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 I 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

LISTA DE ALGORITMOS

LISTA DE SIGLAS

BGP – Border Gateway Protocol

COAP – Constrained Application Protocol

HTTP – Hypertext Transfer Protocol

IP – Internet Protocol

LE – Low Energy

NIST – National Institute of Standards and Technology

TCC – Trabalho de Conclusão de Curso

TCP – Transmission Control Protocol

RFC – Request for Comments

UDP – User Datagram Protocol

SUMÁRIO

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.

Como a computação em nuvem não é panacéia[1], podemos utilizar a definição de Bonomi, Milito, Zhu e Addepalli [4] 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 provendo armazenamento, computação e serviços de rede entre dispositivos finais e os servidores na nuvem [17].

Atualmente a computação em névoa tornou-se um paradigma próprio e não mais uma mera extensão da computação em nuvem. Esse paradigma criou o conceito de *fog nodes*, que abrangem desde dispositivos finais com baixa capacidade computacional até servidores poderosos na nuvem. Assim, os *fog nodes* passam a fazer parte da implementação dos serviços em nuvem. O que torna a computação em névoa interessante é a capacidade dessa variedade de dispositivos cooperarem uns com os outros de forma distribuída.

A Figura [1] apresenta uma representação do conceito de computação em névoa. Em uma abordagem *bottom-up*, podemos descrever a arquiterura da computação em névoa

[1] Mecanismos ou práticas que, hipoteticamente, são capazes de solucionar os problemas e/ou dificuldades [2].

[2] Empresa estadunidense líder na fabricação de equipamenteos de rede [18].

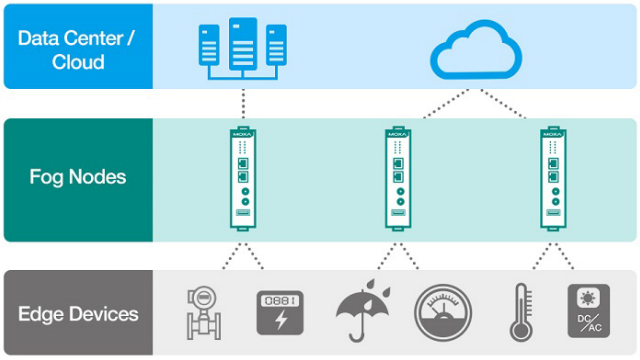

Figura 1.1 – Organização arquitetural.
[1]

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 nao é essencial para a execução dos serviços em névoa [17].

1.2 Motivação e justificativa

De acordo com Vaquero e Rodero-Merino [23], serão sete os desafios que a computação em névoa deverá enfrentar para tornar-se 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 gerenciando sensores ou um computador, mapeie 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 da existência 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 [23]. Para que esta meta seja atingida, construiremos um protocolo de rede simples que

[3] Dispositivo que controla o fluxo de dados no limite entre duas redes [19].

execute em redes locais e que atue de maneira automatizada para que o processo de descobrimento e sincronização de recursos transcorra de forma transparente.

2. FUNDAMENTAÇÃO TEÓRICA

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

2.1 BGP

O protocolo BGP está situado na quinta camada, a camada de aplicação, do modelo de referência TCP/IP [22]. Abaixo, de forma sucinta, elencaremos algumas funcionalidades básicas do protocolo que embasarão o restante deste trabalho.

- A responsabilidade deste protocolo é manter a troca de informações sobre roteamentos entre sistemas autônomos [15].
- O roteador ao entrar na rede pela primeira vez deve-se conectar ao seu vizinho. Após a conexão estabelecida, os roteadores compartilham entre sí suas tabelas de roteamento [15].
- Posteriormente, as atualizações nas tabelas dos roteadores dão-se de forma incremental à medida que as mudanças na rotas ocorrem [15].
- Mensagens de *keep alive* são trocadas periodicamente a fim de garantir conectividade entre os roteadores [15].

2.2 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[21].

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. 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.

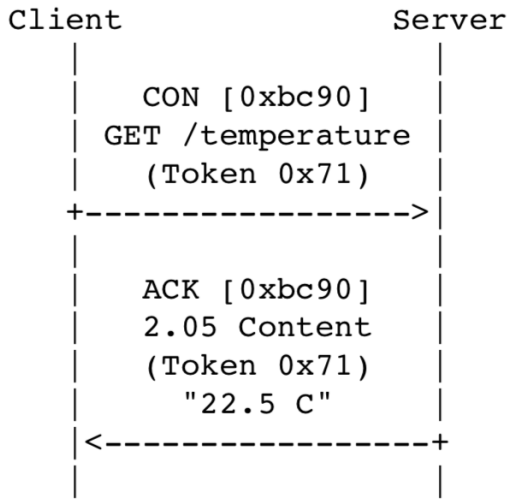

Figura 2.1 – Requisição e resposta utilizando mensagens não confirmáveis. [21]

A arquitetura REST, assim como no protocolo HTTP[7], 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[21], que converte o método ou tipo de resposta, tipo de mídia e opções para os valores HTTP correspondentes.

Além de modelo de comunicação cliente/servidor e arquitetura REST, o CoAP também dispõem de outros princípios comuns ao HTTP e que são conceitos padrão na web como suporte a URIs[3] e tipos de mídia da Internet(MIME)[10].

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[16] estão entre as principais caracteristicas do protocolo de aplicação restrita, CoAP.

2.3 Constrained RESTful Environments (CoRE) Link Format

Segundo a RFC-6690, a finalidade deste especificação é realizar REST em nodos com recursos limitados sendo importante em aplicações maquina-a-maquina[20]. Em aplicação deste tipo é primordial que as configurações não dependam de interação de humanos, e portanto, que mantenham 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[11]. Este prefixo é utilizado para que o servidor exponha suas políticas e recursos disponíveis.

O protocolo CoAP preve suporte à CoRE e ao prefixo */.well-known/core*. Contanto com esses recursos, é possível traçarmos uma pequena analogia entre esquemas HTTP/HTTPS e *coap*/*coaps*, este utilizando segurança DTLS. Estes esquemas são utilizados para identificar e localizar os recursos coap na rede. Sendo assim, servidores coap ficam aguardando por requisições que utilizem tal esquema.

Para exemplificarmos, abaixo esta 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, porem 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 Trabalhos Relacionados

Spencer Lewson implementou um protocolo em nível de aplicação [22] 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 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 [13].

3. PROPOSTA DE TRABALHO

Seguindo a organização arquitetural da Figura 1.1, a proposta deste trabalho é fazer com que um *fog node* conheça, de forma autônoma, os recursos disponibilizados por seus vizinhos. Assim, 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 na rede capaz de medir a temperatura, por exemplo.

Podemos observar na topologia da Figura 1.1, que os *fog nodes* não possuem uma um nodo central como servidor. Em razão da topologia distribuída, se for necessário escalarmos a quantidade de nodos na rede, a mesma não deverá sofrer impactos significativos de performance.

Nas seções a seguir abordaremos a arquitetura, módulos e submódulos que compõem o projeto, bem como resultados esperados, validações e cenários de teste.

3.1 Arquitetura

Esta Seção define a pilha de protocolos a serem utilizados neste projeto, bem como a justifica pela escolha dos mesmos. 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 3.1 explicita a pilha de protocolos que o projeto fará uso para implementar as funcionalidades propostas.

Figura 3.1 – Pilha de protocolos.

O modelo de referência TCP/IP é constituido de cinco camadas: física, enlace, rede, transporte e aplicação [22]. 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

O protocolo proposto, entitulado *Resource Mapping*, como apresentado na Figura 3.1, atuará na camada de aplicação do modelo de referência TCP/IP [22] 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.

3.2 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, bem como ilustrações relacionadas aos fluxos, serão abordadas nas proximas subseções.

3.2.1 Middleware

Observando a Figura 1.1, notamos que há comunicação entre um fog node e seus respectivos edge devices. A comunicação entre estes não é o foco deste projeto, portanto, será tratada de forma simulada. Assim sendo, a simulação deverá prever alguns casos que, por vezes, possam suceder. Dentre alguns dos eventos possíveis está o acréscimo ou remoção de um edge device vinculado à algum fog node.

A manutenção dos recursos, acréscimo ou remoção, transcorrerá utilizando o protocolo CoAP. Utilizar CoAP para estas funcionalidades faz sentido, uma vez que este prevê meios para vinculação de recursos a nodos.

Segundo a definição de POST do protocolo CoAP, sua função é determinada pelo servidor que está recebendo a requisição e pelo do recurso referenciado na URI. Geralmente a sua utilização resulta na criação ou atualização de um recurso.[21]. Já a mesma RFC define o método DELETE como sendo um método de remoção de recursos baseado em URIs[21].

Nessa perspectiva, portanto, o uso do método POST faz sentido para gerarmos *(CoRE) Link Format* e o método DELETE para removermos.

3.2.2 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 broadcast indicando que possui recursos a serem disponibilizados.

Ao receberem o pacote enviado por broadcast, os nodos que desejarem saber quais recursos estão sendo providos por este novo membro, deverão realizar uma query 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.

Com o objetivo de exemplificar a primeira fase do protocolo, as Figuras 3.2 e 3.3 apresentam o fluxo para a descoberta de recursos.

A topologia da névoa utilizada nas imagens é definida por fog nodes enumerados de um à cinco, sendo o nodo FN5 o ultimo a entrar na rede. A figura 3.2 demonstra o nodo FN5 entrando na névoa e, portanto, deverá anunciar-se por broadcast indicando que possui recursos a serem disponibilizados.

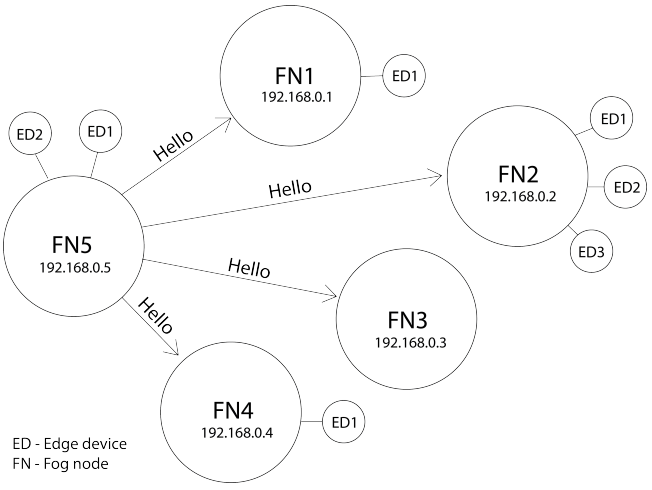

Figura 3.2 – Nodo entrando na névoa.

Após FN5 enviar mensagem “Hello“ por broadcast, os nodos FN1, FN2 e FN4 realizam a query diretamente ao FN5 afim de obter os recursos disponíbilizados por ele, já o nodo FN3, por não estar executando o protocolo de mapeamento, não.

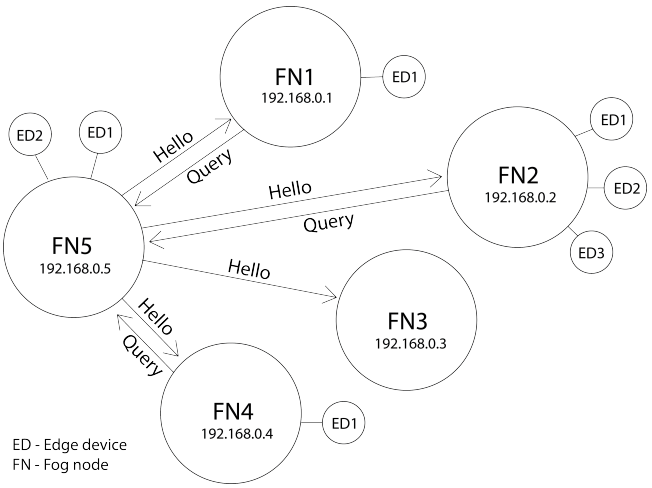

Figura 3.3 – Nodos realizando query.

3.2.3 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 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 3.3 quando necessário. O Pseudocódigo 3.1 demonstra, de forma sucinta, a política de atualização que cada nodo deverá implementar.

```
1: function Police(ip, resources)
2:     if exists(ip)
3:         update(ip, resources);
4:     else
5:         insert(ip, resources);
```

Algoritmo 3.1 – Política de atualização de recursos.

No momento em que o nodo recebe a resposta de sua query, 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 3.3 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, por exemplo. 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 broadcast em um intervalo de trinta segundos, porém, este é apenas um valor inicial, e conforme o desenvolvimento do

projeto esse parâmetro pode ser ajustado para que a solução obtenha uma eficiência satisfatória. 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 sí quanto em cada item da lista de fogs. O valor da época será incrementado em uma unidadade a toda alteração observada no nodo, seja pelo acréscimo ou pela remoção de edge devices.

A partir de agora, então, todas as mensagens de keep alive deverão conter a época do nodo que está realizando o broadcast. 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 divergencia de épocas, este poderá 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;
```

Além do protocolo de mapeamento em sí, este trabalho disponibilizará uma API para o gerenciamento de recursos locais. Esta API deverá ser executada localmente e possuirá métodos para listagem, remoção e criação de recursos, portanto, a partir dela será possível simularmos o gerenciamento, inserção e remoção, de recursos que o protocolo implementará.

3.3 Resultados esperados

Espera-se que este trabalho resulte em um protocolo funcional, simples e que seja capaz de descobrir e sincronizar recursos de dispositivos sob computação em névoa. Além disso, temos como objetivo fazer com que a névoa configure-se de forma autônoma, ou seja, quando um recurso ou nodo entrar ou sair da rede, a mesma deverá manter-se coerente. Esta coerência significa manter o estado global dos recursos acessível a todos os nodos.

A névoa será construída de forma simulada utilizando a ferramenta Common Open Research Emulator (CORE)[12]. Nela é possível criar as mais variadas topologias de redes, além disso, cada nodo adicionado à topologia contém um terminal Unix que pode ser utilizado para executar comandos. Então, os comandos da API descritos acima, podem ser utilizados a fim de auxiliar o gerenciamento dos recursos locais dos nodos. A ferramenta de monitoramento de trafego de rede, Wireshark[24], será utilizada em conjuntos com o CORE, pois, nela será possível apurarmos se os pacotes estão sendo enviados e recebidos da forma correta.

3.4 Validação

A validação deste trabalho consiste em realizar simulações que faça com que o protocolo execute suas funcionalidades de acordo com os resultados descritos na Seção anterior, sendo assim, algumas situações devem ocorrer para que as validações sejam realizadas. Estas situações serão abordadas nos cenários de teste da Seção a seguir.

3.5 Cenários de teste

Utilizando como base a Figura x.x, 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.

O Item 1 não será explicado neste momento, 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 refere-se ao fato de um equipamento deixar de responder as mensagens de keep alive, e para ilustar esse funcionamento as Figuras x.x e x.x1 representam a saída do nodo fn3 da rede e sua marcação como parcialmente ativo e posteriormente a sua remoção da lista por completo.

4. METODOLOGIA E CRONOGRAMA DE DESENVOLVIMENTO

Para construção do projeto utilizaremos Python[8], em sua versão 3.x, como linguagem de programação. Mais precisamente, utilizaremos a interface de baixo nível de rede da biblioteca padrão da linguagem [9]. Algumas ferramentas para auxiliar o desenvolvimento serão utilizadas, como Visual Studio Code[5] para edição de código e GitHub[6] para o versionamento.

O cronograma deste trabalho foi baseado em iteraçõesm como pode ser visto no cronograma abaixo.

Atividades	Março				Abril				Maio				Junho				Julho			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Definição do tema em específico		X	X																	
Elaboração da introdução, motivação e justificativa			X	X																
Elaboração da proposta de trabalho				X	X															
Elaboração da fundamentação teórica					X															
Revisão e refinamento do documento					X															
Entrega do documento						X														
Definição dos capítulos posteriores							X	X												
Aprofundamento teórico							X	X	X	X	X	X	X	X						
Elaboração dos próximos capítulos							X	X	X	X	X	X	X	X	X					
Revisão do documento final															X	X				
Entrega do documento final																	X			

Figura 4.1 – Cronograma de atividades.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] “Should you consider fog computing for your iiot?.” Capturado em: https://www.moxa.com/newsletter/connection/2017/11/feat_02.htm, 2018.

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

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

[4] 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.

[5] Corporation, M. Capturado em: https://code.visualstudio.com, 2018.

[6] Corporation, M. Capturado em: https://github.com, 2018.

[7] 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.

[8] Foundation, P. S. “Python software foundation”. Capturado em: https://python.org, 2018.

[9] Foundation, P. S. “socket — low-level networking interface”. Capturado em: https://docs.python.org/3/library/socket.html, 2018.

[10] 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.

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

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

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

[17] 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.

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

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

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

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

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

[23] 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.

[24] wireshark org. “Wireshark”. Capturado em: https://www.wireshark.org, 2018.