

**PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL  
ESCOLA POLITÉCNICA  
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO**

# **MAPEAMENTO DE RECURSOS SOB COMPUTAÇÃO EM NÉVOA**

**FELIPE BRIZOLA BERGUES DURO**

Trabalho de Conclusão II apresentado  
como requisito parcial à obtenção  
do grau de Bacharel em Ciência da  
Computação na Pontifícia Universidade  
Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Sérgio Johann Filho

**Porto Alegre  
2018**

“Há três caminhos para o fracasso: não ensinar o que se sabe, não praticar o que se ensina, e não perguntar o que se ignora.”  
(São Beda)

# MAPEAMENTO DE RECURSOS SOB COMPUTAÇÃO EM NÉVOA

## RESUMO

A computação em névoa pode ser vista como um paradigma que provê armazenamento, computação e serviços de rede entre dispositivos finais e os servidores na nuvem. Desse paradigma surge o conceito *fog node*, que abrange desde dispositivos finais com baixa capacidade computacional até servidores poderosos na nuvem. Comumente um fog node gerencia recursos, tais como sensores e atuadores. Esses recursos comunicam-se com o fog node através do protocolo CoAP. Sendo assim, um nodo interage exclusivamente com seus recursos, pois, desconhece os sensores e atuadores gerenciados por seus vizinhos de rede.

Os problemas referentes à padronização, descoberta e sincronização são os motivadores deste projeto, uma vez que atualmente não existem mecanismos no qual um fog node, mapeie os recursos disponíveis e divulgue os seus na rede.

Este projeto apresentará a modelagem e implementação de um protocolo com baixo custo de transmissão, distribuído, adaptativo e simples de ser empregado em topologias de rede já implantadas. Por fim, cenários de teste e estudos de caso serão apresentados com o objetivo de validar a solução proposta.

**Palavras-Chave:** mapeamento de recursos, computação em névoa, internet das coisas, protocolo de mapeamento.

# MAPPING RESOURCES UNDER THE FOG COMPUTING

## ABSTRACT

Fog computing can be seen as a paradigm that provides storage, computing, and network services between end devices and servers in the cloud. From this paradigm comes the *fog node* concept, which ranges from end-devices with low computational capacity to powerful servers in the cloud. Commonly a fog node manages resources such as sensors and actuators. These features communicate with the fog node through the CoAP protocol. Thus, a node interacts exclusively with its resources because it is unaware of the sensors and actuators managed by its network neighbors.

The problems related to standardization, discovery and synchronization are the motivators of this project, since there are currently no mechanisms in which a fog node maps the available resources and disseminates their resources on the network.

This project will present the modeling and implementation of a protocol with low transmission cost, distributed, adaptive and simple to be used in already implemented network topologies. Finally, test scenarios and case studies will be presented with the objective of validating the proposed solution.

**Keywords:** resource mapping, fog computing, internet of things, mapping protocol.

## LISTA DE FIGURAS

Figura 1.1 – Organização arquitetural . . . . .	11
Figura 2.1 – Requisição e resposta utilizando mensagens não confirmáveis . . . . .	15
Figura 3.1 – Topologia LAN . . . . .	19
Figura 3.2 – Arquitetura de um fog node . . . . .	20
Figura 3.3 – Arquitetura Resource-Mapping . . . . .	21
Figura 3.4 – Pilha de protocolos . . . . .	21
Figura 3.5 – Detalhamento Resource Mapping . . . . .	22
Figura 3.6 – Nodo entrando na névoa . . . . .	24
Figura 3.7 – Requisição entre fog nodes . . . . .	25
Figura 4.1 – Fog node container . . . . .	30
Figura 4.2 – Topologia base para cenários de teste . . . . .	31
Figura 4.3 – Nodo marcado como parcialmente inativo . . . . .	32
Figura 4.4 – Nodo removido da lista de recursos . . . . .	32
Figura 4.5 – Edge device fora de operação . . . . .	33
Figura 4.6 – Estado da névoa após atualização de recursos . . . . .	33
Figura 4.7 – Métricas alimentares na criação de porcos . . . . .	35
Figura A.1 – Fog node desligado enquanto curral permanece vazio . . . . .	38
Figura A.2 – Fog node ligado enquanto curral permanece ocupado . . . . .	38

## **LISTA DE TABELAS**

Tabela 2.1 – Comparação de funcionalidades. ....	18
--	----

## LISTA DE ALGORITMOS

Algoritmo 3.1 – Política de atualização de recursos .....	26
---	----

## **LISTA DE SIGLAS**

BGP – Border Gateway Protocol  
COAP – Constrained Application Protocol  
CPU – Central Processing Unit  
DTLS – Datagram Transport Layer Security  
HTTP – Hypertext Transfer Protocol  
IP – Internet Protocol  
LE – Low Energy  
NIST – National Institute of Standards and Technology  
P2P – Peer to Peer  
REST – Representational State Transfer  
RFC – Request for Comments  
RMI – Remote Method Invocation  
RPC – Remote Procedure Call  
SDK – Software Development Kit  
TCC – Trabalho de Conclusão de Curso  
TCP – Transmission Control Protocol  
UDP – User Datagram Protocol



# SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b>	<b>10</b>
1.1	CONTEXTUALIZAÇÃO	10
1.2	MOTIVAÇÃO E JUSTIFICATIVA	11
1.3	OBJETIVO	11
<b>2</b>	<b>FUNDAMENTAÇÃO TEÓRICA</b>	<b>13</b>
2.1	MODELOS PARA TROCA DE MENSAGENS EM APLICAÇÕES	13
2.1.1	PUBLISH/SUBSCRIBE	13
2.1.2	P2P	13
2.2	PROTOCOLOS RELACIONADOS	14
2.2.1	COAP	14
2.2.2	CONSTRAINED RESTFUL ENVIRONMENTS (CORE) LINK FORMAT	16
2.2.3	MQTT - MESSAGE QUEUE TELEMETRY TRANSPORT	17
2.3	TRABALHOS RELACIONADOS	17
<b>3</b>	<b>PROJETO</b>	<b>19</b>
3.1	ARQUITETURA	19
3.2	REQUISITOS E LIMITAÇÕES	21
3.3	FORMATO DE MENSAGENS	21
3.4	MÓDULOS	23
3.4.1	DESCOBERTA DE RECURSOS	23
3.4.2	GERENCIAMENTO DE RECURSOS	25
<b>4</b>	<b>VALIDAÇÕES</b>	<b>29</b>
4.1	AMBIENTE DE SIMULAÇÃO	29
4.2	CENÁRIOS DE TESTE	30
4.3	ESTUDOS DE CASO	34
4.3.1	SEGURANÇA RESIDENCIAL	34
4.3.2	MÉTRICAS NO AGRONEGÓCIO	34
	<b>REFERÊNCIAS</b>	<b>36</b>
	<b>ANEXO A – Depuração de métricas no agronegócio</b>	<b>38</b>

# 1. INTRODUÇÃO

## 1.1 Contextualização

A computação em nuvem tem sido amplamente adotada nos últimos anos por usuários finais e empresas de vários segmentos e portes. As facilidades proporcionadas por este modelo de computação faz com que exista tal preferência, pois, segundo a definição adotada pelo NIST [14], computação em nuvem é um modelo que permite acesso a um conjunto compartilhado de recursos computacionais configuráveis que podem ser provisionados e liberados com um pequeno esforço de gerenciamento ou interação com provedor de acesso. Toda essa flexibilidade pode justificar o aumento no emprego desse modelo de computação.

Uma das desvantagens da computação em nuvem está relacionada a aplicações sensíveis à latência, que requerem que nodos próximos atendam suas necessidades de forma eficiente. Como a interação entre os nós e os servidores na nuvem ocorrem através da Internet, a baixa latência torna-se indispensável para aplicações que requerem eficiência na comunicação entre nós (por exemplo robôs, drones, e carros autônomos). Portanto, há uma lacuna entre aplicações que já utilizam modelos de computação em nuvem e aplicações que necessitam de baixa latência de rede e comunicação entre nós próximos, e é nesse hiato que a computação em névoa surge.

A computação em névoa é um assunto relativamente novo e teve sua primeira definição, dada pela Cisco Systems<sup>1</sup> em 2012, como uma extensão do paradigma de computação em nuvem que provê armazenamento, computação e serviços de rede entre dispositivos finais e os servidores na nuvem [18].

Atualmente a computação em névoa tornou-se um paradigma próprio e não mais uma mera extensão da computação em nuvem, e desse paradigma criou-se o conceito de *fog nodes*, que abrangem desde dispositivos finais com baixa capacidade computacional até servidores poderosos na nuvem.

Desta forma, os *fog nodes* passam a fazer parte da implementação dos serviços em nuvem como uma camada intermediária entre ela os edge devices. Essa camada intermediária tem o propósito de aumentar a eficiência, tanto no processamento quanto no armazenamento de informações, além de reduzir a quantidade de dados transportados dos edge devices para a nuvem[15].

Em uma abordagem *bottom-up*, podemos descrever a arquitetura da computação em névoa como um conjunto de *edge devices*<sup>2</sup> que se comunicam com os *fog nodes*, e

---

<sup>1</sup>Empresa estadunidense líder na fabricação de equipamentos de rede [19].

<sup>2</sup>Dispositivo que controla o fluxo de dados no limite entre duas redes [20].

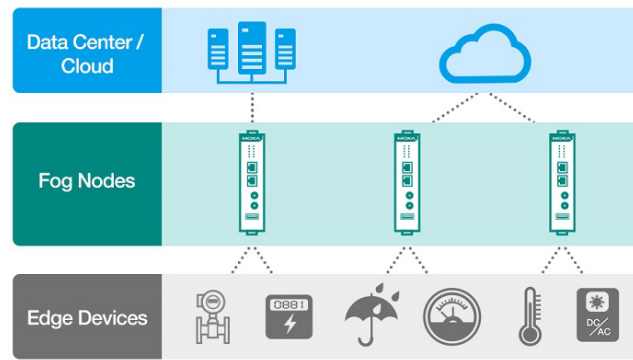


Figura 1.1 – Organização arquitetural.  
[4]

esses com servidores centrais. Entretanto, a comunicação entre os *fog nodes* e os servidores centrais não é essencial para a execução dos serviços em névoa [18]. A Figura 1.1 representa este conceito de computação em névoa.

## 1.2 Motivação e justificativa

De acordo com Vaquero e Roderio-Merino [26], serão sete os desafios que a computação em névoa deverá enfrentar para se tornar realidade. Os problemas referentes à padronização, descoberta e sincronização são os motivadores deste trabalho, uma vez que atualmente não existem mecanismos no qual um membro da rede, seja ele um dispositivo com limitações de memória e processamento ou um computador, mapeiem os recursos disponíveis e divulgue os seus na rede.

A ausência desses mecanismos faz com que cada membro interaja exclusivamente com seus recursos, portanto, não há compartilhamento entre os nodos da rede. O motivo pelo qual os nodos não consomem os recursos de seus vizinhos, está no desconhecimento dos mesmos. Sendo assim, é inviável utilização de um recurso que não esteja atrelado ao próprio nodo. Este compartilhamento de informações, referente aos nodos e recursos, justifica o desenvolvimento deste trabalho.

## 1.3 Objetivo

O objetivo principal deste trabalho de conclusão é resolver os desafios relacionados à padronização, descoberta e sincronização referidos por Vaquero e Roderio-Merino [26].

Modelar e implementar um protocolo com baixo custo de transmissão, que seja distribuído, que se adeque a ambientes dinâmicos e que seja simples de ser empregado

em topologias de rede já implantadas são os objetivos desse trabalho. Dessa maneira, o processo de descoberta e sincronização de recursos ocorrerá de forma automatizada.

## **2. FUNDAMENTAÇÃO TEÓRICA**

Este capítulo apresentará alguns modelos de troca de mensagens entre aplicações e protocolos de comunicação que servirão de apoio para o desenvolvimento deste TCC. Por fim, alguns trabalhos relacionados a área serão expostos, bem como o posicionamento deste perante aos demais.

### **2.1 Modelos para troca de mensagens em aplicações**

Atualmente existem diversos modelos para troca de mensagens entre aplicações, dentre eles RPC, RMI, Web services e cliente/servidor [24]. Para um melhor embasamento no desenvolvimento do projeto, focaremos nos modelos publish/subscribe e P2P que serão expostos nas Subseções a seguir.

#### **2.1.1 Publish/Subscribe**

Neste modelo, o produtor da mensagem (publisher) a rotula de alguma forma e, muitas vezes, isso é feito associando um ou mais nomes de tópicos a mensagem. Em seguida, os destinatários da mensagem (subscriber) especificam os tópicos para os quais desejam receber as mensagens associadas[11].

O uso de seleção de mensagem baseada em tópico ou conteúdo é denominado filtragem de mensagens. Observe que, em cada um desses casos, encontramos um relacionamento muitos-para-muitos entre editores e inscritos. O middleware de mensagens de publish/subscribe permite a implementação direta de modelos de notificação ou de programação baseada em eventos[11]. Assim, por exemplo, as mensagens que contêm dados relacionados a um erro ou sucesso de operação são capturadas apenas pelos inscritos nesses tópicos.

#### **2.1.2 P2P**

P2P é, basicamente, uma técnica que aproveita recursos computacionais e de conteúdo disponível na Internet para concluir tarefas enormes, como distribuição de conteúdo. Os recursos na Internet têm disponibilidade intermitente, pois, são adicionados e removidos inesperadamente o tempo todo. A computação P2P elimina a coordenação por um servidor central e nenhum ponto tem uma visão global de todo o sistema. Em vez disso,

os pares agem como servidores e clientes, compartilhando recursos e serviços diretamente uns com os outros[11]. Segundo Kai Hwang, algumas das principais características do P2P estão descritos abaixo.

- **Descentralização:** em um sistema de computação P2P puro, os pares são iguais e não há um servidor central para coordenar o sistema. Cada par tem uma visão parcial de todo o sistema para construir a rede de sobreposição e controlar seus dados e recursos.
- **Auto-organização:** significa que não é necessário um ponto central de gerenciamento para organizar os recursos de computação dos pares e os recursos de dados que se espalham pelos pares. Os recursos nos sistemas de computação P2P são dinâmicos ou flutuam no sentido de que eles vêm e vão a qualquer momento, à vontade. Seria muito caro dedicar um servidor para gerenciar esses sistemas.
- **Escalabilidade:** o modelo P2P elimina o problema do ponto único de falha relacionado ao modelo cliente/servidor centralizado tradicional. Cada par só mantém um número limitado de estados do sistema e compartilha recursos diretamente com os outros. Esses recursos permitem a alta escalabilidade dos sistemas de computação P2P.
- **Tolerância a falhas:** em uma rede P2P, todos os pares são criados iguais e nenhum ponto pode dominar o sistema ou se tornar um ponto único de falha para derrubar todo o sistema. Recursos podem ser armazenados em vários pares. Esses recursos facilitam a tolerância a falhas.

## 2.2 Protocolos relacionados

### 2.2.1 CoAP

Especificado pela RFC-7252, o CoAP, protocolo de aplicação restrita, foi projetado para aplicações máquina-a-máquina e tem como foco a transferência de documentos web entre nodos com recursos limitados em redes de baixa qualidade[22].

O modelo de interação cliente/servidor é o padrão adotado pelo CoAP, entretanto, o fato do protocolo ter sido projetado para aplicações máquina-a-máquina faz com que os dispositivos comumente desempenhem o papel de cliente e servidor simultaneamente.

Quando mensagens CoAP de requisição e resposta são trocadas, estas devem conter o código do método ou código da resposta, respectivamente. Além dos códigos, as mensagens podem conter outras informações, como o recurso que se deseja acessar e

o tipo de mídia que se está transportando. Por fim, um token é utilizado para que haja a correspondência entre requisição e resposta.

A Figura 2.1 ilustra uma solicitação entre cliente e servidor, na qual o cliente deseja obter a temperatura medida pelo sensor atrelado ao servidor. Analisando a troca de mensagens, notamos que o cliente envia uma solicitação não confirmável com token 0x74 e código GET para acessar o recurso de temperatura no servidor. O servidor por sua vez retorna o código de resposta 2.05, que indica sucesso, o mesmo token que recebeu na solicitação do cliente e o valor 22.5 C.

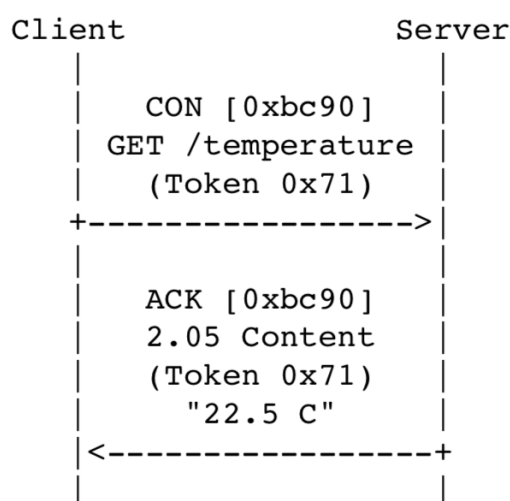


Figura 2.1 – Requisição e resposta utilizando mensagens não confirmáveis.  
[22]

A arquitetura REST, assim como no protocolo HTTP[6], foi utilizada na prototipação do protocolo CoAP. Como ambos compartilham da mesma arquitetura, realizar o mapeamento de HTTP para CoAP e vice-versa é bastante simples. Para realizar tal mapeamento basta utilizarmos o *cross-proxy*, definido na seção 10 da própria RFC-7252[22], que converte o método ou tipo de resposta, tipo de mídia e opções para os valores HTTP correspondentes.

Além do modelo de comunicação cliente/servidor e da arquitetura REST, o CoAP também dispõe de outros princípios comuns ao HTTP e que são conceitos padrões na web, como suporte a URIs[5] e tipos de mídia da Internet(MIME)[9].

Entre o HTTP e CoAP nem tudo são semelhanças, pois, obviamente, o CoAP possui particularidades para que consiga atender aos requisitos no qual se propõe a resolver. Entre as diferenças estão a troca de mensagens assíncronas utilizando transporte orientado a datagramas, como UDP. Apesar do protocolo UDP não prover confiabilidade, no CoAP é possível definir que as mensagens possuam tal aspecto.

*Constrained RESTful Environments (CoRE) Link Format*, que será abordado na seção a seguir, possibilidade de envio de solicitações multicast e unicast, *service discovery*,

cabeçalho com baixa sobrecarga de dados e segurança na forma de DTLS[17] estão entre as principais características do protocolo de aplicação restrita, CoAP.

## 2.2.2 Constrained RESTful Environments (CoRE) Link Format

Segundo a RFC-6690, a finalidade desta especificação é realizar REST em nós com recursos limitados, sendo tal característica importante em aplicações máquina-a-máquina[21]. Em aplicação deste tipo é primordial que as configurações não dependam de interação de humanos, portanto, devem manter seu funcionamento sem configurações estáticas.

A principal função desta especificação é fornecer identificadores, denominados URIs, para os recursos hospedados em servidores. Para além, é possível que essas URIs possuam atributos relacionados aos recursos. Estes atributos dividem-se basicamente em dois: *rt*, *resource type* e *if interface description*. O primeiro é utilizado para atribuir um nome que descreva de forma clara e enxuta um recurso, já o segundo tem por objetivo indicar a interface específica que interagirá com o recurso destino.

Um aspecto relevante ao CoRE, e de suma importância para aplicações máquina-a-máquina, é o fato de possuir como URI padrão o prefixo */.well-known/core*, definido na RFC-5785[10]. Este prefixo é utilizado para que o servidor exponha suas políticas e recursos disponíveis.

O protocolo CoAP prevê suporte à CoRE e ao prefixo */.well-known/core*. Contando com esses suportes, é possível traçarmos uma pequena analogia entre a arquitetura REST e o protocolo CoAP, pois, tanto em REST quanto em CoAP, as URIs são utilizadas para a identificação de recursos. Sendo assim, servidores CoAP ficam aguardando por requisições que utilizem tal URIs.

Para exemplificarmos, abaixo está um ciclo de requisição e resposta entre um cliente e um servidor com nomenclatura *example.net*. Este cliente deseja saber as políticas e recurso disponíveis pelo servidor, e para tal utiliza o prefixo padrão */.well-known/core*.

```
REQ: GET coap://example.net/.well-known/core
```

```
RES: 2.05 Content
      </sensors/temp>;if="sensor",
      </sensors/light>;if="sensor"
```

Analisando a resposta é fácil notar que este servidor possui dois recursos disponíveis, sendo ambos com interface do tipo sensor, porém, um utilizado para medir temperatura e outro para medir intensidade de luz. Juntamente aos dados, o código 2.05 é retornado indicando que a operação transcorreu com sucesso.



### 2.2.3 MQTT - Message Queue Telemetry Transport

O protocolo MQTT, inicialmente desenvolvido pela IBM, tornou-se um padrão para aplicações de IoT, pois, se mostrou escalável em ambientes com redes instáveis. Essa característica dá-se pelo fato do protocolo implementar um modelo assíncrono de troca de mensagens, deste modo, o emissor mantém-se desacoplado do receptor. Diferentemente de um modelo cliente/servidor, HTTP por exemplo, o protocolo faz uso da técnica *publish/subscribe*.

Essa técnica requer a existência de duas entidades, clientes e ao menos uma entidade central denominada *broker*. O funcionamento básico do protocolo em questão é demonstrado em três passos.

- O cliente pode assinar qualquer tópico de mensagens no broker, e para tal precisa conectar-se a ele. Essa conexão pode ser uma conexão TCP/IP simples ou uma conexão TLS criptografada para mensagens sensíveis.
- O cliente publica as mensagens em um tópico do broker.
- Em seguida, o broker encaminha a mensagem a todos os clientes que assinam esse tópico.

Baseado nas características descritas, notamos que o protocolo necessita de um ponto central de comunicação entre os nodos do sistema. Neste caso o broker faz com que o MQTT seja um protocolo centralizado, pois, caso ele fique inoperante toda a rede MQTT passa a ficar desatualizada.

## 2.3 Trabalhos Relacionados

Spencer Lewson implementou um protocolo em nível de aplicação [23] capaz de realizar a comunicação entre nodos sob computação em névoa. A especificação do protocolo e um *middleware* capaz de realizar o gerenciamento dos recursos dos dispositivos, são os principais componentes deste trabalho [13].

Sua implementação requer que haja um ponto central de comunicação entre os nodos, uma vez que a conectividade entre eles ocorrem via *Bluetooth LE*. A existência desse ponto justifica-se pelas regras de implementação do *Bluetooth LE*, na qual descreve dispositivos de duas naturezas: centrais e periféricos. Dispositivos centrais são responsáveis por descobrir dispositivos periféricos que estão interessados em criar conexão. Portanto, a característica do *Bluetooth LE* faz com que a topologia de rede e a arquitetura do projeto não seja distribuída [13].

A plataforma Iotivity [7], mantida pela Linux Foundation, tem como cerne a criação de um padrão no qual os dispositivos conectam-se entre si e a Internet. Esta plataforma possui quatro funcionalidades chaves, sendo elas, descoberta de recursos, gerenciamento de dados, gerenciamento de dispositivos e transmissão de dados.

Focaremos na descoberta de recursos providos pela plataforma, pois, é o item que mais se assemelha a proposta deste trabalho de conclusão. Como primeiro passo para a execução desta funcionalidade, devemos registrar o recurso utilizando sua URI e IP. Após o registro, é possível especificarmos que este recurso é observável, ou seja, quando houver algum evento o nodo que o registrou como observável será notificado. Apesar de a plataforma Iotivity possuir outras funcionalidades, ela não possui mecanismos para manter o mapeamento de recursos em redes locais.

A Tabela 2.1 faz comparações entre as principais funcionalidades providas, tanto pelos trabalhos relacionados quanto pelos protocolos elencados acima. O protocolo proposto é o único na Tabela 2.1 capaz de realizar a sincronização do mapeamento dos recursos com uma baixa complexidade de implementação.

Tabela 2.1 – Comparação de funcionalidades.

	CoAP	S. Lewson	Iotivity	MQTT	Protocolo proposto
Complexidade	média	média	alta	baixa	baixa
Bluetooth	não	sim	sim	sim	não
Lan	sim	sim	sim	sim	sim
Internet	sim	não	sim	sim	não
Map. de recursos	sim	sim	sim	não	sim
Sincronização do map.	não	não	não	não	sim
Recursos observáveis	sim	não	sim	sim	não
Distribuído	não	não	não	não	sim

### 3. PROJETO

A organização e a descrição deste projeto serão apresentados neste capítulo. Para tal, nas seções a seguir abordaremos a arquitetura, o formato de mensagens, módulos e submódulos que o constituem.

#### 3.1 Arquitetura

Seguindo a topologia da Figura 3.1, podemos observar que os *fog nodes* não possuem um nodo central como servidor. Sendo assim, os fog nodes e os edge devices comunicam-se utilizando o protocolo CoAP. Dessa maneira, as Requisições entre fog nodes só ocorre caso o nodo requisitante saiba, previamente, o endereço IP relativo ao nodo solicitado.

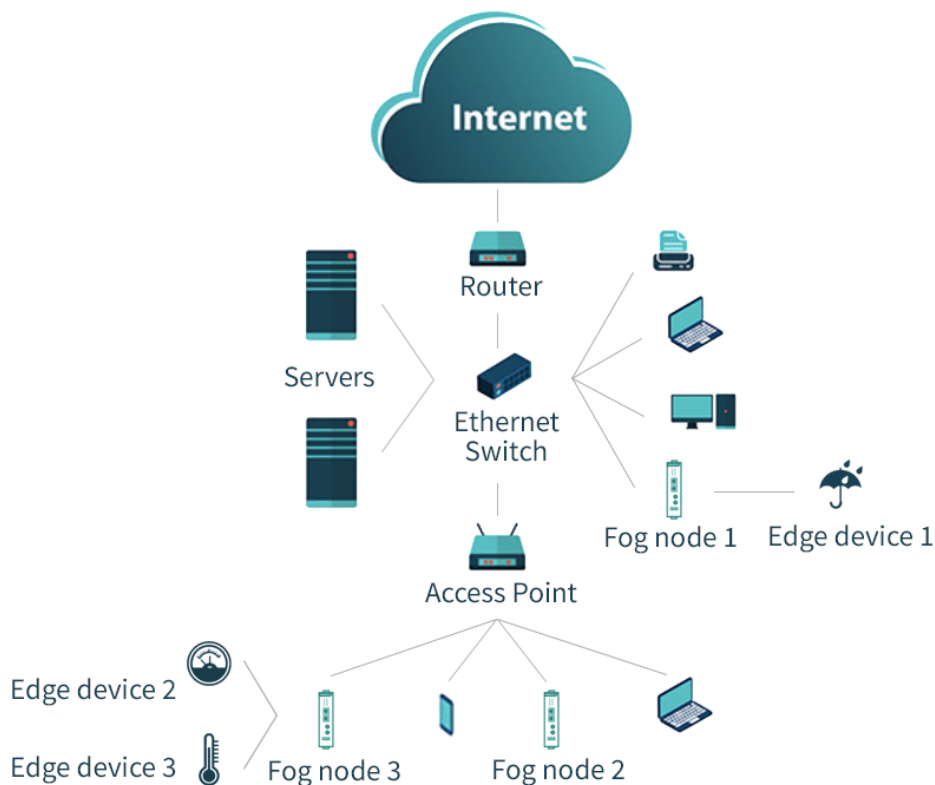


Figura 3.1 – Topologia LAN.

Em razão da topologia distribuída, a arquitetura deve ser capaz de mapear e sincronizar, de forma autônoma, os recursos providos pelos fog nodes. Dessa forma, cada *fog node* saberá quais são os *edge devices* disponíveis na rede, portanto, o nodo que possui o sensor de chuva saberia que existe um outro nodo em sua LAN capaz de medir a temperatura, por exemplo.

A arquitetura proposta, no que se refere a comunicação entre fog nodes e edge devices, utilizou como base um fork da implementação do protocolo CoAP[25]. Esse fork, implementado em Python e de acordo com a RFC-7252, consiste em dois grandes módulos: *CoAP-Server* e *CoAP-Client*. O primeiro, é responsável pelo recebimento das Requisições, e foi alterado com o objetivo de proporcionar o dinamismo na inserção e remoção dos recursos sem a necessidade de sua reinicialização. O segundo módulo é encarregado de realizar as Requisições diretamente aos nodos, e para essa funcionalidade nenhuma alteração no projeto original foi realizada.

Observando a Figura 3.2, podemos notar que a arquitetura de um fog node é composta por três grandes camadas, sendo elas hardware, sistema operacional e aplicações.

Na camada de hardware podemos utilizar desde um dispositivo com limitações de memória e CPU até um aparelho com grande capacidade computacional. Já na camada de sistema operacional, temos a liberdade de utilizar uma vasta diversidade de distribuições Unix. Por fim, na camada de aplicação devemos executar os processos *CoAP-Server*, *CoAP-Client* e *Resource-Mapping*.

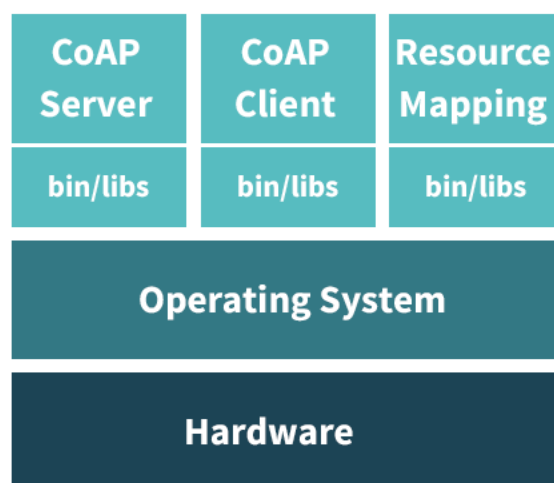


Figura 3.2 – Arquitetura de um fog node.

A Figura 3.3 exibe, de maneira detalhada, os processos envolvidos na execução no protocolo *Resource-Mapping*. Nela podemos destacar a presença de três entidades principais, *Observer*, *Keep Alive* e *Receptor*. Abaixo elencaremos as funcionalidades de cada entidade.

- *Observer* é responsável por manter o estado de seus próprios recursos atualizados.
- *Keep Alive* tem a missão de manter seus vizinhos de rede atualizados sobre seu estado de operação.
- *Receptor* é incumbido de receber e processar as requisições recebidas via *Resource-Mapping*.



Figura 3.3 – Arquitetura Resource-Mapping.

### 3.2 Requisitos e limitações

Para que a solução opere corretamente, tanto no mapeamento quanto na sincronização de recursos, algumas limitações e requisitos necessitam ser expostas.

Como proposição inicial temos que o sistema operacional deve ser Unix e o usuário corrente deve dispor de privilégios de root. Além disso, a linguagem de programação Python em sua versão 2.7 com seu respectivo gerenciador de pacotes, *PIP*[8], devem estar instalados. Por fim, os dados que os edge devices fornecerão aos fog nodes serão tratados de forma simulada, pois o funcionamento exato dos sensores e atuadores fogem do escopo desse projeto. As Seções 4.3.1 e 4.3.2 utilizarão essas simulações a fim de validar o protocolo.

### 3.3 Formato de mensagens

Esta Seção define a pilha de protocolos a serem utilizados neste projeto, a justificativa pelas suas escolhas, e por fim, o detalhamento do protocolo proposto. A pilha de protocolos atuará em conjunto com a organização arquitetural previamente definida na Figura 1.1.

A fim de facilitar a compreensão da arquitetura deste projeto, a Figura 3.4 explicita a pilha de protocolos que o projeto fará uso para implementar as funcionalidades propostas.

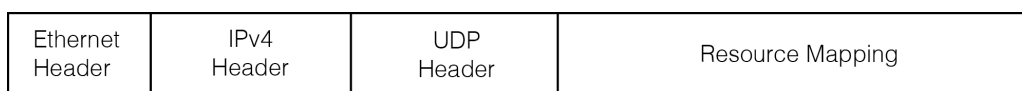


Figura 3.4 – Pilha de protocolos.

O modelo de referência TCP/IP é constituído de cinco camadas: física, enlace, rede, transporte e aplicação [23]. Nesse trabalho, os níveis de rede e transporte (IPv4 e UDP respectivamente) serão utilizados para a implementação do modelo proposto.

A utilização de IPv4 na camada de rede justifica-se pelo fato do protocolo ser empregado em redes locais, que geralmente não necessitam de uma quantidade de en-

dereçamento tão grande se comparado ao IPv6, mas não existem impedimentos para que implementações futuras utilizem IPv6 na camada de rede.

Manter o contexto de conexão entre os nodos, utilizando TCP por exemplo, dependeria uma quantidade de tráfego desnecessário na rede. Visto que o baixo custo na transmissão de dados é um dos objetivos desse trabalho, a utilização de datagramas UDP faz sentido para que se aumente o desempenho da solução.

A Figura 3.5 apresenta, de forma detalhada, a estrutura dos dados contidos no protocolo *Resource Mapping*.

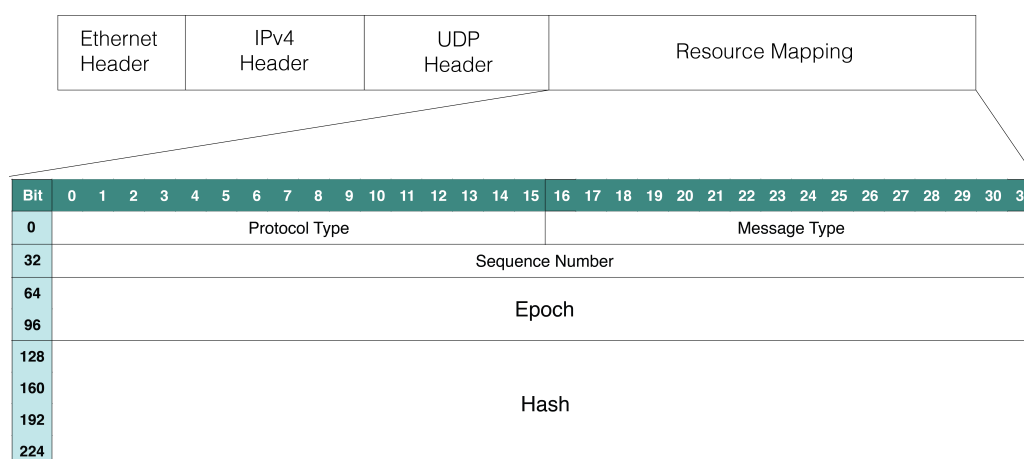


Figura 3.5 – Detalhamento Resource Mapping.

- O campo Protocol Type é reservado para indicar a forma de encapsulamento do pacote. Dessa forma, o restante dos campos do pacote podem ser tratados de acordo com a definição dada pelo campo.
- Atualmente existem dois tipos de Messages Types possíveis, keep alive e acknowledgement.
- Sequence Number tem o intuito de identificar o pacote enviado.
- O campo Epoch é utilizado para indicar alterações nos recursos providos pelo fog node. Sendo assim, quando um recurso é adicionado ou removido de um fog node, o campo Epoch é acrescido em uma unidade. Os detalhes de seu comportamento serão abordados na Seção 3.3.2.
- O campo Hash utiliza a função de criptografia MD5 para validar a integridade dos demais campos contidos no pacote.

O protocolo proposto, intitulado *Resource Mapping*, como apresentado nas Figuras 3.4 e 3.5, atuará na camada de aplicação do modelo de referência TCP/IP [23] e será

responsável por padronizar, descobrir e sincronizar os nodos da névoa. Os maiores desafios neste modelo proposto são manter o estado global dos recursos acessível a todos os nodos, e garantir que o desempenho seja satisfatório com o objetivo permitir a escalabilidade da solução.

### 3.4 Módulos

De forma geral, cada nodo da rede mantém uma lista com os endereços IP's que fazem parte do mapeamento. Atrelado à cada endereço IP, há uma lista com os recursos providos por este. Em vista disso, cada nodo contém um mapeamento global de recursos disponíveis na névoa.

O detalhamento das funcionalidades que o projeto possui, tal como ilustrações relacionadas aos fluxos, serão abordadas nas próximas subseções.

#### 3.4.1 Descoberta de recursos

Partindo do pressuposto que os nodos da névoa já possuem seus recursos devidamente criados e acessíveis via CoAP, como primeiro passo do mapeamento devemos considerar a entrada de um novo nodo na rede. No momento em que o nodo dispor de um endereço IP válido, este deverá enviar um pacote para o endereço de multicast indicando que possui recursos a serem disponibilizados.

Ao receberem o pacote enviado por multicast, os nodos que desejarem saber quais recursos estão sendo providos por este novo membro, deverão realizar uma Requisição unicast para a URI */.well-known/core* utilizando o protocolo CoAP. Vale lembrar que este fluxo de requisição e resposta, utilizando a URI */.well-known/core*, faz com que o nodo requisitado retorne todos seus recursos ao solicitante.

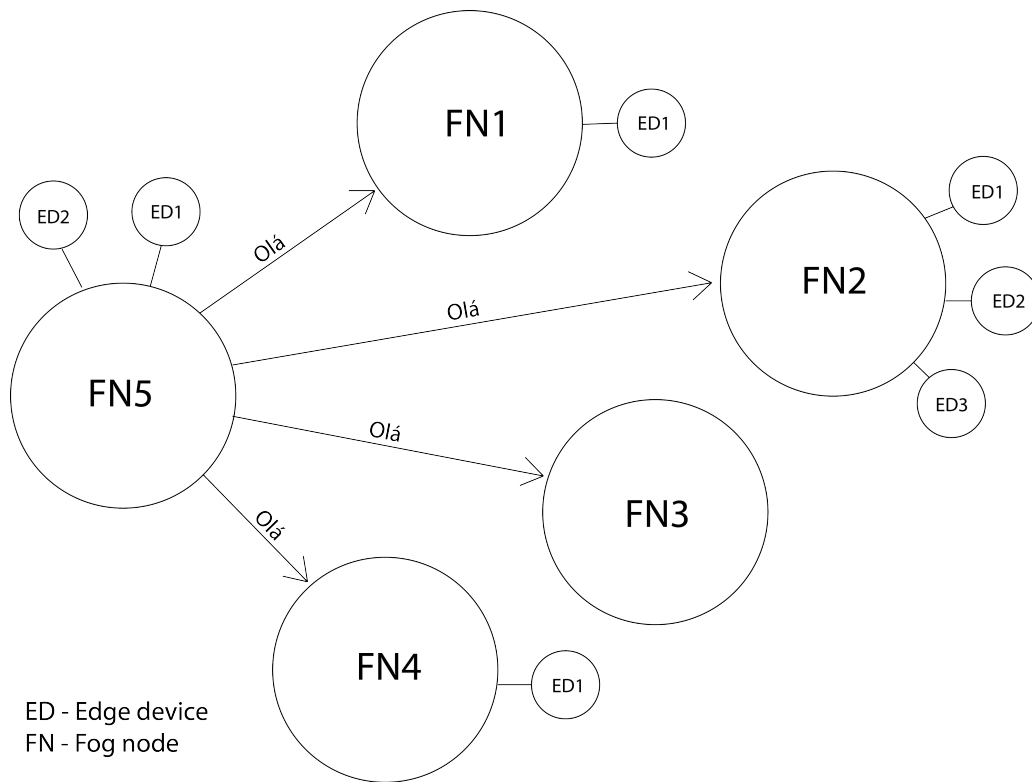


Figura 3.6 – Nó entrando na névoa.

A topologia da névoa utilizada nas Figuras 3.6 e 3.7 é definida por fog nodes enumerados de um a cinco, sendo o nó FN5 o último a entrar na rede. A figura 3.6 demonstra o nó FN5 entrando na névoa e, portanto, deverá anunciar-se por multicast indicando que possui recursos a serem disponibilizados.

Após FN5 enviar mensagem de Olá por multicast, os nós FN1, FN2 e FN4 realizam a Requisição CoAP diretamente ao FN5 afim de obter os recursos disponibilizados por ele, já o nó FN3, por não estar executando o protocolo de mapeamento, não realiza a Requisição CoAP.



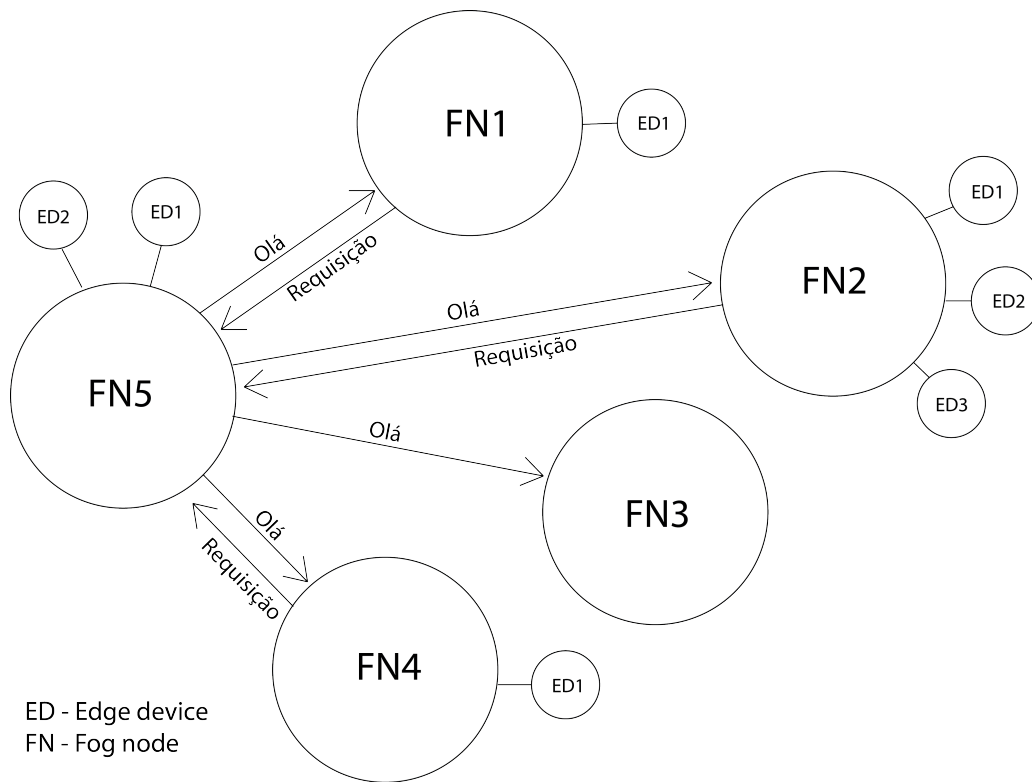


Figura 3.7 – Requisição entre fog nodes.

### 3.4.2 Gerenciamento de recursos

A manutenibilidade da lista de recursos globais é relevante para que o protocolo funcione de acordo com a especificação, pois, a névoa deverá saber quando um nodo, ou recurso dele, deixou de fazer parte da rede. Para tal, faz-se necessário a utilização de alguns mecanismos de controle. Esses controles são realizados em duas esferas, a primeira trata da inserção ou remoção de um nodo na rede, já a segunda refere-se a inserção ou remoção de um edge device vinculado a um nodo qualquer.

Inicialmente abordaremos a entrada e saída de nodos da névoa, e para esta subseção será utilizado o cenário da Figura 3.7 quando necessário. O Pseudocódigo 3.1 demonstra, de forma sucinta, a política de atualização que cada nodo deverá implementar no recebimento de mensagens de keep alive.

```

1: function Policy(ip, epoch, myEpoch)
2:   if exists(ip)
3:     if epochHasChanged(epoch, myEpoch);
4:       resources = getResourcesCoAP(ip, '/.well-known/core');
5:       update(ip, resources);
6:   else
7:     resources = getResourcesCoAP(ip, '/.well-known/core');
8:     insert(ip, resources);
9:   sendAcknowledgement(ip);

```

Algoritmo 3.1 – Política de atualização de recursos.

No momento em que o nodo recebe a resposta da chamada de função *getResourcesCoAP*, contendo os recursos providos pelo nodo requisitado, aquele deverá armazenar as informações em uma estrutura de dados adequada. Essa estrutura de dados estará implementada em todos os nodos da névoa, e a partir dela será possível realizar o gerenciamento dos recursos de forma simples e eficiente. O trecho abaixo define, por ora, o formato dos dados que serão utilizados em cada nodo.

```

Fog = {
    string ip;
    string resources[];
};
Fog fogs[];
string myResources[];

```

De posse da Figura 3.7 como cenário, do algoritmo de atualização 3.1, e da estrutura de dados previamente descrita, demonstramos abaixo o estado em que se encontram os dados armazenados em FN2 após a aplicação do método.

```

fogs: [
    {
        ip: '192.168.0.1',
        resources: [ 'ED1' ]
    },
    {
        ip: '192.168.0.3',
        resources: [ ]
    },
    {
        ip: '192.168.0.4',

```

```

        resources: [ 'ED1' ]
    },
    {
        ip: '192.168.0.5',
        resources: [ 'ED1', 'ED2' ]
    }
];
myResources: [ 'ED1', 'ED2', 'ED3' ];

```

Visto isso, é imprescindível que os nodos mantenham seus dados consistentes, pois, após adicionar o novo nodo em sua lista de fogs, o protocolo precisa ser capaz de perceber quando um elemento deixou de fazer parte do processo. Assim, a manutenção dos estados será abordado de forma similar as mensagens de *keep alive* utilizadas no protocolo BGP[16]. Mensagens de keep alive são adotadas para que os nodos da rede avisem seus vizinhos que ainda estão em operação, pois, sem esse procedimento seria difícil saber quando remover um IP da lista de recursos. Portanto, para manter a lista atualizada, este protocolo implementa mensagens desse tipo.

As mensagens de keep alive serão transmitidas sob multicast em um intervalo de trinta segundos, porém, este é apenas um valor arbitrário, e pode ser alterado para que a solução obtenha uma melhor eficiência. Após o recebimento da mensagem de keep alive, os nodos deverão respondê-las indicando que ainda estão em operação. Caso o nodo não responda a mensagem de keep alive, este será marcado como parcialmente inativo pelo remetente da mensagem. Quando o nodo solicitante realizar outra mensagem de keep alive e o nodo que já estava marcado com parcialmente inativo não responder, o mesmo será removido da lista de recursos do solicitante, e assim é possível saber quando um nodo deixou de fazer parte da névoa. Para realizar este controle será preciso adicionar na estrutura *Fog* a propriedade booleana, denominada *isReplyingKeepAlive*, que indica se o nodo está respondendo a solicitações de keep alive.

As mensagens de keep alive mantém os nodos informados sobre o estado de seus vizinhos, mas não conseguem indicar informações relacionadas aos edge device. Consequentemente, não é possível saber quando um edge device parou ou iniciou sua operação em um nodo ativo.

Para que seja possível detectar esse tipo de comportamento, o protocolo proposto deverá implementar a técnica denominada *época*, e servirá para que os nodos mantenham ciência sobre o estado dos edge devices de seus vizinhos de rede. Para isso, uma nova propriedade denominada *epoch*, do tipo inteiro, deverá ser adicionada a estrutura de dados, e estará presente tanto no nodo em si quanto em cada item da lista de fogs. O valor da *epoch* será incrementado em uma unidade a toda alteração observada no nodo, seja pelo acréscimo ou pela remoção de edge devices.

Esta observação atuará realizando requisições para a URI `/.well-known/core` no endereço de loopback do próprio nodo, assim, obterá seus recursos disponíveis e poderá comparar com os que possui armazenado em sua estrutura de dados. Havendo divergências, como mencionado anteriormente, a propriedade época será incrementada em uma unidade.

A partir de agora, então, todas as mensagens de keep alive deverão conter a época do nodo que está realizando o multicast. Assim, todo nodo que receber a mensagem poderá comparar a época recebida na mensagem com a época que possui armazenada em sua lista de fogs referente ao remetente da mensagem. Havendo divergência de épocas, este deverá realizar uma nova requisição para atualizar sua lista de recursos referente aquele nodo em específico.

Após as alterações realizadas na estrutura de dados, abaixo temos o novo modelo que suportará as funcionalidades propostas.

```
Fog = {  
    string ip;  
    string resources[];  
    int epoch;  
    boolean isReplyingKeepAlive;  
}  
Fog fogs[];  
string myResources[];  
int epoch;
```

## 4. VALIDAÇÕES

Este capítulo abordará temas relacionados as validações realizadas neste projeto. Sendo assim, nas seções a seguir serão apresentados temas como ambientes de simulações, cenários de testes e, por fim, estudos de casos.

### 4.1 Ambiente de simulação

Em um primeiro momento a névoa foi construída de forma simulada utilizando a ferramenta Common Open Research Emulator (CORE)[12]. Para os primeiros experimentos a ferramenta obteve resultados satisfatórios, porém, quando foi necessário escalarmos a solução ela se mostrou ineficiente. A ineficiência estava na criação dos nodos e na execução dos protocolos CoAP e Resource Mapping, uma vez que eles deveriam ser executados em todos os hosts da rede e não havia meios de automatizar este processo.

Esta automatização foi implantada utilizando a SDK de gerenciamento de containers Docker para Python, desta forma, foi possível criarmos névoas com tamanhos arbitrários, portanto, a escalabilidade da solução pode ser amplamente testada[3].

Cabe ressaltar que containers Docker[2] consomem recursos da máquina hospedeira, tais como memória, processamento e disco rígido. Portanto, a criação de instâncias de nodos (executando em containers) está limitada à quantidade de memória e capacidade de processamento da máquina hospedeira.

Cada container executa uma distribuição Linux chamada Alpine, de aproximadamente 5Mb, que possui apenas as funcionalidades básicas de um sistema operacional[1]. O fato da distribuição ser Open Source faz com que existam customizações da mesma, e neste projeto foi utilizado a customização que inclui a versão 2.7 da linguagem Python. Tal customização faz-se necessária para a execução dos protocolos CoAP e Resource Mapping.

A Figura 4.1 não só expõe os elementos que fazem parte da arquitetura de cada container, mas também os detalha ao nível de requisições entre si. Conforme apresentado na figura, o primeiro nodo recebe uma informação atualizada sobre a época de um segundo fog node, o que faz com que o primeiro realize uma requisição */.well-known/core* de modo a obter os recursos disponíveis pelo segundo.

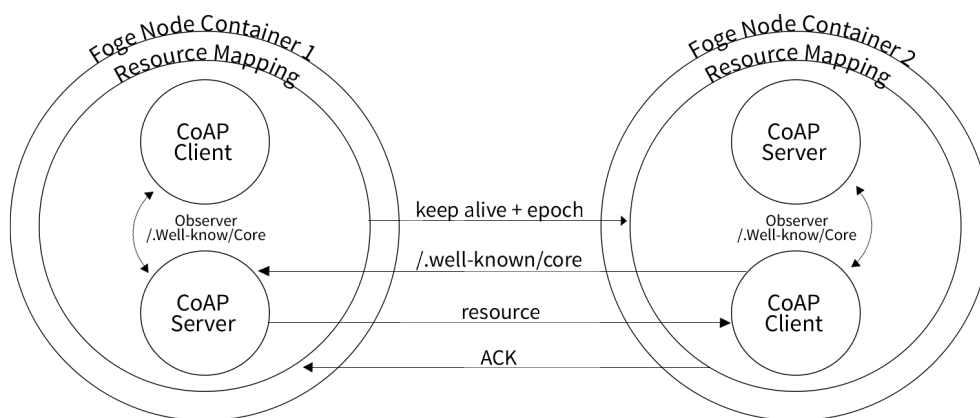


Figura 4.1 – Fog node container.

Além do protocolo de mapeamento em si, este trabalho dispõe de uma API para o gerenciamento de recursos providos pelos CoAP servers. Esta API possui métodos para listagem, remoção e criação de recursos e deve ser executada a partir do host que gerencia os container. Assim, conseguimos proporcionar o dinamismo que a névoa necessita para a realização de testes no comportamento do protocolo de sincronização.

## 4.2 Cenários de teste

Utilizando como base a Figura 4.2, que retrata uma névoa com três fog nodes e seus respectivos edge devices, e partindo do pressuposto que todos os nodos da névoa estão com seus recursos sincronizados corretamente. Iremos exemplificar os cenários de testes elencados abaixo.

1. Entrada de algum equipamento na rede e este anunciando seus recursos.
2. Atualização das listas globais quando algum equipamento deixar de responder as mensagens de keep alive.
3. Atualização da lista de recursos quando algum edge device é adicionado ou removido de um nodo da névoa.

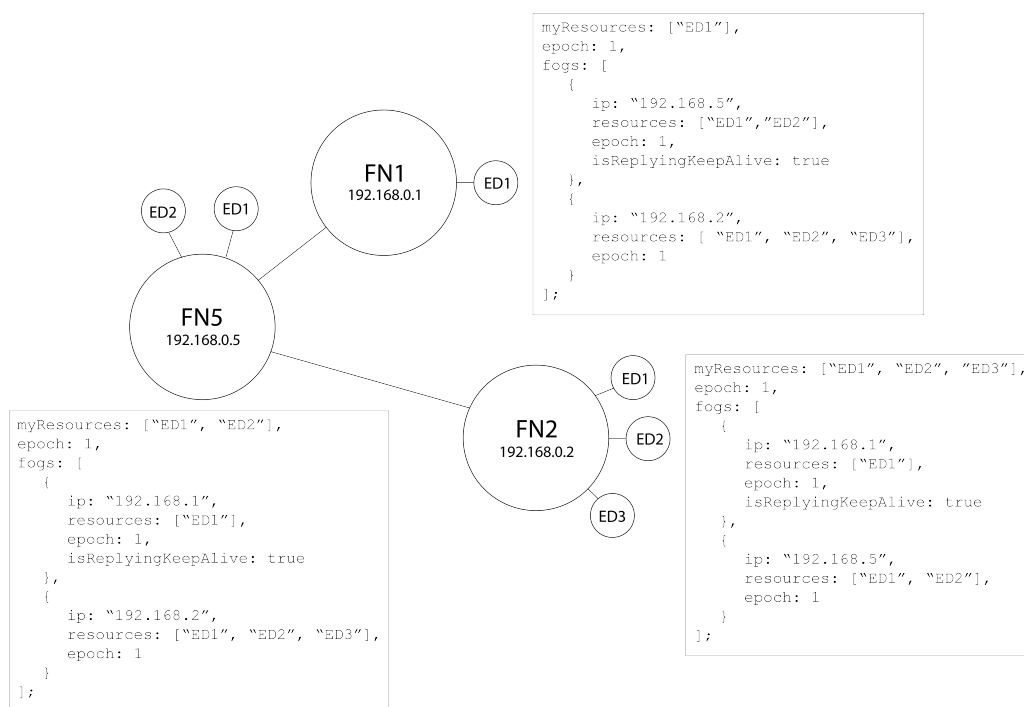


Figura 4.2 – Topologia base para cenários de teste.

O primeiro item da listagem acima não será demonstrado agora, uma vez que as Subseções 3.3.1 e 3.3.2 já o fizeram, portanto, partiremos diretamente para o segundo item.

O segundo item trata-se de quando um nodo deixa de responder mensagens de keep alive, e a Figura 4.2 será utilizada para ilustrar esse funcionamento. Na Figura 4.3, que representa a continuação da Figura 4.2, o nodo FN1 não respondeu no tempo previamente estipulado a mensagem de keep alive enviada pelo nodo FN2, portanto, o nodo destinatário foi marcado em FN2 como parcialmente inativo.

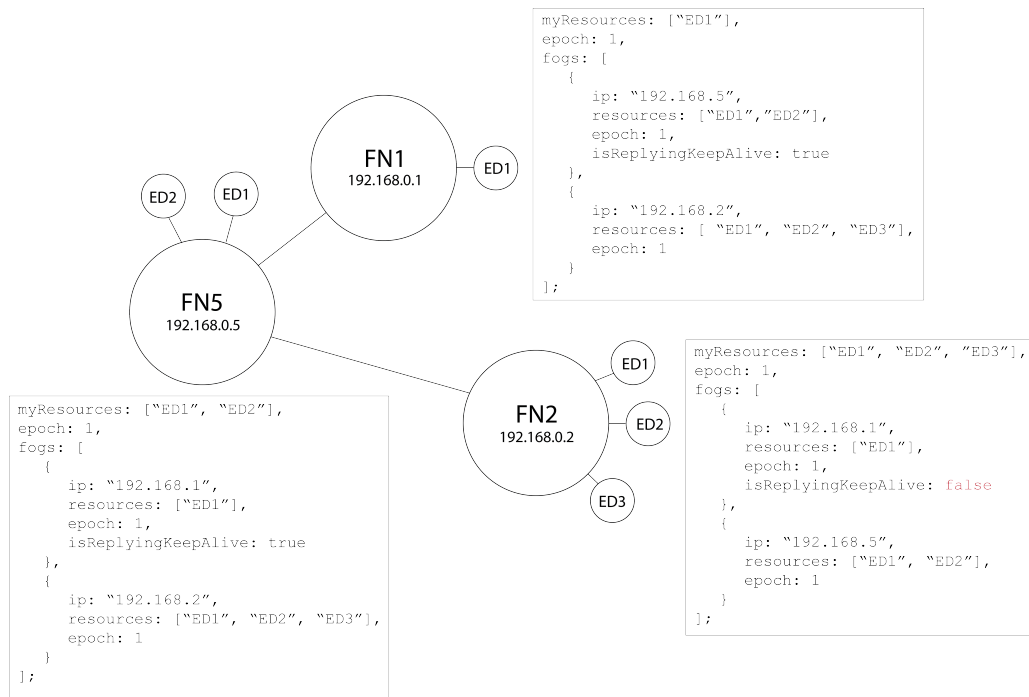


Figura 4.3 – Nodo marcado como parcialmente inativo.

Já a Figura 4.4 representada como continuação da Figura 4.3, demonstra que o nodo FN1 não respondeu novamente à mensagem de keep alive enviada por FN2, por isso, foi removido de sua lista de recursos.

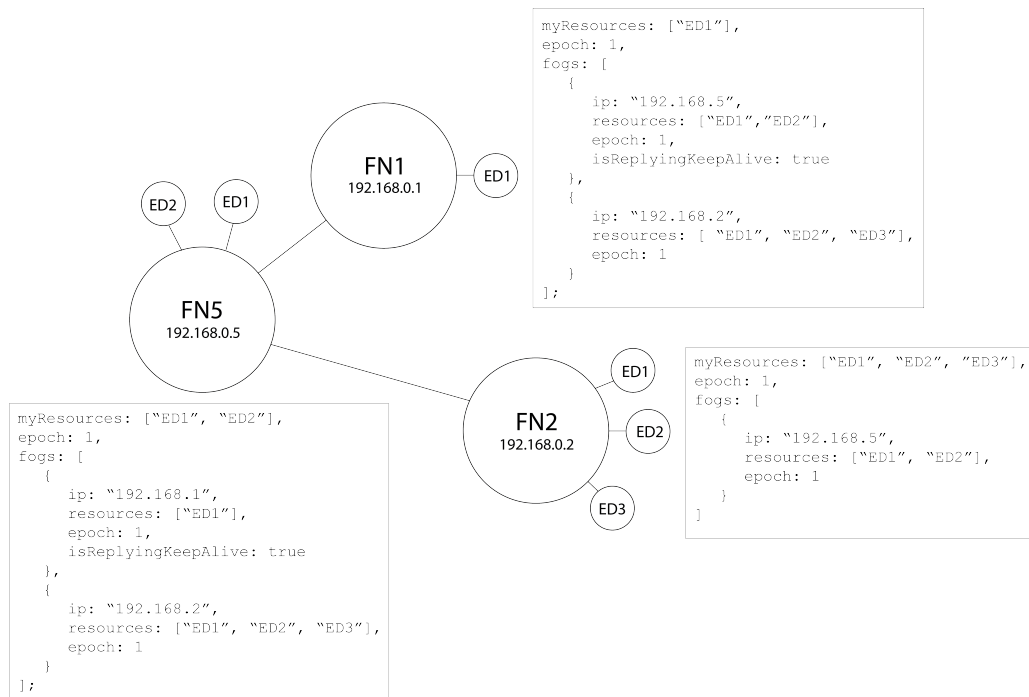


Figura 4.4 – Nodo removido da lista de recursos.

O terceiro item dos cenários de teste trata de quando o nodo está normalmente em operação, porém, um de seus recursos foi alterado, seja por incremento de um novo edge device ou pela remoção de um.



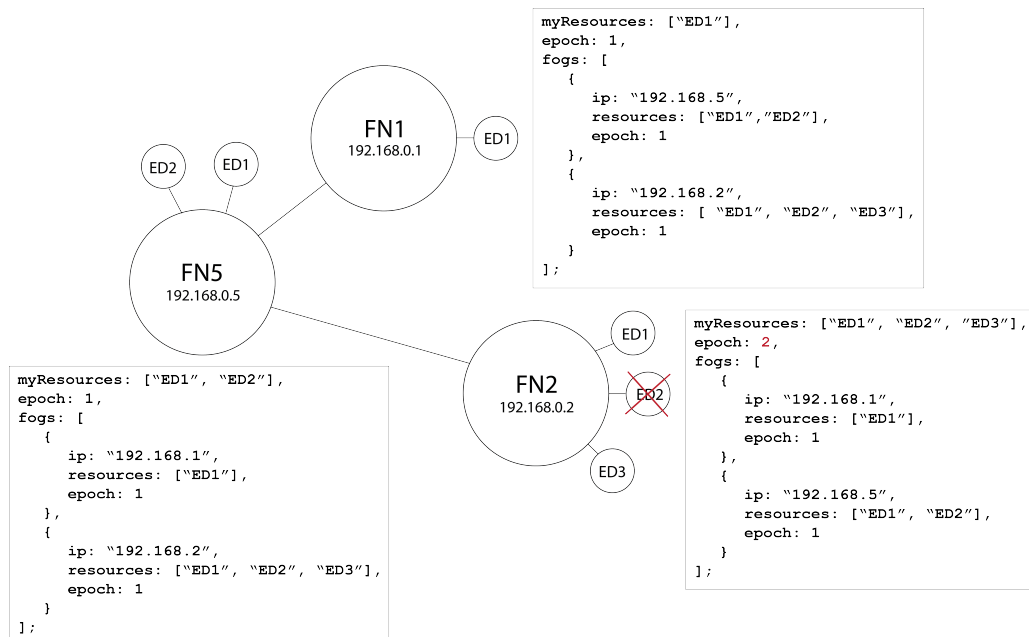


Figura 4.5 – Edge device fora de operação.

Utilizaremos a Figura 4.5 como exemplo básico, e nela podemos observar que o edge device denominada ED2 vinculado ao nodo FN2 parou de funcionar. Para este cenário, o nodo que hospeda ED2 deve estar ciente que este edge device está inoperante, e assim atualizar a sua *época*. O nodo que possui o edge device inoperante, então, terá sua *época* enviada juntamente com as mensagens de keep alive. Assim, quando o nodo receber esta mensagem poderá comparar a *época* armazenada em sua lista de fogs com a *época* recebida pelo keep alive, e após a validação da divergência realizará um Requisição diretamente ao nodo para que possa atualizar sua lista de recursos globais.

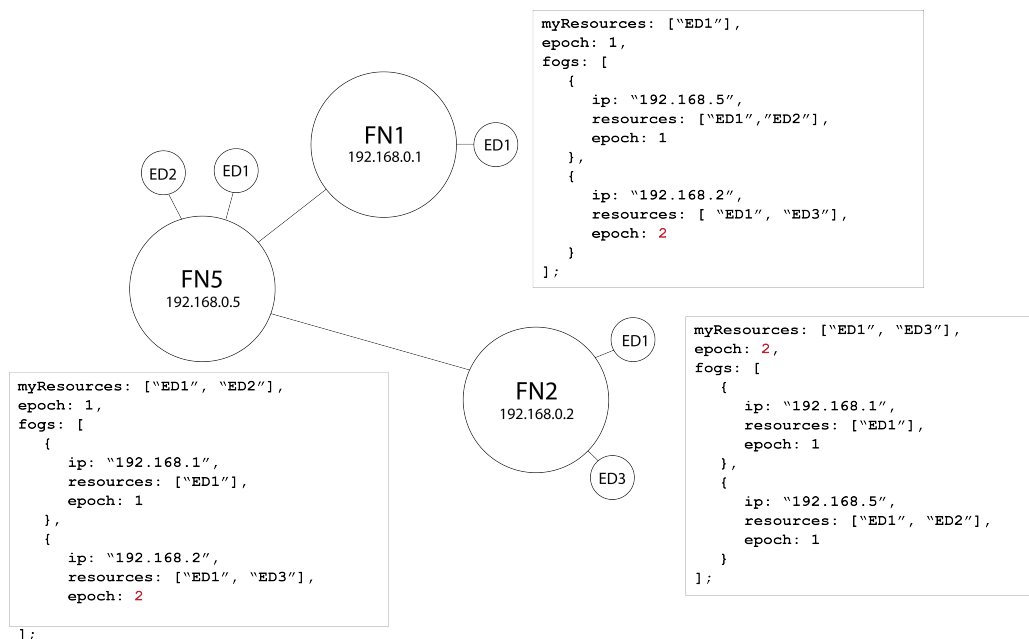


Figura 4.6 – Estado da névoa após atualização de recursos.

### 4.3 Estudos de caso

As Subseções a seguir foram criadas com o intuito de ilustrar os testes previamente descritos, e para tal, dois cenários serão propostos.

#### 4.3.1 Segurança residencial

Como cenário para esse estudo de caso, podemos imaginar um bairro no qual os moradores estejam preocupados com a segurança de suas propriedades. Partindo dessa suposição, as casas poderão contar com diversos atuadores tais como sensores de presença, câmeras de vigilância, luzes automatizadas, etc.. Em vista disso, cada casa pode ser considerada um fog node conectado a diversos edge devices.

Sabendo que o protocolo exposto atua no mapeamento e na sincronização de recursos, cada casa do bairro passa a dispor dessas funcionalidades. Dessa maneira, as casas terão condições de monitorar eventos suspeitos que possam estar acontecendo nas redondezas. Um exemplo de monitoramento está no uso das imagens captadas pelas câmeras de vigilância instaladas na casa A, mas acessados a partir da casa B. Além do mais, cada sensor adicionado ou removido estará, automaticamente, disponível a todos os fog nodes do bairro.

#### 4.3.2 Métricas no agronegócio

Como cenário para esse estudo de caso, podemos imaginar uma fazenda que faz criação de porcos. Nessa fazenda os animais se alimentam em coxos individuais que possuem sensores que captam a temperatura, o peso, e a identificação do animal, bem como medem os níveis de comida e água que são consumidos.

Quando o animal entra nesse ambiente, o fog node é ligado e seus sensores começam a atuar nas medições. Enquanto isso, um fog node com conexão a Internet é encarregado de coletar os dados, processá-los e enviá-los a um servidor de armazenamento na nuvem. Por fim, com o objetivo de economizar energia elétrica, os fog nodes e os sensores são colocados em modo standby quando os animais deixam o local de alimentação. A Figura 4.7 ilustra o cenário previamente exposto. Os detalhes sobre depuração são apresentados no Anexo A.

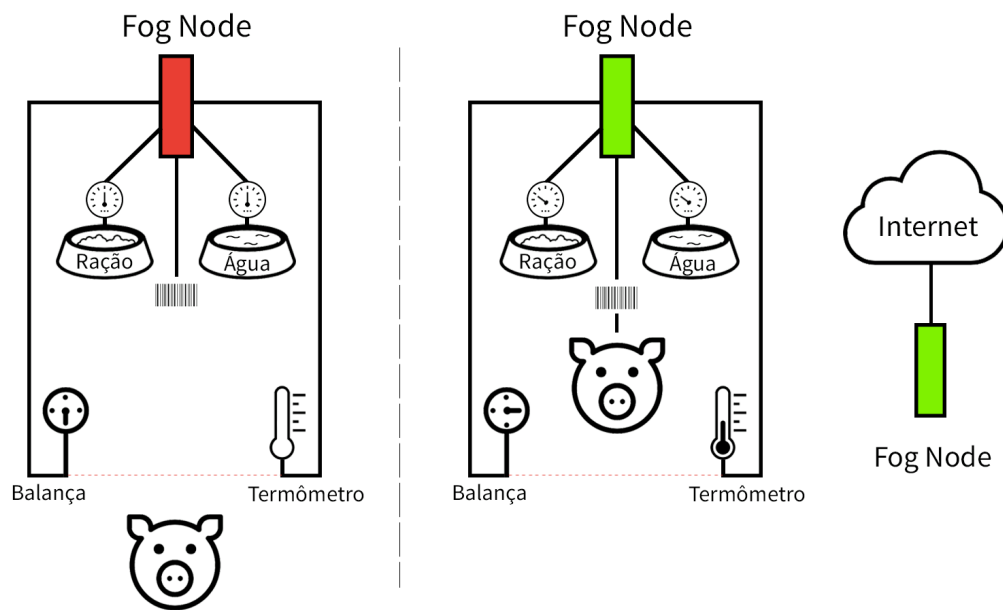


Figura 4.7 – Métricas alimentares na criação de porcos.

## REFERÊNCIAS BIBLIOGRÁFICAS

- [1] “Alpine linux is a security-oriented, lightweight linux distribution based on musl libc and busybox.” Capturado em: <https://alpinelinux.org>, 2018.
- [2] “Docker containerization.” Capturado em: <https://www.docker.com>, 2018.
- [3] “Docker sdk for python.” Capturado em: <https://docker-py.readthedocs.io/en/stable/>, 2018.
- [4] “Should you consider fog computing for your iiot?.” Capturado em: [https://www.moxa.com/newsletter/connection/2017/11/feat\\_02.htm](https://www.moxa.com/newsletter/connection/2017/11/feat_02.htm), 2018.
- [5] Berners-Lee, T.; Fielding, R.; Masinter, L. “Rfc 3986, uniform resource identifier (uri): Generic syntax”. Capturado em: <http://rfc.net/rfc3986.html>, 2005.
- [6] Fielding, R.; Gettys, J.; Mogul, J.; Frystyk, H.; Masinter, L.; Leach, P.; Berners-Lee, T. “Rfc 2616, hypertext transfer protocol – http/1.1”. Capturado em: <http://www.rfc.net/rfc2616.html>, 1999.
- [7] Foundation, L. “Iotivity”. Capturado em: <https://iotivity.org/documentation/architecture-overview>, 2018.
- [8] Foundation, P. S. “A package manager for python packages”. Capturado em: <https://pypi.org/project/pip/>, 2018.
- [9] Freed, N.; Borenstein, D. N. S. “Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types”. Capturado em: <https://rfc-editor.org/rfc/rfc2046.txt>, Nov 1996.
- [10] Hammer-Lahav, E.; Nottingham, M. “Defining Well-Known Uniform Resource Identifiers (URIs)”. Capturado em: <https://rfc-editor.org/rfc/rfc5785.txt>, Abr 2010.
- [11] Hwang, K.; Dongarra, J.; Fox, G. C. “Distributed and Cloud Computing: From Parallel Processing to the Internet of Things”. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011, 1st ed..
- [12] Laboratory, U. N. R. “Common open research emulator (core)”. Capturado em: <https://www.nrl.navy.mil/itd/ncs/products/core>, 2018.
- [13] Lewson, S. “Fog protocol and fogkit: A json-based protocol and framework for communication between bluetooth-enabled wearable internet of things devices”, 06 2015.
- [14] Mell, P. M.; Grance, T. “Sp 800-145. the nist definition of cloud computing”, Relatório Técnico, Gaithersburg, MD, United States, 2011.

- [15] of Things Agenda, I. "Definition fog computing". Capturado em: <https://internetofthingsagenda.techtarget.com/definition/fog-computing-fogging>, 2016.
- [16] Rekhter, Y.; Li, T. "A border gateway protocol 4 (bgp-4)", 1995.
- [17] Rescorla, E.; Modadugu, N. "Datagram Transport Layer Security Version 1.2". Capturado em: <https://rfc-editor.org/rfc/rfc6347.txt>, Jan 2012.
- [18] Roman, R.; Lopez, J.; Mambo, M. "Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges", *CoRR*, vol. abs/1602.00484, 2016, 1602.00484.
- [19] Rouse, M. "Cisco systems, inc." Capturado em: <https://whatis.techtarget.com/definition/Cisco-Systems-Inc>, 2018.
- [20] Rouse, M. "Edge device." Capturado em: <https://whatis.techtarget.com/definition/edge-device>, 2018.
- [21] Shelby, Z. "Constrained RESTful Environments (CoRE) Link Format". Capturado em: <https://rfc-editor.org/rfc/rfc6690.txt>, Ago 2012.
- [22] Shelby, Z.; Hartke, K.; Bormann, C. "The Constrained Application Protocol (CoAP)". Capturado em: <https://rfc-editor.org/rfc/rfc7252.txt>, Jun 2014.
- [23] Tanenbaum, A.; Wetherall, D.; Translations, O. "Redes de computadores". PRENTICE HALL BRASIL, 2011.
- [24] Tanenbaum, A. S.; Steen, M. v. "Distributed Systems: Principles and Paradigms (2Nd Edition)". Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006.
- [25] Tanganelli, G. "Coapthon is a python library to the coap protocol aligned with the rfc". Capturado em: <https://github.com/FelipeBrizola/CoAPthon>, 2018.
- [26] Vaquero, L. M.; Rodero-Merino, L. "Finding your way in the fog: Towards a comprehensive definition of fog computing", *SIGCOMM Comput. Commun. Rev.*, vol. 44–5, Out 2014, pp. 27–32.

## ANEXO A – Depuração de métricas no agronegócio

A fim de simplificar a demonstração, o cenário proposto contém apenas um curral e um fog node reponsável por coletar os dados, processá-los e enviá-los a um servidor de armazenamento na nuvem. Esse contexto está representado na Figura 4.7.

Em um primeiro momento, conforme a Figura A.1, o curral está vazio, portanto, o nodo responsável pelas medições encontra-se desligado e apenas o nodo responsável pelo processamento das informações encontra-se em operação.

```
MY IP: 192.168.0.2
MY RESOURCES: []
MY EPOCH: 0
MY IS_REPLYING_KEEPALIVE: True
```

Figura A.1 – Fog node desligado enquanto curral permanece vazio.

No momento em que o animal adentra o curral, o fog node e seus edge devices são ligados. Assim, os recursos passam a fazer parte da névoa, pois suas informações são propagadas utilizando protocolo de Resource Mapping, conforme Figura A.2. Ao sair do local de alimentação, o nodo e seus sensores são desligados e, assim, a rede volta ao estado representado pela Figura A.1.

```
MY IP: 192.168.0.3
MY RESOURCES: </identifier>;ct=50,</weight>;ct=50,</food>;ct=50,</temperature>;ct=0,</water>;ct=50,
MY EPOCH: 2
MY IS_REPLYING_KEEPALIVE: True
  IP: 192.168.0.2
  RESOURCES: None
  EPOCH: 0
  REPLYING_KEEPALIVE: True

MY IP: 192.168.0.2
MY RESOURCES: []
MY EPOCH: 0
MY IS_REPLYING_KEEPALIVE: True
  IP: 192.168.0.3
  RESOURCES: </identifier>;ct=50,</weight>;ct=50,</food>;ct=50,</temperature>;ct=0,</water>;ct=50,
  EPOCH: 2
  REPLYING_KEEPALIVE: True
```

Figura A.2 – Fog node ligado enquanto curral permanece ocupado.