Automating Network Operation Centers with Superhuman Performance

Sa'di Altamimi, *Member, IEEE*, Basel Altamimi, David Côté, and Shervin Shirmohammadi, *Fellow, IEEE*

Abstract—Today's Network Operation Centres (NOC) consist of teams of network professionals responsible for monitoring and taking actions for their network's health. Most of these NOC actions are relatively complex and executed manually; only the simplest tasks can be automated with rules-based software. But today's networks are getting larger and more complex. Therefore, deciding what action to take in the face of non-trivial problems has essentially become an art that depends on collective human intelligence of NOC technicians, specialized support teams organized by technology domains, and vendors' technical support. This model is getting increasingly expensive and inefficient, and the automation of all or at least some NOC tasks is now considered a desirable step towards autonomous and self-healing networks. In this article, we investigate whether such decisions can be taken by Artificial Intelligence instead of collective human intelligence, specifically by the Machine Learning method of Reinforcement Learning (RL), which has been shown in computer games to outperform humans. We build an Action Recommendation Engine (ARE) based on RL, train it with expert rules or by letting it explore outcomes by itself, and show that it can learn new and more efficient strategies that outperform expert rules designed by humans. ARE can be used in face of network problems to either quickly recommend actions to NOC technicians or autonomously take actions for fast recovery.

Note to Practitioners: *Abstract*—In the industry, a Network Operation Center (NOC) manages a large-scale network to ensure the network is operating efficiently. If anything goes wrong in the network, it is the job of the human operators in the NOC to fix the problem as soon as possible. Today, the NOC operators use predefined expert rules to take remedial actions when something goes wrong. These expert rules are designed from past experience, and are improved as operators get more experience with the network. But the larger the network, the less efficient and more difficult-to-design the expert rules, due to the complexity of large networks. The work in this article was motivated by this complexity, and is an attempt to automate an NOC's remedial actions. We design an action recommendation engine that uses machine learning and figures out by itself the relationship between the network's raw data and remedial actions. Because the engine can discover relationships that are too complex for humans to figure out, the engine outperforms expert rules and archives superhuman performance. To achieve superhuman performance, we use the machine learning method of Reinforcement Learning (RL), which has achieved superhuman performance in other domains. But in our case RL has two practical problems: it takes a long time to train it to achieve good performance, and during that training it makes mistakes, which can be costly to make in a real network. Therefore, in addition to proposing our RL model, we also solve these two practical problems, allowing the model to be trained offline in a matter of minutes. Although our model

Fig. 1. A typical Network Operation Center (NOC). The Action Recommendation Engine (ARE) is proposed by our work. Figure reused from [3] with permission.

can be used to fully automate an NOC, its other practical usage is as a recommendation engine that recommends an action to the human operator, still leaving the final decision in the hands of people and not machines.

Index Terms—Primary Topic keywords: autonomous networks, network operation center automation, Secondary Topic Keywords: superhuman performance, computer networks, reinforcement learning

I. INTRODUCTION

THE Internet now serves 9 billion clients world-wide and consists of a large number of interconnected networks, users, sensors and devices sending petabytes of data through the network every millisecond. The task of ensuring the efficient operation of the network typically lies with the Network Operation Centre (NOC), consisting of teams of network professionals responsible for monitoring and taking actions for their network's health. As we can see in Figure 1, a typical NOC collects performance monitoring and alarm data, sometimes logged as tickets. When a problem is identified, the NOC technicians analyze the situation and come up with a suitable action that resolves the problem, normally also taking into account the constraint of the ISP's operational costs (OPEX).

Today's NOCs perform their tasks either manually [1] or, for the simplest tasks, with pre-determined expert rules [2]. Essentially, a team of NOC technicians works 24/7 to ensure

S. Altamimi, B. Altamimi, and S. Shirmohammadi are with University of Ottawa, Canada. (e-mail: syaltamimi@gmail.com; bs.tamimi@gmail.com; shervin@ieee.org)

David Côté is with Blue Planet Analytics, Ciena Corp. (e-mail: dcote@ciena.com)

Manuscript received August 15, 2021; revised N/A

the proper operation of the network. For non-trivial issues, the NOC team requires the aid of technology- and domain-specialized support teams who in turn rely on vendors' technical support services. So far, operating in such a pyramidal and manual model has worked, and the totality of this team eventually resolves problems by taking proper remediation actions. However, this model is getting increasingly expensive and inefficient as networks become larger and more complex. The automation of all or at least some NOC tasks can be a huge contribution and a big step towards realizing autonomous and self-healing networks. However, automation with explicit rules have generally not been very successful because 1) there is an intrinsic difficulty in defining expert rules that robustly work in complex and dynamic networks, and 2) no single expert knows all the rules for different network technologies; i.e., deciding what action to take is based on collective human intelligence, not a single human's intelligence.

In our previous work [3], we hypothesized that such decisions can be taken by Artificial Intelligence (AI), specifically Machine Learning (ML), because there are logical relationships between network problems and their remediation actions, even if those relationships are difficult to codify for complex networks. ML is therefore a good tool to model those relationships and take actions autonomously. The use of ML is further justified by the fact that the industry is already moving towards ML tools for network analytics and state detection, such as Ciena's Blue Planet Unified Assurance and Analytics¹, or Blue Planet Route Optimization and Analysis², which help the NOC technicians gain deeper insights into the network to make intelligent data-driven decisions that lead to improved efficiency, lowered costs, and providing more personalized services. It therefore follows naturally to leverage ML one step further and help NOCs in automating much of their decision making too.

In [3] we showed empirically that an ML-based automation system for NOCs indeed has a comparable performance with human expert rules. This is shown in Figure 2, which depicts the performance of various NOC methods in a network over time. Global QoE-OPEX is the difference between user's Quality of Experience (the higher the better for the user) and OPEX (the lower the better for the network operator). ARE is an ML-based automated Action Recommendation Engine, tested in its Decision Tree (DT), Gradient Boosting (GB), and Extreme Gradient Boosting (XGB) incarnations. NOC mimic refers to human-defined expert rules, static network refers to not taking any action, and Anchor method refers to expert rules but without taking OPEX into account. As we can see from the figure, the ML-based ARE-XGB has a similar performance as the expert rules, demonstrating the feasibility of NOC automation.

While the results in [3] were highly encouraging, they also raise the question of whether such ML-based automation systems have the potential to outperform the human-defined expert rules, leading to superhuman performance. This great promise of AI has been demonstrated for games already [4],

Fig. 2. Performance comparison between ML-based models (ARE) and human defined expert rules (NOC-mimic). Figure reused with permission from [3].

and we speculate that a growing number of such tasks will be established in a variety of domains.

In this work, we investigate the above question, and we propose the first automated NOC system with superhuman performance. To this end, we propose an OPEX-aware Action Recommendation Engine (ARE), shown as such in Figure 1, using Reinforcement Learning (RL), a type of ML method, and we show that indeed ARE is able to outperform humans by learning new effective rules on its own. ARE uses raw data, tickets, and feedback from its previously recommended actions to either recommend actions to the NOC technicians, shown in the figure with a blue arrow labeled Action recommendation, or directly apply the recommended action in a human-out-of-the-loop fashion, shown in the figure with a blue arrow labeled Autonomous action. In the context of automation, this superhuman performance will lead to an automation system with unprecedented efficiency and optimality in NOCs: efficiency because ARE makes decisions much faster than humans, and optimality because ARE's decisions will lead to much better results, in terms of cost saving and user's quality of experience, compared to human decisions.

The rest of this paper is organized as follows: in the next section we cover related work and show our contributions compared to existing literature, while in section III we present the detailed design of our system. Section IV describes our experiments, followed by result and their analysis in Section V. Finally, the paper is concluded in Section VI.

II. RELATED WORK

Current state-of-the-art either use handcrafted expert rules for recovery in case of network problems, or use ML to detect network problems but do not recommend actions. For example, the work in [5] uses handcrafted rules to achieve fast recovery for OpenFlow networks by actively using backup and primary paths before and after failures to achieve high utilization. A similar approach is proposed in [6] where recovery paths are adaptively updated based on the current load state of the network. By providing the backup forwarding rules in advance,

¹<https://www.blueplanet.com/products/uuaa.html>

²<https://www.blueplanet.com/products/route-optimization.html>

fast recovery is achieved upon a failure. Similarly, the work in [7] adopts segment routing in order to reduce the number of forwarding rules and deal with link failure problem in SDNs. This was done by regarding the affected flows through the same link as an aggregated flow and establish a backup path for the aggregated flow. While the above methods can be successful, the main problems with expert rules are its complexity and its suboptimality as the network scales.

One of the earliest works that uses ML to localize network faults is [8] which detects and localizes network problems leading to Quality of Experience (QoE) degradations. It does so by placing measurement probes at multiple vantage points along the path and training a supervised ML model on a combination of synthetic and in-the-wild measurements, leading to more than 80% diagnosis accuracy in the wild. Using video QoE measured at the client side as the only input, the work in [9] detects and localizes network faults in server side, ISP, and client side with up to 97% accuracy. It achieves that by utilizing an artificial neural network trained with a dataset of actual video streaming traces. The work in [10] also uses ML to detect the network status as normal, congestion, and network fault, as well as localize the fault with accuracies of up to 99%. But, as interesting as these methods are, they only detect the fault and do not take autonomous actions to fix it.

In fact, we found only 2 other works that have attempted to automate NOC operations. The earliest work is [11] which uses raw measurement data and logs to reveal “symptoms”; i.e., degradation conditions, and then applies ML to learn a mapping between these symptoms and remedial actions. It uses 3 months of data collected every 15 minutes from 1800 node, with each sample having 1100 feature, so it has the advantage of being wide-scale. On the negative side, the only action it recommends is whether or not to reboot an interface, it only achieves an accuracy of about 40% compared to expert rules, training requires extensive computing resources and time, and any reconfiguration in the network requires retraining because the model has no memory. Another work in NOC automation is [3] which was already discussed in Figure 2 and can at best match expert rules. It should also be mentioned that there is much research in using ML for network routing [12] [13] [14], but network routing and NOC operations are not at all the same, and the former is outside the scope of this paper.

To the best of our knowledge, this work is the first to explore the feasibility of automating an NOC with super-human performance. As such, our proposed ARE not only saves both time and money for NOC tasks but also achieves unprecedented remedial performance, taking us one step closer towards next generation autonomous networks. Compared to [3] which matches the performance of expert rules, our method outperforms expert rules, and compared to [11], our method achieves superhuman performance, has memory and is hence retrained much easier, plans ahead for 100 time steps, and recommends a variety of actions compared to [11]’s only action of interface rebooting. Our contributions can be summarized as follows:

- we design an RL model that can autonomously self-drive a lab network from raw data, and we demonstrate that

the model can learn new implicit and non-trivial rules on its own, exceeding human performance with expert rules;

- since it can take a long time to train an RL model to achieve good performance, for practicality we design a training solution with simulation that can train the RL model in a matter of minutes compared to thousands of hours if trained in real time;
- since an RL model during training can make mistakes that are costly if performed in a real network, we design a training solution that takes in field-collected data and trains the RL model offline, avoiding the said mistakes from happening in an actual network; once the model achieves the desired performance, it can be deployed in the real network.

III. SYSTEM DESIGN

A. Testbed Setup

Figure 3 is a screenshot of our testbed running in GNS3³. The network topology of the testbed must be neither too simple such that it doesn’t sufficiently reflect the real world nor too complicated such that it’s difficult to determine the aforementioned expert rules, which are needed to compare the performance of the RL model against that of the expert rules. The topology shown in the figure is a compromise between the two, and consists of 3 Autonomous Systems (AS) representing consumer-side and its ISP (AS1), an external ISP (AS3) which is used as backup in case of failure, and the service provider (AS2), which in this case is a video service provider like a Netflix or YouTube. We chose video because of two reasons. First, video is by far the dominant IP traffic, expected to constitute 82% of all IP traffic by 2022 [15]. Second, video traffic is inelastic and bulky with a sigmoidal QoE function, making it very challenging to optimize users’ QoE [16], so it represents the worst-case scenario. In our testbed, video consumers (clients), indicated in the figure with play buttons, stream video content from the provider’s cloud through the network. We used OSPF as the underlying routing protocol and IP/MPLS to define tunnels (paths) connecting clients to the service provider. Clients can randomly connect/disconnect from the network. A new client is assigned to a random path and the whole traffic that passes through a path is considered as an aggregated flow. During a streaming session, we consider two types of problems that can negatively affect the clients’ QoE: 1) failure in a router or a link and 2) congestion.

B. Input Metrics

In addition to the number of clients, we collect two types of metrics inside AS1: QoS (E2E) which includes end-to-end IPSLA metrics for each path such as delay, jitter, and packet loss, and QoE which includes video-related metrics such as client’s bitrate, buffer, and download-time. These metrics are later aggregated per path and averaged over a fixed time-interval before being fed to ARE. Averaging is important because, in practice, different metrics are reported at different rates. We also collected per-port metrics such as port packet

³<https://www.gns3.com>

Fig. 3. Our system with its network topology.

loss, but we found that we didn't need them, and our RL worked fine with E2E QoS & QoE metrics only.

It is worth noting that while the traffic aggregation step is performed per path to mimic a real world scenario, this poses a real challenge to the RL agent. In particular, we calculate the reward signal based on QoE metrics from individual clients. This information is not available to the RL agent. For example, assume that the agent receives a reward of +5 for every client stream at high quality (say at bitrate > 3000 Kbps). When the agent observes an aggregated traffic of 7000 Kbps from 2 clients, it can't easily tell whether it will receive +10 (if both were streaming at 3500Kbps) or +5 (if one was streaming at 6000 Kbps while the other at 1000Kbps)

C. The Design of ARE

To understand the design of ARE, readers can benefit from some basic understanding of RL. To readers without this understanding, we suggest to first read an introductory tutorial on RL, such as [17].

1) *RL Basics*: In response to network problems, ARE must suggest/take actions to achieve and maintain an optimal goal set by the operator. In our case, the goal is to maximize clients' QoE and minimize the ISP's OPEX. We formulate this problem as a decentralized and partially observable Markov Decision Process (MDP) and solve it using RL. We chose RL for 3 reasons: its recent achievements in reaching superhuman performance in computer games, the difficulty of labeling data (needed by other ML methods, say supervised learning) in a complex network because the best actions are not always known, and because we can satisfy the preconditions of using RL – quantifying a reward function, accessing the environment's variables, and affording RL making mistakes as it's exploring. The process is specified by the RL components $\langle S, \mathcal{O}, \mathcal{A}, \mathcal{T}_s, \mathcal{T}_o, \mathcal{R}, \gamma \rangle$ defined as follows:

States S and State Transitions $\mathcal{T}_s$: our states are set to be the unobservable condition of each link and router within the network, described as *normal*, *congested link*, *broken link*, or *broken router*. The states represent the ground-truth of the root-cause issues in the network. We use the state to

both calculate the rewards (see §III-B) and as input to the expert rules to help the expert rules fairly compete against the RL agent (see IV-B). $\mathcal{T}_s$ is the state transition function that provides $P(s'|s, a)$, the probability of transition to a next state s' given that the agent starts at state s and takes action a . The Simulator Environment is designed around the idea of trying to mimic this function (see §III-C2).

Observations $\mathcal{O}$ and Observation Function $\mathcal{T}_o$: the observations are the set of metrics available to the RL agent to infer the unobserved state of the environment (see §III-B). $\mathcal{T}_o$ specifies the probability $P(o'|s', a)$ that the agent will receive an observation o' of state s' after reaching this state through an action a . We use this function to perform the data augmentation step in the Simulator Environment (see §III-C2).

Actions $\mathcal{A}$: The action taken by an agent at a given time step t is: *Do Nothing*, *Fix a Link*, and *Reroute* traffic. For the latter, the source and destination paths are specified by the agent. However, we don't let the agent specify which client from the source path to reroute; we simply select a client by random.

Reward $\mathcal{R}$ and discount factor γ : For OPEX, we add three main expenses: the cost of performing the action, the hop-count cost of carrying traffic within the clients' ISP (AS1), and the cost of carrying traffic through the external backup ISP (AS3) which is more expensive because the backup ISP will charge extra. We also map the collected QoE to a reward using the function $\Phi(\cdot)$ which assign +5, 0, and -5 for high, medium, and low QoE, respectively. Therefore, the gain metric $\mathcal{G}$ (which we will call QoE-OPEX) to be maximized can be calculated as:

$$\mathcal{G} = \Phi(QoE) - OPEX \quad (1)$$

We adopt the well-accepted QoE model in [18] for adaptive video streaming. The QoE is defined as

$$\begin{aligned} QoE = & \alpha \sum_k \frac{q_k}{K} - \beta \sum_k |q_{k+1} - q_k| \\ & - \gamma \sum_k \left(\frac{d(q_k)}{C_k} - B_k \right)^+ - \delta \end{aligned} \quad (2)$$

Where q_k is bitrate of k^{th} segment and $d(q_k)$ is its size, B_k is the current buffer level. To define low, medium and high QoE, we normalize the QoE between $[0, 5]$ to mimic 5-star rating and consider 1-2 stars as low QoE, 3 stars as medium QoE, and 4-5 stars as high QoE.

The expected cumulative discounted reward can be calculated over the horizon h as follows:

$$R = \mathbb{E}[\sum_{i=1}^h \gamma^i \mathcal{G}] \quad (3)$$

where h specifies how far into the future the agent is trying to forecast. For example, using $h = 100$ and $\gamma = e^{-1/h} = 0.99$ means the actions taken by the RL agent will be influenced by 100 future steps. On the one hand, h should be large enough to provide a foresighted optimization, as opposed to a myopic one. On the other hand, it should be small enough for the problem to remain computationally solvable.

2) *Training and Testing Environments*: We built three custom OpenAI Gym environments [19] to train and test our RL agent: GNS3 Environment, Simulator Environment, and Batch-RL Environment. A summary of the three environments is listed in Table I.

GNS3 Env. In the GNS3 Environment, traffic generation and network problems were generated by randomly breaking links resulting in three types of problems: *persistent*, *transient*, and *recurrent*. We define a persistent problem to last for 15 time steps, and we expect the agent to have fixed it by then. Transient problems come and go in 3 time steps, and the agent can ignore them in some cases. Recurrent problems repeat every 100 steps, and we expect the agent to predict them before happening in the future. For traffic generation, we defined a sinusoidal pattern where clients start in random paths. Then for most of the time we have 3 to 4 clients who stream videos and select bitrates independently using DASH⁴. Finally, clients turn off and the cycle repeats.

In order to train the RL agent in GNS3 environment, each $\langle s, a, r, s' \rangle$ step requires around 30 seconds to complete, so the size of our time step is 30s. While this environment worked well, it is not really practical because of 2 reasons. First, the agent's training might need hundreds of thousands of steps, so the training will take a long time if done in real time. Second and more importantly, RL learns based on trial and error, and in a live network we cannot afford to make errors. To deal with those two issues, we designed the other two Gym environments: the Simulator Environment which runs much faster than real time, and the Batch-RL Environment for offline training with a labeled dataset and with no risk of crashing an actual network. Both are described next.

Simulator Env For the Simulator Environment to be useful, it has to closely follow the dynamics of the GNS3 devices. In our formulation, environment dynamics are represented by the transition probability $P(s'|s, a)$ of the MDP. However, it is usually difficult to represent $P(s'|s, a)$ explicitly. In such cases, the simulator is used to model the MDP implicitly by providing samples from the transition distributions. One way

to implement this is through a *generative model* [20]. Formally speaking, the Simulator Environment is a randomized algorithm that, given an input of a state-action pair $\langle \hat{s}, a \rangle$, it outputs a $\langle r, \hat{s}' \rangle$ pair where r is the reward calculated according to a deterministic function given by equation 1 and $\hat{s}'$ is randomly drawn according to the estimated transition probability $\hat{P}(\cdot|\hat{s}, a)$.

To estimate $\hat{P}$, we let the agent interact with GNS3 and collected 50 hours of $\langle s, a, r, s' \rangle$ transitions. The scenarios were diversified to allow an accurate estimate of transition probability. In particular:

- we collected enough transient, recurrent, and persistent *network problems* on all links. Also, network problems were created on empty and congested links alike.
- we mixed between three types of adaptive bitrate (ABR) algorithms: buffer-base, throughput-based, and fixed bitrate. This ensured all available bitrates had been selected by the DASH clients and diverse *congestion patterns* had been created. Furthermore, we collected hundreds of thousands of video metrics (1 metric every 2 seconds) and calculated the probabilities of switching bitrates between the QoE levels {high, medium, low}.
- we let the agent operate in three modes: random policy (total exploration), gradually improving policy (decaying exploration), and expert rules (total exploitation). This ensured all actions had been presented and that good and bad actions had been chosen as well.

Once $\hat{P}$ is estimated, the Simulator Environment can be used to generate synthetic data $\langle \hat{s}, a, r, \hat{s}' \rangle$ to train the RL agent. Note that what we actually generate is the observation vector $\hat{o}$ that corresponds to the chosen state $\hat{s}$.

In order to make sure all actions are well presented in as many states as possible, we apply data augmentation to the next observation $\hat{o}'$. However, since the agent uses transition probabilities between states $\langle \hat{s}, \hat{s}' \rangle$ to plan future actions, we apply data augmentation in a way that preserves this transition. Formally speaking, let $\mathcal{T}_o^{-1} : \mathcal{O} \rightarrow \mathcal{S}$ be a deterministic function that maps observations to corresponding states. We want to perform data augmentation using observation function $\mathcal{T}_o : \mathcal{S} \rightarrow \mathcal{O}$ such that the augmented observation $\tilde{o}' = \mathcal{T}_o(s')$ and $\hat{o}'$ are mapped to the same state $\hat{s}'$. For example, if a client was streaming video at rate q_1 at time t with high QoE, and then a link failure happened causing its bitrate to drop to q_2 at time $t+1$ with low QoE, we augmented this by generating examples where q_1 and q_2 are randomly drawn from the set of all bitrates associated with high and low QoE, respectively. In other words, we change the observations in a way that doesn't change the underlying state nor the reward.

One advantage of using the Simulator Environment for training is the ability to put emphasis on rare cases as a way to help the RL agent explore. Also, to avoid overfitting, we tested the RL agent trained with this synthetic data on unseen scenarios in the GNS3 Environment to make sure it can generalize well.

Batch-RL Env. In practice, an operator might have access to an existing labelled dataset, but no access to a simulator. The Batch-RL Environment was built for such a situation, and utilizes labelled datasets collected from the field. Here, the

⁴<https://reference.dashif.org/dash.js/>

TABLE I
SUMMARY OF THE DEVELOPED GYM ENVIRONMENTS

	Simulator Env.	GNS3 Env.	Batch-RL Env.
Goal	For fast & safe exploration	To test with real devices	To experience realistic field data
Features	Synthetic Traffic (Video / FTP)	Diverse traffic (Video/FTP)	
	Random traffic patterns (modeled by $\hat{P}(s' s, a)$)	Random traffic patterns (described by $P(s' s, a)$)	
	MDP-aware data augmentation	Congestion using real throughput traces	Convert time-series data (metrics, tickets) into $\langle s, a, r, s' \rangle$ tuples
	Problem types: persistent, recurrent, and transient	Problem types: persistent, recurrent, and transient	

environment accepts two CSV files as an input and converts them to a Gym environment. The first file corresponds to time-series data collected from the devices log files. The second file contains the taken actions typically in a form of ticket with timestamps. Depending on dataset format, time alignment between observation logs and actions logs could be needed. Once aligned, the next state can be derived from each row by observing the next row. Finally, a reward signal can be computed and $\langle s, a, r, s' \rangle$ are stored in a CSV file. Each CSV file represents one episode in the Batch-RL environment.

To train the RL agent with the Batch-RL Environment, the agent is put on 100% exploration mode where instead of picking an action randomly, the agent picks its actions according to the CSV file. Each line in the file is read and the reward is provided accordingly. Note that despite restricting the agent's actions in every time step, Batch-RL can still be valuable if the dataset is rich enough. Once this offline pre-training is done, the agent can be deployed in a real network either for inference or to continue learning online and improve itself beyond the policy learned from the states and actions in the dataset.

3) *Training Algorithm:* RL algorithms can be generally divided into Model-based and Model-free algorithms. The agent in the former approach tries to model the environment purely from experience. The biggest challenge in this approach go beyond being expensive and complicated; it is that any bias in the model can be exploited by the agent, resulting in an agent which performs well with respect to the learned model but fail in the real environment. For that, we focus on the Model-free algorithms.

Model-free algorithms can be further divided based on the into Q-based algorithms and Policy-based algorithms. In this work, we chose double DQN and A2C as two candidate algorithms to represent those two families.

A2C In the Advantage Actor-Critic (A2C) algorithm [21], each agent is composed of a policy network (actor) and a value network (critic). The two networks interact and learn how to not only take actions but also evaluate their effectiveness. The key idea in A2C is to update the parameters of the agent's current policy in the direction of increasing the accumulated reward. The size of the step in its gradient algorithm depends on the value of the advantage function calculated by the critic. To prevent the actor from overoptimization on a small portion of the environment, we add an entropy regularization term to

the loss function to encourage exploration. We followed the standard Temporal Difference method [17] to train the critic network parameters. For readers who are not familiar with A2C, it helps to mention that the critic network merely helps to train the actor network. Once trained, only the actor network is required to execute and make decisions. Also, the critic network does not need to compute or approximate the exact value function, which is a high-dimensional object. Instead, the critic should ideally compute a certain projection of the value function onto a low-dimensional subspace spanned by a set of basis functions that are completely determined by the parameterization of the actor.

Double-DQN In double DQN [22], two NN are used to avoid the large overestimations of action values seen in the original DQN which leads to poor performance in some stochastic environments. We chose to use NN form of this algorithm as opposed to tabular form for two reasons. First, our state space is continuous, and NN can produce better representation compared to simply discretize the state space. Second, using two NN makes double DQN match closely the architecture of A2C algorithm (which also uses two NN). This makes it easier for us to compare the two algorithms. We tested both algorithms and found that A2C achieved better performance. Specifically, while double DQN converged a little bit faster, A2C surpassed the performance of double DQN after 80k steps. Hence, we picked A2C.

It is worth noting that while other recent improvements on the Actor-Critic algorithm exists, we chose A2C because it is still widely deployed in practice. This is true mainly because it is very efficient to train, and most importantly, very scalable thanks to the fact that A2C can be trained in a decentralized way [23]. For example, in adaptive video streaming, [24] achieved state-of-the-art performance using an asynchronous version of it. And in a recent paper [25], the authors' RL approach outperforms the existing ABR expert rules in a week-long worldwide deployment with more than 30 million video streaming sessions.

4) *Pretraining Algorithm:* In order to train the RL agent in practice; i.e., without RL executing "bad" decisions in a real network during training, we need a pretraining phase. For this purpose, we trained the A2C algorithm with two approaches: with the synthetic data in the Simulator Environment, and with Batch-RL using the offline data in the Batch-RL Environment. In the Simulator Environment, we first train the RL agent with

non-zero exploration from the beginning. We then transfer the pre-trained agent from the simulator to the real network, validate that the agent’s performance is as expected, then use it in production. In the Batch-RL Environment, the historical time-series data in which the context and action data is already pre-collected is traversed, the reward after each historical action is computed, and the learning algorithm is updated accordingly. Hence, offline RL can learn about the effectiveness of actions even if the decision to take these actions was not taken by the agent itself. In this approach, training the RL agent involves three steps:

- 1) pre-train with historical data from the target network. In our work, historical action data comes from NOC expert-rule decisions. Already at this step, ARE recommendations can be leveraged as suggestions to the NOC operators for manual actions.
- 2) deploy the RL agent with zero exploration, only exploitation, and confirm its behavior in the production environment. In our work, we expect to approximately reproduce the performance of expert rules at this point.
- 3) prudently allow small RL exploration during run time, to learn new and better action policies. At this step, we expect to significantly outperform the expert rules’ performance, eventually leading to superhuman performance.

Compared to the Batch-RL Environment, the Simulator Environment has an advantage and a disadvantage. The advantage is that the synthetic data can be produced in virtually unlimited amounts, while historical data from the real network, as used in the Batch-RL Environment, has a finite amount and can be expensive to collect. The disadvantage is that the transfer from the simulator to the real network is sensitive to simulation defects; therefore, developing a good simulator is both crucial and difficult, while learning in the Batch-RL Environment happens from a real network situation.

IV. EVALUATION

A. Experiment Setup

We compiled a set of videos to account for differences in scene characteristics: high motion in sports, low detail in animations, etc. All videos were encoded by the H.264/MPEG-4 codec at bitrates of 254, 507, 759, 1013, 1254, 1886, 3134, 4952, 9914, and 14931 kbps. The DASH segment length was the standard 2 seconds.

The RL neural network consisted of a value network (V-Net) of size [input=18, 256, 128, 18, output=1] and a policy network (π -Net) of size [input=18, 128, 64, 64, output=12] with the ReLU activation function and no weight sharing between V-Net and π -Net. This allowed the two networks to update their weights at different rates according to how difficult each task is. In other words, it is hard to learn the value-function due to the highly dynamic nature of the environment, but once a state-value is determined, it is relatively easy to pick the best possible action. For the other hyper-parameters, we used a discount factor of $\gamma = 0.99$, actor and critic learning rates of $\alpha = 10^{-4}$ and $\alpha' = 10^{-3}$ respectively, and an entropy factor controlled to decay from 1 to 0.1.

As mentioned earlier, for the QoE model we assigned a reward of +5, 0, and -5 for high, medium and low QoE levels. The running costs of carrying traffic inside AS1 was considered cheaper than going through the external AS3. The shortest and longest paths in AS1 had a cost of 1 and 3, respectively, while going through AS3 had a cost of 5. Finally, the costs of “*doing nothing*”, “*rerouting the traffic*”, and “*fixing a link*” were set at 0, 2, and 5, respectively.

We used the same topology shown in Figure 3, emulated in GNS3 and consisting of 5 Cisco routers, 3 IP/MPLS tunnels connecting varying number of AS1 DASH video clients to the AS2 video server. This varying number of clients occasionally created congestion, and we also randomly introduced router issues. The 3 ASs ran OSPF. We used MPLS tunnels and ACL per path. In total, we had 3 paths, 5 links, and 4 clients each running multiple instances of Dash.js to ensure links are going to be congested.

It is worth noting that despite using the same network topology for training and testing, the agent is not over-fitting for two good reasons: First, the traffic pattern is totally different. This is true because (i) the clients are turned on and off in a random pattern, (ii) when clients are turned on for the first time, they join a random path, (iii) the bitrates selected by clients ABR algorithm are always different. Second, the network issues are introduced randomly and can cause a change in network topology. For example, when a link is broken, the network topology changes from having three different paths to two or even one (if the link is shared among two paths). Combining these facts together, we tested the RL agent in cases that it didn’t see before and it worked well.

B. Evaluation Metrics

We used *Gain* as defined in Equation 1 to compare the performance of all methods. To mimic NOC operator actions, we implemented the expert rules shown in Algorithm IV-B. While this algorithm is still understandable, it also demonstrates why expert rules become complicated quickly as the network size increases, and why we didn’t choose a network larger than that of Figure 3.

V. RESULTS

All results reported are the average of 10 runs of experiments. Figure 4a shows the performance of ARE (A2C) versus that of the expert rules. Here, the agent was pretrained using synthetic data on the Simulator Environment and then tested on the GNS3 Environment. During pretraining, the agent required approximately 500K steps (4167 hours in real time) to outperform expert rules, but this was accomplished in about 6 minutes on the simulator because training on the simulator is much faster than real time. Once this pretrained ARE was deployed in GNS3, we can see in Figure 4a that it immediately had better and ever-increasing *Gain* performance while also its Reward was never worse than expert rules’ at any point.

But can ARE keep up its performance and not collapse? To answer that, we ran the same simulator-pretrained agent in GNS3 for 18 hours. The results are shown in Figure 4b,

Algorithm 1 Baseline

```

1: Input: State  $\mathcal{S}$ , Observations  $\mathcal{O}$ 
2: Input: Paths( $I_1 : R_{1,2,5}, I_2 : R_{1,2,4,5}, E_3 : R_{1,3,4,5}$ )
3: Output Action
4: if an internal path  $I_j$  is congested then,
5:    $I_k \leftarrow \text{Select}(I_k \text{ s.t. } k \neq j \text{ AND } I_k \text{ is not congested})$ 
6:   client  $\leftarrow \text{Select}(\text{random client from path } I_j)$ 
7:   if path  $I_k \neq \phi$  then
8:     action  $\leftarrow \text{Reroute}(\text{client to path } I_k)$ 
9:   else if external path  $E_3$  is not congested then
10:    action  $\leftarrow \text{Reroute}(\text{client through } E_3)$ 
11:   else
12:     action  $\leftarrow \text{Do Nothing}$ 
13:   end if
14: else if Network issue then,
15:   action  $\leftarrow \text{fix the issue (if persisting for 3 time steps)}$ 
16: else if External path  $E_3$  is being used then,
17:    $I_k \leftarrow \text{Select}(I_k \text{ s.t. } I_k \text{ isn't congested})$ 
18:   if  $I_k \neq \phi$  then,
19:     action  $\leftarrow \text{Select}(\text{random client from path } E_3)$ 
20:     action  $\leftarrow \text{Reroute}(\text{client to path } I_k)$ 
21:   end if
22: else
23:   Do Nothing.
24: end if

```

Fig. 4. ARE pre-trained using synthetic data on the Simulator Environment and tested on the GNS3 Environment. (a) Comparing both reward and gain. (b) testing the stability of ARE for 18 hours.

where we can see that ARE clearly maintains its stability and archives superhuman performance.

In terms of scalability, we argue that our proposed algorithm for ARE is scalable for the following three reasons:

- 1) The used RL algorithm is scalable: as mentioned before, there are many examples of this algorithm deployed in large scale such as [23] [25]. This is mainly because the NN in Actor-critic algorithms tend not to be very deep (which is very beneficial for inference), and it can be trained in a multi-agent setting where agent can be trained in a decentralized fashion and synchronize their experience together.
- 2) The network topology we chose, despite its small size, is not trivial and it includes many advanced concepts that won't change with scale. For example, the agent will experience (i) a network with multiple paths, (ii) with different path lengths i.e. number of hops (iii) some links are shared between (and will affect) multiple paths (iv) clients randomly join (or drop from) streaming sessions to generate different traffic patterns (v) client stream a real video traffic at adaptive bitrates (vi) some traffic can be rerouted through an external AS which can cost more.
- 3) The pre-training step is very fast. As we mentioned earlier, the agent can experience more than 4000 hours of real time in about 6 minutes.

To gain some intuition about that superhuman performance, we studied in detail the network problems that occurred and the actions taken by ARE and the expert rules as shown in Figure 5. We noticed that ARE was able to learn three main strategies beyond the expert rules, and those strategies led to fewer but more effective actions:

- 1) It learned to ignore any network issue that is not likely to affect any client. Such cases include problems in empty paths as well as transient problems.
- 2) It learned that it is better to pack all traffic on path 1 (R_1 - R_2 - R_5) as much as possible, instead of load-balancing between path 1 and path 2 (R_1 - R_3 - R_2 - R_5), because path 1 has fewer hops (lower cost). While on the surface one might think that packing all traffic into one path can cause congestion, the DASH algorithm inside the clients does tend to lower the video bitrate when it senses reduced available bandwidth, and this avoids congestion to some extent. ARE learned that it only needs to move traffic out of path 1 if clients are no longer able to maintain high QoE. In our tests, the link capacity allows all 4 clients to simultaneously achieve high QoE only if all clients are at one bitrate: 3134 kbps; anything above that bitrate will result in congestion, and anything below that bitrate will result in a significant drop in the reward signal. It is remarkable that ARE learned to “live dangerously” by allowing all clients to be in path 1 as long as their QoE is high and maintained, as shown in Figure 4b where we can see that the expert rules cause much fluctuations in the buffer size while ARE maintains the buffer sizes with more stability.
- 3) It paid particular attention to links that are shared between different paths: link R_2 - R_5 is shared among paths 1 and 2, while link R_1 - R_3 is shared among paths 2 and path 3 (R_1 - R_3 - R_4 - R_5). We noticed that ARE immediately fixed link R_2 - R_5 and immediately rerouted

Fig. 5. Actions and video buffer levels corresponding to the timeline of Figure 4a. (a) actions taken by ARE and the expert rules. Rx-Ry means fix the link connecting the two specified routers, Path: $x \rightarrow y$ means reroute a client between the specified paths. (b) average video buffer size at the client (higher means better QoE).

any traffic going through link R1-R3 in case it identified any problems with R2-R5 and R1-R3, respectively. The result of this can be seen in Figure 3a and Figure 4a, between timestep 50 and 75 where the performance of the expert rules drops significantly but ARE maintains its high performance.

It is interesting to note that none of the above strategies were part of the expert rules; ARE learned them by itself, validating our choice of using RL for complicated networks. This demonstrates that ARE can indeed progress beyond its initial training to achieve exceedingly high performance.

Finally, to evaluate the Batch-RL Environment, we pre-trained ARE using Batch-RL with an offline dataset that we created from running the expert rules on the Simulator Environment and labelling the data. The resulting agent was then tested in the Simulator Environment as follows: for the first 400k steps, the agent was not allowed to explore or train; it could only apply the rules it had learned during pretraining. Then, from step 400k onward it was allowed to perform small non-zero exploration. The reason we tested in the Simulator Environment, and not in GNS3 as we did for the previous

Fig. 6. ARE pre-trained on the Batch-RL Environment using labelled data and tested on the Simulator Environment.

test, is that in this test case we are allowing ARE to continue

training/exploring after step 400K and, as mentioned earlier, the Simulator Environment is much faster than the GNS3 Environment for training. The results of this test are shown in Figure 6. As expected, ARE achieves similar effectiveness as expert rules for the first 400K steps, because training with a labelled dataset is similar to supervised learning, which can roughly match but not highly exceed the rules from which it has learned. After step 400k, at which point ARE is allowed to explore and train further, ARE progressively outperforms the expert rules and never falls behind them at any point, demonstrating again that ARE is able to improve and learn new rules by itself.

VI. CONCLUSION

We showed that using RL, it is possible to automate NOC operations with superhuman performance, and this is significant for building autonomous and self-healing networks. We also showed that training such RL systems is practical, because it can be trained orders of magnitude faster than real time in either simulators or offline, without disturbing normal operations of a live network. For future work, we plan to train and test our system in a real setting. The challenges there are how and what kind of metrics/actions can we actually extract from a live network, and whether they will be sufficient for the RL agent to perform at the outstanding level shown in the reported emulations.

ACKNOWLEDGMENT

This work was financially supported in part by a MITACS Accelerate Cluster Grant #IT12571 and in part by Ciena Corporation.

REFERENCES

- [1] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," Proceedings of the ACM SIGCOMM Conference, 2011, pp. 350–361.
- [2] B. Raciis and A. Giorgetti, "Software-based fast failure recovery in load balanced SDN-based datacenter networks," Proc. International conference on information communication and management (ICIM). 2016, pp. 95–99.
- [3] S.A. Mohammed, A.R. Mohammed, D. Côté, and S. Shirmohammadi, "A Machine-Learning-Based Action Recommender for Network Operation Centers", IEEE Transactions on Network and Service Management, July 8 2021, 12 pages. DOI: 10.1109/TNSM.2021.3095463
- [4] D. Silver et. al., "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play", American Association for the Advancement of Science, Vol. 362, Issue 6419, December 2018, pp. 1140–1144. 4
- [5] Aiman Ghannami and ChenXi Shao, "Efficient fast recovery mechanism in software-defined networks: Multipath routing approach," Proc. International Conference for Internet Technology and Secured Transactions (ICITST), 2016, pp. 432–435.
- [6] R. Kanagavelu and Y. Zhu, "A pro-active and adaptive mechanism for fast failure recovery in SDN data centers," in Advances in Information and Communication Networks, 2019, pp. 239–257.
- [7] S. Wang, H. Xu, L. Huang, X. Yang, and J. Liu, "Fast recovery for single link failure with segment routing in SDNs," Proc. IEEE International Conference on High Performance Computing and Communications, 2019, pp. 2013–2018.
- [8] G. Dimopoulos, I. Leontiadis, P. Barlet-Ros, K. Papagiannaki, and P. Steenkiste, "Identifying the root cause of video streaming issues on mobile devices," Proc. ACM Conference on Emerging Networking Experiments and Technologies, 2015
- [9] H. Dinaki, S. Shirmohammadi, E. Janulewicz, and D. Côté, "Deep Learning-Based Fault Localization in Video Networks Using Only Client-Side QoE", IEEE Transactions on Artificial Intelligence, Vol. 1, Issue 2, October 2020, pp. 130–138.
- [10] A.R. Mohammed, S.A. Mohammed, D. Côté, and S. Shirmohammadi, "Machine Learning Based Network Status Detection and Fault Localization", IEEE Transactions on Instrumentation and Measurement, Volume 70, 2021, Article Sequence Number 3521710, 10 pages.
- [11] S. Deb, Z. Ge, S. Isukapalli, S. Puthenpura, S. Venkataraman, H. Yan, and J. Yates. 2017. "AESOP: Automatic Policy Learning for Predicting and Mitigating Network Service Impairments". In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17). Association for Computing Machinery, New York, NY, USA, 1783–1792. DOI:https://doi.org/10.1145/3097983.3098157
- [12] Z. Mammeri, "Reinforcement Learning Based Routing in Networks: Review and Classification of Approaches," in IEEE Access, vol. 7, pp. 55916–55950, 2019, doi: 10.1109/ACCESS.2019.2913776.
- [13] A.J. Dey, H.K.D. Sarma "Routing Techniques in Internet of Things: A Review", Trends in Communication, Cloud, and Big Data. Lecture Notes in Networks and Systems, vol 99. Springer, Singapore, 2020, https://doi.org/10.1007/978-981-15-1624-5_5
- [14] Y. Zhang, J. Xin, X. Li, S. Huang, "Overview on routing and resource allocation based machine learning in optical networks", Optical Fiber Technology, Volume 60, 2020, 102355, ISSN 1068-5200, https://doi.org/10.1016/j.yofte.2020.102355.
- [15] "Cisco Visual Networking Index (VNI) Complete Forecast Update, 2017–2022" February 27, 2019, Cisco Document ID: 1551296909190103
- [16] M. Hemmati et al., "QoE-Aware Bandwidth Allocation for Video Traffic Using Sigmoidal Programming", IEEE Multimedia, Vol. 24, Issue 4, October–December 2017, pp. 80–90
- [17] R. Sutton and A. Barto, "Reinforcement learning: An introduction," MIT Press, 1998.
- [18] J. De Vriendt, D. De Vleschauer and D. Robinson, "Model for estimating QoE of video delivered using HTTP adaptive streaming," 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), 2013, pp. 1288–1293
- [19] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," arxiv:1606.01540, 2016.
- [20] M. Kearns, Y. Mansour, A.Y. Ng, "Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes", Machine Learning 49, 193–208 (2002). https://doi.org/10.1023/A:1017932429737
- [21] V. Konda and J. N. Tsitsiklis. 2002. "Actor-critic algorithms", Ph.D. Dissertation. Massachusetts Institute of Technology, USA. Order Number: AAI0804543.
- [22] H. Hasselt, A. Guez, and D. Silver. 2016. "Deep reinforcement learning with double Q-Learning", In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI'16). AAAI Press, 2094–2100.
- [23] T. Chu, J. Wang, L. Codecà, Z. Li, "Multi-agent deep reinforcement learning for large-scale traffic signal control", IEEE Transactions on Intelligent Transportation Systems 21.3 (2019): 1086–1095.
- [24] H. Mao, R. Ravi, and M. Alizadeh, "Neural adaptive video streaming with pensieve", Proceedings of the Conference of the ACM Special Interest Group on Data Communication. 2017.
- [25] H. Mao, S. Chen, D. Dimmery, S. Singh, D. Blaisdell, Y. Tian, M. Alizadeh, E. Bakshy, "Real-world Video Adaptation with Reinforcement Learning", ICML workshop, 2019