Electronic Communications of the EASST
Volume 080 (2021)

Conference on Networked Systems 2021
(NetSys 2021)

Navigating Communication Networks with Deep Reinforcement Learning

Patrick Krämer Andreas Blenk

16 pages

Guest Editors: Andreas Blenk, Mathias Fischer, Stefan Fischer, Horst Hellbrueck, Oliver Hohlfeld, Andreas Kassler, Koojana Kuladinithi, Winfried Lamersdorf, Olaf Landsiedel, Andreas Timm-Giel, Alexey Vinel
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Navigating Communication Networks with Deep Reinforcement Learning

Patrick Krämer[1] **Andreas Blenk**[12]

[1] Chair of Communication Networks, Technical University of Munich, Germany [2] Faculty of Computer Science, University of Vienna, Austria

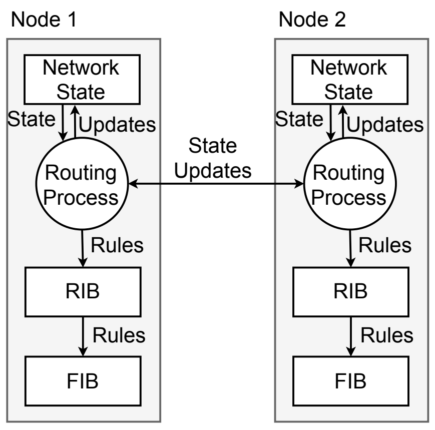
Abstract: Traditional routing protocols such as Open Shortest Path First cannot incorporate fast-changing network states due to their inherent slowness and limited expressiveness. To overcome these limitations, we propose ComNav, a system that uses Reinforcement Learning (RL) to learn a distributed routing protocol tailored to a specific network. ComNavinterprets routing as a navigational problem, in which flows have to find a way from source to destination. Thus, ComNavhas a close connection to congestion games. The key concept and main contribution is the design of the learning process as a congestion game that allows RL to effectively learn a distributed protocol.Game Theory thereby provides a solid foundation against which the policies RL learns can be evaluated, interpreted, and questioned. We evaluate the capabilities of the learning system in two scenarios in which the routing protocol must react to changes in the network state, and make decisions based on the properties of the flow. Our results show that RL can learn the desired behavior and requires the exchange of only 16 bits of information.

Keywords: Reinforcement Learning, Game Theory, Routing

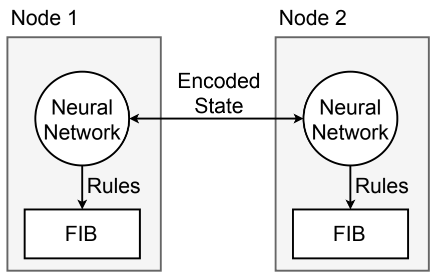
1 Introduction

After proper configuration, communication networks are able to adjust to changes in their state, e.g., to changes in the topology [CRS16]. Communication networks achieve this with a rule-based learning system. Routers exchange information about the network state among each other or a central entity [BSL+18]. A routing protocol-dependent algorithm, e.g., Dijkstra's algorithm [CRS16], computes the Routing Information Base (RIB) from this information [KR16]. The routing protocol derives forwarding rules, i.e., which traffic is forwarded through which port, from the RIB. These rules form the Forward Information Base (FIB) [KR16]. The data-plane matches packets against those rules and derives the applicable action set [AED+14].

This approach suffers from two shortcomings: 1) limited adaptability and 2) limited expressiveness. The process to translate a change in the network into new rules on the FIB has multiple steps and is thus slow to react. It is difficult for this rule-based system to condition forwarding decisions on fast-changing network state attributes such as utilization or queue length [AED+14]. To counteract this, rules for different contingencies could in principle be computed and installed on the devices. However, forwarding rules based on dynamic network state could become complex and intricate and exceed the rule table size. Increasing memory is possible but expensive [SB18]. The restricted space for rules can thus limit their expressiveness and thus the extend to which forwarding decision can be conditioned on network state.



(a) Traditional.



(b) COMNAV.

Figure 1: System diagrams for traditional routing protocols, COMNAV and a comparison of the induced logical control planes. COMNAV integrates the storage of network state, RIB, and calculation of route into a Neural Network.

We propose COMNAV, a system that relies on network programmability, Reinforcement Learning (RL), Neural Networks (NNs) and hardware acceleration to overcome this limitation. The idea of COMNAV is to train a Recurrent Neural Network (RNN) with a binary hidden state that implements a routing protocol. This routing protocol can be tailored to a specific network and its workload during training. Using a binary representation for the hidden state allows concise encodings through a distributed representation of information in the RNN weights [GBB11]. Forwarding devices implement only limited functionality: The execution of the *forward* pass of an RNN that can be implemented directly in hardware [SSB18, SB18].

Fig. 1a and Fig. 1b illustrate the architectural differences of a traditional rule-based system and COMNAV. Network devices in traditional systems have a routing process that computes the rules of the RIB and thus the FIB from the current network state, and configured policies. The network devices exchange local state on the level of the routing processes through well defined messages. In COMNAV, the computation of rules for the FIB, and the storage of network state is implemented in a RNN. Nodes in COMNAV do also exchange messages. Those message are exchanged only with direct neighboring nodes and are not explicitly specified. Instead, the RNN learns how to design those messages during a training phase. This allows the network to exchange exactly the information that is necessary to achieve a specific objective.

To obtain such a protocol, COMNAV takes a flow-centric perspective: we teach flows navigational skills with RL, which allows them to traverse a communication network, thereby exploiting its current state (e.g., current loads, failed links, etc.). In other words, we present a first building block towards a routing that is learned by the network itself, providing the opportunity to let the network react to changing traffic conditions, hardware or topological changes. To this end, we design and train a novel neural architecture that caters for the specific properties of communication networks and routing. The focus on flows introduces a connection to game theory and in particular congestion games [RT02], which we will discuss in more detail.

As a first step, we investigate the feasibility of using an RNN with a binary hidden state to make routing decisions based on the current state on network nodes. Using our proof-of-concept implementation, we show the potential of such an approach in two case studies: learning to evade congested links and learning to navigate flows with different properties. Indeed, we find that RL can be used to learn strategies for different network states and differentiate path selection critera based on observed flow characteristics and network state information.

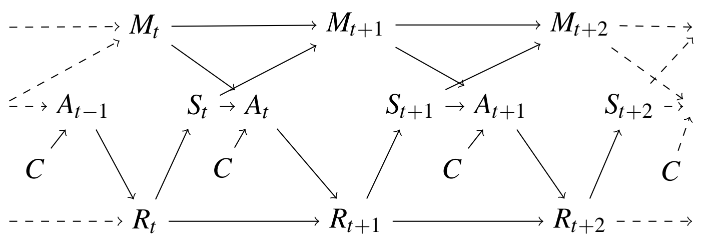
The remainder is organized as follows: We present our vision, identify challenges and present our approach in Sec. 2. Sec. 3 introduces the the environmental model for RL. Sec. 4 discusses the used RL architecture. In Sec. 5 we evaluate our design in two experiments to provide a proof-of-concept for our proposed methodology, Sec. 6 discusses related work, and Sec. 7 concludes this work.

2 Vision: Navigation vs Routing

We envision networks that adapt their routing strategy for traffic that changes over short and long time-scales. In order to achieve this vision, we use RL, which allows an *agent* to evolve its behavior by adapting it to reward signals In our work, we take an unconventional approach as to the choice of our agent. Intuitively, a forwarding device or central entity is referred to as agent [BL94, CY96, VSST17], i.e., the entity would use RL for taking its actions in order to maximize its reward. In contrast to the forwarding device-centered view, we identify with agents the entities that *traverse* the network: a flow, a flowlet, or a packet. We understand routing as a *navigation* problem. In this interpretation, multiple entities traversing the network *actively take actions*, which can be interpreted as a congestion game [RT02].

The choice to take on this view-point is motivated by recent advances in the navigation of artificial agents in complex simulated and real environments [MPV$^+$16, MGM$^+$18]. In robotics, navigational tasks are called Simultaneous Localization and Mapping (SLAM): a robot has to infer its location from sensory information, and creates a map of its environment through exploration, which can then later be used for planning routes [FGS07]. The agent has to associate sensory information of the goal description, e.g., and address or image, and sensory information with a decision like moving forward or turning in order to reach its goal.

Our vision of routing is similar: Packets navigate a network and decide at each switch where to go next, i.e., choose an outgoing port. The decision is based on the state of the network, the agent's previous experiences, and a time invariant context. Fig. 2 illustrates this process. At time t the network is in state R_t. R_t is not fully observable. Instead, the agent obtains sensory information S_t from R_t. Sensory information can include port utilization on the switch the agent is situated on, latency information, or congestion information. The only restriction on S_t is that it can be obtained efficiently to support line rate. The agent then chooses an action A_t based on S_t, its memory M_t and a time invariant context information C. The action A_t corresponds to an outgoing port on the switch. The memory summarizes past experiences of the agent, and allows her to include this information during decision making. For example, the utilization pattern on earlier links can include relevant information for later decisions. The time invariant context C can correspond to IP-Addresses, Quality-of-Service requirements, the volume of the flow, etc.



$$\begin{tikzcd}
{} \arrow[rr, dashed] & & M_t \arrow[rrr] \arrow[drr] & & & M_{t+1} \arrow[rrr] \arrow[drr] & & & M_{t+2} \arrow[rr, dashed] \arrow[drr, dashed] & & {} \\
{} \arrow[r, dashed] & A_{t-1} \arrow[ur, dashed] \arrow[ddr] & & S_t \arrow[r] & A_t \arrow[ur] \arrow[ddr] & & S_{t+1} \arrow[r] & A_{t+1} \arrow[ur] \arrow[ddr] & & S_{t+2} \arrow[r, dashed] & {} \\
& C \arrow[u] & & & C \arrow[u] & & & C \arrow[u] & & & C \arrow[u, dashed] \\
{} \arrow[rr, dashed] & & R_t \arrow[uur] \arrow[rrr] & & & R_{t+1} \arrow[uur] \arrow[rrr] & & & R_{t+2} \arrow[uur] \arrow[rr, dashed] & & {}
\end{tikzcd}$$

Figure 2: Dependencies between different components of a `FlowAgent` agent: M_t corresponds to the memory of the agent represented by a RNN, A_t corresponds to the action taken, R_t corresponds to the state of the environment, i.e., the network, and S_t corresponds to the sensory information the agent obtains from the environment. C corresponds to a time invariant context.

After choosing an action, i.e., a port on the node, the memory of the agent is updated. Thus, the relevant information from the current node is present on the next node. Then, the environment transitions into a new state R_{t+1}, i.e., the agent moves to the next hop. This process continues until the agent reaches its destination.

3 A Navigation Network Model

In the following we introduce the different components of the agent-environment system.

The Environment. We represent a network as a graph $G = (\mathscr{N}, \mathscr{E}, b, m)$ and model it as a queuing network. A node $v \in \mathscr{N}$ corresponds to a forwarding device, and an edge $e \in \mathscr{E}$ to a link between two forwarding devices. Nodes and edges are both modeled as service units. A link $e \in \mathscr{E}$ corresponds to a service unit with one First-In-First-Out (FIFO) queue. A forwarding device $v \in \mathscr{N}$ has multiple queues, one queue for each flow traversing the node. Queues on nodes are served using Fair-Queuing (FQ) with Byte-by-Byte-Round-Robin.

The function $b : \mathscr{N} \cup \mathscr{E} \rightarrow \mathbb{N}$ returns the service rates of nodes and links, and the function $m : \mathscr{N} \cup \mathscr{E} \rightarrow \mathbb{N}$ the buffer. Forwarding devices enqueue packets into the queues of its connected links, and links enqueue packets into the queues of incident forwarding devices. Packets are pushed into the network by sources using deterministic arrival times, i.e., each source places packets at fixed times into the queue of the incident link. An environment state $r \in \mathscr{R}$ is thus associated with a specific forwarding device and contains the states of the queues of the forwarding device, as well as the queues of the outgoing links.

The `FlowAgent`. We consider flow-level routing, and an agent corresponds to a flow to avoid packet re-ordering. Note that flows consist of multiple packets, i.e., we use a packet-level simulation. In our simulation, the first packet of the flow, e.g. the TCP SYN packet, discovers a path from source to destination. The forwarding decisions are cached in the FIB of the nodes, similar to `CONTRA` [HBC+20]. Subsequent packets follow the path of the first packet. A `FlowAgent` has time invariant contextual information, e.g., source and destination addresses, QoS parameters, Type-of-Service, or flow size. The agent utilizes this contextual information for decision making.

Actions. When arriving on a forwarding device, a flow agent decides which link to take. The

forwarding device then enqueues the `FlowAgent` into the buffer of the chosen link. Thus, the number of actions of a `FlowAgent` can differ between nodes, since each node can have a different number of incident links. Thus, the possible actions in each state correspond to outgoing links at the forwarding device associated with that state, i.e., the set of all actions $\mathscr{A}$ corresponds to the set of edges $\mathscr{E}$. At each time step t the agent can choose from a subset $\mathscr{A}_t \subset \mathscr{A}$.
Sensor. The sensor of a `FlowAgent` defines what the `FlowAgent` perceives, i.e., the observable part of the environment. The `FlowAgent` uses information from the sensors to make a decision. A sensor reading $s \in \mathscr{S}$ could include the MAC-address of the switch or time-dependent flow properties such as the Time-to-Live. Also local information at the forwarding devices can be observed, e.g., the number of flows at each port, or the utilization of each port.
Objective. The goal of each `FlowAgent` is the minimization of the Flow Completion Time (FCT). The reward used during the learning of the protocol with RL is designed to achieve this objective. Other objectives, such as minimizing the maximum link utilization, or a combination of different objectives can be used as well. Depending on the requirements even policies such as specific traffic should not traverse a specific link or node could be included.

4 Learning to Navigate Networks

We represent an agent as a RNN which is trained with RL. For training, we use a simulated network with changing state. Each agent obtains a reward that is proportional to the FCT, i.e., the time it takes until the last packet is received by the destination. The agents are exposed to varying demands and should learn which route to take based on the observed state, i.e., based on the source, destination, current location, and the utilization of the ports at the current location.
A game theoretical perspective. We can view the FCT minimization as a congestion game $\Gamma = (\mathscr{P}, \Sigma, u)$, $\mathscr{P}$ being the set of players, i.e., `FlowAgents`, Σ the space of possible strategy profiles, and $u : \Sigma \times \mathscr{P} \rightarrow \mathbb{R}$ a utility function evaluating a strategy profile with respect to a specific player. A strategy for a player is the selected path from a source node s to a destination node d. The utility function is the FCT. Here, a player then consists of the tuple (s, d, v, t), where $s, d \in \mathscr{N}$ are the source and destination node of a flow, v its volume, and t its arrival time.

Now, let $\sigma \in \Sigma$ be a strategy profile, i.e., for each player in $\mathscr{P}$ the strategy vector σ contains the strategy of that player. Further, let $\sigma' := (\sigma^{-i}, \sigma_i')$ be the profile that is created when player $i \in \mathscr{A}$ changes his strategy σ_i to σ_i'. Further, let $P : \Sigma \rightarrow \mathbb{R}$ be a potential function. Here, the potential function is the sum over all FCTs. Then, the game here is a exact potential game since:

$$u(\sigma', i) - u(\sigma, i) = P(\sigma') - P(\sigma). \tag{1}$$

A simple *proof sketch* is as follows: Since the potential function is a sum of individual FCTs, a change in one FCT translates to a change in P [RT02].

Thus, the above game emits at least one pure strategy Nash Equilibrium [Gib92], and the RL approach converges towards one Nash Equilibrium [GLS17]. That is, it learns a mapping from the state to the respective strategy that optimizes its utility, i.e., FCT.
Asynchronous Advantage Actor-Critic (A3C). We use an actor-critic based algorithm, which maintains a policy $\pi(A_t \mid S_t, C, \theta_\pi)$, and an estimate of the value function $V(S_t \mid \theta_v, C)$ [MBM+16].

We decided for an on-policy algorithm without replay memory to account for the non-stationarity of the environment in multi-agent scenarios [GD21]. We use parametric functions in the form of NNs to represent π and V, where θ_π identifies the parameters of the policy, and θ_v the parameters of the value function. The policy maps states to actions, whereas the value function computes the expected reward which can be obtained with a policy from a specific state. In this case, a *state* corresponds to a sensor reading.

We use an asynchronous variant to update the parameters of the functions. That is, multiple agents and environments run in parallel. Parameter updates are applied asynchronously to a global set of parameters, that are then copied to the local agent. The agents can thus continuously adapt to the changed policies of the other agents [GD21].

Reward. Our problem is episodic. One episode corresponds to the delivery of one flow to a destination. In case of the `FlowAgent` the reward is:

$$g_t = \alpha \frac{t_t^* + t_p^*}{t_t + t_p}, \tag{2}$$

where t_t^* is the transmission time on the shortest path based on residual capacity, and t_p^* the propagation delay on the shortest path based on delays calculated during the arrival of the flow. Those paths serve as lower bounds. This information is only required during the training phase in the simulation. The denominator corresponds to the transmission and propagation delay the flow then experienced. Eq. (2) becomes larger the closer the agent gets to the minimum values. Note, that values larger one are also possible in case flows leave the system providing better shortest paths during the life time of the current flow.

In addition to the terminal reward, we use a per-step reward. For the per-step reward, we exploit the relation to routing games and imitate a form of marginal cost pricing [RT02]:

$$g_s = -\frac{1}{N}\sum_{i=1}^{N}\left(t_t^{(i)\prime} - t_t^{(i)}\right). \tag{3}$$

$t_t^{(i)\prime}$ is the transmission time of flow i with the current flow on the same link, while $t_t^{(i)}$ was the previous transmission time. The current flow is charged with the additional time its presence on the link causes the other flows. This mechanism has been shown to reduce the Price-of-Anarchy in congestion games [RT02], and should, therefore, be a good guidance.

NN Architecture. In our architecture, we make use of parameter sharing. That is, θ_π and θ_v do not correspond to different NNs, but instead share large parts of the parameters. Fig. 3 illustrates the architecture we use during our experiments. The network consists of multiple building blocks: Two encoders, a binary recurrent layer, policy layers, a layer for the value function, and an layer to predict loops in the network path.

The purpose of the encoder is to extract a low dimensional representation from the observation S_t and the context C. The encoder output for the observation S_t is then fed into a binary recurrent layer, serving as memory for the agent. For our experiments, a hidden layer with 16 neurons is sufficient. The output of the recurrent layer is binary, i.e., consists of zeroes and ones. This is an important design aspect since the hidden state of the recurrent layer must be transmitted to the next node. Learning a binary representation results in small and concise codes and is a good fit for the distributed representations of information that NNs learn [GBB11]. The encoded context

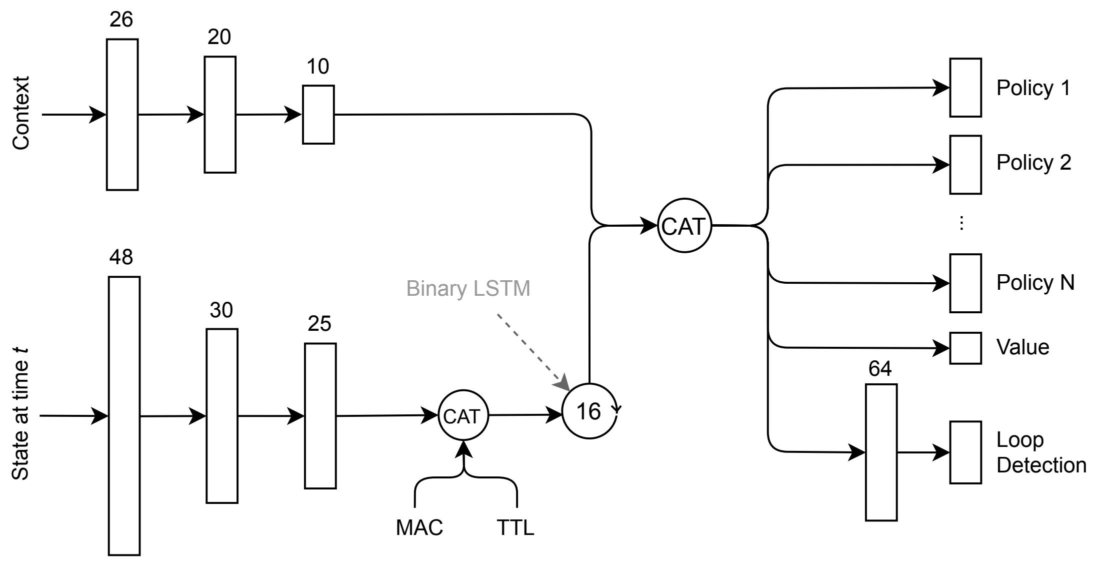


Figure 3: NN architecture. Each switch has its own policy. Except for the policy layer are all weights identical for different switches. The numbers above the layer give their size.

is not fed into the recurrent layer since the context is time invariant. The output from context encoder and recurrent layer are then used for different purposes: To predict the value, actions, and to detect loops. Since the main purpose of encoder and recurrent layer is the extraction of useful information, we share them between policy and value function.

Loop detection is an auxiliar task that is utilized during training. The purpose of the loop detection task is to provide additional training for the recurrent layer and encoders to produce a meaningful representation. If the agent is able to discern that it has visited a certain location already, then the agent discriminated between states and memorized them. Both skills are important for the task of the agent. The auxiliar task provides additional training to the agent, which can improve the data efficiency, i.e., the agent converges faster to a good policy [MPV+16].

Challenges we are facing in the navigation of communication networks is the changing number of actions at each forwarding device and semantical differences between the ”same” action. With a single output layer, the number of actions on each forwarding device must equal the maximum degree in the network. Further, some actions are forbidden on nodes that have a smaller degree. The agent then has to learn which actions are admissible on which location. Also, the ”same” action, e.g., leave trough port zero, must not be related in a meaningful way. For example, choosing the action ”leave through port 2” will not mean that the agent is moving into a similar direction when choosing it at different forwarding devices. In navigational tasks in the real world, the same action (for example turn left) is related between two successive states. Also, forwarding devices are farther apart in communication networks, and information such as the MAC address are not necessarily related. In robotics one usually has a constant visual input stream.

We address these challenges by proposing an architecture featuring multiple policy layers. One layer for each forwarding device. Here we exploit the fact that the agent always knows

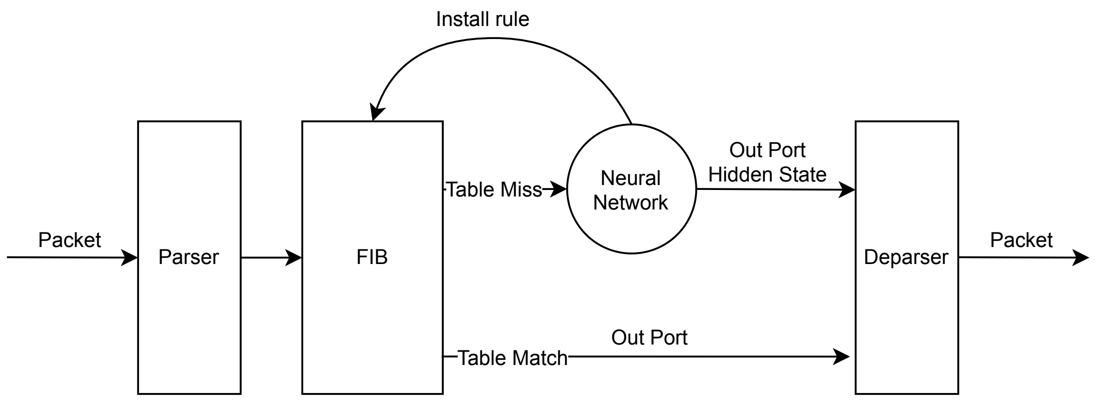


Figure 4: Data plane of a switch in ComNav. Upon a table miss, the NN is queried for an output port to a packet. The output is cached in the FIB and re-used for subsequent packets of the flow.

exactly where it is, which is usually not the case in robotics [MGM+18, MPV+16]. During learning, the weights of the different policy layers can thus be tuned independently, giving credit to the different semantics of actions at different forwarding devices.

Centralized Learning, Distributed Inference. A packet cannot carry a NN. Instead, we propose to distribute the NN *across* the communication network. A `FlowAgent` performs at each time step a forward pass through the encoders and the recurrent layer to the policy and value function, which requires the output of the recurrent layer from the previous time step. We propose to situate the parameters of the NN at the forwarding devices as illustrated in Fig. 4. If an agent, i.e., the first packet of a flow arrives at a node, it first gets parsed. During parsing, relevant information such as the context information, i.e., source and destination addresses, as well as the hidden state of the RNN are extracted. Then, the forwarding device searches for forwarding rules in its FIB. Since no such rule exists, the switch then queries the NN. The NN computes an port through which the packet should be send. The switch installs a corresponding rule in its FIB. Keeping rules for the active set of flows is feasible [HBC+20]. Subsequent packets of the flow will match this rule and the NN is not queried again. In addition, the NN updates the hidden state. The Deparser writes the new hidden state to the packet. Since the output of the recurrent layer is binary, it can be used without further processing. The switch then forwards the packet to the next hop. Note that the network devices *perform only forward passes*, i.e., no backward passes or parameter updates are computed during operation.

While inference can happen in a distributed manner, training must be performed in a central location. Distributed training would incur a very high overhead, since a lot of information would have to be transmitted between nodes. Further, updating the network during operation can lead to unwanted behavior of the network. Instead, we rely on the increasing capability to gather data, e.g., provided by network telemetry [HBC+20], in order to obtain real state information from the network and simulate traffic for learning. In this way, the parameters can continuously be updated, adapted to shifting demands, and most important, can be validated wrtt. to consistency of resulting routes. If the updated weights meet corresponding quality criteria, they can be deployed on the forwarding devices. During training, varying network conditions can be simulated and learned with the neural architecture. In this way, shifting traffic patterns, larger topological changes, or other optimization objectives can be incroported into the protocol. The routing protocol implemented by the resulting NN then reacts to fast-changing network conditions, such as link failures, changes in the utilization of the network, and traffic patterns.

One limitation of using a NN in the data plane is the large amount of inference steps that must be performed to keep line rate. Recent work suggests, that even today's ASICs are capable of executing small NNs at line rate [SB18]. Thus, we believe that per-flow inference is possible, even for larger networks.

5 Evaluation

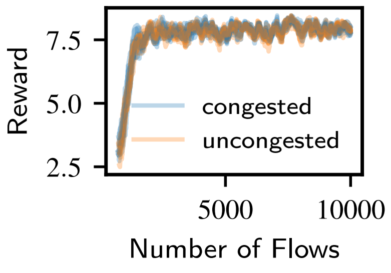
We consider two scenarios as use-cases for our approach: Learning to evade congested links and learning to navigate flows with different properties.
Topology Setting: In both cases, we use the Abilene network [KNF^{+}11] as topology. Fig. 7 shows the Abilene network topology. Based on publicly available data, we added Autonomous Systems (ASes) to the Abilene network from which traffic originates during our evaluation.
Learning Setting: For both set-ups, we use A3C with 16 cores and train the network with 10 000 flows per core. At every location, the observation S_t consists of the utilization of the ports at the forwarding device, and a binary identifier for the in port. Utilization is represented using zero-one-hot encoding based on Fortz et.al [FT02] and passed through an encoder consisting of two fully connected layers with 30 and 25 neurons and Rectified Linear Unit (ReLU) activation. The port identifier is passed to the recurrent layer implemented by a Long-Short-Term-Memory (LSTM) RNN [HS97] with 16 binary hidden neurons. The context C consists of a one-hot-encoding for the source and destination node, which is passed separately through an encoder with 20 and 10 neurons and sigmoidal activation. The two outputs are then concatenated together to a vector with 20 elements. For the policies, we use a linear layer consuming the combined output of context encoder and LSTM with softmax activation. For the value function, we use a linear layer with a single output neuron. Similarly, for the loop-detection, we use a linear layer with single output neuron with a sigmoid activation.

During our experiment, we experienced the problem that the agent quickly converged to a sub-optimal policy. To mitigate this risk, we additionally introduce random actions to increase the exploration and let the agent discover a better strategy.

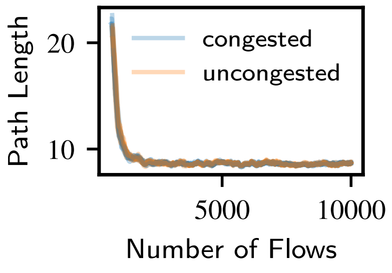
5.1 Learning to Evade

In this scenario, the agent starts at `Nysernet` connected to New-York and has to navigate the network to `Pacific Wave` connected to Seattle. The objective is to minimize the FCT. In the uncongested case, the best FCT is obtained via the path *New-York* - *Chicago* - *Indianapolis* - *Kansas City* - *Denver* - *Seattle*. In the congested case, the flow is slowed down on the link *Indianapolis* - *Kansas City*, and the best path becomes *New-York* - *Washington* - *Atlanta* - *Houston* - *Los-Angeles* - *Sunnyvale* - *Seattle*. We train the agent for both scenarios: 50 % of the trained cases contain congestion on the link *Indianapolis* - *Kansas City*, while the other half does not.

Fig. 5a shows the discounted reward of the agent, and illustrates the learning progress; it shows

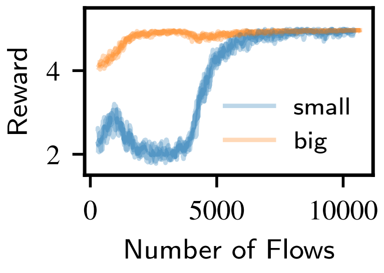


(a) Reward

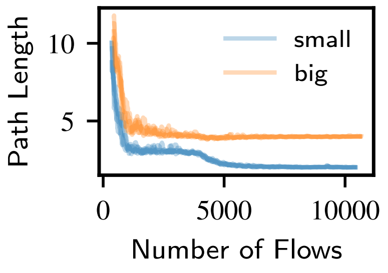


(b) Path Length

Figure 5: Reward for the agent learning to evade a bottleneck link.



(a) Reward



(b) Path Length

Figure 6: Reward for the agent learning to differentiate between different flow patterns based on the source and destination.

that after approximately 2 000 flows the agent has converged to a stable solution, since the reward does not increase anymore. Fig. 5b shows the path length, which drops as the reward increases. The difference between the congested and uncongested case is only marginally as indicated by the overlapping lines. The agent has learned to avoid the link between Indianapolis and Denver when it is congested. The agent thereby chooses the path *New-York* - *Chicago* - *Indianapolis* - *Atlanta* - *Houston* - *Los-Angeles* - *Sunnyvale* - *Seattle* as alternative. This result can be explained by the fact that the agent has no means of knowing at New-York City that the link between Indianapolis and Kansas City is congested. In the absence of additional information, the best choice is indeed to first travel to Indianapolis and then decide based on the conditions there. The agent is very confident in making that decision. In the congested case, the agent chooses with probability one to travel to Atlanta; whereas for the uncongested case the agent chooses to travel to Kansas City with 87 % probability, and to Atlanta with 12 % probability. Those numbers do not change when the agent has more trials.

5.2 Learning to Differentiate

In this scenario, two flows start from the `Oregon Gigaport` connected to Sunnyvale and `Pacific Wave 2` connected to Los-Angeles towards `Pacific Wave` connected to Seattle.

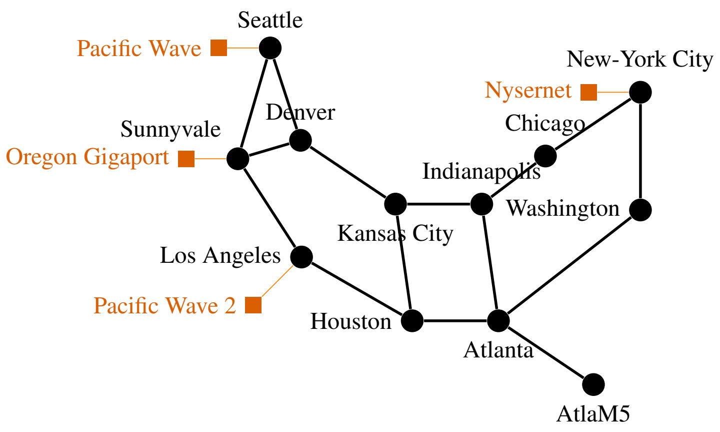


Figure 7: Abilene topology. with additionally attached Autonomous Systems.

We now assume different applications behind the flows. Flows originating at `Pacific Wave 2` are assumed to be mostly larger downloads, whereas flows originating from the `Oregon Gigaport` are small flows. We increase the utilization on the link between Sunnyvale and Seattle, such that the larger flows from `Pacific Wave` experience significant transmission delay, while the flows from the `Oregon Gigaport` are small enough to experience no significant delay. Thus, flows from `Pacific Wave 2` should take a detour over Denver in order to obtain a shorter FCT.

Fig. 6a illustrates the learning progress of the A3C agents. The reward converges after approximately 5 000 flows. Fig 6b illustrates how the agent finds the shorter path for the smaller, and the longer path for the larger agent. Interestingly, the larger agent learned to take the correct path after only 2 000 flows, while the short flow took longer to converge. After the training ended, the two agents are very confident, taking the correct path with probability one. Thus, the agent learned to associate certain sources with specific flow properties and exploited this knowledge when navigating the network

6 Related Work

6.1 Routing and Traffic Engineering

There exist many routing and traffic engineering (TE) solutions that address various challenges and cope with different problems [NR17]. Solutions range from using well-established protocols such as OSPF [CRS16] over new centralized designs [ARR+10, BAAZ11, ZCY+16, HKM+13, JKM+13, KMSB14, LSYR11, LVC+16, ZCB+15] to distributed protocols [KKSB07, MT14, AED+14] that can also utilize data plane programmability [HBC+20].

In contrast, ComNavis a routing system in which relevant parts of the protocol are learned and implemented with a NN. ComNavlearns forwarding rules autonomously with RL. ComNav relies on the flexibility of data-plane programmability to realize the NN in the data plane.

6.2 ML and Networking

Combinations of ML and networking have been explored from different angles [BSL$^+$18]. Beside applications like traffic prediction [CWG16] and classification [EMAW07, EAM06, RRS20], also optimization of reconfigurable topologies [WCX$^+$18, SSC$^+$18], queue management [SZ07, ZDCG09, CLCL18] and implementation of routing schemes have been addressed [Mam19, GC18, XMW$^+$20, YLX$^+$20, BL94, VSST17, CY96]. In particular for routing schemes, there are several approaches considering centralized as well as distributed implementations. Our work differs from previous work in the flow-centric perspective, the resulting formulation of the underlying problem as a congestion game, and the explicit design of the neural architecture with binary hidden layers and multi-head policies. Previous work, especially [YLX$^+$20] relies on large, real-valued vectors that need to be exchanged between nodes, which is impractical in practice. The same is true for the online learning of NN weights. In contrast, ComNav learns the weights during a training phase and exchanges only 16 bits of information between nodes.

6.3 Navigation

Work in [MPV$^+$16] considers the navigation of complex mazes with rich structure using an end-to-end deep RL based approach that learns directly from pixels. In a later work, [MGM$^+$18] design a deep learning based system that is able to reliably navigate in large cities over long distances. They also showed that the learned skills can be transferred to other cities.

In our work, we consider the navigation of communication networks which are quite different from real world navigational tasks. In communication networks, the agent cannot carry its intelligence by itself, also, actions have a completely different semantic between forwarding devices.

7 Conclusion

We presented ComNav, a novel routing architecture that is motivated by programmable dataplanes and machine learning. In ComNav, artificial `FlowAgents` successfully learn to traverse a communication network, accounting for the state at nodes and traffic characteristics to decide their path. `FlowAgents` query binary Recurrent Neural Networks situated at forwarding devices for their next hop. The hidden state of the RNN is piggy-backed on packets, allowing them to memorize their route as well as encountered network conditions. We find that 16 bits are enough to successfully navigate a network. We showcase with two examples the ability of flows, i.e., agents to learn, solely based on information locally available at nodes, which path to take. This limits the action space of the agents and allows distributed inference by placing a forward model at each forwarding device.

We understand our work as a first step and believe that our vision opens many interesting directions for future research. For example, it would be interesting to see whether skills obtained on one topology can be transferred to another one. Another avenue of research could be to further explore and exploit the connection between our approach and game theory (and its formal guarantees). Finally, the performance of the proposed approach could be evaluated in a testbed to explore its limits.

Bibliography

[AED$^+$14] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, G. Varghese. CONGA: Distributed Congestion-aware Load Balancing for Datacenters. *SIGCOMM Comput. Commun. Rev.* 44(4):503–514, Aug. 2014.
doi:10.1145/2740070.2626316

[ARR$^+$10] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *NSDI 2010*. Pp. 281–296. San Jose, CA, USA, 2010.

[BAAZ11] T. Benson, A. Anand, A. Akella, M. Zhang. MicroTE: Fine Grained Traffic Engineering for Data Centers. In *CoNEXT '11*. Pp. 8:1–8:12. ACM, New York, NY, USA, 2011.
doi:10.1145/2079296.2079304

[BL94] J. A. Boyan, M. L. Littman. Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach. In *NIPS 6*. Pp. 671–678. Morgan Kaufmann, 1994.

[BSL$^+$18] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, O. M. Caicedo. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *JISA* 9(1):1–99, 2018.

[CLCL18] L. Chen, J. Lingys, K. Chen, F. Liu. Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization. In *SIGCOM' 18*. Pp. 191–205. Budapest, Hungary, 2018.

[CRS16] M. Chiesa, G. Rétvári, M. Schapira. Lying your way to better traffic engineering. In *CoNEXT' 16*. Pp. 391–398. 2016.

[CWG16] Z. Chen, J. Wen, Y. Geng. Predicting future traffic using hidden markov models. In *ICNP*. Pp. 1–6. 2016.

[CY96] S. Choi, D.-Y. Yeung. Predictive q-routing: A memory-based reinforcement learning approach to adaptive tra c control. *NIPS* 8:945–951, 1996.

[EAM06] J. Erman, M. Arlitt, A. Mahanti. Traffic classification using clustering algorithms. In *SC2D' 06*. Pp. 281–286. 2006.

[EMAW07] J. Erman, A. Mahanti, M. Arlitt, C. Williamson. Identifying and discriminating between web and peer-to-peer traffic in the network core. In *WWW*. Pp. 883–892. 2007.

[FGS07] A. Förster, A. Graves, J. Schmidhuber. RNN-based Learning of Compact Maps for Efficient Robot Localization. In *ESANN*. 2007.

[FT02] B. Fortz, M. Thorup. Optimizing OSPF/IS-IS weights in a changing world. *IEEE J. Sel. Areas Commun.* 20(4):756–767, May 2002.
doi:10.1109/JSAC.2002.1003042

[GBB11] X. Glorot, A. Bordes, Y. Bengio. Deep Sparse Rectifier Neural Networks. In Gordon et al. (eds.), *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Proceedings of Machine Learning Research 15, pp. 315–323. JMLR Workshop and Conference Proceedings, Fort Lauderdale, FL, USA, 11–13 Apr 2011.
http://proceedings.mlr.press/v15/glorot11a.html

[GC18] F. Geyer, G. Carle. Learning and Generating Distributed Routing Protocols Using Graph-Based Deep Learning. In *Big-DAMA '18*. Pp. 40–45. ACM, Budapest, Hungary, 2018. Export Key: geyer2018.
doi:10.1145/3229607.3229610

[GD21] S. Gronauer, K. Diepold. Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review*, Apr. 2021.
doi:10.1007/s10462-021-09996-w

[Gib92] R. Gibbons. *A primer in game theory.* Pearson Academic, 1992.

[GLS17] A. Greenwald, J. Li, E. Sodomka. Solving for Best Responses and Equilibria in Extensive-Form Games with Reinforcement Learning Methods. In Başkent et al. (eds.), *Rohit Parikh on Logic, Language and Society*. Pp. 185–226. Springer International Publishing, Cham, 2017.
doi:10.1007/978-3-319-47843-2_11

[HBC$^+$20] K.-F. Hsu, R. Beckett, A. Chen, J. Rexford, D. Walker. Contra: A programmable system for performance-aware routing. In *NSDI 20*. Pp. 701–721. 2020.

[HKM$^+$13] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, R. Wattenhofer. Achieving High Utilization with Software-driven WAN. In *SIGCOMM '13*. Pp. 15–26. ACM, New York, NY, USA, 2013.
doi:10.1145/2486001.2486012

[HS97] S. Hochreiter, J. Schmidhuber. Long Short-Term Memory. *Neural Comput.* 9(8):1735–1780, Nov. 1997.
doi:10.1162/neco.1997.9.8.1735

[JKM$^+$13] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, A. Vahdat. B4: Experience with a Globally-deployed Software Defined Wan. In *SIGCOMM '13*. Pp. 3–14. ACM, New York, NY, USA, 2013.
doi:10.1145/2486001.2486019

[KKSB07] S. Kandula, D. Katabi, S. Sinha, A. Berger. Dynamic load balancing without packet reordering. *ACM SIGCOMM Comp. Com. Rev.* 37(2):51–62, 2007.

[KMSB14] S. Kandula, I. Menache, R. Schwartz, S. R. Babbula. Calendaring for Wide Area Networks. In *SIGCOMM '14*. Pp. 515–526. ACM, New York, NY, USA, 2014. doi:10.1145/2619239.2626336

[KNF+11] S. Knight, H. Nguyen, N. Falkner, R. Bowden, M. Roughan. The Internet Topology Zoo. *J-SAC* 29(9):1765 –1775, october 2011. doi:10.1109/JSAC.2011.111002

[KR16] J. F. Kurose, K. W. Ross. *Computer Networking: A Top-Down Approach*. Pearson, Boston, MA, 7 edition, 2016.

[LSYR11] N. Laoutaris, M. Sirivianos, X. Yang, P. Rodriguez. Inter-datacenter Bulk Transfers with Netstitcher. In *SIGCOMM '11*. Pp. 74–85. ACM, New York, NY, USA, 2011. doi:10.1145/2018436.2018446

[LVC+16] H. H. Liu, R. Viswanathan, M. Calder, A. Akella, R. Mahajan, J. Padhye, M. Zhang. Efficiently Delivering Online Services over Integrated Infrastructure. In *NSDI 16*. Pp. 77–90. USENIX Association, Santa Clara, CA, 2016.

[Mam19] Z. Mammeri. Reinforcement learning based routing in networks: Review and classification of approaches. *IEEE Access* 7:55916–55950, 2019.

[MBM+16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. *arXiv preprint arXiv:1602.01783*, 2016. http://arxiv.org/abs/1602.01783

[MGM+18] P. Mirowski, M. K. Grimes, M. Malinowski, K. M. Hermann, K. Anderson, D. Teplyashin, K. Simonyan, K. Kavukcuoglu, A. Zisserman, R. Hadsell. Learning to Navigate in Cities Without a Map. *ArXiv e-prints*, Mar. 2018.

[MPV+16] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, R. Hadsell. Learning to Navigate in Complex Environments. *CoRR* abs/1611.03673, 2016. http://arxiv.org/abs/1611.03673

[MT14] N. Michael, A. Tang. Halo: Hop-by-hop adaptive link-state optimal routing. *IEEE/ACM Transactions on Networking* 23(6):1862–1875, 2014.

[NR17] M. Noormohammadpour, C. S. Raghavendra. Datacenter traffic control: Understanding techniques and tradeoffs. *IEEE Communications Surveys & Tutorials* 20(2):1492–1525, 2017.

[RRS20] A. Rashelbach, O. Rottenstreich, M. Silberstein. A Computational Approach to Packet Classification. In *SIGCOMM' 20*. Pp. 542–556. 2020.

[RT02] T. Roughgarden, E. Tardos. How Bad is Selfish Routing? *J. ACM* 49(2):236–259, Mar. 2002. doi:10.1145/506147.506153

[SB18] G. Siracusano, R. Bifulco. In-network Neural Networks. *CoRR* abs/1801.05731, 2018. Export Key: siracusano2018.
http://arxiv.org/abs/1801.05731

[SSB18] D. Sanvito, G. Siracusano, R. Bifulco. Can the Network Be the AI Accelerator? In *Proceedings of the 2018 Morning Workshop on In-Network Computing*. NetCompute '18, pp. 20–25. Association for Computing Machinery, New York, NY, USA, 2018. event-place: Budapest, Hungary Export Key: sanvito2018.
doi:10.1145/3229591.3229594
https://doi.org/10.1145/3229591.3229594

[SSC+18] S. Salman, C. Streiffer, H. Chen, T. Benson, A. Kadav. DeepConf: Automating data center network topologies management with machine learning. In *NetAI*. Pp. 8–14. 2018.

[SZ07] J. Sun, M. Zukerman. An adaptive neuron AQM for a stable internet. In *NETWORKING*. Pp. 844–854. 2007.

[The12] The Open Networking Foundation. OpenFlow Switch Specification. Jun. 2012.

[VSST17] A. Valadarsky, M. Schapira, D. Shahaf, A. Tamar. A Machine Learning Approach to Routing. *CoRR* abs/1708.03074, 2017.
http://arxiv.org/abs/1708.03074

[WCX+18] M. Wang, Y. Cui, S. Xiao, X. Wang, D. Yang, K. Chen, J. Zhu. Neural network meets DCN: Traffic-driven topology adaptation with deep learning. *POMACS* 2(2):1–25, 2018.

[XMW+20] S. Xiao, H. Mao, B. Wu, W. Liu, F. Li. Neural Packet Routing. In *NetAI '20*. Pp. 28–34. ACM, Virtual Event, USA, 2020.
doi:10.1145/3405671.3405813

[YLX+20] X. You, X. Li, Y. Xu, H. Feng, J. Zhao, H. Yan. Toward Packet Routing With Fully Distributed Multiagent Deep Reinforcement Learning. *IEEE TNSM*, 2020.

[ZCB+15] H. Zhang, K. Chen, W. Bai, D. Han, C. Tian, H. Wang, H. Guan, M. Zhang. Guaranteeing Deadlines for Inter-datacenter Transfers. In *EuroSys '15*. Pp. 20:1–20:14. ACM, New York, NY, USA, 2015.
doi:10.1145/2741948.2741957

[ZCY+16] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, Y. Geng. CODA: Toward Automatically Identifying and Scheduling Coflows in the Dark. In *SIGCOMM '16*. Pp. 160–173. ACM, New York, NY, USA, 2016.
doi:10.1145/2934872.2934880

[ZDCG09] C. Zhou, D. Di, Q. Chen, J. Guo. An adaptive AQM algorithm based on neuron reinforcement learning. In *IEEE ICCA*. Pp. 1342–1346. 2009.