Disseminação de Dados com QoS para Internet das Coisas baseada na Plataforma ContextNet/DDS
Relatório de Atividades de Iniciação Científica

Luan Diniz Mazaro Rodovalho[1], **Ricardo Couto Antunes da Rocha**[2]

[1]Estudante, Instituto de Biotecnologia, luan.rodovalho@discente.ufcat.edu.br

[2]Orientador, Instituto de Biotecnologia, rcarocha@ufcat.edu.br

Resumo. *Cidades Inteligentes visam melhorar a qualidade de vida dos cidadãos através de aplicações inteligentes utilizando-se de dispositivos de Internet das Coisas e softwares conectados em uma rede de larga escala de maneira escalável, rede esta que transmite e recebe dados permitindo aos usuários fazerem uso dos mesmos. Nesse contexto, as redes por proximidade surgem como uma abordagem importante, conectando dispositivos físicos próximos de forma direta e eficiente. Essas conexões sem fio facilitam a gestão de envio de mensagens e a propagação de dados na cidade inteligente. Para auxiliar nesse paradigma partimos da ideia da utilização de um sistema de redes de proximidade para a comunicação entre as áreas, utilizando o middleware DDS (Data Distribution Service). Com o DDS foi possível criar um mecanismo de gateway, que atende aos requisitos funcionais da aplicação, permitindo que cada área envie e receba mensagens dos vizinhos, seguindo o modelo Publish-Subscribe. Dessa forma, o DDS possibilita a comunicação confiável e distribuída entre os dispositivos IoT, permitindo que a cidade inteligente opere de maneira coordenada e eficiente.*

Palavras-chave— QoS, publish/subscribe, Internet das Coisas, Disseminação de Dados, Cidades Inteligentes

1. Apresentação

Cidades inteligentes [Cruz et al. 2019], em inglês "*smart cities*", são cidades que utilizam tecnologias de informação e comunicação para melhorar a qualidade de vida dos seus habitantes e a eficiência dos seus serviços públicos. O objetivo de uma cidade inteligente é explorar o uso de tecnologias para melhorar a vida dos cidadão, representando a dinâmica da cidade por meio de dados produzidos por *crowdsensing* e Internet das Coisas (IoT) para facilitar a tomada de decisões. Plataformas para cidades inteligentes devem responder a desafios como a disseminação de informações em tempo real, adaptação dinâmica da cidade, escalabilidade e segurança.

As cidades inteligentes utilizam tecnologias [Autili et al. 2020] como sensores, câmeras, redes de comunicação e análise de dados para coletar informações sobre o ambiente urbano, como o tráfego, qualidade do ar, consumo de energia, gestão de resíduos, entre outros. Esses dados são disseminados pela rede e então processados e analisados por algoritmos, podendo ser algoritmos de inteligência artificial [Habibzadeh et al. 2018], permitindo que aplicações tomem decisões mais informadas e eficazes em tempo real, através da utilização destas tecnologias.

As aplicações de Internet das Coisas (IoT) são um dos pilares para cidades inteligentes, uma vez que elas são caracterizadas por [Gomes et al. 2018] suportar demandas em larga escala e conseguir difundir informação a diversos nós móveis a partir de um protocolo. Por se tratar de um ambiente variado as aplicações nas cidades necessitam de demandas diferentes como um tempo de resposta especifico, a garantia de entrega de algum dado ou o uso de uma quantidade especifica de memória, entrando assim o conceito da qualidade de serviço (QoS), [Gomes et al. 2018] permitindo que dados com especificações relevantes a aplicação sejam disseminados pela a rede, aumentando assim a eficiência dos resultados e auxiliando na analise dos dados.

O cenário de Cidade Inteligente desse trabalho utilizará o paradigma *Publish/Subscribe* por meio do *middleware* Data Distribution Service (DDS) [Corsaro 2014] , especificado pela Object Management Group (OMG)[1] e trabalha em larga escala e atendendo a diversos tipos de requisitos de Qualidade de Serviço de acordo com a demanda. O *middleware* para gerenciamento de nós móveis, muito utilizados no contexto de Cidades Inteligentes e utilizando de base a tecnologia DDS, o projeto do ContextNet [Endler and e Silva 2018],desenvolvido no Brasil possui a característica de permitir o gerenciamento de mobilidade e a descoberta espontânea de dispositivos, o que é uma base importante para o funcionamento das cidades inteligentes.

O objetivo desta iniciação cientifica é desenvolver um *Gateway* para sistemas de proximidade baseado no DDS e utiliza-lo para complementar a ideia de Campus inteligente na Universidade, utilizando o modelo de disseminação de dados do DDS e do ContextNet. Esse texto está organizado da seguinte forma: a seção 2 apresenta atividades extras realizadas pelo iniciado, a seção 3 apresenta o que foi desenvolvido e entendido a partir dos estudos e a seção 4 apresenta o cenário de teste, os requisitos, a arquitetura do modelo de comunicação desenvolvido e a implementação do código.

2. Metodologia

Nesta seção, descrevemos a abordagem metodológica adotada para a condução deste trabalho. Abrangendo estudos em Redes de Computadores e programação para redes, nossos esforços foram direcionados para compreender os principais conceitos dessas áreas, incluindo protocolos de comunicação, fluxo de dados e operação de redes. As aulas ministradas pelo coordenador da pesquisa forneceram um alicerce sólido nesses tópicos, ao mesmo tempo em que consultamos

[1] `www.omg.org`

materiais de referência para aprofundar nossa compreensão.

Um enfoque particular foi dado aos conceitos de *middleware*, escalabilidade e primitivas de comunicação, como *Publish–Subscribe*. O *middleware*, um componente-chave na integração e coordenação de sistemas de software, foi explorado em relação às suas funções e benefícios. A escalabilidade, crucial para a adaptação eficiente do software em diferentes contextos, também foi estudada.

A partir do estudo dos protocolos de comunicação em rede, nos aprofundamos no *Publish–Subscribe* [Coulouris et al. 2013], um protocolo que se baseia na interação entre publicadores e inscritos. Essa interação é fundamental em diversas aplicações e forma a base do protocolo DDS discutido na seção 3.1.

Em seguida, direcionamos nosso foco para o Data Distribution Service (DDS) e o ContextNet. Através de leituras de artigos e materiais relacionados, bem como experiências práticas com o ambiente DDS, incluindo a instalação do CycloneDDS, como detalhado nas seções 3.1 e 4, adquirimos insights sobre seu funcionamento. Para uma melhor compreensão e implementação prática do DDS, exploramos o conceito de Qualidade de Serviço (QoS) e suas aplicações.

Inicialmente, consideramos utilizar o software OpenSplice. No entanto, devido à nossa familiaridade prévia com a linguagem Python e à comunidade ativa em torno do CycloneDDS, bem como suas atualizações mais recentes, optamos por essa mudança. Embora o OpenSplice seja a base do ContextNet, a transição para o CycloneDDS se mostrou viável, respaldada pela praticidade e documentação disponíveis.

Com a escolha do CycloneDDS, nos dedicamos a estudar a biblioteca e a documentação no GitHub[2]. Isso nos permitiu compreender as chamadas de métodos e a criação dos componentes essenciais para habilitar a comunicação por meio do DDS.

Após estabelecermos nosso conhecimento sobre o software, delineamos o cenário de testes e definimos os requisitos funcionais e não funcionais. Prosseguindo, construímos a arquitetura de comunicação, como detalhado na seção 4, e desenvolvemos testes de software para avaliar sua adequação aos objetivos. Concluída a programação, realizamos um teste de campo para avaliar a integridade e a funcionalidade do código em relação ao problema inicial.

3. Resultados e discussão

Nesta seção será tratado sobre os assuntos vistos e sobre o conhecimento adquirido sobre as áreas estudadas, e sobre a literatura que foi trabalhada. A seção está organizada de forma que na seção 3.4 tem os principais pontos de cidades inteligentes e seu funcionamento, na seção 3.1 é falado sobre o *Middleware* DDS, sua origem, arquitetura e funcionamento, na seção 3.3 tem a descrição do *Middleware* Contextnet, a arquitetura e os usos dele e tem a seção 3.5 que trata da ideia de

[2]`https://github.com/eclipse-cyclonedds`

Campus Inteligentes e como pode-se utilizar dos outros tópicos nesse contexto.

3.1. Data Distribution Service (DDS)

O *Middleware* DDS [Corsaro 2014] é um padrão de comunicação para sistemas distribuídos em tempo real por um protocolo *Publish–Subscribe*, de forma onde o sistema trabalha em larga escala que é comumente em aplicações criticas. A rede DDS se baseia em *Writers* que enviam informações e publicam em um tópico especifico e *Readers* que estão conectados ao tópico e recebem o que foi publicado, estes tópicos são parte fundamental do funcionamento do DDS sendo definido por um tipo, um nome e as especificações de qualidades de serviço (QoS) a qual está associado.

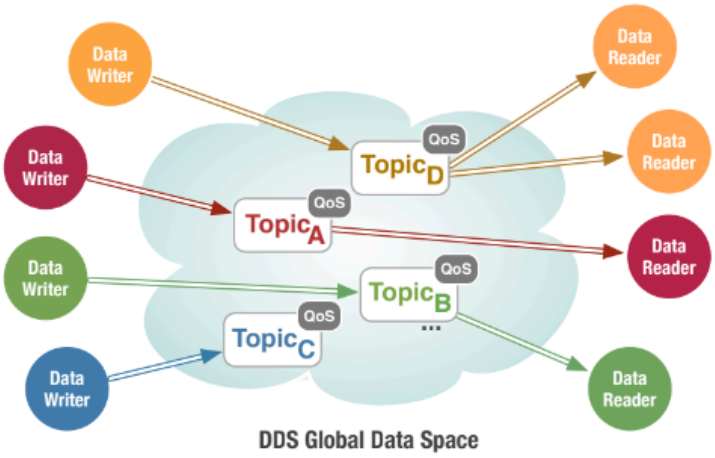

Figure 1. Relação Readers e Writers do DDS

[Corsaro 2014]

Para exemplificar o funcionamento da relação disseminação de dados na rede DDS, para que um *Reader* receba a mensagem do *Writer* ambos devem estar ligados ao mesmo tópico, então a mensagem enviada pelo *Writer* será enviada ao tópico, então o *Middleware* DDS irá comparar os metadados dessa mensagem e comparar as especificações de QoS do *Reader* e vai garantir que o *Reader* receberá os dados produzidos pelo *Writer* se ele atingir os requisitos QoS definidos, essa relação pode ser vista na Figura 1. No exemplo de um *Reader* que deseja receber informações sobre tópico e essa informação necessariamente deve ter sido concedida no máximo de dois segundos, então o DDS irá verificar se as mensagens pertencentes a esse tópico foram enviadas a mais de dois segundos, caso não tenham sido os dados serão enviados ao *Reader* com essas especificações de QoS.

O DDS possui uma arquitetura baseada em visões [Corsaro 2014] onde certos usuários tem acesso através de *Gateways* a certas partes da rede, e o sistema esconde destes usuários a forma com que os dados são transmitidos e qual o protocolo de transmissão está sendo seguido.

Dessa forma o usuário não precisa ter conhecimento com o que está acontecendo na rede e a rede lida em receber e enviar as informações para os usuários baseado na necessidade dele, filtrando, consultando, gerenciando o tempo e cuidando das falhas através da Interface (API), de acordo com a estrutura ilustrada na figura 2.

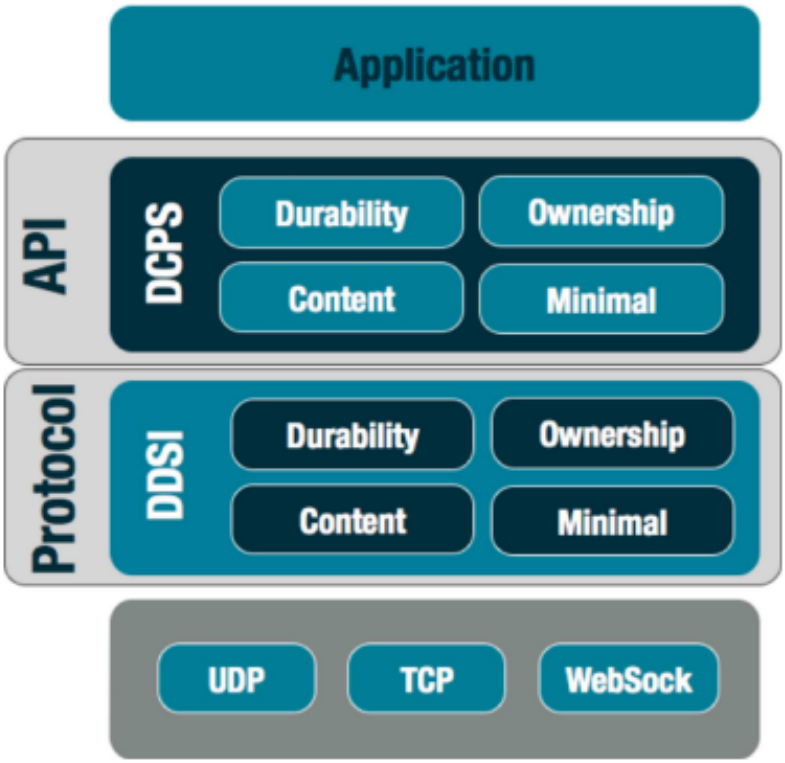

Figure 2. Arquitetura DDS

[Corsaro 2014]

Uma interessante parte do DDS são as diversas possibilidades de se QoS que ele possui, podendo ser programado de diferentes formas para que se adéque aos requisitos não funcionais desejados. Esses recursos englobam [Corsaro 2014]: disponibilidade, confiabilidade, ...

3.1.1. Disponibilidade

A *disponibilidade* determina se um dado está disponível para ser acessado, caso não esteja o DDS verifica se o tópico ou o *Writer* estão fora de funcionamento ou indisponíveis.

3.1.2. Confiabilidade

A *confiabilidade* determina se os dados publicados por um *DataWriter* serão entregues de forma confiável para *DataReaders* correspondentes no DDS.

A *Temporização* está relacionado ao tempo da entrega e postagem dos dados no tópico, possuindo alguns termos importantes como:

Deadline : Em um *DataWriter*, esta política de QoS determina o período máximo em que a aplicação espera chamar a função de escrita no *DataWriter*, publicando assim no DDS. Em um *DataReader,* esta política de QoS determina o período máximo em que a aplicação espera receber novos dados para o tópico.

Latencybudget : Esta política de QoS pode ser usada por uma implementação de DDS para alterar a forma como ela processa e envia dados que possuem requisitos de baixa latência.

Transport priority : configura para que dados específicos mais importantes cheguem antes ao seu destinatário.

3.1.3. Recursos

Os *recursos* visam melhorar o uso da rede por usuários com baixa capacidade de processamento ou de memória. Eles são utilizados para limitar a quantidade de memória do sistema que o DDS pode alocar.

3.1.4. Configuração

A *configuração* gera bits de identificação para os participantes da rede, como *Readers*, *Writers* e tópicos, combinando-os para estabelecer a relação.

3.1.5. Durabilidade (*Durability*)

A política de *durabilidade* controla o armazenamento de dados publicados pelos *DataWriters*. Isso faz com que os dados sejam armazenados no histórico do *Writer* ou em um buffer dentro da rede DDS. Assim, quando um novo *reader* entra, ele pode receber as mensagens do histórico daquele tópico.

3.1.6. Histórico (*History*)

A política de *histórico* define o número de dados que o DDS armazenará localmente para *DataWriters* e *DataReaders*, ou as mensagens de um tópico. Geralmente, ela tenta armazenar as mais recentes e descartar as antigas, se necessário.

3.1.7. *Lifespan*

A política de *vida útil* tem o propósito de evitar a entrega de dados obsoletos para a aplicação. Isso é especificado pelo tempo em que os dados escritos por um *DataWriter* são considerados válidos. Uma vez que a mensagem expira, os dados são removidos do cache do *DataWriter* e do cache do *DataReader*.

Dentre estes ha tambem diversos outros, isso mostra o quão escalável e tolerante a falhas o DDS é, o que o torna ideal para aplicações em tempo real que exigem uma alta disponibilidade e confiabilidade [Bertaux et al. 2014]. O uso do DDS como uma das tecnologias principais na para sistemas de Internet das Coisas (IoT) é fundamental, uma vez que a versatilidade e a eficiência da distribuição de dados em tempo real permite a criação de sistemas de maneira mais eficiente.

3.2. CycloneDDS

O CycloneDDS é uma tecnologia de compartilhamento de dados Open-Source, baseado no DDS da OMG, de alta performasse. O CycloneDDS traz do DDS a transmissão de dados em tempo real e a tolerância a falhas, possuindo implementações QoS e a integridade e consistência do DDS.

A implementação utilizou o CycloneDDS ao invés do openSplice, que é o utilizado pelo ContexNet, por ser uma versão com uma comunidade mais ativa, também ser Open-Source e possuir uma biblioteca em python3, permitindo um desenvolvimento mais ágil na escrita do código.

A biblioteca em python3 possui métodos de implementação para as politicas de QoS, além de possuir métodos simples de criação de componentes do DDS, como Readers, Writes, Tópicos e Entidades. Além disso, é importante mencionar que a biblioteca em Python3 oferece uma página no GitHub[3] que se destaca por sua abordagem didática e clara, tornando-a altamente acessível para o aprendizado e utilização de suas funcionalidades.

3.3. ContextNet

O Contextnet [Gomes et al. 2017, David et al. 2013] é um projeto que visa fornecer serviços para aplicações em larga escala e entidades móveis podendo ser integrado na infraestrutura de Internet das Coisa (IoT), utilizando o DDS como base para a criação de uma rede DDS escalável as demandas sendo assim o SDDL. Como o DDS é comumente utilizado em locais estáticos, o Contextnet vem com a ideia de utilizá-lo em nós móveis, porém para tal uso é necessário utilizar alguns recursos extras para atender os requisitos funcionais e não funcionais.

O DDS oferecia a melhor performance, controlando a latência e sendo eficiente em gerenciar os recursos de *network*, sendo essencial para aplicações de tempo real, porém [David et al. 2013] ele não era eficiente nos nós móveis por conta do IP Multicast, o que limitava o uso de itens móveis e nem em *networks* de larga escala, que era o cenário predominante

[3] `https://github.com/eclipse-cyclonedds`

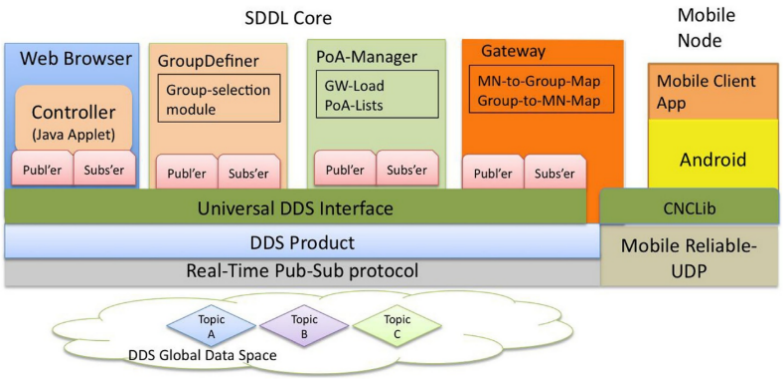

Figure 3. Arquitetura SDDL

[David et al. 2013]

para o uso de celulares. Com essa insuficiência do DDS e a necessidade de escalabilidade, simplicidade e alta performance a equipe do projeto ContextNet projetou o *Middleware Scalable Data Distribution Layer* (SDDL), utilizando protocolos UDP adaptado para maior velocidade.

O SDDL [David et al. 2013] é um *Middleware* que conecta o Domínio DDS estático aos nós móveis, trabalhando com um conexão móvel a base de IP's. Possuindo alguns nós estáticos no DDS para gerir informação e processar dos dados e outros nós que servem como *Gateway* para comunicar-se com os nós moveis e os *Controllers* que são nós que controlam a posição dos nós móveis (ou qualquer outra informação do contexto deles) e criam grupos de nós e enviam mensagens para os nós móveis tanto no formato *unicast* (enviando a um destino) quanto no *broadcast* (enviando a todos os receptores).

A comunicação em tempo real via *pub/sub* utiliza os protocolos UDP e TCP, no sistema do SDDL porem, utiliza-se o protocolo MR-UDP [David et al. 2013] entre os nós moveis e o os nós estáticos/servidores DDS. O protocolo adapta o protocolo TCP ao UDP de forma que evite problemas com as perdas de conexões (*handover*) reduzindo de pacotes de verificação de conexão, problemas de *firewall* estabelecendo pequeno número de pacotes de manutenção de conexão para travessia de Firewall/NAT, problemas de mudança de *IP* através de uma transparência quanto a troca de IP e problemas de interface.

Os nós móveis podem ser qualquer ser ou objeto que possua um aparelho que gere, receba e envie dados, possuindo uma interface sem fio que possa rodar um protocolo IP podendo ter como usuários pessoas, veículos e robôs autônomos. Com sensores conectados aos nós é possível gerar dados como a localização, velocidade ou temperatura de algo, buscando ter uma o compartilhamento desses dados de maneira muito rápida, com o menor *delay* possível. Os tipos de nós utilizados variam, os principais nós do Contextnet [David et al. 2013, Gomes et al. 2017] são:

- Gateways: que ligam um nó estático do Dominio DDS a um nó móvel recebendo e enviando mensagens.

- PoA-Manager:Distribuir os pontos de ligação (POA) e mudar o POA a qual o nó móvel está ligado quando estiver perdendo o sinal, impedindo assim o Handover (processo de perca de sinal ao trocar de ponto de conexão).
- Group Definers: Cria um grupo de nós moveis baseado em alguma semelhança entre eles a partir do ContextUpdate-CxtU (localidade ou ID) e ligam a um Gateway que envia mensagens para todo o grupo quando necessário.

Os Group Definers seguem a ideia da Interface DDS universal [David et al. 2013] e aplicações gerais dos tópicos, os tópicos do SDDL agem juntamente aos tópicos do DDS criados a partir da interface DDS permitindo o reuso e o trabalho em conjunto (interoperabilidade). E com a arquitetura do SDDL, representada na 3, os nós são separados por visões onde cada um possui acesso a uma parte da rede, o ClientLib por exemplo é utilizado para as aplicações mobile com clientes, e não mostra a parte referente a rede e ao protocolo para o usuário, manipulando e gerenciando parte da PoA-List (Lista de pontos de acesso) e lidando com problemas de conexão, funcionando a partir do envio da mensagem no momento em que volta da rede.

3.4. Cidades Inteligentes

Cidades inteligentes [Cruz et al. 2019], também conhecidas como "*smart cities*" em inglês, são aglomerações urbanas que empregam tecnologias de informação e comunicação com o propósito de aprimorar a qualidade de vida dos cidadãos e a eficiência dos serviços públicos. O cerne de uma cidade inteligente reside na exploração de tecnologias para melhorar a experiência dos cidadãos, capturando a dinâmica urbana por meio de dados provenientes de *crowdsensing* e Internet das Coisas (IoT), que por sua vez facilitam processos de tomada de decisão. Plataformas direcionadas às cidades inteligentes devem ser capazes de responder aos desafios de disseminação de informações em tempo real, adaptação dinâmica às condições urbanas, escalabilidade e segurança.

Dentro do contexto das cidades inteligentes, tecnologias diversas [Autili et al. 2020] desempenham papéis cruciais, incluindo sensores, câmeras, redes de comunicação e análise de dados. Através desses recursos, informações sobre o ambiente urbano são coletadas, abrangendo aspectos como tráfego, qualidade do ar, consumo energético, gestão de resíduos, entre outros. Esses dados são disseminados por toda a rede e, então, processados e analisados por algoritmos, que frequentemente incluem algoritmos de inteligência artificial [Habibzadeh et al. 2018]. Essa análise possibilita a tomada de decisões mais informadas e eficazes em tempo real, proporcionando um aproveitamento mais eficiente das tecnologias empregadas.

As aplicações de Internet das Coisas (IoT) desempenham um papel fundamental nas cidades inteligentes, sendo caracterizadas [Gomes et al. 2018] por sua capacidade de atender a demandas em larga escala e disseminar informações entre diversos nós móveis através de protocolos específicos. Dado o ambiente diversificado das cidades, as aplicações requerem diferentes níveis

de resposta temporal, garantia de entrega de dados e alocação de recursos, como memória. Surge, então, o conceito de Qualidade de Serviço (QoS) [Gomes et al. 2018], que permite a disseminação de dados com especificações relevantes para cada aplicação, aumentando a eficiência dos resultados e facilitando a análise dos dados.

Dentro do campo das *Smart Cities*, temos diversas áreas de aplicação que utilizam dos dispositivos IoT como na área de gestão e planejamento urbano [van Zoonen 2016], sendo utilizado por órgãos governamentais e administrativos, e sendo utilizada por empresas e divisões privadas para criação de moradias inteligentes (bairros, prédios e casas). Há também aplicações ligadas ao meio ambiente [Gabrys 2014], que visam gerir o desenvolvimento sustentável e condicionar gastos com recursos ambientais.

No âmbito privado e público utiliza-se o paradigma de cidades inteligentes no campo de recursos sociais, que visam gerir recursos humanos, utiliza-los na área da saúde [Deng et al. 2008], e no parte da escolaridade e acadêmico, como será abordado na seção 3.5.

3.5. Campus Inteligente

A criação de um campus inteligente [Min-Allah and Alrashed 2020] envolve a implementação de tecnologias de cidades inteligentes em um local reduzido, funcionando como uma implementação em pequena escala, porem adaptada para o ambiente universitário para isso, podendo-se utilizar plataformas como o ContextNet e o DDS (Data Distribution Service) em conjunto. A utilização do ContextNet em conjunto com o DDS [Bertaux et al. 2014, David et al. 2013] em um campus inteligente pode permitir que dispositivos IoT distribuídos se comuniquem de forma confiável e eficiente com os Gateways de IoT e outros sistemas de processamento de dados, facilitando a coleta, processamento e análise de dados em tempo real.

O ContextNet pode processar e analisar dados de um sensor transmitido por um Gateway de IoT distribuído pelo DDS, então fornecendo informações relevantes. Com essas informações em mãos, os gestores do campus podem tomar decisões informadas [Min-Allah and Alrashed 2020] sobre como melhorar a eficiência energética, a segurança e o conforto dos usuários do campus dependendo do tipo de sensor. Além disso, a combinação do ContextNet e do DDS pode ser usada em uma variedade de outras aplicações em um campus inteligente, como sistemas de transporte e estacionamento, sistemas de iluminação e sistemas de segurança.

A implementação dessas tecnologias em um campus inteligente pode trazer muitos benefícios, incluindo maior eficiência, redução de custos, melhor qualidade de vida e maior engajamento da comunidade. A utilização do ContextNet e do DDS pode ser uma maneira eficiente e confiável de processar grandes quantidades de dados gerados por dispositivos IoT, permitindo que os usuários tomem decisões informadas com base em informações recebidas em tempo real.

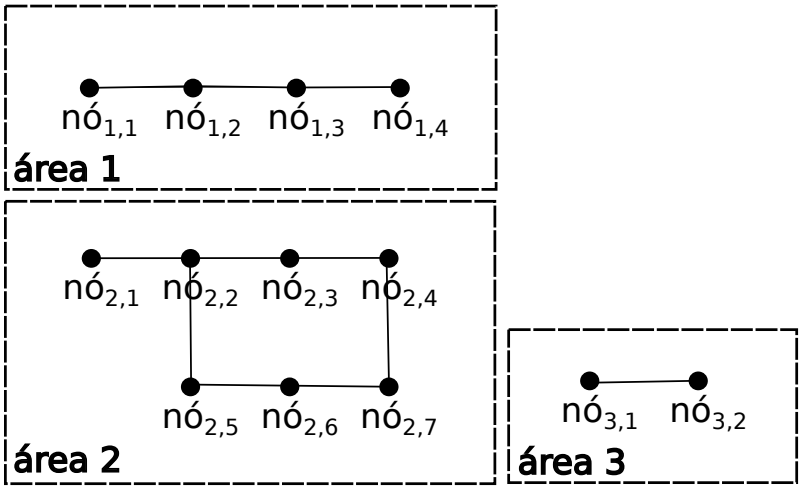

Figure 4. Cenário de Referência

3.6. Comunicação de Redes por Proximidade

Dentro de um cenário inteligente, temos diversos dispositivos que se comunicam através de redes de curto alcance, como conexões via *Bluetooth* ou funcionando via rede *mesh*, e que não possuem acesso direto a internet. Nesse cenário os dispositivos dessa rede, busca se comunicar com estações de uma rede similar que esteja em uma área próxima a eles porem não conseguem realizar a conexão.

Esse cenário é importante pois muito dos dispositivos inteligentes não conseguem funcionar no cenário da internet, uma vez que não possuem endereçamento IP. Eles não possuem IP devido a forma ao qual são desenvolvidos, possuindo endereços.

4. Arquitetura e Implementação

4.1. Cenário Teste

Considere o cenário da figura 4, onde são definidas três áreas que possuem dentro de si máquinas representadas por nós, estes possuem ligações apenas com outros nós dentre de uma mesma área. Porem é de interesse enviar dados para nós presentes em áreas vizinhas e receber dados de maquinas das áreas vizinhas, entrando no cenário da seção 3.6.

As máquinas que compõem o cenário são dispositivos IoT, que não possuem acesso a internet. A comunicação entre eles dentro da área se deve a mecanismos de comunicação por proximidade como a utilização de *Bluetooth* ou redes *mesh*. Porém devido ao curto alcance ou a presença de barreiras físicas, é necessário a adesão de um novo meio de comunicação, para conectar maquinas de uma rede em outra.

4.2. Requisitos

Funcionais:

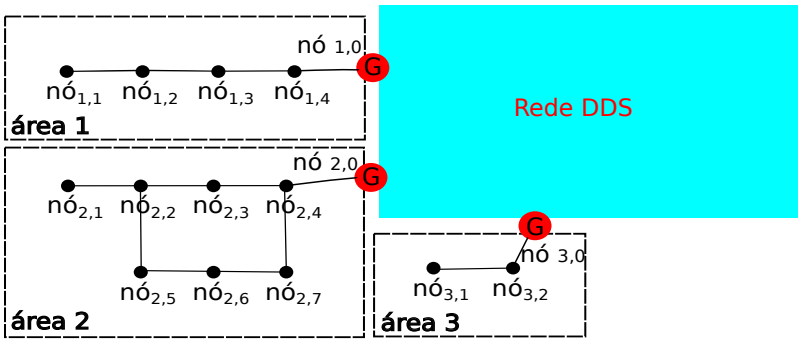

Figure 5. Cenário de Referência com a implementação do DDS

1. Uma área deverá enviar dados a seus vizinhos e receber dados deles.

2. Deve haver uma conexão entre os domínios.

3. O envio de dados deve ocorrer apenas para os destinatários corretos (vizinhos).

4. A comunicação deve ocorrer sem a conexão a internet.

   Não Funcionais:

1. A conexão deve ser segura e não apresentar a localização exata da maquina que está enviando as mensagens.

4.3. Arquitetura

Como pode ser visto na figura 5 para esse cenário, cada área recebeu um ponto de acesso a rede DDS (*Gateway*), que recebe os dados e distribui para as maquinas dentro do domínio. O DDS foi utilizado para conectar os *Gateways* de cada uma das áreas definindo-o como responsável por controlar as mensagens da sub-rede, onde ele terá o papel de receber as mensagens enviadas pelas maquinas da sub-rede e publicá-las em tópico na rede DDS e o papel de receber as mensagens dos vizinhos via DDS e distribui-las para as máquinas da sub-rede.

O *Gateway* criado, fará isso após ser definido como participante da rede DDS e trabalhando tanto como *Writer* quanto um *Reader*, onde possuirá especificações QoS correspondentes a cada um. As funções de QoS trazem uma flexibilidade essencial para esse tipo de comunicação, visto que cada área tem uma tipo de preferência especifica quanto ao envio e recepção de mensagens.

O *Gateway* possui um identificador que define um endereço próprio e lista os endereços dos vizinhos. Esse identificador é utilizado ao enviar uma mensagem para identificar quem está à enviando. E é utilizado ao receber a mensagem para realizar uma comparação entre o endereço de quem a enviou e a lista de vizinhos do *Gateway*, de forma que se for um vizinho listado a

mensagem será recebida e entregue aos elementos na sub-rede, e caso não seja a mensagem é descartada.

- Tópico: Os tópicos serão divididos em conforme os assuntos que são escolhidos para possuir a conversa entre um participante (representado por um nó na imagem 5) de um domínio e as outras entidades das áreas vizinhas (“segurança”, “transito” e “anúncios” como exemplo), o Tópico será escolhido antes da transmissão da mensagem.

- *Writer*: O *Writer* tem a função de publicar os dados (como mensagens por exemplo), enviados por qualquer participante da sub-rede através do *Gateway* que está fazendo conexão, para a rede DDS publicando em um ou mais tópicos selecionados. Ele é definido com politicas QoS especificas para ele conforme as escolhas do usuário.

- *Reader*: O *Reader* fará a coleta de dados dos tópicos em que ele está inscrito, e enviará através do *gateway* para os participantes da sub-rede. Ele é definido com politicas QoS especificas para ele conforme as escolhas do usuário, porém possuindo um *listener* sempre ativo para realizar a leitura em tempo real das mensagens.

- Tipo: O tipo da mensagem foi definido como uma classe onde necessita do endereço, a distancia a qual deve percorrer, um contador e uma *string* contendo a informação que deve ser enviada. Esse tipo é utilizado na comunicação no CycloneDDS, uma vez que o tópico necessita do tipo de mensagem que será transmitido nele para realizar a troca de mensagens entre um *Writer* e um *Reader*.

- Identificador: O identificador serve para definir um endereço para o domínio, do qual a mensagem será enviada, e quais são os vizinhos dele. Funcionando de forma que o endereço possua o id do domínio que ele está presente e dos vizinhos, como exemplo temos o *Gatewa*y 2.0 representado na imagem 5: $endereco = \{id = 2.0, vizinhos = [3.0, 1.0]\}$.

  Ao receber o dado o *Gateway* fará a comparação entre o primeiro id na lista, para ver quem está enviando a mensagem, e os IDs salvos na sua lista. Caso seja compatível ele enviará a mensagem para as maquinas da sub-rede, caso não seja a mensagem é descartada.

  Distancia: A distancia a qual a mensagem será enviada, será pré definida antes do envio da mensagem e funcionara semelhante ao Time to Live, porem ao invés de roteadores será contada em “vizinhos”. O *Gateway* recebe a mensagem do vizinho, caso o tempo de vida dela seja maior que zero, ele republicará a mensagem com seu identificador, reduzindo o tempo de vida dela. Quando o tempo de vida for igual a zero o *Gateway* não fará a republicação.

4.4. Implementação

O *Gateway* dentro do código foi definido como uma classe, que vai possuir um *Reader* e um *Writer* com parâmetros de QoS que podem ser definidos ao gosto do usuário, irá possuir o identificador, contendo o endereço próprio e dos vizinhos definidos em uma lista que possui uma validação utilizando expressões regulares, necessária para a padronização e facilitando implementação da verificação de vizinhança.

Para que a implementação dos *Readers* ocorresse da forma desejada, é necessário que se escolha a forma com a qual as informações serão lidas da rede. Com o CycloneDDS possuímos, diversas funções para realizar essa leitura, como funções baseadas no tempo de espera e na quantidade de leituras.

Por se tratar de um sistema que deve sempre ficar esperando a próxima mensagem dos vizinhos independente de quantas fossem, foi escolhida a função `take_iter()` que funciona até o momento em que o *timeout* acabar, junto com a passagem de parâmetros que estabelece o tempo de *timeout* como infinito, definido no próprio CycloneDDS.

Para definir um modelo base para o identificador de endereços, foram utilizadas expressões regulares. Junto com a criação de duas funções para verificação de integridade dos endereços, de forma a impedir maquinas que não possuem o identificador correto de participarem da rede. Após as leituras do *Reader*, temos a verificação de vizinhança, que coleta o endereço de quem enviou a mensagem, e verifica se o endereço do remetente está presente na lista de endereços.

Por conta do *Gateway* se tratar de um participante da rede que possui tanto as funções *Writer* quanto como *Reader*, sendo o *Reader* uma função que deve permanecer funcionando em tempo integral, é necessário que tenha brechas para o *Writer* seja chamado durante o funcionamento do *Reader*. Utilizando um sistema de *threads*, é possível fazer que funcionem em paralelo.

5. Conclusões

Com o desenvolvimento do *Gateway* utilizando o *middleware* DDS foi possível criar um cenário de conexão entre redes de proximidade que funciona-se sem a necessidade da internet. Por se tratar de um software aberto com diversas possibilidade de uso de QoS, ele é possível de ser implementado em diversas áreas no contexto de Cidades Inteligentes.

6. Agradecimentos

O presente trabalho de iniciação científica foi realizado com apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil (CNPq), dentro do programa PROIP/UFCAT (Edital PROPESQ n° 02/2022).

References

Autili, M., Di Salle, A., Gallo, F., Pompilio, C., and Tivoli, M. (2020). Chorevolution: Service choreography in practice. *Science of Computer Programming*, 197:102498.

Bertaux, L., Hakiri, A., Medjiah, S., Berthou, P., and Abdellatif, S. (2014). A DDS/SDN Based Communication System for Efficient Support of Dynamic Distributed Real-Time Applications. In *2014 IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications*, pages 77 – 84, Toulouse, France. IEEE.

Corsaro, A. (2014). *The Data Distribution Service Tutorial*. PrismTech.

Coulouris, G., Dollimore, J., Kindberg, T., and Blair, G. (2013). *Sistemas Distribuídos-: Conceitos e Projeto*. Bookman Editora.

Cruz, P., Couto, R. S., and Costa, L. H. M. (2019). An algorithm for sink positioning in bus-assisted smart city sensing. *Future Generation Computer Systems*, 93:761–769.

David, L., Vasconcelos, R., Alves, L., André, R., and Endler, M. (2013). A DDS-based middleware for scalable tracking, communication and collaboration of mobile nodes. *Journal of Internet Services and Applications*, 4(1):16.

Deng, Y., Masoud Sadjadi, S., Clarke, P. J., Hristidis, V., Rangaswami, R., and Wang, Y. (2008). Cvm - a communication virtual machine. *Journal of Systems and Software*, 81(10):1640–1662. Selected papers from the 30th Annual International Computer Software and Applications Conference (COMPSAC), Chicago, September 7-21, 2006.

Endler, M. and e Silva, F. S. (2018). Past, Present and Future of the ContextNet IoMT Middleware. In *VLIoT 2018: International Workshop on Very Large Internet of Things*.

Gabrys, J. (2014). Programming environments: Environmentality and citizen sensing in the smart city. *Environment and Planning D: Society and Space*, 32(1):30–48.

Gomes, B. D. T. P., Muniz, L. C. M., Da Silva e Silva, F. J., Dos Santos, D. V., Lopes, R. F., Coutinho, L. R., Carvalho, F. O., and Endler, M. (2017). A middleware with comprehensive quality of context support for the internet of things applications. *Sensors*, 17(12).

Gomes, R., Bouloukakis, G., Costa, F., Georgantas, N., and da Rocha, R. (2018). Qos-aware resource allocation for mobile iot pub/sub systems. In Georgakopoulos, D. and Zhang, L.-J., editors, *Internet of Things – ICIOT 2018*, pages 70–87, Cham. Springer International Publishing.

Habibzadeh, H., Soyata, T., Kantarci, B., Boukerche, A., and Kaptan, C. (2018). Sensing, communication and security planes: A new challenge for a smart city system design. *Computer Networks*, 144:163–200.

Min-Allah, N. and Alrashed, S. (2020). Smart campus—a sketch. *Sustainable Cities and Society*, 59:102231.

van Zoonen, L. (2016). Privacy concerns in smart cities. *Government Information Quarterly*, 33(3):472–480. Open and Smart Governments: Strategies, Tools, and Experiences.

A. Certificados do Ciclo Científico e outras atividades

Devido a problemas com os horários disponíveis para participação de eventos do ciclo cientifico, optamos por realizar o Curso de *Writing in Science*, da Universidade de Stanford disponibilizado pela Plataforma Cousera, com carga horária de 30 horas. O curso possui uma extensa gama de informações de sobre como escrever de forma correta documentos e artigos científicos, com diversos exercícios.

- O primeiro certificado diz respeito ao curso de "Writing in Science"[4], que é um curso online sem créditos com certificado emitido pela Stanford University e ministrado através da Plataforma Coursera, com carga horária de 30 horas.
- O segundo certificado diz respeito ao Minicurso "Estatística aplicada a pesquisa científica", do projeto Ciclo Científico de Pesquisa, ministrada pelo Prof. Msc. Sérgio Luiz Prolo, realizado online nos dias 11 e 12 de maio de 2023, cumprindo a carga horária total de 2 horas.
- O quarto certificado diz respeito à Oficina "WRITING WORKSHOP - LAB REPORT", oferecida no evento 3º CEPEX-UFCAT - CONGRESSO DE ENSINO, PESQUISA E EXTENSÃO, realizado durante o período de 10/11/2022 a 10/11/2022, cumprindo uma carga horária de 3 horas.
- O terceiro certificado diz respeito ao evento 3º CEPEX-UFCAT - CONGRESSO DE ENSINO, PESQUISA E EXTENSÃO (OFICINA - OFICINA PIQUENIQUE DAS MEMÓRIAS: NARRAR, ESCREVER E SE INVENTAR NA FORMAÇÃO) , realizado durante o período de 09/11/2022 a 09/11/2022, cumprindo uma carga horária de 3 horas.

[4] https://www.coursera.org/learn/sciwrite