Andriy Mazayev

Event Processing in Web of Things

2021

Andriy Mazayev

Event Processing in Web of Things

PhD Thesis in Computer Science

Work done under the supervision of:
Prof[a]Dr[a]Noélia Correia

2021

Statement of Originality

Event Processing in Web of Things

Declaração de autoria de trabalho: Declaro ser o autor deste trabalho, que é original e inédito. Autores e trabalhos consultados estão devidamente citados no texto e constam da listagem de referências incluída.

Candidato:

_____

(Andriy Mazayev)

Work done at Research Center of Electronics Optoelectronics and Telecommunications (CEOT)

I would like to thank my parents for their encouragement, education, for their belief in me and for their effort to make my graduation possible.

Special thanks to all my friends that have been by my side since the beginning of this journey. Thanks for your support, all the funny moments that we lived and just for being there for me when I needed. Without you nothing of this would have been possible.

In academic environment I would like to thank my supervisor Prof. Dr. Noélia Correia and my good colleague Dr. Faroq AL-Tam. Without their support and guidance I would not have been able to finish this project. Their experience and expertise were invaluable.

THANK YOU!

Acknowledgements

The research published in this work was supported by the Portuguese Foundation for Science and Technology (FCT) through CEOT (Center for Electronic, Optoelectronic and Telecommunications) funding (UID/MULTI/00631/2020) and by FCT Ph.D grant to Andriy Mazayev (SFRH/BD/138836/2018).

Abstract

The incoming digital revolution has the potential to drastically improve our productivity, reduce operational costs and improve the quality of the products. However, the realization of these promises requires the convergence of technologies — from edge computing to cloud, artificial intelligence, and the Internet of Things — blurring the lines between the physical and digital worlds. Although these technologies evolved independently over time, they are increasingly becoming intertwined. Their convergence will create an unprecedented level of automation, achieved via massive machine-to-machine interactions whose cornerstone are event processing tasks.

This thesis explores the intersection of these technologies by making an in-depth analysis of their role in the life-cycle of event processing tasks, including their creation, placement and execution. First, it surveys currently existing Web standards, Internet drafts, and design patterns that are used in the creation of cloud-based event processing. Then, it investigates the reasons for event processing to start shifting towards the edge, alongside with the standards that are necessary for a smooth transition to occur. Finally, this work proposes the use of deep reinforcement learning methods for the placement and distribution of event processing tasks at the edge. Obtained results show that the proposed neural-based event placement method is capable of obtaining (near) optimal solutions in several scenarios and provide hints about future research directions.

Keywords: Web standards, Internet of Things, Web of Things, event processing, edge computing, load balancing, resource placement, deep neural networks, deep reinforcement learning.

Resumo

A nova revolução digital promete melhorar drasticamente a nossa produtividade, reduzir os custos operacionais e melhorar a qualidade dos produtos. A concretização dessas promessas requer a convergência de tecnologias – desde *edge computing* à *cloud*, inteligência artificial e Internet das coisas (IoT) – atenuando a linha que separa o mundo físico do digital. Embora as quatro tecnologias mencionadas tenham evoluído de forma independente ao longo do tempo, atualmente elas estão cada vez mais interligadas. A convergência destas tecnologias irá criar um nível de automatização sem precedentes. Este processo será alcançado por meio de interações *machine-to-machine*, cujo elemento fundamental são as tarefas para processamento de eventos (*event processing tasks*). Uma *event processing task* consiste num processo de três fases: observação do estado dos dispositivos, análise das alterações dos mesmos e, caso uma condição específica seja satisfeita, envio de comandos para a realização de uma tarefa específica por parte de outro(s) dispositivo(s).

Esta dissertação estuda o papel das tecnologias acima mencionadas no ciclo de vida das *event processing tasks*, desde sua criação, colocação e execução. O pilar das *event processing tasks* é a interoperabilidade entre os dispositivos, atualmente um tópico central do IoT, que está, neste momento, numa fase de uniformização e padronização. O sucesso e a ubiquidade do protocolo de Internet (IP) e do paradigma *representational state transfer* (REST), pilares da Web "normal", chamaram a atenção da indústria e dos investigadores, levando-os a querer trazer os mesmos mecanismos para os dispositivos com reduzida capacidade de processamento. O objetivo é tornar a interação com dispositivos tão simples como uma interação com uma página Web, criando assim a Web das coisas.

Uma das organizações que lidera este esforço chama-se *Internet Engineering Task Force* (IETF), que trouxe protocolos como IPv6 e o protocolo *contrained application protocol* (CoAP) que, tal como *hypertext transfer protocol* (HTTP) na Web "regular", permite criar um modelo de interação e comunicação entre serviços seguindo a arquitetura REST. Seguindo os passos do IETF, organizações como a *World Wide Web Consortium* (W3C), open connectivity foundation (OCF), entre outras, começaram a definir a forma explícita como os dispositivos devem expor informações sobre si e indicar como é possivel interagir com eles, isto é, começou a desenvolver-se o modelo de dados (*data model*) para a interação. Contudo, é importante referir que o processo de padronização ainda está na sua infância e que, de momento, não existe nenhum modelo de dados universalmente

aceite por todos os produtores de dispositivos. No momento em que a aceitação ocorrer, e surgir um padrão *de facto*, será possível ter uma aplicação capaz de controlar milhões de dispositivos, tal como acontece na Web, onde um único navegador (*browser*) consegue interagir com milhões de páginas Web.

Este trabalho tem como objetivo realizar uma pesquisa aprofundada dos *design patterns* existentes, padrões e conceitos da Web que podem ser usados na criação das *event processing tasks*. A reutilização de *design patterns* bem conhecidos e comprovados é importante, pois evita a criação de soluções proprietárias de processamento de eventos, que representam uma barreira de interoperabilidade. O objetivo final é investigar como as propostas e padrões atuais podem ser usados na criação das *event processing tasks* distribuídas.

Contudo, a criação das *event processing tasks* é apenas o primeiro passo, o passo seguinte é a sua colocação e execução. Para tal torna-se necessário ter uma infraestrutura que consiga fornecer a computação e armazenamento. Até muito recentemente, esta tarefa foi relegada para a *cloud computing*. No entanto, a *cloud computing* não é capaz de satisfazer os requisitos de novos aplicativos IoT, nomeadamente: sensibilidade a atrasos, custos de uplink, privacidade, segurança e não interrupção de serviços em ambiente com conectividade intermitente. Por sua vez, o *edge computing* que traz computação e armazenamento para perto dos *end-devices*, terá um papel importante no processamento de eventos. Este trabalho tem como objetivo investigar que mecanismos e protocolos podem ser utilizados por dispositivos *edge* para expor as suas capacidades computacionais, anunciar a sua presença, bem como o seu desejo de participar no processamento de eventos.

Outro foco deste trabalho é a otimização, que irá ter um papel importante no processo de distribuição das *event processing tasks* entre os dispositivos. Dada a natureza heterogénea dos dispositivos *edge* e os diferentes requisitos dos aplicativos IoT, o posicionamento das *event processing tasks* deve ser feito de acordo com os padrões de tráfego, satisfazendo ao mesmo tempo os requisitos das aplicações cliente. Este trabalho investiga as estratégias de posicionamento necessárias para que o processamento distribuído de eventos aconteça de forma eficiente, tomando em consideração as restrições relativas ao IoT tais como: latência, recursos computacionais e custos de operação do dispositivo.

Por fim, este trabalho investiga também a utilização de técnicas de *machine learning* (ML) durante o processo de distribuição das *event processing tasks*. A necessidade de utilização de ML surge como consequência da evolução e cresimento rápido das redes de comunicação, facto que leva ao surgimento de requisitos para os quais não existem estratégias/heurísticas de distribuição que tenham bom desempenho. As abordagens clássicas e manuais consomem muito tempo e requerem experiência na área de modo a terem bom desempenho. Por outro lado, métodos ML que, sem muita engenharia, oferecem a possibilidade de descobrir novas estratégias de distribuição parecem ser uma escolha atraente para lidar com tais cenários [1]. Avanços recentes em *deep learning* (DL)

mostram um desempenho notável em tarefas como reconhecimento de imagem [2] ou processamento de linguagem natural (NLP) [3]. Além disso, o *deep learning* em combinação com o *reinforcement learning* (RL), habitualmente conhecido como *deep reinforcement learning* (DRL), está a mostrar resultados promissores em problemas de tomada de decisão como jogos de tabuleiro [4, 5], condução [6], robótica e muito mais [7]. Os resultados obtidos neste trabalho mostram que as redes neurais são capazes de obter soluções ótimas (ou próximas do ótimo) em vários cenários estudados.

Palavras-chave: Padrões da Web, Internet das coisas, Web das coisas, processamento de eventos, *edge computing*, balanceamento de carga, colocação de recursos, rede neuronais, *deep reinforcement learning*.

Contents

List of Figures

List of Tables

Nomenclature

6LoWPAN	IPv6 over Low power WPAN
A2C	Advantage Actor Critic
ACK	Acknowledgement
API	Application Programming Interface
CBOR	Concise Binary Object Representation
CIM	Context Information Management
CoAP	Constrained Application Protocol
CoRE	Constrained RESTful Environment
CPLEX	IBM ILOG CPLEX Optimization Studio
CPU	Central Process Unit
CRUDN	Create, Read, Update, Delete and Notify
DL	Deep Learning
DNN	Deep Neural Network
DPWS	Devices Profile for Web Services
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
DTDL	Digital Twin Definition Language
DTLS	Datagram Transport Layer Security
DynLink	Dynamic Linking
EoS	End of Sequence
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IFTTT	If-This-Then-That
IoT	Internet of Things
IP	Internet Protocol
IPv6	Internet Protocol version 6
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
LLN	Low-Power and Lossy Network
LSTM	Long Short-Term Network
LWM2M	Lightweight Machine-2-Machine

M2M	Machine-2-Machine
MDP	Markov Decision Process
MEC	Mobile Edge Computing
ML	Machine Learning
MQTT	Message Queueing Telemetry Transport
MTU	Maximum Transferable Unit
NLP	Natural Language Processing
OFC	Open Connectivity Foundation
oneDM	One Data Model
oneM2M	One Machine-2-Machine
OPDPG	Off-Policy Deterministic Policy Gradient
OSCORE	Object Security for Constrained RESTful Environments
Ptr-Net	Pointer Network
RAM	Random Access Memory
RD	Rule Distribution
REST	Representational State Transfer
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RP	Rule Placement
RPL	Routing Protocol for Low-Power and Lossy Networks
SenML	Sensor Measurement Lists
seq2seq	Sequence-2-Sequence
SL	Supervised Learning
SMCP	Simple Monitoring and Control Protocol
SOM	Splot Object Model
SoS	Start of Sequence
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TSP	Traveling Salesman Problem
UE	User Equipment
URI	Uniform Resource Identifier
VRP	Vehicle Routing Problem
W3C	World Wide Web Consortium
WG	Working Group
WoT	Web of Things

List of Publications

1. Mazayev, Andriy, Al-Tam, Faroq and Noélia Correia. “Attention-Based Model and Deep Reinforcement Learning for Distribution of Event Processing Tasks”, Elsevier Internet of Things Journal (Submitted).

2. Mazayev, Andriy, and Noélia Correia. “A Distributed CoRE-Based Resource Synchronization Mechanism”, IEEE Internet of Things Journal 7.5 (2019): 4625-4640, DOI: http://dx.doi.org/10.1109/jiot.2019.2958375

3. Mazayev, Andriy, Jaime A. Martins, and Noelia Correia. “Interoperability in IoT through the Semantic Profiling of Objects”, IEEE Access 6 (2017): 19379-19385, DOI: https://doi.org/10.1109/ACCESS.2017.2763425

4. Mazayev, Andriy, Jaime A. Martins, and Noélia Correia. “Semantically enriched hypermedia APIs for next generation IoT”, International Conference on Interoperability in IoT (2017): 19-26, DOI: https://doi.org/10.1007/978-3-319-93797-7_3

5. Mazayev, Andriy, Jaime A. Martins, and Noélia Correia. “Semantic Web Thing Architecture”, IEEE Experiment@ International Conference (2017): 43-46, DOI: https://doi.org/10.1109/EXPAT.2017.7984368

CHAPTER 1

Introduction

***Abstract*:** This chapter presents the technological background for this thesis by introducing the ingredients necessary to enable the ad hoc creation and placement of event processing tasks at heterogeneous devices. Furthermore, it presents the motivation and outlines the research goals of this work.

1.1 Background

The Internet of Things (IoT) is the perfect representation of Mark Weiser's vision of profound technologies: devices with processing and communication capabilities that are embedded into the fabric of everyday life at a level that they vanish into the background [12]. Mark Weiser first coined this concept in 1991 and has, since then, turned into reality. We are no longer amazed by the smart lights nor watches capable of tracking our location, steps and heart rate. Checking, in real-time, the weather, pollution levels or traffic of a place located on the other side of the globe became a banality. These are just a few examples of what the advances in microelectronics and miniaturization have brought. Nowadays, millions of constrained devices with wireless communication capabilities are measuring the environment and sending this information to the Internet. According to CISCO, in 2021 IoT devices will generate around 850 zettabytes of data and this number will continue to grow [13]. We are already starting to struggle to integrate and process such massive amounts of data generated by heterogeneous devices. The struggle is also exacerbated by device manufacturers that build their entire business models on top of proprietary communication protocols, data models and closed source applications. As a result, the IoT environment is now highly fragmented. Figure 1.1 is a good illustration of the current state of IoT. It has stumbled into vertical data silos, i.e., closed vertical systems, designed for a single purpose, that are hard to integrate with other systems from other domains.

Until recently, the integration of multiple devices into a single application, in particular when different vendors are involved, often required hand crafted solutions, which is an expensive and time consuming process. Additionally, providing continuous support (e.g.,

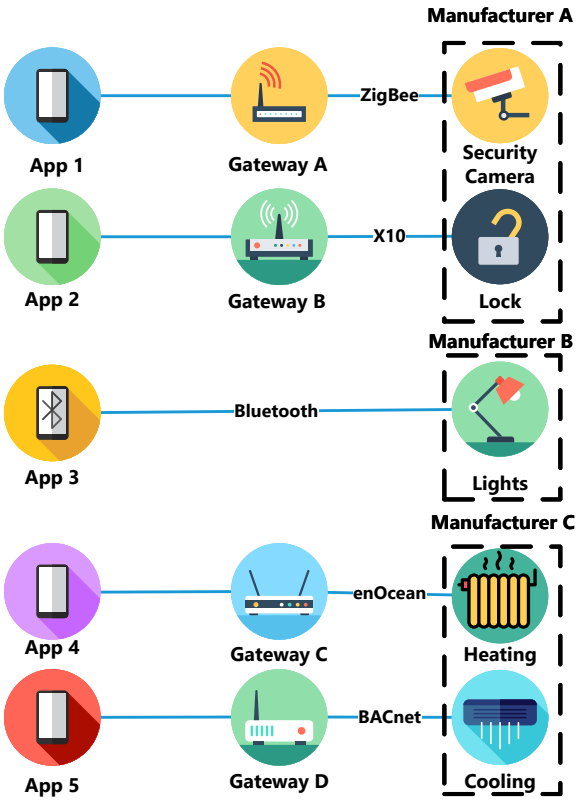


Figure 1.1: The current fragmented state of IoT. One-app to one-device relationship (adapted from [8]).

bug fixing, migration to newer API versions) involves extremely high costs, an expense that often does not receive the necessary attention.

To put the cost of fixing a bug in a perspective, consider a 53$ hourly rate which, according to the U.S. Bureau of Labor Statistics, corresponds to the average value charged by a software engineer in 2020 [14]. Moreover, consider the findings reported by Jeff Sutherland, co-creator of Scrum, who stated that, regardless of the project, fixing any software bug three weeks after its detection takes 24 times longer than if it was fixed on the same day that it was created [15]. The time increase is mainly due to the need to remember why the code was written the way it was written, isolating the bug and fixing it. Providing long-term support for an integration can be seen as the process of solving old bugs, hence the extremely high costs. As an example, a bug that takes 1 hour to solve, but is found 3 weeks later, has a cost of 1.272$. Finding one bug per month equals to 15.264$ in yearly expenses.

In order to mitigate the above mentioned limitations and reduce the maintenance costs, IoT practitioners introduced the Web of Things (WoT) concept, which aims to provide a set of uniform and standardized interfaces (at the Application layer of the TCP/IP model) to address the interoperability between heterogeneous devices. Inspired by, and based on, the experience gained from the development of the World Wide Web, WoT pioneers like

the Internet Engineering Task Force (IETF), World Wide Web Consortium (W3C), open connectivity foundation (OCF), one data model (oneDM) group, connectivity standards alliance (CSA), among other, started to work on the standardization of the architecture and data model design for IoT systems, which includes REST API design, M2M protocol integration and hypermedia controls. The outcomes of these working groups resulted in several proposals such as WoT Thing description [16], WoT binding templates [17], OCF core specification [18], IoT schema [19], the matter project [20], oneM2M [21], oneDM semantic definition format [22], Azure digital twin definition language (DTDL) [23], lightweight M2M (LWM2M) [24], object linking and embedding for process control unified architecture (OPC UA) [25]. As part of a research background for this thesis, some of these proposals were reviewed and analyzed, work that resulted in three publications [26], [27], [28].

Generally speaking, despite their differences, the above mentioned research groups agreed on a common set of design patterns in their architectures: IPv6, REST APIs to interact with the devices (although other interaction models are also supported) and three main classes to describe their resources and capabilities. These classes correspond to device's:

- State values that can be read or set (e.g., temperature value)
- Control entry points to invoke a function (e.g., open a garage door)
- Asynchronous signals that push notifications to the data consumers (e.g., overheating events)

While there is an agreement on these base concepts, there is still no common way of describing them. At the moment of writing, there is an ongoing battle between several competing standards backed by several consortia. Much like happened with the VHS versus Betamax (won by VHS), Blu-ray versus HD DVD (won by Blu-ray) or protocol wars that occurred between the 70s and 90s (won by TCP/IP protocol set), each consortia introduces its own data model, vocabulary and data structure to describe the devices. Nevertheless, these efforts are very important because interoperability between the devices is not enabled by simply agreeing to use JavaScript object notation (JSON) and hypertext transfer protocol (HTTP), or any other application protocol and data exchange format. For example, JSON can be structured in an infinite number of ways while transmitting exactly the same information. HTTP POST can be used for several different purposes. Automated device discovery, interaction and machine-to-machine communication requires strict, structured, machine readable and self-described interface definitions [29]. Therefore, true machine-to-machine communications and seamless integration will occur when device manufacturers reach an agreement and start using a common data model, which will become *de facto* standard. Once it happens, a single application will be able to control

several devices produced by different manufacturers, as illustrated in Figure 1.2. In addition to the ease of use, this approach will also bring benefits to the developers. Instead of learning specialized, complex, and difficult to use protocols (e.g., ZigBee[1], Devices Profile for Web Services (DPWS)[2]), interacting with a WoT-based device will be as simple as interacting with a "regular" Web page. This will lower the entry barrier into the IoT world and allow more developers to build new applications. Moreover, from the maintenance point-of-view this approach is much more effective. Any new software release containing a bug fix or new feature will automatically work with other devices.

Despite the benefits of a unified and centralized control, the real value of WoT will be the ability to create connections between the devices (virtual wires), allowing properties on different devices to be dependent on each other. For example, in order to prevent an explosion, a gas leak detector could use virtual connections to send a signal to turn off all home appliances. Another example of digital wiring is a smartphone that turns on the home heating system 5 minutes before the owner arrives. The key idea is that the software at devices should not need to know anything specific about other IoT devices in order to be able to interact with them without human intervention.

Formally speaking, the creation of virtual wires is achieved via event processing tasks, which constantly observe state changes at devices, analyze incoming data and, if a certain condition is satisfied, trigger an actuation. However, the creation and the placement of such tasks raises several issues/questions that are the focus of this work.

1.2 Motivation and Research Goals

The main motivation behind this work is to investigate and enable event processing in the new generation of IoT devices. To this end, this work aims to explore several research topics and to answer the following questions:

Standards – *What standards are required for distributed event processing to happen across heterogeneous devices?*
This work aims to perform an in-depth research of existing design patterns, standards and Web concepts that can be used when creating event processing tasks. Re-using well known and proven design patterns is important as it avoids the creation of custom made and proprietary event processing solutions, which pose an interoperability barrier. The end-goal is to investigate how current proposals and standards can be used in the creation of distributed event processing.

[1] `https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf`
[2] `http://docs.oasis-open.org/ws-dd/dpws/wsdd-dpws-1.1-spec.html`

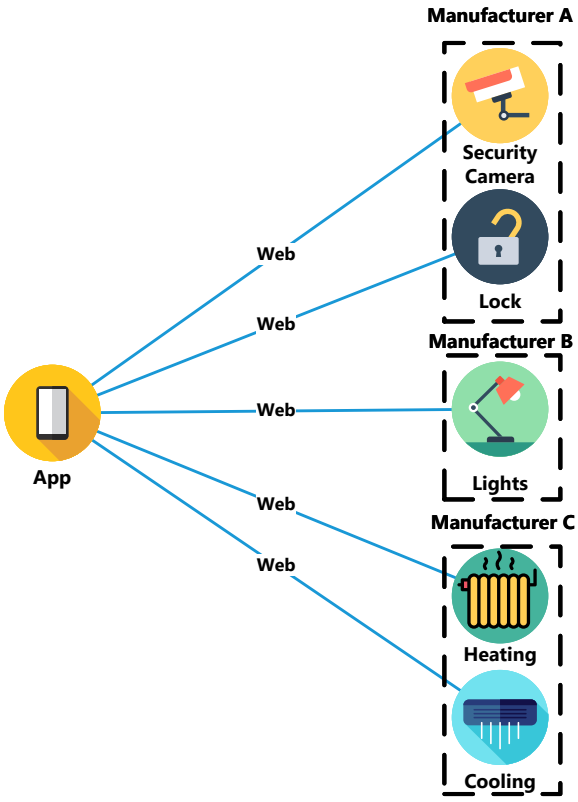


Figure 1.2: Envisioned WoT (adapted from [8]).

Edge Computing – *What is the role of edge computing in event processing? How edge devices can participate in event processing?*
Cloud computing is unable to satisfy the requirements of new IoT applications, including event processing. Some of these requirements are delay sensitivity, uplink costs, privacy, security, and non-interruption of services in environment with intermittent connectivity. Edge computing, which brings computation and storage near the end-devices, will play an important role in event processing. This work aims to investigate what mechanisms and protocols can edge devices use to expose their computational capabilities, announce their presence and their desire to participate in event processing.

Optimization – *What is the optimal placement of tasks for event processing to happen?*
Given the heterogeneous nature of edge devices, and different requirements of IoT applications, the placement of event processing tasks must be done according to the traffic patterns while satisfying user's demands. This work aims to formally investigate the placement strategies that are required for distributed event processing to happen efficiently, while taking into account IoT-related restrictions such as latency, computational resources, and device operation costs.

Machine Learning – *How can machine learning methods help in the distribution of event processing tasks?*
The rapid growth and evolution of communication networks will likely generate requirements for which there is no good performing distribution strategies/heuristics. Designing handcrafted solutions is time consuming and requires expertise in the area to achieve good performance. On the other hand, data-driven methods that, without much engineering, offer a possibility of discovering new distribution strategies seem to be a compelling choice to deal with such scenarios [1]. Recent advances in deep learning (DL) show remarkable performance in tasks like image recognition [2] or natural language processing (NLP) [3]. Moreover, deep learning in combination with reinforcement learning, commonly known as deep reinforcement learning (DRL), is showing some promising results in decision making problems like board games [4, 5], driving [6], robotics, and many more areas [7]. This work aims to investigate the use of DRL in the distribution process of the event processing tasks, which will enable the enhancement of WoT applications.

1.3 Thesis Overview

This work is divided in four chapters that provide answers to the previously outlined questions. Please note that Chapter 2 and Chapter 3 are based on two independent journal articles and, as such, some small sections and concepts may overlap.

The remainder of this thesis is organized as follows:

Chapter 2 focuses on standards, optimization and edge computing. This chapter explores currently existing approaches for event processing. It discusses their advantages, limitations and unique characteristics. Furthermore, this chapter investigates the "discovery" techniques/strategies that edge devices can use in order to announce their desire to participate in event processing. The overall trend in event processing is also outlined in here. Following the protocol and standard analysis, a new distributed event processing mechanism is presented, discussed and its optimal placement is mathematically formulated and studied. The content of this chapter was published in [30].

In Chapter 3 the attention falls on the edge computing, optimization and machine learning. This chapter investigates how event processing is currently happening at the cloud and what is necessary for it to smoothly transition to the edge. To that end, this chapter looks at the architectural and technological similarities between the cloud and the edge. Additionally, it looks into how edge devices will organize themselves during the event processing and what are the requirements for load balancing at the edge. After the analysis, three common load balancing problems are studied, involving the quality of the service provided, the fair usage of device resources and, finally, the minimization of operational costs. An attention-based neural network, trained with reinforcement learning methods, is used to provide a load balancing strategy for the three problems under consi-

deration. This chapter has been submitted to IEEE IoT special issue on “Empowering the Future Generation Systems: Opportunities by the Convergence of Cloud, Edge, AI, and IoT”.

In Chapter 4, the achievements, concluding remarks, and ideas for future research are outlined.

CHAPTER 2

A Distributed CoRE-based Resource Synchronization Mechanism

***Abstract*:** Representational state transfer (REST) application programming interfaces and event processing are the cornerstone of dynamic Internet of Things. While the first one is required for device interoperability, the latter is important for autonomous and responsive systems. In recent years, both topics have received a lot of attention and have been drastically changing due to the emergence of new applications, which end up working inefficiently with current standards and architectures. More recently, event processing started to move down from the top (cloud) to bottom (edge devices). At the same time, the Internet Engineering Task Force, which normally solves low-layer protocol related problems, has also started looking at event processing and resource synchronization from a bottom-up perspective. This chapter explores the intersection of these efforts by making an in-depth overview of currently existing standards, and Internet drafts, that allow building complex event processing chains. Next, a new reusable and scalable event processing mechanism, which can be distributed across multiple end-devices is introduced. Its optimal distribution across end-devices is mathematically addressed, and results confirm its effectiveness.

2.1 Introduction

With continuous advances in microelectronics, device miniaturization and reduction of production cost, an increasing number of devices is being built with wireless communication modules capable of emitting all kind of data to the Internet. In fact, according to IoT Analytics, by the year 2025 there will be around 34 billion devices connected to the Internet [31]. In order to seamlessly integrate the incoming wave of new devices with the already existing Internet infrastructure, the academics and the industry have made an enormous effort to bring the existing protocols, design patterns and architectures, used on the "regular" Web, to Class 1 and Class 2 constrained devices[1]. In other words, there is

[1] See "Terminology for Constrained-Node Networks" [32]

a move towards the creation of the Web of Things (WoT) [16, 18, 33, 34]. One of such efforts is being done by the Internet Engineering Task Force (IETF) IPv6 over Low power WPAN (6LoWPAN) [35] and the IETF Constrained RESTful Environments (CoRE) [36] Working Groups (WG).

The 6LoWPAN WG focused on standards for IPv6 to be used in 802.15.4 networks, specifying 802.15.4 payload and header compression [37], [38]. Additionally, in [39], IETF defined the IPv6 routing protocol for low-power and lossy networks (RPL). These and other low level protocols paved the way to the creation of new application protocols for constrained devices to adhere to the RESTful architectural style[2]. The development of such application protocols is being done by the IETF CoRE WG whose most notable outcomes are constained application protocol (CoAP) [41] and its extension, CoAP Observe [42].

CoAP opened the possibility to expose device functionalities using the well-known (to the developers and users) RESTful principles. CoAP supports standard GET, POST, PUT and DELETE methods alongside with response codes that are closely related to HTTP specification. Additionally, CoAP resources are addressable by uniform resource identifier (URI), it supports multiple Internet media types [43] and offers the possibility for RESTfull caching and proxying [44]. Overall, CoAP fits the requirements of a scalable protocol.

While CoAP brought a uniform REST-like request/response interaction pattern into low-power and lossy networks (LLNs), the CoAP Observe extended it by introducing the observer design pattern, thus creating the Create, Retrieve, Update, Delete and Notify (CRUDN) [18] interaction model. With this extension the client, interested in state changes of a resource, can subscribe and be notified by the server each time the observed resource changes its internal state. Such observation pattern plays an important role in M2M LLN environments as it reduces the usage of network resources by removing the periodic GET requests (polling) sent by clients interested in observing a certain event. CoAP's CRUDN methods can be implemented in Class 1 and 2 constrained devices for their integration into dynamic and reactive IoT environments. Such interaction model serves general interoperability goals, allowing flexible and dynamic solutions for event processing to be implemented. Event processing is considered a fundamental tool to derive real-time (or near real-time) conclusions from data in IoT applications.

In WoT, where every device resource is URI addressable, the event processing can be seen as a three stage process: `observation` of one or many endpoints, `evaluation` of the data produced and `actuation` by sending notifications to one or more endpoints. Together, these elements form the resource synchronization mechanism, where resources are located at URI addressable endpoints. Over the last decade, the `evaluation` part of this mechanism has changed significantly. It has been moving from the cloud down to the

[2]See "Architectural Styles and the Design of Network-based Software Architectures" [40].

fog and edge (mobile edge mostly). This transition is being motivated by the continuous improvement of computing capabilities of smaller devices, and due to the emergence of applications whose restrictions (e.g., delay sensitivity) cannot be satisfied when using the cloud. Substantial research has been done to make this transition as smooth as possible. Many different algorithms, protocols and heuristics, which bring the computation closer to the data producers, are being proposed, explored and applied with success [45–47].

The general trend is to move towards decentralized and distributed IoT applications, i.e., distributed `observe-evaluate-actuate` chains. Despite the promising results, current research focuses only on the idea that data from IoT applications can be split for independent `evaluation`, across multiple devices, and it ignores the possibility of discovering evaluation blocks (by clients/applications) for re-utilization. The nonreuse of evaluation blocks becomes critical in constrained environments. Also, most of the scientific research on distributed processing focus on the optimization part and abstracts from standardized communication protocols and data models. Such kind of approaches tend to create custom made solutions and, therefore, present an interoperability barrier. In our opinion, IoT environment will greatly benefit from a uniform, reusable and distributed synchronization mechanism that does not rely on external applications/controllers and is built upon open standards such as CoRE.

The main contributions of this chapter are:

- A study is made on how CoRE-related standards and ongoing research work documented in Internet-draft proposals can be used to build a robust and reliable IoT synchronization mechanism. We will take a deep look at their internals, capabilities and limitations. A close attention will be given to the most recent synchronization mechanism called `Rule` that is currently in early stage of development and is now under evaluation to be a part of open connectivity foundation (OCF) core specification[3,4,5].

- A decomposition mechanism for `Rule` is proposed allowing it to be split into a set of small, chainable and reusable components. A distributed implementation of the components allows scalable and reliable solutions to be built in constrained environments.

- Finally, the impact of `Rule` decomposition and chaining is analyzed by addressing the rule placement (RP) problem.

The remaining of this chapter is organized as follows: in Section 2.2 the existing fog and edge-based proposals for distributed resource synchronization are presented together

[3]See "Rules Definition" [11]

[4]Original specification: `https://github.com/openconnectivityfoundation/core/pull/129`

[5]Updated specification: `https://github.com/openconnectivityfoundation/core-extensions/pull/29`

with their limitations; in Section 2.3 a deep look at the CoRE-related synchronization proposals, with a special attention to `Rule` mechanism, is carried out; Section 2.4 discusses how rules can be chained, reused and distributed across different devices; Section 2.5 introduces the RP problem and presents its mathematical formalization; Section 2.6 studies the impact of RP while ensuring a fair use of device resources; Finally, in Section 2.7 concluding remarks are outlined.

2.2 Related Work

So far, and to the best of our knowledge, there were no attempts to explore distributed event processing in CoRE-based IoT synchronization mechanisms. Nevertheless, and although using different terminology and technologies, many attempts have been made to build distributed event processing at cloud, fog, edge and sensor networks. This section takes a look at such related work.

In [48] the authors outline the importance of distributed computation as a critical enabler for a fast and responsive IoT system. The main idea is to share and use the computational capabilities of the nearby devices and, whenever necessary, use the cloud capabilities to assist the computation. This workload sharing mechanism assumes that each IoT application can be split into independent execution units and, therefore, distributed across different devices. This is a common approach and similar studies have been done for the fog in [49] and [50]. In these, authors study a delay-sensitive offloading schema in fog computing. However, these approaches do not consider that multiple applications can share similar execution units and, therefore, reuse is not addressed. Also, the possibility of creating a chained processing is not explored.

In [51] the authors introduce the concept of virtual sensor, which is an abstraction of real sensors extended with computational capabilities. Different types of virtual sensors are defined, such as aggregator, qualifier, selector, etc. The authors study the performance of virtual sensors in a cluster of web servers. This approach is similar to flow programming applications such as Node-RED [52] or Eclipse Kura [53]. Although allowing virtual sensor chaining, at the moment of writing these services do not support WoT standards [18], [54], [55]. These and other applications, such as if-this-then-that (IFTTT) [56] [57], rely mostly on centralized solutions (e.g., cloud and edge), and fair distribution of virtual sensors across devices is yet to be tackled.

In [58] the authors explore the use of virtual sensors in body sensor networks and how they can be chained together in signal processing tasks. The virtual sensor presented is a computational component that has a set of inputs, a processing task and an output, which can be further used as input by other, higher level, virtual sensors. The components create distributed, fault tolerant graph, enabling efficient utilization of resources and application interoperability. Authors show the feasibility of their approach by creating a virtual

sensor graph capable of recognizing events from a walking person. However, the presented approach was only implemented in SPINE software, not used for the assessment of protocols.

Countless solutions proposed by academics [59–61] and commercial projects such as Amazon IoT [62], Azure IoT Hub [63] and cloudMQTT [64] use message queuing telemetry transport (MQTT) [65] protocol, which was first introduced in 1991, as a basis for IoT applications. Despite their differences, they are all based on publish/subscribe paradigm. In this paradigm, constrained devices (clients) connect to a server, also known as broker, and use it to publish or to subscribe/receive messages. This way of communication decouples completely the client that sends a message (the publisher) from the client(s) that receive it (the subscribers). For this reason, event processing is simple to achieve: sensors will publish data that an application (e.g., rule engine) subscribes; then if the data satisfies the triggering condition, the application publishes another message that an actuator is listening to. However, the flexibility that MQTT provides has also some drawbacks. For example, the upper bound of network traffic is limited by the capacity that a local broker or brokers can handle. Additionally, the broker is a single point of failure meaning that clients would keep working and wasting energy despite broker's unavailability, which is critical in battery powered devices. MQTT has also no content negotiation mechanism meaning that all the devices must know *a priori* the message formats to be able to communicate [66], [67].

2.3 CoRE-based Resource Synchronisation Mechanisms

IETF started looking at CoRE-based synchronization mechanisms in a bottom-up manner, i.e., from *Access*, to *Find*, *Share* and *Compose* layers[6], which can be seen as sublayers of the application layer (Internet Protocol stack). IETF's effort was motivated by the need of providing more autonomy to constrained devices in M2M synchronization tasks and to ensure seamless and uninterrupted service in environments with intermittent connectivity.

2.3.1 Conditional Observe in CoAP

One of the first IETF attempts to achieve this was presented in 2012 [68]. The authors propose to extend CoAP with an additional `Condition` header option that is used to specify up to 32 different conditions for notification triggering. This mechanism offers means for a direct conditional notification between clients and servers. In their work the authors show that conditional observation reduces the energy consumption by eliminating the transmission of unnecessary packets. While being energy advantageous, such direct resource observation is not enough by itself as it can only convey one condition

[6] See Web of Things Architecture in "Building Web of Things" [8]

per CoAP request. For example, if a client is interested in receiving a notification whenever temperature goes below 17℃ or beyond 23℃ then it would be necessary to send two separate Observe requests. Additionally, embedding these conditions directly into the CoAP header means that the condition types must be predefined (32 types of conditions are proposed in [68]). Due to such limitations this proposal became stale and was replaced by CoRE dynamic linking.

2.3.2 CoRE Dynamic Linking

Current IETF CoRE WG effort to introduce a synchronization mechanism, that can be directly executed on constrained devices, is called CoRE dynamic linking [9] and it first appeared in 2016. The state updates between the endpoints are exchanged using the link binding concept. A link binding is a unidirectional (virtual) connection between two resources whereby state updates at the source resource are sent to the destination resource. A binding allows to exchange the updates between the resources in a direct and distributed manner, i.e., without relying on any external application or data aggregation service. For the management of binding instances, the CoRE WG defined the binding table where bindings can be created, updated and removed. Current draft specification defines three different binding methods:

- `Polling` - Located at the destination device performing periodic GET requests in order to obtain the source resource representation.
- `Observe` - Located at the destination device that makes a GET Observe request in order to obtain the source resource representations.
- `Push` - Located at the source device performing PUT requests in order to notify the destination resource about state changes at the source.

That is, binding instances will be stored either on a source or destination device depending on the binding method used. In addition, current CoRE dynamic linking specification defines the following binding attributes:

- `pmin` - Minimum time between notifications.
- `pmax` - Maximum time between notifications.
- `st` - Threshold indicating how much the value of a resource should change, compared to the value from the last notification, before triggering a new notification.
- `gt` - Upper limit value that the resource should cross for notification triggering.
- `lt` - Lower limit value that the resource should cross for notification triggering.

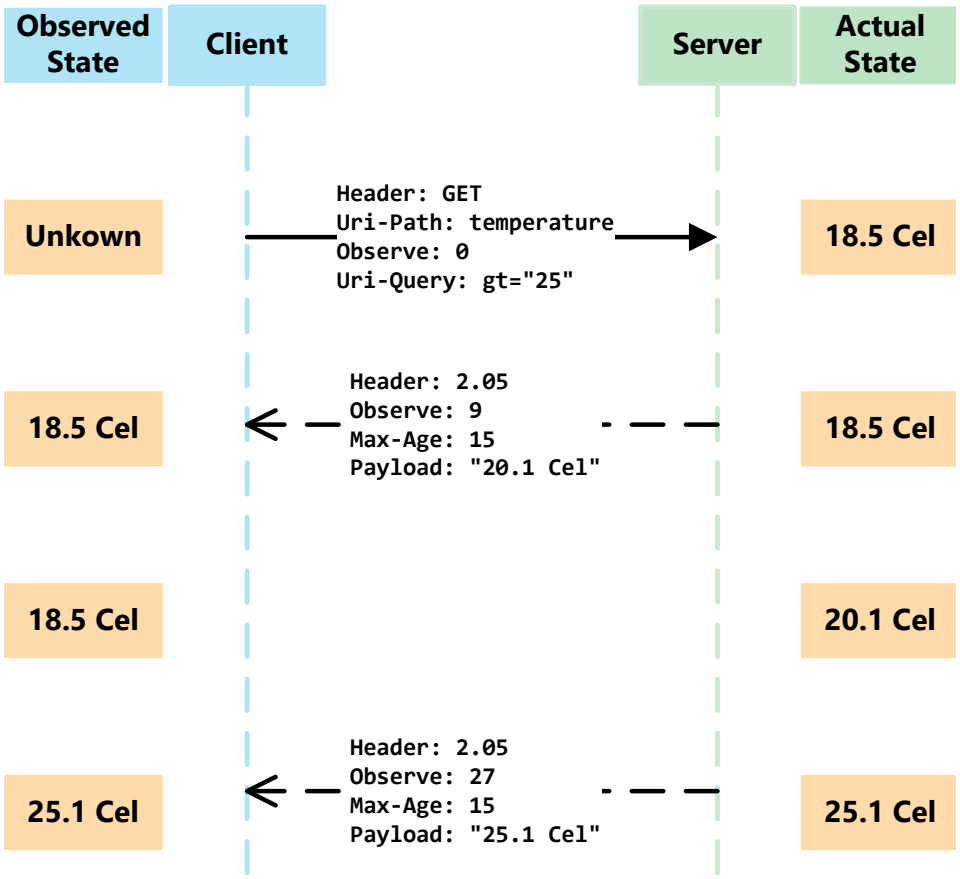


Figure 2.1: CoAP dynamic linking conditional observation (Adapted from [9]).

- `band` - A boolean value that acts as a behavior modifier of `gt` or `lt`. Without the `band` parameter, the `lt` and `gt` convey only one notification, when these values are exceeded. On the other hand, with the `band` in place the notifications are sent as long as the value of the observed resource exceeds the `lt` and `gt` limits.

Figure 2.1 shows an example of a binding process. In this case, the client performs a conditional observation of temperature resource. The condition (`gt=25`) is passed to the server as a URI query in the observation request.

While providing means for a direct resource synchronization, the dynamic linking specification does not define the ability to customize notifications. As an example, let us consider the descriptions for the presence sensor (PIR) [69] and a lock [70] (defined by OCF) where the first produces a boolean value as output and the latter accepts "`Locked`" or "`Unlocked`" string as input. Creating a simple sensor-actuator binding *if* presence detected *then* send "`Unlocked`" *else* send "`Locked`" would be impossible by relying on dynamic linking mechanisms only. Either client side processing and filtering or an ad-

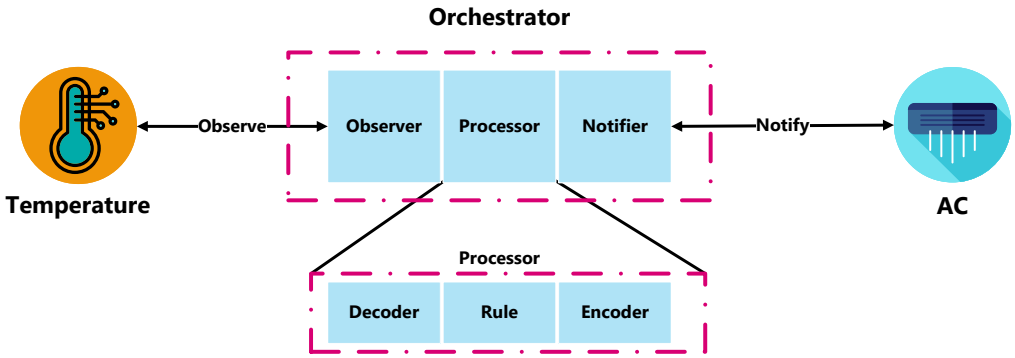


Figure 2.2: Monitor components and synchronization flow (Adapted from [10]).

ditional middleware service performing the translation of PIR output into lock acceptable input would be necessary in order to enable the desired interaction.

2.3.3 Monitor

A solution to dynamic linking limitation was proposed, in 2016, to IETF Thing-2-Thing Research Group by one of the co-authors of CoRE dynamic linking. The author introduced the Monitor resource [71], an extension of dynamic linking that encompasses the possibility to customize the notification that is sent. The Monitor instance, represented in Figure 2.2, is composed of three components:

1. `Observer` - Set of parameters indicating how the state representation of the source resource should be obtained.
2. `Processor` - Set of parameters informing how to evaluate the observed state of a resource. This component can be further decomposed in three different stages:
   (a) `Decoder` - Uses `accept-schema` (discussed below) to interpret the observed resource, i.e, to extract the value from the notification.
   (b) `Rule` - Conditional expression evaluating the data extracted from the previous step. Currently, the conditional parameters are the ones defined in Dynamic Linking draft specification.
   (c) `Encoder` - Uses the `transfer-schema` (discussed below) to build the desired notification.
3. `Notifier` - Set of parameters indicating how the destination resource should be notified.

These components are specified through the use of the following parameters:

- `anchor` - URI of an observable resource.

- `accept` - Content format for observable resource response.
- `accept-method` - Transfer method from the observed resource to the Monitor.
- `accept-schema` - Schema used to interpret the observed resource payload.
- `href` - URI of the destination that will receive notifications.
- `content-type` - Content format that will be used to build notifications.
- `transfer-method` - Transfer method used to send notification.
- `transfer-schema` - Schema to use when building the notification payload.

As an extension of dynamic linking, the Monitor can be placed at the source or destination. Moreover, the explicit declarations of observation, evaluation and notification parameters allow Monitor to be outsourced to a third-party device, allowing for distributed, reliable and scalable solutions to be implemented. In such outsourced scenario, the Monitor acts as a man-in-the-middle, just as a MQTT broker does in a publish/subscribe architecture. However, their behavior is different. The latter acts only as a hub, responsible for reception and efficient delivery of the data, while the Monitor accepts the incoming data, processes it and, if triggering parameters are satisfied, sends a notification to the destination. However, although a description for Monitor's functionality is provided, the author does not specify how to interpret the `accept-schema` and `transfer-schema` parameters. Also, the current Monitor specification does not specify the possibility of observing or notifying multiple resources, i.e., many-to-one or one-to-many notifications. Thus, at the moment of writing, the Monitor is a concept, in draft stage, without internal behavior specification.

2.3.4 Rule

The most recent proposal to overcome the limitations of CoRE dynamic linking and Monitor mechanisms appeared in 2018 and is under research in [11]. The proposed solution is a many-to-many synchronization mechanism built upon well-known CoRE standards and drafts such as CoAP [41], CoRE interfaces [72], CoRE dynamic linking [9] and CoRE link format [73]. Although still a draft, it is under evaluation for inclusion into the core framework of OCF core specification. The proposal introduces a `Rule` resource, represented as a collection, for autonomous decision logic to be performed following a simple `observe-evaluate-actuate` pattern. The `Rule` collection has the following main components:

- A set of `Rule Inputs`. A `Rule Input` is a local copy of an external source resource. This resource stores the last known state (snapshot) of a source resource.

```
{
 "rule": "(humid:humid > set-humid:humid) and (temp:temp > set-temp:temp)",
 "ruleresult": false,
 "ruleenable": true,
 "actionenable": true
}
```

Listing 1: `Rule Expression` syntax and `Rule`'s control variables.

- A `Rule Expression` that evaluates the `Rule Input` values and produces a boolean as output. The `Rule Expression` is a conditional logic string that supports simple operators, such as $<$, $>$, $=$, `and`, `or`, `contains`, `exists`, etc.
- A set of `Rule Actions`. A `Rule Action` is a local and pre-configured notification template that is used to notify destination resources. This effectively solves the client side notification filtering/processing problem, occurring in Dynamic Linking, as messages are sent in a pre-configured and client acceptable form.

Every element of the `Rule` collection supports CRUD operations meaning that `Rule`'s behavior, inputs and outputs can be modified on-the-fly with a single client request.

The overall execution logic of the `Rule` resource is as follows: every time a `Rule Input` is updated, either by an external call or by an internal push `Dynamic Link` (see Figure 2.3) instance, the rule evaluation expression is executed. If the `Rule Expression` changes its state from `False` to `True` then all `Rule Action` instances are executed and, as a result, notifications are send to the destination resources. Figure 2.3 is a graphical representation of the internal building blocks of a `Rule` resource. The actual syntax of a rule and its control variables are shown in Listing 1. The `ruleresult` contains the last evaluation result produced by a rule expression. When this variable changes its state from `False` to `True` the processing of `Rule Actions` is triggered. The `actionenable` allows to control if `Rule Actions` are to be executed. That is, allows rules to be tested prior to execution of `Rule Action` instances. The `ruleenable` allows rule enabling/disabling. Finally, the `rule` variable contains the actual expression that is to be evaluated. In this case, checks if temperature and humidity values do not exceed the user-defined (`set-humid` and `set-temp`) ones. Note that the syntax of the evaluation expression (see [11]) is almost identical to the one used in content directory service defined by UPnP [74]. It also shares some similarities with the NGSI-LD Query Language used in NGSI-LD's context information management (CIM) and application programming interface (API) [75]. However, `Rule`'s syntax is more complete as it defines operators (e.g. `contains`, `doesNotContain`, `startsWith`, etc.) that the NGSI-LD Query Language does not have.

Taking everything into account the `Rule` resource is a synchronization mechanism that effectively solves the endpoint-to-endpoint conditional notification between multiple

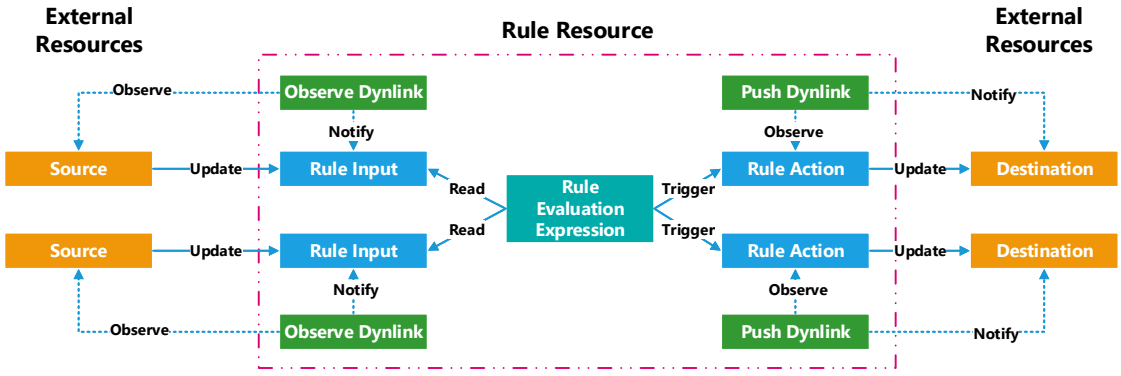


Figure 2.3: `Rule` resource internal components and their relations (Adapted from [11]).

devices. Also, it is a scalable and simple one-to-one synchronization that can be executed in devices with limited resources, while allowing complex many-to-many synchronization scenarios involving multiple devices. Moreover, since it is built upon REST principles and CoRE standards it can benefit from already existing discovery mechanisms (e.g., registration at CoRE resource directory [76] or at Thing directory [77] or use CoAP broadcast to inform clients) that facilitate `Rule` discovery and consumption. Overall, `Rule` resource can be seen as a robust and flexible mechanism capable of creating rich M2M interactions that can be easily integrated into already existing infrastructures and WoT applications following similar principles.

Although using different nomenclature and structure, Google's experimental simple monitoring and control protocol (SMCP) in combination with splot object model (SOM) [78], also introduced in 2018, defines very similar resource synchronization mechanisms to those defined in Rule.

2.4 Rule Decomposition, Chaining and Distribution

2.4.1 Decomposition and Chaining

The `Rule`'s ability to customize notifications, through `Rule Actions`, coupled with an expressive conditional logic syntax can model rich and complex interactions between multiple devices. Due to loose coupling of components included in rule collections, decomposing mechanisms can be developed in order to replace a rule requiring high processing by a set of smaller and lighter ones. This has the advantage of allowing distributed processing among nodes, without losing the original logic, while increasing the potential reuse of rule collections. Since rules are regular URI addressable resources, these can also be chained to achieve multiple levels of abstraction and, therefore, model complex situations and interactions. To clarify rule decomposition and chaining, let us consider a simple illustrative example. Consider a scenario with three resources on a network: `Temperature`, `Humidity` and an `Air Conditioner` (`AC`). The first two produce

integer values, between 0 and 100, as output while the latter accepts either "On" or "Off" as input string. Assume also a user defined demand with the following information:

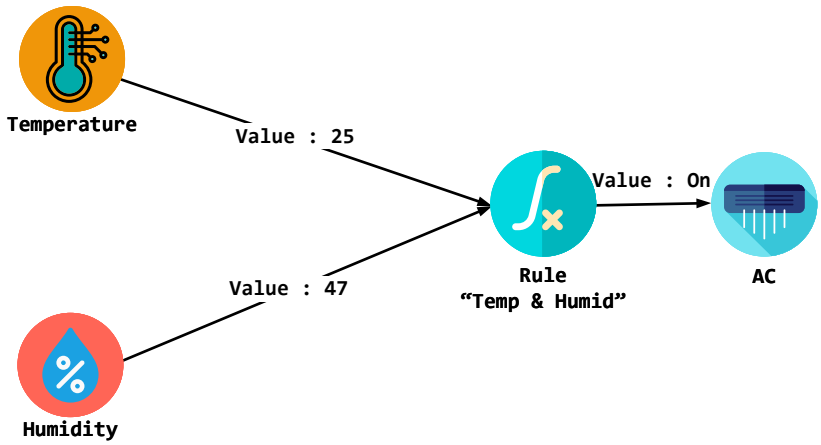
- Input Resources:
  - Temperature
  - Humidity
- Task/Evaluation Expression:
  - temperature $>$ 27 and humidity $>$ 47
- Output Resource:
  - Air Conditioner
- Notification Template:
  - "On" string

This means that the user wants to observe the state of Temperature and Humidity resources and if these are higher than 27 and 47, respectively, then the AC must be notified with an "On" message, which is a notification template. This resource synchronization can be achieved in two ways. The first one, presented in Figure 2.4a, is by directly mapping the demand into a single Rule resource. An alternative, since the evaluation expression is a set of conditions connected by "*and*" or "*or*" operators, is to split the evaluation expression into a set of smaller, generic and reusable evaluation expressions. These smaller expressions can be mapped into two separate Rule resources, as presented in Figure 2.4b. In this case, an additional Collecting Rule, with source resources Rule "Temp" and Rule "Humid", would be required in order build the logic of the original expression. These would notify the Collecting Rule with a boolean value, indicating whether their condition has been satisfied or not. The evaluation expression of Collecting Rule would look like: Rule "Temp" = True and Rule "Humid" = True. Finally, its notification template would be "On" string that would be sent to the AC each time its evaluating expression transits from False to True state.

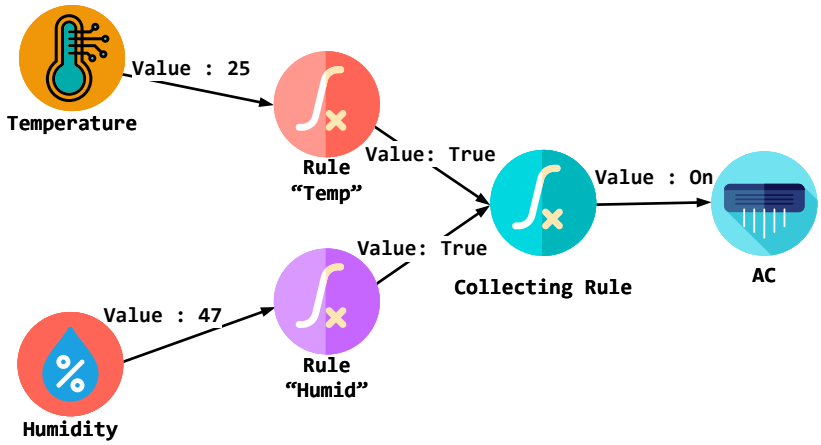
2.4.2 Distributed Placement

In addition to decomposition and chaining, rules can be distributed across multiple devices. The device's ability to participate in rule processing is done by exposing a Rule Collection Resource with a create CoRE interface[7,8] (if="oic.if.create")

[7] See "Create, Delete, and Link Update Interface Definitions" [79]
[8] See "Reusable Interface Definitions for Constrained RESTful Environments" [72]



(a) Demand mapped to a single rule.



(b) Demand mapped to two rules plus a collecting rule.

Figure 2.4: A user demand expressed through a single or multiple rules.

and a resource type[9] array (`rts=["oic.r.rule"]`) entry indicating that it supports creation of `Rule` instances [11]. Note that, the device may also use active methods (e.g., registration at resource directories or broadcast messages) to inform clients about its presence and ability to process rules. Although these rules can be placed at any CoAP-based device, some devices may choose not to accept and process a rule instance due to some limitation. In that case, they should simply return an appropriate status code (e.g., `4.13 Request Entity Too Large` or `4.22 Unprocessable Entity`)[10] informing the client that it refuses to process the request.

To illustrate how a `Rule` can be decomposed, let us assume a local network with three devices (`Thing A`, `B` and `C`) available for rule processing. Assume also that the rule illustrated in Figure 2.4a is being executed at `Thing A`. Moreover, assuming that for network optimization purposes (e.g. network lifetime increase) a move from scenario in Figure 2.4a to scenario in Figure 2.4b should take place. Finally, assume that a `Rule Configuration Tool` is being executed at smartphone, or any other capable device with enough computational capabilities, and that it has no prior knowledge about the resources nor rules available on network. Figure 2.5 illustrates the steps required to achieve the desired changes.

Resource Discovery and Computation Phase

Initially, the `Rule Configuration Tool` must discover the resources available on the network. The discovery can be done through a request to an HSML [71] server located at local network, which acts as a central point where devices register their resources to ease their discovery. Due to HSML's capability to store both data (HSML items) and meta-data (HSML links) about web resources the `Rule Configuration Tool` can obtain, with a single request, information regarding the existing web resources, including the rule presented in Figure 2.4a, and computational resources (e.g. CPU, RAM, etc.) of `Thing A`, `B` and `C`. This information is used to compute an optimal rule configuration in terms of resource usage. In this case, the optimal solution is to decompose the rule in Figure 2.4a into a multi-rule scenario as presented in Figure 2.4b.

Rule Configuration Phase

Once the discovery and computation is completed, the `Rule Configuration Tool` makes two requests to `Thing B` and `C` in order to create two rules - one for evaluation of `Temperature` and another for `Humidity`. Both requests provide a confirmation response containing the URL addresses of newly created rules. These URL addresses are

[9] See "Constrained RESTful Environments (CoRE) Link Format" [73]

[10] See Error Handling in "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)" [80]

used as inputs to an aggregation rule called `Collecting Rule`, which is created when the `Rule Configuration Tool` makes a request to `Thing A`. When `Thing A` receives the request to create the `Collecting Rule` it automatically sends two Observation requests to the `Temp` and `Humid` rules, located at their respective URI addresses, in order to be notified about state changes. Finally, after chain creation, the original rule is safely deleted. The discovery of newly created rules is facilitated by their automatic registration at HSML server.

Rule Execution Phase

During the rule execution phase, temperature notifications are sent to the "`Temp`" rule for evaluation and, when the condition is satisfied, a notification is passed up in the chain to the `Collecting Rule` that is responsible for interacting with the `AC`.

The presented approach leaves the device operation unmodified, and, in theory, it can be executed at any CoAP-capable device. The only constraint is the computational complexity associated with the rule's conditions. Regarding the computation of the optimal rule placement, it can always be outsourced to more powerful devices (e.g., smartphone).

The scenario presented in Figure 2.4b can be seen as a mesh of small processing units that can be organized and, when needed, rearranged in an efficient way. This distributed resource synchronization mechanism does not have a single point of failure and thus it improves the overall reliability and availability. However, distributing rules over multiple devices may have multiple challenges related with security, privacy and delay. Distribution of rules on a local (trusted) network, instead of relying on external services (e.g., cloud or edge computing) can increase user's security and privacy as compartmentalized and locally distributed data is harder to eavesdrop by malicious users. On the other hand, distribution in public networks may pose a serious threat. However, since the presented approach is based on Web protocols, security and privacy mechanisms currently applied to the Web can always be used. For example, secure CoAP (CoAP + DTLS) was standardized in [41]. Additionally, IETF CoRE-WG is currently working on other encryption, authentication and end-to-end protection mechanisms such as COSE [81] and OSCORE [82] for CoAP-based devices.

2.5 Rule Placement Problem

2.5.1 Terminology and Notation

Generally speaking, any device built upon REST and CoRE standards could, in theory, perform rule processing and, therefore, play a role in resource synchronization. However, due to the large plethora of heterogeneous devices, each with different computational

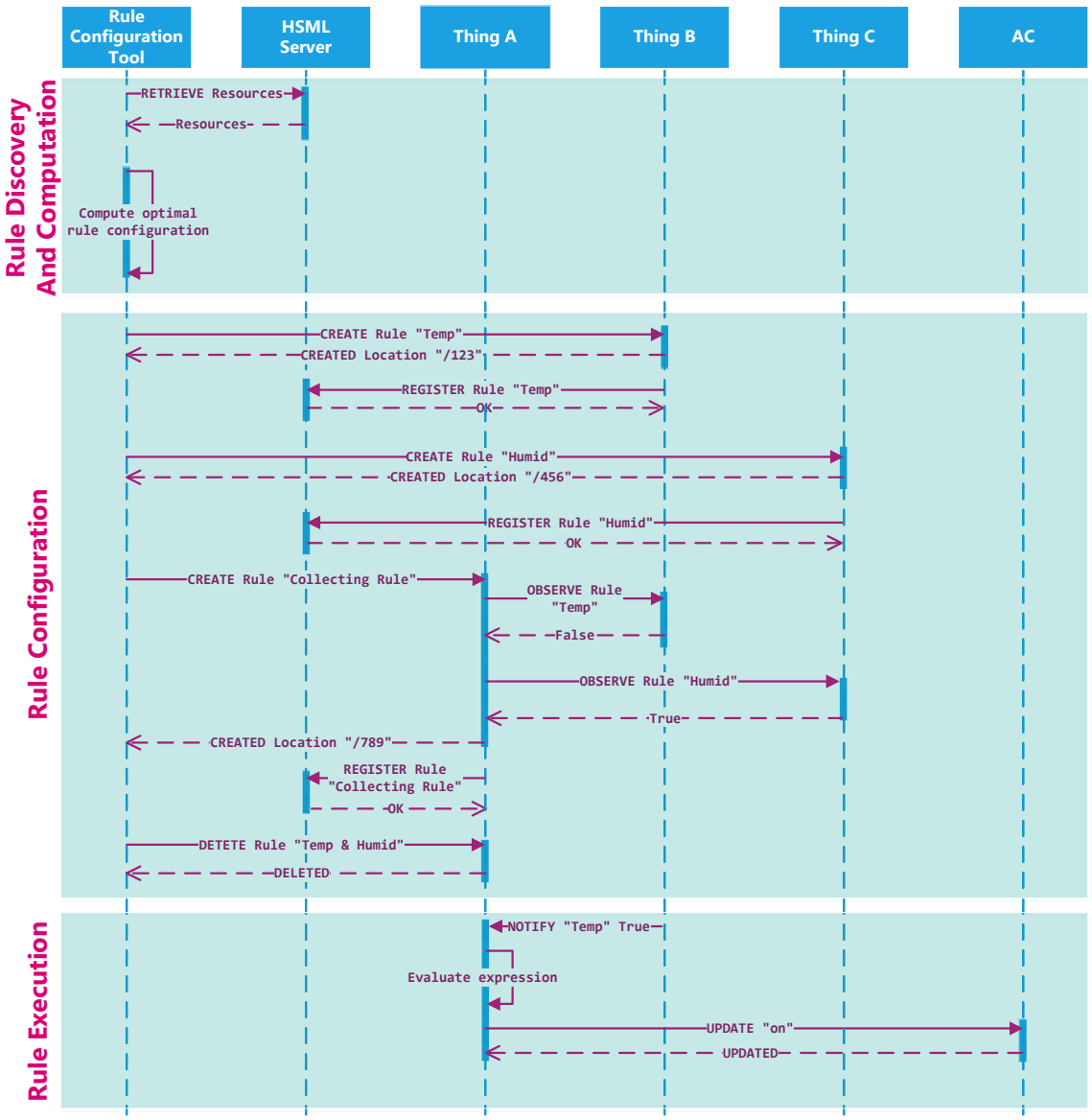


Figure 2.5: Requests necessary to decompose and chain the rules as presented in Figure 2.4b.

capabilities and resources, a careful placement of rule instances is required in order to ensure fair device loads and to provide a fast sensing-actuation response. Overall, this problem can be seen as a Rule Placement (RP) problem. For clarity, terminology and notation are summarized prior to problem formulation.

Node Constrained device inside the network. The overall set of heterogeneous nodes is denoted by $\mathcal{N}$.

Computational Resource Hardware resource (e.g., CPU, RAM, storage) of a node $n \in \mathcal{N}$. A node n has an amount/capacity for M computational resources, given by vector $\mathbf{c}^n = [c_{n,m}]$, $m \in \{1, ..., M\}$.

Web Resource URI addressable endpoint storing the internal state of one or more properties (e.g., sensed temperature, battery level, etc). The set of resources available at node $n \in \mathcal{N}$ is denoted by $\mathcal{R}^n$.

State Set Universe of states that a resource $r \in \mathcal{R}^n$, $n \in \mathcal{N}$, can accept/return. The state set for r is denoted by $\mathcal{S}_r^n$, and a state $s \in \mathcal{S}_r^n$ is a tuple including as many elements as the number of properties referenced by resource r.

Task An evaluation expression (task), defining some logic on the inputs and returning a boolean value. Besides simple logic, complex processing on the inputs (eventually integrating other info; e.g., historical info) can be performed. The overall set of possible tasks is denoted by $\mathcal{T}$.

Notification A tuple including an evaluation expression (task) $t \in \mathcal{T}$ and an element s from $\bigcup_{\{n \in \mathcal{N}, r \in \mathcal{R}^n\}} \mathcal{S}_r^n$, denoted by $< t, s >$. That is, s is used for the update of a resource and is a result from task t.

Rule Defines a set of inputs, which are resources, along with a set of notifications. More specifically, assuming $\mathcal{X} = \{x_1, x_2, ..., x_{|\mathcal{X}|}\}$ to be a set of rules, each x_i:

- Requires a set of resources as input, denoted by $\mathcal{I}^{x_i}$. Each element j of $\mathcal{I}^{x_i}$ is a tuple denoted by $j =< n_j, r_j >$, $n_j \in \mathcal{N}$, $r_j \in \mathcal{R}^{n_j}$.
- Produces a set of one or more notifications, denoted by $\mathcal{O}^{x_i}$. Notifications are triggered by a task.
- Has a demand vector $\mathbf{d}^{x_i} = [d_{x_i,m}]$, $m \in \{1, ..., M\}$ containing computational resource requirements.

- Has special hardware/software requirements that restrict the set of nodes where it can be executed, denoted by $\mathcal{N}^{x_i} \subseteq \mathcal{N}$.

Whenever a resource $r \in \mathcal{R}^n$ is to be updated by a notification, one or more rules from $\mathcal{X}$ may be activated. That is, one possibility is to activate a single rule that performs the requested task, processes all required inputs, and returns the right state. Another possibility, for rule reuse and, therefore, network optimization, is to use more elementary rules, also defined in $\mathcal{X}$, that would perform part of the logic/processing. These can then feed their notifications into collecting rules that perform the additional logic. Therefore, collecting rules can be defined as having input $< t, s >$ tuples, this is, notifications provided by rules in $\mathcal{X}$.

Collecting Rule Defines a set of inputs, which are notifications. More specifically, assuming $\mathcal{C} = \{c_1, c_2, ..., c_{|\mathcal{C}|}\}$ to be a set of collecting rules, each c_i:

- Requires a set of inputs, denoted by $\mathcal{I}^{c_i}$. Each element j of $\mathcal{I}^{c_i}$ is a notification denoted by $j =< t, s >, t \in \mathcal{T}, s \in \bigcup_{\{n \in \mathcal{N}, r \in \mathcal{R}^n\}} \mathcal{S}_r^n$.
- Produces a set of one or more notifications, denoted by $\mathcal{O}^{c_i}$.
- No computational resource requirements are assumed, as these will implement basic logic over the inputs.
- Has no special hardware/software requirements that limit the set of nodes where it can be executed, meaning that $\mathcal{N}^{c_i} = \mathcal{N}$.

Dominant Resource Given a rule $x_i \in \mathcal{X}$ to be executed at $n \in \mathcal{N}^{x_i}$, the dominat resource (see [83]) of x_i at n will be the resource $m \in \{1, ..., M\}$ with the highest allocated percentage share. That is, $\Delta(x_i, n) = \arg\max_{m \in \{1,...,M\}} \{\frac{d_{x_i,m}}{c_{n,m}}\}$. This is the resource that most limits the allocation of the rules to nodes. The dominant resource of x_i can be different at each n, as computational resources of nodes may not be the same.

2.5.2 Problem Definition and Formalization

Since rule notifications may serve as input to collecting rules, or may flow directly towards nodes for resource update, a virtual graph will be defined for the required flow to be ensured in the following formalization. This is denoted by $\mathcal{G}(\mathcal{V}, \mathcal{L})$, where $\mathcal{V} = \mathcal{X} \cup \mathcal{C} \cup \mathcal{N}$. Regarding $\mathcal{L}$, there will be the following virtual links:

- $(x, c), \forall x \in \mathcal{X}, c \in \mathcal{C}$;
- $(x, n), \forall x \in \mathcal{X}, n \in \mathcal{N}$;

- $(c_i, c_j), \forall c_i, c_j \in \mathcal{C}$;

- $(c, n), \forall c \in \mathcal{C}, n \in \mathcal{N}$.

Let $\mathcal{D}^n, \forall n \in \mathcal{N}$, denote a set of demands for the update of resources at n. A demand $d \in \mathcal{D}^n$ is a tuple $< r, t >$, stating that resource $r \in \mathcal{R}^n$ is to be updated with the result of task $t \in \mathcal{T}$. Let us also assume the following variables:

ω_v	One if rule $v \in \mathcal{X} \cup \mathcal{C}$ is to be executed, requiring some place to be determined, zero otherwise.
γ_v^d	One if rule $v \in \mathcal{X} \cup \mathcal{C}$ is to used by demand $d \in \mathcal{D}^n, \forall n \in \mathcal{N}$, zero otherwise.
β_v^n	One if rule $v \in \mathcal{X} \cup \mathcal{C}$ is to be placed at node $n \in \mathcal{N}^v$, zero otherwise.
ε_l^d	Real value that when greater than 0 indicates that there is flow through link $l \in \mathcal{L}$ for the update of resource r in d's tuple $< r, t >$, $d \in \mathcal{D}^n$, zero otherwise.
$\vartheta_{x_i}^d$	One if the update of resource r in d's tuple $< r, t >$, $d \in \mathcal{D}^n$ has $x_i \in \mathcal{X}$ as a source of the rule chain, zero otherwise.
$\pi_{n_i,n_j}^{d,l}$	One if $\varepsilon_l^d > 0$ and l's endpoints are placed at nodes $n_i, n_j \in \mathcal{N}$, zero otherwise.
$\varphi_{x_i,n}^d$	One if the update of $d \in \mathcal{D}^n$, has $x_i \in \mathcal{X}$ as feeder/source placed at $n \in \mathcal{N}$, zero otherwise.
$\alpha_{x_i,\Delta(x_i,n)}$	Percentage of dominant resource that is allocated to rule $x_i \in \mathcal{X}$, if placed at node $n \in \mathcal{N}^{x_i}$.
Δ^{MIN}	Lowest percentage of dominant resource allocated to rules.

Having these definitions and notation in mind, the RP problem can be formulated as follows.

Objective Function

$$\text{Maximize } \Delta^{\text{MIN}} \tag{2.1}$$

This goal ensures fairness among rules, regarding assigned resources, because an overall lower bound on the percentage of allocation resource is being maximized. That is, rules will have similar working conditions.

Placement of Rules

$$\sum_{n \in \mathcal{N}^v} \beta_v^n = \omega_v, \forall v \in \mathcal{X} \cup \mathcal{C} \tag{2.2}$$

$$\omega_v \geq \gamma_v^d, n \in \mathcal{N}, \forall d \in \mathcal{D}^n, \forall v \in \mathcal{X} \cup \mathcal{C} \tag{2.3}$$

These constraints ensure that placement must be defined whenever a rule (collecting or not) is to be executed. Constraints (2.2) ensure single placement, if the rule is to be executed ($w_v = 1$), while Constraint (2.3) force the execution of any rule under utilization by at least one demand.

Filling Demand's Content

$$\sum_{l \in \mathcal{L}: dst(l) = v} \varepsilon_l^d - \sum_{l \in \mathcal{L}: src(l) = v} \varepsilon_l^d = \begin{cases} 1, & if\ v = n \\ 0, & if\ v \in \mathcal{C} \end{cases}, n \in \mathcal{N}, \forall d \in \mathcal{D}^n, \forall v \in \mathcal{C} \cup \{n\} \tag{2.4}$$

$$\sum_{\{l \in \mathcal{L}: dst(l) = c \wedge j \in \mathcal{O}^{src(l)}\}} \varepsilon_l^d > \gamma_c^d - 1, n \in \mathcal{N}, \forall d \in \mathcal{D}^n, \forall c \in \mathcal{C}, \forall j \in \mathcal{I}^c \tag{2.5}$$

Constraints (2.4) implement the flow conservation law from rules towards the resource to be updated using, if necessary, collecting rules for intermediate processing, while Constraints (2.5) ensure that all inputs are fulfilled.

$$\vartheta_{x_i}^d \geq \varepsilon_l^d, n \in \mathcal{N}, \forall d \in \mathcal{D}^n, \forall x_i \in \mathcal{X}, \forall l : src(l) = x_i \tag{2.6}$$

$$\gamma_v^d \geq \varepsilon_l^d, n \in \mathcal{N}, \forall d \in \mathcal{D}^n, \forall v \in \mathcal{X} \cup \mathcal{C}, \forall l \in \mathcal{L} : src(l) = v \tag{2.7}$$

Constraints (2.6) determine which rules in $\mathcal{X}$ will be at the beginning of the flow chain. These are required to compute latency. These constraints (2.7) to activate the required rules (collecting or not) according to the existing flow defined by Constraints (2.4).

Final Step of Rule Chain

$$\varepsilon_l^d = M_d^{src(l)} \gamma_{src(l)}^d, \forall n \in \mathcal{N}, \forall d =< r, t >\in \mathcal{D}^n, \forall l \in \mathcal{L} \tag{2.8}$$

where $M_d^{src(l)}$ is known information: $M_d^{src(l)} = 1$, if $\{o =< t', s >\in \mathcal{O}^{src(l)} : t' = t \wedge s \in \mathcal{S}_r^n\}$ is non empty, 0 otherwise. Constraints (2.8) force rules (collecting or not) to provide the notification that is being required.

Bound on Latency

$$\pi_{n_i,n_j}^{d,l} \geq \beta_{src(l)}^{n_i} + \beta_{dst(l)}^{n_j} + \varepsilon_l^d - 2, \forall n \in \mathcal{N}, \forall d \in \mathcal{D}^n, \forall l \in \mathcal{L}, \forall n_i, n_j \in \mathcal{N} \tag{2.9}$$

$$\varphi_{x_i,n_i}^d \geq \beta_{x_i}^{n_i} + \vartheta_{x_i}^d - 1, \forall n \in \mathcal{N}, \forall d \in \mathcal{D}^n, \forall x_i \in \mathcal{X}, \forall n_i \in \mathcal{N} \tag{2.10}$$

$$\begin{aligned} \sum_{x_i \in \mathcal{X}} \sum_{j=<n_j,r_j>\in \mathcal{I}^{x_i}} \sum_{n_i \in \mathcal{N}} \varphi_{x_i,n_i}^d \times L[n_i, n_j] + \\ + \sum_{l \in \mathcal{L}} \sum_{n_i \in \mathcal{N}} \sum_{n_j \in \mathcal{N}} \pi_{n_i,n_j}^{d,l} \times L[n_i, n_j] \leq LB, \\ , \forall n \in \mathcal{N}, \forall d \in \mathcal{D}^n \end{aligned} \tag{2.11}$$

Constraints (2.9) allow to determine if the latency of virtual link l, assuming its endpoints are placed at n_i and n_j, is to be considered. Constraints (2.10) determine if a rule is both a feeder and placed at a specific place, so that the latency associated with rule inputs can be accounted in the next set of constraints. The total latency is accounted at Constraint (2.11), where an upper bound LB is imposed. The LB is given information.

Lower Bound on Dominant Resource

$$\alpha_{x_i,\Delta(x_i,n)} + (1 - \beta_{x_i}^n) \geq \Delta^{\text{MIN}}, \forall x_i \in \mathcal{X}, \forall n \in \mathcal{N} \tag{2.12}$$

where Δ_n^{MIN} represents the lowest percentage of dominant resource allocated to rules being executed at node n. This is maximized by the objective function for fairness (regarding allocated resources) among rules.

Computational Resource Limitation

$$\sum_{x_i \in \mathcal{X}} \alpha_{x_i,\Delta(x_i,n)} \times d_{x_i,m} \leq c_{n,m}, \forall n \in \mathcal{N}, \forall m \in \{1, ..., M\} \tag{2.13}$$

These constraints avoid exceeding the available resources at processing nodes. All resources required by a rule are assumed to get the same percentage share as the dominant resource.

Non-negativity assignment to variables

$$\begin{aligned} 0 \leq \varepsilon_l^d, \alpha_{t_i,\Delta(x,n)}, \Delta^{\text{MIN}} \leq 1 \\ \gamma_v^d, \beta_v^n, \vartheta_x^d, \pi_{n_i,n_j}^{d,l}, \varphi_{x,n}^d \in \{0, 1\} \end{aligned} \tag{2.14}$$

The RP problem can be seen as rule selection process, followed by their placement in a way that Δ^{MIN} is maximized across all the devices available on the network. Δ^{MIN} less that 1 indicates that at least one of the demands did not obtained the required computational resources and, therefore, normal execution of demands is compromised. Δ^{MIN} equals 1 if all demands can be fully satisfied.

2.6 Performance Analysis

Dataset generation and scenario setup will be described prior to performance analysis.

2.6.1 Dataset Generation

Web Resource Pool

Each generated resource has a unique identifier and a state set, including all states it can take.

Node Pool

Each node has a unique identifier, a vector of available computational resources, a set of Web resources and a two-dimensional position. Node's computational resources and the position values are generated randomly[11] within defined lower and upper bounds. The distance between any two nodes is given by the Euclidean distance. Web resources are randomly selected from the resource pool generated in Section 2.6.1.

[11] Uniform distribution is used in every random generation/selection.

Atomic Task Pool

This is a pool of atomic tasks. Atomic task is the most basic computation function. Each atomic task has a unique identifier and a vector that indicates the amount of computational resources required to perform the task. These atomic computation functions are used to generate demands.

Demand Pool

A demand is intended to request the update of a Web resource with the output of a task, where a task includes one or more atomic tasks. Therefore, each demand will have an identifier, an array of atomic tasks and a <node, resource> destination tuple. These elements are extracted from the respective pools.

Rule Pool

This pool is generated having the demand pools as a basis. That is, the power set on the array of atomic tasks is generated with *js-combinatorics*[12], a Node.js package. The rule's demand vector is built by summing the corresponding atomic tasks's demand vectors.

Collecting Rule Pool

Any rule combination matching a demand will give rise to one or more `Collecting Rules`.

2.6.2 Scenario Setup

To assess the impact of `Collecting Rules`, a set of RP problem instances is generated and solved using CPLEX[13]. The following parameters are used to generate the problem instances.

- Resource related parameters:
  - Resource pool size: 15
- Task related parameters:
  - Atomic task pool size: 30
  - Lower bound of an element in the demand vector of an atomic task: 40
  - Upper bound of an element in the demand vector of an atomic task: 50

[12] `https://github.com/dankogai/js-combinatorics`
[13] IBM ILOG CPLEX Optimizer

- Node related parameters:
  - Number of nodes: 10
  - X-axis position range 0-100
  - Y-axis position range 0-100
- Demand related parameters:
  - Minimum number of atomic tasks: 2
  - Maximum number of atomic tasks: 3
  - Number of demands: 5

A total of 50 datasets is generated resulting into 30 `Rules` and 25 `Collecting Rules` each (in average). To assess the impact of computational resource, each dataset is tested for different resource computational ranges. For the required computations resources, associated with tasks, a multiplicative factor is also tested. This is intended to model the overall reduction of the required computational resources when complex tasks are not fragmented. That is, aggregated computation requires fewer hardware resources in total. Finally, since the nodes in all datasets are scattered across a 100 by 100 square, the latency value selected is set to 150 units in order to ensure that each dataset is feasible. Overall, the tests consisted in observing the performance, under different computational resources and multiplicative factors, of two approaches called:

- `Rules`-only, where every demand is satisfied by a single `Rule` instance that performs all required tasks. Since no `Collecting Rules` are used, fragmentation into smaller rules is not allowed.
- `Collecting` approach, an extension of `Rules`-only, where a demand can be satisfied either by a single `Rule` instance, performing all required tasks, or a combination of smaller `Rules` and `Collecting Rules`. Such smaller `Rules` are called `Feeder Rules` because they feed (are inputs to) `Collecting Rules`. Contrarily to a `Rule`, a `Feeder Rule` does not perform all tasks requested by the demand.

2.6.3 Results and Discussion

Overall Performance

Table 2.1 shows the performance of `Collecting` and `Rules`-only approaches for the scenarios described previously. Table includes the corresponding Δ^{MIN} average value, denoted by (Δ_C^{MIN}) and (Δ_R^{MIN}), and difference between them, for each tested parameter

Table 2.1: Average Δ^{MIN} of `Collecting` (Δ_C^{MIN}) and `Rules`-only (Δ_R^{MIN}) approaches for different computational ranges and multiplicative factors. Green highlighted rows represent parameter configurations where Δ^{MIN} achieved 100%, i.e., the demands are satisfied in all tested datasets. Red rows indicate that network's computational resources are not enough to match demand needs.

	Factor: 0.7			Factor: 0.8			Factor: 0.9			Factor: 1.0		
Compute Range	Δ_R^{MIN}	Δ_C^{MIN}	Difference	Δ_R^{MIN}	Δ_C^{MIN}	Difference	Δ_R^{MIN}	Δ_C^{MIN}	Difference	Δ_R^{MIN}	Δ_C^{MIN}	Difference
1-50	40.80%	50.99%	10.19%	35.70%	48.76%	13.06%	31.73%	47.08%	15.35%	28.56%	45.63%	17.07%
11-60	52.09%	68.35%	16.26%	45.58%	65.29%	19.71%	40.52%	62.39%	21.87%	36.58%	60.89%	24.31%
21-70	64.75%	87.50%	22.75%	56.69%	83.14%	26.45%	50.39%	79.67%	29.28%	45.35%	78.14%	32.79%
31-80	76.36%	98.11%	21.75%	67.36%	96.57%	29.21%	59.87%	94.20%	34.33%	53.89%	92.88%	39.00%
41-90	85.68%	100.00%	14.32%	76.13%	100.00%	23.87%	68.02%	99.71%	31.70%	61.24%	99.48%	38.24%
51-100	95.26%	100.00%	4.74%	86.44%	100.00%	13.56%	78.02%	100.00%	21.98%	70.48%	100.00%	29.52%
61-110	98.01%	100.00%	1.99%	91.63%	100.00%	8.37%	83.70%	100.00%	16.30%	76.46%	100.00%	23.54%
71-120	99.98%	100.00%	0.02%	97.59%	100.00%	2.41%	91.91%	100.00%	8.10%	84.85%	100.00%	15.15%
81-130	100.00%	100.00%	0.00%	99.63%	100.00%	0.37%	96.44%	100.00%	3.56%	90.17%	100.00%	9.83%
91-140	100.00%	100.00%	0.00%	99.71%	100.00%	0.29%	98.70%	100.00%	1.30%	94.84%	100.00%	5.16%
101-150	100.00%	100.00%	0.00%	100.00%	100.00%	0.00%	99.79%	100.00%	0.21%	98.27%	100.00%	1.73%
111-160	100.00%	100.00%	0.00%	100.00%	100.00%	0.00%	100.00%	100.00%	0.00%	99.74%	100.00%	0.26%
121-170	100.00%	100.00%	0.00%	100.00%	100.00%	0.00%	100.00%	100.00%	0.00%	100.00%	100.00%	0.00%
131-180	100.00%	100.00%	0.00%	100.00%	100.00%	0.00%	100.00%	100.00%	0.00%	100.00%	100.00%	0.00%
141-190	100.00%	100.00%	0.00%	100.00%	100.00%	0.00%	100.00%	100.00%	0.00%	100.00%	100.00%	0.00%
151-200	100.00%	100.00%	0.00%	100.00%	100.00%	0.00%	100.00%	100.00%	0.00%	100.00%	100.00%	0.00%

Table 2.2: Computational range gap

	Factor: 0.7	Factor: 0.8	Factor: 0.9	Factor: 1
Gap	37.91%	47.81%	44.28%	48.11%

configuration. In general, the table shows that in every tested scenario the `Collecting` approach presents much better performance, given that it is able to achieve Δ^{MIN} equal to 100% using considerably less computational resources, when compared to `Rules`-only approach. Furthermore, results in Table 2.1 indicate that the multiplicative factor has direct impact over the performance of both approaches: a higher multiplicative factor implies that more computational resources are needed to satisfy the demands.

Let us use ρ_C^* and ρ_R^* to denote the computational ranges where `Collecting` and `Rules`-only approaches achieve 100% Δ^{MIN} for the first time, and let $\bar{X}(\rho_C^*)$ and $\bar{X}(\rho_R^*)$ denote the mid value of such ranges. When looking at the gap given by

$$\text{Gap} = \frac{\bar{X}(\rho_R^*) - \bar{X}(\rho_C^*)}{\bar{X}(\rho_C^*)}, \tag{2.15}$$

it is possible to see that the `Collecting` approach is able to satisfy the demands using 44.53% less computational resources than `Rules`-only approach. Table 2.2 summarizes such gaps for the tested multiplicative factors.

The performance of the `Collecting` approach can be explained by analyzing Figure 2.6 in combination with Figure 2.7. Figure 2.6 is a plot showing the Δ^{MIN} values produced by `Collecting` and `Rules`-only approaches for the tested multiplicative factors and node's computational resources. On the other hand, Figure 2.7 shows the type of rules chosen by the `Collecting` approach for the solutions plotted in Figure 2.6. To be precise, it shows how many `Rules`, `Collecting Rules` and `Feeder Rules` are

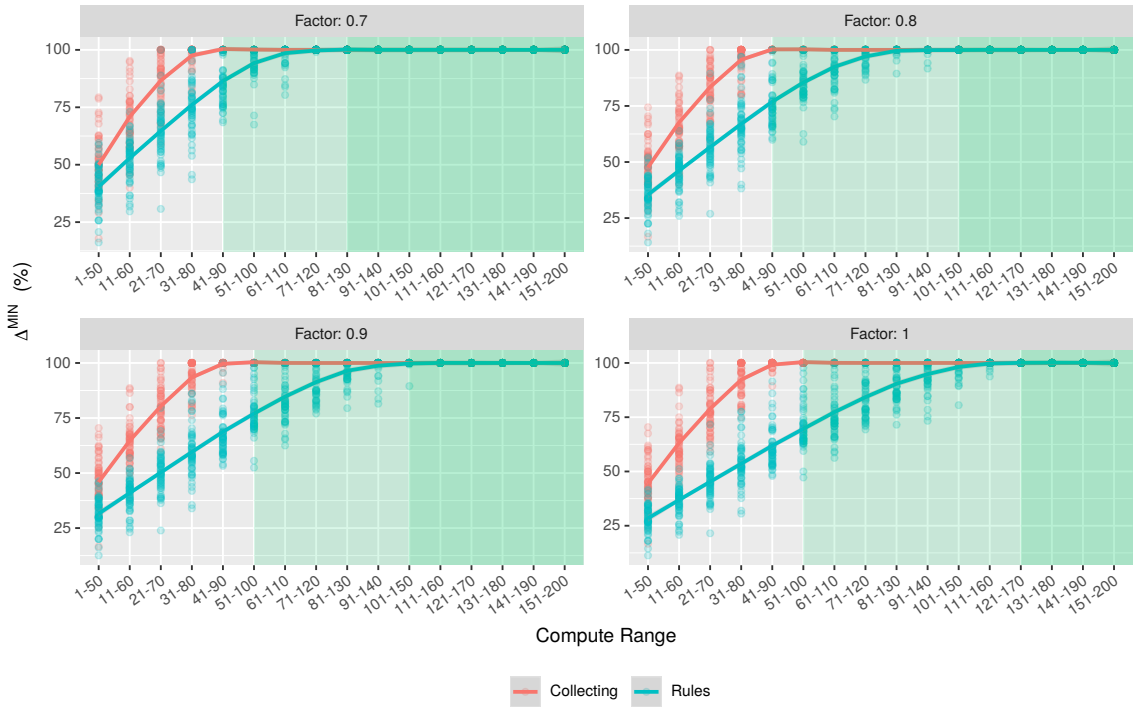


Figure 2.6: Evolution of Δ^{MIN} for different computational ranges and rule multiplicative factors. Each dot in the plot depicts the solution value of each test, while the line represents the overall trend. Light green colored area outlines the computational ranges where the `Collecting` approach achieve Δ^{MIN} equal to 100% in all tested datasets. Darker green corresponds to the `Rules`-only approach.

used to satisfy the demands in each test.

Looking at both plots it is visible that the performance improvement, in networks with limited computational resources, of the `Collecting` approach over the `Rules`-only approach is due to the use of `Feeder Rules` (blue dots in Figure 2.7), which require less computational resources and, therefore, give more freedom for CPLEX to optimize their placement across the available nodes. As the node's computational capabilities increase, the optimization process leans towards the use of `Rules` (green dots in Figure 2.7). By using these rules, data goes through a single node, for processing, reducing the overall latency involved in data delivery, when compared with the use of `Feeder Rules`. Hence, more processing places can be explored by Constraint (2.11) for a higher Δ^{MIN} to be achieved. The shift to `Rules` is mostly visible at the beginning of the highlighted area in Figure 2.7, i.e., area where all tested datasets are capable to achieve Δ^{MIN} equal to 100%.

Network Usage

Let us consider a 802.15.4 network with a Maximum Transferable Unit (MTU) of 127 bytes and CoAP's minimal packet header structure of 12 bytes (fixed size 4-byte header and a 8-byte token), leaving only 115 bytes for the payload. Let us also assume the

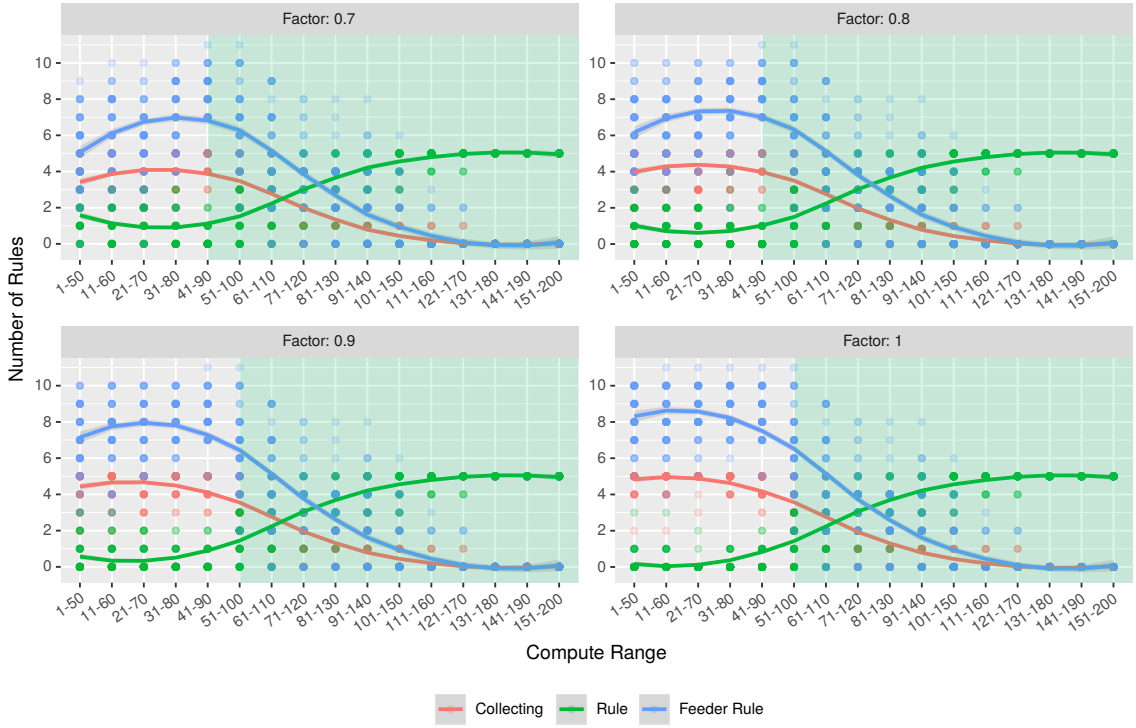


Figure 2.7: Type of rules chosen by the `Collecting` approach, for solutions plotted in Figure 2.6. For each dot in Figure 2.6, there are three dots representing the number of different rules that are used to obtain the actual solution. Color intensity of the dots reflects the frequency of occurrences, while the line represents the overall trend.

use of CoAP block-wise transfer mechanism [84] when the payload size is bigger than 115 bytes, and use of 12 byte acknowledgement (ACK) packets to acknowledge block reception. Finally, assume that all nodes are one-hop away from each other. Given these assumptions, the total number of transmitted bytes by each synchronization approach can be expressed as:

$$\mathrm{Tx}^{Bytes} = \Phi^{init}|\mathcal{S}| + \Phi^{inter}|\mathcal{M}| + \Phi^{final}|\mathcal{D}| + 12|\mathcal{A}| \tag{2.16}$$

where:

- Φ^{init} is the average size (in bytes) of the payload at input resources.
- $|\mathcal{S}|$, where $\mathcal{S} = \{(j,x) : \forall j \in \mathcal{I}^x, \forall x \in \mathcal{X}\}$, is the number of input links ($input\ resource \rightarrow rule$).
- Φ^{inter} is the size (in bytes) of payload exchanged between `Feeder` and `Collecting` rules. The packets containing this payload are of constant size as they only indicate whether a rule is satisfied or not. Assuming a CBOR encoded SenML notification ({`"vb": true`} or {`"vb": false`}) with a size of 5 bytes, the total size of inter rule packets is of 17 bytes.

- $|\mathcal{M}|$, where $\mathcal{M} = \{(x, c_j), (c_i, c_j) \in \mathcal{L} : \forall x \in \mathcal{X}, \forall c_i, c_j \in \mathcal{C}\}$, is the number of inter rule links ($feeder \rightarrow collecting$).
- Φ^{final} is the average size (in bytes) of notifications to be sent to the destination resources.
- $|\mathcal{D}|$, where $\mathcal{D} = \{(x, n), (c, n) \in \mathcal{L} : \forall x \in \mathcal{X}, \forall n \in \mathcal{N}, \forall c_i, c_j \in \mathcal{C}\}$, is the number of final links ($rule/collecting \rightarrow destination\ resource$).
- $|\mathcal{A}|$ is the total number of acknowledgement (ACK) packets involved in the synchronization process.

Overall, the difference between the number of bytes sent by each approach can be expressed as:

$$Diff = \text{Tx}_{Rules}^{Bytes} - \text{Tx}_{Collecting}^{Bytes} \tag{2.17}$$

In both resource synchronization approaches the $|\mathcal{D}|$ and the Φ^{final} are the same, as the notifications sent to the destination resources do not change. Therefore, the only factors that affect the $Diff$ value are the number of $|\mathcal{S}|$ and $|\mathcal{M}|$ links and the size of the Φ^{init} payload. To assess how these elements impact the utilization of network resources, a set of tests are performed. The tests consist in extracting the number of $|\mathcal{S}|$ and $|\mathcal{M}|$ links (summarized in Table 2.3) from the solutions obtained in Section 2.6.3 and vary the value of Φ^{init}. Due to similarity and for brevity purposes, only the results for the multiplicative factor equal to 1 are shown.

The Figure 2.8 is a surface plot showing the relationship between the payload size sent by source resources (Φ^{init}), the computational ranges and the difference in bytes between `Collecting` and `Rules`-only approaches, as expressed in Expression 2.17. For small computational ranges and Φ^{init} values the `Rules`-only approach requires less bytes that the `Collecting` approach. However, as the Φ^{init} becomes larger and approaches the 100 bytes per notification, the `Collecting` approach becomes more efficient. This effect becomes especially visible when the Φ^{init} is equal to 150 bytes, as is causes a spike on plot's surface. On the other hand, higher values of compute range cause a shift in the `Collecting` approach by making it give preference to `Rule` instances to satisfy demands (as described in Section 2.6.3). This shift leads to a reduction of $|\mathcal{M}|$ and increase of $|\mathcal{S}|$ links causing the decrease of the $Diff$ value, which ultimately becomes equal to zero meaning that both approaches satisfy demands using the same rule resources. Taking everything into account, the optimal location for the `Collecting` approach is when the compute range is equal to 51-100 and $\Phi^{init} \geq 150$ bytes as, under these circumstances,

Table 2.3: Average number of initial ($|\mathcal{S}|$) and inter rule ($|\mathcal{M}|$) links used by `Collecting` and `Rules`-only approaches to obtain solutions in datasets with an multiplicative factor equal to 1. Green highlighted rows represent parameter configurations where Δ^{MIN} achieved 100%, i.e., the demands are satisfied in all tested datasets. Red rows indicate that network's computational resources are not enough to match demand needs.

	`Rules`-only		`Collecting`	
Compute Range	$\|\mathcal{S}\|$	$\|\mathcal{M}\|$	$\|\mathcal{S}\|$	$\|\mathcal{M}\|$
1-50	12	0	9	11
11-60	12	0	9	11
21-70	12	0	9	12
31-80	12	0	9	12
41-90	12	0	10	11
51-100	12	0	11	8
61-110	12	0	11	6
71-120	12	0	11	4
81-130	12	0	12	3
91-140	12	0	12	3
101-150	12	0	12	1
111-160	12	0	12	1
121-170	12	0	12	1
131-180	12	0	12	0
141-190	12	0	12	0
151-200	12	0	12	0

the `Collecting` approach is able to satisfy all the demands (Δ^{MIN} equals 100%) while transmitting less bytes that `Rules`-only approach.

In general, the `Collecting` approach becomes a more attractive option when the size of the payload flowing through the extra initial links ($\mathcal{S}$) in `Rules`-only approach is larger than the payload flowing through the inter rules links ($\mathcal{M}$) in `Collecting` approach.

2.6.4 Rule-based vs Broker-based Event Processing

At the moment of writing the `Rule` mechanism is still an ongoing effort while MQTT-based approaches are the most common way of building dynamic and reactive IoT applications[14]. For this reason it becomes necessary to compare CoRE-based `Rule` and MQTT-based event processing. Note that MQTT-based applications use the same `observe-evaluate-actuate` execution logic and, therefore, these approaches can be compared. However, while both synchronization mechanisms share the same execution chain, they operate over different transfer protocols (CoAP and MQTT) that follow different communication patterns (request/response and publish/subscribe), which ends up determining the execution flow during resource synchronization. Being broker-based, where direct

[14] See "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP" [85]

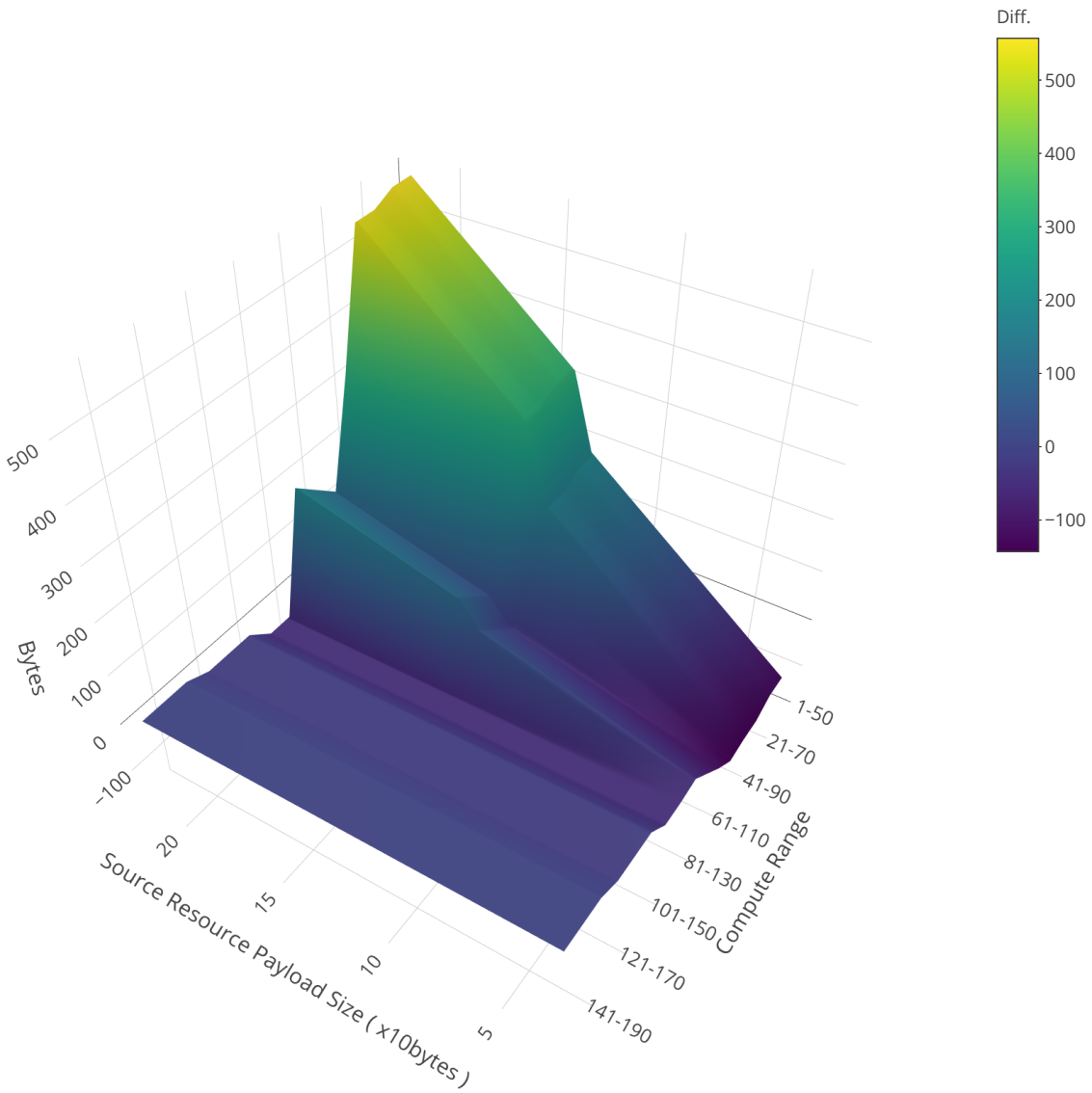


Figure 2.8: Difference of total transmitted bytes of `Collecting` and `Rules`-only approaches during the resource synchronization with an multiplicative factor equal to 1. Positive values indicate areas where the `Collecting` approach uses less bytes than `Rules`-only approach.

client-to-client communication is not supported, MQTT applications usually require the exchange of more messages, when compared with CoAP-based approaches, to achieve the same goal. However, despite the theoretical superiority of CoAP over MQTT, the existing research shows that MQTT can outperform CoAP in certain scenarios. For example, in [86] authors show that when the payload size is larger than 300 bytes and the packet loss rate is 25% MQTT generates less overhead than CoAP. On the other hand, for high packet loss rate CoAP performs better in terms of delay. Similar conclusions were obtained in [85, 87–89]. This may show that the main advantage of CoAP over MQTT is its flexibility. CoAP has built-in support for content negotiation allowing devices to probe each other prior to data exchange and, therefore, it is better suited for ad hoc interactions. Also, in addition to request/response pattern, CoAP has Observe mechanism that opens the possibility to create CoAP brokers and, therefore, provide all the benefits of broker-based communications. IETF's CoAP Publish-Subscribe Broker protocol draft [90] is the first step in this direction. Preliminary results [91] show that CoAP broker has greater packet delivery rate than MQTT in scenarios where the packet loss equals to 40%. However, further research is required to confirm this findings. In summary, there is no clear winner between CoAP-based and MQTT-based approaches and the choice will depend on the scenario in question.

2.7 Conclusions

This chapter presents a deep overview of current CoRE-related standards and Internet-Drafts for resource synchronization and event processing in constrained networks. It also proposes a distributed, scalable and resilient resource synchronization mechanism, called `Collecting Rule` approach, that is based entirely on the existing standards and drafts. The latter is a relevant issue as most of the literature on event processing omit the underlying protocol stack. Furthermore, the Rule Placement (RP) problem is presented and is mathematically formulated, ensuring proper rule execution and a fair utilization of device resources. Mathematical results show that in some scenarios the proposed `Collecting Rule` approach is capable of satisfying user's demands using 44.53% less computational resources when compared against its centralized counterpart, currently under evaluation to be a part of OCF core specification, called `Rules`-only approach. Additionally, results show that the proposed synchronization mechanism uses less network resources than the `Rules`-only approach when the payload of resources at the beginning of synchronization chain (source) is bigger than 150 bytes. In summary, results obtained from mathematical model show that the proposed distributed synchronization mechanism is a viable alternative to `Rules`-only approach in constrained environments. In less constrained environments, the model gives preference to `Rules` in order to avoid sending the data through multiple processing points. Thus, the `Collecting` approach

can be seen as a complement and not a replacement of `Rules`-only approach.

CHAPTER 3

Attention-Based Model and Deep Reinforcement Learning for Distribution of Event Processing Tasks

***Abstract*:** Event processing is the cornerstone of the dynamic and responsive Internet of Things (IoT). Recent approaches in this area are based on representational state transfer (REST) principles, which allow event processing tasks to be placed at any device that follows the same principles. However, the tasks should be properly distributed among edge devices to ensure fair resources utilization and guarantee seamless execution. This chapter investigates the use of deep learning to fairly distribute the tasks. An attention-based neural network model is proposed to generate efficient load balancing solutions under different scenarios. The proposed model is based on the Transformer and Pointer Network architectures, and is trained by an advantage actor-critic reinforcement learning algorithm. The model is designed to scale to the number of event processing tasks and the number of edge devices, with no need for hyperparameters re-tuning or even retraining. Extensive experimental results show that the proposed model outperforms conventional heuristics in many key performance indicators. The generic design and the obtained results show that the proposed model can potentially be applied to several other load balancing problem variations, which makes the proposal an attractive option to be used in real-world scenarios due to its scalability and efficiency.

3.1 Introduction

Event processing is a crucial element in dynamic and reactive Internet of Things (IoT) applications, as it allows to derive real-time (or near-real-time) conclusions from data. An event processing task, usually mapped into a web request (e.g., HTTP), follows a simple `observe-evaluate-actuate` pattern. As the name suggests, this pattern involves performing three tasks: observe one or many devices/sensors, evaluate the produced data and, if the user-defined condition is satisfied, notify one or many actuators. With the recent emergence of the Web of Things (WoT) [16,18,33,34] concept, where every device has its

own internet protocol (IP) address and every device resource is addressable by uniform resource identifier (URI), the `observe-evaluate-actuate` chain operates over a set of URI endpoints.

Event processing has relied primarily on cloud computing, which virtually has an unlimited amount of storage and computational resources. Centralized processing, alongside with the virtualization of both storage and computational resources, offers flexibility and efficiency by scaling up and adapting the system to different situations. However, with the ever-increasing amount of data being produced and consumed directly at the network edge, centralized (cloud) computing ends up becoming a limitation because additional IoT application requirements can not be satisfied. These requirements are related with delay sensitivity, data volume, uplink costs, non-interruption of service in environments with intermittent connectivity, privacy, and security [92].

The need to satisfy the above-mentioned requirements led to the emergence of a distributed computing paradigm, called edge computing. In this new paradigm, substantial computational and data storage resources are located near the mobile devices, sensors and actuators. Edge computing can be seen as a strategy to provide a uniform computation/storage environment all the way from the core data centers to physical locations near users and data. However, unlike the cloud, edge computing has a more dynamic and distributed computing nature, which results in distributed and ad-hoc event processing chains.

Despite the above-mentioned differences, edge and cloud will have a common background in the underlying technologies. In an Internet-Draft called "IoT Edge Challenges and Functions" [93] Internet Engineering Task Force (IETF) emphasizes that "*virtualization platforms enable the deployment of virtual edge computing functions, including IoT gateway software, on servers in the mobile network infrastructure, in edge or regional data centers*". At the moment of writing, there are several initiatives that are moving in this direction. As an example, Edge Virtualization Engine (EVE) [94] aims to create an open edge computing engine that enables the development, orchestration and security of cloud-native and legacy applications on distributed edge computing nodes. It also aims to provide support for containers and clusters (e.g., Docker, Kubernetes), virtual machines, unikernels and offer a flexible foundation for IoT edge deployments with choice of any hardware, application and cloud. Another example is the EdgeX Foundry [95] service platform that also aims to provide management, data persistence or streaming near the edge. This project also provides a dedicated rule engine API for the creation of event processing chains. MobiledgeX [96] is an edge-cloud platform that provides an environment for the developers to deploy software (e.g., as software containers) on the edge. All these initiatives seek to bring orchestration, virtualization, load balancing techniques and technologies from the cloud to the edge. This means that in order to understand how event processing tasks will be handled and distributed across the edge, it is important to see how

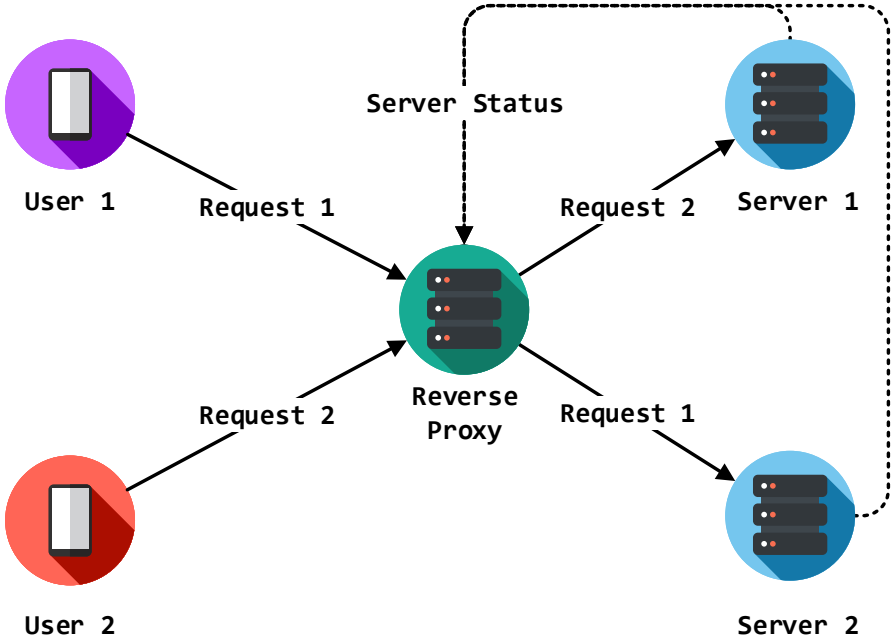


Figure 3.1: An overview of reverse proxy.

this is currently being handled by the cloud.

3.1.1 Adequate Resource Placement and Load Balancing

In typical application deployments, devices/servers (virtual or physical) are placed behind reverse proxies like NGNIX [97], Traefik [98] or Moleculer API Gateway [99]. A reverse proxy, illustrated in Figure 3.1, is a type of server that typically sits behind the firewall in a private network and distributes the incoming client requests among the available back-end servers. Common features that reverse proxies provide include load balancing, web acceleration (e.g., caching, SSL encryption), security and anonymity. In case of load balancing there are several balancing strategies that can be used, each focusing on a specific aspect of Quality-of-Service (QoS). NGNIX, for example, offers round-robin, least-connected, ip-hash [100]; Traefik at the moment only supports round-robin method [101]; Moleculer API Gateway offers round-robin, random, CPU-usage based and sharding [102]. These load balancing strategies do not provide optimal solutions because these are prohibitively expensive to obtain in real-time. Instead, these strategies trade the quality of solution for the response time, i.e., these strategies are fast but the solutions that they provide can be sub-optimal. Moreover, static optimization strategies are not adequate because the environment is dynamic in terms of available servers and user requests.

The arrival of new user requests is usually irregular. That is, bursts of requests during short periods of time can be followed by periods of slowdown. To address this, containerized environments have a container manager (e.g., Kubernetes) that provides autoscaling of the number of servers, ensuring the creation (or removal) of replicas according to the

system load [103]. Similar dynamics are expected at the network edge, and projects like StarlingX [104] solve them by providing tools for management, orchestration and scaling of distributed edge-cloud computing devices.

From the above discussion, it is clear that load balancing and resource planning strategies will play a crucial role in edge computing, together with strategies for the appropriate placement of event processing tasks. Any strategy should be scalable and adaptable to the dynamics of the system: number of incoming requests, number of servers and resources available at that time. In general, the problem of distributing event processing tasks boils down to the multiple knapsack problem, which is known to be NP-hard [105]. Developing handcrafted heuristics for such problems can be challenging, especially under dynamic environments and heterogeneous key performance indicators (KPIs). On the other hand, machine learning (ML) solutions are envisioned as the right answer to an ever increasing complexity of networks and the web [1]. ML approaches are able to learn new distribution strategies which makes them a compelling choice.

3.1.2 Reinforcement Learning as a Solution Framework

The most successful applications of ML methods (e.g. natural language processing (NLP), computer vision) fall under the umbrella of supervised learning (SL), a process that consists in learning to map a set of inputs to output labels. However, applying supervised ML methods to placement problems, or any other combinatorial problem, is troublesome as it is almost always impossible to obtain the output labels. Moreover, even if the labels are provided and a neural network model is trained in a supervised way, it usually has poor generalization to other problem instances [106].

Reinforcement learning (RL), on the other hand, is an attractive approach to solve combinatorial problems because, unlike SL, it does not require labels. RL can be seen as a trial and error process in which an agent interacts with an environment via a sequence of actions and, in return, receives feedback in the form of reward signals. The goal of the agent is to learn an action selection strategy in a way that the total reward is maximized. A continuous interaction with the environment allows the agent to explore several (millions of) possible environment configurations and to develop a strategy that ensures that the maximum reward is obtained in every possible situation, i.e., to generalize.

Recent attempts in integrating deep learning (DL) with RL showed breakthrough results in video games [107], board games [4, 5], robotics and many more areas [7]. This new learning paradigm is widely known as deep reinforcement learning (DRL). In this work we investigate how DRL can be used by reverse proxy servers to distribute event processing tasks among heterogeneous edge devices available for processing.

3.1.3 Motivation and Contributions

The idea of using DRL to solve load balancing and distribution problems has recently attracted many researchers. Surveys show that there are several attempts, with different levels of success, to solve these kind of problems [108, 109]. However, the vast majority of these attempts do not solve problems of variable sizes. The problem is treated as a fixed-sized optimization problem where a neural network is trained to map problem instances to solutions. For instance, in the distribution problem illustrated in Figure 3.1, the number of incoming requests and the number of available servers are assumed to be fixed. Then, during the development (design and training), the model is trained to learn to map the inputs to outputs. This is followed by testing, where the network's performance is measured on solving problems of the same size that it was trained upon and, finally, reporting the obtained results. However, despite the results reported in the above mentioned surveys, the majority of those models have little practical use in real life deployment because a simple change in input size would make those models invalid.

We argue that the assumption of a fixed problem size is problematic and, usually, does not hold true in dynamic environments like IoT and edge. The number of incoming requests and available nodes is dynamic and changes over time. Therefore, the main limitation of the existing literature that involves DRL, load balancing and distribution problems is the inability of the proposed neural networks to adapt to the dynamics of the environment without retraining. In other words, the problem is the scalability of the proposed neural networks.

This works focuses on developing a scalable DRL model that is able to solve problem instances of variable size without retraining. The main contributions of this chapter are as follows:

1. A study is made on how event processing tasks can be distributed across edge devices. Three different distribution criteria are studied, each being formalized mathematically.

2. A new neural network architecture is proposed for the distribution of event processing tasks, while ensuring load balancing. The key feature of the proposed model is its scalability, which means that it does not require retraining every time the number of incoming requests or servers, available for processing, change.

3. A comparative study against the optimal solutions and several baseline heuristics is carried out. Obtained results indicate that the proposed model can generate high quality solutions to problems up to five times larger that the ones it was trained upon.

The remainder of this chapter is organized as follows. In Section 3.2, existing attempts of using RL in load balancing problems are discussed. A review of neural combinatorial

optimization is also carried out. The event task distribution problem is mathematically formulated in Section 3.3. Section 3.4 presents the proposed neural network architecture and is organized as follows: i) Section 3.4.1 presets a brief RL and DNN background; ii) Section 3.4.2 presents the Markov decision process (MDP) representation of the problem; iii) Section 3.4.3 describes the detailed design of the proposed neural network architecture. Section 3.5 describes the training method alongside with the hyperparameters used to obtain the model and evaluates its performance. Finally, in Section 3.6 conclusions are drawn.

3.2 Related Work

RL has been applied in several areas of IoT and edge computing for different purposes, which range from the actuation control (e.g, greenhouse climate control) to resource control (e.g., minimization of communication delay, energy consumption, hardware resources) [108, 109]. However, most of the literature uses deep neural network (DNN) architectures that are not able to scale or adapt to the dynamics of the environment, e.g., increase in the number of edge devices, base stations, sensors or number of requests. That is, the action space is fixed, which limits the practical use of DNNs in realistic scenarios. The following subsections provide an overview of the application of RL to IoT, edge computing and load balancing. Then, ongoing research to tackle combinatorial optimization problems, using RL, is presented [110, 111].

3.2.1 Reinforcement Learning in Edge Computing

In [112] authors propose a DRL solution to minimize the total cotask completion time. A cotask is a task, generated by an IoT device, that can only be completed when all of its constituent sub-tasks are finished. Authors use a DNN architecture with two heads, one that predicts the offloading location of the task and another for the prediction of the cotask completion time. The obtained results show that the proposed model is able to outperform several baseline heuristics. However, the proposed architecture requires retraining whenever the network or the input size changes.

In [113] an RL method is used to distribute the tasks, produced by different user devices, among a set of edge servers. To deal with the combinatorial action space (any task can be placed at any available edge server), authors introduce a multi-agent algorithm where each Deep Q-Network (DQN) [107] agent makes an action that corresponds to the location where the task will be offloaded. The DQN architecture, used in this work, requires retraining once the state or action space changes, i.e., once an additional edge server is connected to the system.

In [114] mobility load balancing in self-organizing networks is investigated. The goal

is to design a handover scheme that transfers a mobile user from its service cell to a neighbor cell that can handle the incoming traffic. The authors use an off-policy deterministic policy gradient (OPDPG) method [115] and propose an asynchronous parallel learning framework to improve the training efficiency in a collaborative manner. The obtained results show that the RL method is able to outperform several baseline heuristics. However, once again, this architecture is designed for fixed-size communication networks, i.e., any change in the number of cells requires retraining the network.

In [116] the authors propose an RL agent to offload computationally intensive tasks, generated by user equipment (UE), to mobile edge computing (MEC) servers, while maximizing the mobile operator revenue and minimizing the energy consumption and time-delays. In the investigated environment the authors considered a queuing model, states of energy harvesting batteries and down-link transmit power costs. The authors conclude that their approach has better performance than the policy-gradient and Q-learning algorithms. The main limitation of this work is the fact that the authors only studied a scenario with a fixed number of MEC servers, which is not always the case in reality. Moreover, any change in the number of MEC servers requires retraining the agent.

Several other works [117–121] used DQN-based models in offloading and load balancing problems. Despite the obtained results, all of these proposals have a common weakness that drastically limits their use in real world deployments. Any change in the state or action space sizes imply retraining the network from scratch. Overall, DQN and its variants are more suitable for static environments with fixed state and action spaces but they have limited uses in dynamic environments

3.2.2 Combinatorial Optimization with Reinforcement Learning

One of the first attempts to solve combinatorial problems with neural networks was presented in [122]. The authors propose a neural architecture, called Pointer Networks (Ptr-Net), which is based on sequence-to-sequence (seq2seq) [123] models with attention mechanisms [124, 125] that are commonly used in NLP. Ptr-Net is an encoder-decoder architecture where recurrent neural networks (RNNs), usually long short-term memory network (LSTM) [126] or gated recurrent unit (GRU) [127], are used to process the input sequence at the encoder and to generate the output sequence at the decoder. Although in this work the Ptr-Net was trained in a supervised way, authors showed that a single architecture with the same hyperparameters can be used to solve different combinatorial problems. The fundamental breakthrough of Ptr-Net is its ability to deal with variable output space without the need for retraining the network. Moreover, the authors showed that the Ptr-Net trained on combinatorial problems of small size is capable to generalize to much larger problems without considerable degradation in the quality of generated solutions.

In [106] the authors used Ptr-Net in combination with RL to solve combinatorial problems, such as the traveling salesman problem (TSP) and the knapsack problem. Authors use RL methods because it is difficult to have access to labels (associated with optimal solutions), as combinatorial problems are usually NP-Hard. The obtained results show that the developed RL agent is capable of finding near-optimal solutions for TSP with up to 100 nodes and for up to 200 items in knapsack problem. In this work the authors also made an empirical analysis between supervised training and RL and reached the conclusion that Ptr-Net networks trained in a supervised way have poor generalization when compared to a Ptr-Net-based RL agent that explores different solutions and observes their corresponding rewards.

In [128] the authors proposed a model, inspired on Ptr-Nets and seq2seq, to solve vehicle routing problem (VRP). Authors removed LSTM from the encoder as they argue that LSTM, and other RNNs, are only necessary when dealing with sequential data. After training, the proposed model was able to generalize well and generate high-quality solutions for all problems sampled from the same distribution that was used during training. Moreover, authors tested the model's ability to handle variable problem sizes. They conclude that their model performs well when training and testing problem sizes are close to each other. However, they report a degradation in performance when the sizes of the testing instances are substantially different from the ones used for training.

The work in [128] is extended by [129], where a model capable of solving several variations of VRP and TSP is proposed. The authors replace LSTM networks by the Transformer [130] and results show that the proposed model is able to achieve (near) optimal solutions for problems of size 20, 50 and 100 nodes.

3.3 The Problem of Event Processing

Prior to presenting the proposed DRL solution, it is necessary to clearly define the problem. This section briefly describes the technological background of event processing, which is then followed by its mathematical formalization.

3.3.1 Rule Synchronization Mechanism

There are multiple Internet-Drafts that attempt to standardize event processing in WoT, many of them have been reviewed and analyzed in our previous work [30]. In this chapter the focus will be on the most recent proposal, called `Rule`, which was also discussed in [30].

A `Rule`, illustrated in Figure 3.2, is a regular web resource, based on RESTful principles, that allows to create complex many-to-many event processing chains. Multiple sources can be observed (e.g., sensors) and, after condition evaluation, multiple destina-

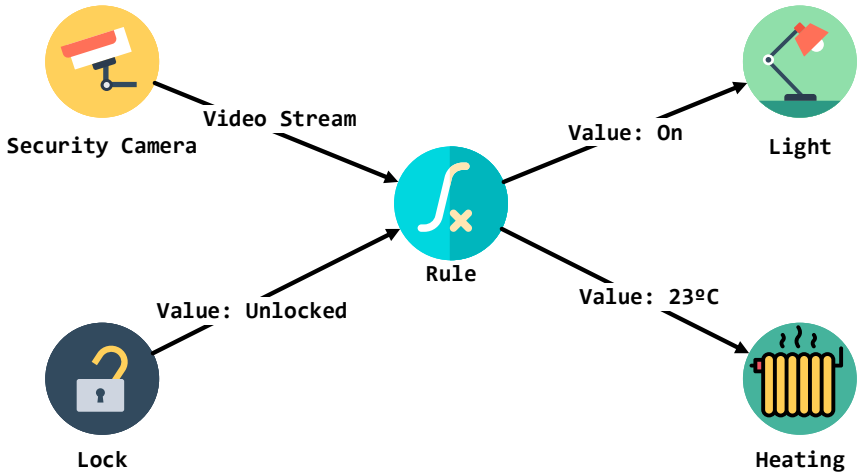


Figure 3.2: An example of a `Rule` with two inputs and two outputs.

tion endpoints can be updated (e.g., actuators). Being a regular web resource means that `Rules` can be placed at any device that supports a `Rule` API and is available for processing. The `Rule` is a continuously running task that is constantly observing resources and processing the data being sent. To stop it, and to release the resources that it is consuming, the user must send an appropriate request (e.g., HTTP DELETE).

End-users are not expected to create `Rules` by hand. Applications are expected to be available (e.g., `Rules` store) where users can find pre-defined `Rules`, together with related meta-data (e.g., CPU, RAM and storage requirements), i.e., `Rule` profiles. The inputs and outputs of a pre-configured `Rule` will have to be defined according to user's need.

Running `Rules` at the edge, where there will be physical, virtual, and containerized nodes with different computational capabilities, which means that these tasks must be carefully distributed.

Definition 1 (Rule Distribution (RD) Problem) *Given a set of `Rule` profiles, decide for the best `Rule` distribution across devices while taking into account randomly arriving `Rule` requests, random number of available edge devices (each having its random amount of available CPU, RAM and storage resources) and QoS requirements.*

3.3.2 Assumptions and Notation

To have a clear understanding of how to solve the RD problem optimally, assumptions and notation have to be clarified. This is followed by the mathematical formalization of the RD problem considering three variants. Such mathematical formalization is important not only to clearly define the problem, but also to have a reference when evaluating the proposed RL-based agent.

Definition 2 (Node) *A virtual, containerized or physical device exposing a* `Rule` *API, being capable of processing a specific set of* `Rules`*. The set of nodes able to host* `Rule` x *is denoted by* $\mathcal{N}^x$*, while the overall set of heterogeneous nodes is denoted by* $\mathcal{N}$*.*

Definition 3 (Computational Resource) *Hardware resource (e.g., CPU, RAM, memory) available at nodes. The amount/capacity of* n*'s computational resources,* $n \in \mathcal{N}$*, is given by vector* $\boldsymbol{c}^n = [c_{n,m}]$*, where* $m \in \{1, ..., M\}$ *and* M *is the total number of computational resources considered.*

Definition 4 (Rule) *Web resource able to evaluate an expression or performing complex processing over certain inputs. Assuming* $\mathcal{X} = \{x_1, x_2, ..., x_{|\mathcal{X}|}\}$ *to be a set of* `Rules`*, each* x_i *has a demand vector* $\boldsymbol{d}^{x_i} = [d_{x_i,m}]$*,* $m \in \{1, ..., M\}$*, containing computational resource requirements.*

Definition 5 (Critical Resource) *Given a set of rules* $\mathcal{X}' \subset \mathcal{X}$ *under execution at node* $n \in \mathcal{N}$*,* n*'s critical resource is given by* $\Omega^n = \arg\max_{m \in \{1,...,M\}} \{\frac{\sum_{x' \in \mathcal{X}'} d_{x',m}}{c_{n,m}}\}$.

In the following section three variants of the RD problem are formalized. Each variant tries to optimize specific performance indicators: number of allocated `Rules`, critical resource and/or number of active nodes. The variables that may be involved are the following:

ω_x	One if `Rule` $x \in \mathcal{X}$ is to be executed, which requires finding a node to place it; zero otherwise.
β_x^n	One if `Rule` $x \in \mathcal{X}$ is to be placed at node $n \in \mathcal{N}^x$; zero otherwise.
Ω^n	Critical resource at node $n \in \mathcal{N}$.
Ω^{MAX}	Most critical resource among all nodes at the network.
Θ^n	One if node $n \in \mathcal{N}$ is in use; zero otherwise.

3.3.3 Mathematical Formalization

Greedy Optimizer

The performance indicator to be optimized is the number of `Rules` placed/distributed across nodes, which should be maximized. This is expressed as follows.

– Objective Function:

$$\text{Maximize} \sum_{\{x \in \mathcal{X}\}} \omega_x \tag{3.1}$$

– Placement of Rules:

$$\sum_{\{n \in \mathcal{N}^x\}} \beta_x^n = \omega_x, \forall x \in \mathcal{X} \tag{3.2}$$

These constraints ensure that there is a single location for each `Rule` that is to be executed.

– Computational Resource Limitation:

$$\sum_{\{x \in \mathcal{X}\}} d_{x,m} \times \beta_x^n \leq c_{n,m}, \forall n \in \mathcal{N}, \forall m \in \{1, ..., M\} \tag{3.3}$$

where $d_{x,m}$ is known information. These constraints avoid exceeding the available computational resources of nodes.

– Non-negativity assignment to variables:

$$\omega_x, \beta_x^n \in \{0, 1\}. \tag{3.4}$$

Critical-Aware Greedy Optimizer

The main performance indicator to be optimized is the number of `Rules` that are placed/distributed across nodes, which should be maximized while ensuring that load is fairly distributed (most critical resource as a reference). Thus, fair distribution is the secondary performance indicator to be optimized. This kind of optimization is important in cooperative scenarios when edge devices are provided by different owners that trade their computational resources for some reward (e.g., financial benefits). Service providers hold reverse proxy servers, which exposes the `Rule` API to the end-users. The computational resources are provided by the device owners.

– Objective Function:

$$\text{Maximize} \sum_{\{x \in \mathcal{X}\}} \omega_x + \Omega^{\text{MAX}} \tag{3.5}$$

This objective function ensures that the largest number of `Rules` is placed/distributed and, at the same time, that it is done in a fair way, so that `Rules` have the best (and similar) working conditions.

– Placement of Rules: Expression (3.2).

– Computational Resource Limitation: Expression (3.3).

– Obtaining Critical Resource:

$$\Omega^n \leq c_{n,m} - \sum_{\{x \in \mathcal{X}\}} d_{x,m} \times \beta_x^n, \forall n \in \mathcal{N}, \forall m \in \{1, ..., M\} \tag{3.6}$$

where $d_{x,m}$ is known information. These constraints finds the critical resource at each node. Finding the most critical resource among all nodes requires:

$$\Omega^{\text{MAX}} \leq \Omega^n, \forall n \in \mathcal{N} \tag{3.7}$$

– Non-negativity assignment to variables:

$$\omega_x, \beta_x^n \in \{0, 1\}; 0 \leq \Omega^n, \Omega^{\text{MAX}} \leq 1. \tag{3.8}$$

Cost-Aware Greedy Optimizer

The main performance indicator to be optimized is the number of `Rules` that are placed/distributed across nodes, which should be maximized while ensuring that the number of nodes in use (hardware cost) is minimized. Thus, the cost is the secondary performance indicator to be optimized. This optimization is important when the `Rule` API provider is responsible for both the reverse proxy and nodes, but is using rented hardware to offer its services. This is a very common scenario in cloud computing, so rental cost minimization ends up being also a relevant performance indicator.

– Objective Function:

$$\text{Maximize} \sum_{\{x \in \mathcal{X}\}} \omega_x - \frac{\sum_{\{n \in \mathcal{N}\}} \Theta^n}{|\mathcal{N}|} \tag{3.9}$$

This goal ensures that the largest number of `Rules` is placed/distributed using the lowest number of nodes, for cost minimization.

– Placement of Rules: Expression (3.2).

– Computational Resource Limitation: Expression (3.3).

– Obtaining nodes in use:

$$\Theta^n \geq \beta_x^n, \forall n \in \mathcal{N}, \forall x \in \mathcal{X} \tag{3.10}$$

$$\sum_{x \in \mathcal{X}} d_{x,m} \times \beta_x^n \leq c_{n,m}, \forall n \in \mathcal{N}, \forall m \in \{1, ..., M\} \tag{3.11}$$

Constraints (3.11) avoid exceeding the available resources at nodes.

– Non-negativity assignment to variables:

$$\omega_x, \beta_x^n, \Theta^n \in \{0, 1\}. \tag{3.12}$$

The above-mentioned optimization problems provide the optimal solution for each problem instance. However, solvers (e.g., CPLEX [131], Gurobi [132]) that rely on these formalizations to compute optimal solutions can only find results, in a reasonable amount of time, if the problem instance is very small. For this reason an RL-based solution is proposed. RL methods have the ability to understand the system dynamics and, after trained, make appropriate placement decisions in real-time.

3.4 Proposed DRL Framework

Prior to presenting the proposed DRL solution designed to tackle the RD problem, formalized in Section 3.3, this section first presents a brief RL and DNN background, which is then followed by the MDP formulation of RD problem.

3.4.1 Deep Reinforcement Learning Background

Reinforcement Learning

In RL, sequential decision problems can be modeled using Markov decision processes (MDPs). An MDP can be represented by the tuple $(\mathcal{S}, \mathcal{A}, p, r)$, where $\mathcal{S}$ is the state space and $\mathcal{A}$ is the action space. The $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the probabilistic transition model (matrix), where $p(s'|s, a)$ is the probability of arriving at state s' from s after taking action a, of which $s', s \in \mathcal{S}$ and $a \in \mathcal{A}$; $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is the reward function.

An agent learns by interacting with the environment. At time step t, the agent observes a state s_t and takes an action a_t. The environment returns a reward signal $r(s_t, a_t)$ to the agent and makes a transition to a next state s_{t+1}. The objective of the agent is to maximize the collected reward (outcome):

$$G_t = \left[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, a_{t+k}) | s_0 = s_t \right] \tag{3.13}$$

where γ is a discount factor used to balance the importance of immediate and long-term rewards.

For the agent to decide which action to take at a given state, it needs to follow a policy. Therefore, the objective of training an agent is to find a policy π that maximizes G_t. A policy is optimal π^* if following it produces the maximum outcome. In addition, a policy π can be deterministic i.e., $\pi(s) = a$, or stochastic i.e., a probability distribution over the action space $\mathcal{A}$.

There are two ways to learn π: value-based and policy-gradient. This work focuses only on the latter.

In value-based learning, π is calculated from a state-value function $V_\pi(s)$ which measures how good it is to be at a given state s:

$$V_\pi(s) = \mathbb{E}\left[G_t | s_t = s\right] \tag{3.14}$$

In policy gradient learning, π can be modeled by a parametric function where the objective is to learn a set of parameters θ that maximizes an objective function J. Assume an episodic setting with episode length T and let τ be a sequence of transitions (trajectory) $\tau = s_0, a_0, \ldots, s_{T-1}, a_{T-1}$. Following the probability general product rule, π can be defined as:

$$\pi(\tau; \theta) = p(s_0) \prod_{t=0}^{T-1} \pi(a_t | s_t; \theta) p(s_{t+1} | s_t, a_t) \tag{3.15}$$

where p is, again, the transition probability density function and θ is the set of parameters.

From Expression (3.15), it is possible to see that the transition from s to s_{t+1} depends on the action taken, a_t, and the probability of reaching s_{t+1} from s_t when taking a_t. Furthermore, following the original assumption about π^*: π^* *will generate the optimal trajectory* τ^* *that collects the maximum accumulated reward*, the objective function J can be formulated as:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[r(\tau)] = \int \pi(\tau; \theta) r(\tau), \tag{3.16}$$

and the optimal set of parameters can be found using:

$$\theta^* = \arg\max J(\theta) \tag{3.17}$$

To find θ, the product in Expression (3.15) has to be simplified and differentiated with respect to θ. Both can be done using the logarithmic differentiation:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\nabla \log(\pi(\tau; \theta)) r(\tau)\right] \tag{3.18}$$

Moreover, since r does not depend on θ, we have:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1} \nabla \log(\pi(a_t|s_t;\theta))G_t\right] \tag{3.19}$$

Following the gradient ascent rule, the update step is:

$$\theta_{k+1} = \theta_k + \alpha\nabla_\theta J(\theta_k) \tag{3.20}$$

where α is the learning rate.

The steps of: $i)$ sampling a trajectory $\tau \sim \pi_\theta$; $ii)$ differentiating with respect to θ (Expression (3.19)); $iii)$ updating θ (Expression (3.20)); are known as the REINFORCE algorithm [133, 134]. This method, however, can suffer from high gradient variance due to trajectory sampling. A common way to reduce the variance is by subtracting a baseline value from the reward term in Expression (3.18). When the baseline is used, Expression (3.19) can be rewritten as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1} \nabla \log(\pi(a_t|s_t;\theta))A(s_t,a_t)\right], \tag{3.21}$$

where $A(s_t, a_t)$ is known as the advantage function [135]:

$$A(s_t,a_t) \approx r(s_t,a_t) + \gamma V_{\pi_\theta}(s_{t+1}) - V_{\pi_\theta}(s_t) \tag{3.22}$$

and $V_{\pi_\theta}(s_t)$ is the actual baseline value (Expression (3.14)).

The method using the Expression (3.21) is called advantage actor critic (A2C) [136]. As the name suggests, this algorithm relies on an actor to learn the π_θ and on the critic to learn the baseline values. An efficient way to model both the actor and the critic is by using DNNs. In this case, their losses can be defined as follows:

$$\text{actor loss: } -\sum_{t=0}^{T-1} \log(\pi(a_t|s_t;\theta))A(s_t,a_t) \tag{3.23}$$

$$\text{entropy: } -\sum_{t=0}^{T-1} \pi(a_t|s_t)\log(\pi(a_t|s_t)) \tag{3.24}$$

where the *entropy* term is used to encourage exploration by penalizing agents' overconfidence. For the critic loss, the following expression is used:

$$\text{critic loss: } \frac{1}{2}\sum_{t=0}^{T-1} A(s_t, a_t)^2 \tag{3.25}$$

Deep Neural Networks

As noted in Section 3.2, the models and architectures that are currently used by the RL community to tackle combinatorial problems are well-suited for handling problems with variable state and action spaces. Therefore, this work relies on the Ptr-Net and the Transformer, which are robust and flexible architectures. For brevity purposes, and due to space limitation, most of the mathematical foundations of these architectures are omitted and a quick overview is presented next. For further details please refer to [122] and [130].

Pointer Network Architecture The main idea of Ptr-Net is that the output sequence is obtained by *pointing* at elements in the input, hence the name of the architecture. In other words, the output is a sequence of input selections, which makes Ptr-Net suitable for modeling selection problems. For example, this architecture was used to solve classical problems like the TSP and knapsack problem [122, 137]. In both cases, the solution is built by selecting the sequence by which cities are visited (in case of TSP) or sequence by which items will be placed into the backpack (in case of knapsack).

This architecture has two main components: the *encoder* and the *decoder*. The encoder's role is to map the input into a feature space, while the decoder generates the pointers. In the encoding mode, the encoder reads the input data, one element at a time, and generates a set of fixed-dimensional vector representations, known as encoder hidden states $(e_1, ..., e_n)$ where n is the number of input elements. After reaching the end of the input sequence, Ptr-Net switches into a decoding mode. In this mode, the decoder is initialized by the encoder's last hidden state e_n and a user-defined start of sequence (SoS) symbol. The decoder then produces the first hidden state d_1, which is used to generate the first pointer (i.e., index) to an element in the input sequence. In the next decoding step, d_1 and the previously pointed element are fed to the decoder to generate a subsequent pointer. This process repeats itself until the decoder points to a user-defined end of sequence (EoS) symbol or a predefined number of pointers is reached. The decoder's hidden states are represented by $(d_1, ..., d_m)$ where m is the size of the output sequence.

The actual pointing mechanism, which we call Ptr-Net head, is computed as follows. First, at any decoding step i the unnormalized log probabilities (logits) u_i are calculated:

$$u_i^j = v^T \tanh(W_1 e_j + W_2 d_i), j \in (1, ..., n) \tag{3.26}$$

where v^T, W_1, W_2 are learnable parameters of the model.

Next, u_i is fed to a `softmax` layer to calculate the probabilities of pointers selection:

$$p_i = \texttt{softmax}(u_i) \tag{3.27}$$

where p_i is the softmax distribution with dictionary size equal to the length of the input. Finally, a pointer is selected by sampling from p_i via a sampling strategy (e.g., greedy, stochastic). In greedy sampling, for example, the selected pointer is the index of the element with the highest probability in p_i.

It is worth to note that in the original Ptr-Net proposal [122] both the encoder and the decoder use LSTMs. However, the authors of Ptr-Net have shown later (same year) that RNNs, including LSTM and GRU, are not suited for solving problems where the order of the input does not matter [138]. The main limitation of RNNs is that they are order-sensible, an undesired behavior when working with sets of data. Moreover, these networks are ineffective in learning long distance relationships between the elements in the input, i.e., finding dependencies between elements that are located far apart in the input sequence [139]. Since RNNs pass the information sequentially, the longer the input sequence, the more likely it is that some information will be lost during sequence processing.

Transformer Architecture This architecture, introduced in 2017, presents a new way of computing relationships between the elements in the input. It completely abandons the use of RNNs and, instead, relies entirely on the attention mechanisms [130]. Dropping the recurrence removed the constrains of sequential computation, allowing larger models to be built and a faster training to be achieved. Current state-of-the-art NLP solutions (e.g., GPT-3 [3]) use the Transformer or one of its variants [140].

The core idea introduced by the Transformer is the self-attention mechanism that allows to find inter-dependencies between elements in the input, regardless of their positions in the input sequence. Broadly speaking, self-attention receives as input an embedding representation of n input elements, denoted by $(x_1, ..., x_n)$, and maps them into $(z_1, ..., z_n)$ output embeddings while preserving the input dimensions. Each z_i embedding, $i \in (1, ..., n)$, contains information about how input element at position i is related to all the remaining input elements. This input-output transformation process can be summarized as follows. First, every input element x_i, $i \in (1, ..., n)$, is mapped via linear transformation into key k_i, query q_i and value v_i vectors. Next, a logit u_i^j measuring the compatibility between q_i and key k_j, $\forall j \in (1, ..., n)$, is computed:

$$u_i^j = \frac{q_i k_j^T}{\sqrt{d^{(k)}}}, j \in (1, ..., n) \tag{3.28}$$

where $d^{(k)}$ is the dimension of k.

Then, just as in Ptr-Net, the compatibility vector u_i is passed through a `softmax` layer (Expression (3.27)) to produce a probability vector p_i, whose size is equal to the length of the input. Finally, the output z_i is computed as:

$$z_i = \sum_{j=1}^{n} p_i^j v_j, \tag{3.29}$$

where p_i^j is the jth weight in p_i.

The generic input-output mapping, provided by the self-attention mechanism, allow these computation blocks to be stacked to create a higher level of abstraction and, potentially, give the network more generalization capacity. In the original work, authors stacked six self-attention blocks in the encoder and decoder and, at that time, were able to achieve state-of-the-art results in NLP tasks.

Please note that the Transformer was originally developed to solve NLP problems, where the order of the input sequence matters. To that end, the authors introduced a so called *positional encoding*[1]. Here in our work the order does not matter and, for this reason, the positional encoding is not included in our design.

3.4.2 MDP Formulation

In the following sections the RD problem, outlined in Definition 1, is formulated as an MDP and described in terms of state, actions, and possible reward functions.

State Space

The state of the RD problem includes: i) a set of `Rules` that arrive at the reverse proxy and need to be distributed across nodes, denoted by $\mathcal{X}' \subseteq \mathcal{X}$; ii) a set of available nodes capable of hosting `Rules` in $\mathcal{X}'$, denoted by $\mathcal{N}' \subseteq \mathcal{N}$. Each `Rule` is represented by its profile that includes CPU, RAM and storage requirements, while each node is represented by its CPU, RAM and storage resources. Note that the node set, i.e., locations where `Rules` can be placed, is also part of the action space, which will be explained in the following section.

For an adequate distribution of `Rules`, reverse proxy must have an up-to-date information about the number of available nodes and their states. In distributed systems, nodes usually share this kind of information either by using passive or active methods. In case of passive methods, nodes expose a specific endpoint, usually `/health`, that responds with its status: an `HTTP 200 OK` status code informs that the device is alive, which may be followed by information on available resources. Then, the reverse proxy (e.g, Traefik)

[1] See "Section 3.5 - Positional Encoding" in [130]

makes periodic requests to these endpoints and uses the obtained information during the distribution process. In case of active methods, as in MoleculerJS [99], nodes periodically broadcast `Heartbeat` packets containing information about their health status [141]. Here we assume that a reverse proxy always has an up-to-date information on the available nodes and their CPU, RAM and storage resources. Note that there might be situations where nodes do not have enough resources to serve all requests and, therefore, some of the requests will be rejected. For this reason, we introduce a dummy node $node_{\text{rej}}$ where rejected `Rules` are placed. This node does not have the ability to process the `Rules`, as it does not have any computational resources, and its purpose is to allow the RL agent to associate rejected `Rules` with a dedicated place. The overall state for the RD problem is the following:

$$\mathcal{S} \triangleq \{\mathcal{N}' \cup node_{\text{rej}}, \mathcal{X}'\}. \tag{3.30}$$

Designing $\mathcal{S}$ this way allows the RL agent to have a global view of the problem, in order to make appropriate placement decisions. Furthermore, this design can be applied to model other optimization problems (e.g, multiple knapsack problem, multiple vehicle routing problem) whose state is described by two sets and the problem solution can be modeled as a sequence of assignment decisions.

Action Space

It is assumed that a `Rule` $x \in \mathcal{X}'$ can be placed at any available node $n \in \mathcal{N}'$, as long as resources are not exceeded. If a `Rule` has requirements higher than the resources available at the nodes in $\mathcal{N}'$ then it is rejected, i.e. placed at the $node_{\text{rej}}$. Therefore, the action space is represented as follows:

$$\mathcal{A} = \mathcal{N}' \cup node_{\text{rej}} \tag{3.31}$$

Placing a `Rule` at node n involves updating the amount of available resources at n. This is done by subtracting `Rule` resource requirements from node's available resources. Placing a `Rule` at $node_{\text{rej}}$ does not involve any computation, as the `Rule` is being rejected.

From the user point-of-view, placing a `Rule` at any node in $\mathcal{N}'$ would result in a `HTTP 201 Created` meaning that the request was accepted and the `Rule` is running. On the other hand, placing a `Rule` at the $node_{\text{rej}}$ would result in a `HTTP 503 Service Unavailable` response, meaning that the system is overloaded at the moment, and can not process the incoming request. Therefore, the state and action spaces design is

consistent with the realistic settings of `Rule` placement.

Rewards

The agent places `Rules`, one at a time, and in return receives a reward value. This reward depends on the RD problem variant under consideration (see Section 3.3).

Greedy Reward The reward signal for greedy optimization is simple and represented as follows:

$$r(s_t, a_t) = \begin{cases} 0, & \text{if } \texttt{Rule} \text{ is rejected} \\ 1, & \text{if } \texttt{Rule} \text{ is accepted} \end{cases} \tag{3.32}$$

Placing a `Rule` at any node in $\mathcal{N}'$ gives a positive reward to the agent, while placing it at $node_{\text{rej}}$, i.e., rejecting the `Rule`, gives zero reward. Since the agent aims to maximize the accumulated reward, it will try to come up with a distribution strategy that maximizes the number of placed `Rules`.

Critical-Aware Greedy Reward To maximize the placement of the `Rules`, while ensuring a fair distribution, the following reward is used:

$$r(s_t, a_t) = \begin{cases} -2, & \text{if } \texttt{Rule} \text{ is rejected} \\ \Omega^{\text{MAX}}, & \text{if } \texttt{Rule} \text{ is accepted} \end{cases} \tag{3.33}$$

For each rejected `Rule` the agent is penalized by receiving a negative reward (-2, which has been selected empirically). On the other hand, placing a `Rule` at node $n \in \mathcal{N}'$ gives a positive reward equal to the most critical resource among all nodes in $\mathcal{N}'$, as described in Expression (3.7).

Penalizing rejections with negative rewards and rewarding successful `Rule` placement with positive rewards has two effects. It encourages the placement of the largest number of `Rules` and, at the same time, it allows the agent to obtain high Ω^{MAX} rewards, which brings fairness to the distribution of `Rules`.

Cost-Aware Greedy Reward The reward for maximizing the `Rule` placement while minimizing the number of used nodes is represented as follows:

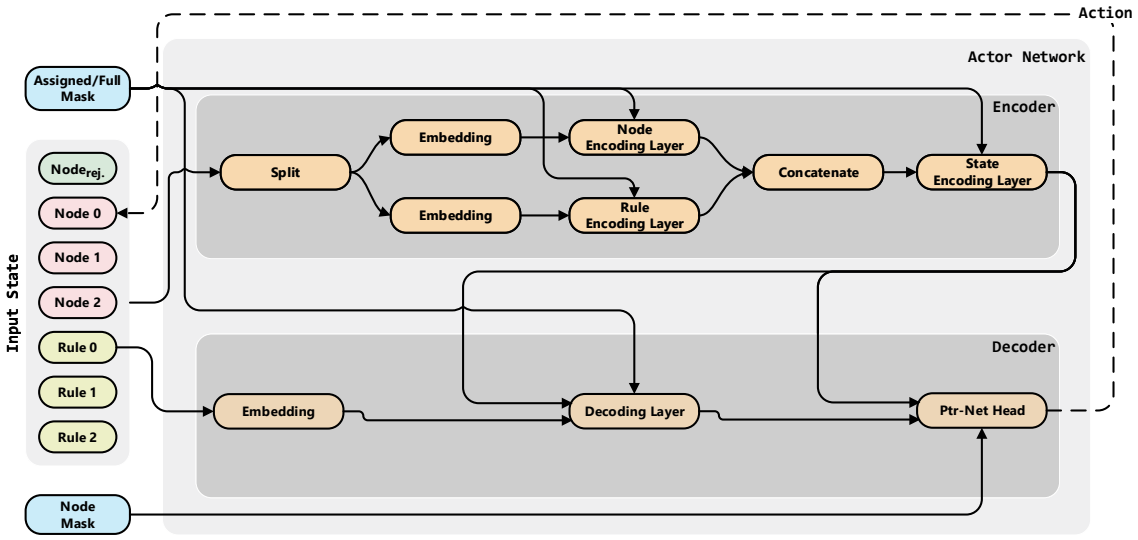


Figure 3.3: Model architecture and decoding step example: `Rule 0` placed at node 0.

$$r(s_t, a_t) = \begin{cases} -2, & \text{if } \texttt{Rule} \text{ is rejected} \\ -1, & \text{if } \texttt{Rule} \text{ placed at an empty node} \\ 0, & \text{otherwise} \end{cases} \tag{3.34}$$

The minimization of rejected requests is ensured by giving a negative reward, -2, for each rejected `Rule`. To enforce that a minimum number of nodes is used, the agent receives a negative reward, -1, every time it places a `Rule` at an empty node, i.e., a node without previously assigned `Rules`. Placing at a node that already has `Rules` assigned to it results in a reward equal to zero. This way the agent will try to find a strategy that uses the lowest number of nodes to place all the `Rules`.

3.4.3 Model Architecture

Figure 3.3 is a graphical representation of the proposed model, which has two main components (encoder and decoder) that are detailed in this section. The source code of the model, implemented in Tensorflow [142], is also publicly available[2].

Encoder

The proposed encoder, illustrated in the upper part of Figure 3.3, uses self-attention mechanisms in all of its encoding layers. It receives $\mathbf{y} = (y_1, y_2, ..., y_{n-m}, y_{n-m+1}, ..., y_n)$ as input, where y_1 is $node_{\text{rej}}$; y_2 to y_{n-m} are the available nodes; and y_{n-m+1} to y_n are the `Rules`.

[2] `https://github.com/AndreMaz/transformer-pointer-critic`

In the proposed architecture, the input **y** is split into two sets: `Rules` and nodes. Each set passes through a dedicated `embedding` layer and a self-attention block. The decision of having two separate processing components, one for each set, has two reasons. First, to give the model the ability to select different pre-processing strategies for each type of input. A pre-processing can be, for instance, sorting the rules and nodes using different strategies (as in traditional heuristic algorithms). However, in this case, the pre-processing strategies are not based on any human preconception, and the model learns adequate strategies that suit the QoS factor at hand. The second reason is to make the model flexible to input data of uneven number of features. In other words, to make the model generic and applicable to different problems. For example, for the cost-aware greedy optimization the node requires an additional binary feature in order to represent whether it is empty or not, meaning that the node is described by four features while the `Rule` is described by three only. Furthermore, the splitting idea can also be applied to more than two input components, which adds more flexibility to the proposed design.

After completing the node and `Rule` pre-processing, the output is concatenated and passed into a final self-attention block where the network performs further processing. This time it looks for the relationships between the nodes and the `Rules`.

An interesting property of the proposed encoder is that it generalizes much better than the typical Transformer encoder, especially when the number of `Rules` or nodes increases. This is mainly due to the flexibility of processing each input type separately using a dedicated location in the encoder. Finally, to improve the training process and enforce problem constraints, the `Rules` already in place and full nodes are masked at each step. Masking allows the encoder to discard specific elements during the processing stage. The masking is done by modifying the logits calculation in Expression (3.28) by:

$$u_i^j = \begin{cases} \frac{q_i k_j^T}{\sqrt{d^{(k)}}}, & j \in (1, ..., n-m), j \neq 1, j \neq \text{empty node} \\ -\infty, & \text{otherwise} \end{cases} \tag{3.35}$$

The rule and the state encoding blocks use exactly the same expression but operate over different indices. In the case of rule encoding block the `Rules` that are already assigned are masked.

Decoder

The proposed decoder is illustrated in the lower part of Figure 3.3. At each step i, it receives encoder's output and a single `Rule` from range $(y_{n-m+1}, ..., y_n)$ and generates a pointer to a node $(y_1, ..., y_{n-m})$.

In the decoding layer, the model looks for a (good) contender node for a given `Rule`. Since the decoder has access to encoder's output, where the model had a global view of

the problem, the selection of the contender takes also into consideration the presence of other `Rules` and the state of the nodes. Finally, the last decoder layer is a Ptr-Net head, described in Section 3.4.1. This head computes the location where the given `Rule` will be placed. In order to avoid selecting unfeasible positions (e.g., full nodes), the logits computation of the Ptr-Net head in Expression (3.26) is modified as:

$$u_i^j = \begin{cases} v^T \tanh(W_1 e_j + W_2 d_i), & j \in (1, ..., n-m, ..., n), \\ & j \leq n-m, j \neq 1, \\ & j \neq \text{empty node} \\ -\infty, & \text{otherwise} \end{cases} \tag{3.36}$$

Furthermore, as in [106], the logits from Expression (3.36) are clipped to $[-C_{\text{logit}}, C_{\text{logit}}]$, where $C_{\text{logit}} = 10$ in this work, to encourage the exploration:

$$u_i^{(\text{clip})} = C_{\text{logit}} \cdot \tanh(u_i) \tag{3.37}$$

Then, the clipped logits $u_i^{(\text{clip})}$ passes through a `softmax` layer, Expression (3.27), to produce the probability vector from which the node is selected. In the following decoding step, the resources of the selected node are updated, the assigned `Rule` is masked and the process repeats for the next `Rule` in $(y_{n-m+1}, ..., y_n)$.

It is important to outline that, in addition to sequential `Rule` feeding, we have tested random and neural-network based feeding strategies. However, other than adding additional complexity to the model we noticed no improvements in using any of these strategies. Since the model has a global view of the problem, via the encoder's output, the order by which the `Rules` are placed has no effect in the final result.

3.5 Performance Evaluation

3.5.1 Experimental Setup

To evaluate the performance of the proposed agent, a comparison will be made against several greedy heuristics and the optimal solution, generated using CPLEX solver. The optimal relies on the mathematical formalizations presented in Section 3.3. The main goals are: i) to assess the performance of the agent for different objective function criteria, and compare against baseline methods; ii) to evaluate the scalability and generalization capacity of the agent, i.e., evaluate the performance of the agent when it has to solve problems larger than the ones used during training. It worth to mention that performance evaluation does not consider other DRL approaches because, as mentioned in Section 3.2,

models used to solve task distribution and load balancing problems do not have the ability to scale, i.e., models have to be retrained each time the state or action space changes. Therefore, to validate the proposed method, optimal solutions are used as a baseline.

All the development and testing was done on a PC with i7 6700 3.4Ghz CPU with 32GB of RAM and a single Nvidia 2080Ti with 11GB GPU.

The simulation environment was configured to generate instances on-the-fly. A uniform distribution, in range $[0, 1]$, is used to assign CPU, RAM, and storage resources to all nodes in an instance. A pool of 1000 `Rules` is made available, for set sampling purposes, and resource requirement values are assigned using a uniform distribution in range $[0.01, 0.30]$. This range ensures that the placement of `Rules` is neither too easy nor artificially hard, while providing some degree of freedom when distributing `Rules`. Additionally, the environment was configured to produce problem instances of different sizes using the following parameters, allowing the assessment of the agent's performance and scalability:

- *Node-related parameters:*
  - Smallest number of nodes: 10
  - Largest number of nodes: 50
  - Step size: 10
- *Rule-related parameters:*
  - Smallest number of `Rules`: 10
  - Largest number of `Rules`: 100
  - Step size: 10

The training process of the agent and baseline heuristics are discussed next, prior to the analysis of results. It should be noted that the choice for relatively simple greedy heuristics, rather than complex heuristics or meta-heuristics, is due to the fact that it is not feasible to run complex approaches in real-time on reverse proxies. As mentioned in Section 3.1.1, the load balancing methods used in actual implementations are often simple strategies. All strategies, RL agent and baseline heuristics, build a solution in a single pass, by starting from an empty solution and expanding it by placing one `Rule` at a time.

3.5.2 Training and Hyperparameters

The agent was trained on a fixed problem size of 10 nodes and 20 `Rules`, an empirically selected size that showed to be a good compromise between the training time, approximately 14 hours, and the agent's ability to generate robust solutions to all the problem instances during the testing.

The agent is trained using an A2C algorithm [136], summarized in Algorithm 1, with mini-batches of size equal to 128, for better gradient estimates. Two dedicated networks are used: i) the actor for learning the policy, denoted by π_θ; ii) the critic to estimate the baseline, denoted by V_ϕ. The actor network is illustrated in Figure 3.3 and described in detail in Section 3.4.3. The critic network is the encoder, from Figure 3.3, followed by three fully connected layers with linear activation. The last layer of the critic network produces a single scalar value that represents the baseline estimation. During the testing phase, only the actor network is used.

The choice for two separate networks, with no parameter sharing, was made in order to have more freedom to control agent's inference time and the quality of critic's baseline estimations. The actor's encoder uses a single self-attention stack $N_{\text{actor}}^{\text{stack}} = 1$ in each encoding block, while the critic uses a stack of three $N_{\text{critic}}^{\text{stack}} = 3$. Critic with less than three self-attention stacks produced poor baseline estimations. Different learning rates were also used for the actor $\alpha_{\text{actor}} = 1e^{-4}$ and the critic $\alpha_{\text{critic}} = 5e^{-4}$ networks, parameters that have shown to be stable and to ensure convergence. Table 3.1 summarizes all the parameters that were adopted in this work. Please note that the batch size and number of layer stacks in critic network were selected as a trade-off between the quality of the solution and the GPU memory constraints.

Algorithm 1: Actor-Critic training algorithm (adapted from [106]).

Input : T=training steps, B=batch size
Output: Trained actor network π_θ

Initialize actor π_θ network parameters;
Initialize critic V_ϕ network parameters;
for $t = 1$ **to** T **do**
  $s_i \leftarrow$ `SampleProblem()`, $\forall i \in \{1, ..., B\}$
  $a_i, r_i \leftarrow$ `SolveProblem`(s_i, π_θ), $\forall i \in \{1, ..., B\}$
  $b_i \leftarrow$ `EstimateBaseline`(s_i, V_ϕ), $\forall i \in \{1, ..., B\}$
  $A \leftarrow$ `ComputeAdvantage`(r_i, b_i) // Expression (3.22)
  $E \leftarrow$ `ComputeEntropy`(π_θ) // Expression (3.24)
  $g_\theta \leftarrow \frac{1}{B} \sum_{i=1}^{B} A \nabla_\theta \log(\pi_\theta(a_i|s_i)) + c_{entropy} \times E$
  $L_\phi \leftarrow \frac{1}{B} \sum_{i=1}^{B} ||A||_2^2$
  $\theta \leftarrow ADAM(\theta, g_\theta)$
  $\phi \leftarrow ADAM(\phi, \nabla_\phi L_\phi)$
end

3.5.3 Baseline Heuristics

Random Insertion

This heuristic randomly distributes the set of `Rules` across a set of available nodes. More specifically, a `Rule` and a node are randomly picked from corresponding sets and then the

Table 3.1: Network(s) and Training Parameters.

General	
Training Steps	100 000
Batch Size	128
Discount Factor (γ)	0.99
Entropy Coefficient (c_{entropy})	0.01
Actor Network	
Weight Initialization	Xavier uniform [143]
Number of Layer Stack ($N^{\text{stack}}_{\text{actor}}$)	1
Embedding Size	128
Number of Heads	8
Inner Layer Dimension	128
Logit Clipping (C_{logit})	10
Gradient Clipping	L2 Norm (1.0)
Optimizer	Adam [144]
Learning Rate (α_{actor})	$1e^{-4}$
Critic Network	
Weight Initialization	Xavier uniform [143]
Number of Layer Stacks ($N^{\text{stack}}_{\text{critic}}$)	3
Embedding Size	128
Number of Heads	8
Inner Layer Dimension	512
Gradient Clipping	L2 Norm (1.0)
Last Layers Dimensions	128
Optimizer	Adam [144]
Learning Rate (α_{critic})	$5e^{-4}$

feasibility of the pair is checked, i.e., a validation is performed to check if the resources available at the node are enough to satisfy the requirements of the `Rule`. If this condition is not met, another node is picked. This process is repeated until all `Rules` are in place or marked as rejected.

Critical Resource Insertion

This is a two step heuristic, summarized in Algorithm 2, that first sorts the `Rules` by their resource requirements and then distributes them across the nodes, possibly giving priority to fairness (Ω^n). The following four variations were generated:

- Descending Rules - Descending Critical (DR-DC)
- Descending Rules - Ascending Critical (DR-AC)
- Ascending Rules - Descending Critical (AR-DC)
- Ascending Rules - Ascending Critical (AR-AC)

DR-DC and AR-DC (C_{order}='desc') prioritize the placement of `Rules` at locations that maximize the availability at the most critical resource, ensuring fairness. DR-AC and AR-AC (C_{order} ='asc'), on the other hand, prioritize placement of `Rules` that minimizes the Ω^n, which potentially leads to fewer nodes in use. The X_{order} variable controls whether smaller or larger `Rules` are placed first.

3.5.4 Evaluation

Evaluation consists in solving three different RD variants, mathematically expressed in Section 3.3.3, and observe how the agent performs in the key performance indicators (KPIs) of each. To ensure the consistency of the agent performance, three different seeds were used to initialize the agent training weights, which essentially results in three distinct versions of the same agent. After the training is finished, each agent solves 100 problem instances for each scenario under consideration. Therefore, the reported performance is an average of the three agents, i.e., an average of 300 executions. The baseline heuristics, which are deterministic methods, also solve 300 problem instances. Finally, to ensure that results of the CPLEX solver are obtained within a feasible time, a time limit of 60 seconds is set. With time limit set, CPLEX solver will either return the optimal solution, if it manages to find one, or the best solution found so far. For this configuration CPLEX takes approximately 8 days to solve all the problem instances of each RD variant under the consideration.

Algorithm 2: Critical resource insertion heuristic.

Input : $\mathcal{N}'$=node set, $\mathcal{X}'$=rule set, $X_{\text{order}} \in \{$'asc','desc'$\}$ is the rule size sorting criteria, $C_{\text{order}} \in \{$'asc','desc'$\}$ is the critical resource sorting criteria.
Output: Feasible solution

$\mathcal{X}' =$ `SortByLargestResource` $(\mathcal{X}', X_{\text{order}})$
foreach x *in* $\mathcal{X}'$ **do**
    Initialize critical resource list L_Ω
    foreach n *in* $\mathcal{N}'$ **do**
        // Determine critical resource
        $\Omega^n =$ `ComputeCritical` (n, x)
        $L_\Omega \leftarrow (\Omega^n, n)$
    end
    // Sort L_Ω by Ω^n
    $L_\Omega =$ `Sort` $(L_\Omega, C_{\text{order}})$
    // Do the first fit
    foreach Ω^n, n *in* L_Ω **do**
        $allocated$ = False
        if $\Omega^n \geq 0$ **then**
            // Place `Rule` x at node n
            $allocated$ = True
            $break$ // Break the loop
    end
    if *allocated* == $False$ **then**
        Reject x
end

Greedy Results

The KPI for the greedy optimizer (see Section 3.3.3) is the `Rule` rejection rate, i.e., the likelihood of a user's request being rejected. Figure 3.4 shows the performance of the different solving methods for this particular case. Looking at the plots, it is visible that CPLEX solver has the lowest rejection rate in the all scenarios and, therefore, has the best performance. Moreover, in the green highlighted areas CPLEX was able to find optimal solutions within the time limit.

The second best performing method is the proposed agent. It is visible that regardless of the problem size, the proposed agent has the smallest rejection gap with respect to the CPLEX solver, which acts as a lower bound. As an example, for problems with 10 nodes the proposed agent has an average rejection rate 4.69% higher than the CPLEX, as shown in Table 3.2. As the number of available nodes increases the rejection gap becomes smaller, reaching an average of 0.01% for problems with 50 nodes. In contrast, the best performing heuristic (AR-DC) has an average rejection gap of 9.26% for 10 nodes, when compared to CPLEX. The DR-DC presents the lowest gap, 0.52%, for 50 nodes, which is still larger than the agent's gap.

Results indicate that agent's generalization capacity is maintained even for large pro-

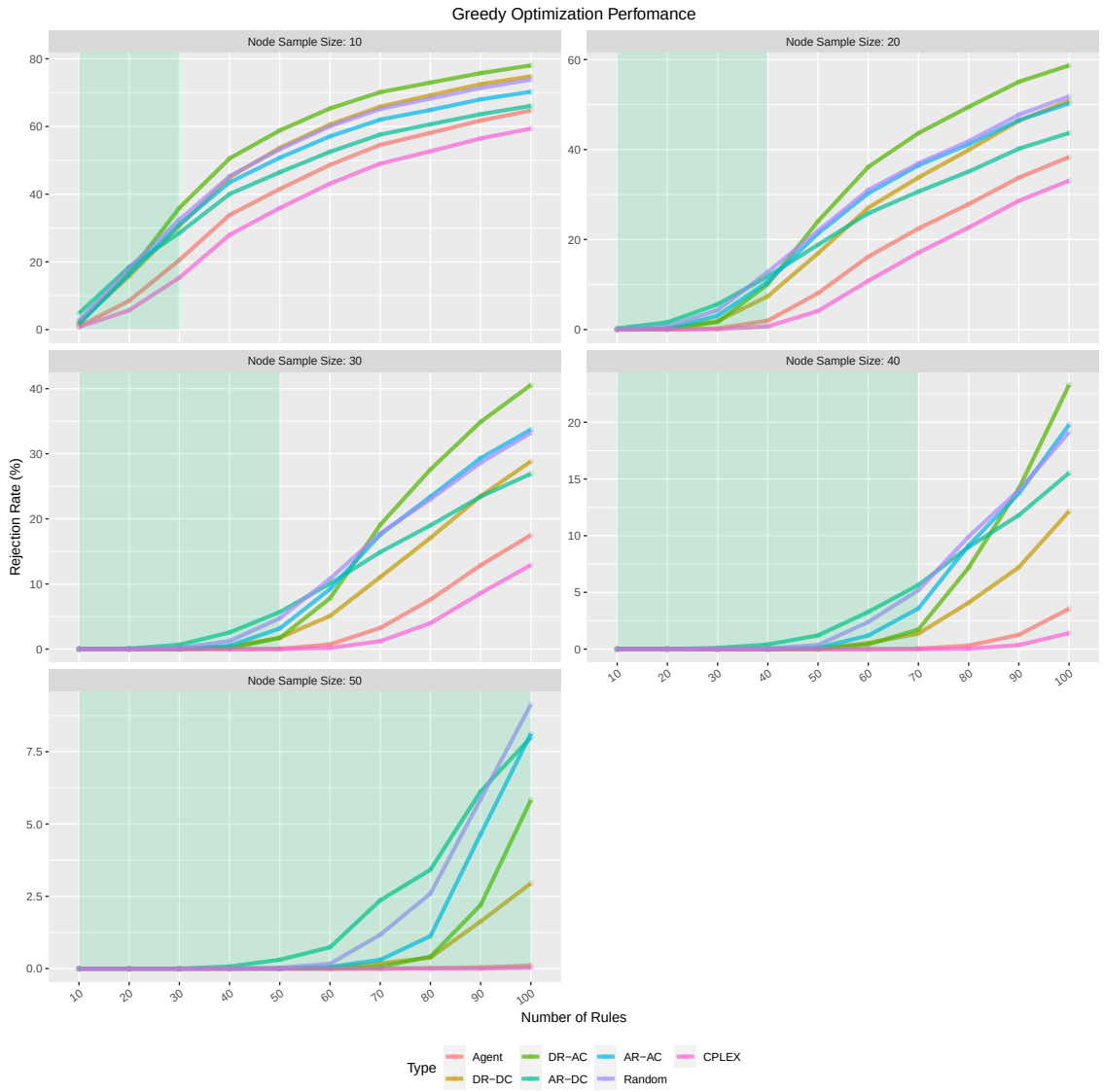


Figure 3.4: `Rule` rejection rate for different problem sizes. Green areas highlight where CPLEX was able to find optimal results for all instances, i.e., all the solution space was searched.

blems. Note that during training the agent solved problems with only 10 nodes and 20 `Rules`. However, results do not show any critical degradation in performance, even for problem instances five times larger than the ones used in training. For example, for the problems with 50 nodes, where CPLEX solves all problem instances to optimality, the agent generates near-optimal solutions.

Critical-Aware Greedy Results

In this scenario, Section 3.3.3, the `Rule` rejection rate and the most critical resource (Ω^{MAX}) are the KPIs, which are shown in Figure 3.5.

As in the case of greedy optimization, the agent shows a similar trend when considering the rejection rate. On the other hand, looking at the most critical resource at different

Table 3.2: Average rejection rate gap with respect to the CPLEX solver for the greedy optimization.

Node Sample Size	Agent	DR-DC	DR-AC	AR-DC	AR-AC	Random
10	4.69	14.4	17.97	9.26	11.95	14.44
20	3.18	10.73	16.18	9.64	12.28	13.2
30	1.51	6.08	10.49	7.64	9.01	9.27
40	0.34	2.36	4.5	4.52	4.58	4.93
50	0.01	0.52	0.85	2.1	1.42	1.89

Table 3.3: Average Ω^{MAX} and rejection rate gaps with respect to the CPLEX solver for the critical-aware optimization.

	Agent		DC-DC		DR-AC		AR-DC		AR-AC		Random	
Node Sample Size	Ω^{MAX}	Rej. Rate	Ω^{MAX}	Rej. Rate	Ω^{MAX}	Rej. Rate	Ω^{MAX}	Rej. Rate	Ω^{MAX}	Rej. Rate	Ω^{MAX}	Rej. Rate
10	0.00210	4.75	0.00175	14.14	0.00553	17.73	0.00093	9.18	0.00544	11.91	0.00453	14.38
20	0.00173	3.21	0.00155	10.77	0.00498	16.04	0.00164	9.62	0.00496	12.18	0.00417	12.99
30	0.00157	1.55	0.00137	6.01	0.00483	10.51	0.00172	7.57	0.00483	8.89	0.00413	9.12
40	0.00080	0.36	0.00076	2.33	0.00366	4.39	0.00118	4.50	0.00366	4.57	0.00312	4.92
50	0.00020	0.02	0.00015	0.50	0.00252	0.85	0.00059	2.06	0.00252	1.41	0.00218	1.89

ranges, the agent struggles to keep up with the CPLEX solver and seem to stay closer the DR-DC heuristic. However, when taking both KPIs into account, it is visible that DR-DC's higher critical resource value is achieved at the expense of a higher rejection rate. Similar behavior happens with the AR-DC heuristic. Considering the KPI gaps between CPLEX and other methods, summarized in Table 3.3, it is visible that in general the overall Ω^{MAX} differences are negligible but due to lower rejection rate the proposed agent offers a smoother user experience.

Cost-Aware Greedy Results

For this RD optimization criteria, described in Section 3.3.3, the `Rule` rejection rate and the number of empty nodes are the KPIs, which are shown in the Figure 3.6.

From all the RD problems under consideration, this one was the hardest for CPLEX. This has to do with a greater number of binary variables involved in the mathematical formalization of the problem. The solver did not manage to obtain optimal solutions for problems with more than 20 `Rules` to distribute. The problem complexity might explain, therefore, the larger agent-CPLEX gap in the rejection rate, when compared with other optimization criteria Figure 3.5. These results are summarized in Figure 3.5. Nevertheless, the agent shows a trend similar to CPLEX solver in both KPIs and does not show critical performance degradation, as it happens with the heuristics.

When looking at gaps in Table 3.4, it is possible to see that for 10 nodes the agent uses in average less -0.03 nodes but has a rejection rate is 4.9% higher that the CPLEX. As the number of nodes increases the rejection rate gap decreases at the expense of additional

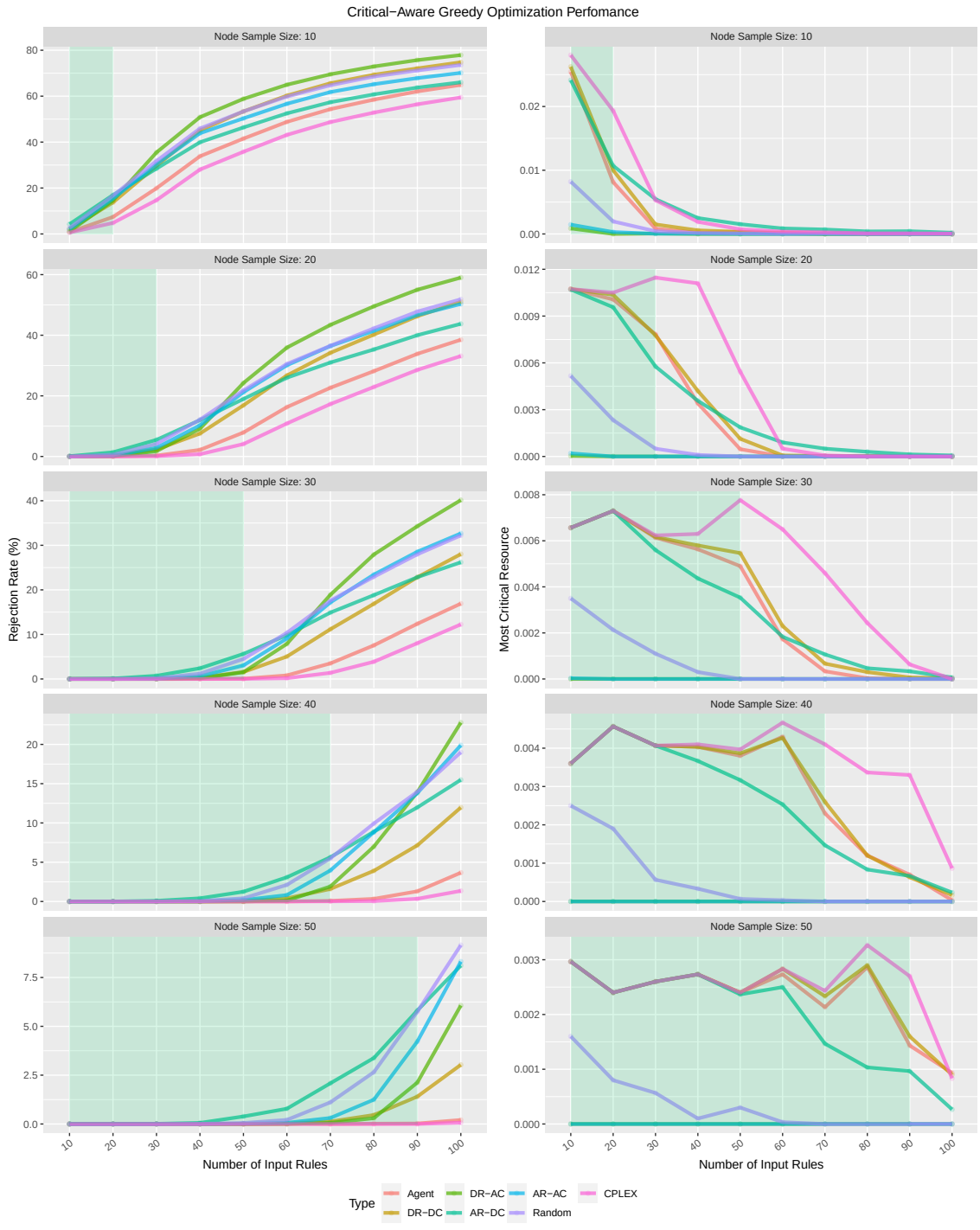


Figure 3.5: `Rule` rejection rate (left column) and most critical resource (right column) for different problem sizes. Green areas highlight where CPLEX was able to find optimal results for all instances, i.e., all the solution space was searched.

Table 3.4: Average empty node and rejection gaps with respect to the CPLEX solver for the cost-aware optimization.

	Agent		DC-DC		DR-AC		AR-DC		AR-AC		Random	
Node Sample Size	Empty Nodes	Rej. Rate	Empty Nodes	Rej. Rate	Empty Nodes	Rej. Rate	Empty Nodes	Rej. Rate	Empty Nodes	Rej. Rate	Empty Nodes	Rej. Rate
10	-0.03	4.9	0.41	14.19	0.76	17.84	-0.04	9.09	0.8	11.82	0.55	14.28
20	0.03	3.63	1.79	10.7	3.39	16.05	0.71	9.61	3.62	12.24	2.66	13.06
30	0.45	2.06	4.31	6.04	7.86	10.43	2.72	7.58	8.38	8.97	6.41	9.14
40	1.23	0.67	7.78	2.32	13.85	4.4	6.03	4.54	14.8	4.53	11.65	4.91
50	1.45	0.07	10.67	0.48	19.28	0.85	9	2.05	20.75	1.38	16.44	1.87

nodes in use. For example, for 50 nodes the agent requires more 1.45 nodes to have almost the same rejection rate as the CPLEX.

In addition, when looking at the results of the heuristics, it is visible that they struggle to keep both KPIs gaps close to the CPLEX and the proposed agent. In fact, from the three scenarios being considered, the cost-aware is where the heuristics struggle the most. This highlights that a change of a single KPI requires redesigning and developing a new heuristic, which is impractical and time consuming. In the case of the proposed agent, which performs well in all scenarios, the only change would be the reward signal.

Inference Time

Reporting the time to obtain the solutions is important but hard to measure. It depends on the hardware used (e.g., CPU vs GPU, GPU model), programming language, code implementation/optimization and so on. Nevertheless, a practical approach was taken and the reported times, summarized in Table 3.5, are averages of 1000 executions obtained on the development machine. For brevity purposes, only the results for 100 `Rules` are shown, i.e., the upper bound. Also, note that the reported measurements are rough approximations that can vary from execution to execution. Still, they show the trend and the orders of magnitude that each method takes to compute the solution. The running times of CPLEX are not reported because, as mentioned earlier, it is not feasible to run it at reverse proxies. Finally, agent's reported times are the average duration of an entire episode, which also accounts the environment's transitions.

Looking at the agent's results it is visible that its time is practically constant, regardless of the number of nodes in the input. This is a result of: i) using the Transformer architecture that, as mentioned in Section 3.4.1, does not use RNNs to compute the relationships between the input elements; ii) GPU parallelization capabilities, allowing to compute the relationships between all the elements in a single pass. It is also worth to mention that, regardless of the tested input size, the model takes approximately 2.5 milliseconds in every decoding step, i.e., to find a location for a single `Rule`.

Comparing the agent's times with the heuristics, it is visible that they operate at different orders of magnitude. For example, for 10 nodes all the heuristics are around 28

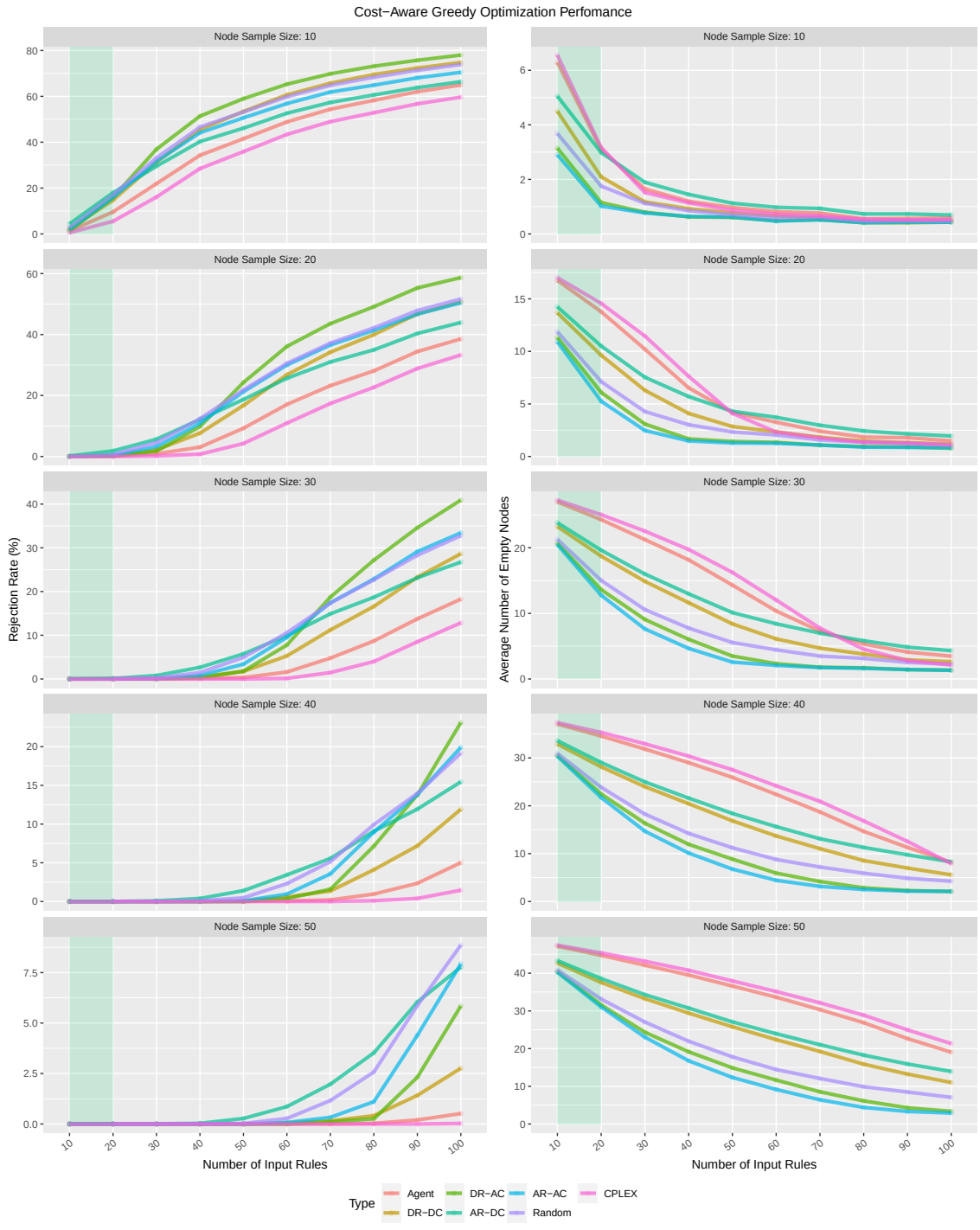


Figure 3.6: `Rule` rejection rate (left column) and empty nodes (right column) for different problem sizes. Green areas highlight where CPLEX was able to find optimal results for all instances, i.e., all the solution space was searched.

Table 3.5: Time, in milliseconds, to obtain the solution for problems with 100 `Rules`. Reported times are an average of 1000 executions.

Node Sample Size	Agent	DC-DC	DC-AC	AR-DC	AR-AC	Random
10	394.82	14.49	14.14	14.22	14.35	12.27
20	398.39	28.53	28.46	28.13	28.60	18.59
30	402.89	42.28	42.61	41.46	42.45	20.05
40	403.31	56.79	57.77	55.80	57.43	20.37
50	404.17	69.86	72.24	69.42	71.80	21.14

times faster that the agent. However, for 50 nodes, this difference drops down to 6 for the critical resource heuristics and 20 for the random heuristic. This time difference reduction is explained by the complexity of the heuristics: $O(r \cdot n\log(n))$ for the critical resource heuristic and $O(r \cdot n)$ for the random heuristics, where r is the number of `Rules` and the n is the number of nodes. Nevertheless, despite the reported time differences, the agent is able to generate high quality or even optimal solutions in 400 milliseconds, which makes the proposed model an appealing option.

Also, it should be noted that, although it is not visible in Table 3.5, the self-attention mechanism in the Transformer has $O(m^2)$ complexity, where m is total number of input elements [140]. The reported constant time for the tested problem instances is ensured by the GPU parallelization. Once the problem size become large enough, and hardware limitations of the GPU are reached, the effects of quadratic complexity become visible in the rapidly growing inference time. However, this limitation can be tackled by replacing the Transformer with a recently introduced variation called the *Linformer* [145], which reduces the self-attention complexity from $O(m^2)$ to $O(m)$ while maintaining similar performance in NLP tasks. Overall, the performance and the inference speed of the proposed model can be improved even further by tuning the hyperparameters, using more advanced RL training algorithms (e.g., rolling baseline [129], proximal policy [146]), using more recent variations of the Transformer [140, 147] and by doing model optimization[3] and compression [148].

3.6 Conclusions

This chapter investigated the use of DNN-based strategies for the distribution of event processing tasks across edge devices, considering three different optimization criteria. Each criteria is mathematically formulated, ensuring proper task execution, fair usage of resource and/or cost minimization related to device operation. Furthermore, an MDP formulation is introduced, which is followed by the presentation of the model based on Transformer and Ptr-Net architectures.

[3]See `https://www.tensorflow.org/lite/performance/model_optimization`

3.6 Conclusions

Simulation experiments show that the proposed model is capable of producing high quality solutions in all scenarios under consideration. Additionally, it is scalable and capable of solving problems with up to five time larger than the ones that it was trained upon. The latter is a very important ability as it allows to solve instances of different sizes without the need of retraining, which makes it easy to deploy in real world. Existing work applying DNN and RL methods to load balancing tend to neglect model scalability and only work with static environment settings, which is not applicable in real world.

In summary, the proposed model is generic and can be applied to problems that are modeled using two sets of elements and solved by taking sequential assignments decisions. The latter is an important characteristic in load balancing problems as the increasing network complexity will likely generate several problem variations for which there is no good performing heuristic, making RL and machine learning methods attractive and of practical value.

CHAPTER 4

Concluding Remarks

***Abstract*:** This chapter summarizes the main conclusions of the research presented in previous chapters, highlights the achievements of the work and outlines the directions for future research.

4.1 Summary

The goal of this thesis is to contribute to the creation and proper placement of event processing tasks connecting billions of devices. As seen throughout this work, achieving this goal requires application-layer interoperability and edge infrastructure, which plays a supportive role in event processing. This final chapter briefly summarizes the findings of this research work and answers to the questions outlined in Chapter 1: i) *What standards are required for distributed event processing to happen across heterogeneous devices?*; ii) *What is the role of edge computing in event processing?*; iii) *How edge devices can participate in event processing?*; iv) *What is the optimal placement of tasks for event processing to happen?*; v) *How can machine learning methods help in the distribution of event processing tasks?*.

Chapter 2 highlighted that standardization of event processing is a recent and rapidly evolving research topic. Many of the analyzed resource synchronization mechanisms are still in development and, therefore, may change or even be abandoned. Nevertheless, the analysis of their characteristics provides a view of the trend that this research area is following, giving some hints about the problems that future proposals should focus on.

The early and large scale adoption of MQTT was due to its ease of use and because it allowed to quickly build IoT applications. Another reason is that at the early stages of IoT there were no real concerns about the data silos, meaning that the interoperability was neglected. Moreover, the majority of applications were cloud-based where all the incoming data was processed at the cloud and, then, commands were sent back to actuators. However, the emergence of new applications, whose restrictions (e.g., delay sensitivity) can not be satisfied when using the cloud, caused a shift towards more machine-to-machine oriented solutions. This change of paradigm led to a standardization need of more strict

interfaces and communication patterns. CoAP Observe [42], conditional observe in CoAP and dynamic linking tackled this issue by providing a one-to-one resource synchronization mechanism. These proposals effectively solved the problem of sending resource state notification directly to a destination. Nevertheless, these proposals require client side processing of incoming data, i.e., require for the client to know how to extract the data that may be encoded in many different data formats. The Monitor proposal identified this limitation and outlined the need for mechanisms that inform the client of how to interpret the incoming data. It also offered a way of outsourcing the synchronization process to a third-party device. Yet, the Monitor proposal is a one-to-one synchronization mechanism that cannot cover complex scenarios. The latest proposals, `Rule` and Splot model, are being designed upon the ideas and the concepts of their predecessors. These synchronization mechanisms provide means to build endpoint-to-endpoint conditional notification between multiple devices. Also, they can scale from a simple one-to-one synchronization that can be executed in devices with limited resources to a complex many-to-many synchronization scenarios involving multiple devices.

Overall, the current trend leans toward many-to-many distributed event processing solutions that rely on REST principles and other design patterns that are common in "regular" Web, which are well-known to the users and the developers. By being designed from the ground up with interoperability in mind, new event processing solutions can be easily integrated into already existing infrastructure and IoT applications that follow the same principles.

Regarding the execution of event processing tasks, edge devices will play a crucial role as they can provide storage and computation near the data producers and consumers. However, due to the extremely dynamic nature of edge environments, where billions of devices will be available for the user, finding the right device for a task may be difficult. The discovery of edge devices, and of their availability to participate in event processing, involves several standards and proposals under development (e.g., HSML [71], CoRE resource directory [76], Thing directory [77]) that were also discussed in Chapter 2. As with event processing, these proposals follow "regular" Web standards and design patterns in order to avoid the creation of interoperability barriers. The dynamic nature of edge is also expressed in different form factor of devices, ranging from small and constrained devices to virtual/containerized nodes with high performance CPU and large amount of storage. For the latter, several tools and technologies for the management of the edge were presented in Chapter 3. Additionally, Chapter 3 presented and discussed the architecture and the organization of virtual edge nodes.

Overall, the current trend shows that tools, architecture, and technologies commonly used in management of cloud services are also moving toward the edge.

The placement and distribution of event processing tasks is another crucial issue that is currently under active research and it was a pivotal research topic of this the-

sis. Chapter 2 focused on the most recent CoRE-based approach for event processing, called `Rule`, and then proposed an improved rule decomposition and chaining method, called `Collecting Rule`, to achieve better reusability and scalability. The optimal placement of both approaches was investigated showing that `Collecting Rule` is a complement to the `Rules`-only approach.

Chapter 3 focused on attention-based DNN architecture and RL for distribution of event processing tasks. The proposed model was able to perform well in the three different scenarios under consideration, which shows that DNN models can be useful in load balancing and task distribution.

In summary, optimization techniques play a crucial role in networks and communication. However, the rapid growth of the complexity of the networks (e.g., complex traffic patterns, new QoS requirements) will make handcrafted solutions less practical. Therefore, we can safely state that the role of DNN in network management will continue to grow and their use in optimization tasks will be more prevalent.

4.2 Final considerations for future research

The completion of this thesis took approximately 4 years. It yielded several achievements in the application of DRL methods to the distribution of event processing tasks, and to combinatorial optimization in general. However, the work presented here is just a starting point, which opens many new research opportunities that can be explored using the proposed solution framework. Further research can be done to optimize the proposed DNN architecture, making it more robust and production-ready. This involves small tasks like hyperparameter tuning or more complex ones, like the development of new RL training algorithms or the design of new network architectures.

This work was based on the vanilla Transformer model, a model that revolutionized the DL when it was first introduced. Prior to its appearance there was a segmentation of architectures (e.g., LSMT for text, convolution networks for image) and techniques for different tasks. However, the Transformer changed this paradigm and it is slowly becoming a building block unifying different areas of research. Current state-of-the-art models used in NLP, computer vision [149], audio analysis [150] use the Transfomer model or its variations. Research in this area is evolving very rapidly, as reported in [140], between March and August 2020 a dozen new efficiency-focused Transformer variations were proposed. The cross-domain nature of Transfomer means that a contribution that improves its performance in NLP tasks will, most likely, apply to other research areas and problem domains. Hence, it would be quite interesting to explore new Transformer proposals and test their applicability in combinatorial optimization.

The current training time of 14 hours limits the exploration of new parameters and model variations. Code profiling revealed that the main bottleneck during training was the

environment implementation, which was running in a general purpose CPU. Therefore, another future research path could be GPU implementation of the RL environment, which would drastically speed up the training time.

Another interesting research topic is the utilization of the proposed DLR framework in other combinatorial problems (e.g., bin packing, vehicle routing problem, multiple knapsack), and subsequent performance evaluation.

In a nutshell, DLR is still in its infancy. There is no definitive cookbook stating what architecture, reward signal, training algorithm should be used to solve a specific problem. Current DRL is a hands-on approach where every small proposal must be experimented and evaluated, which gives a lot of room for further research and improvement.

References

[1] N. Feamster and J. Rexford, "Why (and how) networks should run themselves," *arXiv preprint arXiv:1710.11583*, 2017.

[2] Z. Dai, H. Liu, Q. V. Le, and M. Tan, "Coatnet: Marrying convolution and attention for all data sizes," *arXiv preprint arXiv:2106.04803*, 2021.

[3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[5] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[6] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[7] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[8] D. Guinard and V. Trifa, *Building the Web of Things: With Examples in Node.Js and Raspberry Pi*, 1st ed. Greenwich, CT, USA: Manning Publications Co., 2016.

[9] Z. Shelby, M. Koster, C. Groves, J. Zhu, and B. Silverajan, "Dynamic Resource Linking for Constrained RESTful Environments," Internet Engineering Task Force, Internet-Draft draft-ietf-core-dynlink-09, Jul. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-core-dynlink-09

[10] Interactive Hypermedia and Asynchronous Machine Interaction using Hypermedia Controls. [Accessed 30-03-2017]. [Online]. Available: http://iot-datamodels.blogspot.com/2017/04/interactive-hypermedia-and-asynchronous.html

[11] Working directory for OCF CRs and PRs. [Accessed 02-01-2019]. [Online]. Available: https://github.com/mjkoster/ocf-working/raw/master/docs/CR%20ATG%201968%20Rules.docx

[12] M. Weiser, "The computer for the 21st century," *IEEE pervasive computing*, vol. 1, no. 1, pp. 19–25, 2002.

[13] C. V. Networking, "Cisco Global Cloud Index: Forecast and Methodology, 2016–2021 White Paper," 2018.

[14] Software Developers, Quality Assurance Analysts, and Testers. Occupational Outlook Handbook. U.S. Bureau of Labor Statistics. [Online]. Available: https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm

[15] J. Sutherland and J. Sutherland, *Scrum: the art of doing twice the work in half the time*. Currency, 2014.

[16] Web of Things (WoT) Architecture. [Accessed 14-09-2021]. [Online]. Available: https://w3c.github.io/wot-architecture/

[17] Web of Things (WoT) Protocol Binding Templates. [Accessed 14-09-2021]. [Online]. Available: https://w3c.github.io/wot-binding-templates/

[18] OCF Core Specification 2.0.4. [Accessed 02-09-2019]. [Online]. Available: https://openconnectivity.org/developer/specifications

[19] iotschema.org. [Accessed 14-09-2021]. [Online]. Available: http://iotschema.org/

[20] Matter is the foundation for connected things. [Accessed 14-09-2021]. [Online]. Available: https://buildwithmatter.com/

[21] oneM2M Sets Standards For The Internet Of Things and M2M. [Accessed 14-09-2021]. [Online]. Available: https://www.onem2m.org/

[22] One Data Model. [Accessed 14-09-2021]. [Online]. Available: https://onedm.org/

[23] Azure Digital Twin Definition Language. [Accessed 14-09-2021]. [Online]. Available: https://docs.microsoft.com/en-us/azure/digital-twins/concepts-models

[24] Azure Digital Twin Definition Language. [Accessed 14-09-2021]. [Online]. Available: https://omaspecworks.org/what-is-oma-specworks/iot/lightweight-m2m-lwm2m/

[25] OPC UA. [Accessed 14-09-2021]. [Online]. Available: https://opcfoundation.org/

[26] A. Mazayev, J. A. Martins, and N. Correia, "Interoperability in IoT through the semantic profiling of objects," *Ieee Access*, vol. 6, pp. 19 379–19 385, 2017.

[27] ——, "Semantic web thing architecture," in *2017 4th Experiment@ International Conference (exp. at'17)*. IEEE, 2017, pp. 43–46.

[28] ——, "Semantically enriched hypermedia apis for next generation IoT," in *Interoperability, Safety and Security in IoT*. Springer, 2017, pp. 19–26.

[29] A. Keränen, F. M. Kovatsch, and K. Hartke, "Guidance on RESTful Design for Internet of Things Systems," Internet Engineering Task Force, Internet-Draft draft-irtf-t2trg-rest-iot-08, Aug. 2021, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-irtf-t2trg-rest-iot-08

[30] A. Mazayev and N. Correia, "A distributed core-based resource synchronization mechanism," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4625–4640, 2019.

[31] IoT Analytics - Market insights for the Internet of Things. State of the IoT 2018: Number of IoT devices now at 7B – Market accelerating. [Accessed 31-10-2019]. [Online]. Available: https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/

[32] C. Bormann, M. Ersue, and A. Keränen, "Terminology for Constrained-Node Networks," RFC 7228, May 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7228.txt

[33] S. Duquennoy, G. Grimaud, and J.-J. Vandewalle, "The web of things: interconnecting devices with high usability and performance," in *2009 International Conference on Embedded Software and Systems*. IEEE, 2009, pp. 323–330.

[34] D. Guinard, V. Trifa, T. Pham, and O. Liechti, "Towards physical mashups in the Web of Things," in *Proceedings of the 6th International Conference on Networked Sensing Systems*, ser. INSS'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 196–199. [Online]. Available: http://dl.acm.org/citation.cfm?id=1802340.1802386

[35] IPv6 over Low power WPAN (6LoWPAN). [Online]. Available: https://datatracker.ietf.org/wg/6lowpan/documents/

[36] Constrained RESTful Environments (CoRE). [Online]. Available: https://datatracker.ietf.org/wg/core/about/

[37] G. Montenegro, J. Hui, D. Culler, and N. Kushalnagar, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," RFC 4944, Sep. 2007. [Online]. Available: https://rfc-editor.org/rfc/rfc4944.txt

[38] P. Thubert and J. Hui, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks," RFC 6282, Sep. 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6282.txt

[39] R. Alexander, A. Brandt, J. Vasseur, J. Hui, K. Pister, P. Thubert, P. Levis, R. Struik, R. Kelsey, and T. Winter, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC 6550, Mar. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6550.txt

[40] R. T. Fielding and R. N. Taylor, *Architectural Styles and The Design of Network-based Software Architectures*. University of California, Irvine Doctoral dissertation, 2000, vol. 7.

[41] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," RFC 7252, Jun. 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7252.txt

[42] K. Hartke, "Observing Resources in the Constrained Application Protocol (CoAP)," RFC 7641, Sep. 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7641.txt

[43] Internet Assigned Numbers Authority - Media Types. [Accessed 02-01-2018]. [Online]. Available: https://www.iana.org/assignments/media-types/media-types.xhtml

[44] M. Kovatsch, "Scalable Web Technology for the Internet of Things," Ph.D. dissertation, ETH Zurich, 2015.

[45] M. Aazam, S. Zeadally, and K. A. Harras, "Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities," *Future Generation Computer Systems*, 2018.

[46] M. Mukherjee, L. Shu, and D. Wang, "Survey of Fog Computing: Fundamental, Network Applications, and Research Challenges," *IEEE Communications Surveys & Tutorials*, 2018.

[47] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the Internet of Things," *IEEE Access*, vol. 6, pp. 6900–6919, 2017.

[48] H. Mora, M. T. Signes-Pont, D. Gil, and M. Johnsson, "Collaborative Working Architecture for IoT-Based Applications," *Sensors*, vol. 18, no. 6, p. 1676, 2018.

[49] H. Mora, J. F. Colom, D. Gil, and A. Jimeno-Morenilla, "Distributed computational model for shared processing on Cyber-Physical System environments," *Computer Communications*, vol. 111, pp. 68–83, 2017.

[50] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner, "Resource provisioning for IoT services in the fog," in *2016 IEEE 9th Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2016, pp. 32–39.

[51] L. Kim-Hung, S. K. Datta, C. Bonnet, F. Hamon, and A. Boudonne, "A scalable IoT framework to design logical data flow using virtual sensor," in *2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, 2017, pp. 1–7.

[52] Node-RED - Flow-based programming for the Internet of Things. [Accessed 25-07-2018]. [Online]. Available: https://nodered.org/

[53] Eclipse Kura. [Accessed 04-10-2019]. [Online]. Available: https://www.eclipse.org/kura/

[54] Web of Things (WoT) Thing Description. [Accessed 14-09-2021]. [Online]. Available: https://w3c.github.io/wot-thing-description/

[55] Web Thing API. [Accessed 30-03-2018]. [Online]. Available: https://iot.mozilla.org/wot/

[56] IFTTT every thing works better together. [Accessed 25-07-2018]. [Online]. Available: https://ifttt.com/

[57] Data Infrastructure at IFTTT. [Accessed 04-09-2019]. [Online]. Available: https://medium.com/engineering-at-ifttt/data-infrastructure-at-ifttt-35414841f9b5

[58] N. Raveendranathan, S. Galzarano, V. Loseu, R. Gravina, R. Giannantonio, M. Sgroi, R. Jafari, and G. Fortino, "From modeling to implementation of virtual sensors in body sensor networks," *IEEE Sensors Journal*, vol. 12, no. 3, pp. 583–593, 2012.

[59] A.-R. Al-Ali, I. A. Zualkernan, M. Rashid, R. Gupta, and M. Alikarar, "A smart home energy management system using iot and big data analytics approach," *IEEE Transactions on Consumer Electronics*, vol. 63, no. 4, pp. 426–434, 2017.

[60] A. Antonić, M. Marjanović, P. Skočir, and I. P. Žarko, "Comparison of the CUPUS middleware and MQTT protocol for smart city services," in *2015 13th International Conference on Telecommunications (ConTEL)*. IEEE, 2015, pp. 1–8.

[61] S.-M. Kim, H.-S. Choi, and W.-S. Rhee, "IoT home gateway for auto-configuration and management of MQTT devices," in *2015 IEEE Conference on Wireless Sensors (ICWiSe)*. IEEE, 2015, pp. 12–17.

[62] Azure IoT Hub. [Accessed 04-10-2019]. [Online]. Available: https://aws.amazon.com/iot/

[63] Azure IoT Hub. [Accessed 04-10-2019]. [Online]. Available: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-mqtt-support

[64] CloudMQTT. [Accessed 04-10-2019]. [Online]. Available: https://www.cloudmqtt.com/

[65] Organization for the Advancement of Structured Information Standards (OASIS). (2014) Mqtt version 3.1.1. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

[66] A. Minteer, *Analytics for the Internet of Things (IoT)*. Packt Publishing, 2017. [Online]. Available: https://books.google.pt/books?id=48NCMQAACAAJ

[67] MQTT and CoAP, IoT Protocols. [Accessed 09-04-2019]. [Online]. Available: https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php

[68] G. K. Teklemariam, J. Hoebeke, I. Moerman, and P. Demeester, "Facilitating the creation of IoT applications through conditional observations in CoAP," *EURASIP Journal on Wireless Communications and Networking*, vol. 2013, no. 1, p. 177, 2013.

[69] OCF - oic.r.sensor.presence.json. [Accessed 02-01-2018]. [Online]. Available: https://oneiota.org/revisions/3060

[70] OCF - oic.r.sensor.presence.json. [Accessed 02-01-2018]. [Online]. Available: https://oneiota.org/revisions/3046

[71] M. Koster, "Media Types for Hypertext Sensor Markup," Internet Engineering Task Force, Internet-Draft draft-koster-t2trg-hsml-01, Mar. 2017, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-koster-t2trg-hsml-01#section-9.2

[72] Z. Shelby, M. Koster, C. Groves, J. Zhu, and B. Silverajan, "Reusable Interface Definitions for Constrained RESTful Environments," Internet Engineering Task Force, Internet-Draft draft-ietf-core-interfaces-14, Mar. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-core-interfaces-14

[73] Z. Shelby, "Constrained RESTful Environments (CoRE) Link Format," RFC 6690, Aug. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6690.txt

[74] K. Debique, T. Igarashi, S. Kou, J. Moonen, J. Ritchie, G. Schults, and M. Walker, "Contentdirectory: 1 service template version 1.01," in *Online, UPnP Forum, June*, 2002. [Online]. Available: http://www.upnp.org/specs/av/UPnP-av-ContentDirectory-v1-Service.pdf

[75] I. ETSI, "ETSI GS CIM 004 (v1. 1.1)," Technical Report. ETSI. 2018, Tech. Rep., 2018. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/CIM/001_099/004/01.01.01_60/gs_cim004v010101p.pdf

[76] Z. Shelby, M. Koster, C. Bormann, P. V. der Stok, and C. Amsüss, "CoRE Resource Directory," Internet Engineering Task Force, Internet-Draft draft-ietf-core-resource-directory-23, Jul. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-core-resource-directory-23

[77] W3C WoT Thing Directory implementation. [Accessed 09-09-2019]. [Online]. Available: https://github.com/thingweb/thingweb-directory

[78] Simple Monitoring and Control Protocol (SMCP). [Accessed 09-04-2019]. [Online]. Available: https://github.com/google/splot-java

[79] Create, Delete, and Linkupdate Inerface Definitions. [Accessed 02-01-2019]. [Online]. Available: https://github.com/mjkoster/ocf-working/raw/master/docs/CR%20ATG%201229%20Create.docx

[80] P. V. der Stok, C. Bormann, and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)," RFC 8132, Apr. 2017. [Online]. Available: https://rfc-editor.org/rfc/rfc8132.txt

[81] J. Schaad, "CBOR Object Signing and Encryption (COSE)," RFC 8152, Jul. 2017. [Online]. Available: https://rfc-editor.org/rfc/rfc8152.txt

[82] G. Selander, J. Mattsson, F. Palombini, and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)," RFC 8613, Jul. 2019. [Online]. Available: https://rfc-editor.org/rfc/rfc8613.txt

[83] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant Resource Fairness: Fair Allocation of Multiple Resource Types." in *Nsdi*, vol. 11, no. 2011, 2011, pp. 24–24.

[84] C. Bormann and Z. Shelby, "Block-wise transfers in the constrained application protocol (CoAP)," Tech. Rep., 2016.

[85] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," in *2017 IEEE international systems engineering symposium (ISSE)*. IEEE, 2017, pp. 1–7.

[86] D. Thangavel, X. Ma, A. Valera, H.-X. Tan, and C. K.-Y. Tan, "Performance evaluation of MQTT and CoAP via a common middleware," in *2014 IEEE ninth international conference on intelligent sensors, sensor networks and information processing (ISSNIP)*. IEEE, 2014, pp. 1–6.

[87] M. H. Amaran, N. A. M. Noh, M. S. Rohmad, and H. Hashim, "A comparison of lightweight communication protocols in robotic applications," *Procedia Computer Science*, vol. 76, pp. 400–405, 2015.

[88] S. Bandyopadhyay and A. Bhattacharyya, "Lightweight Internet protocols for web enablement of sensors using constrained gateway devices," in *2013 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2013, pp. 334–340.

[89] N. De Caro, W. Colitti, K. Steenhaut, G. Mangino, and G. Reali, "Comparison of two lightweight protocols for smartphone-based sensing," in *2013 IEEE 20th Symposium on Communications and Vehicular Technology in the Benelux (SCVT)*. IEEE, 2013, pp. 1–6.

[90] M. Koster, A. Keränen, and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)," Internet Engineering Task Force, Internet-Draft draft-ietf-core-coap-pubsub-08, Mar. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-core-coap-pubsub-08

[91] J. Haikara, "Publish-Subscribe Communication for CoAP," 2017.

[92] J. Hong, Y.-G. Hong, and J.-S. Youn, "Problem Statement of IoT integrated with Edge Computing," Internet Engineering Task Force, Internet-Draft draft-hong-iot-edge-computing-02, Mar. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-hong-iot-edge-computing-02

[93] J. Hong, Y.-G. Hong, X. de Foy, M. Kovatsch, E. Schooler, and D. Kutscher, "IoT Edge Challenges and Functions," Internet Engineering Task Force, Internet-Draft draft-irtf-t2trg-iot-edge-01, Oct. 2020, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-irtf-t2trg-iot-edge-01

[94] EVE - LF Edge. [Accessed 01-03-2021]. [Online]. Available: https://kubernetes.io/blog/2016/07/autoscaling-in-kubernetes/

[95] Edge X Foundry. [Accessed 01-03-2021]. [Online]. Available: https://www.edgexfoundry.org/

[96] MobiledgeX. [Accessed 01-03-2021]. [Online]. Available: https://mobiledgex.com/

[97] NGINX — High Performance Load Balancer, Web Server and Reverse Proxy. [Accessed 15-09-2020]. [Online]. Available: https://www.nginx.com/

[98] Traefik - Open Source load balancer and reverse proxy for microservices. [Accessed 15-09-2020]. [Online]. Available: https://docs.traefik.io/

[99] Moleculer - Progressive microservices framework for Node.js. [Online]. Available: https://moleculer.services/

[100] Using nginx as HTTP load balancer. [Accessed 15-09-2020]. [Online]. Available: http://nginx.org/en/docs/http/load_balancing.html

[101] Services - Traefik. [Accessed 15-09-2020]. [Online]. Available: https://docs.traefik.io/routing/services/#load-balancing

[102] Load Balancing — Moleculer - Progressive microservices framework for Node.js. [Online]. Available: https://moleculer.services/docs/0.14/balancing.html#Built-in-strategies

[103] Autoscaling in Kubernetes. [Accessed 15-09-2020]. [Online]. Available: https://kubernetes.io/blog/2016/07/autoscaling-in-kubernetes/

[104] StarlingX. [Accessed 15-09-2020]. [Online]. Available: https://www.starlingx.io/

[105] M. V. Modenesi, A. G. Evsukoff, and M. C. Costa, "A load balancing knapsack algorithm for parallel fuzzy c-means cluster analysis," in *International Conference on High Performance Computing for Computational Science*. Springer, 2008, pp. 269–279.

[106] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *arXiv preprint arXiv:1611.09940*, 2016.

[107] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[108] A. Shakarami, M. Ghobaei-Arani, M. Masdari, and M. Hosseinzadeh, "A survey on the computation offloading approaches in mobile edge/cloud computing environment: a stochastic-based perspective," *Journal of Grid Computing*, pp. 1–33, 2020.

[109] L. Lei, Y. Tan, K. Zheng, S. Liu, K. Zhang, and X. Shen, "Deep reinforcement learning for autonomous internet of things: Model, applications and challenges," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1722–1760, 2020.

[110] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d'horizon," *European Journal of Operational Research*, 2020.

[111] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *arXiv preprint arXiv:2003.03600*, 2020.

[112] Y.-H. Chiang, T.-W. Chiang, T. Zhang, and Y. Ji, "Deep-dual-learning-based cotask processing in multiaccess edge computing systems," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9383–9398, 2020.

[113] Z. Zhang, C. Li, S. Peng, and X. Pei, "A new task offloading algorithm in edge computing," *EURASIP Journal on Wireless Communications and Networking*, vol. 2021, no. 1, pp. 1–21, 2021.

[114] Y. Xu, W. Xu, Z. Wang, J. Lin, and S. Cui, "Deep reinforcement learning based mobility load balancing under multiple behavior policies," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.

[115] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International conference on machine learning*. PMLR, 2014, pp. 387–395.

[116] Z. Zhang, F. R. Yu, F. Fu, Q. Yan, and Z. Wang, "Joint offloading and resource allocation in mobile edge computing systems: An actor-critic approach," in *2018 IEEE global communications conference (GLOBECOM)*. IEEE, 2018, pp. 1–6.

[117] M. G. R. Alam, M. M. Hassan, M. Z. Uddin, A. Almogren, and G. Fortino, "Autonomic computation offloading in mobile edge for iot applications," *Future Generation Computer Systems*, vol. 90, pp. 149–157, 2019.

[118] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4005–4018, 2018.

[119] K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Deep learning empowered task offloading for mobile edge computing in urban informatics," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 7635–7647, 2019.

[120] T. Yang, Y. Hu, M. C. Gursoy, A. Schmeink, and R. Mathar, "Deep reinforcement learning based resource allocation in low latency edge computing networks," in *2018 15th International Symposium on Wireless Communication Systems (ISWCS)*. IEEE, 2018, pp. 1–5.

[121] D. Van Le and C.-K. Tham, "A deep reinforcement learning based offloading scheme in ad-hoc mobile clouds," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2018, pp. 760–765.

[122] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in neural information processing systems*, 2015, pp. 2692–2700.

[123] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[124] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[125] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[126] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[127] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[128] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takác, "Reinforcement learning for solving the vehicle routing problem," in *Advances in Neural Information Processing Systems*, 2018, pp. 9839–9849.

[129] W. Kool, H. Van Hoof, and M. Welling, "Attention, learn to solve routing problems!" *arXiv preprint arXiv:1803.08475*, 2018.

[130] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[131] International Business Machines (IBM) Corporation, "IBM ILOG CPLEX Optimizer," 2021. [Online]. Available: https://www.ibm.com/products/ilog-cplex-optimization-studio

[132] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2021. [Online]. Available: https://www.gurobi.com

[133] R. WILLIAMS, "Simple Statistical Gradient-following Algorithms for Connectionist Reinforcement Learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, MAY 1992.

[134] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[135] Z. Wang, N. de Freitas, and M. Lanctot, "Dueling Network Architectures for Deep Reinforcement Learning," Tech. Rep. arXiv:1511.06581, 2015. [Online]. Available: http://arxiv.org/abs/1511.06581

[136] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1928–1937. [Online]. Available: http://proceedings.mlr.press/v48/mniha16.html

[137] S. Gu and T. Hao, "A pointer network based deep learning algorithm for 0–1 knapsack problem," in *2018 Tenth International Conference on Advanced Computational Intelligence (ICACI)*. IEEE, 2018, pp. 473–477.

[138] O. Vinyals, S. Bengio, and M. Kudlur, "Order matters: Sequence to sequence for sets," *arXiv preprint arXiv:1511.06391*, 2015.

[139] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[140] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," *arXiv preprint arXiv:2009.06732*, 2020.

[141] Registry and Discovery — Moleculer - Progressive microservices framework for Node.js. [Online]. Available: https://moleculer.services/docs/0.14/registry.html

[142] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[143] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[144] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[145] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-attention with linear complexity," *arXiv preprint arXiv:2006.04768*, 2020.

[146] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[147] Y. Tay, M. Dehghani, S. Abnar, Y. Shen, D. Bahri, P. Pham, J. Rao, L. Yang, S. Ruder, and D. Metzler, "Long range arena: A benchmark for efficient transformers," *arXiv preprint arXiv:2011.04006*, 2020.

[148] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," *arXiv preprint arXiv:1710.09282*, 2017.

[149] N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, and D. Tran, "Image transformer," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4055–4064.

[150] Y. Gong, Y.-A. Chung, and J. Glass, "Ast: Audio spectrogram transformer," *arXiv preprint arXiv:2104.01778*, 2021.