PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
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. 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)

LISTA DE FIGURAS

Figura 1.1 – Organização arquitetural. 9
Figura 2.1 – Requisição e resposta utilizando mensagens não confirmáveis. 12
Figura 3.1 – Topologia LAN. 17
Figura 3.2 – Arquitetura de um fog node. 18
Figura 3.3 – Arquitetura Resource-Mapping. 18
Figura 3.4 – Pilha de protocolos. 19
Figura 3.5 – Detalhamento Resource Mapping. 20
Figura 3.6 – Nodo entrando na névoa. 21
Figura 3.7 – Nodos realizando Requisição. 22
Figura 4.1 – Fog node container. 26
Figura 4.2 – Topologia base para cenário de teste. 27
Figura 4.3 – Nodo marcado como parcialmente inativo. 28
Figura 4.4 – Nodo removido da lista de recursos. 29
Figura 4.5 – Edge device fora de operação. 29
Figura 4.6 – Estado da névoa após atualização de recursos. 30

LISTA DE TABELAS

Tabela 2.1 – Comparação de funcionalidades. 15

LISTA DE ALGORITMOS

Algoritmo 3.1 – Política de atualização de recursos 23

LISTA DE SIGLAS

BGP – Border Gateway Protocol

COAP – Constrained Application Protocol

DTLS – Datagram Transport Layer Security

HTTP – Hypertext Transfer Protocol

IP – Internet Protocol

LE – Low Energy

NIST – National Institute of Standards and Technology

REST – Representational State Transfer

SDK – Software Development Kit

TCC – Trabalho de Conclusão de Curso

TCP – Transmission Control Protocol

RFC – Request for Comments

UDP – User Datagram Protocol

SUMÁRIO

1 INTRODUÇÃO **8**
1.1 CONTEXTUALIZAÇÃO 8
1.2 MOTIVAÇÃO E JUSTIFICATIVA 9
1.3 OBJETIVO 9

2 FUNDAMENTAÇÃO TEÓRICA **11**
2.1 TAXONOMIAS EM ARQUITETURAS DE COMPUTADORES 11
2.2 MÁQUINAS MULTIPROCESSADAS 11
2.3 MODELOS PARA TROCA DE MENSAGENS EM APLICAÇÕES 11
2.4 PROTOCOLOS RELACIONADOS 11
2.4.1 COAP 11
2.4.2 CONSTRAINED RESTFUL ENVIRONMENTS (CORE) LINK FORMAT 12
2.4.3 MQTT - MESSAGE QUEUE TELEMETRY TRANSPORT 13
2.5 TRABALHOS RELACIONADOS 14

3 PROJETO **16**
3.1 ARQUITETURA 16
3.1.1 PREMISSAS 18
3.2 FORMATO DE MENSAGENS 19
3.3 MÓDULOS 20
3.3.1 DESCOBERTA DE RECURSOS 21
3.3.2 GERENCIAMENTO DE RECURSOS 22

4 AMBIENTE DE SIMULAÇÃO E ESTUDOS DE CASO **26**
4.1 AMBIENTE DE SIMULAÇÃO 26
4.2 CENÁRIOS DE TESTE 27
4.3 ESTUDOS DE CASO 30

REFERÊNCIAS **31**

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 [15], 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.

Como a computação em nuvem não é panacéia[1], podemos utilizar a definição de Bonomi, Milito, Zhu e Addepalli [7] que relata que a computação em nuvem libera as empresas e os usuários finais de muitos detalhes de especificações. Essa facilidade torna-se um problema para aplicações sensíveis à latência, que requerem que nós 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[2] 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 [19].

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[16].

[1] Mecanismos ou práticas que, hipoteticamente, são capazes de solucionar os problemas e/ou dificuldades [5].

[2] Empresa estadunidense líder na fabricação de equipamenteos de rede [20].

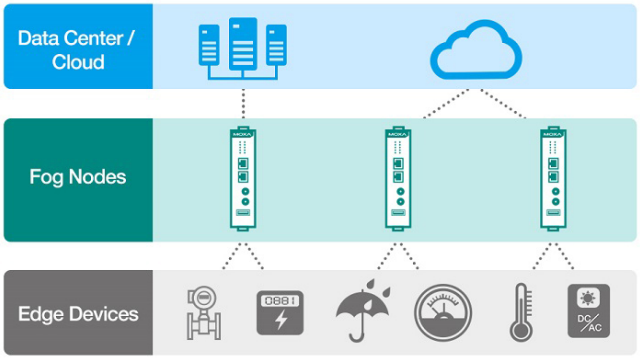

Figura 1.1 – Organização arquitetural.
[4]

Em uma abordagem *bottom-up*, podemos descrever a arquiterura da computação em névoa como um conjunto de *edge devices*[3] que se comunicam com os *fog nodes*, e 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 [19]. A Figura 1.1 representa este conceito de computação em névoa.

1.2 Motivação e justificativa

De acordo com Vaquero e Rodero-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 Rodero-Merino

[3] Dispositivo que controla o fluxo de dados no limite entre duas redes [21].

[26]. Para que esta meta seja atingida, construiremos um protocolo de rede simples que execute em redes locais e que atue de maneira automatizada no processo de descobrimento e sincronização de recursos.

2. FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresentará alguns 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 Taxonomias em Arquiteturas de Computadores

2.2 Máquinas multiprocessadas

2.3 Modelos para troca de mensagens em aplicações

2.4 Protocolos relacionados

2.4.1 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[23].

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 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.

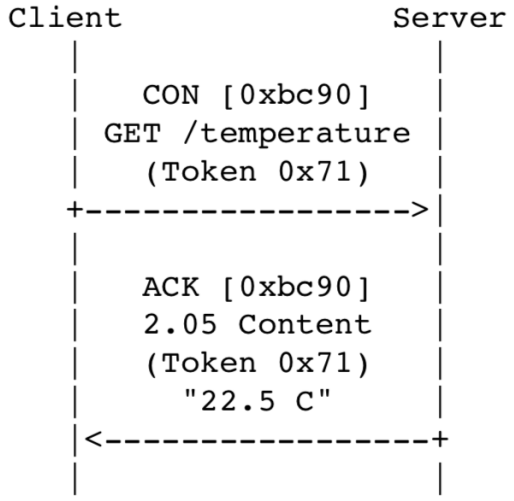

Figura 2.1 – Requisição e resposta utilizando mensagens não confirmáveis. [23]

A arquitetura REST, assim como no protocolo HTTP[8], foi utilizada na projetização 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[23], 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 dispõem de outros princípios comuns ao HTTP e que são conceitos padrão na web, como suporte a URIs[6] e tipos de mídia da Internet(MIME)[11].

Entre o HTTP e CoaP nem tudo são semelhanças, pois, obviamente, este possui particularidades para que consiga atender aos requisitos no qual se propõe a resolver. Entre as diferenças está a troca de mensagens assíncronas utilizando transporte orientado a datagramas, como UDP. Apesar de, naturalmente, o 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[18] estão entre as principais caracteristicas do protocolo de aplicação restrita, CoAP.

2.4.2 Constrained RESTful Environments (CoRE) Link Format

Segundo a RFC-6690, a finalidade desta especificação é realizar REST em nodos com recursos limitados, sendo tal característica importante em aplicações maquina-a-maquina[22]. 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 para atribuir um nome que descreva de forma clara e enxuta um recurso, já o segundo tem por objetivo indicar a interface especifica que interagirá com o recurso destino.

Um aspecto relevante ao CoRE, e de suma importancia para aplicações maquina-a-maquina, é o fato de possuir como URI padrão o prefixo */.well-known/core* , definido na RFC-5785[12]. 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 codigo 2.05 é retornado indicando que a operação transcorreu com sucesso.

2.4.3 MQTT - Message Queue Telemetry Transport

O protocolo MQTT, inicialmente desenvolvido pela IBM, tornou-se um padrão para aplicações de IoT, pois mostrou-se escalável em ambientes com redes instáveis. Essa caracteristica dá-se pelo fato do protocolo implementar um modelo asíncrono de troca de mensagens, deste modo, o emissor mantém-se desacoplado do receptor. Diferente-

mente 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 caracteristicas 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.

2.5 Trabalhos Relacionados

Spencer Lewson implementou um protocolo em nível de aplicação [24] 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 [14].

Sua implementação requer que haja um ponto central de comunicação entre os nodos, uma vez que a conectividade entre eles ocorre 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 [14].

A plataforma Iotivity [9], 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 e 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 da plataforma Iotivity possuir outras funcionalidades, ela não possui mecanismos para manter o mapeamento de recursos em redes locais.

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	sim	sim
Sincronização do map.	não	não	não	não	sim
Recursos observáveis	n/a	n/a	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 uma 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 relativo ao nodo solicitado.

Em razão da topologia distribuída, a arquitetura deve ser capaz de mapear e sincronizar, de forma autônoma, os recursos providos pelos fog nodes. Desta forma, cada *fog node* saberá quais são os *edge devices* disponíveis na rede e, 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.

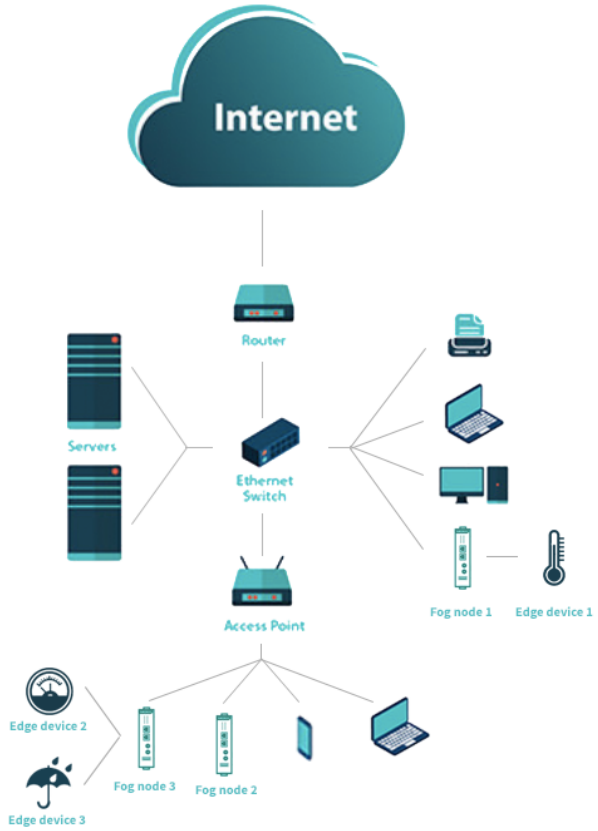

Figura 3.1 – Topologia LAN.

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 modulos: *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.

Baseado na 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 recursos 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*.

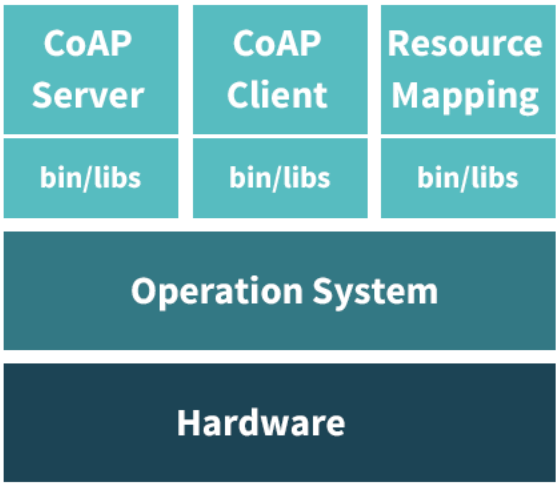

Figura 3.2 – Arquitetura de um fog node.

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 proprios 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 requições recebidas via *Resource-Mapping*.

Figura 3.3 – Arquitetura Resource-Mapping.

3.1.1 Premissas

Para que a solução opere corretamente, tanto no mapeamento quanto na sincronização de recursos, algumas premissas 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*[10], devem estar instalados. Por fim, os dados que os edge devices fornecerão aos fog nodes serão tratados de forma simulada.

3.2 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.

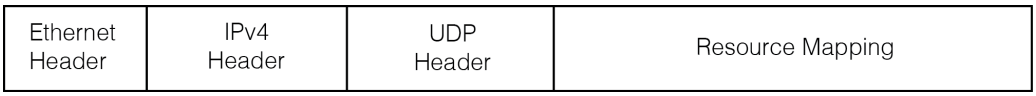

Figura 3.4 – Pilha de protocolos.

O modelo de referência TCP/IP é constituido de cinco camadas: física, enlace, rede, transporte e aplicação [24]. Nesse trabalho, o 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 endereçamento 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, despenderia uma quantidade de trafego desnecessário na rede. Visto que a simplicidade é um dos objetivos deste trabalho e que transitar uma pequena quantidade de dados a cada requisição aumenta o desempenho da solução, a utilização de datagramas UDP faz sentido neste protocolo

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

- O campo Protocol Type é reservado para indicar a forma de encapsulamento do pacote.
- Atualmente existem dois tipos de Messages Types possíveis, keep alive e acknowledgement.

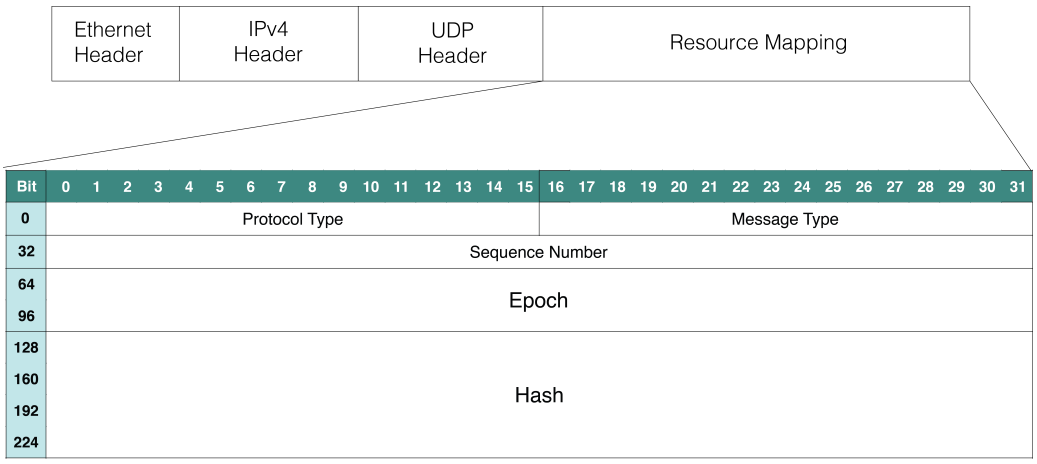

Figura 3.5 – Detalhamento Resource Mapping.

- 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.
- 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 [24] 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.

3.3 Módulos

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

O detalhamento das funcionalides que o projeto deverá dispor, tal como ilustrações relacionadas aos fluxos, serão abordadas nas proximas subseções.

3.3.1 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.

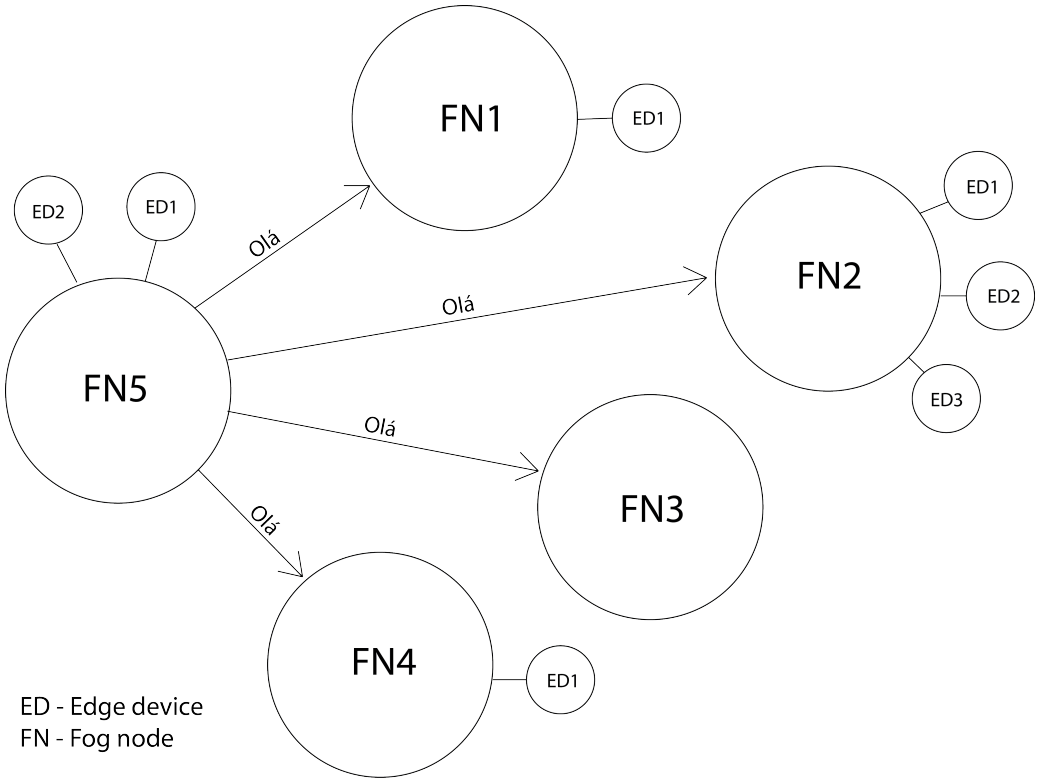

Figura 3.6 – Nodo entrando na névoa.

A topologia da névoa utilizada nas Figuras 3.6 e 3.7 é definida por fog nodes enumerados de um a cinco, sendo o nodo FN5 o último a entrar na rede. A figura 3.6 demonstra o nodo 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 nodos FN1, FN2 e FN4 realizam a Requisição CoAP diretamente ao FN5 afim de obter os recursos disponíbilizados por ele, já o nodo FN3, por não estar executando o protocolo de mapeamento, não.

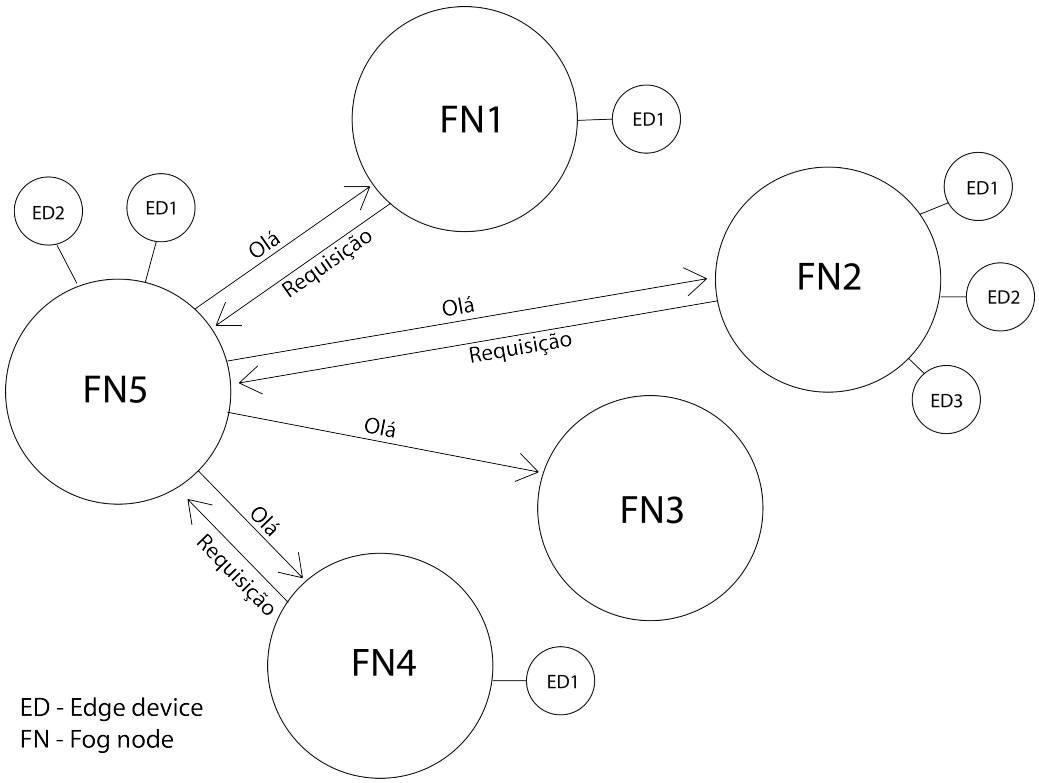

Figura 3.7 – Nodos realizando Requisição.

3.3.2 Gerenciamento de recursos

A manutenibilidade da lista de recursos globais é relevante para que a implementação do protocolo tenha sucesso, 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[17]. 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 implementará 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 boleana, 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 detectarmos esse tipo de comportamento, o protocolo proposto deverá implementar a técnica denominada *época*, e servirá para que os nodos mantenham ciênia 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 unidadade 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 unidadade.

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 mensagen. 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. AMBIENTE DE SIMULAÇÃO E ESTUDOS DE CASO

4.1 Ambiente de simulação

Em um primeiro momento a névoa foi construída de forma simulada utilizando a ferramenta Common Open Research Emulator (CORE)[13]. Para os primeiros experimentos a ferramente 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 e, portanto, a ascalabilidade da solução pode ser amplamente testada[3].

Cabe ressaltar que containers Docker[2] consomem recursos da maquina hospedeira, tais como memória, processamento, disco rígido. Portanto, a criacao de containers da névoa fica atrelado a quantidade de recursos que esse host pode proporcionar.

Cada container executa uma distrbuição Linux chamada Alpine, de aproximadamente 5Mb, que possui apenas as funcioalidades 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 em nível de requisições entre sí.

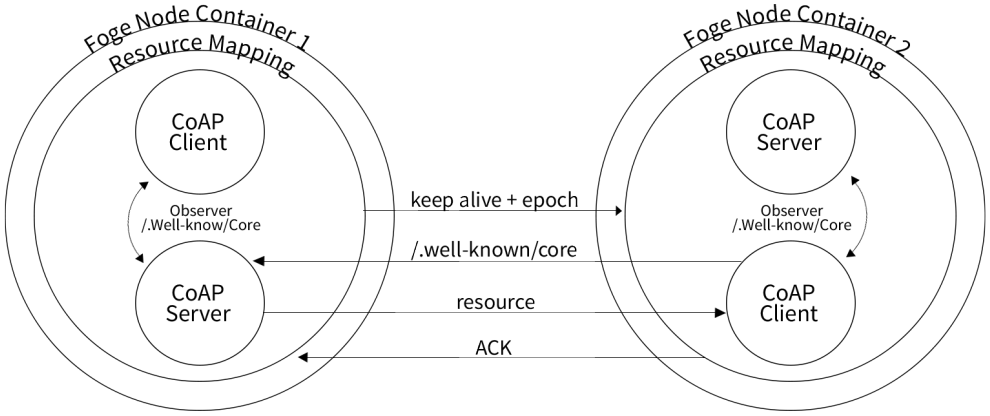

Figura 4.1 – Fog node container.

Além do protocolo de mapeamento em sí, 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, e partindo do pressuposto que todos os nodos da névoa já 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.

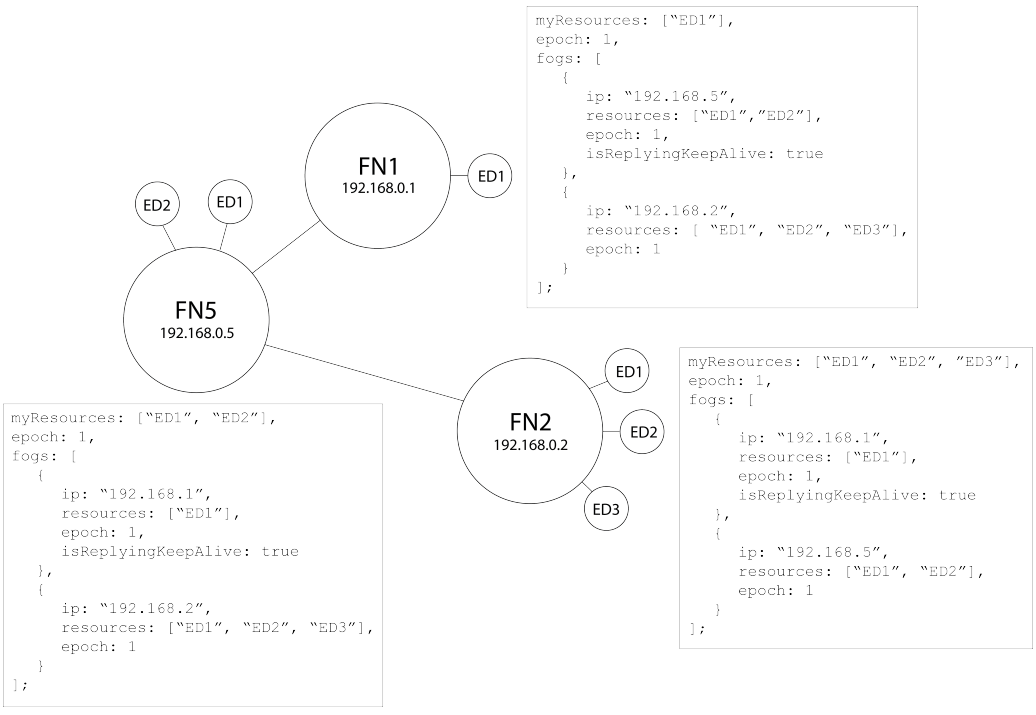

Figura 4.2 – Topologia base para cenário de teste.

O item 1 não será demonstrado agora, uma vez que as Subseções 3.2.2 e 3.2.3 já o fizeram, portanto, partiremos diretamente para o segundo item da listagem.

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 este funcionamento. Na Figura 4.3, 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.

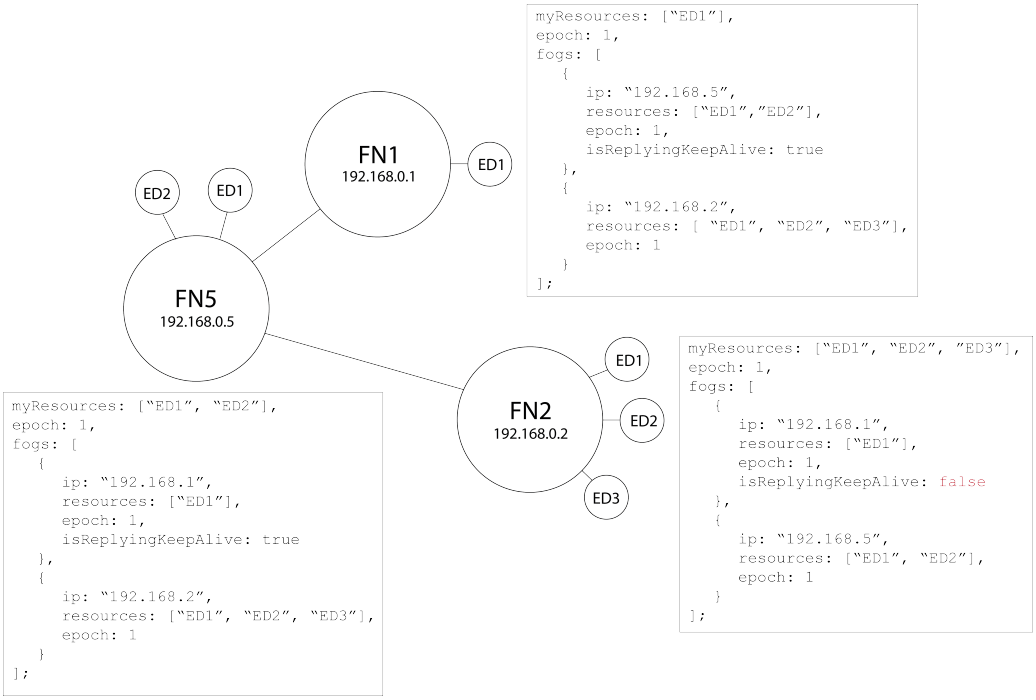

Figura 4.3 – Nodo marcado como parcialmente inativo.

Já a Figura 4.4 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.

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 incremendo de um novo edge device ou pela remoção de um.

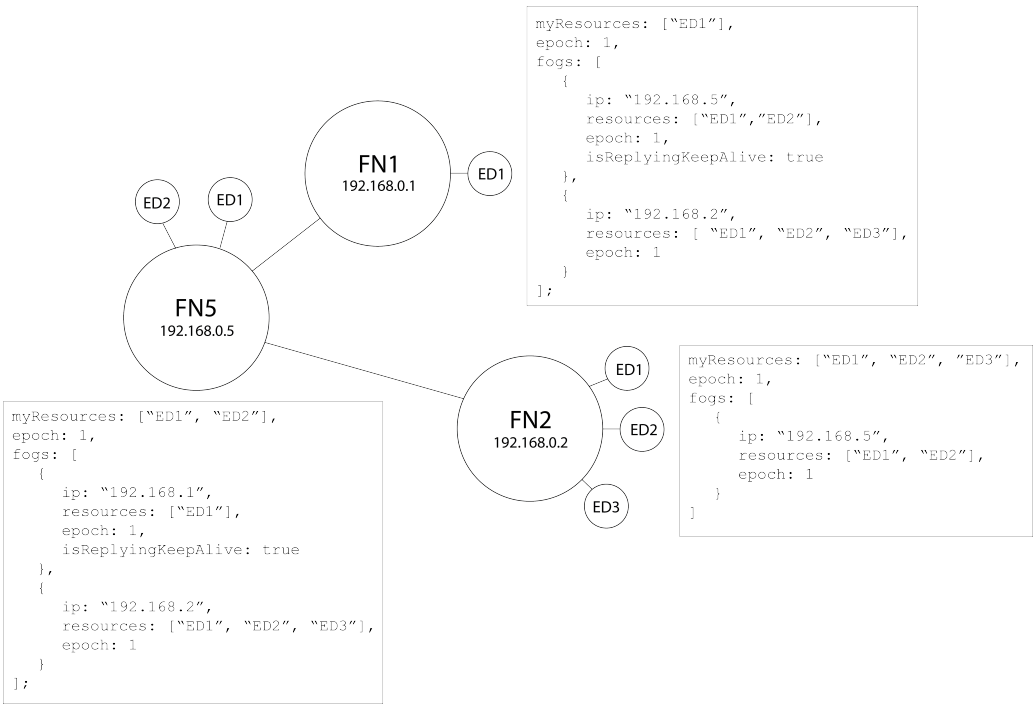

Figura 4.4 – Nodo removido da lista de recursos.

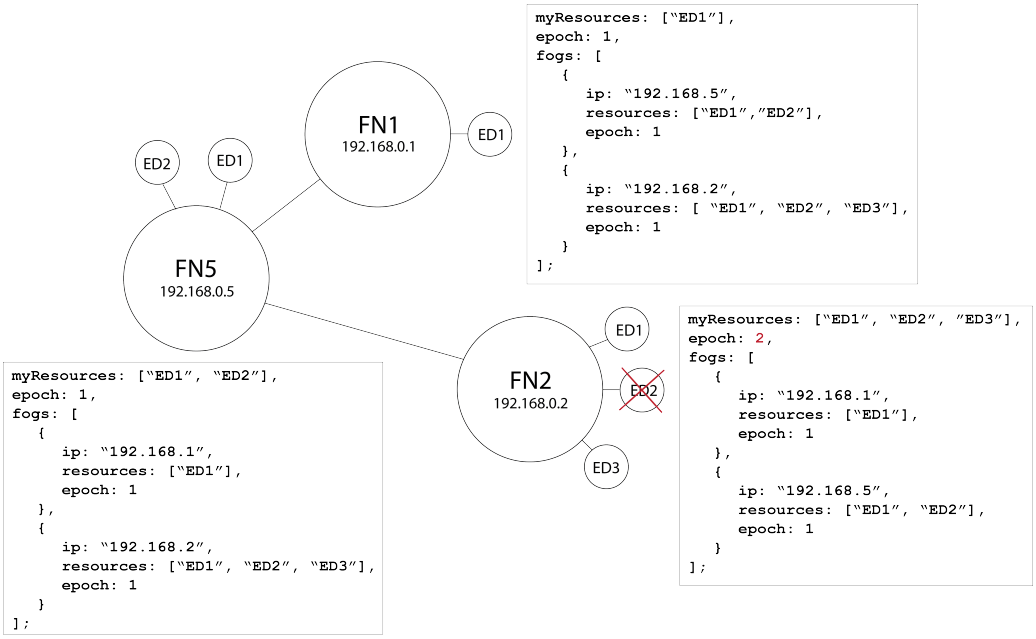

Figura 4.5 – Edge device fora de operação.

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 divergencia realizará um Requisição diretamente ao nodo para que possa atualizar sua lista de recursos globais.

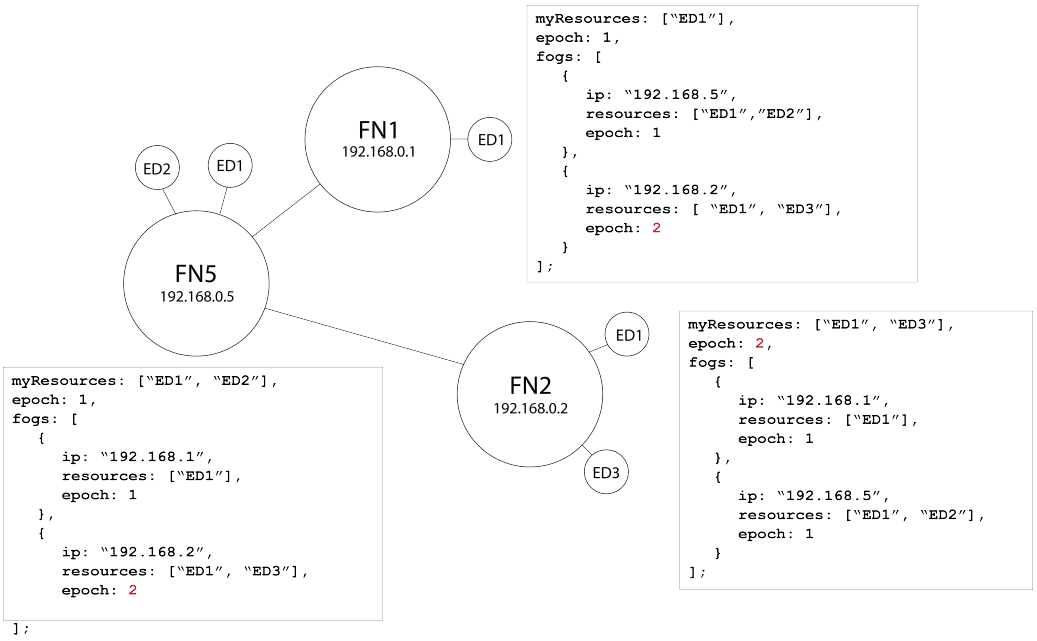

Figura 4.6 – Estado da névoa após atualização de recursos.

4.3 Estudos de caso

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, 2018.

[5] “Significado de panaceia.” Capturado em: https://www.dicio.com.br/panaceia/, 2018.

[6] Berners-Lee, T.; Fielding, R.; Masinter, L. “Rfc 3986, uniform resource identifier (uri): Generic syntax”. Capturado em: http://rfc.net/rfc3986.html, 2005.

[7] Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. “Fog computing and its role in the internet of things”. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, 2012, pp. 13–16.

[8] 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.

[9] Foundation, L. “Iotivity”. Capturado em: https://iotivity.org/documentation/architecture-overview, 2018.

[10] Foundation, P. S. “A package manager for python packages”. Capturado em: https://pypi.org/project/pip/, 2018.

[11] 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.

[12] Hammer-Lahav, E.; Nottingham, M. “Defining Well-Known Uniform Resource Identifiers (URIs)”. Capturado em: https://rfc-editor.org/rfc/rfc5785.txt, Abr 2010.

[13] Laboratory, U. N. R. “Common open research emulator (core)”. Capturado em: https://www.nrl.navy.mil/itd/ncs/products/core, 2018.

[14] Lewson, S. “Fog protocol and fogkit: A json-based protocol and framework for communication between bluetooth-enabled wearable internet of things devices”, 06 2015.

[15] Mell, P. M.; Grance, T. “Sp 800-145. the nist definition of cloud computing”, Relatório Técnico, Gaithersburg, MD, United States, 2011.

[16] of Things Agenda, I. “Definition fog computing”. Capturado em: https://internetofthingsagenda.techtarget.com/definition/fog-computing-fogging, 2016.

[17] Rekhter, Y.; Li, T. “A border gateway protocol 4 (bgp-4)”, 1995.

[18] Rescorla, E.; Modadugu, N. “Datagram Transport Layer Security Version 1.2”. Capturado em: https://rfc-editor.org/rfc/rfc6347.txt, Jan 2012.

[19] 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.

[20] Rouse, M. “Cisco systems, inc.” Capturado em: https://whatis.techtarget.com/definition/Cisco-Systems-Inc, 2018.

[21] Rouse, M. “Edge device.” Capturado em: https://whatis.techtarget.com/definition/edge-device, 2018.

[22] Shelby, Z. “Constrained RESTful Environments (CoRE) Link Format”. Capturado em: https://rfc-editor.org/rfc/rfc6690.txt, Ago 2012.

[23] Shelby, Z.; Hartke, K.; Bormann, C. “The Constrained Application Protocol (CoAP)”. Capturado em: https://rfc-editor.org/rfc/rfc7252.txt, Jun 2014.

[24] Tanenbaum, A.; Wetherall, D.; Translations, O. “Redes de computadores”. PRENTICE HALL BRASIL, 2011.

[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.