Towards Superintelligence-Driven Autonomous Network Operation Centers

Using Reinforcement Learning

uOttawa

Basel Altamimi

School of Electrical Engineering and Computer Science
University of Ottawa

Thesis submitted in partial fulfillment of the requirements for the
Master of Applied Science
degree in
Electrical & Computer Engineering

© Basel Altamimi, Ottawa, Canada, 2021

I would like to dedicate this thesis to
the Palestinian child
Handala!

Acknowledgements

I would like to express my deep sense of gratitude and appreciation to my supervisor, Prof. Shervin Shirmohammadi for his guidance, timely advice, trust, great knowledge and support throughout this study. I would like to thank the rest of my thesis committee: Prof. Emil Petriu and Prof. Chris Joslin for their insightful comments and constructive feedback.

Besides my supervisor, I would like to thank David Cote, Chief Data Scientist at Ciena for the valuable discussions and the continuous support I received during my internship at Ceina. Also, to my brother and teammate Sa'di Altamimi, who has been a great role model for me. His dedication, organization, enthusiasm, and hard work shaped my thoughts.

My sincere appreciation to my lovely wife, Aziza and to my "almost" one year old daughter, Zaina for making my journey full of energy, very special and memorable.

Last but not least, I'm profoundly thankful to my devoted parents, Younis and Sabah, my in-laws Zakaria and Fareeha, and to all my family members, Mohammad, Samar, Wafaa and Jana for their strong and continuous support which propelled me to complete this work.

Abstract

Today's Network Operation Centers (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 work, we investigate whether such decisions can be taken by Artificial Intelligence instead of collective human intelligence, specifically by Deep-Reinforcement Learning (DRL), 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 by as much as 25%.

Table of contents

List of figures

List of tables

List of Abbreviations

NOC	Network Operation Center
OPEX	Operational Cost
ISP	Internet Service Provider
ML	Machine Learning
ARE	Action Recommendation Engine
SL	Supervised Learning
RL	Reinforcement Learning
QoS	Quality of Service
QoE	Quality of Experience
DL	Deep learning
DASH	Dynamic Adaptive Streaming over HTTP
MDP	Markov Decision Process
AI	Artificial Intelligence
ANN	Artificial Neural Network
DNN	Deep Neural Network

OTT	Over-The-Top
API	Application Programming Interfaces
AN	Adaptive Network
MOS	Mean Opinion Score
IID	Independent and Identically Distributed
SDN	Software-defined Network
NFV	Network Function Virtualization
ZSM	Zero-touch network and Service Management
KPI	Key Performance Indicator

Chapter 1

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. 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.

1.1 Motivations

In the industry, NOC manages large-scale networks to ensure the network is operating efficiently. 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 Internet Service Provider's (ISP) operational expenses/costs (OPEX). The set of rules used to map different situations to actions is referred to as a policy. As the network grows in size and complexity, human operators face two major problems: (i) how to identify the root-cause of a problem given the sheer amount of data produced? and (ii) what remedial action to take given a specific problem is detected? This makes the hand-crafted policies less efficient and more difficult-to-design. Fortunately, Machine Learning (ML) can provide an answer to those questions.

ML algorithms have been widely successful in making predictions or decisions without being explicitly programmed to do so. The way ML achieves this is by producing models that learn and improve from past experiences. A major part of what makes ML so valuable is its ability to detect what the human eye misses. ML models can catch complex patterns that would have been overlooked during human analysis. This ability improves as the size of collected data increases. Unlike the case with human operators, the more data produced by large networks, the better the ML algorithm becomes at detecting problems!

Although a ML model can be better than humans at detecting network problems, this ability alone doesn't necessarily translate to better actions. The ability to take good actions depends on the type of supervision used when training the ML model. In supervised learning (SL), the human operator provides the model with a set of labels (i.e., what "action" to take for each input) rather than what "policy" to follow. It is up to the model to learn the rules that map performance metrics to actions. This approach can be easier to implement in practice when the actions are known but hard-to-describe in a form of hand-crafted policy. However, the policy learned by SL model doesn't outperform the human expert who generated these labels. The work in this thesis aims at solving the aforementioned problem using Reinforcement Learning (RL) as described in §1.2.

Although RL models can be used to fully automate an NOC, its other practical usage is as an Action Recommendation Engine (ARE) that recommends an action to the human operator, still leaving the final decision in the hands of people and not machines (for more on AI ethics see [16]). In either case, a successful algorithm should consider the following challenges:

- Networks' states can be highly variable, unpredictable, and unobservable. Despite that, ARE must be able to suggest/take *correct* remedial actions.
- ARE algorithm must take into account the long-term effects of its decisions by *proactively* plan for future rather than relaying on instantaneous decisions.
- Maximizing Quality of Service (QoS) objectives (e.g. bandwidth utilization or avoiding network congestion) does not necessarily result in achieving the optimum goal of NOC. ARE must instead consider optimizing an objective function that takes into account *both* user's Quality of Experience (QoE) and OPEX constraints.
- Training an RL model faces many practical problems: it takes a long time to train to achieve good performance, and during that training models make mistakes,

which can be costly to make in a real network. As a result, it is important to include an offline stage that allows the model to learn *quickly and safely* before being deployed in real networks.

- Given the high cost of wrong actions, ARE should provide some level of *interpretability* that allows the human expert to assess ARE actions and take-over when necessary.
- ARE should be scalable.

1.2 Approach

The work in this thesis was motivated by the challenges mentioned in §1.1, and is an attempt to automate NOC's remedial actions. To do so, we break down this challenging task into four stages: First, we show empirically that a Deep learning (DL)-based automation system for NOC can achieve a comparable performance with human expert rules; demonstrating the feasibility of NOC automation. Second, we formulate the problem as Markov Decision Process (MDP) and designed an action recommendation engine that uses RL to figures out, by itself, the relationship (i.e. policy) between the network's raw data and remedial actions. Because the engine can discover relationships that are too complex for humans to figure out, we show that the engine outperforms expert rules and archives superhuman performance.

Third, we design three different environments to train/test our RL algorithm. Each environment is tailored to tackle a specific practical challenge. The first environment is using *GNS3* emulator with real devices. Its main purpose is to test the performance of different algorithms. However, training RL model on GNS3 has two practical problems: (i) it takes a long time to train it to achieve good performance, and (ii) during that training it makes mistakes, which can be costly to make in a real network. The *Simulator* environment aims at speeding up the training process by approximating the GNS3 environment. *Batch-RL* environment, on the other hand, allows the RL model to learn from field-data in a way similar to SL. This brings the RL model to a level similar to a human expert. summary of the three environments is listed in Table 1.1.

Finally, we define our objective function to be QoE-OPEX, implement a baseline algorithm, and specify new testing scenarios. Results validated the feasibility of automating an NOC with superhuman performance.

Table 1.1 Summary of the Developed Gym[1] Environments

		Simulator Env.	GNS3 Env.	Batch-RL Env.
Goals:	1.	For fast & safe exploration	To test with real devices	To experience realistic field data
	2.		To aid the design of the Simulator Env.	
Features:	1.	Synthetic traffic (Video & FTP)	Diverse traffic (Video & FTP)	Large amount of traffic (dataset-dependant)
	2.	Random traffic patterns (modeled by $\hat{P}(\hat{s'}\|\hat{s},a)$)	Random traffic patterns (described by $P(s'\|s,a)$)	
	3.	MDP-aware data augmentation	Congestion using real throughput traces	Convert time-series data (metrics and tickets) into $< s,a,r,s' >$ tuples
	4.	Simulate persistent, recurrent, and transient problems	Emulate persistent, recurrent, and transient problems	Field problems (dataset-dependant)

1.3 Contributions

In this work, we design and implement an RL-based closed-loop ARE that can autonomously self-drive a lab network from raw data. The goal is that not only learns when and where to apply a specific set of rules automatically, but also use RL to discover efficient rules that are not known to human operators. To the best of our knowledge, this work is the first to explore the feasibility of automating an NOC with superhuman 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. Our contributions can be summarized as follows:

- we formulated the problem of ARE in NOC as MDP process and solve this problem using RL. We used the Dynamic Adaptive video Streaming over HTTP (DASH) as a challenging use-case and demonstrated that the model can learn new implicit and non-trivial rules, on its own, exceeding human performance with expert rules.

- unlike existing algorithms, we proposed an objective function (i.e. reward signal) that goes beyond QoS-based goals (e.g. avoid congestion, improve bandwidth utilization, etc.) to include ISP's OPEX and clients' QoE. Our RL algorithm was trained to anticipate network issues that might degrade the users' QoE and to act within the OPEX constraints.

[1] https://gym.openai.com/

- 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. We equipped the simulator with MDP-aware data-augmentation capabilities that enrich the collected dataset without changing the underlying dynamics of the real environment we are simulating.
- 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.

1.4 Publications

The contributions listed in §1.3 have led to the following publication and patent:

(Patent) S. Altamimi; B. Altamimi; S. Mohammed; S. Shirmohammadi; D. Cote, *"Action Recommendation Engine (ARE) for Network Operations Center (NOC) solely from raw un-labeled data"*, US patent application 10.2846, Feb. 3, 2021

(Paper) S. Altamimi; B. Altamimi; S. Shirmohammadi; D. Cote, (2021): *"Automating Network Operation Centers with Superhuman Performance"*. TechRxiv. Preprint. https://doi.org/10.36227/techrxiv.15176175.v1

The contents of this paper will appear in Chapter 4 and 5 and has been reused with permission.

1.5 Organization of Thesis

In the next chapter we review some theoretical background from areas related to the topic of this project. It includes some details about DASH standard, RL, and QoE metrics. The reviewed algorithms were divided into three groups based on whether they used fixed or learnable heuristic, and whether they were implemented on client-side or server-side. We devoted one section for RL-based algorithm and explained how they are related/different from our proposed algorithm.

In Chapter 3, we present a survey of the recent developments in the field of adaptive networks. The related works have been reviewed in two main categories: (i) traditional methods that use expert-rules to resolve network issues by a mean of traffic-engineering and failure recovery, (ii) ML-based approaches. Chapter 4 contains the system design and implementation which details all our contributions. A summary of the obtained results, and some directions for future research in this area appear in Chapter 5, and 6 respectively.

Chapter 2

Background

This chapter provides an overview of the systems and techniques underlying the whole work. It first describes the main concepts behind network automation. Then, it provides a brief introduction to adaptive video streaming and how to measure users' QoE. Finally, it covers the main concepts of the two ML techniques used in this thesis, namely DL and RL.

2.1 Video Streaming

Video streaming has rapidly emerged in the field of information and technology and watching videos streamed over the Internet is now a ubiquitous and trivial daily activity for the average user. Growing adoption of smartphones to watch movies, TV shows, live sporting, and other events is a major factor promoting the market growth. In fact, the need for video streaming goes beyond entertainment industry. With the ongoing COVID-19 pandemic and the closure of many workplaces and campuses, the Internet has become an essential tool for employees and students. Video streaming technology helped organizations conduct their daily operations remotely via online meetings, live table conferences, and interactive sessions with customers.

2.1.1 DASH Video Streaming

Sandvine reports that the portion of download stream traffic related to video streaming is 65.5% of all mobile traffic and 60.6% of all Internet traffic [28], led by HTTP media streaming services such as Netflix and YouTube. These services are referred to as Over-The-Top (OTT) video, due to directly delivering video to consumers over the Internet, instead of distribution channels such as broadcast cable or satellite. They

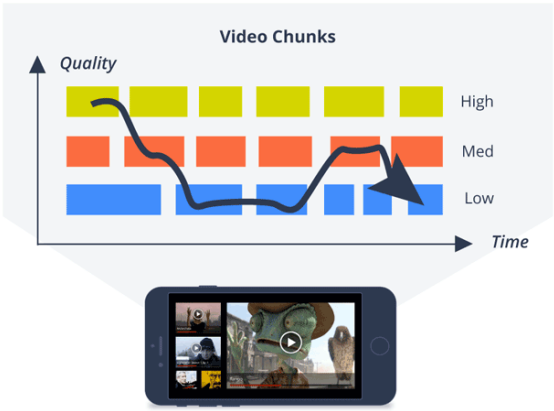


Fig. 2.1 Simplified DASH workflow [source: kaltura.com]

services stream videos in a continuous manner while trying to provide the consumers with the highest Quality of Experience (QoE) possible.

Traditionally, video streaming applications were delivered using a progressive download mechanism. A *progressive* video stream is simply one single video file being streamed over the internet. The progressive video can be stretched and squashed to fit different screen sizes, but regardless of the device playing it, the video file will always be the same. There are two problems associated with progressive streaming. On one hand, the mismatch between the screen size and file size degrades the video quality. On the other hand, if the users have poor-quality internet connection, and cannot download the video stream quickly enough, then the video will need to pause, wait for more data, and then start again. This makes watching a video horrible for the user. This problem is referred to as buffering and is very common, especially on mobile devices, where the connection can vary greatly depending on the user's location.

The Dynamic Adaptive Streaming over HTTP (DASH) is a standard that emerged in 2011 to provide a standardized solution for the efficient and easy streaming of multimedia using existing available HTTP infrastructure. In a conventional DASH application, several copies of every video at different bitrates are stored on the streaming server. The DASH client requests an XML configuration file called Media Presentation Description (MPD), containing all the metadata that the client needs in order to stream the video. Among these data is the list of available encoding bitrates supported for the media content being requested. The real power of adaptive bitrate streaming is that it "adapts" (see Figure 2.1). This means that as a user's internet connection changes, the adaptive stream will switch back and forth between video qualities to ensure smooth streaming and avoid buffering.

Impairment	Quality	MOS	User Satisfaction	QoE
Imperceptible	Excellent	5	Very satisfied	5
Perceptible but not annoying	Good	4		4.3
			satisfied	4
Slightly annoying	Fair	3	some users dissatisfied	3.6
			Many users dissatisfied	3.1
Annoying	Poor	2	Nearly all users dissatisfied	2.6
Very annoying	Bad	1	Not recommended	1

Fig. 2.2 MOS scale for subjective video quality assessment. Source [2]

2.1.2 Quality of Experience

Quality of Experience (QoE) is the perceptual quality of a service from the end-users' perspective. Genuinely measuring the factors that influence or define a user's video QoE is a major technological challenge. As a poor substitute, many vendors pitch solutions that aren't built around providing real insight, but instead are built around what's easy to measure such as network health metrics, transport layer performance (e.g., TCP packet drops, round-trip time, jitter, etc.), bandwidth averages, etc. These solutions don't bring the video service providers any closer to understanding the actual experience of their users.

Ideally, it is desired to assess QoE subjective test, where human viewers evaluate the quality of test videos under a laboratory environment. However, *subjective* tests are costly and time consuming. It is also very challenging to measure it in an "online-mode", specially when it comes down to providing a QoE score for each frame in the video under test. Therefore, *objective* quality models have been developed to predict QoE, in terms of Mean Opinion Score (MOS), based on available objective parameters. One of the well-accepted QoE models for adaptive streaming provides a prediction or estimation of MOS through a linear combination of the following three parameters: the average quality of video segments over a streaming session, the variation of video qualities and the effect of rebuffering. Figure 2.2 shows MOS scale for subjective video quality assessment.

2.2 The Adaptive Network

The recent developments in the multimedia industry, especially OTT services have changed the Internet traffic to multimedia traffic. Such a drastic increase of multimedia traffic poses real challenges for ISPs; today's networks are full of overbuilt legacy systems and protocols, leaving them unable to rapidly scale or adapt. As a result, it is common to attribute the degradation of users' QoE to bottlenecks in the ISPs' networks. In fact, if a customer does not receive adequate QoE, it is more likely to become a churner. This puts more pressure on ISPs compared to OTT service providers because most users switch their Internet connection to another ISP when they are not happy about the QoE [1]. Consequently, this leads to decrease in market share and reputation as well as lower revenue for an ISP.

To address these challenges, ISPs have been looking at a number of technologies to reduce OPEX and increase the agility of their networks. There is universal agreement that whatever technologies are at play, the industry needs a more responsive, automated, and agile network. The concept of *adaptive networks* (ANs) represents the vision of an ideal network that utilizes intelligent automation, guided by streaming telemetry, data-drive analytics, and intent-based policies to rapidly scale, self-configure, self-heal, and self-optimize by constantly assessing network pressures and demands [5].

Figure 2.3 summarizes the three foundational elements of ANs. The first element is programmable infrastructures. It allows ISPs to configure their network devices via common and open APIs, and to access network performance data in real-time. Developing an open, multi-vendor, and multi-domain framework is crucial because it is no secret that no one vendor can deliver best-inbreed hardware, software, and services for every application. The second element is network analytics. As providers face an explosion of data and demand across their networks, the implications may seem complex. But these challenges also have their benefits, as they grant providers access to a growing wealth of information that, if harnessed effectively, can help them make more effective decisions to optimize their network performance and deliver a better customer experience. This is where artificial intelligence (AI) comes into the picture. AI can be used to build AREs that can predict potential network problems, anticipate trends, and turn them into actionable insights. The third element is concerned with automating the process of action execution.

Although it might be safer to use ARE to recommend actionable insights to human operators and leave the final decision in the hands of people and not machines, implementing a closed-loop automation system can help archiving unprecedented efficiency and optimality: efficiency because ARE makes decisions much faster than

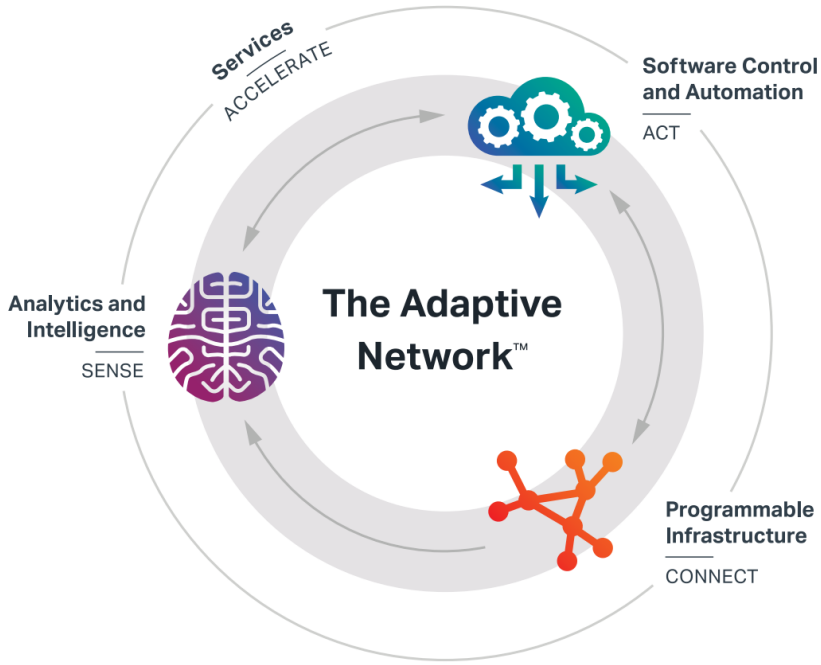


Fig. 2.3 Foundational elements of the Adaptive Network. Source: [5]

humans, and optimality because ARE's decisions will lead to much better results. This thesis attempts to address two practical problems related to AI-based ARE: how to train AI models to achieve superhuman performance in a reasonable time? and how to avoid making mistakes, which can be costly in a real network?

2.3 Deep Reinforcement Learning

The industrial revolution brought machines that can automate many tasks. This was followed by a digital revolution where we have data represented in a digital form. Inventors have long dreamed of creating machines that think. This was first implemented in a form of hard-coded expert rules. These rules serve as a reference for the machines in order to decide what to do in various situations. Today, AI is thriving as a substitute approach. The main idea is to focus on developing algorithms that allows the machine to learn, or even discover, the rules rather than memorizing them.

In the early days of AI, the field rapidly tackled and solved problems that are intellectually difficult for humans as long as it can be described by a list of formal mathematical rules. The true challenge to AI proved to be solving the tasks that are easy for people to perform but hard for people to describe formally; like recognizing spoken words or faces in images. To solve such problems, scientists turned to nature for

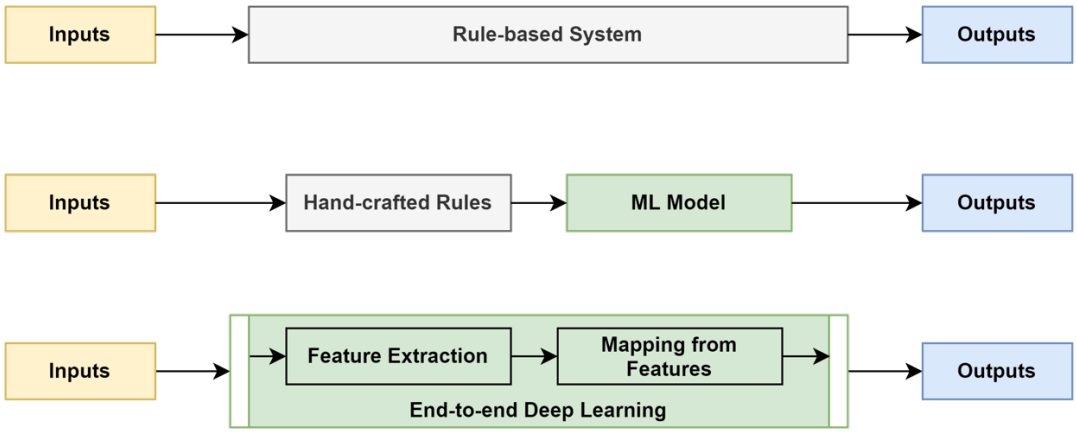


Fig. 2.4 Transition towards end-to-end AI-based systems. Green blocks are learnable.

inspiration. This happened in two main directions: *model architectures* and *learning algorithms*.

The first direction led to the invention of Artificial Neural Networks (ANN) which was inspired by the biological neurons in the brain. These artificial neurons are organized into layers that are responsible of extracting higher level features from its inputs. This type of models eliminated the need for feature engineering by letting the machine extract the description, by itself, from the raw data. ANN models that use many layers are referred to as Deep neural networks (DNN). The field of ML concerned with the design and training of such deep models is called deep learning (DL). Figure 2.4 shows the trend towards an end-to-end DL approach.

The other direction led to the introduction of four major learning paradigms: *supervised* learning, *reinforcement* learning, *unsupervised* learning, and *imitation* learning. This thesis focuses on training DNN using the first two approaches in order to build an ARE that performs at least as good as human experts. In this section, we present the basic concepts of both paradigms in parallel to better understand when, where, and how they can be used. In particular, we define the *task* each approach is trying to solve. Then, present the different *sources of information* available for the model to learn from. Next, we describe how to design a *model* that can represent certain beliefs about the task, and how to design a cost function that measures how well those beliefs correspond with reality. Finally, we compare how SL and RL *training algorithms* help the model minimizing the cost function.

2.3.1 The Task

A typical SL problem is described as a set of input-output pairs that were generated by an *unknown* process g. This set is usually referred to as the *training dataset*. The term supervised learning originates from the view of the correct output (*labels*) being provided by an instructor or teacher who shows the model what to do. The model is required to analyze this training dataset and to tries to learn a mapping function f that best approximates the unknown function g. If the model is good enough, the function f should be able to determine the labels for unseen instances. This requires the model to *generalize* from the training data to unseen situations in a "reasonable" way. This is important because the dataset might be *biased* and might include *noisy examples* that the model should ignore. In RL, however, the true labels are not available. The only available feedback is a "reward" signal that tells the model how good or bad its decisions are. In this case, a typical RL problem can be seen as optimizing the unknown function g that generate these rewards.

Many kinds of tasks can be solved with SL. Some of the most common tasks include classification, regression, prediction, and anomaly detection. In *Classification*, the model is asked to specify which of k-categories some input belongs to. To solve this task, the learning algorithm is usually asked to produce a function $f : \mathcal{R}^n \rightarrow \{1, ..., k\}$. When $y = f(x)$, the model assigns an input described by vector x to a category identified by numeric code y. The model can also produce a probability distribution over classes. In *Regression*, the model is asked to predict a numerical value given some input. *Prediction* tasks involve data that changes over time. In this task, the model is given a set of historical values and is asked to predict value(s) in the future. In *Anomaly detection*, the model sifts through historical values and flags some set of events as being unusual. Anomaly detection can be performed on time-series data, by comparing current value with previous values and decide if the new value follow the same pattern or not.

It is worth noting that, in practice, a SL model can be asked to solve tasks that involve multiple sub-tasks at the same time. For example, in network automation, the model can be asked to monitor network's health and perform real-time anomaly detection to detect problems. When a problem is detected, the model should select the proper action(s) to take from a list of possible actions (i.e., perform classification). It is also desired to be able to predict problems before they happen to avoid any service loss. Regression can be used to produce a single number that represents the current "health score" of the network based on a set of performance metrics.

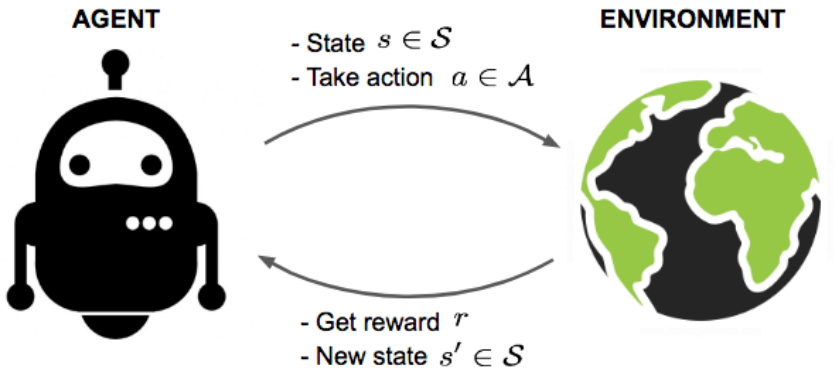


Fig. 2.5 The concept behind Reinforcement Learning

RL tackles problems where the solution is not known but rather a form of feedback is still available. For example, consider a task where the model has to learn a sequence of actions in the correct order to achieve a specific goal. In such a case, a single action might not be useful or important; what is important is the *policy*. Unlike SL, which has to provide the model with a "good action" to take at each step, RL considers an action as good only if it is part of a "good policy". Thus, it generates a feedback signal that reflects how well a specific policy do in terms of achieving the ultimate goal. For example, to teach a robot how to walk we might not have access to the right amount of force a robot should apply on each joint at each point in time, but we can rate its walking skills and provide it with a reward (or a punishment) to encourage it to walk properly.

Planning is another example of tasks suitable for RL. In this type of problems, the goal is to maximize certain type of long-term future reward. Unlike SL, where there is no explicit concept of "agent" or "environment" (and their interaction), the basic mechanism behind RL is to plan to take good actions under uncertainty merely through trial and error. This interaction is formally described using mathematical framework called Markov Decision Process (MDP) which is defined as $\{state, action, reward, transition\}$ tuple. Figure 2.5 illustrates this basic idea. As we can see, an agent observes the *state* of the environment as an input and makes a choice about its *action*. However, rather than receiving the true action as a label to compare with, the RL agent receives some reward and a probabilistic transition to a new state. The transition function helps the RL model predicts how the environment would react in response to its action(s) which in turn is a very instrumental in learning how to plan to take better actions.

It can be observed that many problems can be cast as either SL or RL problems. Take network automation as an example. On one hand, one can setup a lab, design a

reward function, and let an RL agent interact and learn. On the other hand, it is also possible define this problem as a classification problem. This can be done by collecting field data from the lab, manually labelling each example with a suitable action as the desired class, and then training the SL model. One can expect the SL model to learn more efficiently because it has access to direct supervision, but the RL model has an advantage of being able to explore; which can lead to better performance compared to human expert-rules.

2.3.2 The Data

As we mentioned earlier, the examples in SL are presented as a static dataset, where each example is represented as a tuple of {inputs, labels}. The model samples a "batch" of examples from this dataset over and over until it learns the underlying mapping between the inputs and the labels. RL on the other hand packages the examples inside an “environment”. In this setup, the agent is responsible for constructing its own "dataset" (known as episodes) by interacting with the environment. Each episode is represented by a set of $\{state, action, reward, next\ state\}$ tuple.

RL is different from SL in the type of guidance the model will receive. In SL, the model is presented with the true label that represents the solution to the task in hand. This is usually possible because many tasks can be solved easily by humans and thus the solution is known (e.g., face recognition). The challenge might still be in the feasibility of labeling enough amount of data since DNN tends to require huge datasets to perform well. RL tackles problems where the solution is not known but a form of feedback is still available. This form of *indirect* guidance makes training RL agents much harder than SL. RL can also be applicable when the feedback might be delayed. For example, in a game of chess, the agent might not receive a reward after each move but rather a single delayed reward at the end telling it whether it won or lost.

The quality of the input features heavily affects the performance of both SL and RL models. For example, when SL is used to decide whether to dismiss a patient from the hospital or not, the model does not examine the patient directly. Instead, the doctor tells the system several pieces of relevant information (called features). A model learns how each of these features correlates with various outcomes. However, it cannot influence the way that the features are defined in any way. Similarly, in RL, the agent might not be able to fully observe the environment. If the observations were unreliable or incomplete, it will negatively affect the quality of its decisions.

Another issue that affects the performance of SL and RL models alike is the quality of the provided feedback. For example, wrong labels in SL or poorly-designed reward in

RL can deceive the model and push it toward the wrong direction. Similarly, imbalanced data can result in a model that is biased towards a subset of actions/classes and is unable to generalize well. While the SL model can't do anything about imbalanced datasets, the RL agent is responsible for collecting its own dataset and has the power to avoid this problem during the training process. Since the state-action pair decides the next state, it is possible that the agent will keep landing in the same set of states because it keeps repeating the same set of actions. It would be desirable to visit diverse set of environment states and *explore* different actions before start to *exploit* the best learned actions. This phenomenon is described in literature as the exploration-exploitation dilemma.

Chapter 3

Related Work

In this chapter, we review the related algorithms in ARE. We group the most recent related work in this area into two groups based on how ARE policy is defined: (i) traditional methods that uses hand-crafted expert-rules, and (ii) ML-based methods. We highlight the differences and similarities between our proposed algorithm and related works.

3.1 Traditional Methods

Today's NOCs perform their tasks either manually [14] or, for the simplest tasks, with pre-determined expert rules [26]. Essentially, a team of NOC technicians works 24/7 to ensure 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.

Current state-of-the-art either uses handcrafted expert rules for recovery in case of network problems, or uses ML to detect network problems but do not recommend

actions. For example, the work in [13] 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 [17] where recovery paths are adaptively updated based on the current state of network's load. By providing the backup forwarding rules in advance, fast recovery is achieved upon a failure. Similarly, the work in [32] adopts segment routing in order to reduce the number of forwarding rules and deal with link failure problem in Software-defined Networks (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.

3.2 ML-based Algorithms

One of the earliest works that uses ML to localize network faults is [10] which detects and localizes network problems leading to Quality of Experience (QoE) degradation. 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 [11] detects and localizes network faults in server side, ISP network, 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 [24] also uses ML to detect the network status as normal, congestion, and network fault, as well as localize the fault with accuracy 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 [8] 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 to reboot an interface or not, 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. Another work in NOC automation is [25]. It showed empirically that an ML-based automation system for

NOCs has a comparable performance with human expert rules. Similar to our work, the authors optimized their model while taking OPEX into account. However, their work was limited to FTP traffic and was not tailored for DASH video streaming, which is way more challenging.

It should also be mentioned that there is an increasing research trend of using ML for network routing [21][9][33]. However, network routing and NOC operations are not at all the same, and the former is outside the scope of this paper.

Zero-touch network and Service Management (ZSM) is another related field that overlaps with the *adaptive network* concept discussed in Chapter 2.2. The main similarity lies in using ML to automatically orchestrate and manage network resources while assuring the QoE demanded by users [12]. The main difference, however, lies in the fact that our work targets ISPs running traditional networks infrastructures while ZSM mainly relies on SDNs and Network Function Virtualization (NFV) capabilities that can be found in more recent systems such as in 5G networks.

To the best of our knowledge, this work is the first to explore the feasibility of automating an NOC with superhuman 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 [25] which matches the performance of expert rules, our method outperforms expert rules, and compared to [8], 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 [8]'s only action of interface rebooting.

Chapter 4

System Design

This chapter explains the design of our ARE. We first describe the testbed setup in §4.1 and the collected dataset in §4.4. In §4.2, we formulate the problem as a classification problem and use DL to learn a policy that can match human performance. We extend this algorithm using RL with the goal of discovering new policies that can result in superhuman performance. Section §4.3 shows how to use offline-training to safely achieve this performance while avoiding bad actions, which can be costly in real environments.

4.1 Testbed Setup

A network topology can be modeled by a graph $G = (V, E)$, where V is a set of routers and E is a set of links connecting routers. Figure 4.1 is a screenshot of our testbed running in GNS3[1]. 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 this particular topology for two main reasons. First, it reflects many aspects of real-networks. For example, the topology has multiple tunnels with different number of hops, it has tunnels that share the same physical link, it contains loops,

[1] https://www.gns3.com

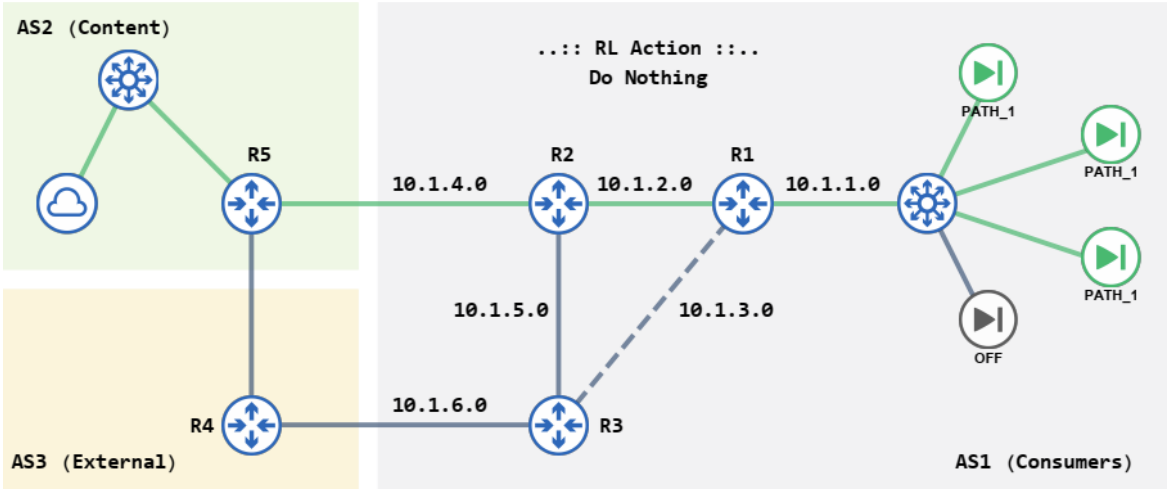


Fig. 4.1 Our system with its network topology.

etc. Second, this topology represents a realistic building block of larger networks. In particular, when it comes to scalability, one can break very large network into subnetworks each consists of one or more tunnels that share the *same endpoints* (R1 and R5 in our case). This way of creating subnetworks abstracts the underlying topology as an *overlay network* similar to the one in Figure 4.1. As a result, it is enough to train one RL-agent on this topology, and then deploy multiple copies of the trained model to control the whole network in a decentralized fashion.

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 [6]. Second, video traffic is inelastic and bulky with a sigmoidal QoE function, making it very challenging to optimize users' QoE [15], 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. Despite being very common, the first problem is not related to the traffic planning algorithm used by the network operator. It is due to hardware issues related to the health of physical devices. On the other hand, congestion can happen due to sub-optimal traffic planning were clients demands exceed link capacity. Formally speaking, congestion can be defined as:

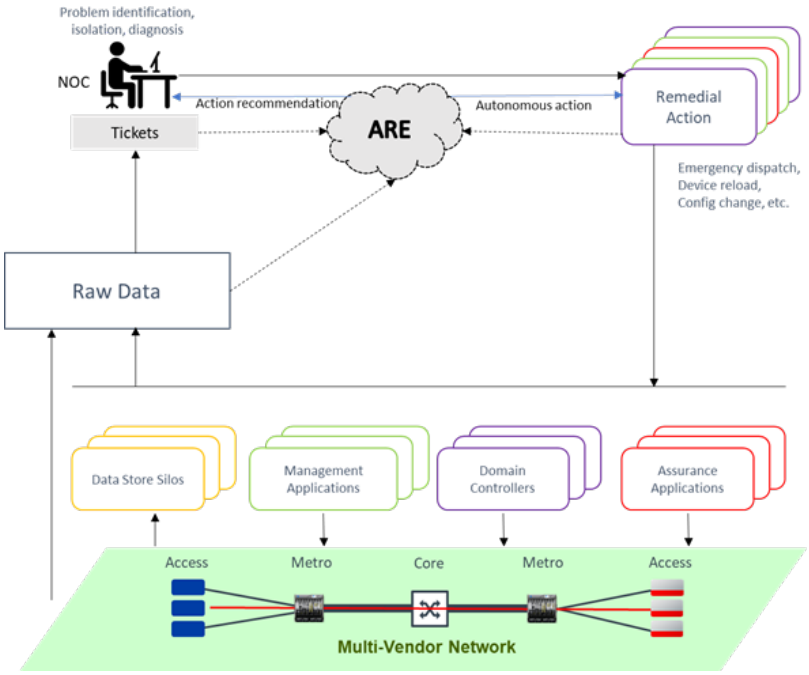


Fig. 4.2 ARE in a typical Network Operation Center (NOC)

$$C_g = min\left\{\frac{r_x + \sum_n r_n}{C_b}, 1\right\} \tag{4.1}$$

where r_x, r_n are background traffic and video traffic respectively. C_b is channel capacity.

4.2 The Design of ARE

In this section, we propose an OPEX-aware ARE, shown as such in Figure 4.2, using RL, with the goal of outperforming 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.

4.2.1 Problem Formulation

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 $\{\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{T}_s, \mathcal{T}_o, \mathcal{R}, \gamma\}$ defined as follows:

States $\mathcal{S}$ 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 true environment state for two purposes: to calculate the rewards and to serve as an input to the expert rules. $\mathcal{T}_s$ 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 §4.2.2).

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 §4.4). $\mathcal{T}_o$ 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 §4.2.2).

Actions $\mathcal{A}$: the action taken by an agent at a given timestep 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 randomly.

Reward $\mathcal{R}$ and discount factor γ: for OPEX, we add three main expenses: the cost of performing an 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 $\Phi(.)$ 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:

$$\mathcal{G} = \Phi(QoE) - \text{OPEX} \tag{4.2}$$

We adopt the well-accepted QoE model in [7] for adaptive video streaming. The QoE is defined as

$$\begin{aligned} \textbf{QoE} = \alpha \sum_k \frac{q_k}{K} - \beta \sum_k \left| q_{k+1} - q_k \right| \\ - \gamma \sum_k \left(\frac{d(q_k)}{C_k} - B_k \right)^+ - \delta \end{aligned} \tag{4.3}$$

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 starts 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:

$$R = \mathbb{E}[\sum_{i=1}^{h} \gamma^i \mathcal{G}] \tag{4.4}$$

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.

4.2.2 RL Environments

We built three custom OpenAI Gym environments [3] 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 1.1.

GNS3 Environment

In the GNS3 Environment, 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 timesteps, and we expect the agent to have

fixed it by then. Transient problems come and go in 3 timesteps, 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 DASH[2]. Finally, clients turn off and the cycle repeats.

In order to train the RL agent in GNS3 environment, each $\{s, a, r, s'\}$ step requires around 30 seconds to complete, so the size of our timestep is $30s$. While this environment worked well, it is not 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 Environment

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* [18]. Formally speaking, the Simulator Environment is a randomized algorithm that, given an input of a state-action pair $\{\hat{s}, a\}$, it outputs a $\{r, \hat{s}'\}$ pair where r is the reward calculated according to a deterministic function given by equation 4.2 and $\hat{s}'$ is randomly drawn according to the estimated transition probability $\hat{P}(.|\hat{s}, a)$.

To estimate $\hat{P}$, we let the agent interact with GNS3 and collected 50 hours of $\{s, a, r, s'\}$ 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.

[2]https://reference.dashif.org/dash.js/

- 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 $\{\hat{s}, a, r, \hat{s}'\}$ 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 $\{\hat{s}, \hat{s}'\}$ to plan future actions, we apply data augmentation in a way that preserves this transition. Formally speaking, let $\mathcal{T}_o^{-1} : \mathcal{O} \to \mathcal{S}$ be a deterministic function that maps observations to corresponding states. We want to perform data augmentation using observation function $\mathcal{T}_o : \mathcal{S} \to \mathcal{O}$ 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. Figure 4.3 illustrates this process.

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 Environment

In practice, an operator might have access to an existing labelled dataset collected from the field, but no access to a simulator. The dataset in this format is suitable for SL-based ARE and is discussed more in Section §4.4. The Batch-RL Environment

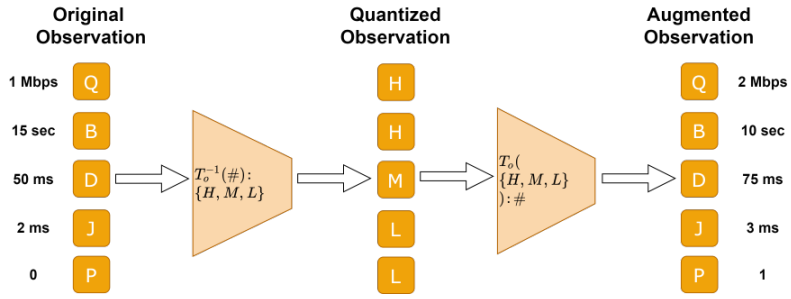


Fig. 4.3 Data augmentation in simulator environment. Observations: $\{Q$: bitrate, B: buffer, D: delay, J: jitter, P: packetloss$\}$. States: $\{H$: High, M: Medium, L: Low $\}$

is built for such a situation in order to utilize labelled datasets in RL settings. In this case, the 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 $\{s, a, r, s'\}$ 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 timestep, 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.

4.2.3 Training Algorithms

The main focus of this thesis is on implementing an ARE based on RL. However, we dedicate the next subsection to describe some of the advantages of using SL-based ARE including: (i) to motivate the feasibility of automating NOC actions, and (ii) to highlight the benefits of using SL (DL in particular) as a pre-training step for RL agents.

Deep Learning

In this work, we hypothesized that there are logical relationships between network problems and their remediation actions, even if those relationships are difficult to codify for complex networks. SL is therefore a good tool to model those relationships and take actions autonomously. The use of SL 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 Analytics[3], or Blue Planet Route Optimization and Analysis[4], 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.

We chose DL among other SL algorithms for two reasons: on one hand, DL models are more flexible, powerful, and known to outperform all traditional ML algorithms. On the other hand, we aim at studying the possibility of using SL to pre-train RL models. The idea is to define a NN model for an RL agent, and use DL to efficiently pre-train it on a time-series dataset that was collected and labeled by human experts. This form of training is called *Beauvoir cloning* [27]. In this framework, the agent receives as training data both the encountered states and actions of the demonstrator, and then uses a *classifier* to replicate the expert's policy (where each class represents an action). As we can see in Figure 4.4, the agent has 3 groups of actions that represent 12 classes: "Do Nothing", "Reroute Traffic" between 3 paths, and "Fix Network Issues" on 5 different links. The input features has 18 different performance metrics as detailed in §4.4.

Regarding network architecture, we used Feed-forward NN (FNN) and ReLU activation function to mimic the NN used by our RL agents. We trained the DL model using the standard back-propagation algorithm. One big challenge in our case is the problem of unbalanced classes; i.e., one class including more data points than other classes. This is typical because "Normal" state is the dominant network condition in reality and problems occur less often than Normal. To deal with this issue, we performed data augmentation (see §4.2.2) and used *Focal Loss* [20] that adds a weighting factor to the standard cross entropy criterion. This factor reduces the relative loss for well-classified examples, putting more focus on hard, misclassified examples.

[3]https://www.blueplanet.com/products/uaa.html
[4]https://www.blueplanet.com/products/route-optimization.html

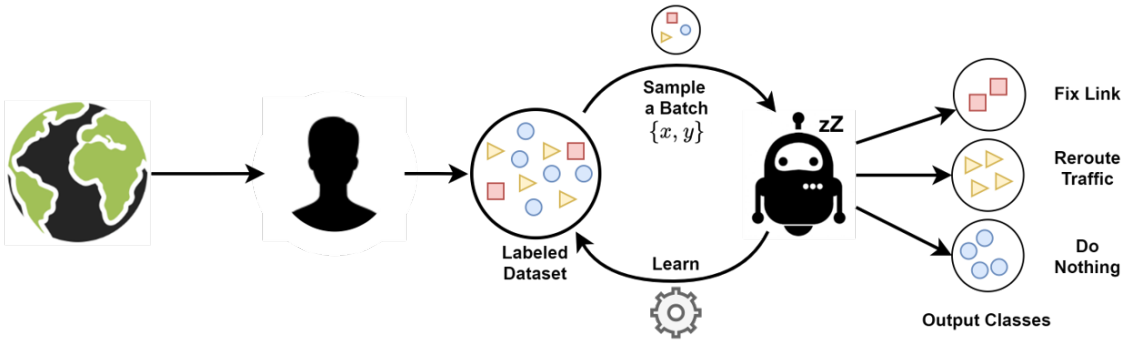


Fig. 4.4 A SL-based ARE trained on labeled dataset generated by human experts.

RL Algorithms

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 goes 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.

Double-DQN [31] is an improved version of the famous Deep Q-Network (DQN) algorithm. As the name implies, DQN uses DL to estimate the Q-function, a function used to estimate the expected value of each possible action given a specific state. Double-DQN uses two NN (see Figure 4.5) to avoid the large overestimation of action values seen in the original algorithm, which leads to poor performance in some stochastic environments. We chose to use the NN form of this algorithm as opposed to the 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 NNs). This makes it easier for us to compare the two algorithms. Finally, since DQN uses a simple FFN to estimate the Q-function, one can benefit from SL to quickly pre-train the DQN model using behaviour cloning as mentioned before.

When the agent interacts with the environment, the collected sequence of experience tuples can be highly correlated. This violates one of the fundamental requirements for a successful training: to have an independent and identically distributed (IID)

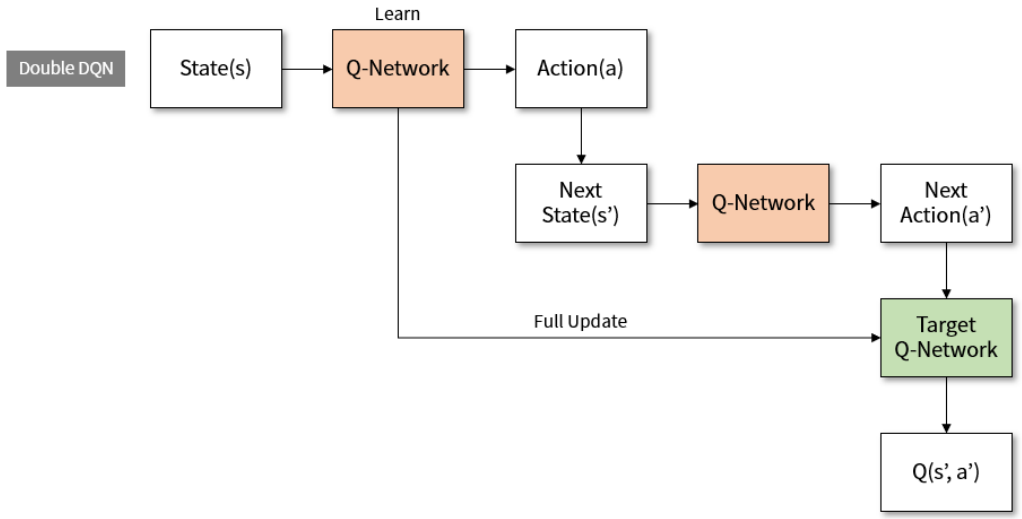


Fig. 4.5 Double Deep Q-Network algorithm. Orange block is the actor network, green block is the learner network. [5]

training data. To break this correlation, our RL agent build a "replay buffer" that contains a collection of past experience tuples (s, a, r, s'). The tuples are gradually added to the buffer during the interaction with the environment (online-step). During the offline-step, the agent randomly samples from the collected episodes in the buffer and learn. The buffer size represents agent's memory; as new data comes in, the oldest experiences are forgotten. In addition to breaking harmful correlations, experience replay allows RL agent to learn more from individual tuples multiple times, recall rare occurrences, and in general make better use of its experience.

In the Advantage Actor-Critic (A2C) algorithm [19], each agent is composed of a policy network (actor) and a value network (critic). The two networks interact and learn how to take actions as well as how to evaluate their effectiveness. The key idea in A2C is to update the *policy* network in the direction of increasing the accumulated reward. The size of the update step depends on the *value* of the advantage function calculated by the critic. To prevent the actor from over-fitting on a small portion of the environment (i.e., keep using the same actions), it is common to add an entropy regularization term to the loss function to encourage exploration.

We followed the standard Temporal Difference method [30] 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.

[5]Source: *https://greentec.github.io/reinforcement-learning-third-en/*

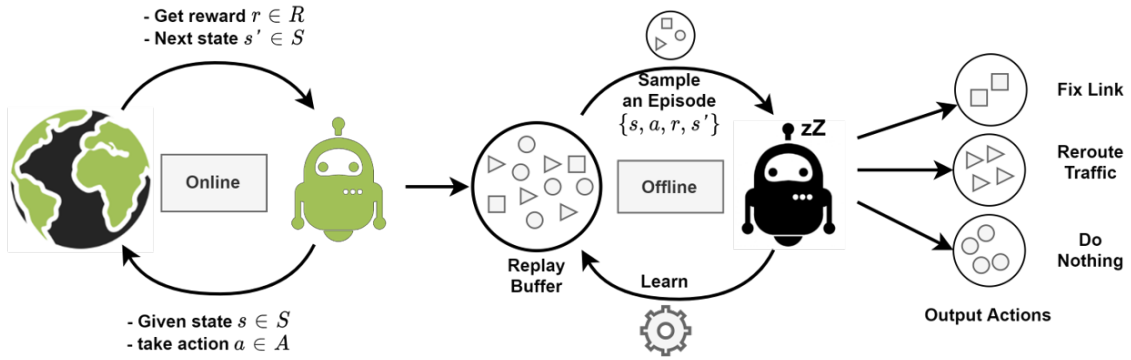


Fig. 4.6 RL-based ARE trained on unlabeled dataset generated continuously by the agent itself. The agent keeps switching between online (data collection) and offline (learning) steps.

It is worth noting that A2C algorithm has another popular variant called A3C. The two algorithms are similar in the sense that both are scalable. The two algorithms support training multiple agents in parallel by allowing each agent (actor) to sample its own experiences and push it to a shared buffer. A master model (the learner) then uses this data to improve and update the actors. A2C is a synchronous, deterministic implementation that waits for each actor to finish its segment of experience before performing an update, averaging over all of the actors. A3C is an asynchronous version (hence the name A3C). One advantage of A2C method is that it can more effectively utilize the GPUs and perform best with large batch sizes.

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 [4]. For example, in adaptive video streaming, [23] achieved state-of-the-art performance using an asynchronous version of it. And in a recent paper [22], the authors' RL approach outperforms the existing ABR expert rules in a week-long worldwide deployment with more than 30 million video streaming sessions.

4.3 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. We explore two different approaches to pre-train the agent: using SL and using Batch-RL.

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.

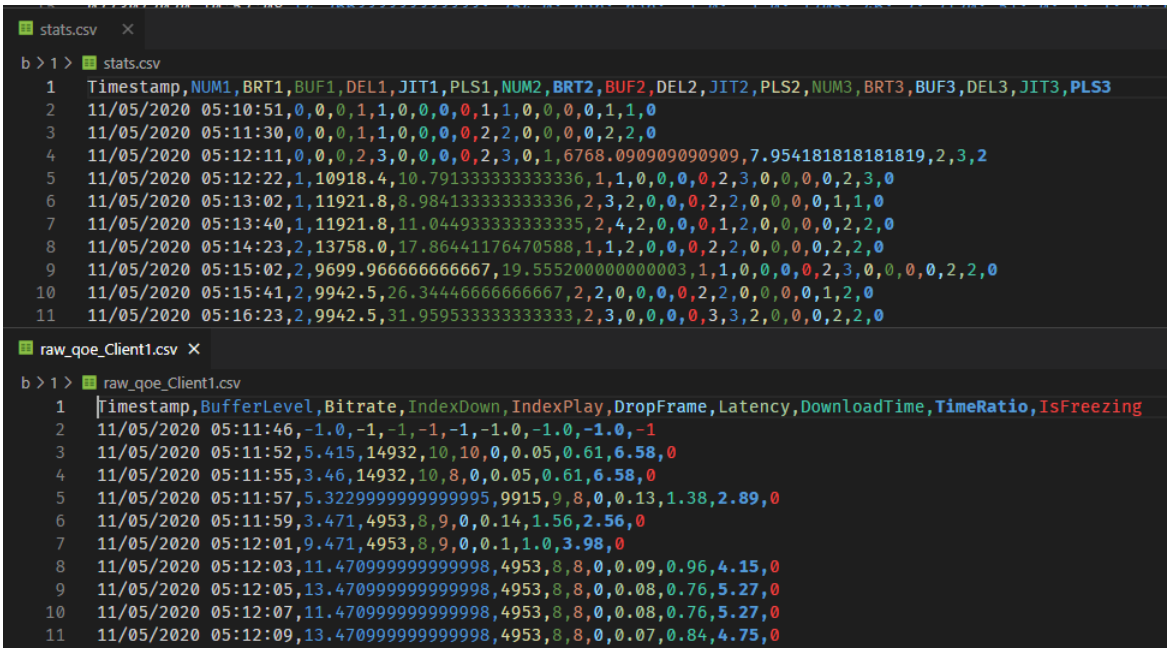
In the Simulator Environment, we first train the RL agent with non-zero exploration from the beginning. We then transfer the pre-trained agent from the simulator to the real network, validate that the agent's performance is as expected, then use it in production. In the Batch-RL Environment, the historical 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.



stats.csv

```
Timestamp,NUM1,BRT1,BUF1,DEL1,JIT1,PLS1,NUM2,BRT2,BUF2,DEL2,JIT2,PLS2,NUM3,BRT3,BUF3,DEL3,JIT3,PLS3
11/05/2020 05:10:51,0,0,0,1,1,0,0,0,0,1,1,0,0,0,0,1,1,0
11/05/2020 05:11:30,0,0,0,1,1,0,0,0,0,2,2,0,0,0,0,2,2,0
11/05/2020 05:12:11,0,0,0,2,3,0,0,0,0,2,3,0,1,6768.090909090909,7.954181818181819,2,3,2
11/05/2020 05:12:22,1,10918.4,10.791333333333336,1,1,0,0,0,0,2,3,0,0,0,0,2,3,0
11/05/2020 05:13:02,1,11921.8,8.984133333333336,2,3,2,0,0,0,2,2,0,0,0,0,1,1,0
11/05/2020 05:13:40,1,11921.8,11.044933333333335,2,4,2,0,0,0,1,2,0,0,0,0,2,2,0
11/05/2020 05:14:23,2,13758.0,17.86441176470588,1,1,2,0,0,0,2,2,0,0,0,0,2,2,0
11/05/2020 05:15:02,2,9699.966666666667,19.555200000000003,1,1,0,0,0,0,2,3,0,0,0,0,2,2,0
11/05/2020 05:15:41,2,9942.5,26.34446666666667,2,2,0,0,0,0,2,2,0,0,0,0,1,2,0
11/05/2020 05:16:23,2,9942.5,31.959533333333333,2,3,0,0,0,0,3,3,2,0,0,0,2,2,0
```

raw_qoe_Client1.csv

```
Timestamp,BufferLevel,Bitrate,IndexDown,IndexPlay,DropFrame,Latency,DownloadTime,TimeRatio,IsFreezing
11/05/2020 05:11:46,-1.0,-1,-1,-1,-1,-1.0,-1.0,-1.0,-1
11/05/2020 05:11:52,5.415,14932,10,10,0,0.05,0.61,6.58,0
11/05/2020 05:11:55,3.46,14932,10,8,0,0.05,0.61,6.58,0
11/05/2020 05:11:57,5.3229999999999995,9915,9,8,0,0.13,1.38,2.89,0
11/05/2020 05:11:59,3.471,4953,8,9,0,0.14,1.56,2.56,0
11/05/2020 05:12:01,9.471,4953,8,9,0,0.1,1.0,3.98,0
11/05/2020 05:12:03,11.470999999999998,4953,8,8,0,0.09,0.96,4.15,0
11/05/2020 05:12:05,13.470999999999998,4953,8,8,0,0.08,0.76,5.27,0
11/05/2020 05:12:07,11.470999999999998,4953,8,8,0,0.08,0.76,5.27,0
11/05/2020 05:12:09,13.470999999999998,4953,8,8,0,0.07,0.84,4.75,0
```

Fig. 4.7 Sample of collected dataset. Top: QoS metrics measured every 30 seconds. Bottom: QoE metrics measured every 2 seconds

4.4 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. Figure 4.7 shows a sample of QoS metrics as well as QoE metrics. Note that the QoS file contains the end-to-end metrics for all 3 paths. However, the QoE file represents the metrics for a single DASH client (i.e., there are 4 files in total). Moreover, the data in the QoS file are all fed to the RL agent. However, the agent doesn't observe the individual QoE metrics. Instead, the QoE metrics and averaged over a fixed time-interval (to match the rate in which IPSLA statistics were collected) and then aggregated per path. We also collected per-port metrics such as port packet loss, but we found that we didn't need them, and our ARE 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

Fig. 4.8 Sample of processed dataset. It includes state, action, rewards, and next state. As well as true information about the true environment states.

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), +5 (if one was streaming at 6000 Kbps while the other at 1000Kbps), or even 0 (if one client is streaming at 7000 kbps while the other is hanging.

Figure 4.8 shows the resulted dataset after applying the aforementioned pre-processing steps. As we can see, the file also contains the true state of the environment. This information is not available to the RL agent and is only being used for evaluation purposes. The main two pieces of information stored are (i) the root-cause of any problem and (ii) the routing-table that tells which client is using what path.

We tried training both SL and RL agent on this dataset in its current form, but the results were not very good. The main two problems with the data are: First, different metrics have different ranges that need to be normalized. For example, the range of possible bitrates are from $0 - 15000kbps$ while the range of video buffer is barely $0 - 15sec$. Such a huge difference between different metrics causes the gradient of the NN to explode. Second, different metrics have different meanings, and they follow different trends. For example, it is more convenient to have all metrics normalized in a way such that the larger the metrics the better (or vice versa). This is not the case with the current setup. As mentioned before, the larger the bitrate value the better, however, the opposite is true for the jitter. To handle those two issues, we normalized all metrics to have values between $(-1, 1)$ and multiplied delay, jitter, and packetloss with -1 such that 1 represents the best case.

Regarding the actions, we plot the percentage of each action in Figure 4.9. This is the result of more than 50 hours of data collection on GNS3. This dataset serves two purposes: on one hand, we would like to collect experiences from high-quality policies (such as expert rules) in GNS3 environment. This dataset serves as a bases for Batch-RL learning. On the other hand, we would like to encourage the agent to explore as many actions (good and bad) as possible in all states to better understand

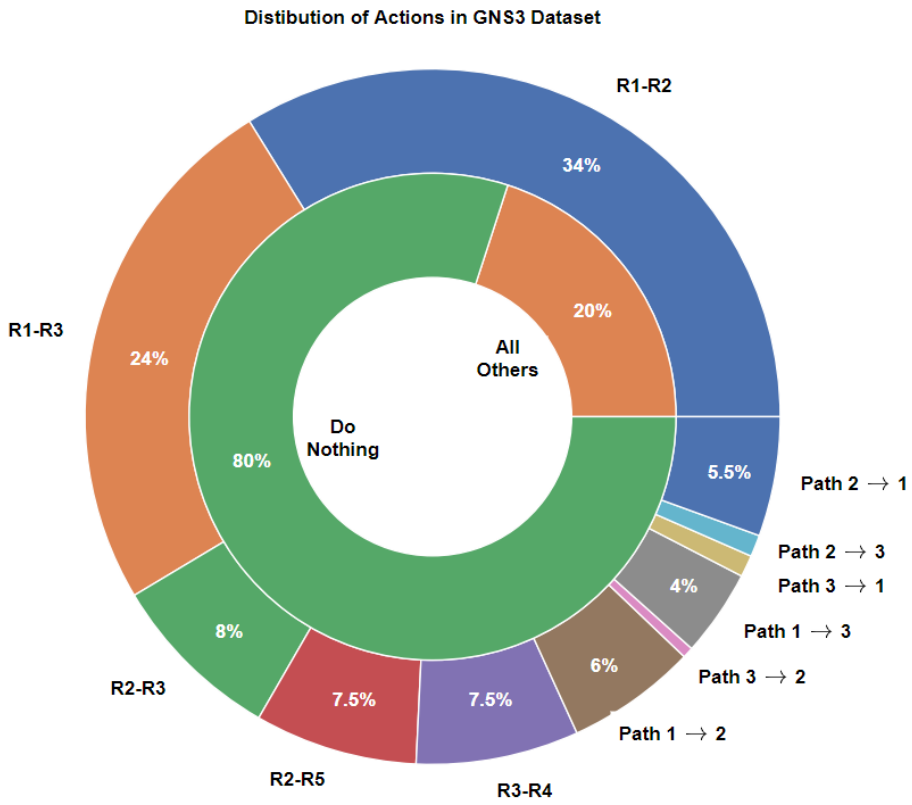


Fig. 4.9 Distribution of different classes withing GNS3 dataset. Inside: overall percentage. Outside: minority class breakdown.

the underlying dynamics of the environment. The resulting dataset serves as the bases for the Simulator environment, which in turn is capable of producing virtually an infinite amount of synthetic data.

Chapter 5

System Evaluation

In this section, we provide the evaluation metrics and results compared to related works, together with comments on results.

5.1 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.

In terms of SL neural network, we trained the same policy network mentioned above in a SL fashion to perform classification on a synthetic dataset collected from the Simulator Environment. We chose to train the policy network and not the value network because the former is the one actually used during inference.

As mentioned in §4.2.1, 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 2 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 4.1, 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.

5.2 Evaluation Metrics

To test the performance of our algorithms, we used *Gain* as defined in Equation 4.2 to compare the performance of RL-based methods and Confusion Matrix for DL-based method. Despite it is common to plot the "learning curves" as an indicator of system performance in SL, we don't show it here because it is not useful for RL. For example, DQN algorithm is considered as an "off-policy" algorithm, where one network is used to take the actions while the other is used for learning. The loss function used to train such an algorithm is different from the standard loss functions in two ways.

First, a loss function is usually defined on a fixed data distribution which is independent of the parameters we aim to optimize. This is not the case with RL since the data must be sampled on the most recent policy. Second, the loss function usually evaluates the performance metric that we care about. Here, we care about expected return, but our "loss" function does not approximate this at all. This "loss" function is only useful because, when evaluated at the current parameters, with data generated by

the current parameters, it has the negative gradient of the performance. However, after that first step of gradient descent, there is no more connection to the performance.

We raise this point because it is common for ML practitioners to interpret a loss function as a useful signal during training - "if the loss goes down, all is well." In policy gradients, this intuition is wrong, and we should only care about average return. The loss function means nothing.

To mimic NOC operator actions, we implemented the expert rules shown in Algorithm 1. 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 4.1.

Algorithm 1 Baseline

Input: State $\mathcal{S}$, Observations $\mathcal{O}$
Input: Paths($I_1 : R_{1,2,5}, I_2 : R_{1,2,4,5}, E_3 : R_{1,3,4,5}$)
Output Action
if an internal path I_j is congested **then**,
    $I_k \leftarrow$ Select(I_k s.t. $k \neq j$ **AND** I_k is not congested)
    client $\leftarrow$ Select(random client from path I_j)
    if path $I_k \neq \phi$ **then**
        action $\leftarrow$ Reroute(client to path I_k)
    else if external path E_3 is not congested **then**
        action $\leftarrow$ Reroute(client through E_3)
    else
        action $\leftarrow$ Do Nothing
    end if
else if Network issue **then**,
    action $\leftarrow$ fix the issue (if persisting for 3 time steps)
else if External path E_3 is being used **then**,
    $I_k \leftarrow$ Select(I_k s.t. I_k isn't congested)
    if $I_k \neq \phi$ **then**,
        action $\leftarrow$ Select(random client from path E_3)
        action $\leftarrow$ Reroute(client to path I_k)
    end if
else
    Do Nothing.
end if

5.3 Results

For the sake of illustration and ease of discussion, we first provide the results of our performance evaluation scenarios of four clients to show the behaviour of different solutions. We then focus on testing our system for larger number of users using the Simulator Environment. Unless mentioned otherwise, all results reported are the average of 10 runs of experiments.

5.3.1 Feasibility of NOC Automation

The first step in testing our ARE model is to show that ARE agent can match human performance. To do so, we trained the policy network in a SL fashion on a synthetic dataset collected from the Simulator. The training dataset consisted of about 500K example. We then tested the model on GNS3 Environment for 2000 steps (around 18 hours in real time).

In Figure 5.1 (right), we see that the agent almost learned to classify most actions correctly (i.e., confusion matrix is almost diagonal). However, some of the confusion between actions can be attributed to the fact that two different actions could lead to same end result. For example, if all 3 paths were occupied and a network issue appeared on the link R1-R2, then rerouting clients from Path 1 to either Path2 or 3 has the exact same cost. This is true because

- both tunnels have same number of hops. As a result, the costs of carrying the traffic in those paths are equal.
- the broken link affects only Path 1. Therefore, rerouting to either Path 2 or 3 will help the clients to stream videos at a good quality.
- since Path 3 is already occupied, the extra cost for using this tunnel is already paid. In this case, rerouting new clients to this path doesn't incur additional costs compared to Path 2.

To address this issue, we allowed the SL model to pick the top two best actions. If any of them matches the true label, we declare this action as correct. Figure 5.1(left) is a proof that SL-based ARE system can, to high-extend, match the performance of expert rules. Next, we'll show that it is possible to improve beyond this point and outperform the expert rules using RL.

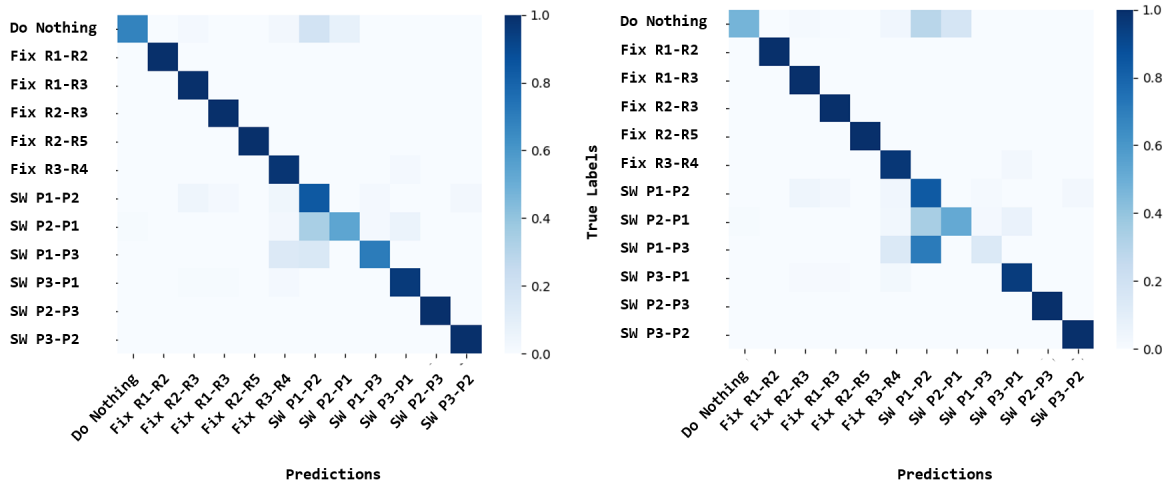


Fig. 5.1 Confusion matrix of SL-based ARE by taking into account top-k actions sorted by their probabilities. (right) $k = 1$ (left) $k = 2$

5.3.2 Outperform Expert-rules

Figure 5.2a 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 5.2a 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 agent in GNS3 for 18 hours. The results are shown in Figure 5.2b, 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 [4][22]. 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.

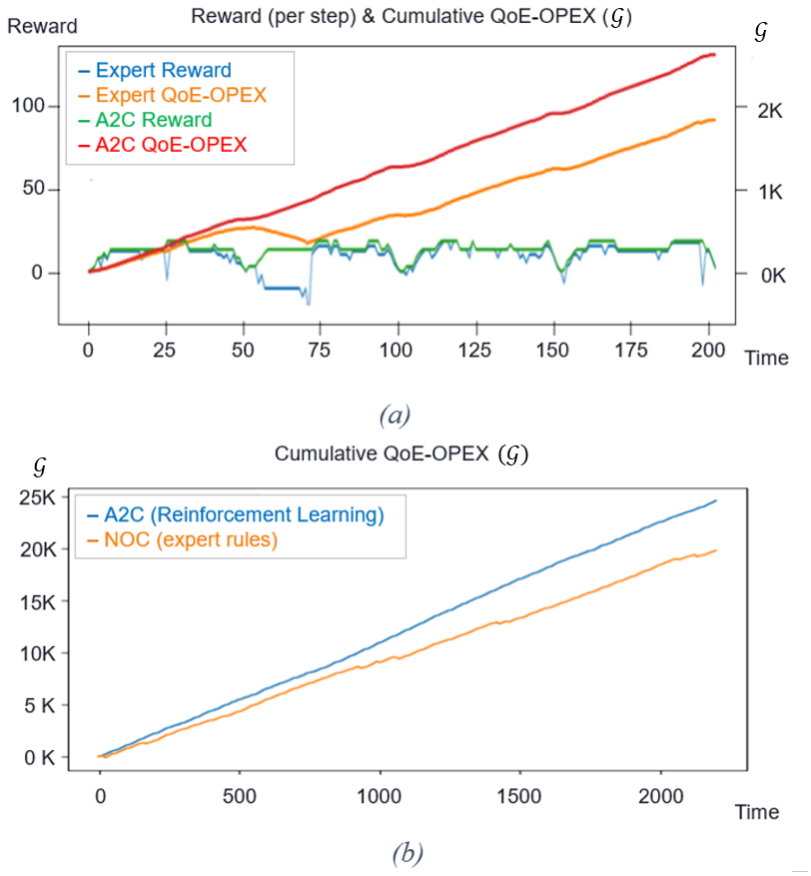


Fig. 5.2 ARE pre-trained using synthetic data on the Simulator Environment (not shown here) and tested on the GNS3 Environment. (a) Comparing both reward and gain. (b) testing the stability of ARE for 18 hours.

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.

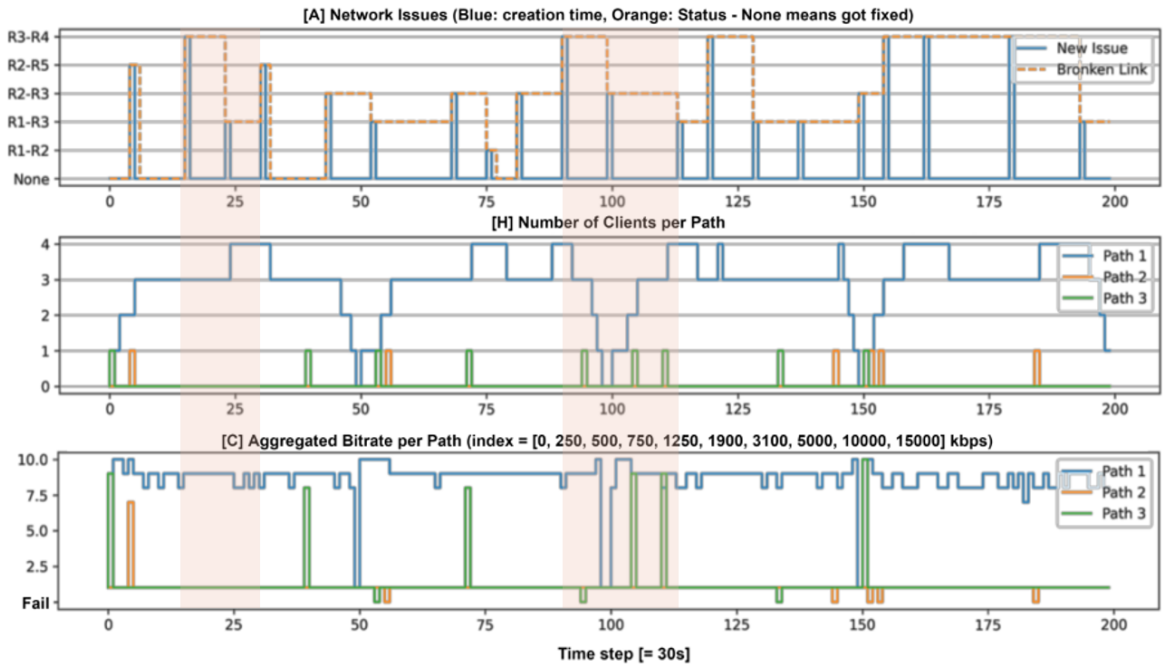


Fig. 5.3 RL agent learned to ignore issues that don't affect clients in order to minimize OPEX.

5.3.3 Detailed Analysis

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. The complete interactions of RL-Agent and expert rules with the environment are shown in Figure 5.6 and Figure 5.7 respectively. However, for better visualization, we'll reuse and highlight sections of them in the three subsequent figures used to illustrate three main strategies beyond the expert rules, and those strategies led to fewer but more effective actions as follows:

1. The RL-agent 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. This can be seen in Figure 5.3 around $t = 25$ and $t = 100$. As we can see in (a), two network issues affected Path 3 (R1-R3-R4-R5) around $t = 25$: first at the link connecting R3-R4 and then at R1-R3 (see blue line). Meanwhile, all clients were on path 1 as seen in (h). We can verify that no client was affected by observing (c) which shows that all clients were streaming at the highest bitrate. As a results, the agent ignored those issues and didn't attempt to fix them (orange line in (a) was not set to *None* indicating that the issue persisted).

2. It learned that it is better to pack all traffic on path 1 (R1-R2-R5) as much as possible, instead of load-balancing between path 1 and path 2 (R1-R3-R2-R5), 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. This is illustrated around $t = 50 - 75$ in Figure 5.4(b) where expert rules cause much fluctuations in the buffer size while ARE maintains the buffer sizes with more stability using fewer actions.

3. It paid particular attention to links that are shared between different paths. For example, link R2-R5 is shared among paths 1 and 2, while link R1-R3 is shared among paths 2 and path 3. As shown in Figure 5.5(a), ARE immediately set the orange line to *None* (i.e. normal state) every time an issue affect R2-R5 and never left the link broken. Furthermore, we can also see that ARE also rushes to reroute any client in Path 3 to avoid the costs of using AS3 infrastructure.

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.

5.3.4 Batch-RL Pre-training

To evaluate the Batch-RL Environment, we pretrained ARE using Batch-RL with an offline dataset that we created from running the expert rules on the Simulator Environment. 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 test, is that in this test case we are allowing ARE to continue training/exploring after step 400K

Fig. 5.4 RL agent achieved better service quality while taking fewer actions.

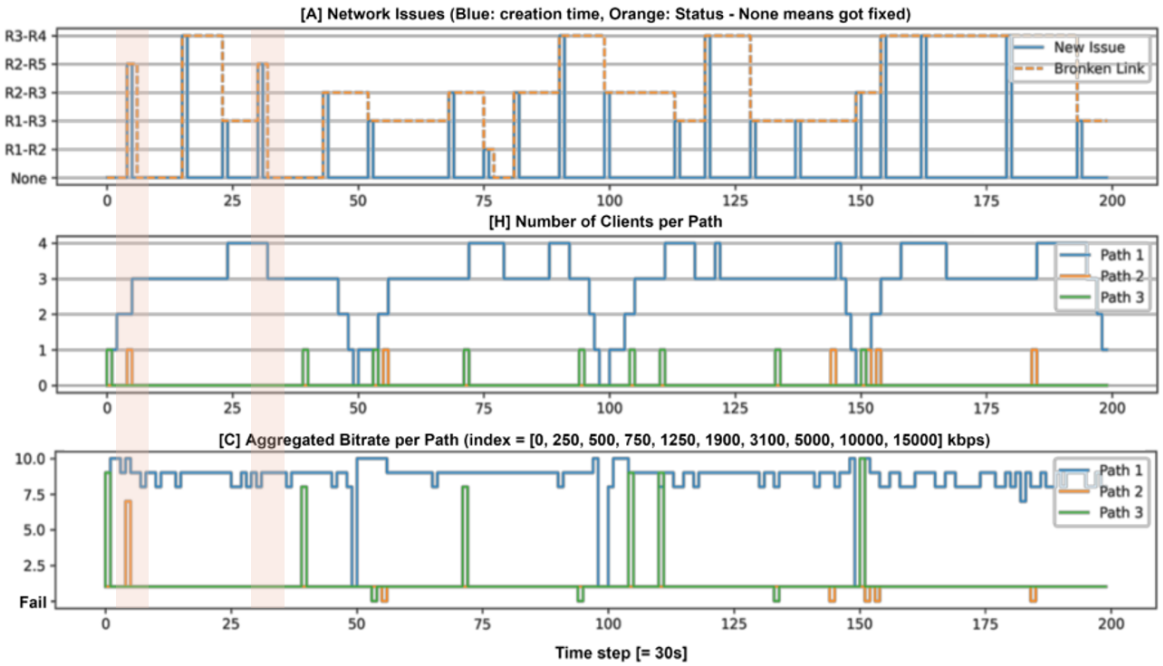


Fig. 5.5 RL agent learned to rush to fix links that are shared among multiple paths.

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 5.8. As expected, ARE achieves similar effectiveness as expert rules for the first 400K steps, because training with a labelled dataset is similar to SL, 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.

5.3.5 Hyper-parameter Tuning

Finally, to better understand the effect of network size on performance, we changed the number of layers and layers' width for both actor and critic as shown in Table 5.1. As we can see, A2C algorithm constantly performs better than expert-rules. We also note that a shallow FFN is enough to perform well. In general, training the critic seems to be a more difficult compared to the actor. This can be seen by the need for deeper and wider value networks compared to policy network. In other words, once the critic learns how to successfully evaluate the situation, it is easy for the actor to learn how to respond.

Fig. 5.6 Detailed performance of A2C algorithm tested on GNS3.

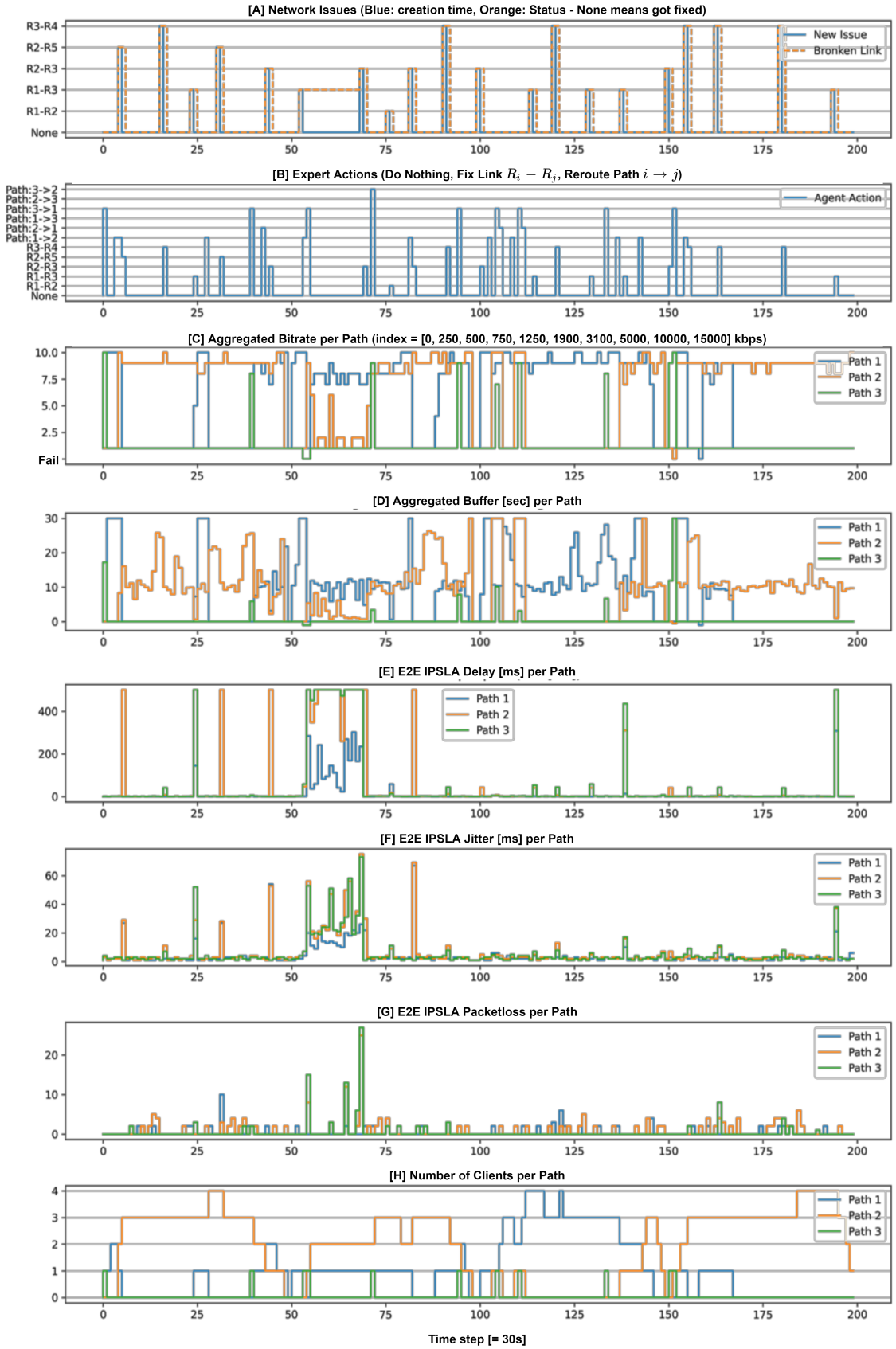


Fig. 5.7 Detailed performance of Baseline algorithm tested on GNS3.

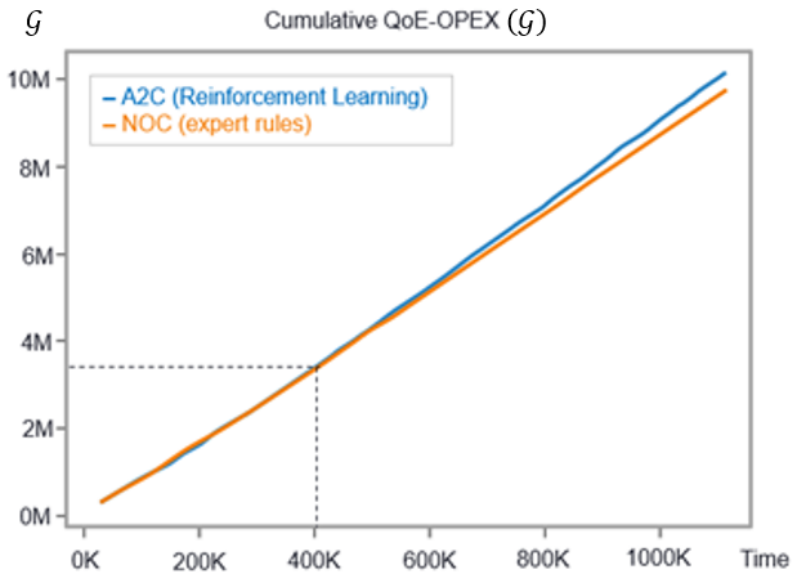


Fig. 5.8 ARE pre-trained on the Batch-RL environment using labelled data and tested on the Simulator environment.

The main reason behind this difficulty is the high fluctuation in the reward signal due to the adaptive bitrate of DASH clients. As we know, the value-network serves as a reward function approximater. However, DASH clients usually behave in a sub-optimal way resulting in huge fluctuation on the reward signal, making it difficult for the ARE to plan. Figure 5.9 illustrates the difficulty to predict the reward signal. As we can see around $t = 90 - 100$, the bitrate of a client on path3 keeps fluctuating between lowest and highest bitrate resulting in a series of traffic re-routing between path 1 and 3. This behaviour is not desired and can be further improved by allowing the clients and the RL agent to collaborate (see [2] for more on this approach).

So far, we have been focused on designing an algorithm that achieves the highest possible reward. However, it is also important to study the speed of convergence of such algorithms. In particular, among all A2C algorithms that outperform the expert-rule, we would like to find the fastest one to achieve that. Figure 5.10 compares the speed of which the QOE-OPEX metric increase for A2C algorithms with different network sizes. One interesting observation is that agents who have shallow value-network ($v_\pi = [256]$) and medium policy network ($\pi = [64, 64]$) seems to learn faster than others for the first 20K steps. However as Table 5.1 shows, this performance doesn't continue but falls 15% behind after 500K steps. In fact, the table shows that the model who have larger value-network and simpler policy network seems to be the winners at the end.

Table 5.1 Hyper-parameters Tuning with the Gain measured at 500K steps.

10 Models			Best Model	
V-Net	Pi-Net	Gain	test	Gain
[256,128,18]	[128,64,64]	4778144	1	5047411
[256]	[64]	4290899	2	4997272
[256]	[64, 64]	4326903	3	4997441
[256]	[64, 64, 64]	4370411	4	5005012
[256]	[128]	4329988	5	5001479
[256, 128]	[64]	4992534	6	5047915
[256, 128]	[64, 64]	4311737	7	5007532
[256, 128]	[128, 64]	4300530	8	5056497
[256, 128]	**[128]**	**5047411**	9	5008460
Expert	-	**3792901**	**Avg**	**5018779**

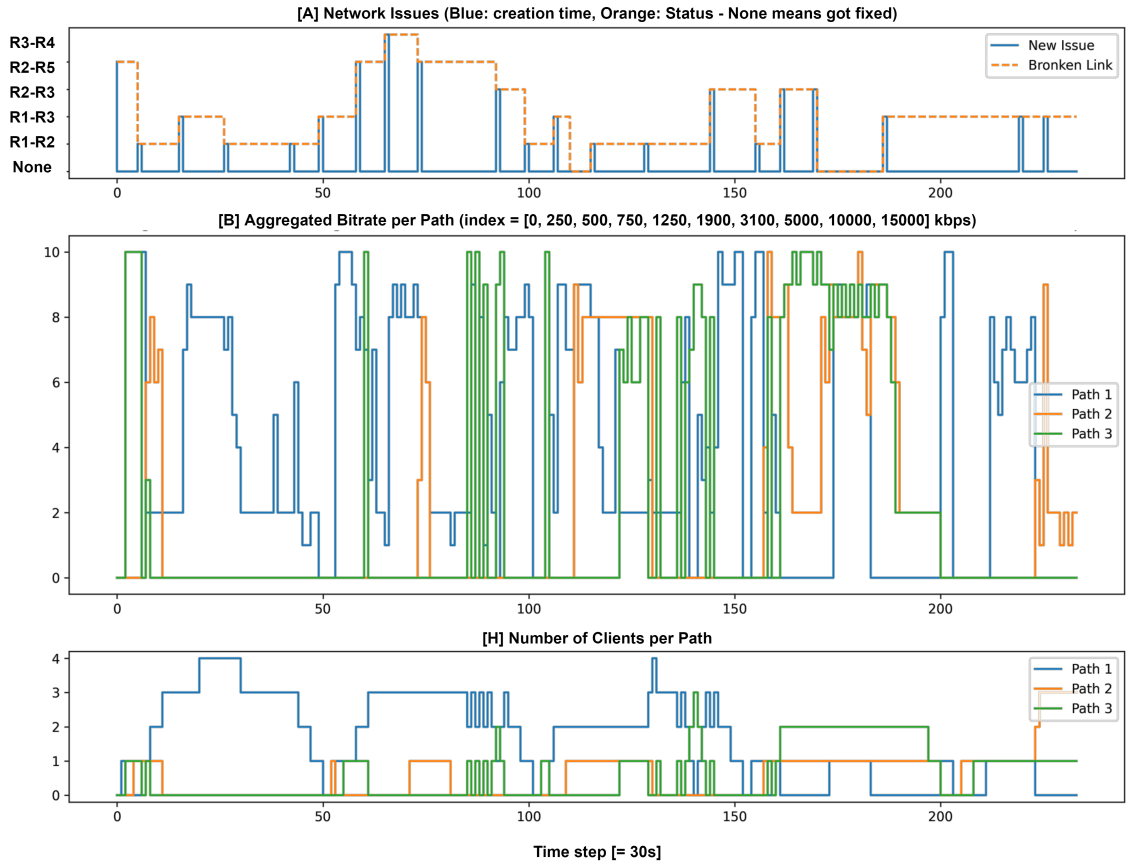


Fig. 5.9 DASH behaviour and the reward function. Baseline algorithm is running on GNS3

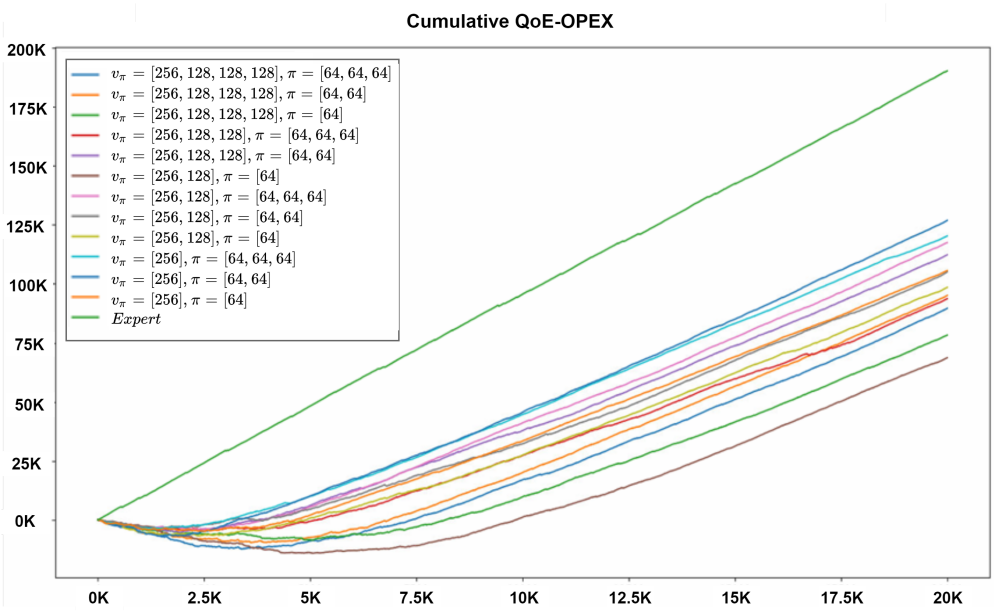


Fig. 5.10 Convergence speed of A2C algorithm for 20K steps on Simulator Environment.

Chapter 6

Conclusion

We studied the problem of automating ARE for NOCs and how to allow RL algorithms to safely discover new rules that can outperform human experts. In this concluding chapter, we summarize the contribution and achieved results and show possible directions for future research in this area.

6.1 Summary of the Results

We showed that using RL, it is possible to automate NOC operations with super-human 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. We summarize our contributions as follows:

- we formulated the problem of ARE in NOC as MDP process and solve this problem using RL exceeding human performance with expert rules.
- we proposed an objective function that goes beyond QoS-based goals to include ISP's OPEX and clients' QoE. Our RL algorithm was trained to anticipate network issues that might degrade the users' QoE and to act within the OPEX constraints.
- we designed 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. We equipped the simulator with MDP-aware data-augmentation capabilities that

enrich the collected dataset without changing the underlying dynamics of the real environment we are simulating.

- Our solution outperformed the human experts by a large margin of 25% on a challenging task (DASH video streaming) without the risk of making costly mistakes.

6.2 Future Works

For future research in the area of ARE for NOC using RL-based algorithms, a number of extensions can be further studied. On a network-level, one idea is to extend this notion of "ARE for layer-3 networks" to span other layers in the OSI model. In particular, the physical layer and application layer. By doing so, ARE will respond better to the need of the application layer through collaboration with DASH clients, and it will have better observability and flexibility over the backbone (fiber network) that carries the whole traffic.

Another idea, is automate the network service monitoring similar to the ones used in ZSM [29]. In this regard, the RL-agent is allowed to decide on what KPI metrics to collect and at what rate. This is important because collecting all type of possible metrics at high-rate results in huge overhead on the network. On the other side, deciding what KPI to collect depends on the QoE metric we want to optimize, and this metric can change over time.

On algorithm-level, one idea is to study the feasibility of performing "transfer learning" in an RL setup. In particular, given an RL agent that was trained on a specific network topology to automate certain NOC actions. Define scenarios, environments (i.e. network typologies), tasks, and potential representations to be transferred between a source domain and a target domain. And test with other alternatives to Batch-RL.

Finally, the results of this work are very promising. The next immediate step would be to deploy and test our system in a real setting.

References

[1] Ahmad, A., Floris, A., and Atzori, L. (2017). Ott-isp joint service management: a customer lifetime value based approach. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 1017–1022. IEEE.

[2] Altamimi, S. and Shirmohammadi, S. (2020). Qoe-fair dash video streaming using server-side reinforcement learning. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 16(2s):1–21.

[3] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

[4] Chu, T., Wang, J., Codecà, L., and Li, Z. (2019). Multi-agent deep reinforcement learning for large-scale traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, 21(3):1086–1095.

[5] Ciena, I. (2018). Introducing the adaptive network vision. *Ciena White Paper*.

[6] Cisco, I. (2016-2021). Cisco visual networking index: Forecast and methodology. *Cisco White Paper*.

[7] De Vriendt, J., De Vleeschauwer, D., and Robinson, D. (2013). Model for estimating qoe of video delivered using http adaptive streaming. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 1288–1293. IEEE.

[8] Deb, S., Ge, Z., Isukapalli, S., Puthenpura, S., Venkataraman, S., Yan, H., and Yates, J. (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, page 1783–1792, New York, NY, USA. Association for Computing Machinery.

[9] Dey, A. J. and Sarma, H. K. D. (2020). Routing techniques in internet of things: A review. In Sarma, H. K. D., Bhuyan, B., Borah, S., and Dutta, N., editors, *Trends in Communication, Cloud, and Big Data*, pages 41–50, Singapore. Springer Singapore.

[10] Dimopoulos, G., Leontiadis, I., Barlet-Ros, P., Papagiannaki, K., and Steenkiste, P. (2015). Identifying the root cause of video streaming issues on mobile devices. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '15, New York, NY, USA. Association for Computing Machinery.

[11] Dinaki, H. E., Shirmohammadi, S., Janulewicz, E., and Côté, D. (2020). Deep learning-based fault localization in video networks using only client-side qoe. *IEEE Transactions on Artificial Intelligence*, 1(2):130–138.

[12] Gallego-Madrid, J., Sanchez-Iborra, R., Ruiz, P. M., and Skarmeta, A. F. (2021). Machine learning-based zero-touch network and service management: A survey. *Digital Communications and Networks*.

[13] Ghannami, A. and Shao, C. (2016). Efficient fast recovery mechanism in software-defined networks: Multipath routing approach. In *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 432–435.

[14] Gill, P., Jain, N., and Nagappan, N. (2011). Understanding network failures in data centers: Measurement, analysis, and implications. *SIGCOMM Comput. Commun. Rev.*, 41(4):350–361.

[15] Hemmati, M., McCormick, B., and Shirmohammadi, S. (2017). Qoe-aware bandwidth allocation for video traffic using sigmoidal programming. *IEEE MultiMedia*, 24(4):80–90.

[16] IEEE (2021). Ethically aligned design for business. *The IEEE Global Initiative on Ethics of Autonomous and Intelligent Systems*.

[17] Kanagavelu, R. and Zhu, Y. (2019). A pro-active and adaptive mechanism for fast failure recovery in sdn data centers. In Arai, K., Kapoor, S., and Bhatia, R., editors, *Advances in Information and Communication Networks*, pages 239–257, Cham. Springer International Publishing.

[18] Kearns, M., Mansour, Y., and Ng, A. Y. (2002). A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine learning*, 49(2):193–208.

[19] Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014.

[20] Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2017). Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988.

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

[22] Mao, H., Chen, S., Dimmery, D., Singh, S., Blaisdell, D., Tian, Y., Alizadeh, M., and Bakshy, E. (2020). Real-world video adaptation with reinforcement learning. *arXiv preprint arXiv:2008.12858*.

[23] Mao, H., Netravali, R., and Alizadeh, M. (2017). Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 197–210.

[24] Mohammed, A. R., Mohammed, S. A., Côté, D., and Shirmohammadi, S. (2021a). Machine learning-based network status detection and fault localization. *IEEE Transactions on Instrumentation and Measurement*, 70:1–10.

[25] Mohammed, S. A., Mohammed, A. R., Côté, D., and Shirmohammadi, S. (2021b). A machine-learning-based action recommender for network operation centers. *IEEE Transactions on Network and Service Management*, pages 1–1.

[26] Raeisi, B. and Giorgetti, A. (2016). Software-based fast failure recovery in load balanced sdn-based datacenter networks. In *2016 6th International Conference on Information Communication and Management (ICICM)*, pages 95–99.

[27] Ross, S. and Bagnell, D. (2010). Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668. JMLR Workshop and Conference Proceedings.

[28] Sandvine, I. (2020). Global internet phenomena report. *North America and Latin America*.

[29] Saraiva, N., Lachos, D., Rothenberg, C. E., and Gomes, P. H. (2021). End-to-end service monitoring for zero-touch networks. *Journal of ICT Standardization*, vol9(Iss2).

[30] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

[31] Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. 30(1).

[32] Wang, S., Xu, H., Huang, L., Yang, X., and Liu, J. (2019). Fast recovery for single link failure with segment routing in sdns. In *2019 IEEE 21st International Conference on High Performance Computing and Communications*, pages 2013–2018.

[33] Zhang, Y., Xin, J., Li, X., and Huang, S. (2020). Overview on routing and resource allocation based machine learning in optical networks. *Optical Fiber Technology*, 60:102355.