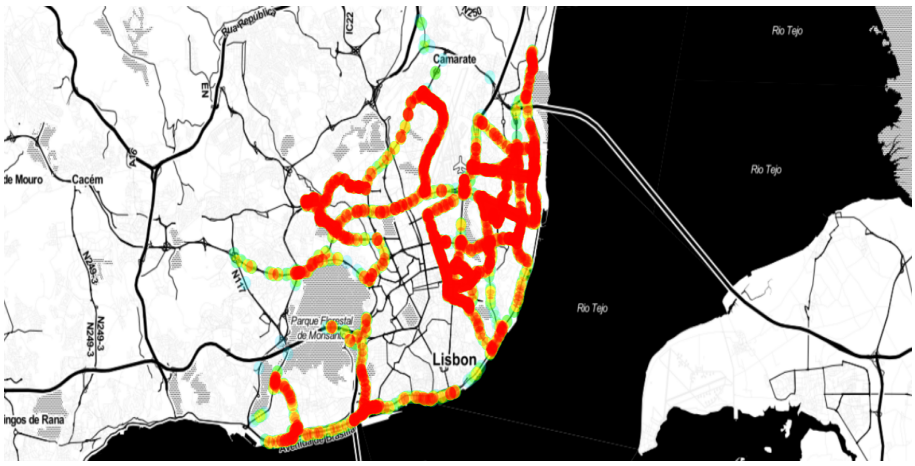
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores



Plataforma de avaliação de QoS de uma rede LoRaWAN

João Pedro Vitorino

Licenciado em Engenharia Informática, Redes e Telecomunicações

Projeto Final para obtenção do Grau de Mestre
em Engenharia Informática e de Computadores

Orientador : Professor Doutor Nuno Cruz

Júri:

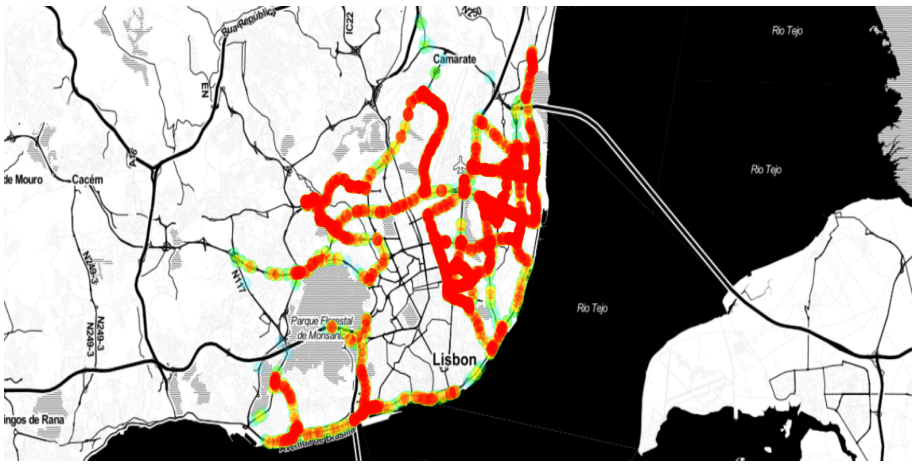
Presidente: Professor Doutor José Simão

Vogais: Professor Doutor Nuno Datia
Professor Doutor Nuno Cruz

Dezembro, 2022

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores



Plataforma de avaliação de QoS de uma rede LoRaWAN

João Pedro Vitorino

Licenciado em Engenharia Informática, Redes e Telecomunicações

Projeto Final para obtenção do Grau de Mestre
em Engenharia Informática e de Computadores

Orientador : Professor Doutor Nuno Cruz

Júri:

Presidente: Professor Doutor José Simão

Vogais: Professor Doutor Nuno Datia
Professor Doutor Nuno Cruz

Dezembro, 2022

À minha família.

Agradecimentos

Gostaria de agradecer ao meu orientador, o Professor Doutor Nuno Cruz, por todo o apoio cientifico e disponibilidade que demonstrou durante a elaboração deste projeto.

A toda a equipa do grupo de investigação Future Internet Technologies, do Instituto Superior de Engenharia de Lisboa. Em particular, aos professores José Simão, Matilde Pato e Nuno Datia, por todo o seu apoio e aconselhamento.

A todos os meus colegas de curso e de bolsa.

Agradeço também à minha família e amigos pelo continuo apoio dado ao longo de todo o percurso académico.

Resumo

As Cidades Inteligentes são, hoje em dia, uma realidade inevitável e em crescimento. Realidade apoiada em plataformas de software de gestão das cidades, através do processamento e apresentação de um grande volume de dados, obtidos a partir de sensores utilizados em todas as cidades. As redes designadas Low-Power Wide Area Networks (LPWAN) alavancam o processo de sensorização. No entanto, a paisagem urbana, por sua vez, induz uma elevada probabilidade de mudança nas condições de propagação das redes LPWAN, exigindo assim soluções de monitorização ativas para avaliar a condição das redes LPWAN presentes nas cidades. As soluções atualmente existentes consideram geralmente a existência de apenas um tipo de rede LPWAN a ser monitorizada. Nesta tese é apresentada uma arquitetura para uma plataforma de agregação de métricas sobre redes LPWAN heterogéneas. A arquitetura, designada como IoTMapper, combina componentes específicos construidos de raiz com componentes existentes dos ecossistemas FIWARE e Apache Kafka. Os detalhes de implementação para as redes LPWAN são abstraídos por adaptadores de modo a que novas redes possam ser facilmente adicionadas. A validação foi realizada utilizando dados reais recolhidos para a Rede de Longa Distância de Área (LoRaWAN) em Lisboa, e um conjunto de dados simulados extrapolados a partir dos dados recolhidos. Os resultados indicam que a arquitetura apresentada é uma solução viável para a agregação de métricas, que pode ser expandida para suportar múltiplas redes. No entanto, alguns dos componentes FIWARE considerados apresentam pontos de limitação de desempenho que podem dificultar a escalabilidade da arquitetura no processamento da receção de novas mensagens.

Palavras-chave: Cidades Inteligentes; Internet das Coisas; FIWARE; Apache Kafka; LoRaWAN; Low-Power Wide Area Network (LPWAN)

Abstract

Smart Cities are, nowadays, an unavoidable and growing reality, supported on software platforms that support city management, through the processing and presentation of a large number of data, obtained from sensors used throughout the cities. Low-Power Wide Area Networks (LPWAN) leverage the sensorization process, however, urban landscape, in turn, induces a high probability of change in the propagation conditions of the LPWAN network, thus requiring active monitoring solutions for assessing the city LPWAN network condition. Currently existing solutions usually consider the existence of only one type of LPWAN network to be monitored. In this thesis a architecture for a plataform for aggregation of metrics from heterogeneous LPWAN networks is presented. The architecture, named as IoTMapper, combines purpose build components with existing components from the FIWARE and Apache Kafka ecosystems. Implementation details for the LPWAN networks are abstracted by adapters so that new networks may be easily added. The validation was carried out using real data collected for Long-Range Wide-Area Network (LoRaWAN) in Lisbon, and a simulated data set extrapolated from the collected data. The results indicate that the presented architecture is a viable solution for metrics aggregation, that may be expanded to support multiple networks. Whoever, some of the considered FIWARE components present performance bottlenecks that may hinder the scaling of the architecture while processing new message arrivals.

Keywords: Smart Cities; Internet of Things; FIWARE; Apache Kafka; LoRaWAN; Low-Power Wide Area Network (LPWAN)

Índice

Lista de Figuras **xvii**

Lista de Tabelas **xix**

Lista de Abreviaturas e Siglas **xxi**

Glossário **xxv**

1 Introdução **1**

1.1 Contribuições 3

1.2 Contexto do Projeto 3

1.2.1 Redes LPWAN 3

1.2.1.1 Redes LoRaWAN 5

1.2.2 O projeto FIWARE 6

1.3 Objetivos 7

1.4 Requisitos da Solução 8

1.5 Resultados Adicionais do Trabalho 8

1.6 Organização do Documento 9

2 Estado da Arte **11**

2.1 Introdução 11

2.1.1 Organização 11

2.1.2 Trabalho Relacionado 12
2.1.2.1 TTN Mapper 13
2.1.2.2 Helium Mapper 15
2.1.2.3 Conclusão do Trabalho Relacionado 16
2.2 Distribuição de Dados 17
2.2.1 Distribuição via Serviços 17
2.2.1.1 Serviços Web 18
2.2.1.2 Serviços RPC 19
2.2.2 Distribuição via Passagem Assíncrona de Mensagens 19
2.2.2.1 Apache Kafka 20
2.2.2.2 MQTT 21
2.2.2.3 NATS 21
2.2.2.4 OpenDDS 21
2.2.2.5 RabbitMQ (AMQP) 22
2.2.2.6 Redis 22
2.2.2.7 ZeroMQ 22
2.2.3 Comparação de Mecanismos 23
2.3 Processamento de Dados 24
2.3.1 Processamento em Fluxo 25
2.3.1.1 Kafka Streams 26
2.3.1.2 Apache Flink 26
2.3.1.3 Apache Spark 27
2.3.2 Comparação de Mecanismos 27
2.4 Persistência de Dados 29
2.4.1 MongoDB 30
2.4.2 Apache Cassandra 30
2.4.3 Apache HBase 31
2.4.4 Comparação de Mecanismos 31
2.5 Interface com Utilizadores e Sistemas Externos 32

2.5.1 Spring Framework 32
2.5.2 React 33
2.6 Resumo do Estudo Realizado 33

3 Arquitetura e Implementação Propostas 35
3.1 Introdução 35
3.2 Visão Geral da Arquitetura 36
3.3 Visão Detalhada da Arquitetura 37
3.4 Arquitetura do Processamento em Fluxo 40
3.5 Modelos de Dados 43
3.5.1 Recolha de Métricas 43
3.5.2 Representação das Métricas no Apache Kafka 44
3.5.3 Representação das Métricas Processadas 45
3.6 Considerações sobre os Componentes FIWARE 47
3.6.1 Adaptação do FIWARE Cygnus 47
3.6.2 Implementação e Adaptação dos LPWANReceivers 47
3.7 Interfaces para Visualização das Métricas 48
3.7.1 IoTMapperBackend 48
3.7.2 IoTMapperPresentation 49
3.8 Considerações Adicionais de Implementação 50
3.8.1 Fluxos de Dados 51
3.8.2 Balanceamento de Carga 52
3.8.2.1 Variação do Volume na *Input Layer* 52
3.8.2.2 Variação do Volume na *Exit Layer* 53

4 Validação, Avaliação e Discussão de Resultados 55
4.1 Introdução 55
4.1.1 Organização 55
4.2 Conjunto de dados 56
4.3 Metodologia de Validação e Avaliação 57

4.3.1 Condições Gerais de Teste 57
4.3.2 Condições para a Fase 1 58
4.3.3 Condições para a Fase 2 58
4.3.3.1 Condições para Testes Direcionados ao Fluxo de Entrada 58
4.3.3.2 Condições para Testes Direcionados ao Fluxo de Saída . 60
4.4 Fase 1 - Validação da Arquitectura 61
4.5 Fase 2 - Avaliação da Implementação 61
4.5.1 Testes direcionados ao Fluxo de Entrada 62
4.5.1.1 Latência 62
4.5.1.2 Taxas de Erros/Perdas 65
4.5.2 Testes direcionados ao Fluxo de Saída 66

5 Conclusões e Trabalho Futuro **71**

Referências **75**

A Exemplo do Formato de Dados da TTN **i**

B Parâmetros das Redes LPWAN **v**
B.1 Descritores das Redes LPWAN v
B.2 Métricas Consideradas nas Redes LPWAN vi
B.3 Factores para Reinício das Agregações das Métricas vi

C Modelos FIWARE **vii**
C.1 Recolha de Métricas vii
C.2 Resultados das Métricas Processadas e Agregadas ix
C.3 Informação sobre as Redes LPWAN xiii

D Web API para IoTMapperBackend **xv**

Lista de Figuras

1.1 Visão de alto nível da arquitetura de uma rede LoRaWAN. Obtido de [8]. 5

1.2 Arquitetura Geral de referência do FIWARE para as Cidades Inteligentes. Obtido de [9]. 6

2.1 Exemplo de utilização do TTN Mapper, centrado em Lisboa a data 11/11/2021. 14

2.2 Exemplo de utilização do Helium Mapper, centrado em Lisboa a data de 11/11/2021. 16

3.1 Visão Arquitetural Geral. 36

3.2 Visão Detalhada da Arquitetura, com escolhas de alternativas tecnológicas. As cores utilizadas são meramente ilustrativas. 38

3.3 Topologia da Arquitetura para processamento em fluxo das métricas recolhidas. As cores utilizadas são meramente ilustrativas. 40

3.4 Arquitetura lógica interna do *IoTMapperBackend*. 49

3.5 Interface gráfica para o *IoTMapperPresentation*, com o conjunto simulado de dados (*Data-02* definido na Secção 4.2). Como exemplo da interface de referência. 50

3.6 Visão da Arquitetura de Implementação sobre Kubernetes. 51

4.1 Metodologia de avaliação, com identificação dos pontos de medição de AL e ALD 59

4.2 Resultado da Fase 1 (validação), apresentando a cobertura recolhida para a cidade de Lisboa, Portugal. 62

4.3 AL e ALD registados para a segunda fase de avaliação. LoRaWAN para 1 dispositivo simulado. 63

4.4 AL e ALD registados para a segunda fase de avaliação. LoRaWAN para 100 dispositivos simulados. 64

4.5 AL e ALD registados para a segunda fase de avaliação. LoRaWAN para 100 dispositivos simulados, e 3 instâncias do IoTAgent-LoRaWAN. . . . 65

4.6 TER registado para a segunda fase de avaliação. 66

4.7 Desempenho da *Exit Layer* para os cenários identificados. 68

4.8 Uso de recursos pela *Exit Layer*, medido na máquina TFM-01, para os cenários identificados. 69

Lista de Tabelas

1.1 Comparação das principais características de algumas tecnologias LPWAN, adaptado de [15]. 4

2.1 Comparação de tecnologias de passagem assíncrona de mensagens, adaptado de [49]. 23

3.1 Modelo lógico para métricas recolhidas e por processar. 44
3.2 Correspondência do modelo lógico da Tabela 3.1 no Orion. 44
3.3 Mapeamento da entidade pelo FIWARE Cygnus. 45
3.4 Modelo lógico para métricas após processamento. 46
3.5 Níveis de *cache* considerados. 54

4.1 Máquinas utilizadas na instanciação e validação. 57
4.2 Mapeamentos para legendas de figuras, definindo pontos de medição para a latência. 59
4.3 Cenários para o fluxo de entrada. 67

B.1 Mapeamento dos descritores. v
B.2 Métricas consideradas para as redes LPWAN. vi
B.3 Métricas consideradas para as redes LPWAN. vi

Lista de Abreviaturas e Siglas

ACE	Adaptive Communication Environment.
AMQP	Advanced Message Queuing Protocol.
API	Application Programming Interface.
ARP	Address Resolution Protocol.
CORBA	Common Object Request Broker Architecture.
CPU	Central Processing Unit.
CQL	Cassandra Query Language.
DCPS	Data-Centric Publish-Subscribe.
DCPSInfoRepo	DCPS Information Repository.
DDS	Data Distribution Service.
EJB	Enterprise Java Beans.
ETL	Extract Transform Load.
FIWARE	Future Internet Software.
GE	Generic Enabler.
GW	Gateway.
HDFS	Hadoop Distributed File System.
HTML	HyperText Markup Language.
HTTP	Hypertext Transfer Protocol.

IdC	Internet das Coisas.
IoT	Internet of Things.
IP	Internet Protocol.
IPC	Inter-Process Communication.
ISEL	Instituto Superior de Engenharia de Lisboa.
JSON	Javascript Object Notation.
LoRaWAN	Long Range Wide-Area Network.
LPWAN	Low-Power Wide-Area Network.
MQTT	Message Queuing Telemetry Transport.
NB-IoT	Narrowband Internet of Things.
NGSI	Next Generation Service Interfaces.
OCB	Orion Context Broker.
OCM	Operadores de Comunicações Móveis.
QoS	Quality of Service.
RDD	Resilient Distributed Dataset.
REST	REpresentational State Transfer.
RMI	Remote Method Invocation.
RPC	Remote Procedure Call.
RSRP	Reference Signal Receive Power.
RSSI	Received Signal Strength Indication.
RTPS	Real-time Publish-Subscribe.
SNR	Signal to Noise Ratio.
SOA	Service Oriented Architecture.
SOAP	Simple Object Access Protocol.
SQL	Structured Query Language.
STOMP	Streaming Text Oriented Messaging Protocol.
TAO	The ACE Object Request Broker.
TCP	Transport Control Protocol.

TTN	The Things Network.
UDP	User Datagram Protocol.
URL	Uniform Resource Locator.
WSDL	Web Services Description Language.
XML	Extensible Markup Language.

Glossário

Cidade Inteligente a sensorização, no contexto da Internet das Coisas, de ambientes urbanos com o intuito de, através de analise dos dados recolhidos, melhorar as condições de vida dos cidadãos assim como a qualidade e sustentabilidade dos serviços prestados.

Coisa objetos do dia-a-dia aumentados com micro-controladores, interface radio, sensores, atuadores e pilha de protocolos necessários a comunicação. Tipicamente usados em ambientes com recursos limitados e de baixo custo.

Consistência propriedade de um sistema capaz de garantir um estado que respeite os requisitos arquitecturais, mesmo perante falhas e conflitos.

Disponibilidade capacidade de um Sistema Distribuído de continuar a responder a pedidos perante falhas, ou, carga elevada.

Escalabilidade capacidade de um Sistema Distribuído de responder adequadamente a variações na pressão (carga) colocada sobre o mesmo.

Fiabilidade capacidade de um Sistema Distribuído tolerar a ocorrência de falhas, continuando a demonstrar o comportamento esperado sem perda de informação pela qual é responsável.

Gateway equipamento, normalmente fixo, com interface radio que oferece uma ou mais pilhas de protocolos as Coisa.

Internet das Coisas o uso de Coisa para recolher e comunicar volumes potencialmente elevados de dados sobre o ambiente de operação, opcionalmente actuando em resposta a estes dados após processamento.

Mapa de Calor mapa com indicação de padrões através de sobreposição de escalas de cores sobre o mesmo.

Microserviço padrão arquitectural de implementação em que uma única aplicação distribuída é composta por múltiplos serviços independentemente instanciáveis.

Persistência característica de um sistema capaz de armazenar o estado associado a um processo para alem do tempo de vida do mesmo.

Serviço componente computacional autónomo que seja Localizável, Contratual e de Acoplamento Fraco.

Tópico abstracção em Infraestruturas Orientadas a Mensagens, servindo de descritor para publicação ou consumo de um conjunto de mensagens.

1

Introdução

Segundo as Nações Unidas, mais de metade da população mundial reside em áreas urbanas, com uma tendência ascendente de crescimento [1]. Este facto, juntamente com o crescimento global da população no mundo, resulta no crescimento significativo da dimensão e complexidade das cidades, o que representa um desafio significativo nos métodos de gestão utilizados nas cidades [2, 3]. São necessárias novas estratégias para a recolha de informação [4–6] e tomada de decisões na gestão eficiente dos recursos disponíveis [2, 3], promovendo o crescimento do número de cidades que implementam as ditas soluções de Cidades Inteligentes, que apoiam diferentes Verticais de intervenção dos municípios. O termo Cidades Inteligentes é entendido como a sensorização, no contexto da Internet das Coisas (IdC), de áreas urbanas, a fim de, através da análise dos dados recolhidos, melhorar as condições de vida dos cidadãos, bem como a qualidade e sustentabilidade dos serviços prestados.

A sensorização da infraestrutura das cidades resulta na recolha de volumes significativos de dados, possivelmente críticos nos novos esquemas de gestão definidos para as Cidades Inteligentes, que precisam de ser transmitidos de forma atempada, fiável, e eficiente. Como tal, os sensores utilizados na recolha de dados requerem meios para comunicações periódicas e consistentes. Este requisito de comunicação é resolvido pelas Low-Power Wide-Area Network (LPWAN). Uma LPWAN é uma rede de comunicação sem fios otimizada para comunicações de longo alcance, baixo consumo de energia e baixo custo da interface de rede associada. Para ambos existem redes que já são utilizadas para apoiar soluções de IdC em cidades [7]. Na validação da arquitetura

proposta a rede de LoRaWAN presente na cidade de Lisboa, Portugal, é particularmente considerado, uma vez que está disponível num modelo de livre uso e pela existência de trabalho prévio no estudo e desenvolvimento de soluções utilizando esta rede específica, como, por exemplo, na gestão de resíduos urbanos [8].

No contexto das redes LPWAN, num ambiente urbano, cujo objetivo é apoiar soluções Cidades Inteligentes, identificam-se uma série de desafios específicos do seu contexto que requerem uma avaliação regular da cobertura da rede:

- A paisagem urbana pode mudar, por exemplo, pela construção de novos edifícios, o que afeta as características de propagação das redes existentes;
- A expansão contínua da cidade, tanto em termos de população como de área geográfica ocupada, pode aumentar a pressão a que as redes estão sujeitas através de um maior volume de tráfego e número de dispositivos - sendo uma questão identificada em pesquisas anteriores como no caso de [2];

Embora não específico do contexto das Cidades Inteligentes, também se considera especificamente que, na mesma área, podem coexistir múltiplas redes LPWAN, cada uma delas exigindo monitorização adequada. Assim, surge a necessidade de uma solução que possa avaliar a qualidade e cobertura oferecidas por múltiplas redes existentes de forma contínua, a fim de identificar potenciais lacunas na cobertura geográfica, ou, áreas com forte saturação, liderando o esforço de manutenção e investimento futuro na adaptação e expansão das redes existentes.

Existem já soluções que permitem responder a alguns dos desafios identificados, como explorado no Capítulo 2, no entanto falham na resposta a todos. Justifica-se a criação de uma nova plataforma que possa ser utilizada para monitorização continua de múltiplas redes LPWAN, sendo necessário o cuidado planeamento da sua arquitetura de forma a garantir a sua Escalabilidade, Fiabilidade e Adaptabilidade.

A arquitetura, apresentada no Capítulo 3, faz uma utilização dos componentes disponibilizados pelo catálogo Future Internet Software (FIWARE) [9]. O projeto FIWARE foi desenvolvido sob financiamento da Comissão Europeia para criar um catálogo de componentes reutilizáveis *Open Source* que podem ser montados em conjunto, e com outros componentes de terceiros, para construir plataformas de Internet das Coisas (IdC) que apoiam o desenvolvimento de Soluções Inteligentes de uma forma mais rápida, fácil e com reduzidos custos de desenvolvimento. Como parte dos objetivos declarados do projeto FIWARE, foram desenvolvidos vários componentes para integração de fontes de dados externas e componentes de processamento de dados. A sua utilização foi

especificamente considerada na conceção da arquitetura, sendo objetivo a validação da viabilidade do uso de componentes do projeto. Além disso, foi considerada a utilização dos modelos de dados em conformidade com a Application Programming Interface (API) do Next Generation Service Interfaces (NGSI)-v2 [10], permitindo a reutilização e integração das métricas de uma forma padronizada.

1.1 Contribuições

A principal contribuição do presente Projeto Final de Tese é a proposta, implementação e validação da arquitetura para uma nova plataforma baseada em componentes *Open Source* capaz de receber, agregar e apresentar métricas sobre a qualidade da cobertura de redes LPWAN presentes em cidades. Este sistema foi nomeado IoTMapper. A arquitetura proposta apresenta uma solução escalável de ponta-a-ponta, desde a receção de métricas de fontes de dados especificadas até à sua disponibilização aos utilizadores finais, ou, sistemas externos, através de uma Web API. Sendo que a plataforma permanece suficientemente genérica para que se seja possível adicionar suporte a redes LPWAN adicionais com um mínimo esforço de desenvolvimento. Este suporte é alcançado através da adição de novos recetores e interpretadores de mensagens. Foi desenvolvida uma implementação de referência, e respetivos *scripts* de instalação, que podem ser utilizados na construção e desenvolvimento adicional de soluções de monitorização baseadas no IoTMapper. Além disso, é apresentada a validação da implementação, bem como a avaliação qualitativa e quantitativa do desempenho global do sistema da plataforma.

De particular consideração é a avaliação dos componentes do catálogo FIWARE [9] quando colocados sob diferentes volumes de carga, que não tenham sido previamente considerados em investigação anterior. Existem contribuições adicionais relativamente à proposta de novos modelos de dados NGSI-v2 para representação das métricas recebidas e agregadas de uma forma padronizada e reutilizável.

1.2 Contexto do Projeto

1.2.1 Redes LPWAN

As redes de comunicação designadas Low-Power Wide-Area Network (LPWAN) apresentam-se como uma solução de comunicação sem-fios caracterizada pelo longo alcance, baixo

consumo energético e baixo custo do equipamento associado. Devido a estas características encontram enquadramento natural no contexto da Internet das Coisass, em geral, e no contexto das Cidades Inteligentes em particular, por exemplo, no suporte a dispositivos que relatam o estado atual das infra-estruturas urbanas, tais como luzes de rua ou caixotes do lixo, no máximo a cada poucos minutos. São exemplos deste tipo de redes protocolos como LoRaWAN [11, 12], NB-IoT [13, 14], LTE-M e SigFox [15]. A titulo meramente ilustrativo apresenta-se na Tabela 1.1 as principais características dos protocolos atrás mencionados, em ambientes urbanos.

Tabela 1.1: Comparação das principais características de algumas tecnologias LPWAN, adaptado de [15].

Protocolo	LoRaWAN	NB-IoT	Sigfox	LTE-M
Alcance	< 5 Km (Urbano)	< 5 Km (Urbano)	10 Km (Urbano)	10 Km (Urbano)
Frequências	463MHz 868MHz (Europa) 433MHz 915MHz (Austrália e América do Norte)	180 KHz	862-928 MHz	14 MHz
Largura de Banda	125 - 500 KHz	200 KHz	100 Khz	1,4 Mhz
Ritmo Binário Máximo	20-50 Kbps	100-128 Kbps	100-128 Kbps	100-150 kbps
Espectro	Não Licenciado	Licenciado	Não Licenciado	Licenciado

As implementações LPWAN seguem uma topologia de rede em estrela, onde as mensagens enviadas pelos dispositivos são recebidas por equipamentos de rede tipicamente designados como Gateways. De entre estes, redes como o NB-IoT e LTE-M dependem do espectro licenciado no seu funcionamento, em particular das bandas já reservadas para comunicações móveis. Como tal, a sua disponibilidade depende dos Operadores de Comunicações Móveis (OCM), que reutilizam as estações base celulares existentes para fornecer uma largura de banda estreita de 200 kHz. As métricas que podem ser recolhidas, sem a colaboração dos OCM correspondentes, estão limitadas às observáveis pelo equipamento responsável pelo envio das mensagens de medição, como se vê em [16]. Por oposição, redes como o LoRaWAN e Sigfox não dependem de espectro licenciado. Nestas redes, em função do modelo de implementação, podem estar disponíveis métricas adicionais.

De entre as redes apresentadas existe particular interesse no LoRaWAN, como já mencionado, pela presença na cidade de Lisboa, Portugal, num modelo de livre uso e pela existência de trabalho no estudo e desenvolvimento de soluções utilizando esta rede específica [8].

1.2.1.1 Redes LoRaWAN

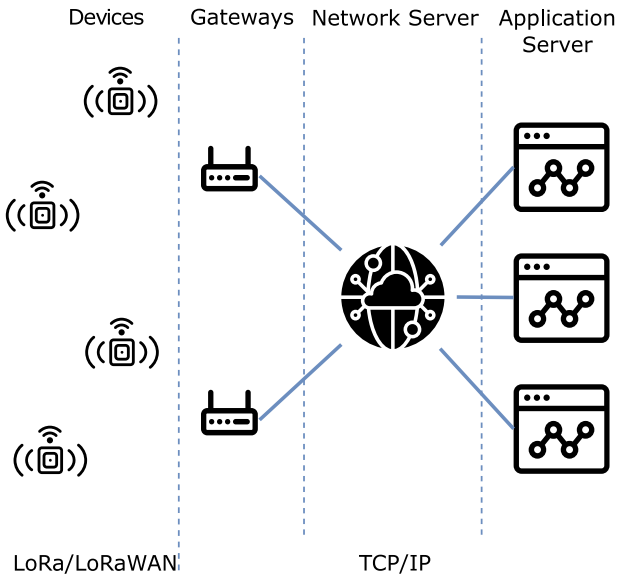


Figura 1.1: Visão de alto nível da arquitetura de uma rede LoRaWAN. Obtido de [8].

Nas implementações LoRaWAN, as camadas de protocolo físico subjacentes, LoRa, são definidas sobre espectro de rádio não licenciado, e estão disponíveis para uso geral, tanto para redes públicas e abertas como para redes privadas para a venda de soluções comerciais. LoRa, conforme normalizado pela LoRa Alliance [11], utiliza frequências de cerca de 868 MHz (para a Europa), 915 MHz (para a América do Norte e Austrália) e 433 MHz em múltiplos países. Na arquitetura LoRaWAN, os Gateways têm uma ligação a um servidor de rede responsável pelo controlo da receção de mensagens e outros problemas de rede, antes de os passar para um servidor de aplicação responsável pelo encaminhamento de mensagens. Neste modelo (visível na Figura 1.1), o LoRaWAN aceita que as mesmas mensagens possam ser recebidas por vários Gateways. Uma vez que todas as receções são válidas, a rede junta-as numa única mensagem, com metadados relativos a todas as entregas. Há múltiplas implementações *Open Source* do LoRaWAN disponíveis. A The Things Network (TTN) oferece uma solução desenvolvida e operada pela comunidade para um servidor LoRaWAN como serviço, que os operadores de Gateways e dispositivos podem utilizar livremente [8, 17–19].

A especificação aberta utilizada por LoRaWAN permite o acesso a métricas significativas sobre a cobertura da rede, se comunicadas pelo Gateway, como por exemplo: RSSI, SNR, Frequência/Canal utilizado, e possivelmente a localização dos Gateways utilizados.

1.2.2 O projeto FIWARE

O FIWARE [9] é um projeto, sob financiamento da Comissão Europeia, para o desenvolvimento de uma estrutura de que oferece um catálogo de componentes reutilizáveis, que possam ser compostos entre si, e com outros componentes de terceiros, para construir plataformas que apoiem o desenvolvimento de Soluções Inteligentes de uma forma mais rápida, fácil e com reduzidos custos de desenvolvimento. A Figura 1.2 apresenta a arquitetura geral do projeto.

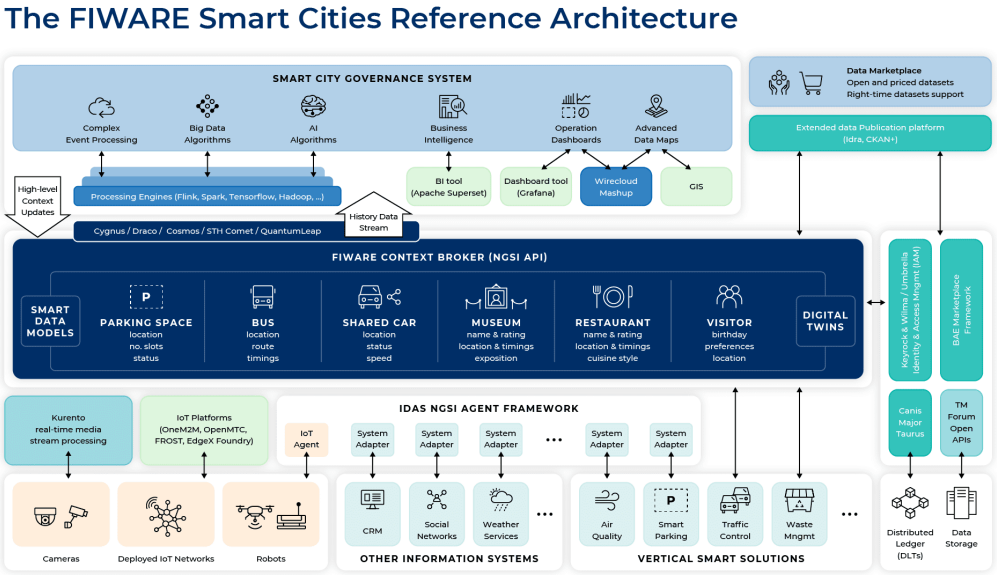


Figura 1.2: Arquitetura Geral de referência do FIWARE para as Cidades Inteligentes. Obtido de [9].

Para a finalidade declarada, o FIWARE define um conjunto de capacitadores genéricos (os Generic Enabler (GE)), componentes de software com responsabilidades bem definidas que oferecem as suas capacidades como serviços. Estes GEs podem ser combinados para comporem soluções funcionais prontas a utilizar. Para permitir a interoperabilidade entre os GEs, todos eles seguem um modelo de gestão de contexto comum, sob a API REST do NGSI-v2 [20], ou, o NGSI-LD [21]. Estas APIs incluem a definição de um conjunto comum e reutilizável, mas expansível, de modelos de dados.

O principal componente habilitador oferecido pelo FIWARE é o Orion Context Broker (OCB), que oferece um serviço de Publicação/Subscrição (ou, Publish/Subscribe em

inglês) para persistência do contexto atual, organizado como um conjunto de entidades. Estes conjuntos devem seguir modelos de dados já definidos sempre que razoável. Este contexto gerido pode ser facilmente partilhado entre os GEs, e componentes externos, permitindo a sua simples interoperabilidade. O mecanismo de subscrição do OCB é assegurado por um sistema assíncrono de "notificações". As notificações são desencadeadas por alterações no estado de uma entidade, resultando em mensagens através de Hypertext Transfer Protocol (HTTP), ou, Message Queuing Telemetry Transport (MQTT) [22], para outro componente previamente identificado por um Uniform Resource Locator (URL). A maior limitação da OCB é que apenas a versão mais atual de cada entidade é persistida na base de dados de suporte. Esta limitação pode ser ultrapassada utilizando um GE adicional, com capacidades de dados históricos, tais como o STH Comet, ou, QuantumLeap.

Os Generic Enablers (GEs) do tipo IoTAgents são um conjunto de GEs inter-relacionados, usando uma biblioteca base comum, para oferecer suporte a diferentes protocolos de comunicação comuns na IdC. Cada IoTAgent é responsável pelo mapeamento das mensagens recebidas em entidades NGSI (seja v2 ou LD) e pela sua inserção no OCB. O IoTAgent-LoRaWAN que oferece integração para LoRaWAN, inclui suporte específico para servidores de aplicação do TTN, sendo uma escolha adequada para suporte a redes como a existente na cidade de Lisboa, Portugal [8].

O catálogo FIWARE oferece, também, GEs para integração com sistemas externos. O Cygnus é um exemplo de tal GE, que pode ser utilizado para exportar dados tanto para armazenamento externo (PostgreSQL , MySQL, MongoDB ou AWS DynamoDB), ou, mecanismos de Publicação/Subscrição assíncrona (como o Apache Kafka [23–25]) que podem ser integrados com estruturas de processamento adicionais.

1.3 Objetivos

O objetivo do trabalho apresentado não é fazer avançar o estado da arte dos modelos de cobertura para redes LPWAN, mas sim aplicar e validar os componentes tecnológicos existentes no desenvolvimento de uma nova solução integrada. Além disso, é de notar que o trabalho não inclui o desenvolvimento dos dispositivos físicos de IdC utilizados na recolha de métricas, mas está centrado na infraestrutura que suporta tais dispositivos.

A arquitetura proposta deve ter em conta a garantia de Escalabilidade, Fiabilidade e Adaptabilidade apropriados a um sistema distribuído responsável pelo contínuo recolha, processamento e apresentação de volumes de dados significativos. É ainda

objetivo principal que a arquitetura proposta faça utilização de componentes do catálogo FIWARE, sendo característica diferenciadora em relação ao trabalho relacionado existente.

1.4 Requisitos da Solução

Atendendo aos objetivos, contexto do problema em análise, consideram-se como requisitos da solução proposta:

- O sistema deve fazer uso de componentes do catálogo FIWARE;
- O sistema deve ser escalável, de tal forma que possa vir a suportar e uma área geográfica composta por múltiplas zonas urbanas, suportar volumes de mensagens na ordem de centenas, ou, milhares de mensagens por segundo em cada zona urbana, e múltiplas redes LPWAN distintas:
  - O sistema deve pelo menos implementar o apoio à LoRaWAN, sobre a The Things Network (TTN);
  - As escolhas de conceção da arquitetura não devem limitar o suporte futuro para redes LPWAN adicionais, implementadas sobre protocolos distintos.
- O sistema deve agregar métricas por alguma área geográfica e agregá-las, obtendo o valor médio para cada métrica recolhida;
- O sistema não deve perder mais de 1% das mensagens recebidas de fontes de dados;
  - É aceitável que como medida de equilíbrio do sistema seja introduzida latência na atualização da métrica agregada, na ordem de alguns segundos. Como limite superior de referência podem ser considerados até 30 segundos;
- O sistema deve disponibilizar as métricas agregadas, tanto para sistemas externos, através de uma API, como para o utilizadores finais através de uma interface gráfica.

1.5 Resultados Adicionais do Trabalho

Como resultado adicional do trabalho desenvolvido, foi elaborado um artigo cientifico expondo a arquitetura proposta e resultados obtidos. O artigo, com o título*IoTMapper:*

A Metrics Aggregation System Architecture in Support of Smart City Solutions [26] foi publicado na revista *Sensors* (ISSN 1424-8220), encontrando-se disponível para acesso.

O código desenvolvido como protótipo de validação da arquitetura é fornecido em repositório publico [27].

1.6 Organização do Documento

O documento do Projeto Final de Tese encontra-se divido nos seguintes capítulos:

- No Capítulo 2 é realizada o estudo do estado da arte, incluindo a análise e comparação de tecnologias candidatas a definição da arquitetura e a apresentação do trabalho relacionado;
- No Capítulo 3 é realizada a exposição da solução arquitetural proposta, sendo identificados os componentes que compõem a arquitetura, suas interações e responsabilidades. São ainda considerados detalhes relativos a implementação de referência;
- No Capítulo 4 é realizada a validação da arquitetura e a avaliação do desempenho da implementação de referência;
- No Capítulo 5 é levada a cabo a reflexão sobre o trabalho desenvolvido assim como do trabalho futuro.

2

Estado da Arte

2.1 Introdução

O presente Capítulo apresenta o estudo prévio realizado sobre o trabalho relacionado e do estado da arte. Em particular são consideradas as tecnologias candidatas para a arquitetura e implementação propostas e questões da conceção arquitetural da solução.

Existe uma grande oferta de tecnologias no mercado que possuem algumas das características necessárias à solução a propor. Sem que nenhuma se imponha como solução completa para o problema em estudo. É necessário analisar cuidadosamente cada uma em função das responsabilidades a desempenhar e o enquadramento da tecnologia na solução.

De forma a evitar a criação de dependências especificas de vendedores deve-se priorizar o estudo e escolha de componentes tecnológicos *Open Source*, sendo que, é possível entre estes encontrar os componentes tecnológicos que garantem os requisitos desejados.

Na definição da arquitetura é necessário identificar soluções tecnológicas para: a recolha e distribuição dos dados; para realizar o processamento dos; e da persistência dos dados.

2.1.1 Organização

O Capítulo está organizado nas secções:

- Secção 2.1.2: o estudo do trabalho prévio existe;

- Secção 2.2: estudo da temática de distribuição de dados entre componentes de um sistema distribuído para processamento de dados;
- Secção 2.3: estudo de soluções para Processamento de Dados em Fluxo, após a sua distribuição;
- Secção 2.4: estudo de soluções para Persistência de dados, antes e após processamento;
- Secção 2.5: estudo de soluções de para interface com utilizadores e sistemas externos;
- Secção 2.6: resumo do estudo realizado.

2.1.2 Trabalho Relacionado

Dado o crescimento significativo nas áreas urbanas verificado nos últimos anos, e que pode ser previsto para as décadas seguintes [1, 2], é possível encontrar um extenso, e crescente, corpo de trabalho sobre soluções baseadas em LPWAN para Cidades Inteligentes, e outras áreas, como a Agricultura Inteligente, destacando a sua importância na literatura. Considerando apenas soluções sobre LoRaWAN, houve 30 artigos publicados em 2016, aumentando para cerca de 400 em 2018 [12]. Embora o corpo de trabalho seja extenso, a grande maioria dos artigos identificados centra-se na comparação de tecnologias LPWAN, na conceção de soluções utilizando tecnologias LPWAN, ou, avaliações sobre a qualidade da LPWAN que simplesmente fazem uma análise pontual das redes existentes. A investigação centrada em soluções para analisar a qualidade da implantação de LPWAN de forma contínua é limitada.

Em termos do trabalho que analisa a qualidade das implantações de LPWAN, em [28], os autores realizam um estudo teórico de uma rede LoRaWAN para o modelo simulado de um ambiente urbano típico. Em [7] os autores relatam um caso de utilização real de uma rede LoRaWAN instalada na cidade de Southampton, no Reino Unido, analisando os dados recolhidos numa última data. Contudo, o estudo de soluções *Open Source* capazes de monitorização contínua de uma rede LPWAN é limitado.

Para [29] os autores realizaram uma campanha de medição a longo prazo para NB-IoT, concluindo que a potência de Reference Signal Receive Power (RSRP) varia mesmo para alvos estacionários. Apresentam, então, um modelo de cadeia de Markov duplamente estocástico a partir de métricas temporais recolhidas. Demonstrando que a abordagem proposta pode ser utilizada para a otimização das infraestruturas LPWAN. Um método

de avaliação da cobertura é apresentado em [30]. Esta avaliação começa com uma campanha de medição em larga escala para NB-IoT, Sigfox, e LoRaWAN. Utilizando os dados recolhidos, os autores propõem um procedimento para identificar o conjunto mínimo de pontos para avaliar a cobertura oferecida por uma rede LPWAN, para a qual não existe informação sobre as estações base. [31] apresenta outra campanha de medição, para Nb-IoT, e disponibiliza o conjunto de dados recolhidos, ao mesmo tempo que identifica o impacto das diferentes decisões de implantação.

Nenhum dos artigos de investigação referenciados propõe uma arquitetura para uma plataforma que apoie ativamente o seu processo de recolha de dados.

Em relação ao trabalho que oferece um mecanismo de monitorização, em [16] os autores apresentam um sistema incorporado de análise de qualidade local em redes NB-IoT, baseado na extração e processamento de mensagens de sinalização de rede, mas sem a definição de infraestrutura de apoio para recolha e agregação de métricas. [32] descreve o NBViewer, um demonstrador para um conjunto de ferramentas de software que, localmente, recolhe e analisa dados para um único dispositivo NB-IoT.

Nenhuma dos mecanismos apresentados propõe uma arquitetura para uma plataforma capaz de recolher de dados de múltiplos dispositivos em simultâneo.

Dentro de soluções ditas *Open Source*, podem ser identificadas duas soluções já disponíveis: TTNMapper [33, 34] e HeliumMapper [35]. Outras soluções, comerciais, também podem estar disponíveis, mas devido à sua natureza *Closed Source* não foi possível considera-las adequadamente [36].

2.1.2.1 TTN Mapper

O TTNMapper é uma solução de mapeamento LoRaWAN, originalmente nas versões 1 e 2, específica para o servidor de aplicação da comunidade The Things Network (TTN) [17, 18]. Para a versão 3 [34] foi adicionado o suporte para outros servidores LoRaWAN, nomeadamente ChirpStack [37] e Helium [38]. A Figura 2.1 apresenta a interface grafica desenvolvida pelo TTNMapper.

A arquitetura analisada para o TTN Mapper tem como base a documentação disponível para a versão 2 e, quando disponível, são indicadas as alterações correspondentes a versão 3.

O TTN Mapper optou por uma arquitectura orientada a Microserviços [39] com comunicação entre componentes garantida através da utilização de filas de mensagens Advanced Message Queuing Protocol (AMQP), implementadas com RabbitMQ [40]. A

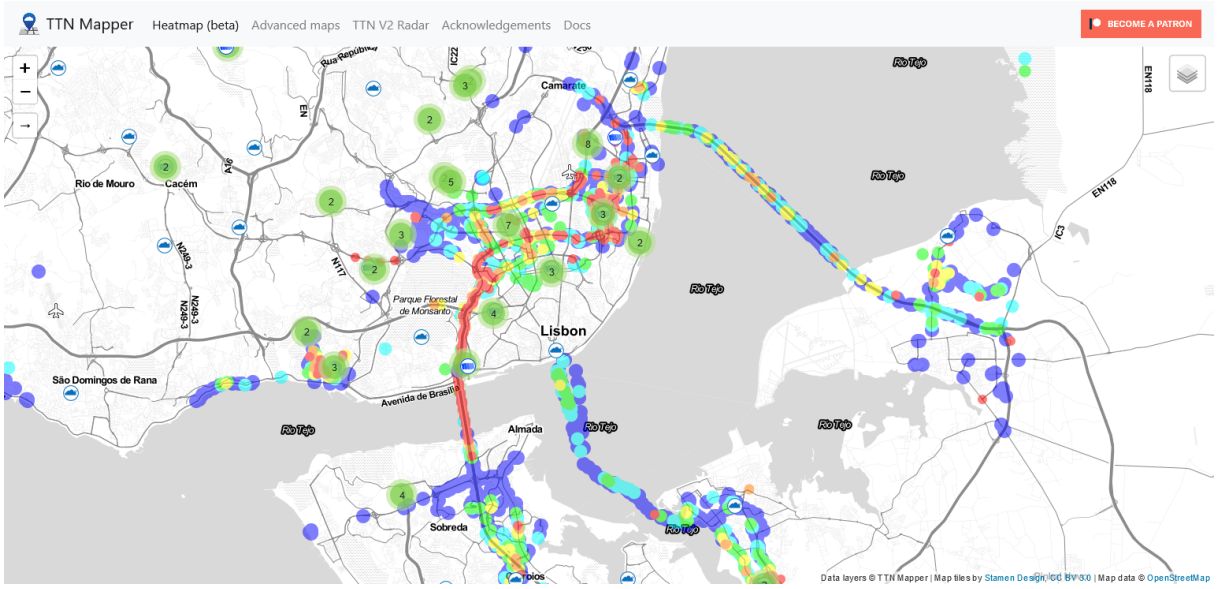


Figura 2.1: Exemplo de utilização do TTN Mapper, centrado em Lisboa a data 11/11/2021.

receção das métricas é realizada através de HTTP, com conteúdo JSON, recorrendo a integração com os servidores TTN, que encaminha os dados recolhidos pelos dispositivos que participam voluntariamente. Os componentes executam sequencialmente etapas de processamento, agregação e persistência de dados numa base de dados PostgresSQL. A agregação executada junta as medições em células geográficas. Os componentes principais são [34]:

- **ingress-api:** ponto de entrada central e obrigatório de todos os pacotes, procedendo a conversão dos dados a um formato comum antes de inserção na fila de mensagens *new_packets*;
- **websockets-live:** implementa fluxos em tempo real para actualizar a camada de apresentação;
- **postgres-insert-raw:** insere os pacotes normalizados numa base de dados PostgreSQL, antes de as reproduzir na fila *inserted_data* para processamento;
- **gateway-update:** recebe todos os pacotes de forma a identificar a última vez que a presença de um dos Gateways foi verificada, registando na base de dados PostgreSQL. Periodicamente utiliza a lista de Gateways da TTN para identificar alterações na localização dos Gateways. Alterações superiores a 100 metros resultam numa mensagem na fila *gateway_moved* para impedir a apresentação de métricas realizadas na localização antiga;

- **postgres-insert-gridcell:** sendo que não existe necessidade de que as visualizações sejam atualizadas em tempo real, de forma a reduzir a transferência de dados, as métricas são agregadas em células *Slippy Map Tile* com um zoom associado de nível 19. Cada agregação associa uma área geográfica a um Gateways. Para o efeito são subscritas as filas *inserted_data* e *gateway_moved*. Alterações na localização implicam apagar a agregação anterior considerando apenas mensagens na nova localização;

- **gateway-channels:** através da subscrição da fila *inserted_data* são obtidas contagens para cada frequência em cada Gateway. Esta informação estatística pode permitir identificar os canais e planos de frequências de cada Gateway;

- **ttnmapper-web-v2:** interface de apresentação do dados recolhidos, na forma de mapas, incluindo Mapas de Calor;

- **ttnmapper-api-v2:** interface entre a camada de apresentação gráfica e persistência de dados após processamento;

A separação do processamento a realizar em várias fases semi-independentes, interligadas por filas de mensagens, proporciona a escalabilidade do sistema.

Considere-se o modelo de dados apresentado no Apêndice A. Pode-se verificar que a rede TTN adiciona a cada pacote metadados relativos às transmissões de rádio efetuadas e os Gateways envolvidos na receção da mensagem. A mensagem pode, quando a informação é pública, possuir a localização dos Gateways, no entanto, a localização do equipamento IoT envolvido na mediação deve ser incluída no campo de *decoded_payload*.

2.1.2.2 Helium Mapper

À semelhança do TTNMapper oprojecto Helium apresenta, também, uma plataforma de visualização da qualidade associada a sua rede LoRaWAN, havendo menos informação disponível sobre a sua arquitetura interna.

As métricas são, novamente, recebidas através de HTTP, através de integração direta com aplicações Helium que forneçam relatórios sobre os seus dispositivos voluntariamente. A receção dos relatórios é seguida pela agregação das métricas em células geográficas H3, originalmente desenvolvidas pela Uber [41], sendo persistidas numa base de dados PostgresSQL. As células H3 são um índice hierárquico que agrupa as coordenadas numa grelha contínua de blocos de área de forma hexagonal com uma identidade única. A sua interface com o utilizador é apresentada na Figura 2.2.

Figura 2.2: Exemplo de utilização do Helium Mapper, centrado em Lisboa a data de 11/11/2021.

O sistema recolhe, como métricas, de entre os dados identificados como expostos aos utilizadores:

- **RSSI:** Received Signal Strength Indication;
- **SNR:** Signal to Noise Ratio;
- **Redundancy:** número total de Gateways escutados na celulda;
- **Distance:** distancia aproximada entre o Gateway e o equipamento.

2.1.2.3 Conclusão do Trabalho Relacionado

No melhor conhecimento do autor, não há nenhum trabalho anterior que utilize componentes Future Internet Software (FIWARE) para recolher métricas sobre as próprias redes LPWAN, em oposição às métricas sobre os dispositivos IoT ou o ambiente monitorizado. Também não existe um sistema *Open Source* disponível que processe métricas em múltiplas redes LPWAN de uma forma próxima da realidade em tempo real. Neste contexto, considera-se que existe a necessidade de conceber e validar uma nova solução *Open Source*, que considera desde o inicio a existência de múltiplas redes LPWAN, e avalia a viabilidade da utilização dos componentes FIWARE existentes para a gestão do contexto e integração com fontes de dados externas.

2.2 Distribuição de Dados

Os mecanismos de distribuição e transferência de dados são um aspecto central nas arquitecturas de sistemas distribuídos, sendo a sua escolha limitadora na definição das interacções entre componentes, no balanceamento de carga, na Fiabilidade e na Escalabilidade do sistema. Kleppmann [42] considera que os principais mecanismos, no contexto de sistemas de processamento de dados, para distribuição dos dados podem ser classificados nas categorias:

- Distribuição via Bases de Dados;
- Distribuição via Serviços (por exemplo, através de REpresentational State Transfer (REST), ou, Remote Procedure Call (RPC));
- Distribuição via Passagem Assíncrona de Mensagens;

Consideram-se principalmente relevantes, na definição de sistemas escaláveis ativamente a receber dados para processamento, os casos de distribuição com serviços e passagem assíncrona de mensagens. O uso de bases de dados na distribuição como parte do processamento requere associação a outros mecanismos para notificação dos componentes da existência de dados a processar, ou, tentativas recorrentes de leitura, sendo principalmente relevante para o Armazenamento de Dados (Secção 2.4).

2.2.1 Distribuição via Serviços

Considera-se, como Distribuição via Serviços, quando a distribuição é realizada via elementos, designados Serviços, que comunicam entre si, sendo a transferência dos dados realizada como parte desta comunicação. Um Serviço é um elemento computacional autónomo que seja Localizável, Contratual e de Acoplamento Fraco, a sua comunicação deve ser realizada através de uma interface previamente definida, e do conhecimento dos restantes elementos, sem dependência de características externas a referida interface. Com o intuito de suportar a construção de Serviços com as características descritas existem múltiplos paradigmas e especificações concorrentes que, geralmente, expõem um modelo de comunicação Cliente/Servidor síncrono.

Um caso de uso comum [42] deste tipo de distribuição é a definição de Web Application Programming Interface (API)s para Aplicações Web. Adicionalmente, a Distribuição via Serviços é também frequentemente utilizada na comunicação interna entre elementos de Sistemas Distribuídos. Esta metodologia de distribuição conduz a um estilo arquitetural

tipicamente conhecido como Service Oriented Architecture (SOA), que tem vindo a ser refinado e rebatizado como Arquitetura de Microserviços [39, 42].

As infraestruturas existentes, em termos do tipo de serviço que originam, podem ser globalmente divididas em duas categorias [42, 43], os Serviços Web e os Serviços RPC. Note-se, no entanto, que em função da tecnologia específica existe significativa sobreposição das categorias, resultante de vários tecnologias para Serviços Web implementarem funcionalidades no estilo RPC, e de vários tecnologias para Serviços RPC recorrem ao protocolo HTTP.

2.2.1.1 Serviços Web

Entende-se por Serviços Web todos aqueles que utilizam o HTTP como protocolo subjacente, naturalmente sendo influenciados pelo seu uso nas escolhas arquiteturais. Nesta categoria considera-se principalmente o REST e SOAP, que possuem visões opostas na arquitetura de serviços. Estando significativamente integrados no ecossistema actual de Aplicações Web, beneficiam de mecanismos presentes na maioria dos navegadores, por exemplo, o controlo local de *cache*.

O REST não é um protocolo específico, mas sim um padrão arquitetural na definição de APIs, em particular no contexto de Aplicações Web, com origem na dissertação doutoral de Roy Fielding [44]. Neste padrão assume-se o uso de HTTP como protocolo de transferência de dados, definida como leitura, criação, alteração e eliminação de recursos identificados por URLs, sem uso de estado para além do contido nos pedidos e respostas. Os serviços e aplicações que seguem este padrão são comumente referidos como RESTful. Sendo apenas uma linha orientadora da arquitetura aplicacional, os detalhes de implementação e operação são específicos de caso de aplicação, com liberdade significativa sobre padrões de utilização e conteúdos transferidos. Para agilizar o seu desenvolvimento, com respeito as boas práticas identificadas, existem ainda múltiplas infraestruturas para aplicação, uniformização e extensão do comportamento REST, por exemplo, o OpenAPI [45].

Por oposição o SOAP é um protocolo descrito em WSDL, uma linguagem baseada em XML, pertencente a um família complexa de protocolos que estendem o seu comportamento (a *Web Services Framework*, conhecida com WS-*), especificando de forma rigorosa a interface exposta pelos serviços. A sua significativa complexidade, aliada a falta de interoperabilidade efetiva entre diferentes implementações tem levado a sua perda de popularidade [42]. Apesar de no presente documento ser apresentada como uma tecnologia para Serviços Web, por ser o seu contexto de implementação principal, a

sua especificação tenta ser agnóstica ao protocolo subjacente e pode ser utilizado como RPC.

2.2.1.2 Serviços RPC

O modelo dos Serviços Remote Procedure Call (RPC) tenta abstrair pedidos a um serviço remoto com o intuito que este seja apresentado ao programador como se tratasse de uma função, ou, um método na sua linguagem de programação, dentro de um mesmo processo. Está abstração é referida como *transparência da localização* [42]. A principal vantagem, e objetivo, é a oferta de transparência a natureza distribuída do processo de distribuição dos dados, apesar de com sucesso limitado. Por oposição ao uso de REST são uma metodologia com menos flexibilidade na definição da interface, e cujas características, como o formato dos dados, são dependentes da tecnologia específica.

As primeiras gerações de tecnologias para serviços RPC, que surgiram em contextos empresariais, sofrem de limitações significativas que impediram a sua adopção em escala. Por exemplo, o Java RMI é limitado a aplicações Java, e o CORBA é excessivamente complexo, sem compatibilidade com versões passadas, ou, futuras de um mesmo serviço [42]. Como tal o desenvolvimento recente nesta área tem privilegiado o desenvolvimento e popularização de tecnologias agnósticas a linguagem de desenvolvimento, suporte a compatibilidade entre diferentes versões de um serviço e que, mesmo quando recorrem a formatos próprios para os dados, utilizam o HTTP. São principais exemplos o gRPC [46], Apache Thrift [47] e Finagle [48]. Sendo que todas são claramente Serviços RPC adotaram um posição de maior flexibilidade relativamente a transparência da distribuição, suportando diferentes mecanismos para comunicações assíncronas.

2.2.2 Distribuição via Passagem Assíncrona de Mensagens

Considera-se, como Distribuição via Passagem Assíncrona de Mensagens, quando a distribuição é suportada sobre uma Infraestrutura Orientada a Mensagens, isto é, uma camada intermédia que recebe mensagens de fontes de dados (Publicadores) e é responsável pela garantia da sua entrega aos componentes de processamento (Consumidores). A camada intermédia é tipicamente composta por componentes que podem ser designados como Intermediários (ou, *Brokers* em Inglês). Uma abstração comum nestas infraestruturas é o conceito de Tópico, servindo de descritor para publicação ou consumo de um conjunto de mensagens entre múltiplos Publicadores e Consumidores, sendo comum a designação deste padrão como Publicador/Consumidor (ou,

Publish/Subcriber em Inglês) e como Subscrição o mecanismo para indicação que um determinado Consumidor pretende receber as mensagens de um ou mais Tópicos.

Comparativamente ao uso de Distribuição via Serviços, as infraestruturas que implementam o padrão Publicador/Consumidor apresentam, tipicamente, várias vantagens [42]:

- Atuam como fila de espera, permitindo a receção e registo de mensagens mesmo se os componentes de processamento de dados não estejam disponíveis, ou, estejam sobrecarregados, dentro de limites do sistema, aumentando a Fiabilidade do mesmo;
- Oferecem mecanismos de reentrega automática da mensagem;
- Permitem o desacoplamento lógico e temporal dos componentes;
- Entregas múltiplas da mesma mensagem, ou, múltiplos consumidores possíveis para uma entrega única.

As características enumeradas estão associadas a natureza meramente unidirecional da comunicação, requerendo a criação de novos canais separados para obtenção de respostas, como tal é um padrão arquitetural adequando para situações onde se pretende distribuição de dados sem que as fontes de dados obtenham uma resposta. Existem vários estudos e comparações [49–51] do estado da arte neste tipo de distribuição, sendo que se podem identificar, como exemplos tecnológicos populares a considerar, o Apache Kafka, MQTT, NATS, OpenDDS, RabbitMQ (AMQP), Redis (através do seu mecanismo Pub/Sub) e ZeroMQ.

2.2.2.1 Apache Kafka

O Apache Kafka [23–25, 52] é uma infraestrutura distribuída para processamento de fluxos de dados originalmente desenvolvida para uso interno na Linkedin como parte da sua arquitetura de processamento de dados. O acesso aos dados é fornecida por uma interface Publicador/Consumidor. O funcionamento do Apache Kafka é dependente do funcionamento do Apache ZooKeeper [53] para gestão de configurações e controlo da coordenação. O limite de dimensão das mensagens é de 2 GB.

Adicionalmente, nota-se que o uso do Apache Kafka permite a integração directa com o Kafka Connect [54] e Kafka Streams [55, 56]. O Kafka Connect permite a ligação directa dos fluxos de mensagens com outros sistemas de dados, tanto como pontos

de entrada como de saída, assim como a aplicação de transformação simples sobre os dados. O Kafka Streams é uma biblioteca que facilita a construção de aplicações para processamento de dados cujas fontes e origens são Tópicos do Apache Kafka.

2.2.2.2 MQTT

O Message Queuing Telemetry Transport (MQTT) [22] é um protocolo de Publicador/-Consumidor desenvolvido com o objetivo ser utilizado em cenários de Internet das Coisas minimizando, então, o uso de recursos. Sendo um protocolo aberto existem múltiplas implementações independentes, como, Mosquitto [57] e HiveMQ [58], compatíveis entre si. Sendo que usa uma definição distinta de configuração de Quality of Service (QoS) do normal, descrevendo as garantias de entrega das mensagens em vez de opções de priorização de entrega, oferece três níveis de QoS (0, 1 e 2) que podem ser configurados, independentemente, do lado do Publicador ou do Consumidor: pelo menos uma vez sem confirmação, no máximo uma vez com confirmação e exactamente uma vez com confirmação. Oferece ainda funcionalidades como sessões persistentes, retendo mensagens para entrega a Consumidores desconectados, e mensagens de *último testamento*, a entregar em caso de desconexão inesperada. O limite de dimensão das mensagens é de 268 MB.

2.2.2.3 NATS

O NATS [59] oferece uma interface Publicador/Consumidor através de conjunção é gestão de múltiplas ligações ponto-a-ponto TCP. Oferece, também, subscrições persistentes e armazenamento persistente de mensagens em memória, ficheiro ou base de dados. Os conjuntos de Intermediários necessitam de estar ligados diretamente a todos os membros do mesmo grupo para replicação de mensagens. Apresenta-se como uma solução unificadora de ambientes de nuvem e Internet das Coisas numa única infraestrutura. Desde a versão 2.2 oferece uma interface para interligação com o MQTT versão 3.1.1. O limite de dimensão das mensagens é de 1 MB.

2.2.2.4 OpenDDS

O OpenDDS [60] é especificamente uma implementação da especificação DDS para sistemas em tempo real do Object Management Group, na linguagem C++. O OpenDDS oferece várias Application Programming Interface (API) com propósitos distintos, incluindo elevada flexibilidade na definição do QoS, sofrendo, no entanto, de alta complexidade. De particular interesse é o Data-Centric Publish-Subscribe (DCPS), que expõe

uma interface Publicador/Consumidor de acordo com os tipos dos dados associados as mensagens. Cada Tópico corresponde a um único tipo de dados. A camada de transporte pode ser configurada para uso de vários protocolos, como TCP ou UDP. Para poderem pariticipar na comunicação os Produtores e consumidores devem obter informação da localização na rede dos restantes participantes, por configuração estática, ou, através de um dos dois serviços de descoberta suportados: DCPS Information Repository (DCPSInfoRepo) e descoberta por Real-time Publish-Subscribe (RTPS). O funcionamento do OpenDDS é dependente do Adaptive Communication Environment (ACE), um componente infraestrutura de programação de redes. Em particular o suporte do DCPSInfoRepo, quando utilizado, é dependente do The ACE Object Request Broker (TAO).

2.2.2.5 RabbitMQ (AMQP)

O RabbitMQ [40] não é, por oposição aos restantes casos descritos, um protocolo, mas sim uma implementação específica, sendo que, originalmente, implementava apenas o AMQP. Suporta agora o interface com outros protocolos (em particular, MQTT, STOMP ou passagem de pedidos HTTP), mas é sempre o AMQP que usa internamente. Como tal, usa-se como implementação de referência do AMQP.

2.2.2.6 Redis

O Redis [61] é uma base de dados em memória, suportando, por exemplo, strings, conjuntos e listas, entre outros. O Redis suporta um mecanismo de sincronização assíncrona entre réplicas, que, combinado com Persistência através de escritas periódicas no sistema subjacente de armazenamento, oferece garantias de Disponibilidade e de Fiabilidade. Adicionalmente expõe uma interface Publicador/Consumidor útil para o objetivo de distribuição assíncrona de dados. As mensagens estão limitadas a 512 Mb.

2.2.2.7 ZeroMQ

O ZeroMQ [62] é uma infraestrutura de comunicação por mensagens desenhada para se comportar como uma infraestrutura de concorrência. Possui flexibilidade em termos de padrões de utilização, suportando tanto comunicação Publicador/Consumidor como Pedido/Resposta, assim como de mecanismos de transporte, como Inter-Process Communication (IPC), TCP e multicast IP. Sendo o foco do ZeroMQ a redução máxima do uso de recurso, o protocolo abdica de depender necessariamente de intermediários normalmente necessários para padrões Publicador/Consumidor, abdica também de

oferecer características como persistência de mensagens, replicação das mensagens, ordenação entre publicadores distintos e mecanismos específicos para balanceamento de carga. Note-se que podem ser construídas componentes ZeroMQ que desempenham funções semelhantes as de um intermediado.

2.2.3 Comparação de Mecanismos

De entre os mecanismos de passagem assíncrona de mensagens pode-se ter em consideração o estudo realizado em [49] e resumido na Tabela 2.1.

Neste estudo identificam-se dois cenários de utilização relevantes, com ligações de rede estáveis:

- BFD (Blue Force Data): enviou 2880 mensagens de 128 bytes num período de 20 minutos;
- SD (Sensor): enviou 21 mensagens de 128 Kbytes num período de 20 minutos.

Tabela 2.1: Comparação de tecnologias de passagem assíncrona de mensagens, adaptado de [49].

Protocolo	Entrega 10 Segundos [%]		Entrega Total [%]		Latência [segundos]		Largura de Banda [KBps]
Cenário	BFD	SD	BFD	SD	BFD	SD	N/A
Kafka	99.2	90.48	99.6	100	934	5922	6820.7
MQTT	99.33	85.71	100	100	566	7887	8103.2
NATS	100	95.24	100	95.24	375	5505	2326.4
OpenDDS	99.73	0	90.85	85.71	1515	34381	1371.2
RabbitMQ (AMQP)	97.73	33.33	100	100	2164	14537	10675
Redis	99.98	100	99.99	100	419	5520	2719.1
ZeroMQ-NORM	99.92	100	99.92	100	39	3255	421.38
Jgroups	98.64	100	98.87	100	82	21811	286.44

Em ambos os cenários o objetivo é a entrega de cada mensagem a um nó em particular de entre um conjunto de 24 nós de processamento. Considerando que no caso em análise, processamento assíncrono de métricas recolhidas sobre redes LPWAN, é prioritário a garantia do processamento eventual de um número máximo de mensagens, pode-se fixar como um limiar inferior para consideração da tecnologia um nível de

entrega de 99% das mensagens tanto para BFD e SD para *Entrega Total* - isto é, sem limite de tempo para entrega da mensagem. Correspondem a este requerimento o Kafka, MQTT, RabbitMQ (AMQP), Redis e ZeroMQ-NORM, sendo que de entre estes o RabbitMQ (AMQP) pode ser rapidamente excluído por apresentar os piores resultados em termos de latência e largura de banda. O ZeroMQ-NORM, que apresenta os melhores práticos no estudo, falha na oferta de características como persistência de mensagens, replicação das mensagens, ordenação entre publicadores distintos e mecanismos específicos para balanceamento de carga sem investimento significativo no desenvolvimento de componentes adicionais. De forma semelhante, apesar de oferecer subscrições persistentes, não integra no seu protocolo mecanismos de balanceamento de carga e replicação entre nós, sem uso de mecanismos específicos de vendedores, ou, recorrendo a novos componentes. O Redis também não oferece quaisquer garantias de entrega, ou persistência, no seu mecanismo de Publicador/Consumidor.

Conclui-se que, apesar do aumento relativo de latência e uso de largura de banda, o Apache Kafka apresenta garantias, configuráveis, ao nível da entrega das mensagens através da integração direta no seu protocolo de mecanismos de replicação das mensagens entre nós. Adicionalmente o uso do Kafka Streams permite estender as garantias de entrega a garantias de processamento, com persistência de resultados intermédios para processamento futuro e redistribuição automática de trabalho perante falhas de nós.

É fundamental realçar, ainda, que o mecanismo de notificações implementado pelo Orion Context Broker (OCB), e apresentado na Secção 1.2.2, é também um mecanismo assíncrono de Publicador/Consumidor com uma fila de mensagens interna. No entanto, não existe investigação anterior, nem os autores do OCB afirmam, que possuam o mesmo nível de tolerância a falha como as infraestruturas analisadas, para além, da transmissão de mensagens para um *broker* de MQTT. Como tal, a escolha de um mecanismo como o Apache Kafka pode oferecer complementaridade natural ao uso de notificações pelo OCB.

2.3 Processamento de Dados

Em complementaridade do estudo da Distribuição dos Dados, cujo processamento é desejado, é necessário o estudo dos candidatos tecnológicos a utilizar no referido processamento. É importante que as escolhas tecnologias não sejam limitadoras na garantia da Escalabilidade e Fiabilidade. Em particular que permitam a distribuição eficaz do trabalho e tolerância a falhas com mínima perda de trabalho já realizado. É

também uma característica desejável a existência da garantia que as métricas recebidas sejam processadas exatamente uma única vez, de forma a evitar a criação de agregações de métricas com repetição de valores, ou, eventual perda de trabalho de campo na recolha das métricas. Reforça-se, novamente, a importância em focar o estudo na escolha de soluções tecnológicas *Open Source*, de forma a evitar a criação de dependências sobre vendedores específicos.

2.3.1 Processamento em Fluxo

Pode-se sugerir, como definição do processamento em fluxo, que seja: o processamento, em tempo perto do real, de dados continuamente produzidos por um fluxo de eventos. Atendendo que o problema apresentado no Capítulo 1 consiste no processamento contínuo de métricas, recebidas ao longo do tempo de forma irregular e imprevisível, de forma a obter agregações das mesmas para apresentação e análise futura, o problema enquadra-se naturalmente na definição do processamento em fluxo (*Stream processing*).

Na definição de um sistema capaz de realizar processamento em fluxo pode-se optar pela definição de componentes especializados ao caso de utilização e geridos diretamente, realizando a distribuição dos dados entre os componentes, sendo a aproximação descrita a apresentada anteriormente pelo TTNMapper [33, 34]. Esta solução pode resultar num maior controlo sobre o processamento realizado, sendo de particular interesse quando o processamento seja razoavelmente simples. No entanto apresenta desafios significativos no desenvolvimento e manutenção dos sistemas. Em particular é necessário, no processo de planeamento e desenvolvimento, a definição de mecanismos que garantam a escalabilidade e fiabilidade desejadas e, perante a evolução dos requisitos aplicacionais, existe esforço adicional na alteração do sistema sem perda das garantias oferecidas. Sendo o processamento em fluxo de grande interesse no cenário atual de crescimento de volumes de dados a processar existem, já, múltiplas infraestruturas que ofereçam as garantias desejadas e mecanismos para, continuamente, adicionar e modificar as funcionalidades de um sistema [63–66]. A maioria das infraestruturas de processo em fluxo tem em comum a necessidade de recorrem a componentes externos para desempenhar as suas funções, nomeadamente, de uma camada de distribuição dos dados e uma camada de persistência subjacente.

Apresentam-se, seguidamente, um conjunto de infraestruturas de processamento em fluxo candidatas a definição da arquitetura.

2.3.1.1 Kafka Streams

O Kafka Streams [24, 55, 65] constrói diretamente sobre a infraestrutura do Apache Kafka, já apresentada, oferecendo forte integração com o seu funcionamento. O Kafka Streams é uma biblioteca de cliente permitindo a adição do processamento em fluxo a aplicações já existentes, ou, criação de novas aplicações responsáveis por supervisionar o processamento. Na integração com o Apache Kafka recorre-se aos tópicos para distribuição dos dados a processar, os seus resultados, finais ou intermédios, e registos de alterações para recuperação do estado perante falhas. Adicionalmente à abstração do fluxo (*KStream*) de dados e operações comuns a realizar sobre este oferece mecanismos variados para suporte ao processamento, nomeadamente: o conceito de tabela (*KTable*), visão do estado mais recente para uma chave num fluxo, de estados armazenados (*State Store*), suportados pelo uso local da RocksDB [67], para uso de estados anteriores no processamento, e de pesquisas interativas (*Interactive Queries*), para acesso externo ao estado armazenado do Kafka Streams. O Kafka Streams oferece duas APIs distintas, o *Streams DSL*, com sintaxe simples para rápido desenvolvimento, e a *Processor API*, com sintaxe mais complexa mas maior nível de controlo de detalhe.

Uma vantagem imediatamente evidente no uso do Kafka Stream, em relação a outras infraestruturas de processamento em fluxo, é a possível eliminação de componentes adicionais para distribuição e persistência dos dados, reduzindo o esforço de desenvolvimento e integração, com a desvantagem de criação de forte dependência no uso do Apache Kafka.

2.3.1.2 Apache Flink

O Apache Flink [65, 68] é uma infraestrutura de processamento em fluxo que suporta tanto fluxos limitados como fluxo ilimitados de dados para processamento continuo. Oferece a garantia que o processamento de uma determinada mensagem é executado uma única vez, mesmo perante falhas, através da escrita assíncrona de pontos de controlo (*Checkpoints*) numa fonte externa de persistência, em particular um sistema de ficheiros, possivelmente distribuído. Igualmente, requer integração com infraestruturas externas para distribuição de dados de forma a realizar a obtenção de dados e escrita de resultados, apresentado, para o efeito, múltiplos conectores. O processamento a realizar é levado acabo diretamente no Aglomerado (*Cluster*) de máquinas atribuídas ao Apache Flink, existindo para o efeito um gestor de recursos (*JobManager*) e um conjunto de gestores de tarefas (*TaskManager*). O Apache Flink oferece múltiplas API que abstraem operaçoes comuns no processamento em Fluxo, com níveis diferentes de integração e

detalhe, permitindo, no desenvolvimento das aplicações, diferentes equilíbrios sobre o controlo sobre processamento e redução do trabalho de desenvolvimento.

2.3.1.3 Apache Spark

Sendo uma infraestrutura desenvolvida originalmente em 2012, em resposta as limitações identificadas no paradigma do MapReduce, o Apache Spark [64, 65, 69], originalmente, possui forte dependência no funcionamento do Apache Hadoop, sendo gradualmente adaptado para execução independente. No processamento em fluxo são oferecidas duas API para definição do processamento, *Spark Streaming* e *Structured Streaming*, juntamente com outras API para abstração de operações comuns, sendo todo o trabalho realizado sobre a mesma arquitetura subjacente. Note-se que, por oposição a *Structured Streaming* e outras infraestruturas de processamento em fluxo comuns o *Spark Streaming* não oferece operações com estado definidas pelo programador [65]. O aglomerado de máquinas atribuídas ao Apache Spark é composto por nós de trabalho (*Workers Nodes*) que, geridos por um componente externo, por exemplo, YARN ou Kubernetes, dividem o trabalho indicado por uma aplicação responsável pela gestão do contexto (*Driver Program*). O principal conceito por detrás da distribuição do trabalho e tolerância falhas é o de Resilient Distributed Dataset (RDD), requerendo uma camada subjacente de persistência, para o qual exista compatibilidade com o Apache Hadoop, preferencialmente um sistema distribuído de Ficheiros com forte garantias de tolerância, por exemplo, HDFS ou HBase. Utilizando o RDD é possível armazenar os dados de entrada, resultados determinísticos e sequências de operações realizadas durante o trabalho já desenvolvido, que permitem recuperar e reconstruir o estado, perante falhas dos Workers Nodes. A semelhança de outras infraestruturas de processamento em fluxo o Apache Spark requer integração como uma infraestrutura externa para distribuição de dados.

2.3.2 Comparação de Mecanismos

Considerando-se as infraestruturas selecionadas como candidatas e os objetivos anunciados para o processamento em fluxo, existem dois principais aspetos relevantes para comparação. A capacidade de processamento das infraestruturas e a sua tolerância a falhas. É interessante notar que todas as infraestruturas candidatas apresentadas recorrem ao Apache Zookeeper na coordenação do trabalho.

Na capacidade de processamento Van Dongen e Van den Poel [65] comparam as duas

variantes descritas para Apache Spark com o Apache Flink e Kafka Streams num ambiente que simula operações típicas de Extract Transform Load (ETL). As diferentes soluções apresentam equilíbrios opostos entre latência e volume de processamento. O Apache Spark, em particular o *Structured Streaming*, baseando-se no agrupamento interno mensagens para processamento (*Micro-Batching*) oferece maior volume de processamento, em termos do máximo de eventos processados por segundo, mas sofre de elevada latência. O Apache Spark e Kafka Streams, sendo orientadas a eventos, permitem latências inferiores, mas o volume de dados processados é significativamente inferior. Quando as infraestruturas são inicializadas pela primeira vez, sendo confrontadas com volumes largos de eventos iniciais, o Kafka Streams e Apache Spark tem a menor latência na apresentação dos primeiros resultados, mas maior latência no processamento de todos os eventos. Havendo a observação oposta para o Apache Spark. Quando os eventos ocorrem de forma concentrada periodicamente o Apache Flink apresenta os melhores resultados tanto em operações sem e com estado local, mas otimizações no uso de estado do Kafka Streams permitem resultados comparáveis. Em termos de recursos computacionais nota-se que o menor uso de CPU é associado ao Apache Spark, ao passo que o Apache Flink usa até o triplo da memória dos outros dois. Os autores concluem que, desejando-se uma solução equilibrada em termos de latência e volume de dados, a solução indicada seria o Apache Flink, sendo o Kafka Streams uma alternativa interessante quando os dados a processar já se encontrem no Apache Kafka.

Na tolerância a falhas, e em continuação do seu trabalho anterior, Van Dongen e Van den Poel [64] compararam o comportamento das duas variantes do Apache Spark com o Apache Flink e Kafka Streams, em diferentes cenários de falhas de uma infraestrutura de processamento semelhante a de [65]. Na maioria dos cenários o Apache Flink requer um recomeço de todo o trabalho implicando tempos de indisponibilidade e de recuperação de 50 e 53 segundos. O Apache Spark, em resultado da divisão de trabalho em tarefas, conseguem recuperar em 20-30 segundos, após apenas 4-6 segundos de indisponibilidade. O Kafka Streams necessita de apenas 2 segundos para se tornar novamente disponível, e de apenas 21 segundos para recuperação. Note-se que para o Apache Flink todo o processamento é interrompido no período de indisponibilidade, tal não acontece para trabalho já distribuídas nas restantes infraestruturas. Importante na tolerância a falhas é a maximização da não repetição e não perda de eventos, sendo preferêncial o uso de semânticas para entregas *Exactamente uma vez*, por oposição a *Pelo menos uma vez* e *No máximo uma vez*. Note-se que as garantias semânticas são dependentes das garantias existentes para as fontes de leitura e escrita, logo, para todos considerou-se o uso de Apache Kafka na distribuição de dados. Nestas circunstâncias o

Apache Spark é o único que não oferece, ainda, a semântica desejada. Com a semântica *Pelo menos uma vez* o Apache Flink é o único que efectivamente repetiu eventos já processados, nos cenários testados.

Da comparação pode-se concluir que o uso do Kafka Streams aparenta apresentar um equilíbrio razoável entre volumes de processamento, latência e tolerância a falhas com reduzido período de indisponibilidade, aliado à facilidade de integração e desenvolvimento com o uso do Apache Kafka, resultante da Secção 2.2.

2.4 Persistência de Dados

No desenvolvimento de um sistema responsável pelo processamento de dados em larga escala outro aspeto fundamental, e em continuidade do estudo desenvolvido na Secção 2.2, é a escolha de tecnologias para a Persistência de dados. Neste contexto considera-se que os dados a armazenar podem ser tanto na forma dos dados iniciais recolhidos e por processar, em termos de resultados intermédios, cujo estado associado se pretende preservar ou transmitir a outros componentes, e principalmente em termos dos resultados obtidos após processamento, com o intuito de virem a ser consumidos, futuramente, por uma solução de visualização.

Na escolha de soluções de Persistência existem um número de aspetos principais que devem ser equilibrados em função dos requisitos existentes. O desempenho da tecnologia da tecnologias, quer em termos de escritas como leituras, as garantias de Consistência e Robustez, relativamente as operações aplicadas, e a sua garantia de *disponibilidade*.

Sendo que a Persistência desejada poderia ser obtida com recurso a um sistema distribuídos de ficheiros, por exemplo o HDFS, sendo um padrão comum em sistemas de processamento de grandes volumes de dados, como referido na Secção 2.2. No entanto, sendo o principal objetivo a produção de resultados facilmente acessíveis a aplicações orientadas aos utilizadores, privilegia-se a escolha, mais tradicional, de um Base de Dados que ofereça níveis superiores de Escalabilidade. Em particular considerou-se o uso de Bases de Dados ditas *NoSQL* [42], por estarem orientadas a dados não estruturados, ou, semiestruturados.

Novamente reforça-se a importância da escolha de soluções *Open Source* na selecção de tecnologias candidatas.

2.4.1 MongoDB

O MongoDB [70, 71] é uma Base de Dados orientada a Documentos, que seguem uma especificação baseada no JSON. Dentro de cada documento suporta a manipulação de campos complexos, que podem ser desde tipos primitivos a colecções de outros documentos. O uso de uma estrutura de documentos semelhante ao JSON permite o fácil mapeamento dos documentos em objetos da linguagem de implementação desejada.

A Escalabilidade no MongoDB é realizada horizontalmente através de um esquema de fragmentação automática, associada a um mecanismo de replicação orientado a uso de arquitecturas lógicas Primário-Secundários. De forma a tolerar falhas, oferece a autoeleição do Primário, que outra maneira seria um ponto único de falha. No entanto, durante o processo de eleição, naturalmente, o MongoDb encontra-se indisponível para para escritas, mas continuando a suportar leituras. As alterações realizadas sobre um único documento são sempre atómicas, com recurso a *locks* de escrita. Para alterações a múltiplos documentos oferece, ainda, uma API para controlo transacional.

O MongoDB oferece, ainda, a aplicação limitada do paradigma MapReduce sobre os dados que possui.

2.4.2 Apache Cassandra

O Apache Cassandra [71, 72] é uma Base de Dados orientada a Colunas de grandes dimensões, surgindo como uma implementação *Open Source* de padrões comprovados do Google BigTable e Amazon Dynamo. A semelhança destas soluções anteriores o modelo de dados concentra-se na manipulação de colunas, e super-colunas (colunas constituídas por outras colunas), que pertencem em linhas, cada linha com número variável de colunas.

Para definição de operações o Apache Cassandra suporta uma linguagem semelhante ao SQL, o Cassandra Query Language (CQL).

Para controlo do esquema de fragmentação automático todas as linhas possuem uma coluna com identificação da partição atribuída. Para Escalabilidade horizontal oferece uma arquitetura sem réplicas Primárias, isto é, todos as réplicas num *cluster* podem realizar escritas sobre as colunas atribuídas. Havendo um nível configurável de Consistência para as operações, os conflitos são resolvidos por comparação do *timestamp* associado, prevalecendo a alteração mais recente. A Persistência é garantida pela escrita em disco dos registos das operações e factores de replicação configuráveis.

2.4.3 Apache HBase

O Apache HBase [71, 73] é outra solução de Base de Dados orientada a Colunas de grandes dimensões, dentro do universo Apache, que surgiu em resposta ao Google BigTable. Possui, portanto, um modelo de dados semelhante ao Apache Cassandra. Uma diferença fundamental é que recorre ao HDFS como camada subjacente de Persistência, em particular aproveitando os seus mecanismos de replicação e tolerância a falhas. Depende ainda do uso do Apache Zookeeper.

Note-se que as colunas do Apache HBase não podem ter tipos explícitos, suportando apenas sequências de *bytes* não estruturados. Esta limitação impede a realização de pesquisas, por indexação, que não sejam baseadas nas chaves primárias. Outros tipos de pesquisa tem de ser sempre realizadas por iteração com auxiliares que interpretam os *bytes*.

A atomicidade de operações é garantida ao nível de cada linhas. Sobre a Consistência é garantido que as alterações nas linhas não podem resultar de operações intercaladas e que perante conflitos são escolhidas as operações mais recentes.

2.4.4 Comparação de Mecanismos

Considera-se que, globalmente, todas as Bases de Dados consideradas como candidatas possuem o conjunto de requisitos desejados, em particular as garantias necessárias de Escalabilidade, Consistência e Disponibilidade. Nota-se que estudos prévios [71] sobre a comparação destas tecnologias concluíram que não existe uma escolha clara que permita considerar uma solução *NoSQL* como superior as restantes em todos os cenários, requerendo uma análise caso a caso.

Em termos do equilíbrio entre desempenho e garantias oferecidas o Apache Cassandra apresenta elevado desempenho na escrita, sem sacrifício da Disponibilidade e Consistência, mas com sacrifícios no desempenho para as leituras [71]. Por oposição, o MongoDB oferece desempenhos elevados na leitura, associados a Disponibilidade e Fiabilidade, mas sacrificando a sua Disponibilidade e desempenho nas escritas. O Apache HBase, oferecendo boa Escalabilidade, possui desempenhos, assim como, garantias de Disponibilidade e Consistência apenas razoáveis.

Conclui-se que, em função de se pretender maximizar a escrita ou a leitura, deve-se optar pelo Apache Cassandra ou MongoDB, tomando em consideração que, apesar de possuir Escalabilidade mais limitada, o MongoDB pode ser preferível em situações que requeiram maior Fiabilidade, ou, beneficiem de um modelo orientado a documentos, ou, suporte direto ao uso de GeoJSON [74].

2.5 Interface com Utilizadores e Sistemas Externos

Recordando que é objetivo a criação de interfaces públicas que permitam tanto a interação com utilizadores finais, apresentando visualizações pré-definidas para a interpretação dos dados, como com sistemas externos que pretendam criar as suas próprias análises e visualizações, é necessário o estudo de alternativas tecnológicas.

Neste ponto a escolha das tecnologias a utilizar é fortemente dependente das limitações impostas pelas escolhas tecnológicas anteriores, isto é, que possuam o suporte adequado para a integração com os restantes componentes. Em particular, é necessário considerar a interação com a solução de Persistência de dados, a escolha do protocolo para implementação da API e suporte para a criação das visualizações desejadas, em particular de Mapas de Calor. Nota-se ainda que existe, neste ponto, uma oferta de soluções tecnológicas muito significativa e variada, sendo difícil a sua comparação em termos do desempenho, privilegiando-se, então, a consideração da facilidade de manutenção de flexibilidade da solução. Considerando ainda o estudo realizado na Secção 2.2, sendo que o acesso as interfaces deve ser realizado de forma síncrona, com menor latência possível, e não sendo razoável o acesso directo a solução de Persistência, considera-se o uso de Distribuição via Serviços. Como notado, no contexto de uma Aplicação Web a metodologia mais comum é o uso de APIs REST.

2.5.1 Spring Framework

O projecto Spring, do qual o Spring Framework é o módulo principal, [75, 76] apresenta uma infraestrutura *Open Source* para desenvolvimento de soluções expressarias no universo Java, originando como alternativa ao uso soluções orientadas Enterprise Java Beans (EJB). Apresenta como principal vantagem a simplificação e agilização do processo de desenvolvimento através de um modelo de Inversão de Controlo (IC), pelo meio de Injeção de Dependências (ID), associado a uma oferta significativa de módulos para integração com outras tecnologias, por exemplo, soluções de Persistências.

A IC é um paradigma em que módulos de os módulos num nível lógico superior, definindo o comportamento geral do componente, escolhem os módulos de um nível lógico inferior, com comportamento especializado, para execução. Dentro deste paradigma o Spring define o funcionamento geral das aplicações e serviços, sendo que o desenvolvedor injeta o comportamento que deseja, na forma de ficheiro de configuração com classes marcadas para o efeito.

Outro módulo de especial relevo é o Spring Boot, que simplifica o processo inicial de

configuração, sendo responsável pela gestão dos ficheiros de configuração dos quais pode depender a configuração de dependências, com integração de ferramentas como Graddle e Maven. Para além de Java o Spring suporta ainda o uso de Kotlin.

2.5.2 React

O [77] é uma biblioteca Javascript para a implementação declarativa de interfaces de utilizador interactivas, originada no Facebook. O seu funcionamento é baseado no conceito de *Components*, que possuem um estado associado. Cada *Component* pode ser carregado, ou carregar outros *Components*, com passagem do estado associado, permitindo a simples composição de interfaces. O estado passado pode incluir funções para passagem de interacções do utilizador para outros *Components*. Perante alterações de estado os módulos React são responsáveis por desencadear a reconstrução dos *Components* afectados.

Cada *Component* pode ser descrito com recurso a excertos simples de HTML, ou, enriquecido com JSX. O JSX é uma sintaxe específica do React que permite a descrição de HTML com inclusão dinâmica de variáveis de outros *Components*.

2.6 Resumo do Estudo Realizado

No presente Capítulo foi levado a cabo o estudo do Estado da Arte existente para o desenvolvimento de uma plataforma de avaliação de Quality of Service de uma rede LoRaWAN, em particular no contexto das Cidades Inteligentes.

Identificam-se como aspetos fundamentais da arquitetura à propor a definição dos mecanismos para recolha e distribuição das métricas a processar, incluindo valores intermédios e resultados finais, assim como dos mecanismos para o seu processamento.

No tópico da distribuição de dados são considerados três tipos de distribuição: via Serviços, via Passagem Assíncrona de Mensagens e via Bases de Dados. A Distribuição via Serviços é considerada como principalmente aplicável para casos de comunicação infrequente com necessidade de um modelo Cliente-Servidor e em que se considera aceitável que só exista comunicação quando tanto a origem e o destino estejam disponíveis, sendo a metodologia privilegiada na construção das interfaces com sistemas externos e utilizadores. A Distribuição via Passagem Assíncrona de Mensagens é preferível no tratamento de volumes significativos de dados, com processamento associado que possa ser moroso e dividido entre múltiplos componentes, com necessidade de garantias na entrega dos dados, na análise comparativa identificou-se o Apache Kafka como a

alternativa tecnológica preferencial. A Distribuição via Bases de Dados é preferencial nas situações em que existem, já, dados processados que são acedidos conforme sejam requeridos pelos componentes.

Sendo que na recolha das métricas em tempo real, assim como, os seus resultados e valores intermédios, se identificou como preferencial a definição de fluxos via Passagem Assíncrona de Mensagens, o uso de infraestruturas de Processamento em Fluxo apresenta um enquadramento natural. Foi identificado como principal solução, para equilíbrio entre volumes de processamento, latência e tolerância a falhas com reduzido período de indisponibilidade o Kafka Streams, em particular quando já se tenha definido o uso de Apache Kafka.

Na estudo da Persistência de Dados, em conexão com o estudo da Distribuição via Bases de Dados, foram analisadas soluções NoSQL por estarem orientadas a dados não estruturados, ou, semiestruturado s. Neste estudo, tendo-se eliminado a hipótese do uso do Apache HBase, não se identificou uma solução preferencial entre Apache Cassandra e MongoDB. Pode-se optar pelo Apache Cassandra privilegiando-se a Escalabilidades, ou, pelo MongoDB privilegiando-se a Fiabilidades e outras funcionalidades oferecidas. Considerando a necessidade de utilizar o Orion Context Broker (OCB), cujo funcionamento é dependente do MongoDB, considera-se que o MongoDB é uma solução válida.

Na construção das Interfaces com Utilizadores e Sistemas Externos, sendo que existe um leque extenso de alternativas tecnológicas possíveis, é privilegiado a escolha de soluções com custo reduzido de implementação, manutenção e adaptação. Considera-se em particular o uso do Spring Framework, para implementação de uma interface REST, e do React para criação das visualizações de referência que consume a interface.

Tomando o estudo realizado, incluindo as considerações já apresentadas, foi definida uma arquitetura proposta que se apresenta no Capítulo 3.

3

Arquitetura e Implementação Propostas

3.1 Introdução

Tendo-se realizado o estudo das tecnologias candidatas no Capítulo 2, no presente capítulo apresenta a solução de arquitetura proposta e principais considerações de implementação para resposta aos objetivos e requisitos identificados no Capítulo 1.

O capítulo encontra-se organizado em secções tais que:

- Secção 3.2: visão geral e lógica da arquitetura proposta, indicado responsabilidades e interações base do sistema da plataforma;
- Secção 3.3: visão detalhada da arquitetura proposta, indicando escolhas tecnológicas para implementação;
- Secção 3.4: definição de um modelo de processamento em fluxo das métricas recolhidas, que seja adaptável e reutilizável independentemente da rede LPWAN;
- Secção 3.5: modelos e estruturas de dados identificados como comuns na arquitetura, no contexto do seu fluxo;
- Secção 3.6: considerações sobre os componentes adoptados do projeto FIWARE, em particular aqueles que forma modificados;
- Secção 3.7: definição da Interface com Utilizadores e Sistemas Externos;

- Secção 3.8: considerações adicionais de implementação. Integração de componentes e balanceamento de carga.

3.2 Visão Geral da Arquitetura

Reforça-se que, o primeiro requisito obrigatório é a utilização de componentes do catálogo FIWARE para integração do sistema e gestão do contexto. Naturalmente, tal arquitetura é construída em torno da utilização do Orion Context Broker (OCB). Considerando esta base de desenvolvimento, com vista a permitir o suporte a múltiplas redes Low-Power Wide-Area Network (LPWAN), propõem-se a visão arquitetural geral apresentada na Figura 3.1.

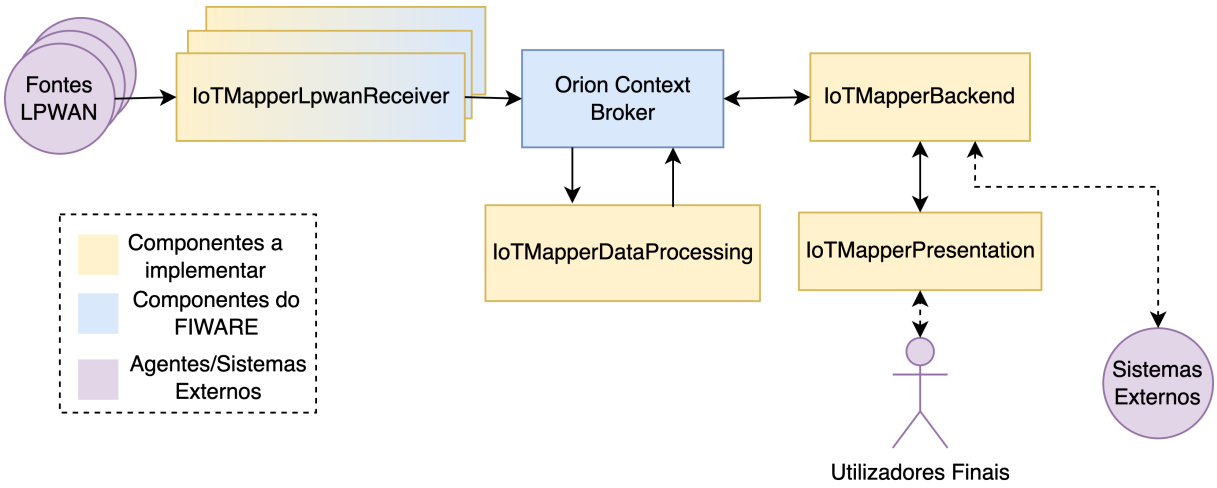


Figura 3.1: Visão Arquitetural Geral.

Os blocos funcionais do sistema podem ser resumidos, com respetivas responsabilidades, da seguinte forma:

- As fontes das LPWAN representam qualquer sistema externo, ou, dispositivo de Internet das Coisas, capaz de reportar métricas sobre uma rede LPWAN;
- Os *IoTMapperLpwanReceivers* são o conjunto de componentes adequados para uma, ou mais, redes LPWAN, mediando a receção e inserção de métricas. Pode ser qualquer componente capaz de utilizar a Application Programming Interface (API) do NGSI-v2 . O catálogo FIWARE já oferece alguns componentes adequados, nomeadamente os IoTAgents;
- O Orion Context Broker (OCB), desempenha o papel de gestor dos dados de contexto, compostos pelos relatórios iniciais de métricas e as agregações obtidas.

Permite a integração de outros componentes FIWARE de acordo com a API do NGSI-v2;

- O *IoTMapperDataProcessing* recebe as métricas iniciais do OCB, processando-as, em tempo perto do real. A receção continua de novos relatórios de métricas configura um fluxo de eventos a serem processados, filtrados e agregados no OCB como entidades de métricas finais;
- O *IoTMapperBackend* intermedia o acesso ao contexto gerido, permitindo tanto a sistemas externos como à *IoTMapperPresentation*, consumir as entidades de métricas agregadas, abstraindo os detalhes de implementação, tais como os serviços FIWARE utilizados para a separação do contexto das entidades;
- O *IoTMapperPresentation* permite a exploração das métricas agregados por utilizadores gerias, apresentando uma visualização de referência.

3.3 Visão Detalhada da Arquitetura

No contexto da secção anterior, e das alternativas tecnológicas identificadas, optou-se pela definição de uma arquitetura assente em três eixos tecnológicos: uso de componentes do projeto FIWARE para contextualização das métricas, uso do Apache Kafka para distribuição e processamento das métricas e uma camada de apresentação para interface e visualização dos dados com o uso do Spring Framework e React.

Considere-se a Figura 3.2 que apresenta a visão detalhada da arquitetura de solução proposta para implementação, agora dividida em três agrupamentos (*clusters*) lógicos.

Tomando a arquitetura apresentada identificam-se os seguintes componentes, com os detalhes das escolhas de implementação:

- **Cluster FIWARE:** responsável pelo gestão das métricas recebidas e agregadas, de forma contextualizada. Executa o papel de receção inicial dos relatórios de métricas, associando imediatamente o contexto da sua criação, assim como o papel da disponibilização das métricas agregadas. Destacam-se, no *cluster*, os principais de componentes:
  - **IoTMapperLpwanReceiver:** conjunto de componentes adequados a uma, ou mais, redes LPWAN, mediando a receção e inserção das métricas. Sendo o único requisito que suporte a API do NGSI-v2. No caso do suporte a redes Long Range Wide-Area Network (LoRaWAN), identifica-se o uso do IoTAgent-LoRaWAN;

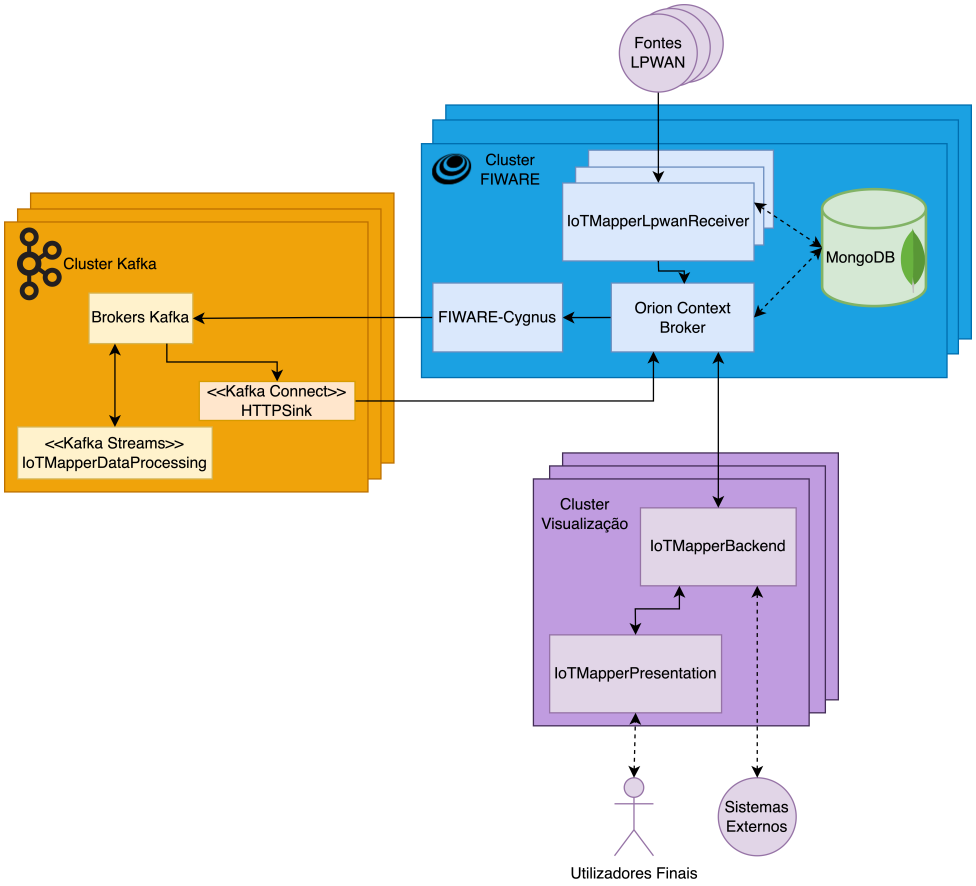


Figura 3.2: Visão Detalhada da Arquitetura, com escolhas de alternativas tecnológicas. As cores utilizadas são meramente ilustrativas.

– **Orion Context Broker (OCB):** gere o contexto do sistema, armazenado como um conjunto de entidades NGSI-v2 criadas pelo restantes componentes. Em resultado de alterações do estado nas entidades geridas emite notificações para o FIWARE-Cygnus;

– **MongoDB:** camada de persistência NoSQL. Requerida pelo OCB para armazenamento das entidades e informação de gestão, como notificações. Serve, ainda, para suporte de alguns dos componentes IoTAgent usados como implementação dos *IoTMapperLpwanReceiver*.

– **FIWARE Cygnus:** em resultado de notificações recebidas realiza a inserção das entidades num mecanismo externo de persistência configurável. Foi escolhido o Apache Kafka. O uso do Apache Kafka, como já notado, oferece complementaridade natural ao mecanismo assíncrono de notificações do OCB, com maior Fiabilidade que o uso de Message Queuing Telemetry Transport (MQTT) suportado diretamente pelo OCB. Seguindo os padrões

de desenho arquitetural do FIWARE, a adição do Cygnus é o modelo recomendado para integração;

- **Cluster Kafka:** é o resultado do desdobramento do componente *IoTMapperDataProcessing* original, na Figura 3.1, para uso de componentes do ecossistema Apache Kafka. Responsável pela distribuição das métricas recolhidas, do seu processamento e persistindo um histórico das métricas e dos resultados do processamento. Em particular, tem-se:

  - **Apache Kafka:** componente principal e definidor do *cluster*. Responsável pelo controlo do padrão Publicador/Consumidor, incluindo a replicação e persistência das mensagens enviados para os tópicos e a integração com os restantes componentes. Após receção de uma mensagem oriunda do FIWARE Cygnus, realiza a entrega da mesma ao, *IoTMapperDataProcessing*, recebendo deste os resultados do processamento;
  - **IoTMapperDataProcessing:** definido sobre Kafka Streams, para integração e persistência mediados pelo Apache Kafka, é responsável pelo processamento das mensagens obtendo agregações sobre as métricas para as diferentes redes LPWAN, sendo o seu comportamento e responsabilidades definidos em detalhe na Secção 3.4;
  - **HTTPSink:** componente de conexão, integrado com o Apache Kafka através do uso do Kafka Connect. Responsável pela inserção, no OCB, dos resultados obtidos, após o processamento no *IoTMapperDataProcessing*, e escritos num tópico do Apache Kafka. Nenhuma implementação identificada possuía o nível de controlo necessário para livre definição de cabeçalhos Hypertext Transfer Protocol (HTTP) necessários e do formato Javascript Object Notation (JSON) usado nos mecanismo de agrupamento (*batch*) do OCB. Como tal, foi desenvolvida uma implementação modificada com a adição das características requeridas;

- **Cluster Visualização:** responsável pela a apresentação das agregações finais. Interage, portanto, com o OCB para pesquisa e obtenção dos resultados a apresentar. Subdivide-se nos componentes:

  - **IoTMapperBackend:** componente principal do *cluster*, sendo responsável pela integração com os restantes Clusters, na obtenção dos resultados a apresentar. Os resultados podem ser requeridos pelo *IoTMapperPresentation*, ou, eventualmente, sistemas externos;

– **IoTMapperPresentation:** responsável pela a apresentação a um utilizador final, construído a interface gráfica, com Mapa de Calor, para as diferentes redes LPWAN suportadas.

3.4 Arquitetura do Processamento em Fluxo

Em resultado do estudo realizado sobre as tecnologias candidatas foi definida a utilização do Apache Kafka, na distribuição das métricas. Tendo em particular consideração que o Apache Kafka oferece forte integração com o uso do Kafka Streams, em particular na Persistência local, partição dos dados e tolerância a falhas durante o processamento, definiu-se, também, o uso do Kafka Streams.

Na arquitetura apresentada anteriormente na Secção 3.3 o uso do Kafka Streams corresponde ao componente *IoTMapperDataProcessing*. Nesta infraestrutura tecnológica é privilegiado um modelo de processamento, e modelo de programação associado, orientado a fluxos de dados, isto é, o processamento contínuo de dados recebidos de uma, ou mais, fontes de dados, através do encadeamento de tarefas, resultando em um, ou mais, resultados processados. Na arquitetura proposta o objetivo consiste na receção continua de métricas, oriundas de múltiplas redes LPWAN, com o objetivo de obterem agregações para apresentação e análise, um modelo de processamento em fluxo é uma escolha natural.

Propõe-se a definição de uma Arquitetura de Fluxo de Processamento, que defina um modelo de processamento comum e flexível á reutilizar para o processamento das métricas das diferentes redes LPWAN suportadas, atualmente e futuramente. No contexto de infraestruturas para processamento em fluxo é comum a designação de Topologia para descrever as arquiteturas definidas. A Topologia lógica proposta é apresentada na Figura 3.3.

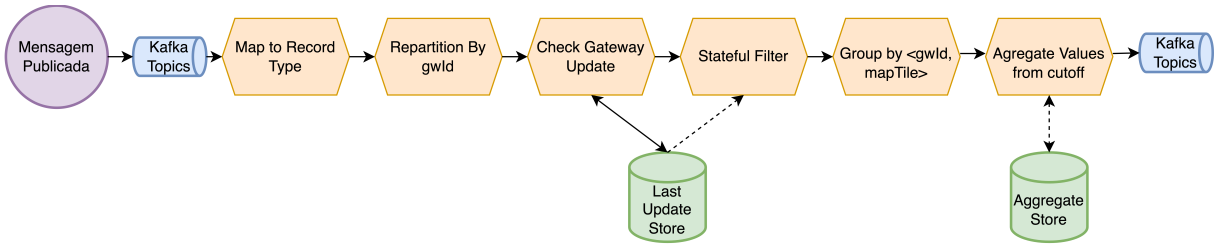


Figura 3.3: Topologia da Arquitetura para processamento em fluxo das métricas recolhidas. As cores utilizadas são meramente ilustrativas.

Sendo a sua implementação especifica do Kafka Streams considera-se que, em termos da sua sequência lógica das tarefas o modelo pode ser adaptado a qualquer infraestrutura

de processamento em fluxo, que ofereça manutenção de estado associado.

Na topologia define-se que as diferentes fontes de métricas sejam representadas por tópicos do Apache Kafka distintos, identificados por padrões como apresentados na Secção 3.5.2. Para cada padrão existe uma especificação da topologia apresentada, responsável pelo processamento para cada rede LPWAN. Em particular, cada especificação é responsável pelo tratamento adequado dos formatos utilizados paras os relatórios de métricas, e pela identificação das métricas a recolher, especificadas no Apêndice B.

Outro aspeto de importância a abordar é o da partição dos tópicos. De forma a garantir a Escalabilidade dos seus sistemas o Apache Kafka, e por extensão as soluções construídas com Kafka Streams, recorre à divisão das mensagens pertencentes a um mesmo tópico entre múltiplas partições configuráveis. Esta partição é resultado da divisão de espaço de chaves, com uma chave, possivelmente repetida, associada a cada mensagem. Os componentes que consumem as mensagens, por exemplo, o *IoTMapperDataProcessing*, tendencialmente, devem ler as mensagens de apenas uma partição, ou, conjunto reduzido de partições em cada tópico. Para criação dos resultados finais desejados é necessário a associação das métricas ao equipamento de rede associado a sua recolha, designado na arquitetura por Gateway (GW) para todas as redes, logo uma única instância do *IoTMapperDataProcessing* deve conseguir obter todos os registos relevantes. Note-se que se deve minimizar a necessidade de particionar as mensagens repetidamente, uma vez que cada repartição no Apache Kafka implica uma nova escrita num tópico, e subsequente leitura.

A lógica sequencial da topologia pode ser dividida em fases tais que:

- **Mapeamento:** as mensagens são lidas do tópico de entrada, identificado pelos nomes descritos na Secção 3.5.2, estando inicialmente no formato genérico *CygnusTopicMessage*. Cada mensagem é mapeada para um *IMetricsRecord*, onde o mapeamento é especializado para cada rede LPWAN;

- **Repartição:** quando inicialmente lidas as mensagens não possuem uma chave associada, sendo a sua partição realizada aleatoriamente pelo conjunto de partições, garante-se a correta distribuição dos dados através da extração de cada *IMetricsRecord* de um identificar do GW, o *gwId*, a usar como chave;

- **Verificação de Alteração:** as métricas recolhidas só são válidas enquanto as características do canal permanecem, razoavelmente, inalteradas. O *IoTMapperDataProcessing* verifica, para cada novo *IMetricsRecord*, um conjunto de mecanismos para confirmação da sua validade. Como mecanimso por omissão, um Gateway não

pode sofre alterações a sua localização superiores a 100 metros, da primeira localização observada. Sempre que as condições de análise do canal tenham sofrido alterações é iniciada uma nova agregação. Outro parâmetro de interesse poderia ser o tempo desde o início da agregação atual;

- **Filtragem das Métricas:** em função da decisão armazenada no passo anterior filtra-se o fluxo de dados de forma a incluir apenas aqueles com origem numa mensagem posterior a ultima alteração do estado;

- **Agrupamento das Métricas:** as métricas resultantes a calcular não estão associadas apenas a localização do GW, mas também do equipamento que realizou a medição. Não é razoável, quer no processamento, ou, na construção das visualizações das métricas, o uso da localização absoluta de todas as medições. Propõem-se que estas sejam agrupadas em zonas geográficas (designadas *mapTiles*) como descrito na secção 3.5. As métricas são, então, agrupadas por pares <*gwId*, *mapTile*>;

- **Agregação das Métricas:** para cada agrupamento as métricas são agregadas, pela lógica de redução definida para cada rede LPWAN, de forma a respeitar as métricas e unidades identificadas no Apêndice B.2. O resultado (um *IMetricsAggregate*) é persistido localmente, seguindo o mecanismo de *KTable* oferecido pelo Kafka Streams. Este mecanismo permite a associação das cópias locais das agregações as instâncias especificas de *IoTMapperDataProcessing*, segundo as mesmas chaves usados para as mensagens. Como tal, as cópias a atualizar estão sempre presentes nas instâncias que realizam o processamento;

- **Escrita dos Resultados:** adicionalmente à escrita local, é necessário disponibilizar o resultado ao OCB. Tal é alcançado pela escrita dos resultados num tópico de saída, seguindo o modelo NGSI-v2 descrito na Secção 3.5.3. Os resultados são lidos pelo *HTTPSink*, gerido pelo Kafka Connect, e injetados no OCB.

A topologia apresentada, descreve o conjunto de passos lógicos que devem ser aplicados a todas as mensagens para todas as redes LPWAN. No entanto, não corresponde diretamente à implementação. Na criação de topologias o Kafka Streams oferece duas APIs, a *Streams DSL* e a *Processor API*. A *Streams DSL*, oferecendo uma sintaxe fluente para composição das tarefas, é utilizada para definição geral da topologia, incluindo a integração de passos definidos com a *Processor API*, quando se deseja maior controlo sobre o estado das tarefas. Em particular a *Processor API* é utilizada na verificação da alteração do estado, na filtragem das mensagens e no agrupamento e agregação como um passo unificado, em resultado da necessidade de manipulações de estado local não previstas na *Streams DSL*. No caso específico dos passos *Agrupamento das Métricas* e

Agregação das Métricas, descritos na topologia, a sua implementação foi agrupada num único passo interno do *IoTMapperDataProcessing*. No fim do passo *Agrupamento das Métricas*, a escolha de uma nova chave para as agregações (os pares <*gwId*, *mapTile*>) resulta na necessidade de repartir internamente as mensagens, antes do passo seguinte. Este novo espaço de chaves corresponde a divisão do espaço existente onde as chaves são os *gwId*. Como tal é possível garantir que cada instância do *IoTMapperDataProcessing* já possui todas as mensagens correspondentes a cada nova chave. Este escolha de implementação permite eliminar uma repartição adicional dos tópicos internos do Kafka Streams, sacrificando a possibilidade da divisão do trabalho a realizar para um único Gateway. Esta escolha é considerada razoável, devido ao peso computacional associado a transmissão das mensagens para instâncias distintas.

3.5 Modelos de Dados

De forma a permitir a interface entre os diferentes componentes arquiteturais é fundamental a definição dos Modelos de Dados a utilizar, tanto na Persistência como na Distribuição da informação. Os modelos definidos devem ter em consideração a necessidade de serem flexíveis para suporte a múltiplas redes LPWAN e protocolos, incluindo a hipótese de adição de novos, como o rigor necessário a definição clara dos requisitos dos dados e interoperabilidade entre componentes heterogéneos, isto é, com implementações distintas. De notar, ainda, que sendo os modelos lógicos agnósticos a tecnologia, os modelos efectivamente implementados e apresentados são influenciados pelas limitações e boas praticas de cada tecnologia escolhida, em particular a definição de Modelos de Dados no contexto do projeto FIWARE.

A organização da secção respeita o fluxo geral dos dados, iniciando na sua recolha e terminado na sua Persistência após o seu processamento.

3.5.1 Recolha de Métricas

Na recolha inicial de métricas os modelos a definir são dependentes dos formatos de dados implementados por cada rede LPWAN, sendo a sua interpretação e conversão responsabilidade dos componentes *IoTMapperLpwanReceiver*.

Para o caso de referência do LoRaWAN considerou-se o formato de mensagem utilizado na rede The Things Network (TTN), incluído no Apêndice A, um formato JSON semiestruturado. Note-se que as métricas a extrair, geradas pela rede, ou, geradas pelos equipamentos de IoT, assim como informações de contexto como os GWs que

receberam a mensagem, são incluídas no campo *uplink_message*. A estrutura do campo é variável em função da informação existente e contexto de origem da mensagem, em particular, se teve origem noutra rede LPWAN antes de ser reencaminhada para a TTN.

Considerando a possível variabilidade do formato de entrada e desejando a existência de um formato comum definiu-se que a mensagem que contem uma métrica, após conversão no *IoTMapperLpwanReceiver*, corresponde a Tabela 3.1. Onde *json()* é a função que corresponde a conversão da mensagem num objecto JSON, ou, não o alterando se já for um objecto JSON. Os valores exactos escolhidos para os campos *service* e *subservice* são indicados no Apêndice B.1.

Tabela 3.1: Modelo lógico para métricas recolhidas e por processar.

Campo	Tipo	Observações
service	Text	Identificação da rede LPWAN a que corresponde a métrica
subservice	Text	Especificação da origem associada ao *service*, para uma métrica
message	Text	Codificação da mensagem especifica da rede LPWAN, tal que *message = json((lpwan)*

O modelo é inserido, por cada *IoTMapperLpwanReceiver*, no OCB, resultando numa entidade NGSI-V2 [20], com as correspondências da Tabela 3.2. Não sendo identificado no repositório publico FIWARE um modelo de dados que enquadre o campo, *message* como descrito, foi definido um novo modelo (*MetricsReport*), cujos detalhes de implementação são indicados no Apêndice C.1. Note-se que, para compatibilidade com o uso de FIWARE IoTAgents, o campo *message* precisa de ser convertido por uma função *urlEscape()*, tal que sejam codificados os caracteres reservados nos RFC 1738 e 3986 [78, 79].

Tabela 3.2: Correspondência do modelo lógico da Tabela 3.1 no Orion.

Modelo lógico	Correspondência Orion
service	Cabeçalho *fiware-service*, associado as entidades criadas
subservice	Cabeçalho *fiware-servicepath*, associado as entidades criadas
message	Campo *raw_message* dentro dos atributos das entidades, tal que: *message = urlEscape(message)*

3.5.2 Representação das Métricas no Apache Kafka

Na inserção dos dados no Apache Kafka para distribuição e processamento é necessário definir os formatos para consumo das mensagens e distribuição das métricas.

No consumo das métricas, conforme indicado na Secção 3.4, a informação condutora da Distribuição dos Dados são os tópicos utilizados para publicação das mensagens

resultantes das entidades descritas na Secção 3.5.1. A associação é realizada através da composição hierárquica dos campos *service* e *subservice* da Tabela 3.1, tais que se obtenham tópicos lógicos no formato <*service*>/<*subservice*>. Tendo-se optado pelo uso do FIWARE Cygnus para integração do Apache Kafka o formato dos tópicos é limitado pelos seus detalhes de implementação. O formato efetivo é <*service*>*xffffx002f*<*subservice*>, correspondendo, de entre os formatos suportados pelo FIWARE Cygnus, ao *dm-by-service-path*. Note-se que em ambos os formatos o carácter "/" no início de *subservice* é omitido, por já estar contemplado no formato.

O formato inicial das mensagens, novamente por imposição da escolha tecnológica, segue o formato especificado pelo FIWARE Cygnus, isto é, um mapeamento da entidade FIWARE com os metadados associados, como resumido na Tabela 3.3. Note-se que a escrita das mensagens nos tópicos é resultante de uma subscrição registada no OCB, ativando uma notificação para todas as alterações.

Tabela 3.3: Mapeamento da entidade pelo FIWARE Cygnus.

Campo	Descrição
Headers	Cabeçalhos gerados, incluindo repetição do *service* e *subservice* e indicação do momento de recepção da notificação
Body	Mapeamento directo da entidade do Apêndice C.1

Sobre o identificador dos GWs, o *gwId*, este é uma representação textual UTF-8 cujo conteúdo é especifico de cada rede LPWAN. Sendo que existe uma separação de domínio realizada pelo *service* e *subservice* assume-se que, dentro do contexto definido por cada par anterior, é um identificador único, sendo responsabilidade dos componentes de receção de métricas, para cada rede LPWAN, a garantia de unicidade.

Sobre o identificador da área geográfica *mapTile* este deve garantir o mapeamento de qualquer posição na superfície terrestre de forma a agrupar as localizações das medições. Deve-se ainda garantir a possibilidade de associar um determinado valor *mapTile* a região a que representa na construção de visualizações, em particular, de Mapas de Calor. Optou-se pelo uso de GeoHash [80, 81] como mecanismo computacionalmente simples e amplamente implementado de agrupamento de coordenadas, isto é, a junção de coordenadas em grupos correspondentes a uma área geográfica de dimensão configurável. Sendo atribuído um identificador único a cada área.

3.5.3 Representação das Métricas Processadas

Após todo o processamento, os dados são inseridos no OCB. Para representação adequada e flexível definiu-se um modelo comum aos conjuntos de métricas produzidas

para as diferentes redes LPWAN. Foram ainda definidos modelos para representação de informação de gestão da plataforma.

Como visão harmonizada dos resultados obtidos no processamento das métricas apresenta-se a Tabela 3.4.

Tabela 3.4: Modelo lógico para métricas após processamento.

Campo	Tipo	Observações
service	Text	Identificação da rede LPWAN a que corresponde a métrica
subservice	Text	Especificação da origem associada ao *service*, para uma métrica
dateLastGwChange	Date	Data da ultimação alteração que gerou um novo cálculo da métrica
dateLastUpdate	Date	Data da ultimação actualização do valor da métrica
mapTile	Text	Identificador textual da região de interesse para a métrica
gwId	Text	Identificador textual do equipamento da rede LPWAN que recolhe as métricas
measures	List	Listagem das métricas calculadas, no mínimo com media do RSSI, dependente da rede LPWAN

A semelhança do modelo definido na recolha de métricas a implementação do modelo corresponde a criação de entidades NGSI-V2 inseridas no OCB, existindo um mapeamento semelhante. Os campos *service* e *subservice* correspondem, novamente, aos cabeçalhos *fiware-service* e *fiware-servicepath* para separação das entidades no NGSI-V2. Os restantes campos são atributos na mesma entidade (*MetricsAggregation*), detalhada no Apêndice C.2. Adicionalmente, considera-se a adição do atributo *location*, do tipo GeoJson, com representação da área associada ao *mapTile*, de forma suportar as pesquisas por área geográfica oferecidas pelo OCB e suportar acessos por sistemas externos que não implementam o mapeamento de *mapTile* adequado.

Não sendo resultado do fluxo de processamento, mas sim da configuração da infraestrutura, considera-se o enriquecimento do contexto dos dados através da criação de entidades que descrevam as redes LPWAN suportadas. Cada entidade, do tipo *LpwanInfo* especificado no Apêndice C.3, contem o nome e lista expectável de métricas para uma das redes implementadas. Para cada métrica são indicadas a sua designação, unidades e obrigatoriedade, em conformidade com listagem do Apêndice B.2.

3.6 Considerações sobre os Componentes FIWARE

Na implementação da arquitetura, juntamente com os componentes implementados, por exemplo, o *IoTMapperDataProcessing*, e os componentes simplesmente configurados, como o Apache Kafka, foi necessário modificar as implementações existentes do FIWARE Cygnus e do FIWARE IoTAgent-LoRaWAN.

Estes componentes modificados enquadram-se no *Cluster* FIWARE, sendo resultado da necessidade de atualizar e entender o comportamento oferecido pelo projeto FIWARE.

3.6.1 Adaptação do FIWARE Cygnus

Como já indicado o FIWARE Cygnus é utilizado como componente de integração entre o OCB e o Apache Kafka. Em particular como alvo de notificações oriundas do OCB que são transpostas como mensagens nos tópicos geridos pelo Apache Kafka.

Sendo que FIWARE Cygnus suporta múltiplos destinos de escrita dos dados transpostos, a data da implementação da arquitetura, os conectores existentes para o Apache Kafka suportam uma versão desatualizada (0.8.2), incompatível com o uso de Kafka Streams [82] e que recorre a métodos atualmente considerados como parte da API interna. A manutenção deste componente foi considerada como não prioritária [83].

Optou-se pela modificação do FIWARE Cygnus de forma avançar a implementação até a versão mínima do Apache Kafka considerada estável e necessária para os objetivos da implementação, em particular a versão (2.0.1).

3.6.2 Implementação e Adaptação dos LPWANReceivers

Na arquitetura proposta definiu-se a existência de componentes do tipo *IoTMapperLpwanReceiver* para mediação a receção e inserção das métricas. Sendo que as redes LPWAN são tipicamente utilizados no contexto do IoT, os componentes FIWARE IoTAgents, identificados previamente na Secção 1.2.2, representam uma solução adaptável para a recolha das métricas como descrito.

Para o caso de referência LoRaWAN o componente escolhido é o IoTAgent-LoRaWAN [84]. A semelhança da situação com o FIWARE Cygnus, a data da implementação, o componente encontra-se desatualizado, não possuindo suporte para a versão 3 do TTN, sendo tal necessário para a implementação proposta. Partindo de trabalho anterior de Ivo Pedroso [85], no sentido do suporte a versão 3, foi modificada a implementação

IoTAgent-LoRaWAN de forma a suportar a integração desejada. Foi ainda adicionado um novo modelo dados de entrada, *application_server_raw_string*, que insere a totalidade da mensagem, por oposição a apenas os campo de dados, correspondendo ao modelo de dados definido na Secção 3.5.1.

É importante realçar que não foi identificada investigação anterior que valide e avalie o desempenho do IoTAgent-LoRaWAN. Como tal, deve ser um dos pontos principais e analise durante a avaliação da arquitetura, tal como apresentado no Capítulo 4.

3.7 Interfaces para Visualização das Métricas

Tendo sido produzidas as agregações resultantes da recolha e processamento das métricas é necessário definir as interfaces para a sua exposição, correspondendo ao *cluster* Visualização. O *cluster* possui dois componentes, o *IoTMapperBackend* e o *IoTMapperPresentation*.

3.7.1 IoTMapperBackend

Em resultado do estudo anterior, nas Secções 2.2.1 e 2.5.1, optou-se pela implementação com recurso ao Spring Framework [75, 76, 86, 87], em particular a inclusão do Spring Boot e Spring MVC. O modelo de Inversão de Controlo permite simplificar o desenvolvimento da solução, dentro de uma infraestrutura comprovada e boas praticas de organização. Note-se ainda que, privilegiando-se a garantias adicionais de controlo de erros, recorreu-se a versão do Spring Framework para a linguagem Kotlin.

Considere-se a Figura 3.4 que apresenta a organização lógica interna do *IoTMapperBackend*, recorrendo aos estereótipos definidos para o Spring MVC. Em particular, note-se que a arquitetura permite o reaproveitamento da lógica de acesso a dados e de domínio, os componentes *@Repository* e *@Service* respectivamente, com possibilidade de definição de múltiplos componentes *@Controller*, correspondendo ao suporte de múltiplos protocolos de Distribuição via Serviços. Como implementação de referência, sendo o modelo mais comum no contexto de aplicações Web, optou-se pela implementação de uma Web API REST.

A Web API, cuja documentação é incluída no Apêndice D, foi definida segundo o modelo OpenAPI 3, que promove produção de documentação de qualidade num formato comum, com o objetivo de facilitar a eventual utilização da Web API por sistemas externos. Os objetos definidos são mapeamentos das entidades definidas na Secção 3.5.3. Sendo que o método principal de consumo da Web API é como uma

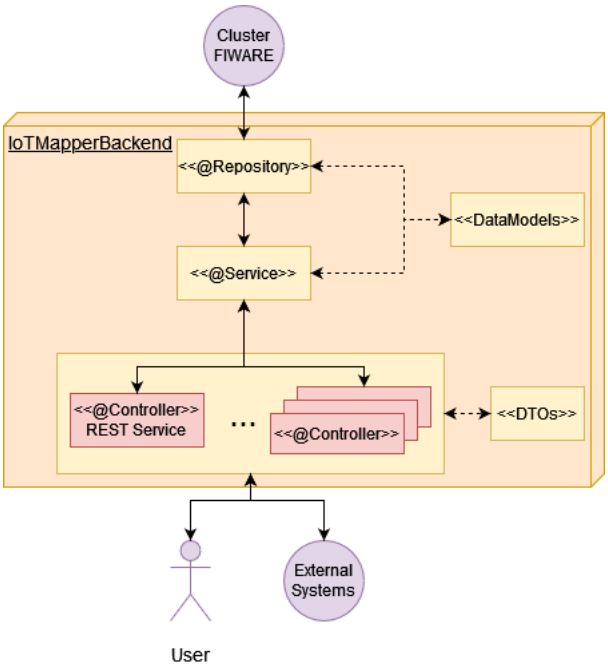


Figura 3.4: Arquitetura lógica interna do *IoTMapperBackend*.

listagem global das métricas, são expostos métodos, também, para acesso a pesquisa por área geográfica do OCB, como distancia máxima de um ponto de procura, e acesso a uma métrica individual.

3.7.2 IoTMapperPresentation

Para disponibilização dos *MetricsAggregation* a utilizadores finais foi desenvolvida, como visualização de referência, o *IoTMapperPresentation*.

A visualização escolhida como referência é um Mapa de Calor, que apresente as intensidades relativas para um par de rede e métrica especifica. Optou-se pela implementação da interface via o uso de React [77], privilegiando-se a facilidade de composição dinâmica das interfaces. Adicionalmente, sendo que a falta de tipos fortes no Javascript permite a ocorrência de erros que, de outra maneira seriam simples de evitar, optou-se pelo uso do React no contexto da linguagem Typescript [88]. Para composição da interface com apresentação gráfica consistente, sendo pratica comum, adotou-se o sistema de design gráfico Semantic UI [89]. Sendo que nem o React, nem o Semantic UI, implementam Mapas de Calor, foi ainda recorrido ao uso do OpenLayers [90], na forma da integração RLayers [91] para React.

A interface para o Mapa de Calor consume a Web API descrita na secção anterior para composição dinâmica da lista de redes LPWAN e métricas a apresentar, sendo que, em resultado da seleção do utilizador, compõem o mapa desejado.

Na construção do *IoTMapperPresentation* recorre-se a mecanismos de *cache* de longo prazo. A maioria do conteúdo é composto por páginas estáticas, cujos os nomes são resultados de *hashing*, e referenciados na página principal. Alterações no conteúdo resultam em novos nomes. Como tal, podem ser carregados um número reduzido de vezes. Um ficheiro de configuração *config.js*, é carregado mais frequentemente, indicando informação como o Uniform Resource Locator (URL) a referenciar nas páginas. O mecanismo de *cache* têm exposição adicional na Secção 3.8.2.2.

Considere-se, como exemplo da interface implementada, a Figura 3.5.

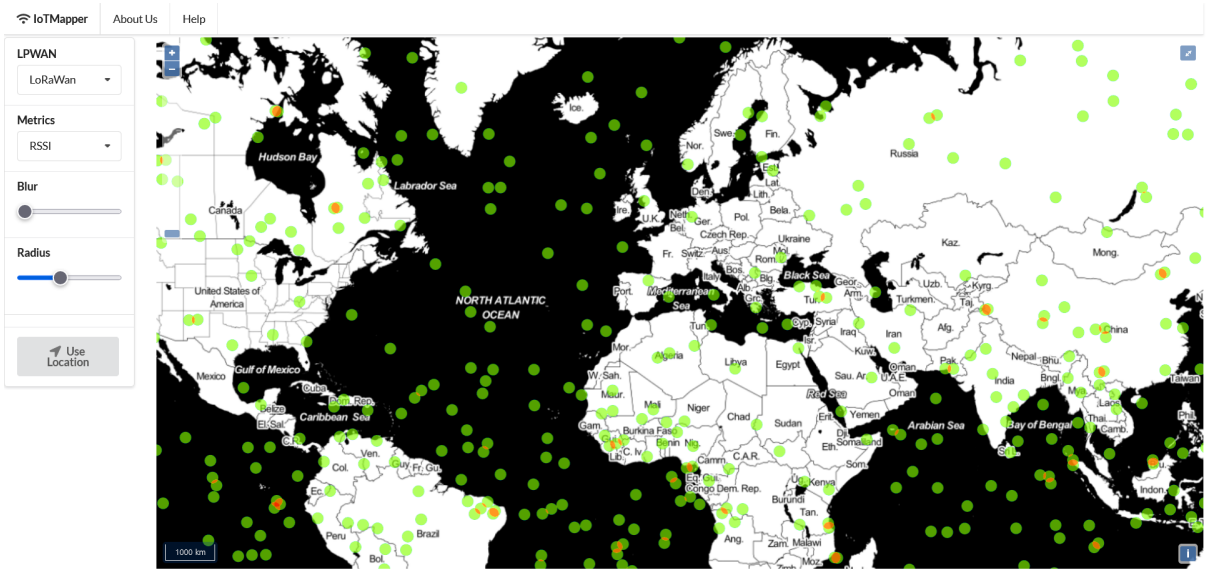


Figura 3.5: Interface gráfica para o *IoTMapperPresentation*, com o conjunto simulado de dados (*Data-02* definido na Secção 4.2). Como exemplo da interface de referência.

3.8 Considerações Adicionais de Implementação

Todos os componentes, quer desenvolvidos especificamente ou já existentes, foram cuidadosamente organizados para que possam ser instalados como contentores. Para implementação do protótipo foi escolhido o uso do Kubernetes [92–94]. O Kubernetes é uma plataforma *Open Source* para orquestração de contentores, com origem na plataforma Borg [95] da Google. Não foram tomadas quaisquer escolhas de implementação que dependem de vendedores *cloud* específicos.

Considere-se a Figura 3.6 que apresenta os detalhes de implementação adicionais.

Naturalmente, a utilização desta infraestrutura motivou o uso de conceitos específicos do Kubernetes na integração dos componentes. Em particular:

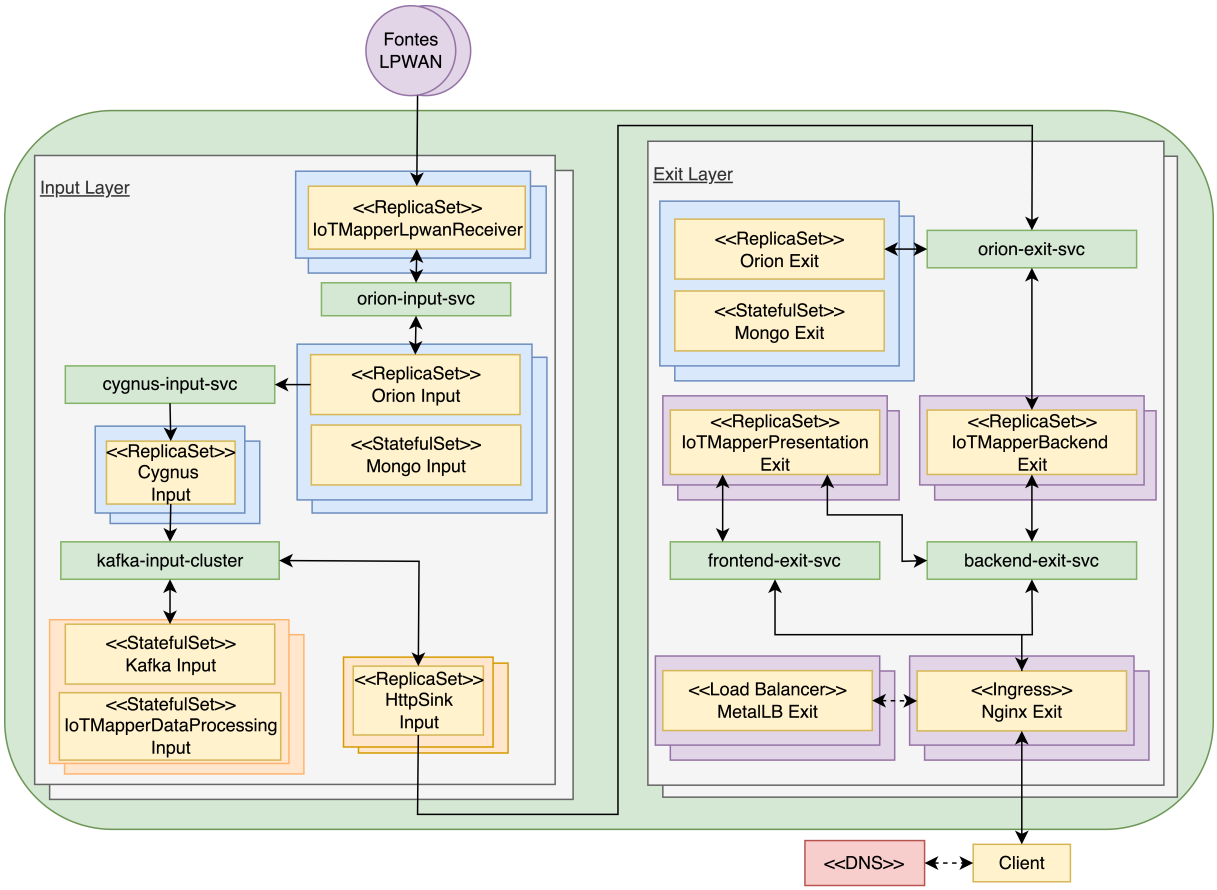


Figura 3.6: Visão da Arquitetura de Implementação sobre Kubernetes.

- O uso de *Services* como identificação interna dos componentes, no estabelecimento de ligações de rede. Este mecanismo oferece, ainda, balanceamento interno entre instâncias de um componente identificado pelo mesmo *Service*;
- Componentes que requerem manutenção de estado, como o OCB, são inicializados como *Stateful Services*;
- Componentes sem manutenção de estado, como o *IoTMapperPresentation*, são inicializados como *Deployments* resultando na inicialização de *Replica Sets*;
- O uso de um *Load Balancer* e *Ingress* como apresentado na Secção 3.8.2.

O processo de instalação e iniciação do protótipo foi unificado sob um único *playbook* Ansible [96], permitindo a instanciação de toda a plataforma com um único comando.

3.8.1 Fluxos de Dados

Ao considerar a identificação de fluxos de dados no sistema da arquitetura proposta, podem ser identificados dois fluxos:

- Um fluxo de entrada: desde a receção dos *MetricsReports* até à criação e atualização dos *MetricsAggregations*;
- Um fluxo de saída: que permite o consumo das agregações por intermédio do *IoTMapperBackend* e *IoTMapperPresentation*.

Ambos os fluxos têm o OCB como o ponto de ligação comum. No entanto, representam dois casos de utilização significativamente distintos. O fluxo de entrada é um cenário com maior peso por parte de ações de escrita, com novos relatórios de métricas a chegar e atualizações das agregações, fazendo adicionalmente uso de notificações assíncronas do NGSI-v2. O fluxo de saída é quase exclusivamente composto por leituras das métricas agregadas a partir do contexto gerido.

Considerando os dois fluxos de utilização diferentes sob a mesma arquitetura, para efeitos de integração da implementação, foi considerada a separação do sistema em duas camadas distintas. Cada camada possuindo a sua própria instância do OCB. De tal forma que se tem:

- **Input Layer:** para o fluxo de entrada. Possuindo os *clusters* FIWARE e Kafka;
- **Exit Layer:** para o fluxo de saída. Possuindo o *cluster* Visualização e os componentes de balanceamento apresentados seguidamente (Secção 3.8.2).

3.8.2 Balanceamento de Carga

Tendo como um dos objetivos a garantia da Escalabilidade da plataforma, e considerando a arquitetura já apresentada, é importante considerar o efeito de variações no volume de carga a que a plataforma é sujeita. Em particular considera-se a variação de volume de carga resultante de um maior volume de relatórios de métricas e acessos as agregações de métricas. Ambos estes aspetos podem ser considerados como desafios de balanceamento de carga, correspondendo, respetivamente, a *Input Layer* e *Exit Layer*.

A separação já descrita na Secção 3.8.1, implicitamente, apresenta uma primeira resposta ao escalonamento da plataforma. Sendo, nas secções seguintes, identificados mecanismos para cada camada.

3.8.2.1 Variação do Volume na *Input Layer*

A *Input Layer* é responsável pela receção de relatórios de métricas, criando entidades *MetricsReports*, e pelo seu processamento no Apache Kafka, resultando na criação de *MetricsAggregations*.

A variação de volume é, previsivelmente, resultado de variações no número de relatórios de métricas recebidas pelos *IoTMapperLpwanReceiver*. No caso de implementações baseados nos IoTAgents, incluindo a referência do IoTAgent-LoRaWAN, o mecanismo de recolha é baseado na recolha das métricas. Como tal, o aumento do volume de relatórios não aumenta a pressão direta sobre o sistema, para alem do que os IoTAgents são capazes de recolher. Após os *IoTMapperLpwanReceiver*, o balanceamento de carga é realziado internamente pela implementação Kubernetes.

Como tal, o primeiro passo necessário é a determinação do volume que os IoTAgents, ou, outras implementações dos *IoTMapperLpwanReceiver* são razoavelmente capazes de processar. Em função desta capacidade, e do volume previsto para cada rede LPWAN, pode ser guiado o trabalho de dimensionamento da *Input Layer*.

O balanceamento interno, entre componentes, possui duas vertentes. Primeiramente, os *Services* do Kubernetes distribuem os pedidos entre todas as instâncias de componente associados a um *Service*. No caso de um *Stateful Set*, podem existir *Services* para cada instância, como no caso do Apache Kafka. A segunda vertente é o resultado dos mecanismos internos do Apache Kafka, já explorados na Secção 3.4.

No Capítulo 4 são levados a cabo testes sobre a capacidade de resposta da *Input Layer*.

3.8.2.2 Variação do Volume na *Exit Layer*

A *Exit Layer* é responsável pela disponibilização do conjunto de *MetricsAggregation*, não existindo criação de dados. Nesta camada o balanceamento de carga pode ser realizado por mecanismo comuns de *cache* e distribuição de conteúdos típicos de aplicações Web [97, 98].

Seguindo os princípios de implementação arquitetural do Kubernetes é necessário a disponibilização de um serviço balanceador de carga. É também importante garantir que o *IoTMapperPresentation* e *IoTMapperBackend* podem ser acedidos através do mesmo endereço Internet Protocol (IP).

As necessidades indicadas podem ser colmatadas por uma regra de encaminhamento do tipo *Ingress*, suportada por um *Ingress Controller* [92–94]. A implementação de referência, a única suportada oficialmente sem dependências de vendedores *cloud* específicos, é o Ingress NGINX Controller [99]. Este cria uma instância do balanceador NGINX [100].

Recebidos os pedidos, o NGINX é responsável por encaminhar cada um para os *Services* associados a *IoTMapperPresentation*, ou, a *IoTMapperBackend*. Os próprios *Services* oferecem balanceamento de carga entre múltiplas instâncias dos componentes. Adicionalmente, pode ser feita utilização dos mecanismos de *cache* oferecidos para reduzir a

pressão sobre os componentes implementados.

Tabela 3.5: Níveis de *cache* considerados.

Nome	Duração por Omissão	Observações
Microcache	1 s	Conteúdo que muda frequentemente
Minicache	60 s	Conteúdo que pode mudar, requerendo confirmação pontual
Longcache	31536000 s	Conteúdo que se sabe que não sofre alterações

Seguindo as boas praticas conhecidas, foram definidos três níveis de *cache*, apresentados na Tabela 3.5, com os valores a definir para o cabeçalho "Cache-Control: max-age=< duração >"do HTTP. Tem-se:

- Para o conteúdo servido por *IoTMapperPresentation* para construção das páginas, sabe-se que a maioria dos componentes não se alteram, sendo que alterações resultam em novos Uniform Resource Locator (URL) na páginas principal. Estes conteúdos são marcados com *Longcache* e a páginas principal com *Microcache*. O ficheiro de configuração (*config.js*) é marcado com *Minicache*;
- Para o conteúdo servido por *IoTMapperBackend*, todas as listas de *MetricsAggregation* são marcadas com *Microcache*, e o conteúdo adicional com *Minicache*, conforme indicado no Apêndice D;

O *Ingress*, por si só, apenas expõe pontos HTTP e encaminha para os *Services* adequados. Para que possa ser alcançável por sistemas externos é necessário um balanceador de rede, com atribuição de IPs. Esta funcionalidade é, normalmente, desempenhada pela infraestrutura do vendedor *cloud*. Com o intuito de evitar dependências teve-se em consideração o *Load Balancer* MetalLB [101]. Este componente, configurado previamente com um conjunto de IPs alcançável através da infraestrutura de rede, responde a pedidos Address Resolution Protocol (ARP) e encaminha todos os pacotes recebidos para o *Ingress Controller* a que atribui-o o IP.

No Capítulo 4 são levados a cabo testes sobre a capacidade da *Exit Layer*.

4

Validação, Avaliação e Discussão de Resultados

4.1 Introdução

Concluído o estudo prévio das alternativas tecnológicas adequadas, seguido da planificação da arquitetura e subsequente implementação de referência, existe a necessidade de validar a arquitetura e avaliar o desempenho obtido pela implementação. Este trabalho foi realizado em ambiente experimental, recorrendo a conjuntos de dados recolhidos previamente para o ambiente urbano na cidade de Lisboa, Portugal, para a rede Long Range Wide-Area Network (LoRaWAN) instalada pelo município. A escolha deste ambiente deve a possibilidade de livro acesso a rede e existência de estudos prévio de soluções de Internet das Coisas neste ambiente [8].

4.1.1 Organização

O presente Capítulo encontra-se organizado em secções tais que:

- Secção 4.2: caracterização do conjunto de dados;
- Secção 4.1.1: caracterização da metodologia usada na validação e avaliação;
- Secção 4.4: resultado da validação da arquitetura;

- Secção 4.5: resultados da avaliação da implementação.

4.2 Conjunto de dados

No desenvolvimento dos testes foram utilizados dois conjuntos de dados interrelacionados.

O primeiro conjunto, designado *Data-01*, é composto por um conjunto de métricas recolhido no ambiente urbano da cidade de Lisboa, Portugal. A recolha foi realizada através de uma viatura em movimento, com recurso a rede LoRaWAN livremente disponibilizada pelo município e ligada a The Things Network (TTN), sendo um descritor fidedigno do ambiente disponível para desenvolvimento de soluções nesta cidade. O conjunto é composto pelas mensagens detalhadas não processadas, produzidas pela TTN, como exemplificado no Apêndice A, cada uma contendo a localização do equipamento associado no campo *decoded_payload*. O conjunto possui 2100 mensagens individuais, cada recebida por 2-3 Gateways, resultando em 3185 agregações (*MetricsAggregation*) individuais de métricas. Este conjunto foi recolhido nos dias 4 e 5 de Novembro de 2021.

O segundo conjunto foi extrapolado do primeiro, sendo designado *Data-02*. Mantendo os valores reais para métricas, foram geradas 10000 mensagens novas, semi-aleatoriamente, de tal forma que exista apenas uma mensagem para cada área geográfica (*Maptile*), garantido que cada mensagem resulta em apenas uma única agregação. Esta característica facilita a deteção de perdas de mensagens ao longo do processamento, sem interferência dos mecanismos de agregação e eliminação de mensagens fora de ordem.

Em conexão ao segundo conjunto de dados foi desenvolvido um componente de software auxiliar, designado simplesmente *InputGenerator*, que reproduz o conjunto de dados, com ritmos de escrita controláveis, inserindo os marcadores temporais atuais, assim como os identificadores das aplicações e equipamentos registados no componente IoTAgent-LoRaWAN. Para suporte a esta funcionalidade, os campos correspondentes nas mensagens, foram marcados com o padrão *{}*. A combinação deste conjunto de dados e do *InputGenerator* permite simular, em ambiente laboratorial, o efeito de volumes variáveis de mensagens recebidas de múltiplas localizações e dispositivos em simultâneo.

4.3 Metodologia de Validação e Avaliação

A metodologia de validação e avaliação escolhida baseia-se na separação do processo em duas fases. Na *Fase 1* procedeu-se a validação da arquitetura, sobre um ambiente real e configuração de exemplo. Na *Fase 2* foi realizada a avaliação do desempenho da implementação, para diferentes condições.

4.3.1 Condições Gerais de Teste

Para validação da arquitectura desenvolvida, o sistema foi integrado num ambiente de laboratorial, disponibilizado pelo Instituto Superior de Engenharia de Lisboa (ISEL), que utilizou três máquinas virtuais para instanciação da implementação de referência, conforme descrito na Tabela 4.1.

Tabela 4.1: Máquinas utilizadas na instanciação e validação.

Legenda	Descrição	CPU	Memória	Sistema Operativo
TFM-00	Executa a *Input Layer*	4 vCPU	16 Gb	Ubuntu 22.04 LTS (5.15.0-40-generic)
TFM-01	Executa a *Exit Layer*	4 vCPU	8 Gb	Ubuntu 22.04 LTS (5.15.0-40-generic)
TFM-02	Corre os testes de carga	4 vCPU	8 Gb	Ubuntu 22.04 LTS (5.15.0-40-generic)

A latência entre as máquinas foi medida e considerada insignificante, sempre inferior a 1 ms, numa experiência simples de utilização do utilitário *ping* entre as máquinas.

Foi utilizada a versão 1.24 do Microk8s [102, 103], escolhido como implementação de referencia para o Kubernetes devido ao reduzido peso de execução, para criar um aglomerado Kubernetes entre as TFM-00 e TFM-01, separando os componentes entre as duas máquinas de acordo a camada a que pertencem. A separação baseia-se num *label* inserido na descrição do nó associado a cada máquina, usado como seletor nas descrições dos componentes, na forma *node.iotmapper/layer:* < *camada* >.

Na inserção controlada do conjunto de dados foi escolhido o uso de um intermediador Message Queuing Telemetry Transport (MQTT), em particular o Eclipse Mosquitto [104], como simulação da infraestrutura externa disponibilizada pela TTN. Este intermediário adicional foi adicionado a máquina TFM-01 durante a Fase 1 e durante os testes ao fluxo de entrada da Fase 2.

4.3.2 Condições para a Fase 1

A validação individual de cada componente foi levada a cabo a mediada que o sistema foi implementado. Tendo-se com sucesso verificado o correto funcionamento individual dos componentes foi realizada a integração dos componentes no ambiente descrito em 4.3.1, seguido da sua validação *ponta-a-ponta*, isto é, desde o consumo de relatórios de métricas (*IMetricsReport*) ao consumo das agregações (*IMetricAggregation*) por um utilizador final.

Com este intuito foi utilizado o conjunto de dados *Data-01*, já apresentado. Este conjunto, que representa as condições reais de propagação, foi inserido no intermediário MQTT a um ritmo ilustrativo de 100 mensagens/segundo. Os resultados obtidos são apresentados na Secção 4.4.

4.3.3 Condições para a Fase 2

4.3.3.1 Condições para Testes Direcionados ao Fluxo de Entrada

Com o conjunto de dados *Data-02*, anteriormente descrito, o sistema foi sujeito a diferentes volumes de escrita de mensagens. Sendo recolhidas duas métricas de desempenho do sistema, o Average Latency (AL) e o Total Error Rate (TER).

O AL indica o atraso médio, em milissegundos, para que o grupo de mensagens processadas com sucesso atinja o componente indicado. A definição é dada pela Equação 4.1.

$$AL = \frac{\sum_{t=1}^{n} |f_t - i_t]|}{n} \tag{4.1}$$

onde f_t são os marcadores temporais, em milissegundos, inseridos no componente em validação, i_t os marcadores temporais inseridos durante a criação inicial de cada mensagem, e n é, novamente, o número de mensagens enviadas para o sistema.

Na medição do AL foi considerado a medição da latência tanto para o fluxo de mensagens de *ponta-a-ponta*, como para dois pontos de medição adicionais, conforme definido na Tabela 4.2 e descrito visualmente na Figura 4.1.

Tabela 4.2: Mapeamentos para legendas de figuras, definindo pontos de medição para a latência.

Legenda	Designação	Descrição
EE	Ponta-a-Ponta	Latência medida desde a criação da mensagem até ser escrita como uma agregação no Orion Context Broker (OCB)
C1	Componente 1	Latência medida no momento em que o IoTAgent-LoRaWAN termina de enviar cada mensagem ao Orion Context Broker (OCB)
C2	Componente 2	Latência medida no momento em que cada agregação é escrita ao Apache Kafka. Correspondendo ao *LogAppendTime*, usado internamente pelo Apache Kafka
D1	Diferença 1	Diferença nas latências medidas em EE e C1. Corresponde a latência para todos os componentes, excluindo IoTAgent-LoRaWAN
D2	Diferença 2	Diferença nas latências medidas em C1 e C2. Corresponde a latência gasta no processamento da mensagem numa agregação
D3	Diferença 3	Diferença nas latências medidas em EE e C2. Corresponde a latência gasta na inserção de cada agregação no OCB

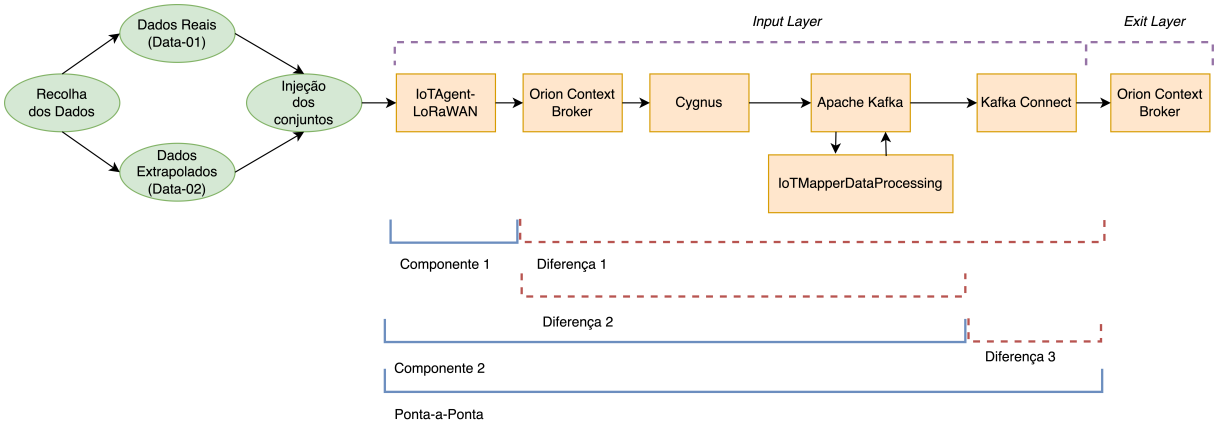


Figura 4.1: Metodologia de avaliação, com identificação dos pontos de medição de AL e ALD

.

A informação presente na Tabela 4.2 é utilizada nas legendas das figuras da Secção 4.5.1. Os pontos de medição foram escolhidos com o objetivo principal de testar o desempenho

do IoTAgent-LoRaWAN, escolhido como implementação do IoTMapperLpwanReceiver na integração de redes LoRaWAN. A escolha foi ainda resultado da disponibilidade dos dados necessários em cada componente, nomeadamente simples acesso a marcadores temporais internos. Sendo que existem apenas três pontos de medição possíveis identificados, o conjunto de métricas obtidas foi reforçado através do cálculo das diferenças médias entre os pontos de medição. Sendo calculadas como resultado de AL, são designadas AL-Diference (ALD). O uso destas métricas compostas permite a identificação do tempo gasto em subconjuntos de componentes não monitorizados diretamente.

O TER indica a percentagem de mensagens que ou foram descartadas pelo sistema, ou, que não foram totalmente processadas num período de tempo razoável. A definição é dada pela Equação 4.2.

$$TER = \frac{S_i}{n} \times 100 \tag{4.2}$$

onde S_i é a contagem das mensagens processadas com sucesso em i segundos e n é o número de mensagens enviadas para o sistema. Na medição do TER foi considerado apenas as perdas ponta-a-ponta, correspondendo ao ponto EE na Tabela 4.2.

4.3.3.2 Condições para Testes Direcionados ao Fluxo de Saída

Com o conjunto de dados *Data-01*, anteriormente descrito, o sistema foi sujeito a diferentes volumes de pedidos de leitura das listas de *MetricsAggregations* e do conteúdo que compõem a visualização apresentada por *IoTMapperPresentation*. O *broker* MQTT não foi incluindo neste conjunto de testes.

O volume de pedidos foi realizado a partir da máquina TFM-02, recorrendo ao Apache JMeter [105]. O Apache JMeter é uma ferramenta criado para realizar testes de carga sobre recursos diversos, no caso em estudo sobre a forma de pedidos Hypertext Transfer Protocol (HTTP). Em todas iterações o Apache JMeter foi configurado com 300 *threads*, um período de inicialização de 5 segundos de forma a realizar 300 000 pedidos.

Como conjunto principal de métricas, no final de cada iteração foram recolhidas as métricas apresentadas pelo Apache JMeter, em particular:

- **Erros**: percentagem de pedidos HTTP que falharam, ou, devolveram código de erro;
- **Latência**: latência média de resposta, ou erro, para cada pedido;

- **Pedidos/segundo**; ritmo efetivo de pedidos realizados por segundo. Onde um pedido é considerado como completo, se concluir com sucesso ou com erro.

Um segundo conjunto auxiliar de métricas foi obtido pela monitorização da máquina TFM-01. Foi adicionado ao *cluster* Kubernetes uma instância do Prometheus, uma infraestrutura para recolha automatizada de métricas. Combinado com o uso de CAdvisor, que recolhe métricas de sistema sobre os contentores, foi possível, obter:

- **Utilização de CPU** : utilização média, em percentagem, de Central Processing Unit (CPU) na máquina;
- **Utilização de Memória**: utilização média, em percentagem, de memória na máquina;
- **Utilização da Interface de Redes**: utilização média, em Megabytes, de recursos de rede.

4.4 Fase 1 - Validação da Arquitectura

Considere-se a Figura 4.2, que apresenta a interface gráfica do utilizador centrada no centro da cidade de Lisboa. A interface gráfica apresenta ao utilizador um mapa de calor, demonstrando a qualidade relativa dividida pelas *MapTiles* para as quais foi registado cobertura. A barra lateral esquerda permite ao utilizador selecionar o conjunto de dados apresentado a partir dos menus de seleção. O *LPWAN* seleciona as Low-Power Wide-Area Network (LPWAN), enquanto que o *Metric* seleciona a métrica agregada. No caso presente para a Figura 4.2, a combinação foi o RSSI para a rede LoRaWAN.

Nesta fase foi possível verificar o correto funcionamento da implementação, sem perda detetada de mensagens com relatórios de métricas, verificando o design arquitetural.

4.5 Fase 2 - Avaliação da Implementação

Em seguimento dos procedimentos de avaliação definidos na Secção 4.3.3, foi levado ao cabo de avaliação do desempenho obtido pela implementação, sendo os seus resultados apresentados.

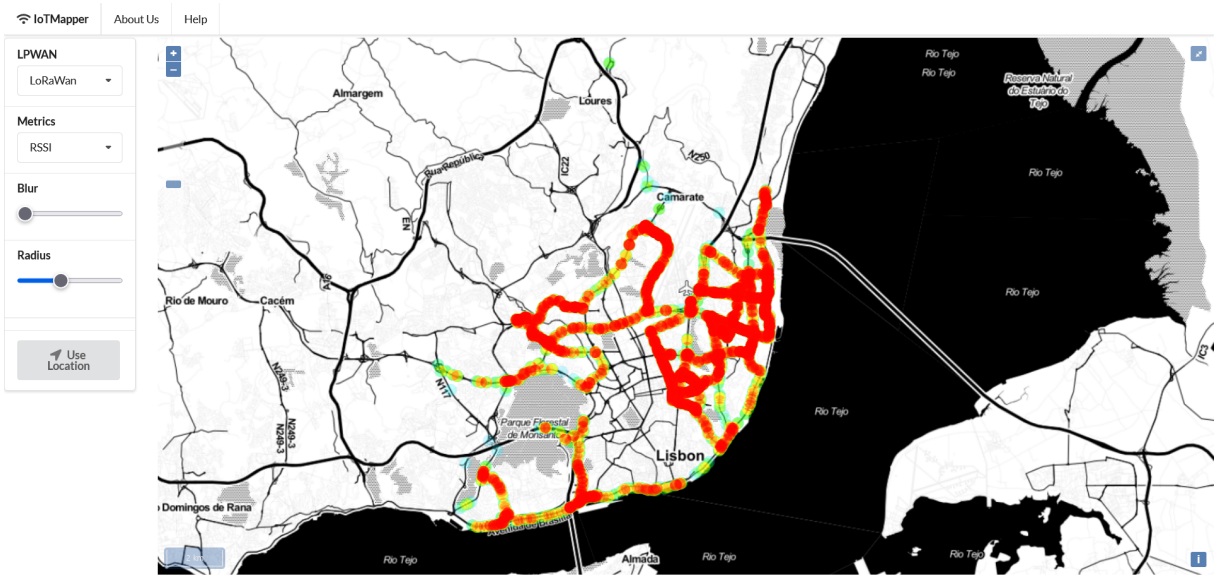


Figura 4.2: Resultado da Fase 1 (validação), apresentando a cobertura recolhida para a cidade de Lisboa, Portugal.

4.5.1 Testes direcionados ao Fluxo de Entrada

Para as subsecções seguidamente apresentadas, o conjunto de testes realizados foram os mesmos para ambas, sendo os resultados para a latência e as perdas apresentados separadamente.

4.5.1.1 Latência

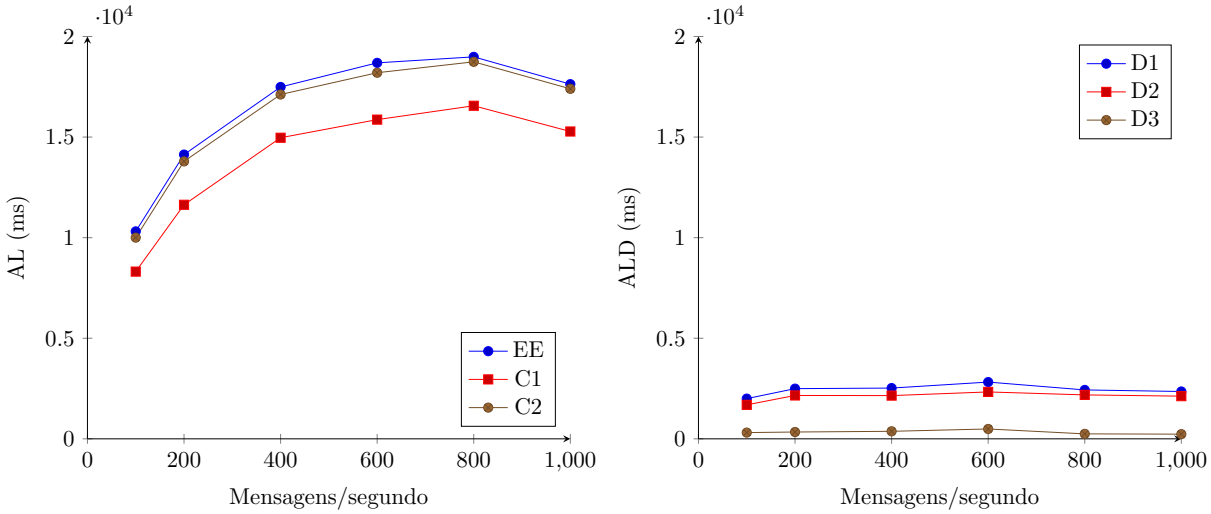
A Figura 4.3 mostra a variação da Latência Média (AL), como definida na Equação 4.1, para os três pontos de medição e diferenças médias entre eles, como definido na Tabela 4.2.

Comparando os valores registados, é evidente que a maioria da latência no sistema é introduzida pela interação entre o primeiro componente do sistema, o IoTAgent-LoRaWAN, e o OCB, correspondente ao ponto de medição C1. C2 e EE, que correspondem aos pontos de medição ao longo do fluxo de processamento, para as mesmas mensagens, mostram apenas pequenos aumentos na latência. É também claro que a latência aumenta significativamente à medida que o número de mensagens/segundo aumenta. Este padrão é ligeiramente contrariado para 1000 mensagens/segundo.

Embora os requisitos listados para o projeto permitissem sacrifícios no tempo de espera para disponibilizar as entidades *MetricsAggregation* atualizadas para consumo, como mecanismo para limitar a perda de mensagens, o crescente atraso introduzido pelo componente IoTAgent-LoRaWAN não é considerado como estando abrangido pelas

concessões feitas neste requisito, e apresenta um ponto de limitação significativo para a Fiabilidade e Escalabilidade do sistema.

Como o IoTAgent-LoRaWAN não utiliza qualquer mecanismo de agrupamento, cada mensagem recebida resulta numa nova, individual, escrita na entidade subjacente armazenada no OCB. Na primeira iteração do teste (Figura 4.3), todas as mensagens recebidas pertenciam ao mesmo dispositivo simulado, sendo mapeadas pelo componente IoTAgent-LoRaWAN para a mesma entidade.



(a) Latências registadas para componentes monitorizados e de ponta-a-ponta.

(b) Diferenças nas latências registadas entre componentes.

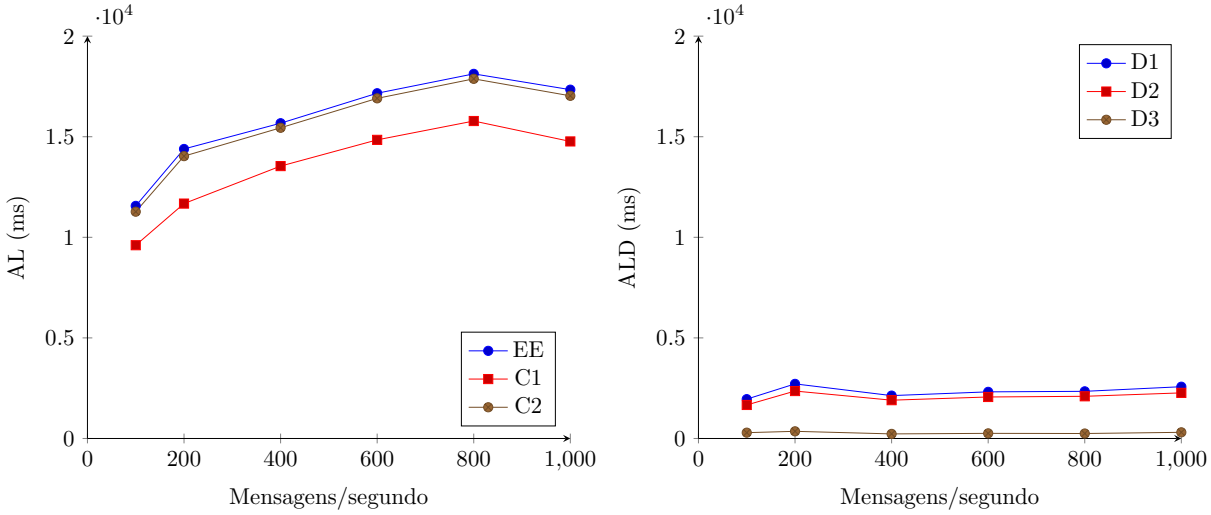
Figura 4.3: AL e ALD registados para a segunda fase de avaliação. LoRaWAN para 1 dispositivo simulado.

A Figura 4.4 repete a experiência anterior, mas, configurando o componente IoTAgent-LoRaWAN com 100 dispositivos simulados. Enquanto se escreviam as mensagens, estas eram atribuídas aleatoriamente a um dispositivo diferente.

Esta modificação não resultou em qualquer ganho significativo, com repetição dos padrões previamente notados. A interação inicial (até ao ponto C1) representa entre 9606,834 ms e 14765,746 ms de latência, enquanto todos os restantes componentes, que incluem a maioria do processamento e interações, representam apenas entre 1955,563 ms e 2714,426 ms de latência até que cada agregação esteja disponível para consumo. É de salientar que, como se vê na Figura 4.4b, as diferenças nas latências medidas são estáveis para todas as taxas de mensagem, indicando que o crescimento observado da latência de ponta a ponta é principalmente o resultado de um comportamento instável entre o IoTAgent-LoRaWAN e o OCB.

A redução da latência média para 1000 mensagens/segundo é novamente confirmada, mas reduzida de 1356,975 ms para 785,422 ms. Esta inversão, para ambos os cenários,

tem sido interpretada como resultado do tamanho utilizado para agrupamento interno (formando um *batch*) por omissão. Sendo 1000 mensagens tanto para os componentes Cygnus e Kafka Connect, diminuindo as latências de cauda. Esta observação é suportada, na secção seguinte, pela Figura 4.6a, onde a percentagem de mensagens com latência total superior a 25 segundos é reduzida de 17,30% para 6,60%, enquanto a latência para períodos de tempo mais curtos permanece inalterada.



(a) Latências registadas para componentes monitorizados e de ponta-a-ponta.

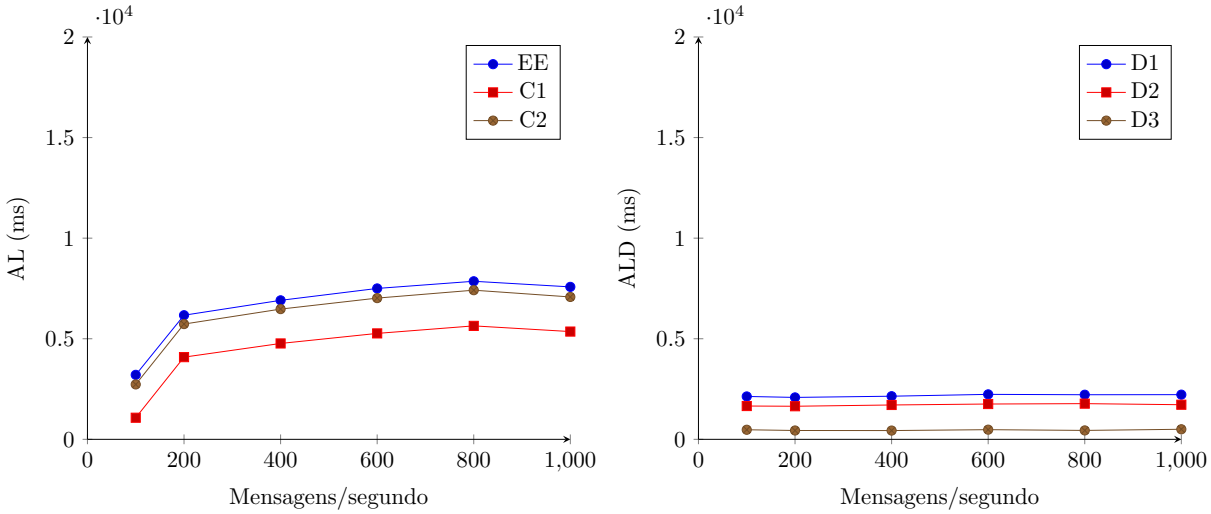
(b) Diferenças nas latências registadas entre componentes.

Figura 4.4: AL e ALD registados para a segunda fase de avaliação. LoRaWAN para 100 dispositivos simulados.

Focando o estudo da redução da latência no componente IoTAgent-LoRaWAN, e considerando que os resultados anteriores, sugerem, que o Orion Context Broker deveria ser capaz de lidar com taxas de mensagens mais elevadas [8, 106], foi considerado um terceiro cenário em que foram utilizadas 3 instâncias do componente IoTAgent-LoRaWAN, cada uma lidando com um terço do volume de mensagens e comunicando independentemente com o OCB. Novamente, foram considerados 100 dispositivos simulados. Os resultados são apresentados na Figura 4.5.

Este cenário, embora semelhante aos resultados anteriores, apresenta uma redução significativa das latências para o ponto de medição C1, e, consequentemente, para a latência total do sistema, como verificado em EE e C2. No entanto, as instâncias IoTAgent-LoRaWAN continuam a ser o principal ponto de limitação do desempenho.

A latência no ponto C1 situa-se, agora, entre 1074,035738 ms e 7577,968 ms, uma diminuição na ordem de 8,945 e 2,756 vezes, respetivamente, em comparação com o cenário da Figura 4.4. Os restantes componentes representam entre 2083,829903 ms e 2235,426207 ms de latência.



(a) Latências registadas para componentes monitorizados e de ponta-a-ponta.

(b) Diferenças nas latências registadas entre componentes.

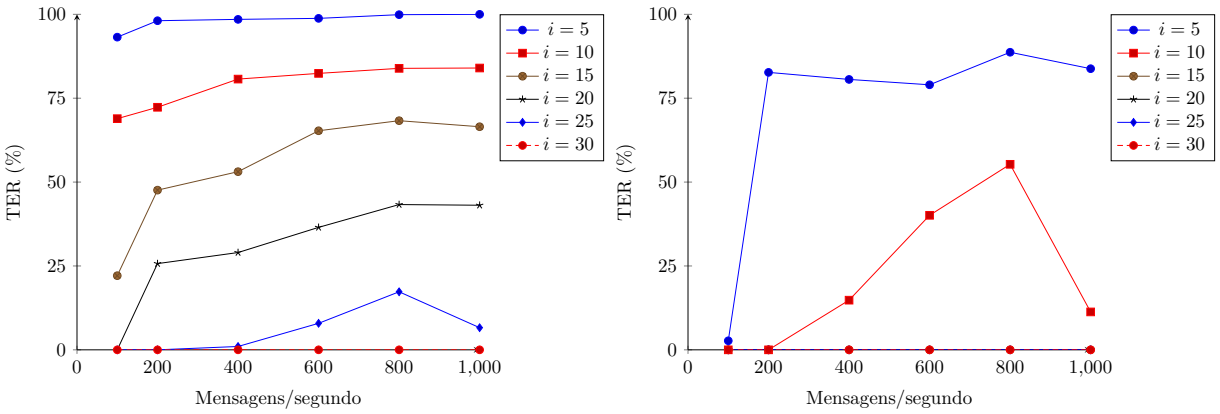
Figura 4.5: AL e ALD registados para a segunda fase de avaliação. LoRaWAN para 100 dispositivos simulados, e 3 instâncias do IoTAgent-LoRaWAN.

Embora a diminuição da latência para 1000 mensagens por segundo ainda esteja presente, não é tão significativa como nos casos anteriores, com apenas 278,664 ms para os 785,422 ms verificados na Figura 4.4.

4.5.1.2 Taxas de Erros/Perdas

A Figura 4.6 mostra as taxas de erro (TER) para o mesmo conjunto de experiências que a Figura 4.4 e a Figura 4.5, mas agora mostrando as taxas de erro, tais como definidas na Equação 4.2, para diferentes valores de i segundos. Como já apresentado, o TER é considerado como o número de mensagens que o sistema não entrega em i segundos, e, portanto são consideradas perdidas. Para todos os testes não houve perda total de mensagens, isto é, uma mensagem que não foi entregue em qualquer período de tempo.

Comparando os resultados obtidos para as Figuras 4.6a e 4.6b, as reduções nas latências anteriormente obtidas, aumentando o número de instâncias do IoTAgent-LoRaWAN de 1 para 3, resultaram em reduções previsíveis nas taxas de erro. Em particular, não foram entregues mensagens após o limite de tempo, para $i = 15$, enquanto que, anteriormente, isto apenas era conseguido para $i = 30$. Para 1000 mensagens/segundo, na Figura 4.6a, há uma clara redução nas latências de cauda, isto é, mensagens com latências de entrega superiores a 20 segundos. O mesmo efeito é também notado, na Figura 4.6b, para 10 segundos, com uma queda mais significativa de 55,30% para 11,30%. Como referido na secção anterior, esta queda foi interpretada como o resultado da dimensão de *batch* internos utilizado pelos componentes Cygnus e Kafka Connect.



(a) 100 dispositivos simulados e 1 IoTAgent-LoRaWAN. Como na Figura 4.4.

(b) 100 dispositivos simulados e 3 IoTAgent-LoRaWAN. Como na Figura 4.5.

Figura 4.6: TER registado para a segunda fase de avaliação.

Considerando o requisito inicialmente definido, que o sistema não poderia falhar a entrega de mais de 1% das mensagens, todos os cenários poderiam ser considerados como tendo cumprido o requisito, dado um limite máximo de 30 segundos de atraso. Mas falham este requisito considerando um limite de 5 segundos. Mesmo o melhor caso obtido, a marca de 100 mensagens/segundo para a Figura 4.6b, tinha um TER de 2,7%.

4.5.2 Testes direcionados ao Fluxo de Saída

O objetivo principal na avaliação do Fluxo de Saída é validar a viabilidade da implementação usada na *Exit Layer*. Em particular, os ganhos de desempenho resultantes dos mecanismos de *cache* oferecidos pelo *Ingress Controller* do NGINX. É feito, ainda, o contraste destes mecanismos com o uso de um maior número de instâncias dos componentes da *Exit Layer*.

Para a avaliação do desempenho da *Exit Layer* foram definidos os cenários apresentados na Tabela 4.3, organizados em três series. A série B define a base de comparação, sem otimizações. A série R demonstra o uso de múltiplas instâncias. A série Ca demonstra o uso dos mecanismos de *cache*. Os cenários B1, R1 e Ca1, pretendem obter um ponto de comparação entre as séries, pedindo apenas a página principal, o menor elemento que pode ser pedido. Os cenários B2, R2 e Ca2 pretendem simular o acesso a Web Application Programming Interface (API) de *IoTMapperBackend*, por sistemas externos interessados apenas na lista de *MetricsAggregation*. Por último, os cenários B3, R3 e Ca3 simulam o acesso a *MetricsPresentation* por utilizadores finais, requerendo para alem da página principal e da lista de *MetricsAggregation* de um conjunto adicional de elementos.

Para efeitos de comparação com os restantes cenários, em B3, R3 e Ca3, considera-se um

"pedido"como a obtenção de todos os recursos que compõem a visualização a apresentar ao utilizador. Este "pedido"é composto por múltiplas ligações HTTP.

Naturalmente, a dimensão dos recursos pedidos têm influência nos resultados obtidos. A dimensão total, a que correspondem os elementos do site implementado por *IoTMapperPresentation*, é 491,3 Kilobytes, onde apenas apenas 1,89 Kilobytes são da página principal. A lista com as primeiras, por ordem de inserção, 1000 entidades do tipo *MetricsAggregation* tem uma dimensão de 690 Kilobytes, para o conjunto de dados *Data-01*.

Tabela 4.3: Cenários para o fluxo de entrada.

Cenário	Conteúdo	Cache	Instâncias
B1	Apenas da página principal.	Não	1 para cada componente
B2	Apenas da lista de agregações, com 1000 agregações	Não	1 para cada componente
B3	Todo o conteúdo, com 1000 agregações	Não	1 para cada componente
R1	Apenas da página principal	Não	3 para cada componente
R2	Apenas da lista de agregações, com 1000 agregações	Não	3 para cada componente
R3	Todo o conteúdo com 1000 agregações	Não	3 para cada componente
Ca1	Apenas da página principal	Sim	1 para cada componente
Ca2	Apenas da lista de agregações, com 1000 agregações	Sim	1 para cada componente
Ca3	Todo o conteúdo, com 1000 agregações	Sim	1 para cada componente

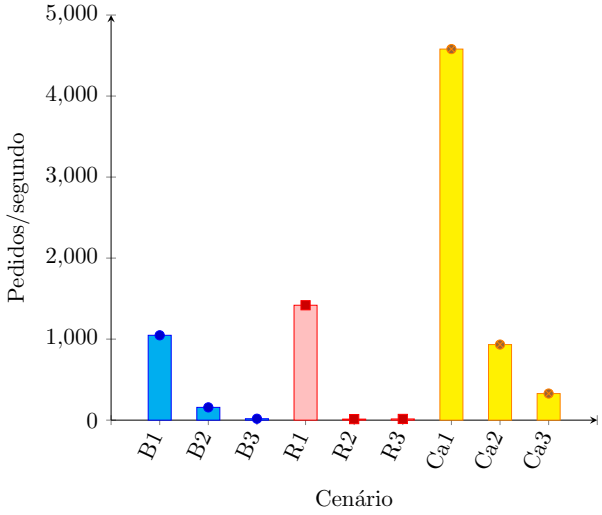
Tomados os cenários descritos, considere-se a Figura 4.7, que apresenta as métricas de desempenho.

Como expectável, os cenários como maior ritmo de pedidos completos são os com menor latência na resposta aos pedidos, estabelecendo de uma relação inversa. Igualmente, os cenários com valores elevados de latência media, possuem as maiores taxas de erro, chegando até 30.61 % de erro para R2. Sendo R2 o cenário com um nível extremo de latência (21178 ms). Os cenários da série Ca apresentam valores significativamente reduzidos de latência, com um máximo de 900 ms para Ca3.

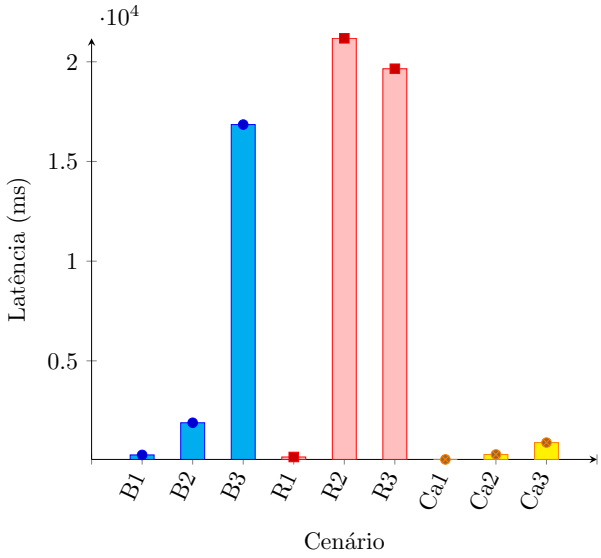
É possível verificar um padrão adicional para a série B para a série Ca. O melhor desempenho é obtido para os cenários que pedem apenas o conteúdo da página principal (B1 e Ca1), seguidos dos pedidos direcionados apenas a Web API (B2 e Ca2), e por último pelos pedidos para tudo o conteúdo necessário a interface gráfica (B3 e Ca3).

A série R não segue este padrão por uma pequena diferença. O cenário R2 possui o pior desempenho com 12.8 pedidos/segundo para os 14.72 pedidos/segundo de R3.

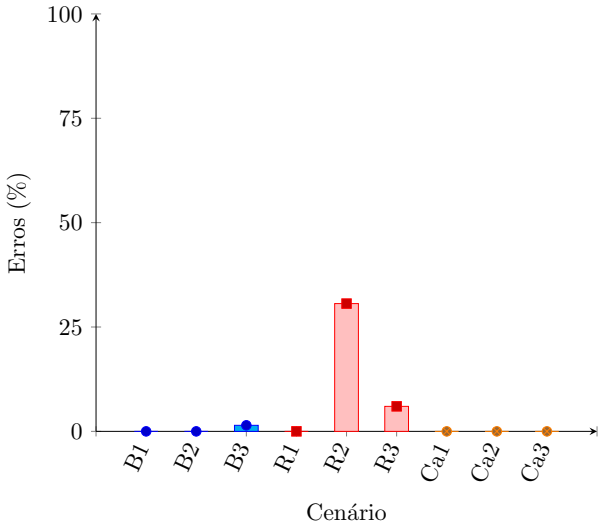
Nota-se que o uso dos mecanismo de *cache* do NGINX apresenta melhorias significativas no número de pedidos completos. Respondendo até 4579.9 pedidos/segundo para Ca1, uma melhoria de 4.373 e 3.230 vezes em comparação com B1 e R1, respetivamente. Esta melhoria é ainda mais marcada nos cenários subsequentes. Ca2 melhora 72.875 e 5.919 vezes para B2 e R2, e Ca3 melhora 18.813 e 22.289 vezes para B3 e R3.



(a) Pedidos completos, em média por segundo.



(b) Latência média dos pedidos.

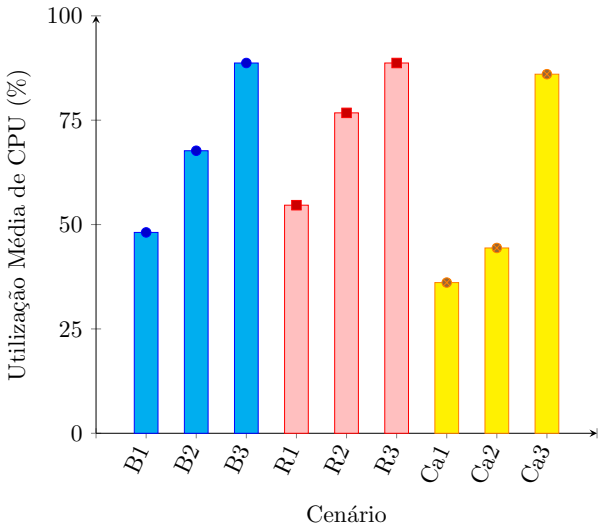


(c) Percentagem de pedidos completos com erro.

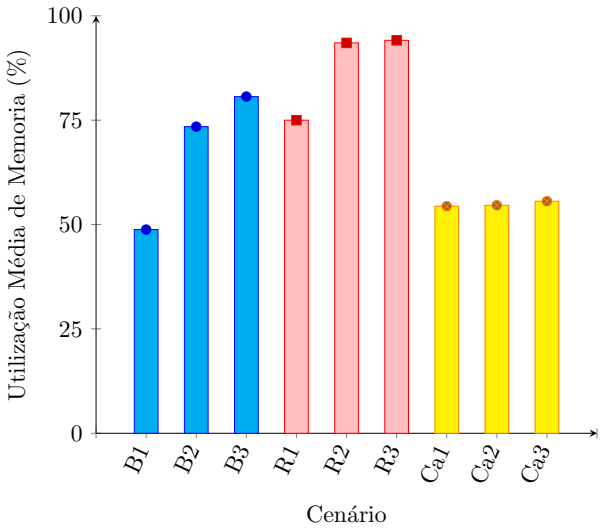
Figura 4.7: Desempenho da *Exit Layer* para os cenários identificados.

Considere-se ainda a Figura 4.8, que apresenta as métricas de uso de recursos na

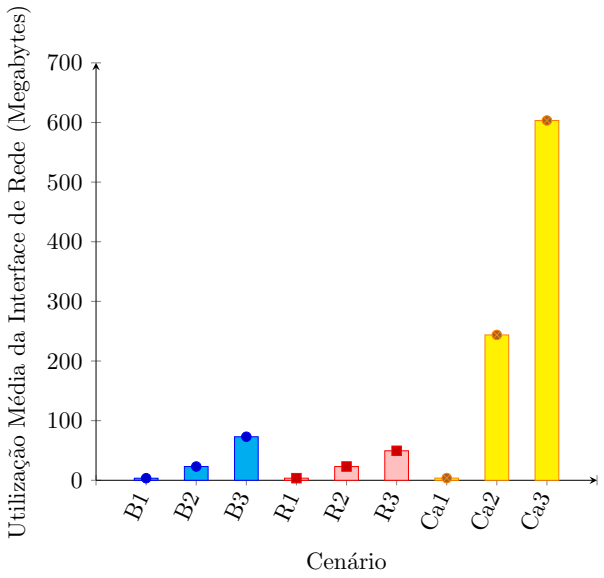
máquina TFM-01.



(a) CPU.



(b) Memória.



(c) Interface de Rede.

Figura 4.8: Uso de recursos pela *Exit Layer*, medido na máquina TFM-01, para os cenários identificados.

A utilização de recursos CPU e de memória demonstra, claramente, uma utilização mais eficiente de recursos na série com *cache* (Ca1 a Ca3). Em particular, o uso de memória não varia significativamente.

Em termos da utilização de recursos de rede, nota-se a existência de um padrão. Tendencialmente, a maior utilização de recursos de rede ocorre nos cenários com menor número de pedidos completos.

Em todas as séries e para todos os recursos o uso mais intensivo é na resposta aos pedidos dos cenários B3, R3 e Ca3. Nestes cenários os pedidos necessitam de obter múltiplos recursos distintos, incluindo a lista de agregações. Como tal, existem múltiplas ligações HTTP abertas para concluir cada pedido.

O seguinte conjunto de cenários com uso mais intensivo de recursos são B2, R2 e Ca2. Nestes cenários é pedido apenas a lista de agregações, no entanto, este é o elemento com maior dimensão a ser transferido.

Os resultados analisados permitem concluir que a solução para o fluxo de saída, a *Exit Layer* é viável como mecanismo para disponibilização dos *MetricsAggregation*, quando usados os mecanismos de *cache* do NGINX. O recurso a múltiplas instâncias é excessivamente intensivo para o ambiente de integração utilizado.

5

Conclusões e Trabalho Futuro

Ao longo do desenvolvimento do projeto apresentado foi possível estudar as metodologias e tecnologias candidatas à construção de um sistema capaz de obter métricas sobre a qualidade de redes Low-Power Wide-Area Network (LPWAN). Em resultado do estudo foi proposta uma arquitetura para recolha, processamento e apresentação de visualizações de métricas. A arquitetura proposta resultou numa implementação de referência. Por último, foi possível validar a aplicabilidade da arquitetura e avaliar o desempenho da implementação.

A arquitetura apresentada no Capítulo 3 define um sistema construído em torno de componentes pertencentes ao projeto FIWARE, considerando mecanismos existentes para a sua integração com fontes de dados e sistemas de processamento externos. A arquitetura foi dividida em três agrupamentos (*clusters*) lógicos de componentes: o *cluster* FIWARE, *cluster* Apache Kafka e o *cluster* Visualização. Cada um correspondendo a um de três eixos tecnológicos principais que guiam a proposta arquitetural. Sendo, respetivamente, o uso de componentes do projeto FIWARE para Contextualização e Gestão das métricas, uso de componentes do universo Apache Kafka para Distribuição e Processamento das métricas e agregações, e uma camada de apresentação para interface e visualização das agregações, implementada com Spring Framework e React.

No melhor conhecimento do autor, não existe uma solução prévia que utilize componentes FIWARE para recolha de métricas sobre as próprias implementações LPWAN, por oposição, a recolha de métricas sobre outros sistemas.

No *cluster* FIWARE tem-se o uso do Orion Context Broker (OCB) como componente

central gestor do contexto gerado. As métricas oriundas de redes LPWAN são recebidas por recetores *IoTMapperLpwanReceiver*. As métricas recebidas são armazenadas como entidades NGSI-v2, do tipo *MetricsReport*, um novo modelo de dados proposto no Apêndice C.1. No suporte a redes Long Range Wide-Area Network (LoRaWAN) foi escolhido o IoTAgent-LoRaWAN como referência e objeto de estudo na implementação dos *IoTMapperLpwanReceiver*. O FIWARE Cygnus, após notificação assíncrona emitida pelo OCB, insere os dados recolhidos no Apache Kafka.

O *cluster* Apache Kafka é responsável pela distribuição e processamento dos *MetricsReport*, incluindo resultados intermédios, com oferta de garantias de entrega, tolerância a falhas e volumes variáveis de dados. O processo de distribuição é integrado com processamento oferecido pela biblioteca Kafka Streams. Em particular, o componente *IotMapperDataProcessing*, implementa um fluxo contínuo de processamento, resultando em agregações das métricas, respeitando o modelo NGSI-V2 *MetricsAggregations* proposto no Apêndice C.2. Cada agregação corresponde a um par <*gwId*,*mapTile*>. Os *gwId* correspondem aos identificadores disponíveis em cada rede LPWAN. Os *mapTile* correspondem as GeoHashes do local onde a medição foi efetuada. As agregações são inseridas no OCB por um componente *HTTPSink*, implementado através do mecanismo Kafka Connect.

O *cluster* Visualização disponibiliza o acesso intermediado as entidades *MetricsAggregations*. O componente principal é o *IoTMapperBackend*, uma implementação sobre Spring Framework, de uma Web Application Programming Interface (API) RESTfull aberta e documentada no modelo OpenAPI 3, no Apêndice D. Como interface para visualização implementou-se o *IoTMapperPresentation*, uma interface React, com composição gráfica através de Semantic UI e RLayers, para apresentação de um Mapa de Calor.

Em termos de integração dos *clusters*, foram identificados dois fluxos distintos na passagem de dados: um fluxo de entrada, para o consumo de métricas (*MetricsReport*), e um fluxo de saída, para consumo das agregações (*MetricsAggregation*). Estes fluxos resultaram na separação da arquitetura em duas camadas semi-independentes, uma camada de entrada (*Input Layer*) e uma camada de saída (*Exit Layer*). Cada camada possui a sua própria instância do Orion Context Broker (OCB), sendo as métricas recebidas na *Input Layer* assincronamente fornecidas a *Exit Layer*. A *Input Layer* agrupa, então, os *clusters* FIWARE e Kafka. A *Exit Layer* agrupa o *cluster* Visualização é uma instância do OCB.

A validação da arquitetura foi realizada com sucesso, utilizando dados reais recolhidos no ambiente da cidade de Lisboa, Portugal, para obter uma visualização da rede LoRaWAN disponível na cidade. Foi levada um processo de validação adicional, avaliando

o desempenho individual da *Input Layer* e da *Exit Layer*.

Para a *Input Layer* foi realizada a avaliação das latências de processamento e taxas de erro/perdas, utilizando um conjunto de dados simulados alimentados diretamente a arquitetura. Para a *Exit Layer* foi realizada a avaliação do desempenho na resposta a pedidos e uso de recursos computacionais para vários cenários. Foi utilizado o conjunto de dados reais recolhidos para validação.

O processo de avaliação concluiu que, embora o objetivo de evitar a perda de mensagens tenha sido alcançado com sucesso, o sistema apresenta uma latência significativa no processamento das mensagens. A maior parte deste atraso é introduzido pelo componente IoTAgent-LoRaWAN, utilizado como implementação de referência para os recetores *IoTMapperLpwanReceivers*. O aumento do número de instâncias de IoTAgent-LoRaWAN, e a divisão da recepção de mensagens entre elas, reduziu a latência entre 8.945 e 2.756 vezes para os volumes de mensagens testados.

Na disponibilização de informação conclui-se que a implementação de referência obtida sobre Kubernetes é uma solução viável. O uso cuidado dos mecanismos de *cache*, através do NGINX, oferece melhorias significativas de desempenho, com uso mais eficiente dos recursos computacionais.

Considera-se que, em geral, os componentes FIWARE são viáveis para a construção de soluções com os objetivos definidos. Considera-se, ainda, que a arquitetura proposta não é limitadora da Escalabilidade e Tolerância a Falhas, sendo uma proposta válida como resposta aos requisitos apresentados. A implementação avaliada é também considerada válida como referência, no entanto, é necessário trabalho adicional para a resolução de estrangulamentos em torno de IoTAgent-LoRaWAN, tais como a implementação de apoio a mecanismos de *batch* suportados pelo OCB e a identificação de pontos de limitação interna que o impeçam de processar simultaneamente múltiplas mensagens.

Sobre o trabalho futuro é possível identificar a possibilidade de desenvolver avaliação adicional sobre o afinamento das dimensões de *batch* interno usadas pelos componentes, o desempenho das etapas de processamento levadas a cabo pelo *IoTMapperDataProcessing*, o uso de múltiplas implementações para os *IoTMapperLpwanReceivers*, assim como, adicionar suporte a redes LPWAN adicionais, por exemplo, NB-IoT, requerendo nova validação e avaliação.

Referências

[1] United Nations Population Fund. "Urbanization". (mai. de 2022), URL: https://www.unfpa.org/urbanization.

[2] Sara Paiva, Mohd Abdul Ahad, Gautami Tripathi, Noushaba Feroz & Gabriella Casalino, "Enabling Technologies for Urban Smart Mobility: Recent Trends, Opportunities and Challenges", *Sensors*, vol. 21, n.º 6, 2021.

[3] Robert Ighodaro Ogie, Pascal Perez & Virginia Dignum, "Smart infrastructure: an emerging frontier for multidisciplinary research", *Proceedings of the Institution of Civil Engineers - Smart Infrastructure and Construction*, vol. 170, n.º 1, páginas 8–16, 2017. DOI: 10.1680/jsmic.16.00002. eprint: https://doi.org/10.1680/jsmic.16.00002. URL: https://doi.org/10.1680/jsmic.16.00002.

[4] Emmanuel Utochukwu Ogbodo, Adnan M. Abu-Mahfouz & Anish M. Kurien, "A Survey on 5G and LPWAN-IoT for Improved Smart Cities and Remote Area Applications: From the Aspect of Architecture and Security.", *Sensors (14248220)*, vol. 22, n.º 16, pág. 6313, 2022, ISSN: 14248220. URL: https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,shib&db=a9h&AN=158948487&lang=pt-pt&site=eds-live&scope=site.

[5] Renato Caminha Juacaba Neto, Pascal Merindol, Antoine Gallais & Fabrice Theoleyre, "Scalability of LPWAN for Smart City Applications.", *2021 International Wireless Communications and Mobile Computing (IWCMC), Wireless Communications and Mobile Computing (IWCMC), 2021 International*, páginas 74 –79, 2021, ISSN: 978-1-7281-8616-0. URL: https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,shib&db=edseee&AN=edseee.9498698&lang=pt-pt&site=eds-live&scope=site.

[6] Bassel Al Homssi, Akram Al-Hourani, Kagiso Magowe, James Delaney, Neil Tom, Joan Ying, Hans Wolf, Simon Maselli, Sithamparanathan Kandeepan, Ke Wang & Karina Mabell Gomez, "A Framework for the Design and Deployment of Large-Scale LPWAN Networks for Smart Cities Applications.", *IEEE Internet of Things Magazine, Internet of Things Magazine, IEEE, IEEE Internet Things M*, vol. 4, n.º 1, páginas 53 –59, 2021, ISSN: 2576-3180. URL: https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,shib&db=edseee&AN=edseee.9310563&lang=pt-pt&site=eds-live&scope=site.

[7] Philip J Basford, Florentin M. J. Bulot, Apetroaie-Cristea, Mihaela; Cox, Simon J.; Ossont & Steven J., "LoRaWAN for Smart City IoT Deployments: A Long Term Evaluation", *Sensors*, n.º 648, 2019.

[8] Nuno Cruz, Nuno Cota & João Tremoceiro, "LoRaWAN and Urban Waste Management—A Trial", *Sensors*, vol. 21, n.º 6, 2021, ISSN: 1424-8220. DOI: 10.3390/s21062142. URL: https://www.mdpi.com/1424-8220/21/6/2142.

[9] "FIWARE". (dez. de 2021), URL: https://www.fiware.org.

[10] José Fonseca, Fermín Márquez & Tobias Jacobs, "NGSIv2 API specification (v2.0)", nov. de 2021. URL: https://fiware.github.io/specifications/ngsiv2/stable/.

[11] Inc LoRa Aliance, "LoRaWAN™ 1.1 Specification", out. de 2017.

[12] Roberto Omar Andrade & Sang Guun Yoo, "A Comprehensive Study of the Use of LoRa in the Development of Smart Cities", *Applied Sciences*, vol. 9, n.º 22, 2019, ISSN: 2076-3417. DOI: 10.3390/app9224753. URL: https://www.mdpi.com/2076-3417/9/22/4753.

[13] GSMA, "NB-IoT Deployment Guide – Release 3", jul. de 2013.

[14] J Gozalvez, "New 3GPP standard for IoT", *IEEE Veh. Technol*, vol. 11, páginas 14–20, 2016.

[15] M.L. Liya & D. Arjun, "A Survey of LPWAN Technology in Agricultural Field", em *2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, 2020, páginas 313–317. DOI: 10.1109/I-SMAC49090.2020.9243410.

[16] Lei Zhang, Zhenxian Hu, Guangtao Xue, Yichao Chen, Minglu Li, Ming Wang & Feng Lv, "City-Wide NB-IoT Network Monitoring and Diagnosing", *Wireless Communications and Mobile Computing*,

[17] The Things Network. "The Things Network". (nov. de 2021), URL: http://thethingsnetwork.org.

[18] Norbert Blenn & Fernando A Kuipers, "LoRaWAN in the Wild: Measurements from The Things Network", *CoRR*, 2018.

[19] Michail J. Beliatis, Hussam Mansour, Szabolcs Nagy, Annabeth Aagaard & Mirko Presser, "Digital waste management using LoRa network a business case from lab to fab", *2018 Global Internet of Things Summit (GIoTS)*, páginas 1–6, 2018.

[20] "NGSIv2". (nov. de 2021), URL: https://fiware.github.io/specifications/ngsiv2/stable/.

[21] "NGSSI-LD". (nov. de 2021), URL: https://www.etsi.org/standards#page=1&search=Context\%20Information&title=1&etsiNumber=1&content=1&version=1&onApproval=1&published=1&historical=1&startDate=2020-01-01&endDate=2021-11-23&harmonized=0&keyword=&TB=854&stdType=&frequency=&mandate=&collection=&sort=1.

[22] "MQTT". (nov. de 2021), URL: https://mqtt.org/.

[23] Jay Kreps, Neha Narkhede & Jun Rao, *Kafka: a distributed messaging system for log processing. NetDB'11*.

[24] N. Narkhede, G. Shapira & T. Palino, *Kafka: The Definitive Guide: Real-Time Data and Stream Processing at Scale*. O'Reilly Media, 2017, ISBN: 9781491936139.

[25] Aditya Auradkar et al., "Data Infrastructure at LinkedIn", em *2012 IEEE 28th International Conference on Data Engineering*, 2012, páginas 1370–1381. DOI: 10.1109/ICDE.2012.147.

[26] João Pedro Vitorino & Nuno Cruz, "IoTMapper: A Metrics Aggregation System Architecture in Support of Smart City Solutions", *Sensors*, vol. 22, n.º 19, 2022, ISSN: 1424-8220. DOI: 10.3390/s22197484. URL: https://www.mdpi.com/1424-8220/22/19/7484.

[27] João Pedro Vitorino. "MEIC-2022-TFM-71-JPV". (dez. de 2022), URL: https://github.com/vitorinojp/MEIC-2022-TFM-71-JPV.

[28] Davide Magrin, Marco Centenaro & Lorenzo Vangelista, "Performance evaluation of LoRa networks in a smart city scenario", *IEEE International Conference on Communications*, páginas 1–7, 2017.

[29] Martin Stusek, Dmitri Moltchanov, Pavel Masek, Sergey Andreev, Yevgeni Koucheryavy & Jiri Hosek, "Time-Dependent Propagation Analysis and Modeling of LPWAN Technologies", em *2020 IEEE Globecom Workshops (GC Wkshps*, 2020, páginas 1–7. DOI: 10.1109/GCWkshps50303.2020.9367525.

[30] Martin Stusek, Dmitri Moltchanov, Pavel Masek, Konstantin Mikhaylov, Jiri Hosek, Sergey Andreev, Yevgeni Koucheryavy, Pavel Kustarev, Otto Zeman & Martin Roubicek, "LPWAN Coverage Assessment Planning Without Explicit Knowledge of Base Station Locations", *IEEE Internet of Things Journal*, vol. 9, n.º 6, páginas 4031–4050, 2022. DOI: 10.1109/JIOT.2021.3102694.

[31] Konstantinos Kousias, Giuseppe Caso, Ozgu Alay, Anna Brunstrom, Luca De Nardis, Maria-Gabriella Di Benedetto & Marco Neri, "Coverage and Deployment Analysis of Narrowband Internet of Things in the Wild", *IEEE Communications Magazine*, vol. 58, n.º 9, páginas 39–45, 2020. DOI: 10.1109/MCOM.001.2000131.

[32] Deliang Yang, Liqian Shen, Xianghui Zhang, Xiangmao Chang, Jun Huang & Guoliang Xing, "Software Suite for NB-IoT Measurement Analysis.", em *EWSN*, 2019, páginas 279–281.

[33] JP Meijers. "TTN Mapper". (nov. de 2021), URL: https://ttnmapper.org/.

[34] ——, "TTN Mapper Documentation". (nov. de 2021), URL: https://github.com/ttnmapper/documentation/.

[35] Helium. "Helium Mappers". (nov. de 2021), URL: https://mappers.helium.com/.

[36] Roman Zhohov, Dimitar Minovski, Per Johansson & Karl Andersson, "Real-time Performance Evaluation of LTE for IIoT", *IEEE 43rd Conference on Local Computer Networks (LCN)*, páginas 623–631, 2018.

[37] Chirpstack. "Chirpstack". (nov. de 2021), URL: https://www.chirpstack.io/.

[38] Helium. "Helium". (nov. de 2021), URL: https://helium.com/.

[39] C. Richardson, *Microservices Patterns: With examples in Java*. Manning, 2018, ISBN: 9781638356325.

[40] "RabbitMQ". (nov. de 2021), URL: https://www.rabbitmq.com/.

[41] Isaac Brodsky. "H3: Uber's Hexagonal Hierarchical Spatial Index". (jun. de 2022), URL: https://eng.uber.com/h3/.

[42] M. Kleppmann, *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O'Reilly Media, 2017, ISBN: 9781491903100.

[43] L.H. Etzkorn, *Introduction to Middleware: Web Services, Object Components, and Cloud Computing*. CRC Press, 2017, ISBN: 9781498754101. URL: https://books.google.pt/books?id=AZgnDwAAQBAJ.

[44] Roy Thomas Fielding, "Architectural Styles and the Design of Network-based Software Architectures", Tese de Doutoramento, 2000.

[45] "OpenAPI Specification v3.1.0". (fev. de 2022), URL: https://spec.openapis.org/oas/latest.html.

[46] "gRPC". (fev. de 2022), URL: https://grpc.io/.

[47] "Apache Thrift". (fev. de 2022), URL: https://thrift.apache.org/.

[48] "Finagle". (fev. de 2022), URL: https://twitter.github.io/finagle/.

[49] Niranjan Suri, Maggie R. Breedy, Kelvin. M. Marcus, Roberto Fronteddu, Eelco Cramer, Alessandro Morelli, Lorenzo Campioni, Mike Provosty, Conner Enders, Mauro Tortonesi & Jan Nilsson, "Experimental Evaluation of Group Communications Protocols for Data Dissemination at the Tactical Edge", em *2019 International Conference on Military Communications and Information Systems (ICMCIS)*, 2019, páginas 1–8. DOI: 10.1109/ICMCIS.2019.8842801.

[50] Stipe Celar, Eugen Mudnic & Zeljko Seremet, "State-Of-The-Art of Messaging for Distributed Computing Systems", em jan. de 2016, páginas 0298–0307, ISBN: 9783902734082. DOI: 10.2507/27th.daaam.proceedings.044.

[51] Cavide Gemirter & Sebnem Baydere, "A Comparative Evaluation of AMQP, MQTT and HTTP Protocols Using Real-Time Public Smart City Data,", out. de 2021. DOI: 10.1109/UBMK52708.2021.9559032.

[52] "Apache Kafka". (nov. de 2021), URL: https://kafka.apache.org/.

[53] "Apache ZooKeeper". (dez. de 2021), URL: https://zookeeper.apache.org/.

[54] "Kafka Connect". (dez. de 2021), URL: https://kafka.apache.org/documentation/#connect.

[55] "Kafka Streams". (dez. de 2021), URL: https://kafka.apache.org/30/documentation/streams/.

[56] Bill Bejeck, *Kafka Streams in Action: Real-Time Apps and Microservices With the Kafka Streams API*. Manning, 2018, ISBN: 9781617294471.

[57] "Mosquitto". (nov. de 2021), URL: https://www.mosquitto.org/.

[58] "HiveMQ". (nov. de 2021), URL: https://www.hivemq.com/.

[59] "NATS". (nov. de 2021), URL: https://nats.io/.

[60] "openDDS". (nov. de 2021), URL: http://www.opendds.org.

[61] "Redis". (nov. de 2021), URL: https://redis.io/.

[62] "ZeroMQ". (nov. de 2021), URL: https://zeromq.org/.

[63] S. Saxena & S. Gupta, *Practical Real-time Data Processing and Analytics: Distributed Computing and Event Processing using Apache Spark, Flink, Storm, and Kafk*. Packt Publishing, 2017, ISBN: 9781787289864. URL: https://books.google.pt/books?id=9JlGDwAAQBAJ.

[64] Giselle van Dongen & Dirk Van De Poel, "A Performance Analysis of Fault Recovery in Stream Processing Frameworks", *IEEE Access*, vol. 9, páginas 93 745–93 763, 2021. DOI: 10.1109/ACCESS.2021.3093208.

[65] Giselle van Dongen & Dirk Van den Poel, "Evaluation of Stream Processing Frameworks", vol. 31, n.º 8, páginas 1845–1858, 2020. DOI: 10.1109/TPDS.2020.2978480.

[66] Hae Sun Jung, Chul Sang Yoon, Yong Woo Lee, Jong Won Park & Chang Ho Yun, "Cloud computing platform based real-time processing for stream reasoning", em *2017 Sixth International Conference on Future Generation Communication Technologies (FGCT)*, 2017, páginas 1–5. DOI: 10.1109/FGCT.2017.8103400.

[67] "RocksDB". (jan. de 2022), URL: https://rocksdb.org/.

[68] "Apache Flink". (jan. de 2022), URL: https://flink.apache.org/.

[69] "Apache Spark". (jan. de 2022), URL: https://spark.apache.org.

[70] "MongoDB". (dez. de 2021), URL: https://github.com/FIWARE/context.Orion-LD.

[71] João Ricardo Lourenço, Bruno Cabral, Paulo Carreiro, Marco Vieira & Jorge Bernardino, "Choosing the right NoSQL database for the job: a quality attribute evaluation", *Journal of Big Data*, vol. 2, n.º 1, pág. 18, ago. de 2015, ISSN: 2196-1115. DOI: 10.1186/s40537-015-0025-0. URL: https://doi.org/10.1186/s40537-015-0025-0.

[72] "Apache Cassandra". (jan. de 2022), URL: https://cassandra.apache.org/_/index.html.

[73] "Apache HBase". (jan. de 2022), URL: https://hbase.apache.org/.

[74] H. Butler, M. Daly, A. Doyle, Sean Gillies, T. Schaub & Stefan Hagen, *The GeoJSON Format*, RFC 7946, ago. de 2016. DOI: 10.17487/RFC7946. URL: https://www.rfc-editor.org/info/rfc7946.

[75] "Spring Framework". (jan. de 2022), URL: https://spring.io/.

[76] Michal Gajewski & Wojciech Zabierowski, "Analysis and Comparison of the Spring Framework and Play Framework Performance, Used to Create Web Applications in Java", em *2019 IEEE XVth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, 2019, páginas 170–173. DOI: 10.1109/MEMSTECH.2019.8817390.

[77] "React Framework". (jan. de 2022), URL: https://reactjs.org/.

[78] "RFC 1738: Uniform Resource Locators (URL)". (dez. de 2021), URL: https://datatracker.ietf.org/doc/html/rfc1738.

[79] "RFC 3986: Uniform Resource Identifier (URI): Generic Syntax". (dez. de 2021), URL: https://datatracker.ietf.org/doc/html/rfc3986.

[80] "GeohashUtils". (dez. de 2021), URL: https://locationtech.github.io/spatial4j/apidocs/org/locationtech/spatial4j/io/GeohashUtils.html.

[81] "OpenStreetMap - GeoHash". (dez. de 2021), URL: https://wiki.openstreetmap.org/wiki/Geohash.

[82] "Compatibility Matrix". (dez. de 2021), URL: https://cwiki.apache.org/confluence/display/KAFKA/Compatibility+Matrix.

[83] "Unable to deliver event to Kafka #2014". (dez. de 2021), URL: https://github.com/telefonicaid/fiware-cygnus/issues/2014.

[84] "FIWARE IoTAgent-LoRaWAN". (nov. de 2021), URL: https://fiware-lorawan.readthedocs.io/en/latest/.

[85] Ivo Pedroso. "IvoPedrosoIoTagent-LoRaWAN". (nov. de 2021), URL: https://github.com/IvoPedroso/IoTagent-LoRaWAN.

[86] J. Arthur & S. Azadegan, "Spring Framework for Rapid Open Source J2EE Web Application Development: A Case Study", em *Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Networks (SNPD/SAWN'05)*, Towson, MD, USA: IEEE, 2005, páginas 90–95, ISBN: 9780769522944. DOI: 10.1109/SNPD-SAWN.2005.74. URL: http://ieeexplore.ieee.org/document/1434872/ (acedido em 24/09/2022).

[87] M. Mythily, A. Samson Arun Raj & Iwin Thanakumar Joseph, "An Analysis of the Significance of Spring Boot in The Market", em *2022 International Conference on Inventive Computation Technologies (ICICT)*, Nepal: IEEE, 2022, páginas 1277–1281, ISBN: 9781665408370. DOI: 10.1109/ICICT54344.2022.9850910. URL: https://ieeexplore.ieee.org/document/9850910/ (acedido em 24/09/2022).

[88] "TypeScript is JavaScript with syntax for types". (fev. de 2022), URL: https://www.typescriptlang.org/.

[89] "Semantic UI React". (fev. de 2022), URL: https://react.semantic-ui.com/.

[90] "Openlayers 6+". (fev. de 2022), URL: https://openlayers.org/.

[91] "RLayers 1.3.1". (fev. de 2022), URL: https://mmomtchev.github.io/rlayers/#/.

[92] Kubernetes. "Kubernetes". (ago. de 2022), URL: https://kubernetes.io/.

[93] B. HußR Ibryam, *Kubernetes Patterns: Reusable Elements for Designing Cloud-Native Applications*. O'Reilly Media, 2019, ISBN: 9781492050254.

[94] Leila Abdollahi Vayghan, Mohamed Aymen; Saied, Maria Toeroe & Ferhat Khendek, "Deploying Microservice Based Applications with Kubernetes: Experiments and Lessons Learned", *I018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, páginas 970–973, 2018. DOI: 10.1109/CLOUD.2018.00148.

[95] Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune & John Wilkes, "Large-scale cluster management at Google with Borg", em *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, 2015.

[96] Pavel Masek, Martin Stusek, Jan Krejci, Krystof Zeman, Jiri Pokorny & Marek Kudlacek, "Unleashing Full Potential of Ansible Framework: University Labs Administration", *22nd Conference of Open Innovations Association (FRUCT)*, páginas 144–150, 2018. DOI: 10.23919/FRUCT.2018.8468270.

[97] Dom Robinson, *Content Delivery Networks: Fundamentals, Design, and Evolution*. John Wiley & Sons, 2017, ISBN: 9781119249887.

[98] Dom Robinson, Mukaddim Pathan & Ramesh K. Sitaraman, *Advanced Content Delivery, Streaming, and Cloud Services*. John Wiley & Sons, 2014, ISBN: 9781119249887.

[99] Kubernetes. "NGINX Ingress Controller". (jun. de 2022), URL: https://kubernetes.github.io/ingress-nginx/.

[100] NGINX. "NGINX". (jun. de 2022), URL: https://nginx.org/.

[101] MetalLB. "MetalLB". (jun. de 2022), URL: https://metallb.universe.tf/.

[102] Cannonical. "Microk8s". (ago. de 2022), URL: https://microk8s.io/.

[103] Sebastian Böhm & Guido Wirtz, "Profiling Lightweight Container Platforms: MicroK8s and K3s in Comparison to Kubernete", *13th Central European Workshop on Services and their Composition*, 2021.

[104] Eclipse. "Eclipse Mosquitto". (nov. de 2021), URL: https://mosquitto.org/.

[105] The Apache Software Foundation. "Apache Jmeter". (jun. de 2022), URL: https://jmeter.apache.org/.

[106] Victor Araujo, Karan Mitra, Saguna Saguna & Christer Åhlund, "Performance evaluation of FIWARE: A cloud-based IoT platform for smart cities", *Journal of Parallel and Distributed Computing*, vol. 132, páginas 250–261, 2019. DOI: 10.1016/j.jpdc.2018.12.010.

[107] "Especificação YAML". (fev. de 2022), URL: https://yaml.org/.

Exemplo do Formato de Dados da TTN

Exemplo comentado do formato de dados JSON, como fornecido na documentação da TTN [34].

```
{
  "end_device_ids" : {
    "device_id" : "dev1",                    // Device ID
    "application_ids" : {
      "application_id" : "app1"              // Application ID
    },
    "dev_eui" : "0004A30B001C0530",          // DevEUI of the end device
    "join_eui" : "800000000000000C",         // JoinEUI of the end device (
    also known as AppEUI in LoRaWAN versions below 1.1)
    "dev_addr" : "00BCB929"                  // Device address known by the
    Network Server
  },
  "correlation_ids" : [ "as:up:01...", ... ],// Correlation identifiers of
   the message
  "received_at" : "2020-02-12T15:15..."      // ISO 8601 UTC timestamp at
   which the message has been received by the Application Server
  "uplink_message" : {
    "session_key_id": "AXA50...",            // Join Server issued
    identifier for the session keys used by this uplink
    "f_cnt": 1,                              // Frame counter
    "f_port": 1,                             // Frame port
    "frm_payload": "gkHe",                   // Frame payload (Base64)
```

```
18    "decoded_payload" : {                    // Decoded payload object,
      decoded by the device payload formatter
19      "temperature": 1.0,
20      "luminosity": 0.64
21    },
22    "rx_metadata": [{                        // A list of metadata for each
       antenna of each gateway that received this message
23      "gateway_ids": {
24        "gateway_id": "gtw1",                // Gateway ID
25        "eui": "9C5C8E00001A05C4"            // Gateway EUI
26      },
27      "time": "2020-02-12T15:15:45.787Z",    // ISO 8601 UTC timestamp at
      which the uplink has been received by the gateway
28      "timestamp": 2463457000,               // Timestamp of the gateway
      concentrator when the message has been received
29      "rssi": -35,                           // Received signal strength
      indicator (dBm)
30      "channel_rssi": -35,                   // Received signal strength
      indicator of the channel (dBm)
31      "snr": 5.2,                            // Signal-to-noise ratio (dB)
32      "uplink_token": "ChIKEA...",           // Uplink token injected by
      gateway, Gateway Server or fNS
33      "channel_index": 2                     // Index of the gateway
      channel that received the message
34      "location": {                          // Gateway location metadata (
      only for gateways with location set to public)
35        "latitude": 37.97155556731436,       // Location latitude
36        "longitude": 23.72678801175413,      // Location longitude
37        "altitude": 2,                       // Location altitude
38        "source": "SOURCE_REGISTRY"          // Location source. SOURCE_
      REGISTRY is the location from the Identity Server.
39      }
40    }],
41    "settings": {                            // Settings for the
      transmission
42      "data_rate": {                         // Data rate settings
43        "lora": {                            // LoRa modulation settings
44          "bandwidth": 125000,               // Bandwidth (Hz)
45          "spreading_factor": 7              // Spreading factor
46        }
47      },
48      "data_rate_index": 5,                  // LoRaWAN data rate index
49      "coding_rate": "4/6",                  // LoRa coding rate
50      "frequency": "868300000",              // Frequency (Hz)
51    },
```

```
52      "received_at": "2020-02-12T15:15..."      // ISO 8601 UTC timestamp at
        which the uplink has been received by the Network Server
53      "consumed_airtime": "0.056576s",          // Time-on-air, calculated by
        the Network Server using payload size and transmission settings
54      "locations": {                            // End device location
        metadata
55        "user": {
56          "latitude": 37.97155556731436,        // Location latitude
57          "longitude": 23.72678801175413,       // Location longitude
58          "altitude": 10,                       // Location altitude
59          "source": "SOURCE_REGISTRY"           // Location source. SOURCE_
        REGISTRY is the location from the Identity Server.
60        }
61      }
62    }
63    "simulated": true,                          // Signals if the message is
      coming from the Network Server or is simulated.
64  }
```

B

Parâmetros das Redes LPWAN

Existem múltiplas variáveis necessárias na definição e implementação da arquitetura, cujos valores exatos são dependentes dos detalhes da rede LPWAN. Reúne-se no presente apêndice a listagem destes valores.

B.1 Descritores das Redes LPWAN

Sempre que se pretenda descrever ou referir, de forma uniforme, as redes LPWAN consideram-se as regras de mapeamento indicadas na Tabela B.1. A tabela aplica-se em particular aos separadores de serviços no projecto FIWARE e seleccionadores de rede para a Web API.

Tabela B.1: Mapeamento dos descritores.

Rede	Campo	Valor
LoRaWAN	service	lorawan
LoRaWAN	subservice	/ttnmapper
NB-IoT	service	nbiot
NB-IoT	subservice	/<nome operadora>

B.2 Métricas Consideradas nas Redes LPWAN

O conjunto de métricas disponíveis para recolha varia entres as diferentes redes LPWAN. Para definição da interoperabilidade dos componentes, em particular que os componentes responsáveis pelas visualizações e interface com sistemas externos conhecem as métricas disponíveis, recolhe-se as métricas e suas designações na Tabela B.2.

Para cada métrica identificada é definida a sua agregação para calculo do valor médio, associado a pares de *Gateway* e *MapTile* no formato <*Gateway*>:<*MapTile*>, desde um ultimo evento na rede LPWAN, geralmente uma alteração da condição da condição do *Gateway*.

Tabela B.2: Métricas consideradas para as redes LPWAN.

Rede	Metrica [Unidade]	Obrigatório	Designação
LoRaWAN	RSSI [dBm]	Sim	rssi
LoRaWAN	RSSI Sinal [dBm]	Não	signal_rssi
LoRaWAN	RSSI Canal [dBm]	Não	channel_rssi
LoRaWAN	Desvio Padrão do RSSI [dBm]	Não	rssi_standard_deviation
LoRaWAN	SNR [dBm]	Não	snr

B.3 Factores para Reinício das Agregações das Métricas

Quando se considera o reinicio de uma agregação, isto é, que se elimine as métricas já acumuladas de forma a obter novas métricas, o fator de decisão pode variar entre redes LPWAN. Inicialmente considera-se, apenas, a distancia máxima de alteração de um Gateway.

Tabela B.3: Métricas consideradas para as redes LPWAN.

Rede	Factor	Valor associado
LoRaWAN	Distancia máxima desde a localização original do GW	100 metros

C

Modelos FIWARE

Apresentam-se os modelos de dados, na versão YAML [107], propostos como modelos de contextualização das métricas no FIWARE Orion Context Broker. Neste apêndice não são considerados os cabeçalhos associados aos modelos de dados para separação de contexto.

C.1 Recolha de Métricas

Modelo para representação genérica da mensagem, oriunda de um dado equipamento IoT, contendo as métricas a recolher.

```
1 MetricsReport:
2   description: 'Description of a generic and raw message entity,
     containing a measurement coming from a device or other data source
     .'
3   properties:
4     dateCreated:
5       description: 'Entity creation timestamp. This will usually be
     allocated by the storage platform.'
6       format: date-time
7       type: string
8       x-ngsi:
9         type: Property
```

```
10      dateModified:
11        description: 'Timestamp of the last modification of the entity.
       This will usually be allocated by the storage platform.'
12        format: date-time
13        type: string
14        x-ngsi:
15          type: Property
16      id:
17        anyOf:
18          - description: 'Property. Identifier format of any NGSI
      entity'
19            maxLength: 256
20            minLength: 1
21            pattern: ^[\w\-\.\{\}\ \+\ \[\]`| ^@ ,:\\]+
22            type: string
23          - description: 'Property. Identifier format of any NGSI
      entity'
24            format: uri
25            type: string
26        description: 'Unique identifier of the entity'
27        x-ngsi:
28          type: Property
29      name:
30        description: 'The name of this item.'
31        type: string
32        x-ngsi:
33          type: Property
34      reportValue:
35        description: 'The raw textual value of the message, typically a
       JSON formatted string. Must be URL safe'
36        type: string
37        x-ngsi:
38          model: https:  schema.org Text
39          type: Property
40      type:
41        description: 'NGSI Entity type. It has to be MetricsReport'
42        enum:
43          - MetricsReport
44        type: string
45        x-ngsi:
```

```
        type: Property

  required:
    - id
    - type
    - MetricsReport
  type: object
  version: 0.1.0
```

C.2 Resultados das Métricas Processadas e Agregadas

Modelo para representação genérica dos resultados obtidos após agregação das métricas.

```
MetricsAggregation:
  description: 'Description of a generic aggregation of measures
   metrics'
  properties:
    dateCreated:
      description: 'Entity creation timestamp. This will usually be
   allocated by the storage platform.'
      format: date-time
      type: string
      x-ngsi:
        type: Property
    dateModified:
      description: 'Timestamp of the last modification of the entity.
    This will usually be allocated by the storage platform.'
      format: date-time
      type: string
      x-ngsi:
        type: Property
    id:
      description: 'Property. Identifier format of any NGSI entity'
      maxLength: 256
      minLength: 1
      pattern: ^[\w\-\.\{\}\ \+\ \[\]`| ^@ ,:\\]+
      type: string
      x-ngsi:
```

```
23          type: Property
24      name:
25          description: 'The name of this item.'
26          type: string
27          x-ngsi:
28            type: Property
29      type:
30          description: 'NGSI Entity type. It has to be MetricsAggregation
        '
31          enum:
32            - MetricsAggregation
33          type: string
34          x-ngsi:
35            type: Property
36      dateLastGwUpdate:
37          description: 'Timestamp of the last action that led to the rest
        of the this aggregation'
38          format: date-time
39          type: string
40          x-ngsi:
41            type: Property
42      dateLastUpdate:
43          description: 'Timestamp of the last measure aggregated'
44          format: date-time
45          type: string
46          x-ngsi:
47            type: Property
48      gwId:
49          description: 'Unique identifier of the Gateway'
50          maxLength: 256
51          minLength: 1
52          pattern: ^[\w\-\.\{\}\ \+\ \[\]`| ^@ ,:\\]+
53          type: string
54          x-ngsi:
55            type: Property
56      mapTile:
57          description: 'Unique identifier of the geographic area'
58          maxLength: 128
59          minLength: 1
60          pattern: ^[\w\-\.\{\}\ \+\ \[\]`| ^@ ,:\\]+
```

```
61        type: string
62        x-ngsi:
63          type: Property
64      measures:
65        description: 'List of aggregated measures'
66        type: array
67        items:
68          properties:
69            name:
70              type: string
71            value:
72              type: double
73          type: object
74      location:
75        description: 'Geojson reference to the location covered by the
    item. It can be Polygon or MultiPolygon'
76        oneOf:
77          - description: 'Geoproperty. Geojson reference to the item.
    Polygon'
78            properties:
79              bbox:
80                items:
81                  type: number
82                minItems: 4
83                type: array
84              coordinates:
85                items:
86                  items:
87                    items:
88                      type: number
89                    minItems: 2
90                    type: array
91                  minItems: 4
92                  type: array
93                type: array
94              type:
95                enum:
96                  - Polygon
97                type: string
98            required:
```

```
99            - type
100           - coordinates
101         title: 'GeoJSON Polygon'
102         type: object
103       - description: 'Geoproperty. Geojson reference to the item.
    MultiPolygon'
104         properties:
105           bbox:
106             items:
107               type: number
108             minItems: 4
109             type: array
110           coordinates:
111             items:
112               items:
113                 items:
114                   items:
115                     type: number
116                   minItems: 2
117                   type: array
118                 minItems: 4
119                 type: array
120               type: array
121             type: array
122           type:
123             enum:
124               - MultiPolygon
125             type: string
126         required:
127           - type
128           - coordinates
129         title: 'GeoJSON MultiPolygon'
130         type: object
131     x-ngsi:
132       type: Geoproperty
133
134 required:
135   - id
136   - type
137   - gwId
```

```
    - mapTile
    - measures
    - location
  type: object
  version: 0.1.0
```

C.3 Informação sobre as Redes LPWAN

Modelo para representação genérica da informação sobre uma rede LPWAN, incluindo métricas previsivelmente disponíveis após agregação.

```
LpwanInfo:
  description: 'Description of a LPWAN network'
  properties:
    dateCreated:
      description: 'Entity creation timestamp. This will usually be
   allocated by the storage platform.'
      format: date-time
      type: string
      x-ngsi:
        type: Property
    dateModified:
      description: 'Timestamp of the last modification of the entity.
    This will usually be allocated by the storage platform.'
      format: date-time
      type: string
      x-ngsi:
        type: Property
    id:
      description: 'Property. Identifier format of any NGSI entity'
      maxLength: 256
      minLength: 1
      pattern: ^[\w\-\.\{\}\ \+\ \[\]`| ^@ ,:\\]+
      type: string
      x-ngsi:
        type: Property
    name:
      description: 'The name of LPWAN Network.'
```

```
      enum:
        - lorawan
        - nbiot
      type: string
      x-ngsi:
        type: Property
    type:
      description: 'NGSI Entity type. It has to be LpwanInfo'
      enum:
        - LpwanInfo
      type: string
      x-ngsi:
        type: Property
    measureLists:
      description: 'List of predictable aggregation measures'
      type: array
      items:
        properties:
          name:
            type: string
          unit:
            type: string
          mandatory:
            type: boolean
          description:
            type: string
        type: object

  required:
    - id
    - name
    - type
    - measures
  type: object
  version: 0.1.0
```

D

Web API para IoTMapperBackend

Apresenta-se a documentação referente a Web API desenvolvida para o componente *IoTMapperBackend*, como mecanismo de interface com o *IoTMapperPresentation*, ou, sistemas externos. A mesma encontra-se na versão YAML [107] para o formato OpenAPI 3.0.3 [45].

```yaml
openapi: 3.0.3
info:
  title: IoTMapperBackendApi
  description: IoTMapperBackendApi. Specifies the public Web API to
   access info on supported LPWAN, calculated metrics and information
    on gateways
  version: 0.1.0
servers:
  - url: 'https:  iotmapper api v1 '
    description: Main production server
externalDocs:
  description: IoTMapper API Guide
  url: https:  iotmapper help api
paths:
  # Metrics endpoints
   metrics list {lpwan}:
    get:
      description: Get the list of calculated metrics.
      responses:
```

```
18        200:
19          description: Ok - Regular response, list of metrics, or
    empty.
20          content:
21            application vnd.siren+json:
22              schema:
23                ref: '# components responses ListMetrics'
24      parameters:
25        -  ref: '# components parameters LpwanQuery'
26        -  ref: '# components parameters Limit'
27        -  ref: "# components parameters Offset"
28     post:
29      description: Search based on central location
30      responses:
31        200:
32          description: Ok - Regular response, list of metrics, or
    empty.
33          content:
34            application vnd.siren+json:
35              schema:
36                ref: '# components responses ListMetrics'
37      parameters:
38        -  ref: '# components parameters LpwanQuery'
39        -  ref: '# components parameters Limit'
40        -  ref: "# components parameters Offset"
41      requestBody:
42        description: Filter by a region
43        content:
44          application json:
45            schema:
46               ref: "# components schemas LocationFilterInputModel"
47    metrics entity {lpwan} {id}:
48     get:
49      description: Get a specific metric by LPWAN dependent id
50      responses:
51        200:
52          description: Ok - Regular response,the metric, or empty.
53          content:
54            application vnd.siren+json:
55              schema:
```

```
56                  ref: '# components responses SingleMetrics'
57        parameters:
58          -  ref: '# components parameters LpwanQuery'
59          -  ref: '# components parameters IdMetric'
60
61  # Gateways endpoints
62     gateways list {lpwan}:
63       get:
64         description: Get the list of gateways
65         responses:
66           200:
67             description: Ok - Regular response, list of gateways, or
    empty.
68             content:
69               application vnd.siren+json:
70                 schema:
71                   ref: '# components responses ListGateways'
72         parameters:
73           -  ref: '# components parameters LpwanQuery'
74           -  ref: '# components parameters Limit'
75           -  ref: "# components parameters Offset"
76       post:
77         description: Search based on central location
78         responses:
79           200:
80             description: Ok - Regular response, list of gateways, or
    empty.
81             content:
82               application vnd.siren+json:
83                 schema:
84                   ref: '# components responses ListGateways'
85         parameters:
86           -  ref: '# components parameters LpwanQuery'
87           -  ref: '# components parameters Limit'
88           -  ref: "# components parameters Offset"
89         requestBody:
90           description: Filter by a region
91           content:
92             application json:
93               schema:
```

```
94                ref: "# components schemas LocationFilterInputModel"
95    gateways entity {lpwan} {id}:
96     get:
97       description: Get a specific gateways by LPWAN and specific id
98       responses:
99         200:
100          description: Ok - Regular response, the gateway, or empty.
101          content:
102            application vnd.siren+json:
103              schema:
104                 ref: '# components responses SingleGateways'
105      parameters:
106        -  ref: '# components parameters LpwanQuery'
107        -  ref: '# components parameters GwMetric'
108
109 # Info endpoints
110    info list:
111     get:
112       description: Get the list of avaible lpwans.
113       responses:
114         200:
115          description: Ok - Regular response, list of lpwans.
116          content:
117            application vnd.siren+json:
118              schema:
119                 ref: '# components responses ListLpwan'
120      parameters:
121        -  ref: '# components parameters LpwanQuery'
122        -  ref: '# components parameters Limit'
123        -  ref: "# components parameters Offset"
124    info entry {lpwan}:
125     get:
126       description: Ask for additional information regarding a lpwan,
    if there is any
127       responses:
128         200:
129          description: Ok - Regular response, info about a lpwan
130          content:
131            application vnd.siren+json:
132              schema:
```

```
                ref: '# components responses SingleLpwan'

components:
  responses:
    ListMetrics:
      description: List of requested metrics
      content:
        application vnd.siren+json:
          schema:
            ref: '# components schemas ListMetrics'
    ListGateways:
      description: List of requested gateways
      content:
        application vnd.siren+json:
          schema:
            ref: '# components schemas ListGateways'
    ListLpwan:
      description: List of lpwans
      content:
        application vnd.siren+json:
          schema:
            ref: '# components schemas ListLpwan'
    SingleMetrics:
      description: A single metrics object
      content:
        application vnd.siren+json:
          schema:
            ref: "# components schemas SingleMetrics"
    SingleGateways:
      description: A single gateway object, if exists, else empty
      content:
        application vnd.siren+json:
          schema:
            ref: '# components schemas SingleGateways'
    SingleLpwan:
      description: A single lpwan object
      content:
        application vnd.siren+json:
          schema:
            ref: "# components schemas SingleLpwan"
```

```
  parameters:
    LpwanQuery:
      in: path
      name: lpwan
      required:
      schema:
        type: string
      description: The LPWAN id to filter by.
    Limit:
      in: query
      name: limit
      required:
      schema:
        type: integer
        format: Int64
        default: 50
      description: The limit of results to return. No guarantees on
     its ordering
    Offset:
      in: query
      name: limit
      required:
      schema:
        type: integer
        format: Int64
        default: 0
      description: The starting point of results to return. No
     guarantees on its ordering
    IdMetric:
      in: path
      name: id
      required:
      schema:
        type: string
      description: The LPWAN specific id of a metric, given as H(
    GeoHash || gwId )
    GwMetric:
      in: path
      name: id
      required:
```

```
210       schema:
211         type: string
212       description: The LPWAN specific id of a metric, given as H(
    GeoHash || gwId )
213
214   schemas:
215     # IoTMapper base models
216     ListMetrics:
217       type: object
218       properties:
219         class:
220           type: array
221           items:
222             type: string
223         properties:
224           type: object
225           properties:
226             list:
227               type: array
228               items:
229                   ref: "# components schemas MetricsBaseObject"
230     SingleMetrics:
231       type: object
232       properties:
233         class:
234           type: array
235           items:
236             type: string
237         properties:
238           type: object
239           properties:
240             result:
241                 ref: "# components schemas MetricsBaseObject"
242     ListGateways:
243       type: object
244       properties:
245         class:
246           type: array
247           items:
248             type: string
```

```
249         properties:
250           type: object
251           properties:
252             list:
253               type: array
254               items:
255                 ref: "# components schemas GatewaysBaseObject"
256     SingleGateway:
257       type: object
258       properties:
259         class:
260           type: array
261           items:
262             type: string
263         properties:
264           type: object
265           properties:
266             result:
267               ref: "# components schemas GatewayBaseObject"
268     ListLpwans:
269       type: object
270       properties:
271         class:
272           type: array
273           items:
274             type: string
275         properties:
276           type: object
277           properties:
278             list:
279               type: array
280               items:
281                 ref: "# components schemas LpwansBaseObject"
282     SingleLpwan:
283       type: object
284       properties:
285         class:
286           type: array
287           items:
288             type: string
```

```
289         properties:
290           type: object
291           properties:
292             result:
293                ref: "# components schemas LpwansBaseObject"
294     MetricsBaseObject:
295       type: object
296       properties:
297         location:
298            ref: "# components schemas Geometry"
299         mapTile:
300           type: string
301         gwId:
302           type: string
303         id:
304           type: string
305         measures:
306           type: object
307           description: A LPWAN dependant object with metric values.
    At least must have RSSI.
308           properties:
309             rssi:
310               type: number
311               format: double
312         dateLastGwChange:
313           type: string
314           required:
315           description: The timestamp for the last action that
    trigered the measures reset
316         dateLastUpdate:
317           type: string
318           required:
319           description: The timestamp for the last observed change, if
     avaible.
320     GatewayBaseObject:
321       type: object
322       properties:
323         point:
324            ref: "# components schemas Point"
325         id:
```

```
326           type: string
327         timestamp:
328           type: string
329           required:
330           description: The timestamp for the last observed change.
331     LpwanBaseObject:
332       type: object
333       properties:
334         id:
335           type: string
336         timestamp:
337           type: string
338           required:
339           description: The timestamp for the last observed change.
340     LocationFilterInputModel:
341       type: object
342       properties:
343         maxDistance:
344           type: integer
345           format: Int64
346           required:
347         point:
348            ref: "# components schemas GeoPair"
349     GeoPair:
350       type: object
351       properties:
352         lat:
353           type: number
354           format: double
355           required:
356         lon:
357           type: number
358           format: double
359           required:
360
361     # Siren base models
362     Entity:
363       type: object
364       properties:
365         class:
```

```
366           type: array
367           items:
368             type: string
369         entities:
370           type: array
371           items:
372             ref: '# definitions SubEntity'
373         actions:
374           type: array
375           items:
376             ref: '# definitions Action'
377         links:
378           type: array
379           items:
380             ref: '# definitions Link'
381     SubEntity:
382       type: object
383       properties:
384         href:
385           type: string
386         rel:
387           type: array
388           items:
389             type: string
390         class:
391           type: array
392           items:
393             type: string
394         properties:
395           type: object
396         entities:
397           type: array
398           items:
399             ref: '# definitions SubEntity'
400         actions:
401           type: array
402           items:
403             ref: '# definitions Action'
404         links:
405           type: array
```

```
406           items:
407             ref: '# definitions Link'
408     Action:
409       type: object
410       required:
411         - name
412         - href
413       properties:
414         class:
415           type: array
416           items:
417             type: string
418         name:
419           type: string
420         method:
421           type: string
422         href:
423           type: string
424         title:
425           type: string
426         type:
427           type: string
428         fields:
429           type: array
430           items:
431             ref: '# definitions Field'
432     Field:
433       type: object
434       required:
435         - name
436       properties:
437         name:
438           type: string
439         type:
440           type: string
441           enum:
442             - hidden
443             - text
444             - search
445             - tel
```

```
446             - url
447             - email
448             - password
449             - datetime
450             - date
451             - month
452             - week
453             - time
454             - datetime-local
455             - number
456             - range
457             - color
458             - checkbox
459             - radio
460             - file
461         title:
462           type: string
463         value:
464           type: string
465     Link:
466       type: object
467       required:
468         - rel
469         - href
470       properties:
471         class:
472           type: array
473           items:
474             type: string
475         title:
476           type: string
477         rel:
478           type: array
479           items:
480             type: string
481         href:
482           type: string
483         type:
484           type: string
485     # GeoJSON Base models
```

```
    Geometry:
      type: object
      description: GeoJSon geometry
      required:
        - type
      externalDocs:
        url: http:  geojson.org geojson-spec.html#geometry-objects
      properties:
        type:
          type: string
          enum:
            - Point
            - LineString
            - Polygon
            - MultiPoint
            - MultiLineString
            - MultiPolygon
          description: the geometry type
    Point3D:
      type: array
      description: Point in 3D space
      externalDocs:
        url: http:  geojson.org geojson-spec.html#id2
      minItems: 2
      maxItems: 3
      items:
        type: number
    Point:
      type: object
      description: GeoJSon geometry
      externalDocs:
        url: http:  geojson.org geojson-spec.html#id2
      allOf:
        -  ref: "# definitions Geometry"
        - properties:
            coordinates:
              ref: '# definitions Point3D'
    LineString:
      type: object
      description: GeoJSon geometry
```

```
526       externalDocs:
527         url: http:  geojson.org geojson-spec.html#id3
528       allOf:
529         -  ref: "# definitions Geometry"
530         - properties:
531             coordinates:
532               type: array
533               items:
534                   ref: '# definitions Point3D'
535     Polygon:
536       type: object
537       description: GeoJSon geometry
538       externalDocs:
539         url: http:  geojson.org geojson-spec.html#id4
540       allOf:
541         -  ref: "# definitions Geometry"
542         - properties:
543             coordinates:
544               type: array
545               items:
546                 type: array
547                 items:
548                     ref: '# definitions Point3D'
549     MultiPoint:
550       type: object
551       description: GeoJSon geometry
552       externalDocs:
553         url: http:  geojson.org geojson-spec.html#id5
554       allOf:
555         -  ref: "# definitions Geometry"
556         - properties:
557             coordinates:
558               type: array
559               items:
560                   ref: '# definitions Point3D'
561     MultiLineString:
562       type: object
563       description: GeoJSon geometry
564       externalDocs:
565         url: http:  geojson.org geojson-spec.html#id6
```

```
566       allOf:
567         -  ref: "# definitions Geometry"
568         - properties:
569             coordinates:
570               type: array
571               items:
572                 type: array
573                 items:
574                   ref: '# definitions Point3D'
575     MultiPolygon:
576       type: object
577       description: GeoJSon geometry
578       externalDocs:
579         url: http:  geojson.org geojson-spec.html#id6
580       allOf:
581         -  ref: "# definitions Geometry"
582         - properties:
583             coordinates:
584               type: array
585               items:
586                 type: array
587                 items:
588                   type: array
589                   items:
590                     ref: '# definitions Point3D'
591     GeometryCollection:
592       type: object
593       description: GeoJSon geometry collection
594       required:
595         - type
596         - geometries
597       externalDocs:
598         url: http:  geojson.org geojson-spec.html#geometrycollection
599       properties:
600         type:
601           type: string
602           enum:
603             - GeometryCollection
604         geometries:
605           type: array
```

```
606           items:
607             ref: '# definitions Geometry'
608       Feature:
609         type: object
610         description: GeoJSon Feature
611         required:
612           - type
613           - id
614           - geometry
615         externalDocs:
616           url: https:  tools.ietf.org html rfc7946#section-3.2
617         properties:
618           type:
619             type: string
620             enum:
621               - Feature
622           id:
623             type: integer
624           geometry:
625              ref: '# definitions GeometryCollection'
626           properties:
627             type: object
628       FeatureCollection:
629         type: object
630         description: GeoJSon Feature collection
631         required:
632           - type
633           - features
634         externalDocs:
635           url: https:  tools.ietf.org html rfc7946#section-3.3
636         properties:
637           type:
638             type: string
639             enum:
640               - FeatureCollection
641           features:
642             type: array
643             items:
644               ref: '# definitions Feature'
```