Deep Neural Networks for Network Routing

Joao Reis[1], Miguel Rocha[1,2], Truong Khoa Phan[1], David Griffin[1], Franck Le[3], Miguel Rio[1]
[1]University College London, [2]University of Minho, [3] IBM T.J. Watson Research Center
[1]{joao.reis.16,t.phan,d.griffin,miguel.rio}@ucl.ac.uk, [2]mrocha@di.uminho.pt, [3]fle@us.ibm.com

***Abstract*—In this work, we propose a Deep Learning (DL) based solution to the problem of routing traffic flows in computer networks. Routing decisions can be made in different ways depending on the desired objective and, based on that objective function, optimal solutions can be computed using a variety of techniques, e.g. with mixed integer linear programming. However, determining these solutions requires solving complex optimization problems and, thus, cannot be typically done at runtime. Instead, heuristics for these problems are often created but designing them is non-trivial in many cases. The routing framework proposed here presents an alternative to the design of heuristics, whilst still achieving good performance. This is done by building a DL model trained on the optimal decisions over flows from known traffic demands. To evaluate our solution, we focused on the problem of network congestion, even though a wide range of alternative objectives could be fitted into this framework. We ran experiments using two publicly available datasets of networks with real traffic demands and showed that our solution achieves close-to-optimal network congestion values.**

I. Introduction

In computer networks, the task of modelling and controlling network traffic to optimize network performance has been extensively studied, and many algorithms and solutions have been proposed. This task, called Traffic Engineering (TE), can be undertaken in different ways and with distinct purposes. A typical use case is to optimize network congestion [1], [2], [3], a problem that can be caused by a disproportionate amount of traffic going through a small set of the available paths between a source and a destination. Congestion can be due to several reasons, e.g, a routing protocol that always chooses the shortest path. This may lead to a less reliable service as overloaded equipment fails to handle traffic correctly. Other examples of TE applications exist, such as energy consumption minimization [4], [5] e.g. finding a routing solution minimizing the number of active links in order to save energy with minimal impact on the network performance.

Frequently, the solutions proposed by the community define an objective function and model an optimization problem that can be solved with methods such as (Mixed) Integer Linear Programming (MILP). However, in many cases the problem is proven to be too complex to solve in a reasonable amount of time [1], [6]. This is especially problematic with online algorithms, those that determine paths at runtime rather than planning them beforehand, since routing requires decisions to be taken in very short amounts of time. Because of that, researchers often have to design heuristics that optimize the same objective function, a task that is not trivial.

With applications to a number of different fields, including Computer Networks [7], [8], [9], Machine Learning (ML) models such as Deep Neural Networks (DNN) can be a suitable tool to address TE. This comes from their ability to generalize from previously seen examples and, once trained, take a relatively short time to process new inputs.

The idea we propose in this paper is an alternative to the design of heuristics by replacing them with a DL model trained in a supervised fashion. The training samples, consisting of traffic flows, are obtained by collecting or simulating traffic in a network. After that, the paths from source to destination, which are the training labels, are obtained by setting an objective function and solving the underlying optimization problem using MILP. It should be noted that although solving the MILP problem may take several seconds, or longer, this is feasible during the off-line training phase of the DNN. Having a trained model, it is then possible to query it in a relatively short time (compared to solving an optimization problem), and so determining paths as flows arrive, effectively being an online TE mechanism. The advantage of this approach is that, by not having to design a heuristic for each objective desired, a network operator has the flexibility to easily implement a wide range of objectives.

The proposal of using Supervised Learning (SL) techniques as a tool for TE is not new. Previous work include SL models to forecast future traffic [10] and use those predictions with conventional optimization methods. Furthermore, there has been also research on SL models to directly learn paths [11], [12]. While in [11], the authors proposed a routing strategy in a decentralized setting where each node in the network trains a model for each destination and the output is the next hop, [12] is more similar to our work, presenting the idea of building centralized models that use knowledge from the whole network as inputs and output full paths from source to destination; however, they train a different model for each source and destination pair in the network. Furthermore, their evaluation consists of generating traffic, choosing a heuristic to obtain paths, training the model and testing how the trained system performs against that heuristic.

Below, we summarize the differences from previous work and our contributions:

1) We propose a flexible SL-based framework that learns paths directly from optimal MILP solutions and trains a single model incorporating all source-destination pairs. Furthermore, we propose a post-processing algorithm that ensures the validity of the paths obtained.

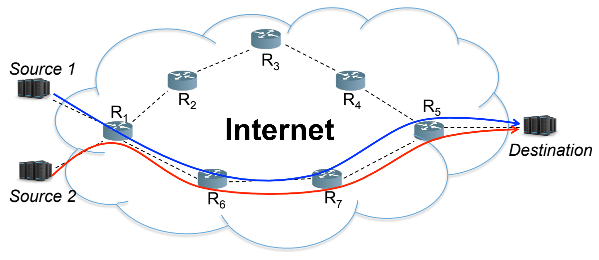


Fig. 1: Example of shortest path routing.

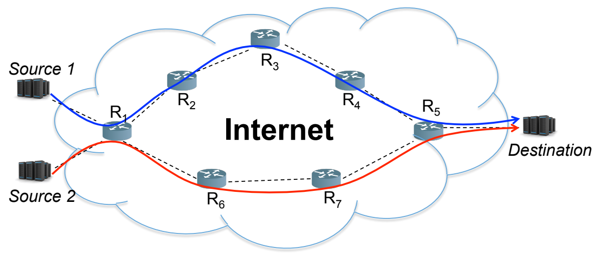


Fig. 2: Example of MLU routing.

2) We evaluate the flexibility of this kind of approach by testing it with two different objective functions.
3) Finally, we use datasets generated from real traffic obtained from real networks [13], [14].

The rest of this work is organized as follows. In Section II we give an overview of routing in the Internet. In Section III we review and discuss prior work on using ML in the computer networking field, especially for TE. We explain our model in Section IV. In Section V the methodology of the experiments is explained, including how the data was obtained and generated. In Section VI we report and discuss our results. Finally, in Section VII we present our conclusions.

II. Routing in the Internet

A. Routing policy

Devices that are connected in a network and want to communicate with each other must follow a series of rules and protocols. For instance, communications in the Internet are done according to the Internet protocol suite, that includes the Transmission Control Protocol (TCP) and the Internet Protocol (IP).

One of the challenges that these protocols address is what should be the path taken by traffic from its source to the destination. This is the routing problem, and in the cases where more than one path between source and destination exists, this decision must follow some criteria. A simple one would be to make traffic go through the least number of intermediate nodes, as shown in Figure 1.

More generally, routing can be done according to a specific objective or metric, such as minimizing delay or congestion, which can be determined with TE. In all these cases, there is an optimal solution that can be determined using some mathematical tool such as linear programming. In practice, however, a network is expected to be dynamic and optimal solutions change frequently. In the case of congestion, for example, new traffic that enters the network uses resources and influences how future traffic should be routed. Therefore, to constantly route according to the optimal solution, the algorithm would also have to be constantly re-run. In a real-world scenario that could be prohibitively expensive. So, what is done instead is to design a heuristic that can be run with greater speed, but that does not give guarantees of optimality. In the Open Shortest Path First (OSPF) [15] protocol, widely used in real networks, this can be implemented by adapting the link weights accordingly and running an algorithm such Dijkstra's shortest path [1]. An illustration of how TE affects routing can be seen in Figure 2, where the chosen path from Source 1 to Destination is not the shortest one but the one that minimizes congestion.

Our proposed solution is to train a DL model on these optimal solutions so that the routes provided lead close to optimal networks states without having to design any heuristic. In this work, we focus on two congestion metrics, maximum link utilization (MLU) and the one proposed by Fortz [1], but this could have wider applications with alternative objective functions, e.g. energy consumption minimization or other Quality of Service (QoS) metrics such as network latency.

B. Network flow

Information in networks can generally be seen as being transmitted by packets. In TCP/IP networks packets have, among other properties, a destination IP address. Standard routing protocols, such as OSPF, look at packets individually and take their decisions solely based on the destination address. This type of routing, destination-based, routes any packet that has the same destination address along the same path, regardless of other properties, such as source address or TCP ports.

Additionally, there is the notion of flow, which is a set of packets that share some properties. These properties could be, for instance, the 4-tuple composed by the source and destination IP address and TCP port. The concept of routing can also be applied to flows such that each packet of a flow goes through the same path. This gives a much more granular control over routing, as packets that have e.g. the same source and destination, but differ on TCP ports can have different routes. For example, video or web applications between the same source and destination might have distinct requirements and, thus, benefit from going through different paths. In this work, we assume packets are already aggregated and consider routing based on flows, i.e. all packets of the same flow will be routed through the same paths.

III. Related Work

The application of ML to Computer Networks is not centered in one topic, rather it covers a wide range of problems. For instance, ML has been used to improve networks security addressing issues such as intrusion detection [7] or attacks on Internet of Things networks [16]. As an example, in [9] an

ML model is trained on examples of malicious and benign connections to detect Distributed Denial of Service (DDOS) attacks. A different area where ML can contribute is energy efficiency in data centers, where there are many variables to control. The work done in [8] shows that an ML model can predict power usage effectiveness (PUE) by leveraging data collected in a setting where many monitoring mechanisms are present.

In terms of network routing and TE, the proposed solutions can be categorized by their approach.

A. Reinforcement Learning (RL)

In general, RL-based routing models the problem as sequence of routing decisions, where the reward is based on a network performance metric, e.g delay or congestion. The algorithm presented in [17], Q-routing, based on the known Q-learning [18], trains an RL agent in a router to react to the dynamics of networks, e.g increasing network load. Interestingly, they show it works better than some standard routing protocols, even though it is done in a distributed fashion where each agent only receives local information. Also, RL has been used to route traffic in networks where energy efficiency [19] or stability [20] are considered. In [10], RL is used to learn a routing policy with a reward that depends on the optimal solutions. This is done in a centralized way, unlike [17].

B. Supervised Learning (SL)

SL approaches to routing seem to be less prevalent when compared to RL. Nevertheless, there has been some recent work in this topic. Before suggesting an RL approach, [10] proposes an SL-based solution. Instead of directly addressing routing, it predicts future traffic and then optimizes the routing plan with respect to the predicted values. However, simulation results show that this approach might be ineffective.

Furthermore, in [21] they make the case for intelligent routing and propose a Deep Learning (DL) based solution as a use case. Later, they propose in [11] a decentralized DL-based routing system that outputs the next hop in the path. Therefore, each node in the network trains a different model for each possible destination that it can send traffic to. On the other hand, the value of the input is the same for all models and consists of the traffic measurements made by each node in the network. These are periodically propagated by the nodes in order to keep the models synchronized. They conclude that their system has a better performance in terms of signalling, throughput and delay than OSPF.

Another routing system using SL, that shows promising results, is presented in [12] which, like the framework we propose here, assumes a central controller that gathers information from the whole network and uses it as inputs to a ML model. The framework trains a separate model for each source-destination pair, unlike ours where a single model incorporating all source-destination pairs is trained. Compared to previous work, a difference our system brings is the way it deals with the output. Instead of directly interpreting the output as a path, we employ a post-processing mechanism that ensures path correctness. That way we avoid issues such as non existent paths or loops.

In terms of evaluation, [11] trains the models with paths from a standard protocol, whereas [12] uses the solutions provided by a heuristic to a congestion optimization problem. In contrast, we train our models with paths directly obtained from the optimization algorithm. Furthermore, we test our framework with two different objective functions to evaluate its flexibility.

IV. ML BASED ROUTING

A. Problem formulation

We address the problem of finding the best path for a given sequence of flows, each arriving at the network at a given time point (discrete), in a way that optimizes an objective function defined along a period in time (e.g. minimizing network congestion). Although many different objective functions may fit this framework, in the current work, we have addressed the minimization of network congestion, measured by different metrics.

In this work, two objective functions were used, both aiming to minimize the network congestion: the classical maximum link utilization (MLU) and the congestion measure proposed by Fortz and Thorup in their seminal paper [1]. Although the purpose is to route individual flows, this problem can be considered a variant of the *general routing problem*. The network is defined as a directed graph $G = (V, E)$, where V represents a set of nodes, while E represents links connecting them. Each link e has a capacity, defined as $c(e)$ defining the amount of traffic it can accommodate, and from this the link utilization $u(e)$ may be calculated by dividing the load passing through the link, $l(e)$, by $c(e)$.

In this case, the traffic to route is defined by a sequence of flows F, each represented as a tuple $f_i = (toa_i, s_i, d_i, b_i, dur_i)$, where f_i is the i-th flow in the sequence, toa_i is its time of arrival, s_i and d_i its source and destination nodes, b_i its requested bandwidth, and dur_i its predicted duration. When a flow f_i arrives at the network, the controller needs to define a path $\mathbf{p_i}$, consisting of a list of links from E leading from s_i to d_i. Here we assume single-path, i.e. the whole set of packets belonging to this flow will follow the same path. During the lifetime of f_i, from toa_i to $toa_i + dur_i$, this flow is active and its bandwidth b_i is added to the loads of all links in $\mathbf{p_i}$. Thus, when performing a simulation considering a network G and a sequence of flows F, it is possible to calculate the link utilization for all links at all time steps $i = 1, \ldots, |F|$, and from those to calculate a metric of congestion, also at each point.

Here, we consider two metrics: maximum link utilization (MLU), consisting of the largest value of $u(e)$, for $e \in E$, or the one proposed by Fortz et al. [1] taking into account all links in the network. The latter is defined by a cost Φ which is the sum of the cost function $\Phi_e(u(e))$ for each link $e \in E$. This is a convex piecewise linear penalty function, shown in Figure 3, that has relatively low growth for small link

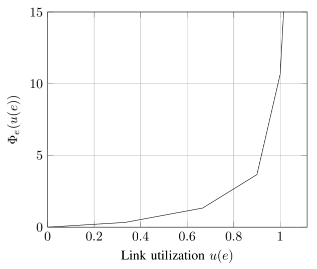


Fig. 3: Fortz link cost function [1].

utilizations, but increases more quickly with the utilization as it approaches unity, and when $u(e)$ goes above 110% grows with a very large derivative. This congestion measure, named Φ^*, can be normalized, taking into account overall demands and the shortest paths between sources and destinations. This normalized version has a lower bound of 1, and will be used in this work. Notice that, if all link utilizations are equal to 1, the value of the normalized congestion is 10.67.

B. Model description

We address the previous routing problem by using a framework based on Supervised Learning (SL) models to determine the best path for each flow. The proposed routing system consists of two components: a deep neural network (DNN) to determine the path and a post-processing routing algorithm to build a valid path from the DNN's outputs.

a) Input: The SL model, in our case a DNN, determines the route of each flow, given a set of features characterizing the flow and the current network state. In SL, we are interested in using a set of N examples consisting of input-output pairs $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ to create a model $f(\mathbf{x}) = y$ that can accurately predict the value of an output vector y^* given an input $\mathbf{x}^*$. In our case, each input is the concatenation of the features of the flow (namely its source, destination and the predicted bandwidth) and the current network state (represented by the link utilization vector, with $u(e)$ for all links in the network) at the time the flow arrives. For practical purposes, we use one hot encoding [22] for the source and destination nodes, and bandwidth is normalized to the range [0,1].

b) Output: The ground truth for each flow is its optimal path according to the objective function being used. In this work, to find these optimal paths, we formulate a Mixed Integer Linear Program (MILP) that minimizes the selected objective function, either the MLU or the Fortz et al. congestion measure. For the latter, the linear programming formulation is fully described in [1]. The MILP solution for a flow will depend not only on the flow properties - source, destination and required bandwidth - but also on the link loads $l(e)$ with $e \in E$, which will determine the available capacity of each link (the original subtracted by the load imposed by previous flows active in the network).

So, we solve a MILP problem for each flow and obtain the optimal path. This allows to calculate the link utilization induced by those paths along time, as well as to compute the defined objective functions. Notice that the solutions provided by MILP are locally optimal, since they provide the optimal allocation of the flow given the active flows at that time. However, this does not guarantee optimal congestion as new flows arrive, since we do not re-optimize the paths for flows already in the network, and thus does not guarantee optimal mean or median congestion measures across the period of observation.

In this context, knowing a flow's duration is only useful to keep track of active flows in the network, necessary to update the available link capacities. We assume that available link capacities in each point are known, and flow durations are not used by the MILP. Thus, the input to the DNN does not include duration, as it would possibly add noise without providing additional information.

We represent the path $\mathbf{p}_i$ for a flow f_i as an indicator vector of E. Thus, $p_{ik} \in \mathbf{p}_i$ is 1 if flow f_i is routed through the k-th link in E, and 0 otherwise, for $k = 1, \ldots |E|$. These binary vectors of size $|E|$ will be the desired outputs for the DNN.

c) Deep Neural Networks: As mentioned before, the ML model selected in our work was a Deep Neural Network, which has as inputs the flow features and link utilizations (as defined above), and outputs a value for each link. This defines the input and output layers, whose sizes will be $2 \times |V| + 1 + |E|$ and $|E|$, respectively.

The architecture chosen was a fully connected (dense) neural network, with the ReLu activation function used in all neurons of the hidden layers and the sigmoid function used in the output layer. The DNN was trained using either the Adam [23] or the RMSProp algorithms, optimizing the binary cross entropy loss function. To avoid overfitting, we employed a dropout [24] mechanism.

To choose the best model for each dataset, we performed a hyperparameter optimization through a grid search procedure, trying different configurations regarding the topology, the training algorithm, the number of epochs in training, the dropout rate and use of early stopping. For each configuration, we checked the performance of the network by testing it on a validation set, and selected the model which minimizes the congestion metric in the validation set.

d) Post-Processing Algorithm: A simple way to convert the output of the DNN into a path would be to apply a threshold (e.g. 0.5) to each output unit adding the corresponding link to the path if its value was above the threshold. However, this might easily lead to invalid paths. Therefore, to ensure that a valid path is obtained, we apply a post processing algorithm on the neural network outputs. Two alternatives were tried, including the use of Dijkstra shortest path algorithm, where link weights were set according to the inverse of the DNN's output. This allows to get valid paths that take into account the DNN's preference since the chosen path is more likely to go through links with higher DNN output values, thus lower link weights.

An alternative that more directly takes into account the DNN's outputs is the use of a greedy heuristic, where we start by the source node of the flow, and follow the outgoing

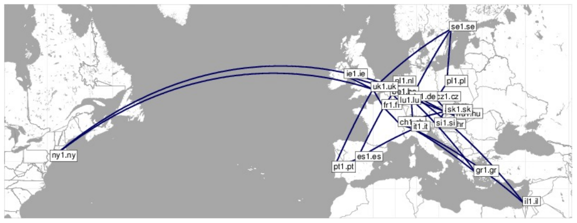


Fig. 4: GÉANT topology [27].

link of that node with the highest output from the DNN. This process is repeated until the destination node is reached (and a valid path obtained) or we get stuck into a cycle. In this latter case, we resort to the Dijsktra algorithm, as described in the previous paragraph, to obtain a valid path.

After training a DNN as stated above, it can be used to route a sequence of flows F, computing the paths for each flow using its output post-processed by the previous algorithms. The process is similar to the one described for MILP above, processing each flow in the order of arrival, calculating the path, updating the network state (link utilizations) and calculating the target objective functions (MLU or Fortz).

V. Methodology

A. Topology and Traffic Matrices

To perform a representative evaluation of our method, a real world scenario was needed. That meant having information about a known network, including its topology and the link capacities. In addition to that, it was also required to know the traffic going over that network in a certain period of time. Specifically, in terms of traffic we were interested in obtaining a temporal sequence of flows with known source, destination, bandwidth, duration and time of arrival. However, to the best of our knowledge, no such set of flow data and its underlying topology is publicly available. Either the datasets have a real sequence of flows, but no information about the topology is given [25] or the topology is given, but not the information about individual traffic flows [26].

However, there are some datasets with topology and aggregated information about traffic. The information comes in the form of a $|N| \times |N|$ Traffic Matrix (TM) for which the element in row i and column j represents the total amount of traffic (or average bandwidth) in a certain period of time between nodes i and j.

B. Dataset

For the evaluation of this work we used two datasets: one with the topology and TMs from the GÉANT network [14] and another with the same information from the Abilene network [13]. The GÉANT (illustrated in Figure 4) network dataset makes available its network topology including information about the link capacities. Moreover, it includes 11460 TMs, each corresponding to a period of 15 minutes totalling 4 months of observations.

The Abilene (Figure 5) dataset follows offers similar information with the difference that each TM refers to a period

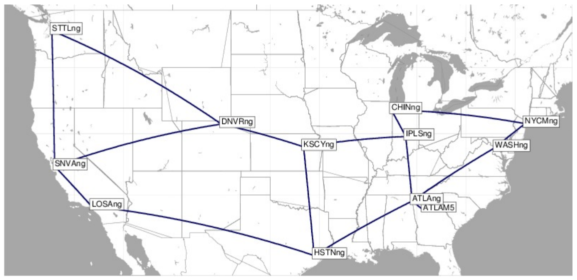


Fig. 5: Abilene topology [27].

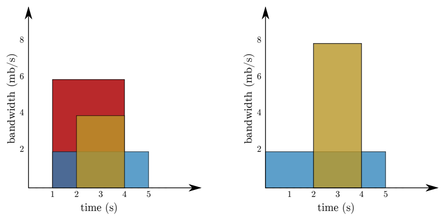


Fig. 6: Two possible sequences of flows that can be obtained from 34 Mb traffic volume between two nodes.

of 5 minutes. It contains 48096 TMs in a period equivalent to 167 days (although not continuous, as there are some days when no measurements were made).

Given that in a TM only aggregated information is available, some assumptions had to be made to generate a sequence of flows. This is because different sequences of flows, when combined, can translate to the same TM. This problem is illustrated in Figure 6, where each blue rectangle represents a flow with width and height proportional to duration and bandwidth, respectively. In that example, it is known that the volume of traffic between two nodes is 34 Mb, however no information about individual flows is provided. Two possible options, out of an infinite number of them, are shown in each plot.

In this way, we assumed the number of flows and the distributions for the total traffic, duration and arrival time of each flow. Furthermore, to create a larger and more representative dataset we combined sequential TMs to obtain a dataset with a period of one day, 96 (15 minutes TMs) for the GEANT network and 288 (5 minutes TMs) for the Abilene network. To have independent train, validation and test datasets we picked 3 different sets of 96 consecutive TMs such that they were from different days but the same day of the week.

For each TM, the flow generation procedure works as follows. We iterate over all possible pairs of source and destination where the traffic demand is greater than zero. For each source-destination pair (s, d), we take the value $t_{s,d}$ from the TM and calculate $K_{s,d}$, the number of flows that should exist between s and d. To determine $K_{s,d}$, we first calculate the number of flows per TM, obtained by dividing the desired size for the dataset (an input to the procedure) by the number of TMs, and then find $K_{s,d}$ by the proportion of traffic between s and d, $t_{s,d}$, relative to the whole TM, $\sum_s \sum_d t_{s,d}$. Notice that because the number of flows must be a positive

TABLE I: Summary of network and flow generation parameters.

	GÉANT	**Abilene**
$\lvert V\rvert$	22	12
$\lvert E\rvert$	72	30
Nr. Flows	96000	96000
Nr. TM	96	288
Time per TM (s)	900	300
Traffic Volume (Mb)	$\mathcal{N}(\frac{t_{s,d}}{K^{s,d}}, \frac{t_{s,d}/K^{s,d}}{3})$	
Flow Duration (s)	$\mathcal{N}(60, 12)$	
Flow Arrival (s)	$\mathcal{U}(t_{tm}, t_{tm}+900)$	$\mathcal{U}(t_{tm}, t_{tm}+300)$

integer, rounding computations may lead to an actual number of flows that is different than the one input to the procedure. Next, $K_{s,d}$ samples $tr^k_{s,d}, k \in [1, K]$ of traffic volume are drawn from a Gaussian distribution such that they add up to $t_{s,d}$. After that, the time of arrival $toa^k_{s,d}$ and the duration $dur^k_{s,d}$ are respectively sampled from a Uniform and Gaussian distributions. Finally, the bandwidth $b^k_{s,d}$ is set as $\frac{tr^k_{s,d}}{dur^k_{s,d}}$ and each flow has the form $f^k_{s,d} = (s, d, b^k_{s,d}, toa^k_{s,d}, dur^k_{s,d})$ being stored by order of arrival.

The procedure is then repeated for the next TM and the time of arrival distribution is shifted by the TM period. Therefore, all the flows generated from a given TM, tm, have a time of arrival between an initial time for that window, t_{tm}, and t_{tm}+P with the TM period P being 900 or 300 seconds depending on the network.

The link utilization, that together with the flows form the input, and the paths, the ground truth, are determined together. The procedure works by obtaining the optimal path for a flow, updating the link utilization and using that for the next flow. At this stage, we created two different sets of optimal paths, one optimized for MLU and another for the Fortz metrics.

Finally, as a simple baseline, we computed the routing based on shortest paths (SP), for each flow. The calculation of these paths did not take congestion into account, rather it looked for the least possible number of hops. We chose this baseline as SP is the most widely used routing algorithm in the Internet and any suggested improvements or advances in routing should be compared to it.

In Table I, a summary of the network and features of the datasets is shown, where Nr. Flows is the desired size for each dataset. The actual sizes of the datasets are detailed in Table II. Note that the flows for the MLU and Fortz are the same, so the number of samples in both datasets is identical. Furthermore, we scaled up the Abilene traffic volume by 10 times as it was too low relative to the link capacities, thus making the problem more challenging and allowing a better comparison of the performance between MILP, DNN and SP.

C. *Experimental methodology*

More than aiming for path prediction accuracy, the evaluation done in this work was focused on how our proposed solution compared to the optimal solution in terms of the metric being optimized. In practice, after routing each flow we measured the link utilization and computed the MLU or Fortz metrics over time. To compare different models we aggregated them by calculating the mean in the case of MLU, and the median for Fortz measure (since the penalties in this case are best viewed in log scale).

TABLE II: Train, validation and test set sizes for both networks.

	GÉANT	**Abilene**
Train	82402	79043
Validation	81877	78006
Test	82943	77296

TABLE III: Hyperparameter used in grid search optimization

Parameter	**Values**
Configuration	
1 hidden layer	[20], [100], [500]
2 hidden layers	[20,100], [100,100], [500,200], [500,250]
Epochs	2, 5, 10, 50, 100
Early Stop	5, 10
Dropout Rate	0, 0.2
Optimization Algorithm	Adam, RMSProp

We performed a hyperparameter optimization by repeatedly training DNNs with different hyperparameters, in a grid search, considering the set of values for each parameter shown in Table III). For each, we evaluated the MLU of Fortz metrics in the validation set and present here the results that the model with best validation performance had on the test set.

VI. Results and Discussion

A. *Results*

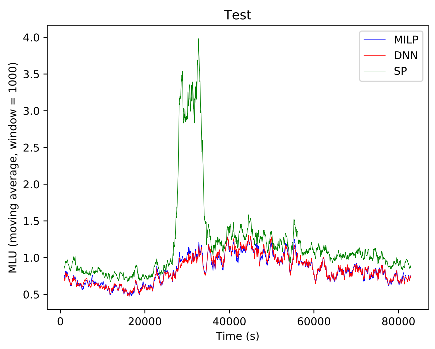
The results of our experiments over the test set flows are shown in Figures 7 and 8 and aggregated in Table IV.

In the GÉANT network, the aggregated scores of the DNN models are close to those obtained by MILP. In terms of MLU, we can observe that it is only 1.08% higher and it does an even better job at optimizing Fortz as the difference drops to 0.44%. On the other hand, in line with our expectations, the performance of the SP is much worse than that of the DNN for both metrics, reporting an average MLU and a median Fortz that were 41.93% and 27.95% higher than MILP, respectively.

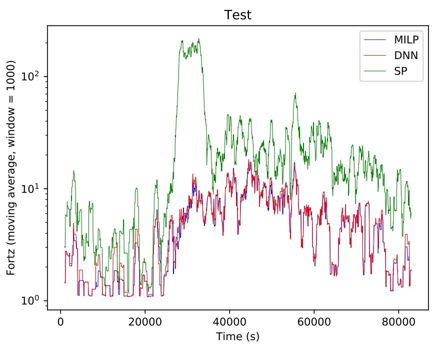
Even more interesting is the performance of our proposed approach in the Abilene network dataset. The results show that it outperforms the MILP induced paths by showing better average MLU by 0.96%, and median Fortz score by 1.92%. Note that, when the network state is the same, the performance

TABLE IV: Overall results for network congestion.

Dataset	**Objective**	**MILP**	**DNN (△)**	**SP (△)**
GEANT	MLU	0.830	0.839 (+1.08%)	1.178 (+41.93%)
	Fortz	1.145	1.150 (+0.44%)	1.465 (+27.95%)
Abilene	MLU	0.521	0.516 (-0.96%)	0.862 (+65.45%)
	Fortz	1.201	1.178 (-1.92%)	1.359 (+13.16%)

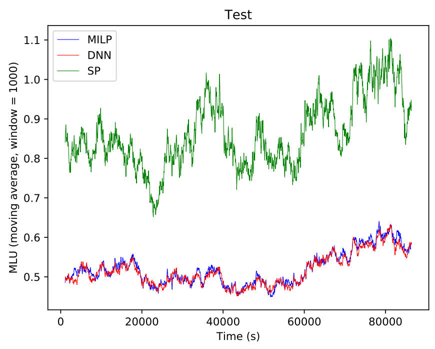


(a) MLU

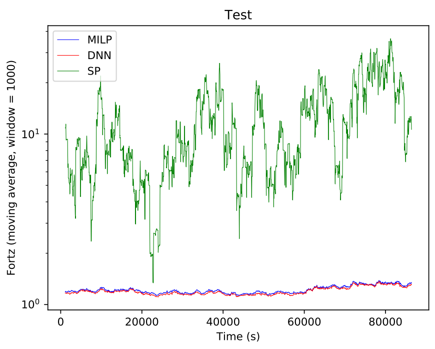


(b) Fortz

Fig. 7: Congestion measures for GÉANT dataset.



(a) MLU



(b) Fortz

Fig. 8: Congestion measures for Abilene dataset.

of the MILP solution is always at least as good as the DNN one. However, because we are optimizing locally, flow by flow, the network state at each flow is not necessarily the same. For example, if a flow f is routed through path p_{MILP} by MILP and path p_{DNN} by the DNN and if $p_{\text{MILP}} \neq p_{\text{DNN}}$, then the network state will be different when the next flow arrives. Because the DNN in our experiments did not show 100% accuracy, thus providing different paths than MILP for some flows, it is possible that its performance is better than MILP at certain points. This may occur since the DNN training may capture regularities in traffic and optimize its behaviour towards this, in a certain sense forecasting the next flows when routing the current one.

A more dynamic perspective is provided by Figures 7 and 8, where we show the moving average of the congestion metrics over the simulated time of 1 day (86400 seconds). Each line corresponds to a different approach and the reason for plotting a moving average, rather than the actual values, is because it would not be possible to compare the approaches given the high number of points and consequent overlapping between lines.

Figure 7a shows the MLU results in the GÉANT network and it is possible to see the two lines of DNN and MILP having similar shape. Although the SP approach is normally relatively near the other two, it is interesting to observe the period around second 30000 where it gets values almost four times worse then its mean. This is possibly due to the existence of a high volume of traffic between two nodes whose shortest path is of relatively low capacity and serves as a good illustration of network congestion when no TE is done. A similar interpretation can be done for the Fortz experiments, in Figure 7b, while the difference here from SP to the other two is even more pronounced. A log scale was used for plotting those results as it seemed more appropriate given the Fortz link cost function exponential nature in the domain shown in Figure 3.

Similar remarks can be made about the plots in Figure 8. These show that, in the Abilene network, the SP performs consistently worse than MILP and DNN, both for MLU and Fortz. It is also possible to see that the performance of the DNN and MILP, across the time period, was similar for the two objective functions.

B. Discussion

With a good performance across two objective functions and two networks, the results show that a DL-based system could replace the design of heuristics without compromising congestion. To change from one objective function to the other, we simply reformulated the MILP and ran the same pipeline. The most time consuming task of this procedure was the hyperparameter optimization, which can be easily automated.

With typical network equipment, the implementation of such solution would be very challenging as it lacks the required computational resources. However, recent changes to the paradigm of networks have been occurring with the popularization of Software Defined Networking (SDN) [28]. One of the novelties of SDN is that the control plane of a network is done by a central controller which can be an intelligent device. This controller can have a global view of the network and the capacity to issue commands to any other device in the network.

Regarding protocol overhead, we require that routers periodically send to the network (and the controller) the status of the link loads. We do not require this to be very accurate and since there is good evidence that load does not change significantly in 10 minute intervals [29], this overhead is negligible.

VII. Conclusions and Further Work

We presented a novel routing framework that allows to replace heuristics for Traffic Engineering with Deep Learning models and evaluated it with datasets based on real traffic. Our results show that the same ML framework can achieve quasi-optimal performance with two different objectives functions.

Future work will address research on methods that include memory (e.g. Long Short-Term Memory (LSTM) networks), seeking to overcome the need to input network utilization. Additionally, we will research on Reinforcement Learning as a way to learn without explicit training data, just from rewards from the objective functions. Moreover, we want to explore what can be done with a trained model when the topology of the network changes, i.e, how resilient is it if the post-processing algorithm is updated accordingly. Finally, we will evaluate different objective functions, incorporating, for instance, delay requirements or energy consumption metrics.

VIII. Acknowledgements

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001.

References

[1] B. Fortz and M. Thorup, "Internet Traffic Engineering by Optimizing OSPF Weights," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2000.

[2] A. Elwalid, C. Jin, S. Low, and I. Widjaja, "Mate: Mpls adaptive traffic engineering," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2001.

[3] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the tightrope: Responsive yet stable traffic engineering," in *ACM SIGCOMM Computer Communication Review*, vol. 35, pp. 253–264, ACM, 2005.

[4] N. Vasić and D. Kostić, "Energy-aware Traffic Engineering," in *International Conference on Energy-Efficient Computing and Networking*, pp. 169–178, 2010.

[5] F. Giroire, J. Moulierac, and T. K. Phan, "Optimizing rule placement in software-defined networks for energy-aware routing," in *IEEE Global Communications Conference (GLOBECOM)*, 2014.

[6] M. Kodialam and T. Lakshman, "Minimum interference routing with applications to mpls traffic engineering," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2, pp. 884–893, IEEE, 2000.

[7] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, and W.-Y. Lin, "Intrusion Detection by Machine Learning: A Review," *Expert Systems with Applications*, vol. 36, no. 10, pp. 11994–12000, 2009.

[8] J. Gao, "Machine Learning Applications for Data Center Optimization (Google White Paper)." https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42542.pdf.

[9] A. A. Diro and N. Chilamkurti, "Distributed attack detection scheme using deep learning approach for Internet of Things," *Future Generation Computer Systems*, vol. 82, pp. 761–768, 2018.

[10] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, "Learning to Route," in *ACM ACM Workshop on Hot Topics in Networks (HotNets)*, 2017.

[11] B. Mao, Z. M. Fadlullah, F. Tang, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "Routing or Computing? The Paradigm Shift Towards Intelligent Computer Network Packet Transmission Based on Deep Learning," *IEEE Transactions on Computers*, vol. 66, no. 11, pp. 1946–1960, 2017.

[12] L. Yanjun, L. Xiaobo, and Y. Osamu, "Traffic Engineering Framework with Machine Learning Based Meta-layer in Software-defined Networks," in *IEEE International Conference on Network Infrastructure and Digital Content (IC-NIDC)*, 2014.

[13] "Abilene Dataset." http://www.cs.utexas.edu/~yzhang/research/AbileneTM/.

[14] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing Public Intradomain Traffic Matrices to the Research Community," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 83–86, 2006.

[15] "OSPF Version 2 (RFC 2328)." https://tools.ietf.org/html/rfc2328.

[16] M. Mamdouh, M. A. I. Elrukhsi, and A. Khattab, "Securing the Internet of Things and Wireless Sensor Networks via Machine Learning: A Survey," in *International Conference on Computer and Applications (ICCA)*, pp. 215–218, IEEE, 2018.

[17] J. A. Boyan and M. L. Littman, "Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach," in *Advances in Neural Information Processing Systems 6*, pp. 671–678, Morgan Kaufmann, 1994.

[18] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[19] F. Kiani, E. Amiri, M. Zamani, T. Khodadadi, and A. Abdul Manaf, "Efficient Intelligent Energy Routing Protocol in Wireless Sensor Networks," *International Journal of Distributed Sensor Networks*, vol. 11, no. 3, p. 618072, 2015.

[20] A. Ghaffari, "Real-time Routing Algorithm for Mobile Ad-hoc Networks using Reinforcement Learning and Heuristic Algorithms," *Wireless Networks*, vol. 23, no. 3, pp. 703–714, 2017.

[21] Z. M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "State-of-the-Art Deep Learning: Evolving Machine Intelligence Toward Tomorrow's Intelligent Network Traffic Control Systems," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2432–2455, 2017.

[22] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-power CMOS Digital Design," *IEEE Journal of Solid-State Circuits*, 1992.

[23] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *CoRR*, vol. abs/1412.6980, 2014.

[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[25] "The CAIDA UCSD Anonymized Internet Traces 2016." https://www.caida.org/data/passive/passive_2016_dataset.xml.

[26] S. Knight, H. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet Topology Zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765 –1775, 2011.

[27] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0–Survivable Network Design Library," *Networks*, 2010.

[28] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, 2008.

[29] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford, "DONAR: Decentralized Server Selection for Cloud Services," in *ACM SIGCOMM*, 2010.