally expensive tasks. Several studies exist on characterization of the slow path in OpenFlow [25]; *it was surprising to us, however, that a neglected aspect in the edge computing literature [2] is the latency minimization of the inter-process communication among offloading servers and devices, while passing through an edge network.* To reduce such latency, we optimize the end-to-end path between the communicating parties by trying to predict not congested paths. In the rest of this section we explore the use of deep learning techniques to achieve this goal.

We exploit the congestion-agnostic limitation of traditional routing algorithms when applied in this context. These algorithms do not consider how rapid network load changes may affect the data-intensive, latency-sensitive needs of edge computing applications. Relying on higher level TCP-based solutions for congestion and flow control, that by design are (mostly) end-to-end, is insufficient. The path computed by standard routing protocols is computed by taking into account parameters such as the nominal interface speed.

The intuition behind our proposed solution is that *collaborative traffic steering* should be able to identify and avoid congestion situations, without using TCP or other active queue management approaches such as Explicit Congestion Notification (ECN). By collaborative we mean requiring (a priori or on demand) the participation of multiple network elements in the routing decision process. The information used by our protoype is the number of incoming packets on any given edge switch or router (node). The idea is that the packet distribution on the nodes reflects the network conditions. For example, a high packet count on a router is an indicator of a big load that is probably going to lead to packet loss and retransmission. We also have to consider that the distribution of packets on the routers is influenced by the routing algorithm: nodes that appear in multiple paths will probably have a higher count than less traversed nodes because they forward packets for multiple source-destination pairs. If routers were able to see all possible outcomes of a routing protocol in a network and extract the consequent traffic patterns, they could try to choose the less busy path.

Of course checking all possible outcomes is not scalable; it is known, however, that deep learning models use pruning search space strategies. We compare performance of multiple deep learning models by emulating a small network with ten routers, and using input given by the widely deployed routing algorithm Open Shortest Path First (OSPF) for training the deep learning component. We vary the network configurations and record the traffic patterns. A posteriori, we use the collected data and the routing choices taken by the routing protocol to build a model capable of predicting each hop of the path, from each source to each destination. With our approach, we are correlating traffic patterns and routing decisions; this correlation allows our system to dynamically adapt to the network conditions, a behavior that would not occur with a traditional routing algorithm.

The following section describes that steps we followed to converge to the final deep learning model.

A. Data Generation Process

The majority of datasets available to the community refer to traffic captured in datacenters, non-edge networks, or do not contain details about the underlying topology or the logged routing strategies. For these reasons, we created our own novel dataset by means of a network emulation strategy that considers all the elements we require to train our deep learning system. This includes *(i)* the network topology, *(ii)* the routing information, and *(iii)* the packet count on each node. The final dataset consists of a collection of samples containing, at any given time, the packet count together with the routing decision that was made. During the data generation process, the network is torn down and rebuilt with new link speeds, so that the OSPF configurations are different. We generated a dataset of $17,696$ samples; we then used 85% of these samples as a training set, and the remaining 15% was used as a test set.

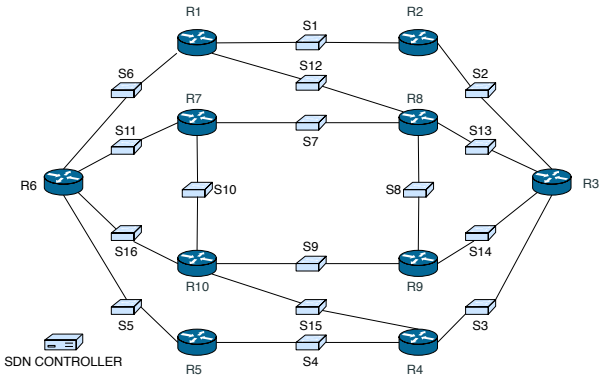

Fig. 1: Model topology: R1-R6 are outer routers while R7-R10 are inner routers. Each router runs a next-hop predictor based on LSTM.

B. Deep Learning Model

The deep learning model chosen for this work is a Long-Short Term Memory (LSTM) Recurrent Neural Network, a class of neural networks capable of using sequential information and to exhibit a dynamic temporal behavior. We wanted our model to learn the correlation between changes in the packet distribution and routing decisions over time.

Given the computational complexity of standard routing algorithms, training a single model to route all traffic becomes quickly infeasible as the network between mobile user and offloading server grows in size. To this aim, we trained a separate deep learning model for every source-destination pair, resulting in multiple simpler i.e., smaller, models. Assuming that each router only has a single outgoing interface, training an edge network with N routers will result in $N(N-1)$ deep learning models. Each model can be trained independently, making the training phase easy to parallelize.

1) Modeling Input and Output of the RNN: Supervised learning involves a sample space X and a label space Y, with the neural network responsible for learning a mapping function from values in X to labels in Y, for each $(x_i, y_i) \in X \times Y$. Our input/output modeling follows an approach similar to the one described in [26]. Given a set of outer routers O, and the set of all the routers in the edge network R, for each source-destination pair $(s, d) \in R \times O$, the deep learning system learns the next hop for destination d.

The easiest way to model the input is an N-dimensional array, with N being the number of routers in the network. Such an array is indexed by the router identifier, so the *i-th* element of the array is the number of incoming packets on router *i*. The output is modeled as a *one-hot encoded* [1] router indexed array, with a 1 in the position indexed by the predicted next hop; the size of the output is again equal to the number of routers in the network.

2) Choosing the Right Neural Network Architecture: In the cross-validation phase, we have tested several configuration by trying different combinations of hyperparameters such as the number of hidden layers and the number of neurons. As a result of this analysis, the model achieves the best performance when composed of: one input layer (10 neurons), two hidden layers (128 neurons ea., hyperbolic tangent activation [2]), one output layer (10 neurons, sigmoid activation).

After choosing the correct LSTM architecture, we also apply proper input normalization and regularization techniques to improve the training performance in terms of both accuracy and loss.

III. Evaluation Results

In this section we evaluate our architecture prototype. All our code is available at [24]. Our evaluation focus is the core of our novel LSTM based algorithm to predict least congested offloading paths. First, we detail the technologies used in our evaluation testbed, then we discuss how our system can emulate OSPF by analyzing the results of the model training; finally we discuss the performance of the path prediction model as a substitute to more traditional routing algorithms. For a more complete analysis, we also implement the same Deep Neural Network (DNN) described in [26], a traditional neural network with four hidden layers and sixteen neurons in each layer. We use this network to compare the performance between DNNs and LSTM for the same task and understand if our hypothesis about RNNs is correct.

A. Evaluation Testbed

Our prototype has been implemented using the following technologies: we employ Ryu [28] as an SDN controller and Google Protocol Buffers [29] as serialization/deserialization abstract syntax notation. To emulate the edge network we used MiniNExT [22], a Mininet [30] extension layer that supports routing engines and process identifier namespaces. Finally, we used Quagga [31] as a routing software suite and Keras [32] as a machine learning library.

[1]In machine learning, one-hot is a group of bits among which the legal combinations of values are only those with a single high bit and all the others low.

[2]The activation function defines the output of a node given an input [27].

Connectivity rate	Validation accuracy
30%	99.1%
35%	98.5%
40%	84.6%
45%	88.8%
50%	86.7%

TABLE I: Impact of the network density on the average validation accuracy of the deep learning model (randomly connected physical networks).

B. Overwriting OSPF Routing Decisions

To evaluate its performance when overwriting OSPF rules, we observe the behavior of the path prediction system in a functioning network. In particular, we use the same topology (Figure 1) and traffic simulator adopted in the dataset generation phase; to ease the analysis process, all links are set to the same rate. Afterwards, we select a source router and a destination address and examine the difference in behavior between OSPF and our system.

In general, our emulated edge network shows a dynamic behavior, and our prototype predicted several paths for the same destination under different traffic conditions. In particular, we run four traffic simulations, each of them for fifteen minutes, varying the loss rate on the link chosen by OSPF to connect source and destination; at the same time, the path prediction component computes a new path every five seconds. Considering Figure 1, the selected target is $(R1, R3)$, with the default path being $R1, R2, R3$ and the loss being varied on the link between R1 and R2. Being performance unaware, OSPF always chooses the same path, even when the link has (some) losses. To adapt the threshold, a human needs to manually reconfigure each router. Our system, on the contrary, shows the ability to behave dynamically by proposing four alternative paths.

By studying the system behavior in the presence of losses, it is possible to understand if our model is able to detect and overcome these problems. We test loss rates of 0%, 5%,10%, 15% and count the number of predictions different from OSPF (table II). With the loss set to 0%, 43% of the time the predicted path is different from OSPF; if the loss is increased to 5%, the ratio of paths different from OSPF slightly rises to 45%, indicating that the system is able to detect the change. The same happens for a loss of 10%, with a much more noticeable improvement in the system behavior; 63.5% of the proposed paths are in fact, different from the one chosen by OSPF. For the successive loss rate, equal to 15%, the performance goes down a little with only a 59.5% different path ratio; the reasons for this loss in performance are discussed in section IV. The ideal behavior would be for the system to detect the link loss and consequently stop predicting paths going through the damaged link. In our analysis this happens only with a limited loss rate.

Table III compares the resulting retransmission rate of our

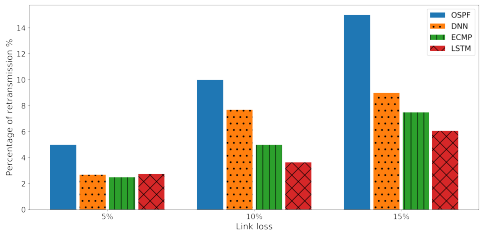

Fig. 2: Routing policies retransmission comparison. Our proposed LSTM policy has the highest throughput by minimizing retransmissions in challenged scenarios.

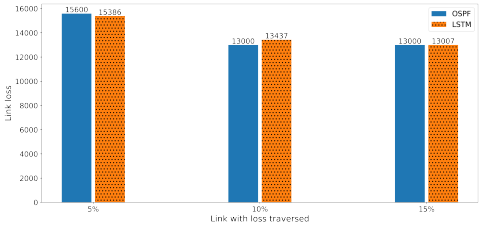

Fig. 3: Comparison of the number of (severely) lossy links traversed by OSPF and LSTM.

Link loss	Different path rate	Same OSPF path rate
0%	43%	57%
5%	45%	55%
10%	63.5%	36.5%
15%	59.5%	40.5%

TABLE II: Path predictions different and equal to OSPF.

	Routing Strategy			
Link loss rate	OSPF	ECMP	DNN	LSTM
0%	0%	0%	0%	**0%**
5%	5%	**2.50%**	2.70%	2.75%
10%	10%	5%	7.70%	**3.65%**
15%	15%	7.50%	9%	**6.07%**

TABLE III: Routing strategies retransmission rate comparison.

system, OSPF, and Equal Cost Multi Path (ECMP) routing algorithms. The retransmission rate is computed by taking into account how many times traffic would pass through the leaky link, considering two equal-cost paths for ECMP and the ratios in table III for our system. Overall, the LSTM policy that we propose has a lower retransmission rate than the other policies, therefore reaching a higher throughput. In Figure 2 we compare these three policies (LSTM, ECMP and OSPF), showing the overhead needed to transmit the same amount of data. When there is no link loss, the three policies behave very similarly; however, as soon as a loss rate is introduced, the performance gap of our proposed LSTM policy increases with the loss rate.

C. Evaluation in Challenged Scenarios

We compared several routing policies in critical scenarios, where network connectivity is scarce. We decide to simulate a network in which statistically, half of the links are affected by a loss rate; we use the same loss rates of the previous experiments (5%, 10%, 15%), running each experiment ten times, and generating traffic between five different targets. The purpose of this experiment is to understand if our approach is used with our LSTM policy, has higher resiliency than OSPF when up to half of the edge network are unavailable. To compare the performance of the two routing policies, we counted the number of times the lossy links were selected (Figure 3). The chart compares the total number of defective links traversed in all runs for each link loss rate. In this case, the LSTM policy does not introduce any significant advantage under critical circumstances; overall, the performance of the two policies are similar, with OSPF performing even better when the link loss rate is set to 10% and 15%. The reasons for the poor performance of the LSTM policy are due to our training approach; our LSTM policy predicts alternative paths based on the network conditions, proposing alternative paths. Given that half of the links in the network are affected by loss, the majority of the proposed alternative paths pass through these links, resulting in poor performance. In Section IV we give a few hints on how to overcome such limitations of these and other deep learning policies.

IV. Discussion and Conclusion

In this work, we presented an approach for task and path offloading. Our main goal has been to provide a testing platform for task offloading and routing policies, in support of offloading tasks traversing challenged edge networks. Our virtual network testbed prototype based on MiniNExT found interesting results and was released to allow the community to compare novel or existing routing policies in different edge computing scenarios [24].

In our prototype evaluation, we focused on a specific traffic offloading policy tradeoff. In particular, we compared deep learning based routing policies with ECMP and OSPF. Our policy tradeoff analysis exposed advantages and challenges of using deep learning as alternative to traditional routing algorithms, when deployed on a single node and not as centralized (SDN) controller application.

Despite the limited size of our dataset, our initial policy tradeoff analysis results have shown how a cooperative routing policy may lead to better performance than traditional routing methods at the edge, especially with unstable network conditions such as those that arise within an IoT network trying to operate at the network edge during a natural or man-made disaster.

Acknowledgment

This work has been supported by the National Science Foundation awards CNS-1647084 and CNS-1836906.

References

[1] M. Sharifi, S. Kafaie, and O. Kashefi, "A survey and taxonomy of cyber foraging of mobile devices," *IEEE Communications Surveys Tutorials*, vol. 14, no. 4, pp. 1232–1243, Fourth 2012.

[2] J. Wang, J. Pan, F. Esposito, P. Calyam, Z. Yang, and P. Mohapatra, "Edge cloud offloading algorithms: Issues, methods and perspectives," Oct 2018.

[3] G. Castellano, F. Esposito, and F. Risso, "A distributed orchestration algorithm for edge computing resources with guarantees," in *IEEE International Conference on Computer Communications*, ser. INFOCOM, 2019.

[4] A. Crutcher, C. Koch, K. Coleman, J. Patman, F. Esposito, and P. Calyam, "Hyperprofile-based computation offloading for mobile edge networks," in *The 14th International Conf, on Mobile Ad-hoc and Sensor Systems (IEEE MASS 2017)*, Orlando, USA, Oct. 2017.

[5] A. Ventrella, F. Esposito, and A. Grieco, "Load profiling and migration for effective cyber foraging in disaster scenarios with formica," in *IEEE 4th Conf. on Network Softwarization (NetSoft 2018)*, June 2018.

[6] F. Esposito, A. Cvetkovski, T. Dargahi, and J. Pan, "Complete edge function onloading for effective backend-driven cyber foraging," in *2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob 2017)*, Rome, Italy, Oct. 2017.

[7] J. Franz, T. Nagasuri, A. Wartman, A. Ventrella, and F. Esposito, "Reunifying families after a disaster via serverless computing and raspberry pis (demo)," in *IEEE Inter. Symposium on Local and Metropolitan Area Networks (LANMAN 2018)*, Washington, DC, June 2018.

[8] M. Chiesa, G. Rétvári, and M. Schapira, "Lying your way to better traffic engineering," ser. CoNEXT, 2016.

[9] D. Chemodanov, P. Calyam, and F. Esposito, "A near optimal reliable composition approach for geo-distributed latency-sensitive service chains," in *IEEE International Conference on Computer Communications*, ser. INFOCOM, 2019.

[10] F. Esposito, J. Wang, C. Contoli, G. Davoli, W. Cerroni, and F. Callegati, "A behavior-driven approach to intent specification for software-defined infrastructure management," in *IEEE Conference on Network Function Virtualization and Software Defined Networks*, ser. NFV-SDN, 2018.

[11] B. Eriksson, R. Durairajan, and P. Barford, "Riskroute: A framework for mitigating network outage threats," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '13. ACM, 2013, pp. 405–416.

[12] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, "A knowledge plane for the internet," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM, 2003, pp. 3–10.

[13] J. Jiang, V. Sekar, I. Stoica, and H. Zhang, "Unleashing the potential of data-driven networking," in *International Conference on Communication Systems and Networks*. Springer, 2017, pp. 110–126.

[14] A. Mestres, A. Rodriguez-Natal, J. Carner, P. Barlet-Ros, E. Alarcón, M. Solé, V. Muntés-Mulero, D. Meyer, S. Barkai, M. J. Hibbett *et al.*, "Knowledge-defined networking," *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 3, pp. 2–10, 2017.

[15] D. Chemodanov, F. Esposito, A. Sukhov, P. Calyam, H. Trinh, and Z. Oraibi, "Agra: Ai-augmented geographic routing approach for iot-based incident-supporting applications," *Future Generation Computer Systems*, 2017.

[16] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, Jun 2018.

[17] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[18] V. Bui, W. Zhu, A. Pescape, and A. Botta, "Long horizon end-to-end delay forecasts: A multi-step-ahead hybrid approach," in *2007 12th IEEE Symposium on Computers and Communications*, July 2007.

[19] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 197–210.

[20] A. Scalingi, F. Esposito, W. Muhammad, and A. Pescapé, "Scalable provisioning of virtual network functions via supervised learning," in *2019 IEEE Conference on Network Softwarization (NetSoft) (NetSoft 2019)*, Paris, France, Jun. 2019.

[21] R. Mayer and H. Jacobsen, "Scalable deep learning on distributed infrastructures: Challenges, techniques and tools," *CoRR*, vol. abs/1903.11314, 2019. [Online]. Available: http://arxiv.org/abs/1903.11314

[22] "MinineXt," http://mininext.uscnsl.net/.

[23] B. Schlinker, H. Kim, T. Cui, E. Katz-Bassett, H. V. Madhyastha, I. Cunha, J. Quinn, S. Hasan, P. Lapukhov, and H. Zeng, "Engineering egress with edge fabric: Steering oceans of content to the world," in *Proc. of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17, 2017, pp. 418–431.

[24] A. Gaballo and F. Esposito. ADELE code github.com/alegaballo/adele.

[25] R. Sanger, B. Cowie, M. Luckie, and R. Nelson, "Characterising the limits of the openflow slow-path," in *IEEE NFV-SDN*, Nov 2018.

[26] N. Kato, Z. M. Fadlullah, B. Mao, F. Tang, O. Akashi, T. Inoue, and K. Mizutani, "The deep learning vision for heterogeneous network traffic control: Proposal, challenges, and future perspective," *IEEE Wireless Communications*, vol. 24, no. 3, pp. 146–153, 2017.

[27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, 2016.

[28] "Ryu," https://osrg.github.io/ryu/.

[29] "Google protocol buffers," developers.google.com/protocol-buffers/.

[30] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010.

[31] "Quagga," http://www.nongnu.org/quagga/.

[32] "Keras the python deep learning library," https://keras.io/.