UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

ANDRÉ DEXHEIMER CARNEIRO

Análise de desempenho de redes baseadas em abordagem Information Centric Networking (ICN) na implementação de redes de comando e controle (C2)

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Engenharia da Computação

Orientador: Prof. Dr. Edison Pignaton de Freitas

Porto Alegre
2021

AGRADECIMENTOS

Primeiramente a meus pais Roberto e Carla que, por meio de muito amor, esforço, trabalho duro e dedicação, fizeram de tudo para me ver feliz e me possibilitar uma vida de privilégios, os quais eu tentei, e sigo tentando, aproveitar ao máximo.

Ao meu irmão Bruno e meus primos Laura e Arthur, que me acompanham desde que me conheço por gente, e servem como exemplos de como agir, de como seguir em frente e daquilo que eu almejo ser algum dia.

Aos meus dindos Carlos e Cláudia, que tem um lugar no meu coração como minha segunda família e de quem a saudade apenas cresce a cada dia.

Além disso, a todos os meus amigos, que sempre me ajudaram a lembrar daquilo que há de melhor na vida.

E a todos que me acompanharam ao longo da graduação e do desenvolvimento desse trabalho, principalmente o Iulisloi Zacarias que me ensinou muito e me ajudou nos momentos de aperto técnico.

RESUMO

Aplicações de comando e controle impõem severos desafios tanto nas condições quando nos requisitos de funcionamento da rede. Seja pelos cenários complexos e, ocasionalmente, imprevisíveis que refletem na topologia da rede ou pelo nível de desempenho exigido neste contexto. Na esperança de alinhar o problema com a abordagem que melhor se adapte, serão conduzidos estudos do desempenho de redes C2 com Information Centric Networking. Este trabalho de graduação tem como objetivo reunir dados concretos a fim de comparar esta prática com outras possíveis e conhecidas, discorrendo sobre as vantagens e desvantagens encontradas.

Palavras-chave: ICN. SDN.

ABSTRACT

Command and control applications impose tough challenges on the conditions and requirements that the network must endure. Both for the complex, and occasionally unforeseeable scenarios which reflect on the topology of the network, and for the level of performance that must be achieved in this context. In hopes to match the problem with the most suitable approach, performance studies will be conducted about C2 networks with Information Centric Networking. This graduation work aims to collect concrete data so as to compare this set of ideas with others used for this same purpose, discussing the advantages and disadvantages found along the way.

Keywords: SDN, ICN.

LISTA DE ABREVIATURAS E SIGLAS

NDN	Named Data Networking
ICN	Information Centric Network
C2	Command And Control
SDN	Software Defined Network
IoBT	Internet of Battle Things
NFD	Named Data Forwarding Daemon
NLSR	Named Data Link State Routing Protocol
QoS	Quality of Service
TC	Traffic Control
AP	Access Point
WLAN	Wireless Local Area Network
RTT	Round Trip Time
FIB	Forwarding Information Base
IP	Internet Protocol
UDP	User Datagram Protocol
TTL	Time To Live
ARP	Address Resolution Protocol
NACK	Negative Acknowledgement
TCP	Transmission Control Protocol
CPU	Central Processing Unit

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A aplicação de tecnologias de coleta de dados e de comunicação para as mais variadas necessidades, bem como refinamentos e otimizações de dispositivos e interfaces, traz consigo o emprego desses meios em situações cada vez mais críticas e de alta performance. Uma consequência disso é o emprego dessas soluções em contextos militares, dentro dos quais podemos destacar IoBT (*Internet of Battle Things*) e aplicações C2 (*Command And Control*).

Redes IoBT (KOTT; ALBERTS, 2017) herdam várias de suas características de redes IoT (*Internet of Things*), que são redes compostas de dispositivos variados que se conectam e trocam dados entre si (TAN; WANG, 2010). Estes aparelhos podem ser especializados na coleta de dados (como sensores e câmeras) ou na análise e processamento de dados (como computadores pessoais e celulares). O ponto que todos tem em comum é a distribuição de informação.

Devido ao alto grau de mobilidade e os variados tipos de interfaces físicas que cada dispositivo pode apresentar, estas redes tipicamente contam com protocolos de comunicação sem fio (como WiFi) para o estabelecimento de conexões. Isto permite à rede a aos seus dispositivos um nível muito maior de versatilidade.

Ao transicionar para o meio de aplicações militares, outras demandas se tornam muito relevantes. Como a tendência dos dispositivos de serem mais inteligentes e de terem um grau mais elevado de independência e poder de tomada de decisões. Isto acompanhado de uma demanda maior por segurança da informação e baixíssima tolerância a atrasos e perdas.

Portanto, são redes compostas de dispositivos diversos e desprovidas de canais de comunicação fixos devido a falta de infraestrutura fixa (como antenas ou enlaces cabeados) e à alta taxa de mobilidade que torna as topologias dinâmicas.

Aplicações de comando e controle (ALBERTS D. S. HAYES, 2006) devem funcionar de forma distribuída entre todos os dispositivos que compõem a rede e manter as garantias de alta dependabilidade, corretude e segurança na transmissão dos dados. E, ao mesmo tempo, são responsáveis por gerenciar todos os dados relacionados ao controle da missão. Sempre coordenando diversos grupos de pessoas, sensores e veículos de forma simultânea.

Estas aplicações são responsáveis pela definição de todos os parâmetros relevantes à missão e à rede que possibilita a comunicação entre todos os seus participantes. A esta

parametrização, se atribui o nome de abordagem C2.

Outros trabalhos (KOTT; ALBERTS, 2017) definem esta abordagem como uma série de etapas (*Sensemaking*, *Execution* e *Situation Monitoring*). Respectivamente, estes passos tratam da coleta, processamento e compartilhamento de informações, de ações reais tomadas no campo de batalha e da avaliação em tempo real dos efeitos dessas operações. Além disso, a rapidez com a qual esses passos são tomados define a agilidade C2 da aplicação (ALBERTS et al., 2013). Estas rápidas alterações de cenário levantam a necessidade de mecanismos de rede que permitam a rápida alteração dos parâmetros necessários para o seu correto funcionamento.

Ao longo da operação, os dispositivos podem se colocar nas mais variadas posições no campo de batalha, as quais podem ser mais ou menos favoráveis para a comunicação sem fio. Um soldado que adentra um cenário urbano com construções muito próximas entre si terá um canal de comunicação com menores taxas de transmissão, devido a obstáculos físicos que obstruem a propagação do sinal e a interferências com dispositivos e redes de telefonia móvel próximos. Outro caso pode ser observado em um *drone* que se afasta dos outros dispositivos para sobrevoar uma área de interesse específica. Ao longo desse percurso ele sofre com a redução da taxa de transmissão e, possivelmente, eventuais desconexões devido à atenuação do sinal imposta pela ampla distância até os outros nós da rede.

Além disso, há a preocupação em validar os dispositivos que fazem parte da rede de forma a evitar a entrada de agentes inimigos. Oponentes poderiam ter a intenção de prejudicar a operação coletando dados sigilosos, propagando informações falsas ou até enviando dados a uma alta taxa de transmissão a fim de obstruir os enlaces da rede. Uma possível solução seria a implementação de um controlador SDN (WICKBOLDT et al., 2015) que gerencia as tabelas de encaminhamento dos nós e determina o acesso à rede com base em mensagens recebidas dos dispositivos (KARMAKAR et al., 2020). Um algoritmo seria capaz de validar o acesso por meio de senhas, métodos de criptografia ou até padrões de movimentação e comportamento dos diferentes equipamentos.

Mesmo com todas essas preocupações em mente, a infraestrutura usada neste ambiente deve possibilitar altas taxas de transmissão e baixos tempos de propagação para permitir a rápida e certeira tomada de decisões. Conteúdos de diferentes tipos e propriedades podem ser requisitados pelos clientes: arquivos de foto e vídeo usados para monitorar pontos estratégicos do campo de batalha ou observar a movimentação de esquadrões inimigos, dados de áudio de soldados ou relatórios com informações sobre a

missão ou status de funcionamento de quaisquer dispositivos na rede.

Em meio aos requisitos apresentados, duas abordagens de arquitetura de rede se sobressaltam como possíveis soluções. O uso de ICN (*Information Centric Networking*) e SDN (*Software Defined Networking*). A primeira tende a favorecer em muito a distribuição de informação devido ao paradigma orientado a dados (YAQUB; H.; KIM, 2017). Esta característica permite que aplicações disseminem informações pela rede sem a necessidade de que isto seja implementado a nível de aplicação, já que é providenciado por roteadores adaptados a este tipo de encaminhamento. Já a segunda, tende a atender aos requisitos de performance, devido à lógica centralizada para a definição das rotas de encaminhamento de pacotes na rede. Ao mesmo tempo em que permite a configuração deste encaminhamento por uma aplicação C2, abrindo caminho para uma rede muito mais adaptável e versátil.

Este trabalho busca validar, por meio de experimentos comparativos, uma solução técnica que melhor se adapte aos requisitos apresentados no cenário descrito. Está solução compreende redes ICN, SDN e com protocolos de comunicação sem fio. Afim de permitir a realização de experimentos, foi desenvolvida uma aplicação que permita a combinação de todos estes três fatores.

Este trabalho está organizado da seguinte forma. O capítulo 2 apresenta em mais detalhes os mecanismos teóricos empregados: ICN, SDN e IoBT e mostra alguns detalhes a respeito de seu funcionamento que podem ser proveitosos para o cenário estudado. A capítulo 3 disserta a respeito de trabalhos relacionados que serviram como fundamentação e fonte inspiração para este estudo. O capítulo 4 mostra qual foi a aplicação desenvolvida para a obtenção dos resultados. O capítulo 5 disserta a respeito de algumas das dificuldades encontradas ao longo do desenvolvimento. O capítulo 6 mostra os resultados obtidos e apresenta uma análise acerca. Por fim, o capítulo 7 conclui o texto.

2 FUNDAMENTAÇÃO TEÓRICA

Como ferramentas para atingir o funcionamento desejado nas redes propostas, serão usados alguns princípios já conhecidos e bem desenvolvidos. O ICN deve ter alto impacto na eficiência do uso de banda devido ao mecanismo de cache que permite o armazenamento temporário de dados nos *hosts*. O SDN melhora o funcionamento do roteamento e encaminhamento de pacotes e permite outros atributos, como os aspectos relacionados à segurança e a comunicação com redes externas. E IoBT é o contexto em que todo o trabalho se encaixa, o qual define várias das características do cenário estudado e foi abordado em (KOTT; SWAMI; WEST, 2016).

2.1 ICN (Information Centric Networking)

O modelo de *Information Centric Networking* (ICN) (ZHANG et al., 2014) é uma alternativa ao paradigma centrado em *hosts*, que é usado em larga escala em toda a infraestrutura da rede global de *internet*. Neste, um canal de comunicação entre dois clientes é determinado por um par de endereços origem e destino, de forma que mensagens são trocadas enviando dados a um destinatário conhecido e explícito. A alternativa proposta pelo ICN a este cenário é uma abordagem orientada à distribuição de conteúdo, em que os dados são enviados sob demanda mediante a recepção de uma mensagem de interesse, conforme é ilustrado na Figura 2.1.

Para requisitar uma informação, um cliente envia uma mensagem de interesse à rede com uma solicitação explícita por um dado específico. Um nodo da rede, ao receber a mensagem, determina se consegue responder com o dado requisitado e, caso positivo, envia a informação ao dispositivo solicitante ou, caso contrário, propaga a mensagem adiante para que os outros nodos da rede possam tomar a mesma decisão.

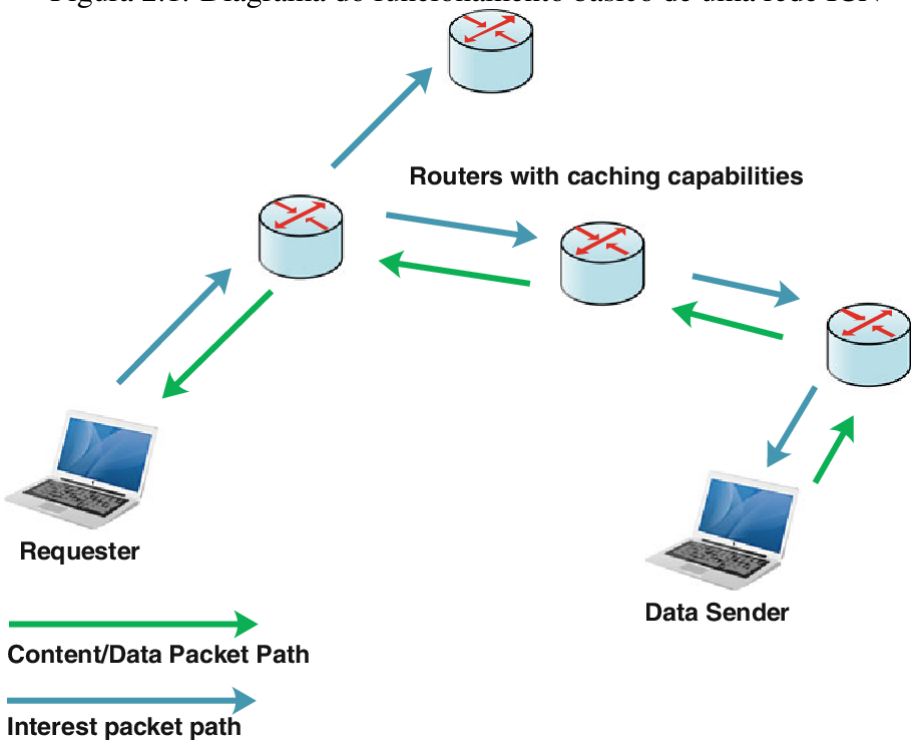
Claramente, o uso de tal abordagem depende da implementação de um protocolo único entre todos os usuários da rede e portanto só é possível em casos restritos onde todos os dispositivos estão executando uma aplicação compatível. O contexto de redes *ad-hoc* é ideal para a adoção deste mecanismo devido ao número de clientes relativamente baixo e restrito e, também, por ser uma rede independente e separada das demais. Inclusive sendo possível o uso de um mecanismo de conversão para que a comunicação com redes externas ainda seja possível.

O maior ganho em desempenho possibilitado por esse paradigma se deve ao me-

canismo de *cache* que pode ser incorporado aos dispositivos. Desta forma, dados que trafegam por um dispositivo podem ser mantidos em uma cache local para que, se requisitados novamente, podem ser enviados por este nodo, assim não sendo necessária a propagação da mensagem de interesse até o dispositivo que de fato produz o dado mencionado. Consequentemente, informações requisitadas por diversos dispositivos tendem a contar com tempos de propagação cada vez menores, ao passo em que são armazenadas nas *caches* locais dos aparelhos.

Neste trabalho serão explorados duas variáveis relacionadas a cache: o dimensionamento da capacidade e o tempo de vida (TTL). A capacidade depende da quantidade de armazenamento disponível do dispositivo, e portanto será menor em dispositivos portáteis e maior em dispositivos mais robustos, como veículos e drones. Já o TTL depende do tipo de dado, e será maior para informações que demoram mais para perder sua utilidade. Por exemplo, notificações de acionamento de sensores devem ter um TTL baixo já que a informação pode se tornar defasada em um curto intervalo de tempo, enquanto vídeos e imagens terão valores maiores. Esta variação no tempo de vida tem uma influência direta nos tempos de propagação e no uso dos enlaces da rede.

Figura 2.1: Diagrama do funcionamento básico de uma rede ICN



Fonte: (YAQUB; H.; KIM, 2017)

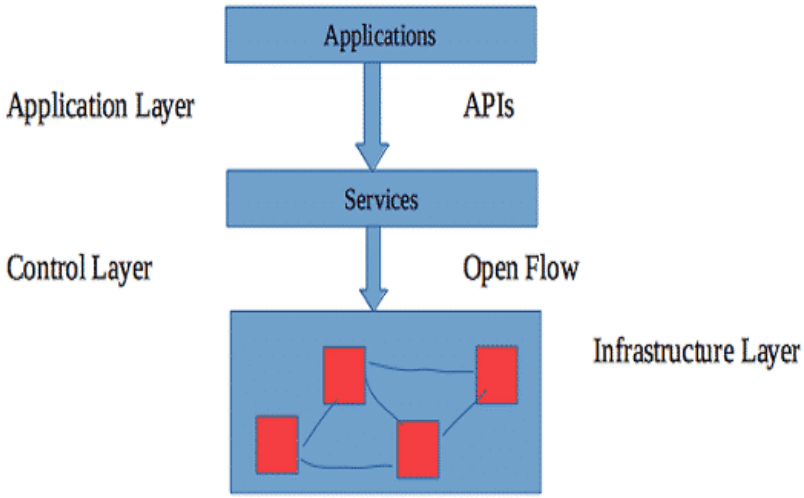
2.2 SDN(Software Defined Networking)

Redes definidas por software contam com um controlador logicamente centralizado que determina o conteúdo das tabelas de encaminhamento dos roteadores da rede (WICKBOLDT et al., 2015). Em teoria, devido à limitada infraestrutura de uma rede ad-hoc sem fio, os próprios nós da rede devem os responsáveis por repassar e encaminhar os pacotes, portanto eles devem exercer o funcionamento dos roteadores configurados pelo controlador.

O controlador tem uma visão da rede como um todo e a capacidade de calcular as tabelas com base em quaisquer parâmetros que possam ser processados por um *software* devido separação de responsabilidades por camada como é mostrado na Figura 2.2, isso o coloca à frente das redes tradicionais. Desta forma é possível tomar decisões mais complexas e levar em consideração informações mais detalhadas da rede, como por exemplo a taxa de transmissão e tempo de propagação de cada enlace, bem como a capacidade de processamento, disponibilidade de cache ou nível de bateria dos diferentes clientes.

Além disso, é possível aplicar um mecanismo de autenticação no controlador para que dispositivos não autorizados não façam parte da rede, o que atende a alguns dos requisitos de segurança do cenário de atuação (KARMAKAR et al., 2020). Outra funcionalidade possível é usá-lo como uma ponte para que a rede *ad-hoc* se comunique com redes externas, as quais podem não usar o ICN.

Figura 2.2: Diagrama lógico de funcionamento do SDN



Fonte: (SAKET, 2018)

2.3 IoBT (Internet of Battle Things)

Dispositivos cada vez melhor preparados para coletar e processar informações são acompanhados por uma demanda de supri-los com a capacidade de transmitir esses dados. Sobretudo lidando com dispositivos heterogêneos, há a necessidade de adaptar a rede às distintas características entre eles como, por exemplo, o nível de bateria e a capacidade de processamento. Um *drone*, cuja principal limitação é a autonomia, pode capturar vídeos e enviar a um veículo que, por estar equipado com um gerador e com um computador de alto desempenho, se responsabiliza por processar as imagens recebidas. Aplicando algoritmos complexos, ele é capaz de extrair informações estratégicas como o posicionamento de tropas ou esconderijos inimigos. Dado o valor estratégico desses dados, também é essencial garantir que eles não cheguem à pessoas, ou dispositivos, errados.

É uma questão de tempo até que todo e qualquer aparelho em um campo de batalha esteja conectado a uma rede de comunicação. Nesse cenário, as preocupações com agilidade e segurança aumentam consideravelmente, já que ambos devem ser providos nos mais diversos aparelhos inteligentes que formam essa rede heterogênea em quaisquer que sejam o ambiente e as condições em que se encontram (KOTT; ALBERTS, 2017).

A baixa dependência em infraestrutura fixa, como grandes antenas ou enlaces cabeados é essencial. Essas estruturas são um ponto fraco na configuração da rede já que, a qualquer momento, podem ser o alvo de algum ataque que teria como consequência a perda ou a debilitação na capacidade de comunicação entre os nós da rede. Dessa forma, a rede deve conseguir operar com a menor infraestrutura possível, contando muitas vez com as próprias interfaces sem fio dos dispositivos participantes.

Mesmo com limitações no tipo de infraestrutura utilizado, estas redes de grande porte devem prover baixos tempos de latência limitando ao máximo a taxa de perda de pacotes. Dados de diferentes tipos, dimensões e características devem circular, sendo tratados de diferentes formas pelo encaminhadores encontrados ao longo da sua rota de forma que, mesmo em cenários de sobrecarga nos enlaces, nenhuma informação crítica seja perdida. Uma das formas de prover este serviço é tratando estes diferentes tipos de informações com níveis de prioridade distintos. Dessa forma, mensagens de controle da missão devem se sobrepor a, por exemplo, pacotes com status de dispositivos ou imagens não críticas à operação.

Devido ao constante movimento e alterações de contexto que estes dispositivos

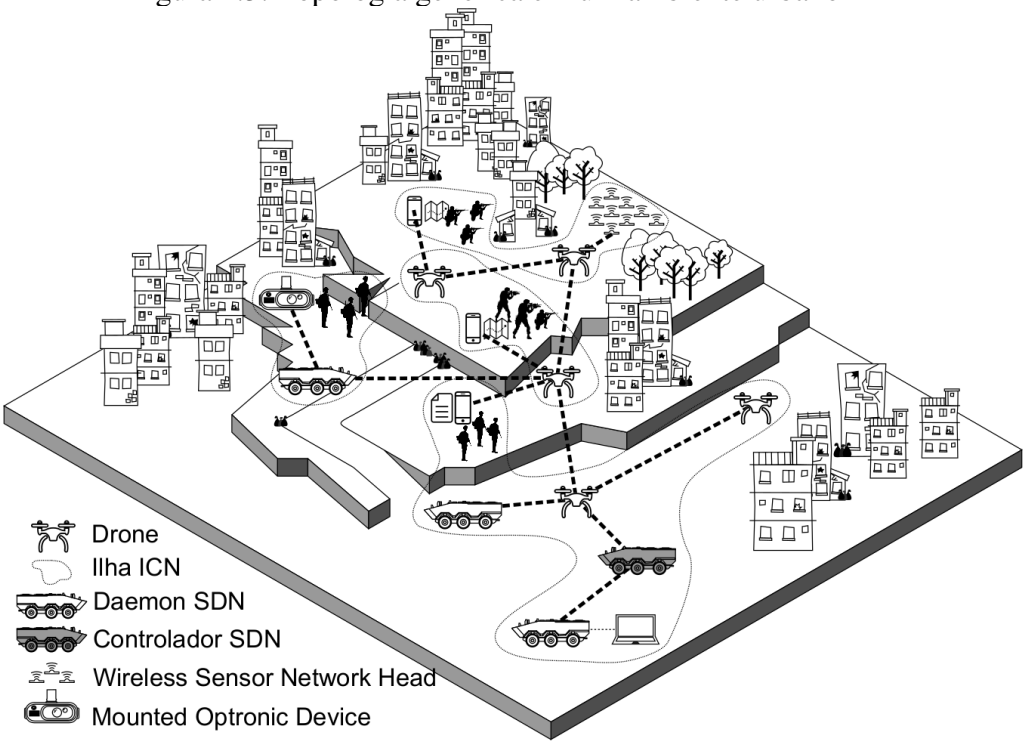
enfrentam, ainda é necessário que estes parâmetros sejam alterados dinamicamente ao longo da missão. Alterando prioridades dos fluxos de dados a partir de parâmetros como os tipos dos aparelhos, as áreas de interesse, os níveis de hierarquia das tropas e quaisquer outros parâmetros da missão é possível adaptar as características dos fluxos de dados à topologia em que se encontra momentaneamente em busca do melhor nível de desempenho possível.

Um exemplo de cenário operacional IoBT empregada em ambiente urbano é apresentado na Figura 2.3 (LEAL et al., 2019). Nessa figura é possível se observar diversos tipos de nós compondo uma rede militar, desde veículos blindados, drones, tropas a pé e sensores estáticos no terreno. Nesse cenário, drones sobrevoam a área adquirindo imagens para dar suporte à tomada de decisão do comandante da operação em uma viatura mais à retaguarda e também para as tropas se deslocando pelo complicado ambiente de uma favela, como representado na figura. Tropas à pé que progridem no terreno são equipadas com dispositivos optrônicos que permitem o recebimento de informações e visualização, tanto do drones quanto dos veículos blindados, além do envio de mensagens de voz e texto reportando o status da ação ao escalão superior. Nós sensores espalhados no terreno permitem a detecção de movimentos suspeitos em áreas de interesse específico e a informação de tais movimentos para as tropas e veículos de retaguarda. Todas essas informações que trafegam nessa rede tem diferentes níveis de prioridade, de acordo com o nó que as solicita ou transmite, e ainda do momento no qual a transmissão ocorrer. Por exemplo, uma imagem de um drone solicitada pelo comandante da operação e também por um soldado em uma posição avançada pode ser mais necessária ao soldado do que ao comandante, por exemplo, no caso do soldado estar na iminência de se encontrar com a ameaça levantada através daquela imagem. Nesse caso, a rede deve priorizar a entrega do dado ao soldado que está na frente de batalha, de modo que ele possa reagir a tempo, em relação ao veículo do comandante que está na retaguarda. Nesse sentido, a rede deve ser capaz de tratar a prioridade de entrega conforme a semântica da operação. Um cenário como o exemplificado na Figura 2.3 é muito rico em termos de aspectos relacionados aos requisitos ligados à aplicação de conceitos de agilidade de comando e controle, e tal cenário necessita de suporte de rede que enderece tais requisitos.

Além disso, a segurança é um elemento central deste tipo de rede e nunca pode ser deixado em segundo plano. Devido ao alto valor estratégico das informações que circulam entre os dispositivos, pode ser do interesse de alguma outra entidade se infiltrar na rede para interceptar dados ou até mesmo propagar informações falsas ou adulteradas afim de

prejudicar o andamento da missão. Esta entidade pode tomar a forma de um criminoso, um dispositivo inimigo ou até um *malware*. Portanto se torna estritamente necessário um mecanismo robusto de segurança capaz de detectar anormalidades no padrão de uso da rede (AZMOODEH; DEHGHANTANHA; CHOO, 2018).

Figura 2.3: Topologia genérica em um ambiente urbano



Fonte: (LEAL et al., 2019)

3 TRABALHOS RELACIONADOS

Neste capítulo serão discutidos, em mais detalhes, alguns dos trabalhos que serviram como base teórica e prática do que foi realizado.

O uso da arquitetura orientada a dados foi abordada em (OH; LAU; GERLA, 2010) como uma potencial alternativa para os problemas que emergem em situações de alta mobilidade e canais de comunicação instáveis e com altas taxas de perda. Dessa forma, melhorando e tornando mais estáveis as taxas de entrega tão necessárias em situações críticas e de emergência.

Outro caso de uso foi explorado em (B.; CHARIGLIONE, 2013), evidenciando ganhos de desempenho em outros tipos de aplicações, como a transmissão de vídeos e no gerenciamento da rede em situações de desastre.

Como sugerido em (AHLGREN et al., 2012), cada vez mais os padrões de uso gerais de rede tem evoluído na direção do desacoplamento entre remetentes e destinatários, favorecendo uma abordagem orientada a conteúdo. Nesse contexto, um nó da rede não precisa ter conhecimento de um endereço específico para requisitar um conteúdo, somente de qual é o dado desejado. Para se adequar melhor a essa semântica, requisições podem são representadas por mensagens de dois tipos: interesse e dado. Um interesse é enviado e propagado pela rede até que uma mensagem contendo o dado associado a esse filtro seja retornada. O que pode acontecer tanto no produtor desse conteúdo ou até mesmo ao longo do caminho, onde o dado pode estar armazenado na cache de outro dispositivo.

Essas e outras características tornam esta abordagem interessante para dispositivos móveis organizados em ilhas na borda da rede. O compartilhamento de dados pode se tornar mais eficiente, tanto na economia de banda quanto na eficiência energética, o que torna esse paradigma especialmente interessante em dispositivos móveis e de baixa potência, como foi estudado em (BACCELLI et al., 2014). Além disso, a obtenção de dados em conexões intermitentes pode ser facilitada com cache dos dispositivos da rede.

Ao mesmo tempo, o SDN, abordado em (NUNES et al., 2014) e (WICKBOLDT et al., 2015) possibilita a distinção entre aplicação, controle, encaminhamento e gerência da rede como um todo. Isto torna possível a tomada de decisões por um controlador logicamente centralizado capaz de determinar o conteúdo das tabelas de encaminhamentos espalhadas pelos *switches* da rede , portanto elevando o nível complexidade que estas decisões podem atingir.

Outro estudo relacionado a este paradigma em (POULARAKIS; IOSIFIDIS; TAS-

SIULAS, 2018) fundamenta a expectativa de benefícios em organizar uma rede ad-hoc de dispositivos móveis com diferentes coalizões por meio de uma rede com controlador SDN.

O trabalho realizado em (SIRACUSANO et al., 2018) apresenta uma possível solução para o *deploy* de redes ICN nos dispositivos IP usados em larga escala na internet por meio de redes definidas por software.

No que compete à simulação de redes ICN, uma ferramenta de simulação disponível empregada em trabalhos relacionados é o CeforeSim, baseado na implementação de redes orientadas a dados do Cefore. Em (HAYAMIZU; MATSUZONO; ASAEDA, 2020) foram realizados experimentos com aplicações de *streaming* de vídeo em tempo real em redes de dispositivos móveis. A partir disso, foram estabelecidas avaliações de desempenho em cenários como o de *handover* de dispositivos, redes de sensores em larga escala e computação distribuída nas condições impostas por ambientes reais.

Outra ferramenta de simulação para redes NDN é o ndnSIM, cujos detalhes acerca do *design* e evolução foram apresentados em (MASTORAKIS; AFANASYEV; ZHANG, 2017). Esse programa *open-source* é baseado no *framework* de simulação de redes NS-3 e na biblioteca NFD (*Named-Data Forwarding Daemon*), que implementa a funcionalidades necessárias para o processamento de requisições e respostas NDN em cada dispositivo da rede.

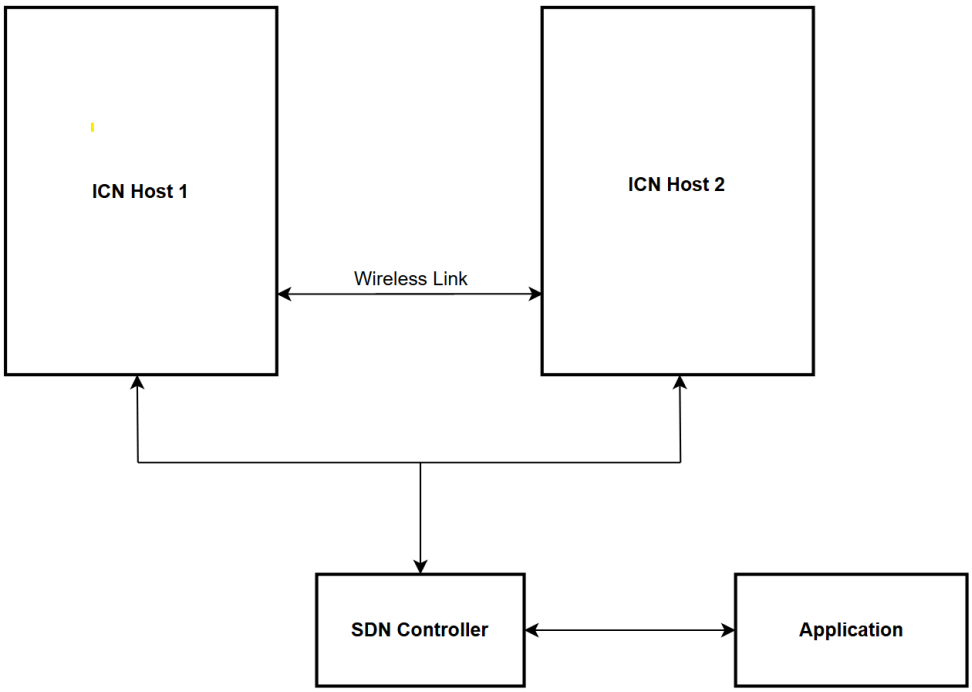
Um trabalho aliando a emulação de redes ICN e protocolo sem fio foi realizado em (MOLL; THEUERMANN; HELLWAGNER, 2018). Nele foi criado um *testbed* para experimentos acerca do impacto da mobilidade de dispositivos e de fenômenos físicos no nível de desempenho de aplicações de *streaming* de vídeo. Além disso, o estudo vai mais afundo ao testar os benefícios de mecanismos de detecção precoce de perda de pacotes na rede no contexto NDN.

4 DESENVOLVIMENTO DE SOLUÇÃO PARA INTEGRAÇÃO SDN-ICN

Para atender aos requisitos apresentados na contextualização do problema, é necessária uma ferramenta para a simulação ou emulação da troca de dados em uma rede orientada a interesses (ICN), com um controlador centralizado SDN capaz de definir diferentes níveis de prioridades para os fluxos de transmissão e com o uso de comunicação sem fio. Esta ferramenta atualmente não existe e, portanto, foi o elemento central de desenvolvimento do trabalho apresentado.

A arquitetura genérica da ferramenta para realização dos experimentos é mostrada na Figura 4.1. Os *hosts* ICN, interligados por interfaces de comunicação sem fio, compõem os dispositivos produtores e consumidores de dados e encaminham as informações baseando-se nos nomes dos conteúdos. O controlador SDN gerencia as tabelas de encaminhamento e define as características de roteamento da rede como um todo, enquanto a aplicação estabelece as regras e definições da abordagem que deve ser utilizada na configuração de rede. Juntos, esses componentes proveem os requisitos necessários para a realização dos experimentos relacionados. Cada um deles poderia ser escolhido, ou implementado, de forma livre, desde que funcionem corretamente quando interligados. Este capítulo descreve os detalhes técnicos que circundam a implementação e apresenta as bibliotecas e ferramentas instanciadas, bem como os motivos pelos quais foram escolhidas.

Figura 4.1: Arquitetura genérica da solução



Fonte: O Autor

O Mininet-WiFi é o elemento central que possibilitou o desenvolvimento dos experimentos. Ele permite a emulação de redes compostas por estações, *access points* enlaces cabeados e sem fio por meio da instanciação de aplicações em *namespaces* distintos do Linux. Esta separação é essencial para que as aplicações de cada dispositivo executem separadamente, como se estivessem em dispositivos físicos distintos. Embora existam outras ferramentas com um propósito similar, o Mininet-WiFi foi escolhido pela sua ubiquidade, o que se traduz em muito conteúdo de amparo disponível na internet. Além disso, os modelos matemáticos que compõem a emulação de protocolos sem fio já foi demonstrada e provada em outros trabalhos, como em (FONTES et al., 2015).

Para obter a arquitetura NDN (Named Data Networking) de forma transparente para a aplicação, foi usada a biblioteca NFD (Named-Data Forwarding Daemon). Ela é responsável pela lógica de encaminhamento de dados baseada em mensagens de interesse e de dados a partir de tabelas mantidas em cada dispositivo. Esta aplicação funciona sobre a camada IP orientada a endereços presente em larga escala na estrutura atual da internet e, portanto, funcionou sem a necessidade de modificações sobre o ambiente construído pelo Mininet.

O protocolo de comunicação sem fio empregado na comunicação entre os dispositivos ICN foi o WiFi. Embora não seja a escolha mais realista considerando o caso de uso de uma rede militar de longo alcance ou de sensores (onde poderia-se usar *WiMAX* ou *6LowPAN*, por exemplo), ela foi a mais rápida e simples em termos de configuração e implementação. Como a variação entre os cenários testados se dá a nível de arquitetura, a escolha do protocolo sem fio não causa variações na relação entre os diferentes paradigmas, apenas nos tempos absolutos de latência.

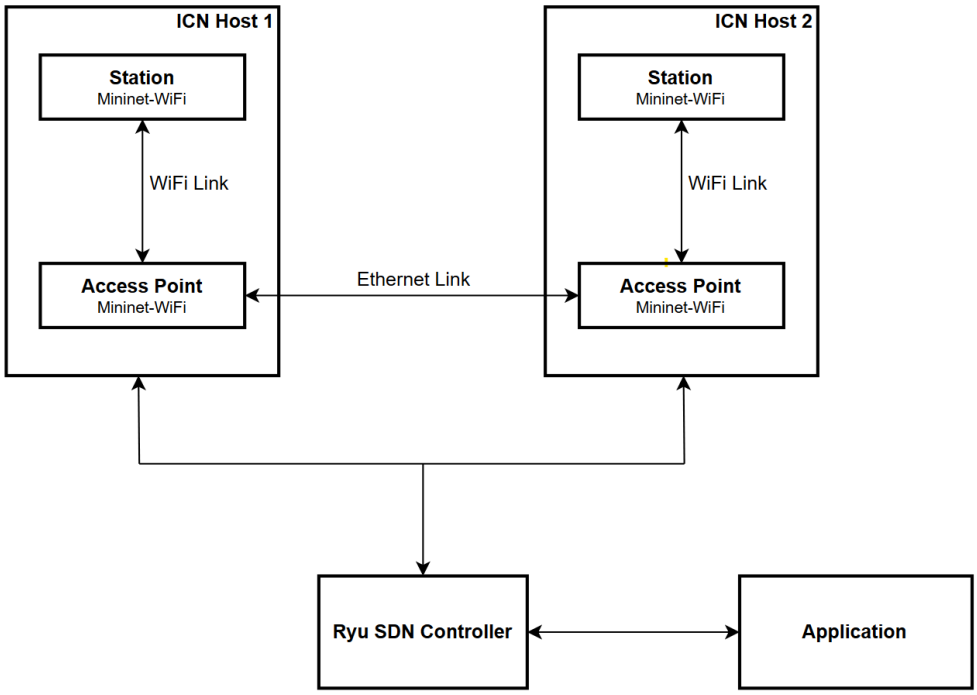
O controlador utilizado foi o *Ryu Controller* devido ao suporte ao protocolo OpenFlow 1.3 e à familiaridade adquirida pelo autor em trabalhos anteriores. Entretanto poderia ser substituído por outro controlador similar. Ele foi acoplado ao ambiente sem a necessidade de muitos ajustes, usando alguns *scripts* diferentes que realizam a construção das tabelas de encaminhamento usando uma visão global ou distribuída da rede. Um dos maiores desafios relacionados ao controlador foi atingir a conectividade completa entre todos os dispositivos da rede, parte da solução foi o uso do algoritmo de *spanning tree* para que a topologia, mesmo que com loops, conseguisse encontrar todas as rotas ponto a ponto entre os *hosts*. Outro aspecto necessário foi a inicialização da conexão de cada *switch* da rede com o controlador no período de inicialização da aplicação do experimento.

O último elemento central necessário para o correto funcionamento da solução

desenvolvida foi a redução de complexidade nos dispositivos, o que foi feito na forma do acoplamento de estação com *access point* conectados por um enlace sem fio. O dispositivo, no formato da estação, executa as aplicações de consumo e produção de dados enquanto o *access point* encaminha os pacotes para o resto da rede, conectada via enlace cabeado. Uma demonstração dessa configuração é mostrada na Figura 4.5 e será abordada em mais detalhes nas seções seguintes.

Um diagrama que representa a arquitetura da solução final desenvolvida pode ser visto na Figura 4.2.

Figura 4.2: Arquitetura final da solução



Fonte: O Autor

4.1 Mininet-WiFi

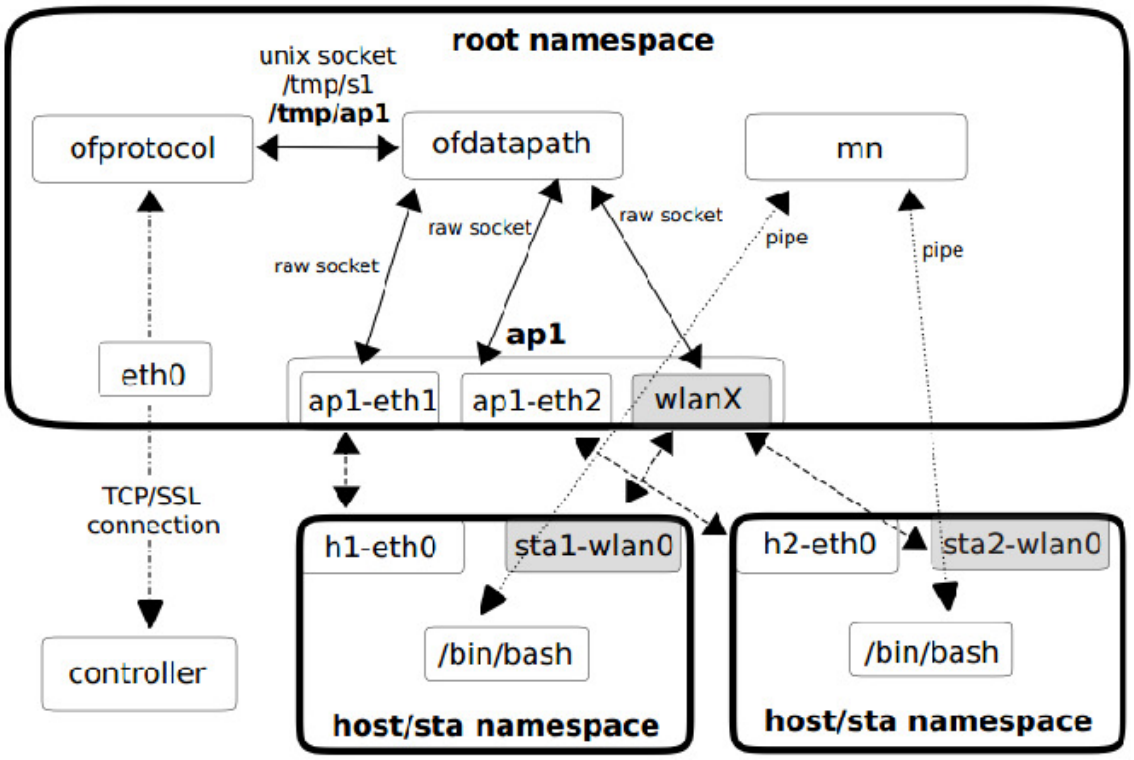
O Mininet-Wifi foi apresentado em 2015 por pesquisadores da Universidade Estadual de Campinas (FONTES et al., 2015) e, desde então, serviu de base para a implementação de diversos trabalhos na área de comunicações em redes sem fio. Ele é o produto da incorporação de diversos elementos relacionados a simulação de protocolos de comunicação *wireless* no Mininet.

Esta ferramenta é responsável pela criação da rede como um todo. A partir de diretivas implementadas na API do MininetWifi, são criados os elementos fundamentais da

rede: estações, *access points* e links. Ele realiza isto por meio de diversos comandos que usam funcionalidades do *kernel* do Linux que permitem que sejam criados *hosts* virtuais em diferentes *namespaces*, o que permite que cada um execute comandos, por meio de um terminal ou pela API usada no script Python. Um breve diagrama do funcionamento dessa ferramenta pode ser visto na Figura 4.3.

Entre os aspectos herdados do Mininet padrão está o que se trata da criação de enlaces cabeados e a simulação desse meio físico com atributos como: taxa de perda, banda, *delay* entre outros. Além da criação dos *host* virtuais, que são posteriormente usados como base para as estações, que são basicamente hosts com a capacidade de usar interfaces sem fio.

Figura 4.3: Diagrama da arquitetura do MininetWifi



Fonte: (FONTES et al., 2015)

O aspecto central que a ferramenta adiciona ao Mininet padrão é a simulação de protocolos de comunicação e aspectos físicos de meios fio que afetam a comunicação de dados. Existem diversas opções que não foram abordadas nesse trabalho por limitações de tempo, como os diferentes protocolos IEEE emulados pela ferramenta, cenários e padrões de mobilidade de hosts no espaço físico e elementos relacionados a interferências que poderiam ser causadas entre os diferentes dispositivos.

Nos experimentos executados com a ferramenta desenvolvida, os quais serão apre-

sentados em capítulos seguintes, foi usado o protocolo *WiFi*. Outros protocolos suportados pelo Mininet-WiFi poderiam ser usados afim de tornar a rede mais realista, como o *6LowPAN*, que é mais frequentemente usado em redes de sensores. Espera-se que a troca do protocolo tenha impacto sobre os tempos absolutos medidos, entretanto não interfere na relação entre os diferentes paradigmas testados, o que é o foco do trabalho.

No que se tange à interface entre as aplicações desenvolvidas e o MininetWifi, ela é bastante simples e intuitiva. A topologia da rede é lida de um arquivo definido em seções que configuram os tipos de elementos: estações, *access points* e enlaces. A partir da sua correta leitura e interpretação, são chamadas as funções que criam as estações com as propriedades informadas, como alcance do sinal, posição dentro da rede e endereços IP e MAC, que tratam de executar todos os comandos *bash* necessárias para que isto se realize. Também criam os *access points*, os quais para este experimento não possuem parâmetros adicionais e os enlaces, cujas configurações serão abordadas na seção que comenta sobre a criação das topologias.

Durante a execução do experimento, é possível executar comandos diretamente nas estações da rede. O que foi utilizado no momento em que são elencadas as transmissões a serem realizadas entre *hosts* que executam programas consumidores e produtores.

No Mininet, não existe uma medição virtual do tempo discorrido. Isto é, o tempo de duração de cada transmissão na ferramenta, e portanto do experimento, se dá em tempo real. Portanto sendo necessário medir o tempo das transmissões com diretivas do sistema operacional para capturar o tempo real no momento da execução. Isto apresenta um desafio ao executar experimentos maiores já que um cenário de falta de recursos de processamento da máquina vai impactar o tempo de execução e, portanto, também o tempo medido.

Dessa forma, uma das funcionalidades que teve de ser adicionada na camada de aplicação para tornar o experimento possível foi a gerência dos processos criados em cada nó da rede. O que foi realizado na forma da limitação do número de processos concomitantes de cada estação, terminando os processos mais antigos de cada uma, já que processos produtores executam indefinidamente enquanto consumidores terminam assim que o dado é consumido por completo.

Essa ferramenta se mostrou capaz de executar experimentos até, pelo menos, 40 estações. Experimentos ainda maiores podem ser executados, entretanto existe um tempo considerável até que a rede com controlador SDN atinja a convergência das tabelas de encaminhamento. Porém deve-se atentar para o consumo de memória que escala junto

com o número de aplicações sendo executadas e, portanto, com o número de nós da rede.

4.2 NFD

A biblioteca NFD executa em todos os nós da rede NDN e implementa as funcionalidades relacionadas à comunicação por nomes. Para isso, cada nó da rede deve ter seu próprio *namespace*, funcionalidade essa que é provida pelo Mininet, e ter seu próprio arquivo de configuração local.

A configuração de cada dispositivo tem parâmetros relacionados a aplicação, como o nível de detalhamento do log, ao funcionamento da cache (neste contexto denominada de *ContentStore*) entre outros, que não foram abordados neste trabalho como certificados e endereços de sub-rede. O log do NFD foi usado para a obtenção dos tempos de transmissão dos dados devido ao seu alto nível detalhe na configuração *DEBUG*, em que registra todas as informações de alterações de configuração na tabela de encaminhamento por nomes, entrada e saída de interesses, pesquisas na cache e entrada e saída de dados.

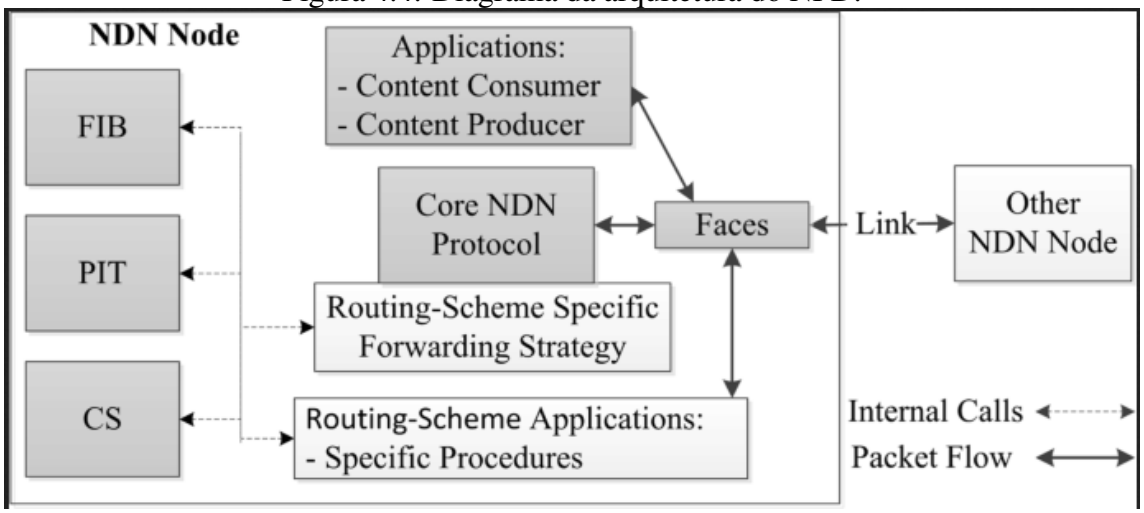
O NFD funciona mantendo conexões ponto a ponto com outros *hosts* da rede, as quais servem para a definição de quais rotas devem ser seguidas para cada nome de interesse requisitado, muito similar à tabela de encaminhamento de um *switch*.

Cada membro da rede mantém localmente uma tabela de encaminhamento por nomes, em que cada prefixo possui um endereço de IP que é o próximo *hop* por onde a solicitação deve ser enviada. Desta forma, o NFD consegue propagar as mensagens de interesse até que estas encontrem o dado que procuram, seja na cache local de algum dispositivo da rede, ou no próprio produtor do dado. Um diagrama representativo da arquitetura do NFD pode ser visto na Figura 4.4

Estas rotas são a base do funcionamento em redes orientadas a dados e devem ser mantidas fiéis à topologia da rede ao longo de toda a execução. Na prática, isto se torna um desafio maior conforme a conectividade dos enlaces oscila e dispositivos entram ou saem da rede. Para combater esse problema pode ser usado o NLSR (*Named-Data Link State Routing Protocol*), que propaga informações de prefixos e endereços ao longo da rede além de implementar medidas de segurança, tão necessárias em cenários reais, por meio de certificados e chaves.

A definição destas rotas foi um ponto de atenção desse trabalho. Inicialmente, eram criadas faces entre cada par de estações da rede, o que é um problema que interfere com o correto funcionamento do experimento. Entretanto, isto não resultou em um

Figura 4.4: Diagrama da arquitetura do NFD.



Fonte: (V. et al., 2019)

problema ao longo da parte inicial do desenvolvimento, que estava sendo usada a ferramenta MiniNDN, a qual faz uso do NLSR para definição dinâmica das rotas. Portanto, as conexões definidas inicialmente durante a inicialização experimento eram redefinidas a tempo de execução fazendo com que não causassem uma diferença perceptível no comportamento da rede.

Mas no momento em que o MiniNDN foi substituído e o NLSR foi removido da ferramenta final, as rotas inicialmente criadas perpetuaram ao longo de todo experimento fazendo com que a única consulta na cache realizada fosse diretamente no nó produtor. Portanto, a presença de cache não refletia em qualquer diferença nos tempos de transmissão.

Para corrigir este problema, a criação de todas as faces que configuram as rotas entre os nós NFD tiveram que ser realizadas durante a inicialização da rede de forma mais inteligente, levando em considerações os caminhos reais na topologia entre cada par de dispositivos. Portanto foi utilizado o algoritmo de *Djisktra*, usando o delay de cada enlace como peso da aresta correspondente, para calcular a rota entre cada dois pontos da rede e definir corretamente o endereço de IP usado para o próximo *hop* de cada prefixo.

Esta solução configura uma possível limitação para a expansão do trabalho, já que estas rotas devem ser redefinidas mediante qualquer alteração na rede. Por exemplo se for removido um enlace, o que pode ocorrer em tempo de execução se estiver sendo testada a interrupção de comunicação de algum dispositivo ou até se um nó troca de posição e, portanto, tem novos vizinhos.

Para contornar esse problema, uma possível solução seria o uso do NLSR, que

implementa a configuração dinâmica das rotas do NFD por meio de comandos como o *advertise*, que propaga um prefixo pela rede, a partir do qual cada nó que receber a mensagem pode definir uma entrada equivalente na sua tabela de encaminhamento.

4.3 Controlador Ryu

O controlador Ryu foi usado por dar suporte ao protocolo OpenFlow 1.3, o que era necessário inicialmente para que fossem criadas as regras de QoS que definem a prioridade da comunicação de cada fluxo de dados. Ele foi usado tanto nos experimentos que simulam redes IP quando nos que simulam redes SDN. A diferença entre esses dois casos se dá no *script* utilizado pelo controlador.

Para redes IP, foi usado o componente *simple_switch_stp*, que tenta montar uma representação da rede em *spanning tree* como se executado em cada *switch* individualmente afim de reproduzir um comportamento logicamente distribuído em que nenhum *switch* possui uma visão global da rede. Dessa forma, é possível reproduzir o comportamento de *switches* que não tem a funcionalidade SDN de forma fiel. Para isso, cada dispositivo mantém uma relação de por quais portas encaminhar os pacotes afim de evitar *loops*, o que pode ser feito fechando as portas conforme é identificado que um fluxo encaminhado pelo *switch* acabou voltando para ele mesmo.

Nas redes SDN, foi usada uma versão modificada do *simple_switch_13* que monta uma *spanning tree* única e centralizada afim de determinar a tabela de encaminhamento de cada *switch* e permitir o correto funcionamento mesmo em redes com *loop*. Este modo consegue atingir rotas melhores para o encaminhamento de pacotes, as quais impactam de forma significativa os tempos médios de transmissão dos dados.

Um fator que teve de ser considerado ao longo dos experimentos executados para a coleta dos resultados apresentados no fim do trabalho é que o controlador demora para aprender sobre a rede e estabelecer as rotas da melhor forma possível. Dentro do escopo deste trabalho, não está inclusa qualquer tentativa de diminuir esse tempo de convergência, portanto os experimentos foram executadas diversas vezes com o mesmo controlador afim de não contabilizar os elevados tempos de transmissão resultantes do processo de treinamento da rede.

O controlador também consegue definir regras mais complexas sobre o comportamento de cada *switch*, dessa forma foi possível estabelecer prioridades entre os fluxos de diferentes tipo de dispositivo. Foram abordadas duas formas de estabelecer estas priori-

dades, foram elas: o uso regras de QoS e a definição de taxas de *ingress*.

A taxa de *ingress* pode ser configurada por meio da diretiva *ingress_policing_rate* no comando *ovs-vsctl set interface*. E, em alto nível, ela define a taxa na qual os pacotes são recebidos no *switch*. A limitação do uso dessa configuração é que ela depende apenas da interface, não tendo qualquer regra ligada aos endereços IP de origem ou destino. Desta forma, a única maneira de usá-la para os efeitos práticos desejados nesse trabalho seria nas interfaces *wireless* dos *access points* os quais sempre se conectam a um único host por *wifi* e, ao resto da rede, por enlaces *ethernet*.

As regras de QoS são mais flexíveis e podem ser definidas com base em qualquer campo que possa ser usados nas regras de encaminhamento. O mecanismo de funcionamento do QoS é por filas, em que cada fila pode ter uma taxa de entrada e de saída e cada switch pode ter regras que encaminham os pacotes que chegam às suas designadas filas. Com esse mecanismo, poderia ser criada uma fila para cada categoria diferente de tráfego, e os pacotes de cada dispositivo seriam encaminhados a sua respectiva fila com base no endereço de IP origem. Portanto este é o candidato ideal para estabelecer as taxas de transmissão dos diferentes níveis de prioridade.

Por mais que este seja uma funcionalidade consolidada do protocolo *OpenFlow* a partir da versão 1.3, ele não funcionou no ambiente utilizado devido a uma incompatibilidade do componente TC (*Traffic Control*), que é o componente que implementa a lógica das filas no Linux.

A solução encontrada foi a criação das filas via TC, onde pode ser definida a taxa de transmissão, e a posterior adição de uma regra via controlador que marca um campo dos pacotes com o identificador da fila à qual ele deve ser encaminhado. Desta forma, quando o pacote chega ao nível to TC, ele é destinado à fila correspondente.

A limitação da banda por tipo de dispositivos, pode ser feita pelo controlador SDN, pode ser alterada ao longo do experimento. Isto é um dos requisitos para o correto funcionamento segundo o critério de *C2 Agility*. Isto porque diferentes momentos da missão podem requerer diferentes configurações afim de distribuir as informações críticas para o nodos que precisam recebê-la, ou a partir do nós que precisam enviá-la, mais rapidamente. No contexto dos experimentos conduzidos para este trabalho, as prioridades foram mantidas fixas.

O diferentes níveis de prioridades podem ser vistos na Tabela 4.1, onde as porcentagens são realizadas em cima da banda dos enlaces mostrada na Tabela 6.1.

Tabela 4.1: Uso de banda permitido por tipo de dispositivo

Tipo	*Uso de banda*
Veículo	100%
Drone	75%
Humano	50%
Sensor	25%

Fonte: O Autor

4.4 Aplicação

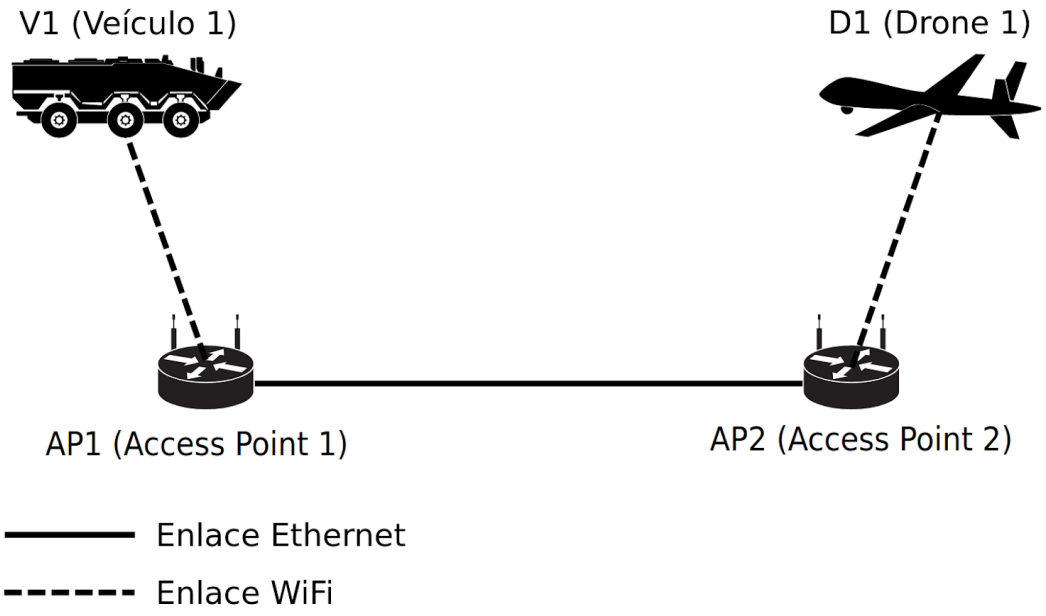
As aplicações desenvolvidas para a execução dos experimentos realizam a criação dos arquivos de topologia, geração dos arquivos contendo a lista de transmissões que serão realizadas pelos dispositivos, inicialização da topologia a partir das diretivas do MininetWifi e NFD além da execução do experimento em si, onde são instanciados os processos responsáveis pela troca de dados entre consumidores e produtores. Outro aspecto desenvolvido é a coleta e processamento dos resultados, o qual será abordado em menos detalhes por se tratar de uma aplicação genérica e trivial.

4.4.1 Criação das topologias

Um aspecto importante da topologia é que cada nó usuário da rede é definido como um par estação transmissora e *access point*. Isso de deve ao fato de que era necessário ter *switches* na rede para o correto funcionamento das regras de encaminhamento definidas pelo controlador SDN. E, afim de incorporar o uso do WiFi na rede, esse *switch* tem a forma de um *access point*. Desta forma, cada AP está conectado via WiFi à sua estação designada e via *Ethernet* ao restante da rede, já que não foi possível fazer a rede funcionar corretamente usando apenas interfaces *wlan*. Um exemplo dessa configuração, conectando dois dispositivos da rede, pode ser observado na Figura 4.5

As topologias são criadas posicionando os dispositivos aleatoriamente em uma área com o tamanho especificado. Para cada nó, são criadas conexões entre ele o os n nós mais próximos. Para as topologias usadas no experimento, o valor de n foi definido como 3. A Figura 4.6 mostra a topologia usada nos experimentos com 20 dispositivos.

Figura 4.5: Visão de dois nós da rede: um veículo e um drone.



Fonte: O Autor

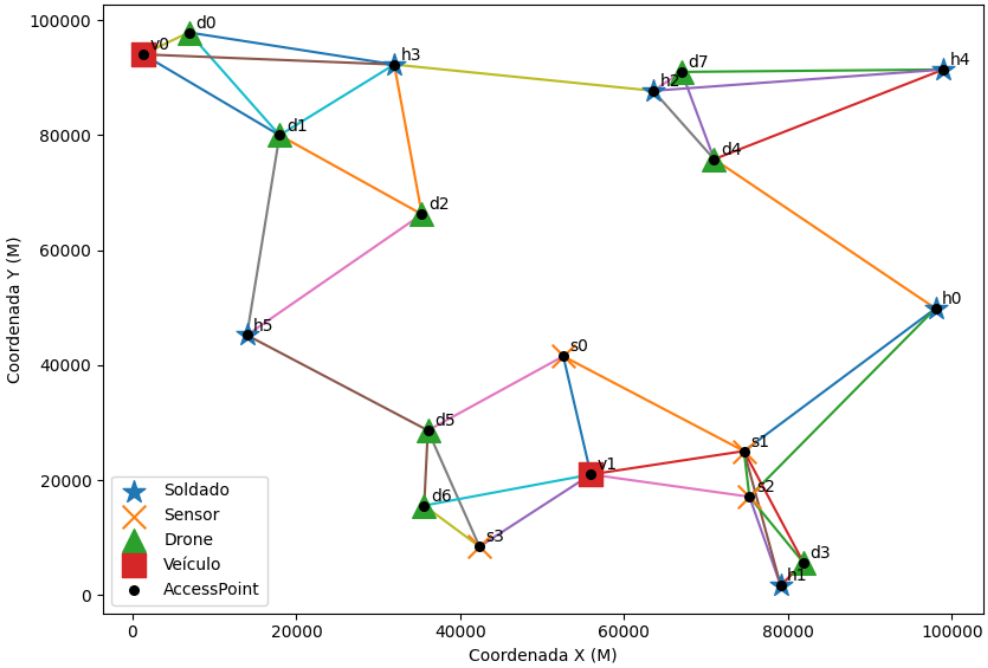
4.4.2 Geração dos fluxos de dados

Os dados gerados compõem uma abstração do que poderia ser transmitido no cenário simulado. Foram determinados 6 diferentes tipos de dados que possuem características distintas no que compete a periodicidade, o tamanho do *payload* e o tempo de vida na cache dos dispositivos a partir do momento da sua produção. Os dados podem ser separados em dois conjuntos: dados de controle e dados operacionais.

Dados de controle representam informações simples que seriam trocados entre todos os dispositivos de forma automática por uma aplicação C2 sendo executada localmente em cada dispositivo. Estes dados são compostos de: informações de status dos dispositivos, como por exemplo nível de bateria, relatórios de posição e movimentação ou breves mensagens de texto ou áudio enviadas pelos nós da rede. Eles têm uma frequência de transmissão maior, *payload* e TTL reduzidos. Isto porque são informações que perdem a utilidade de forma muito mais acelerada, por exemplo: um relatório de posição atrasado em 5 minutos é uma informação muito defasada no contexto de uma operação militar.

Dados operacionais são o oposto. Carregam informações maiores e de maior duração na rede, como por exemplo: um vídeo da movimentação de um pelotão inimigo a partir do qual podem ser identificados os rostos dos soldados ou o tipo de equipamento

Figura 4.6: Visão aérea da topolgia com 20 dispositivos.



Fonte: O Autor

que eles levam consigo ou imagens aéreas de alto valor estratégico para a definição de rotas a serem seguidas no caso de um desastre ambiental.

Para se enquadrar no escopo do trabalho e possibilitar a execução de simulações locais, estes dados tiveram seu tamanho reduzido, as especificações resultantes podem ser vistas na Tabela 4.2.

Tabela 4.2: Abstração dos dados enviados

	Tipo 1	*Tipo 2*	*Tipo 3*	*Tipo 4*	*Tipo 5*	*Tipo 6*
Período	30s	30s	60s	150s	300s	300s
TTL	5s	30s	60s	150s	300s	600s
Payload	1KB	5KB	100KB	1MB	5MB	10MB

Fonte: O Autor

A geração da fila de dados que percorrerão a rede é feita em cima desses valores, considerando que cada dado produzido é, então, consumido por todos os outros dispositivos da rede.

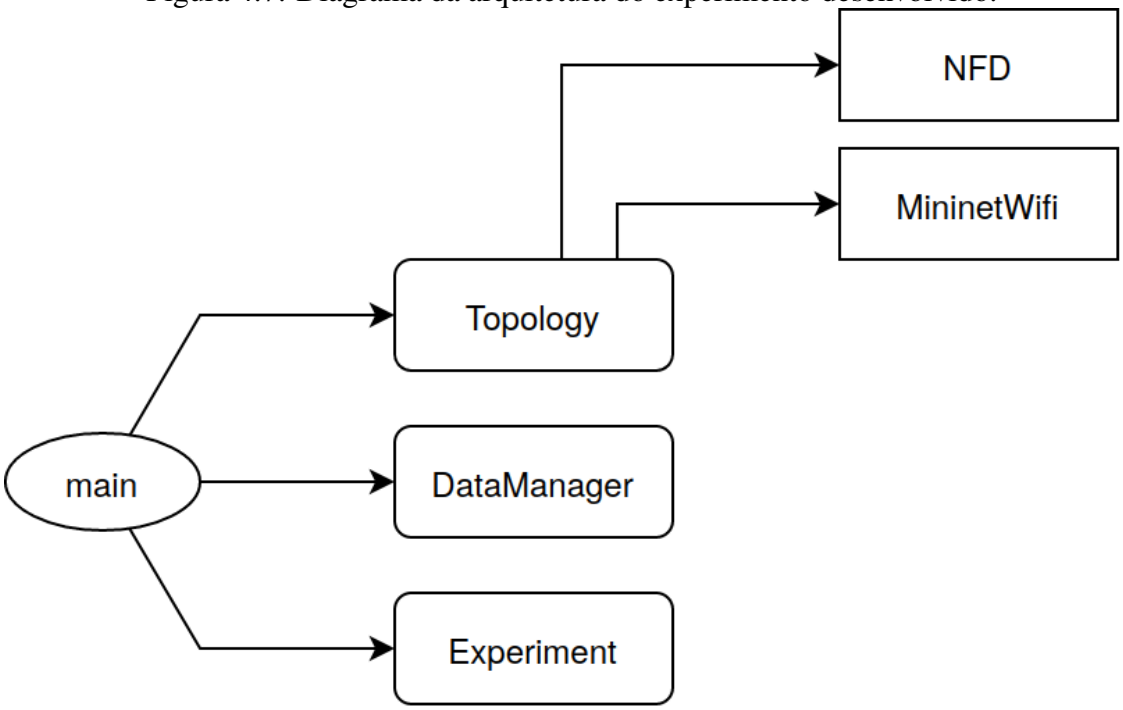
Além disso, o tipo de dado produzido e consumido pode variar de acordo com o tipo do dispositivo. No caso do experimento implementado, sensores enviam apenas dados de controle (tipos 1 a 3) e não consomem dados de outros dispositivos. Uma justifica-

tiva para isso é que sensores não gravam vídeos ou trechos de áudio de alta definição, mas sim pequenos relatórios de acionamento ou outras informações como nível de bateria, posição ou status de operação. Ademais, são dispositivos estacionários cuja funcionalidade não necessita da requisição de dados de outros dispositivos. Entretanto, isto nada interfere na capacidade de servir como propagador de solicitações ou respostas de interesses para outros elementos da rede.

4.4.3 Arquitetura do experimento

O experimento foi dividido em módulos no formato de classes e arquivos *Python* para facilitar o desenvolvimento e o reuso. Uma breve descrição da arquitetura pode ser vista na Figura 4.7, em que são mostrados alguns elementos descritos nas subseções anteriores, sua interação com as bibliotecas do NFD e do MininetWifi, além do experimento em si, que gerencia a transmissão dos dados entre os dispositivos da rede.

Figura 4.7: Diagrama da arquitetura do experimento desenvolvido.



Fonte: O Autor

A classe *Topology* é o núcleo de processamento da topologia e tudo que a compreende. Nela é lido e interpretado o arquivo que define a topologia de um experimento, a partir do qual são realizados os comandos para coerente criação do cenário no MininetWifi. Além disso, por ter a visão global da rede, ela é a responsável pelo cálculo dos

melhores caminhos entre os dispositivos e o preenchimento das tabelas de encaminhamento de todas aplicações NFD.

Esta classe também realiza as configurações relacionadas ao dimensionamento da cache dos dispositivos, realizada pelo arquivo de configuração do NFD, e a execução dos comandos para a criação dos níveis de prioridades descritos na subseção do controlador Ryu.

O desacoplamento dessa classe da lógica do experimento se mostrou necessário pelo fato de que o mesmo experimento deve ser executado diversas vezes em cima da mesma topologia para a coleta de tempos confiáveis. Isto devido ao tempo que o controlador SDN leva para atingir a convergência das rotas configuradas nos *switches*.

Além disso, outros fatores demorados para a execução do experimento são a criação e a finalização da rede do Mininet. Com essa solução, n experimentos podem ser executados com apenas uma criação de topologia. Evidentemente, é necessário realizar o processamento dos *logs* e a remoção de todos os dados em cache entre as iterações para evitar a leitura de dados duplicados ou a sobrevida de dados em cache.

O *DataManager* realiza a criação e a leitura da fila de dados a serem transmitidos ao longo do experimento. Esta lista é criada como foi descrito na subseção anterior e deve ser ordenada pelo *timestamp* de envio de cada dado afim de facilitar a leitura e processamento.

O módulo *Experiment* lida com a execução do experimento em si, ele possui a lista de dispositivos presentes na rede e a lista de dados a serem transmitidos. A partir disso, ele lê a fila sequencialmente e instancia todos os respectivos produtores e consumidores para cada dado. Para as transmissões foram utilizados os comandos *ndnputchunks* e *ndncatchunks* da biblioteca *ndn-tools*. Estes comandos abstraem toda a lógica existente por trás que implementa o correto envio e gerenciando o fluxo de dados de acordo o os *timeouts* e atrasos de pacotes.

Além disso, este módulo faz o gerenciamento de memória do experimento. Ele mantém um *pool* de processos por estação, para que, após algum tempo, os processos dos produtores possam ser terminados. Isto é necessário porque, diferente do consumidor, o produtor não termina sua execução quando o dado é transmitido, portanto sendo necessário terminá-lo para liberar os recursos da máquina.

O componente *main* amarra os outros componentes definindo uma aplicação. O produto final tem alguns parâmetros como o arquivo de topologia, o número de execuções que devem ser realizadas e o tipo da rede, entre IP, IP com SDN e ICN com SDN.

Evidentemente, deve ser executado o controlador adequado com o tipo de rede.

5 DIFICULDADES ENCONTRADAS

A implementação se mostrou mais complexa do que o inicialmente esperado. A adição de funcionalidades foi, por muitas vezes, acompanhada da dificuldade em adaptar o que já havia sido feito para que tudo funcionasse em conjunto. Tendo em vista o viés prático do trabalho, nesta seção serão detalhadas dificuldades técnicas encontradas ao longo do caminho bem como as soluções encontradas para superá-las. Espera-se que essa seção auxilie no desenvolvimento de aplicações similares.

O ponto de partida da implementação foi o MiniNDN. O qual era citado em diversas publicações de experimentos similares e adiciona diversas funções relacionadas ao uso das bibliotecas que compõem o ambiente NDN, como o NFD e NLSR. Esta ferramenta tem algumas limitações no que se relaciona aos requisitos pretendidos. Inicialmente, ela funcionava apenas para redes cabeadas, portanto não sendo possível usar qualquer simulação de comunicações sem fio. Além disso, por mais que baseado no Mininet, a integração com o controlador se mostrou difícil no primeiro momento.

5.1 Transmissão sem fio

Como inicialmente a transmissão sem fio não era possível no MiniNDN, originalmente cogitamos aproximar a simulação cabeada à sem fio por meio da definição de taxas de perda similares às do protocolo *WiMAX* para cada enlace da rede. Esta aproximação esbarrou em uma limitação do Mininet já que ele permite a definição de taxas de perda em valores inteiros entre 0 e 100%, o que já não seria adequado para representar a comunicação sem fio.

Além disso esta abordagem minimalista à comunicação sem fio não teria os efeitos desejados, já que em diversos casos, não observamos perdas mas sim *delays* maiores. O que inclusive foi o caso para os experimentos conduzidos futuramente e apresentados em seções futuras deste trabalho. Com isso, a simulação de meios sem fio foi deixada de lado até que alguma solução melhor se apresentasse.

Em outubro de 2020, houve uma atualização no MiniNDN que adicionava as funcionalidades necessárias para a simulação da comunicação sem fio e, portanto atualizei o código para usar *access points* e estações que deveriam construir a nova topologia de rede. Entretanto, a simulação de cenários com a nova topologia apresentava problemas na interface com o controlador, o qual não inseria as rotas corretamente nos *access points*

causando diversas perdas de pacotes ao longo das execuções.

A não adição dessas rotas causava dois grandes problemas: a conectividade entre *hosts* da rede era impossibilitada pela falta de resposta às requisições ARP entre estações e, nos casos em que as tabelas ARP estavam preenchidas, os pacotes NDN poderiam ser descartados ao longo da rota.

Como o NFD usa o protocolo UDP, essas perdas eram muito visíveis e tinham impacto direto no resultado final, o qual apresentava mais *timeouts* do que transmissões bem sucedidas, assim tornando a métrica de *delay* muito enviesada e aumentando bastante o desvio padrão dos resultados.

Além disso, se dois dispositivos não conseguiam trocar mensagens ARP entre si, a comunicação entre eles é impossível.

Eventualmente, o experimento foi migrado para o MininetWifi, em que os experimentos funcionaram corretamente. Isto se não se deve exclusivamente ao MiniNDN, mas a diversos fatores que foram encaixados no trabalho ao longo do tempo para a correções de outros problemas que acabaram por solucionar os problemas relacionados à falta de conectividade.

5.2 Controlador

O primeiro controlador elencado a ser usado foi o Ryu devido a algumas experiências passadas o utilizando. Entretanto, inicialmente tive problemas ao tentar estabelecer a conexão entre controlador e os *switches* MiniNDN. Logo que a topologia era criada, a conexão com o controlador era fechada de forma abrupta e o processo do controlador terminava.

O que estava acontecendo era que, durante a inicialização da conexão, o controlador envia uma mensagem de *hanshake* para estabelecer a conexão, esta mensagem é a *OFPT_FEATURES_REQUEST*. No caso, os *switches* estavam respondendo a essa mensagem com o conteúdo nulo, portanto o controlador era finalizado.

A solução para este problema foi troca do controlador para o *Pox* especificamente na versão *fangtooth*, o qual conseguia estabelecer a conexão com *switches* sem finalizar.

Adicionalmente, foi necessário inicializar o campo de ID de cada *switch* já que, por motivos desconhecidos, todos eram inicializados no mesmo valor, o que causava ainda mais problemas na comunicação com o controlador.

Uma vez com a comunicação estabilizada entre os *switches* e o controlador, ainda

surgiram outros problemas relacionados. Primeiramente, a topologia não funcionava se tivesse *loops*, isso fazia com que as rotas não pudessem ser corretamente estabelecidas entre os dispositivos e estes não tivessem conectividade entre si. Este problema foi resolvido alterando os parâmetros de inicialização do *Pox* adicionando as flags: *openflow.of_01, openflow.discovery, openflow.spanning_tree, no_flood=truem forwarding.l2_learning*. Desta forma, consegui estabelecer conexões entre os diferentes nós da rede mesmo quando esta apresentava *loops*.

Em seguida, tive os mesmos problemas abordados na seção anterior com relação a *timeouts* de pacotes NDN ao longo do experimento, o que comprometia bastante os resultados finais.

Eventualmente, o controlador foi trocado pelo Ryu, que consta na versão final do trabalho. Ainda tive problemas similares com relação a presença de *loops* na rede, porém isso também pode ser corrigido com alterações no *script* que o encaminhamento na rede. Nesse momento, foi atingida uma situação de relativa estabilidade nas conexões entre os nós, tanto no que se trata do encaminhamento de mensagens ARP quando NDN.

5.3 Prioridades dos fluxos de dados

A ideia inicial para o estabelecimento das prioridades nos fluxos de dados era a configuração via SDN, por algum meio, baseado no endereço IP de origem do pacote. Entretanto isto se provou mais desafiador do que o esperado.

Se fosse feito unicamente pelo encaminhamento preferencial dos pacotes com prioridades por rotas melhores do que os demais, isto seria uma tarefa muito extensa e digna do se próprio trabalho de graduação. Portanto a segunda maneira definida foi pela criação de regras de Qos nos switches da rede. Assim, diferentes filas poderiam ter diferentes vazões e a fila de um pacote poderia ser definida com base no seu endereço IP de origem. Esta abordagem funcionava no Mininet em conexões cabeadas, porém não no meio Wifi.

Antes de descobrir o por que dessa limitação, foi descoberta outra ferramenta que poderia ser empregada para este mesmo fim, embora com uma limitação. A ferramenta no caso é o parâmetro *ingress_policing_rate*, que pode, em alto nível, determinar a taxa de entrada por uma interface de um *switch*. A limitação no uso desta solução é que as regras teriam que ser as mesmas para todos os pacotes que entram por uma interface e, portanto, essa regra teria que ser aplicada nas interfaces *wlan* dos *access points*, as quais lidam exclusivamente com um dispositivo.

Entretanto, eventualmente foi descoberto que a falha no QoS ocorria por uma incompatibilidade entre o TC do Linux e as filas do QoS. Sendo assim, o melhor caminho foi utilizar os comandos do TC para criar as filas com diferentes taxas de transmissão, mais detalhes sobre esse procedimento são descritos na seção detalha a implementação do controlador. O SDN é usado nesse caso para marcar um campo do pacote quando ele chega no *switch*, esse campo é posteriormente consultado pelo TC para colocá-lo em uma fila específica.

5.4 NACKs

Este problema foi encontrado diversas vezes ao longo do desenvolvimento e é sempre indicativo de alguma falha na implementação. No contexto de redes NDN, o NACK ocorre quando há uma falha na requisição de interesse e significa que houve algum erro ao processar o nome na mensagem de interesse. Dentro dessa semântica ele ainda pode assumir pelos menos duas distintas causas: não há rota para o nome ou ele não está definido na tabela FIB local.

O primeiro caso pode acontecer quando a mensagem de interesse está buscando um nome que não existe na rede, o que aconteceu ao longo do desenvolvimento quando os consumidores buscavam um modelo de interesse e os produtores produziam outro. E também, ainda quando o NLSR ainda estava sendo usado, quando a rede havia sido inicializada porém o protocolo não havia tido o tempo suficiente para propagar todos os prefixos de interesse pela rede, neste caso a solução era aguardar alguns segundos antes de inicializar o experimento.

Nos casos em que o nome não está definido na tabela FIB do nó que está processamento a mensagem, a causa era sempre uma falhar ao inicializar as tabelas as rotas NFD. O que deve ser feito sempre no início do experimento.

5.5 Timeouts de requisições NDN

O tempo de *timeout* de uma requisição NDN é definida no momento da sua criação. Quando esse tempo expira a partir do momento em que o interesse é enviado, a aplicação consumidora desiste de esperar pelo dado e registra um *timeout*.

Em situações de uso normal da rede, um evento de *timeout* não é necessariamente

um caso de erro, pode ser que aplicação tenha definido um período curto demais e não condizente com a atual situação de congestionamento da rede.

Ao longo do período de desenvolvimento, este *timeout* foi um problema em duas distintas situações: durante o desenvolvimento de uma aplicação consumidora própria e durante o período em que foi usado o controlador *Pox*.

O caso do controlador *Pox*, já foi abordado na seção que comenta sobre o as dificuldades encontradas com o controlador SDN, e portanto não será abordada em mais detalhes.

O desenvolvimento de uma aplicação consumidora própria foi uma tentativa de simplificar o experimento e torná-lo mais otimizado. Com uma aplicação própria, cada nó da rede poderia ter apenas um processo consumidor, que leria o arquivo de dados a serem consumidos e executaria os comandos necessários, o que diminuiria de forma significativa o *overhead* causado pela criação dos diversos processos consumidores ao longo da execução do experimento.

Porém esse desenvolvimento esbarrou na complexidade necessária para que aplicações NDN funcionam de forma confiável na rede. Como a comunicação é feita sobre o protocolo UDP, não existe nenhuma forma de controle de congestionamento. Portanto se dados forem enviados a uma taxa muito alta, irão sobrecarregar a rede e serão descartados nos *switches*. Entretanto, como a comunicação entre produtores e consumidores é gerenciada pelos processos NFD, há ainda uma camada adicional sobre o UDP, dessa forma uma sobrecarga de pacotes nessa camada faz com que ela pare de funcionar corretamente e pare de responder a quaisquer requisições.

Um mecanismo básico de controle de congestionamento foi implementado na tentativa de corrigir este problema, porém a solução simples adotada não foi o suficiente para que a rede funcionasse da forma correta. Portanto, a solução definitiva foi adotar os comandos *ndnputchunks* e *ndncatchunks* para que realizassem essa comunicação. Estes programas, que fazem parte da biblioteca *ndn-tools*, os quais implementam uma solução inspirada no mecanismo *TCP CUBIC* ao ajustar o tamanho da janela de transmissão dinamicamente com base em marcas de congestionamento nos pacotes e o número de *timeouts*.

5.6 Desempenho

O último problema encontrado ao longo do desenvolvimento foi a performance ao longo da execução dos experimentos. As ações tomadas para resolver estes problemas tinham como objetivo reduzir o tempo de CPU e a quantidade de memória usada pelo experimento.

Foi criado um *pool* de processos produtores por dispositivo, assim liberando recursos ao terminar processos antigos conforme novos eram criados. Para acelerar o ritmo da execução de diversos experimentos em cima da mesma rede, foi modularizado o trecho de código que realiza a criação e a destruição da rede, o qual pode rodar uma vez a cada n execuções da mesma topologia. A última medida tomada foi a redução do tamanho dos arquivos que trafegam pela rede, já que estes arquivos são lidos por cada um dos processos consumidores e produtores que rodam simultaneamente.

6 EXPERIMENTOS E RESULTADOS

Alguns cenários foram elaborados afim de comparar métricas de desempenho de três diferentes abordagens para a estrutura de redes: IP, IP com SDN e ICN com SDN. Todos implementados em cima da mesma arquitetura descrita no capítulo anterior, usando as bibliotecas do *MininetWifi* e do NFD. Os experimentos foram executados entre 20 e 30 vezes em cada configuração e os gráficos apresentam os resultados médios dessas execuções com uma barra de erro demonstrando a maior variação possível.

No que diz respeito aos modelos de rede usados nos experimentos apresentados a seguir, a rede denominada de IP não tem cache e conta com um controlador logicamente descentralizado, que define o conteúdo das tabelas de roteamento dos *switches* de forma individual e, portanto, sem visão global da rede. A rede SDN também não apresenta cache, mas tem um controlador logicamente centralizado que determina as rotas com base no resultado do algoritmo *spanning tree* executado em cima da topologia da rede. A rede denominada ICN+SDN, possui cache nos dispositivos de acordo com o apresentado na Tabela 6.1 e o mesmo controlador da rede SDN.

Com esses experimentos, espera-se validar a abordagem ICN com SDN como a que tem o melhor potencial para a distribuição de informação em topologias pequenas e médias. Os experimentos foram executados em topologias com tamanho entre 10 e 40 nós.

As configurações detalhadas de cada topologia usada nos experimentos são apresentadas na tabela 6.1. Os valores de cache foram determinados a partir de um valor máximo de 8000 dados NFD (o que corresponde a 625MB) e um fator de 50% para *drones* e humanos, 75% para sensores e 100% para veículos.

Tabela 6.1: Configurações das topologias

	10 nós	*20 nós*	*30 nós*	*40 nós*
Sensores	2	4	8	10
Drones	4	8	12	14
Humanos	3	6	9	12
Veículos	1	2	1	4
Delay nos enlaces	2ms			
Banda dos enlaces *Ethernet*	50mbps			
Cache	Sensores: 468.75MB Drones: 312.5MB Humanos: 312.5MB Veículos: 625 MB			

Fonte: O Autor

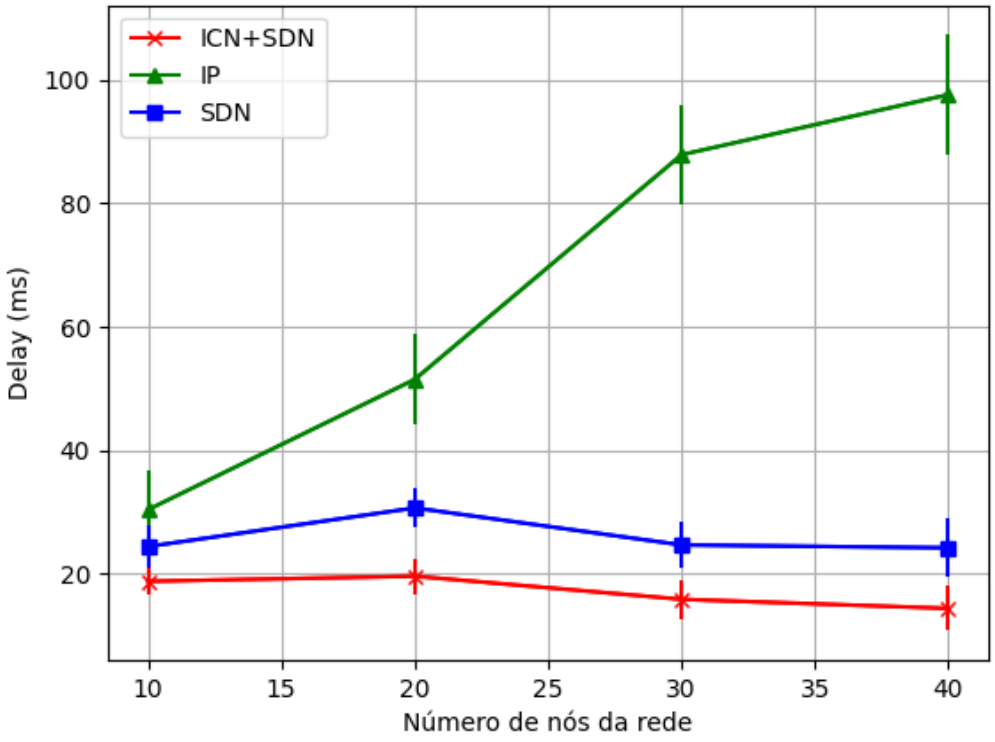
6.1 Delay RTT

Neste experimento foi medido o tempo de transmissão médio ida e volta de requisições por dados nomeados em diferentes tamanhos de rede. Espera-se avaliar o desempenho de cada paradigma frente a este cenário.

Na Figura 6.1 é apresentado o resultado para as diferentes topologias. Nela, fica clara a relação de performance entre as diferentes abordagens utilizadas, o que se dá pelo uso de um controlador centralizado e de cache.

A rede IP tem os piores tempos, já que não possui cache e têm as tabelas de encaminhamento menos otimizadas que nos demais casos. O IP com SDN conta com o o controlador para definir as melhores rotas de acordo com uma visão centralizada e global da rede. Enquanto o ICN conta com esse mesmo controlador, além da presença de cache nos nós da rede, fazendo com que o caminho de requisições feitas possa ser encurtado quando o dado requisitado é encontrado na cache local de algum dispositivo ao longo da rota.

Figura 6.1: Delay RTT



Fonte: O Autor

6.2 Percentual de cache

Nesse experimento, queremos avaliar o impacto do volume de cache na rede ICN. Espera-se que, conforme a disponibilidade de cache aumenta, diminuam os tempos de transmissão.

Foi usada a topologia com 20 nós, na qual os experimentos executam mais rapidamente e, portanto, pode ser elevado o número de iterações, variando apenas o percentual de cache, entre 25%, 50%, 75% e 100%, na mesma taxa para todos os dispositivos, o resultado é mostrado na figura 6.2.

Assim como no cenário anterior, é mantida a ordem nos tempos de delay fim-a-fim entre as diferentes estruturas. Porém aqui se torna visível o fato de que a cache é o mecanismo que diferencia os tempos entre as arquiteturas IP e ICN, ambas com SDN. No momento em que a cache é zerada, ambas abordagens apresentam o mesmo nível de desempenho enquanto, conforme a cache é aumentada, os tempos de distanciam, favorecendo a abordagem ICN pois ela conta com alguns pacotes já espalhados pela rede.

Ademais, o gráfico evidencia que a cache estava sobredimensionada, uma vez que passando de 25%, o seu aumento não tem mais impacto sobre os tempos de transmissão médios. Em cenários reais, este seria um caso a ser evitado, já que a maior capacidade de armazenamento de baixa latência teria um custo relativamente alto e também impactaria no consumo e eficiência energética dos dispositivos.

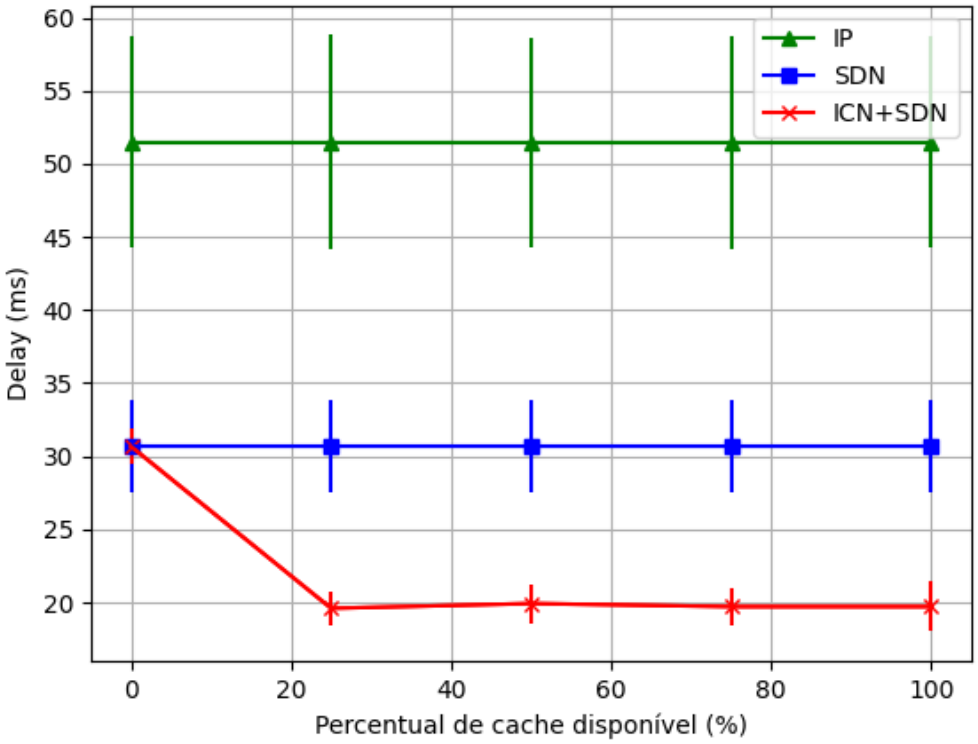
6.3 Número de fluxos de dados

Neste cenário, considerando que um fluxo de dados é uma transmissão de uma informação entre um par de dispositivos da rede, espera-se avaliar que, conforme mais fluxos são disseminados pela rede, maior a probabilidade de que algum dado seja encontrado na cache, o que ficaria evidenciado na redução dos tempos médios de transmissão.

Foi usada uma topologia com 20 nós, porém com um conjunto de dados modificado. Diferente dos outros experimentos, que usam as definições da tabela 4.2, nesse caso foi usado um único tipo de dado com período e *payload* fixos de 5 segundos e 500KB respectivamente. Variando o número de consumidores em cada transmissão, foram obtidos diferentes conjuntos de dados para cada valor do número de fluxos, representado no eixo horizontal do gráfico na Figura 6.3.

A fórmula do cálculo do número de fluxos está representada na Equação 6.1, em

Figura 6.2: Impacto da variação de cache no delay RTT de uma rede com 20 nós



Fonte: O Autor

que T representa o período, N_{Prod} o número de produtores (igual ao número de nós da rede), N_{Cons} o número de consumidores (variado em cada execução), t a duração do experimento (fixado em 100 segundos) e $N_{DataFlows}$ é o resultado final.

$$N_{DataFlows} = \frac{t}{T} N_{Prod} N_{Cons} \tag{6.1}$$

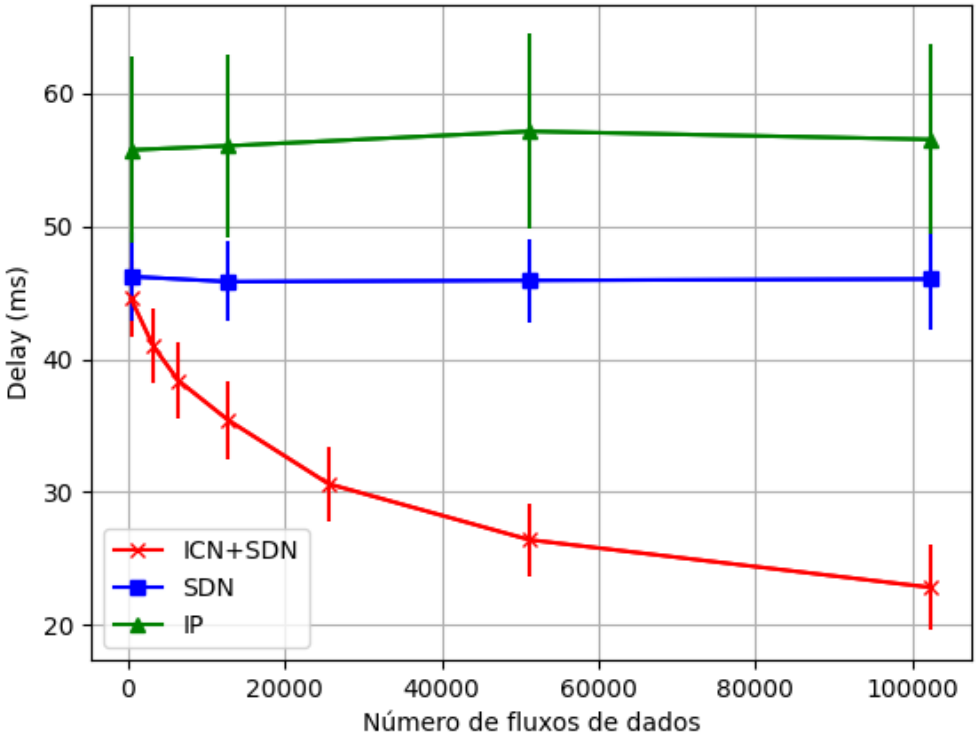
Para os experimentos com redes IP, os tempos se mantém constantes exceto por pequenas variações causadas por peculiaridades da lista de dados transmitidos no experimentos com menos fluxos. Conforme são distribuídos mais dados pela rede, essas peculiaridades tendem a ser dissolvidas nos tempos médios de transmissão.

No caso da rede ICN, os tempos tendem a diminuir conforme são introduzidos novos fluxos pois os dados são dispersos na rede, os aproximando cada vez mais dos dispositivos consumidores e aumentando a probabilidade de serem encontrados na cache.

Conforme são adicionados mais fluxos além do que foi testado, a tendência é que a rede siga o comportamento demonstrado até que fique congestionada, causando um aumento nos tempos de transmissão. Espera-se que esse ponto de congestionamento

apresente um valor maior para as redes IP do que para a rede ICN já que, com a cache distribuída, haverá menos fluxo pelos enlaces entre dispositivos. Entretanto, este cenário não poderia ser demonstrado para a rede ICN neste experimento, já que o volume de dados diferentes que entra na rede não varia, mas sim apenas quantas vezes os dados que entram na rede são consumidos por outros nós.

Figura 6.3: Impacto do número de fluxos de dados no Delay RTT de uma rede com 20 nós



Fonte: O Autor

6.4 *Overhead* de requisições de interesse

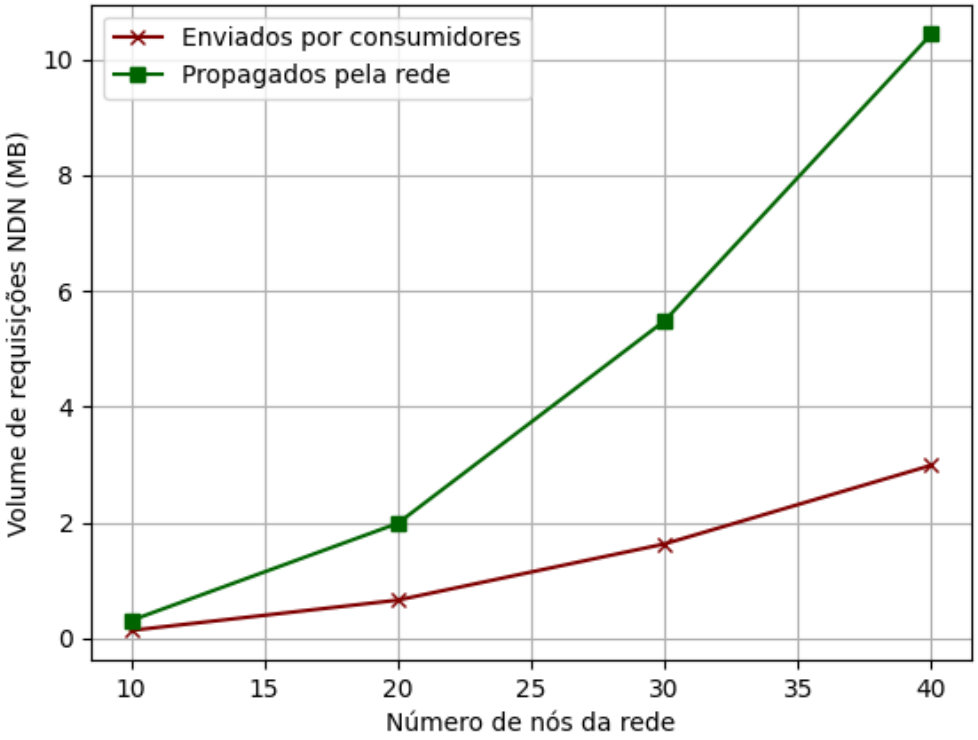
A base do funcionamento da comunicação orientada a dados é o envio de mensagens de requisição. Diferente do que ocorre na comunicação orientada a endereços, em que dispositivos podem estabelecer fluxos de dados de uma via entre dois pontos.

Com isso, existe um volume de dados adicional que é gerado a partir das mensagens de interesse enviadas pelo nó que requisita um dado e dos repasses feitos dessa mensagens pela rede. Estas mensagens tem tamanho variável na média de 94 bytes, considerando os cabeçalhos de todas as camadas de rede adicionados ao pacote.

Como as requisições são propagadas pela rede até que encontrem o dado sendo procurado, espera-se que o número de requisições geradas pelos consumidores seja amplificado pela rede. Tendo um valor para o volume de dados que estas requisições geram na rede, podemos avaliar qual é o impacto do *overhead* adicional gerado por mensagens de interesse no uso do ICN.

Uma visão desse *overhead* é representada no gráfico da Figura 6.4. Nesse experimento foi avaliado o volume de dados gerados pelas requisições de um nó consumidor e o reflexo disso no total de repasses dessas mensagens pela rede como um todo, variando o número de nós da topologia.

Figura 6.4: Volume de requisições NDN



Fonte: O Autor

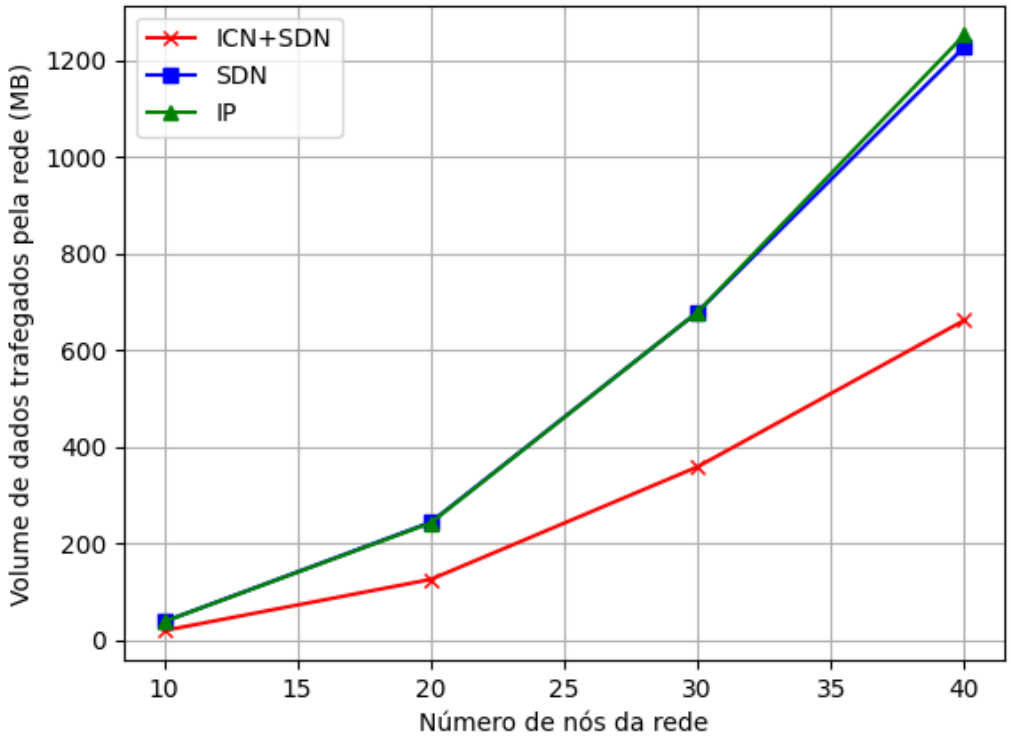
6.5 Volume de dados propagados na rede

Aprofundando a análise iniciada na subseção anterior, queremos avaliar o volume de dados que trafegam pela rede com o emprego do ICN. A partir desses dados, podemos ter uma melhor ideia do *tradeoff* entre o volume de dados por mensagens interesses e

o volume de dados em mensagens de resposta e, a partir disso, concluir se o *overhead* compensa no volume de dados total.

Esta representação, que usa as mesmas topologias do gráfico mencionado anteriormente, pode ser vista na Figura 6.5.

Figura 6.5: Volume de dados trafegados na rede



Fonte: O Autor

Desta forma, podemos concluir que, mesmo com esse *overhead* adicional de mensagens de interesse, a rede ainda se beneficia da comunicação orientada a dados. Isto ocorre porque as mensagens de interesse são pequenas e, aliadas ao uso de cache, fazem com que sejam evitadas diversas transmissões quando um dado requisitado é encontrado ao longo do caminho até o produtor. Assim evitando o uso de banda em todos os enlaces pelos quais o dado teria que trafegar caso não fosse encontrado em cache.

7 CONCLUSÃO

A ferramenta desenvolvida permitiu que fossem realizados experimentos comparativos entre três diferentes arquiteturas de rede, agregando requisitos expressos pelo contexto em que este trabalho se dá. De maneira simplificada, a emulação da comunicação sem fio ilustra o cenário expresso em uma rede IoBT, o controlador SDN permite o correto funcionamento de aplicações de comando e controle e o ICN foi a abordagem estudada sob a premissa de melhor desempenho para os padrões de uso no cenário estudado.

O uso da abordagem ICN com controlador SDN pode atuar de algumas formas para melhorar a qualidade da rede, principalmente no que tange à eficiência no uso da capacidade dos enlaces e à priorização de fluxos de dados afim de atingir diferentes taxas de entrega para diferentes tipos de informações.

O SDN permite a definição dinâmica da priorização de fluxos com base em parâmetros da missão. Essa definição é essencial para o conceito de agilidade C2, já que as condições e os parâmetros da missão podem mudar drasticamente em uma pequena janela de tempo. Mas, mesmo diante da necessidade desta rápida alteração em cenários adversos, conseguimos ainda manter o padrão de qualidade de funcionamento da rede.

Do ponto de vista da distribuição de informação, o ICN e o SDN são combinados para possibilitar entregas mais rápidas, eficientes e inteligentes. Isto com um *overhead* baixo tendo em vista o volume total de dados que trafegam pela rede e a economia de tempo e de banda utilizada. O custo real desta abordagem se dá nas camadas necessárias para o seu funcionamento na infraestrutura atualmente usada em larga escala da internet, que não tem suporte nativo ao ICN.

Estas observações mostram que a combinação elencada como promissora para os requisitos levantados na contextualização do problema proposto de fato supera as outras alternativas comparadas.

Os cenários analisados só não se beneficiariam da abordagem ICN + SDN se não fossem voltados a distribuição de informação. Se cada um dos os fluxos de dados se desse apenas entre dois únicos dispositivos da rede, não teríamos um cenário otimizado. Isto porque, nessa situação, não haveria quaisquer dados compartilhados entre dois consumidores e, portanto, nunca seriam encontrados na cache.

Porém, este cenário é muito pouco provável, especialmente conforme o uso geral de redes de dispositivos evoluem no sentido de ter cada vez mais dados compartilhados, como por exemplo: sites, vídeos e serviços de *streaming* de músicas e programas de

televisão.

O resultado deste trabalho agregou funcionalidades presentes em algumas outras ferramentas em uma só, portanto facilitando o desenvolvimento de experimentos com ICN, SDN e Wifi. A partir desse produto, poderiam ser realizados diversos experimentos e melhorias afim de possibilitar o estudo de outras configurações e parâmetros da rede e dos dispositivos que a compõem.

Em trabalhos futuros, poderiam ser realizados experimentos afim de explorar cenários mais detalhados e específicos relacionados à comunicação sem fio, variando o protocolo utilizado afim de melhor emular redes de sensores com o protocolo *6LowPAN*, por exemplo. Ou estudar o comportamento da rede diante de dispositivos que perdem a conectividade ou mudam de posição no espaço físico e, portanto, na topologia da rede. Ou também diferentes formas de organização da rede, tanto física quanto logicamente, com o auxílio do controlador SDN.

REFERÊNCIAS

AHLGREN, B. et al. A survey of information-centric networking. **IEEE Communications Magazine**, v. 50, n. 7, nov. 2012.

ALBERTS, D. S. et al. Sas-085 final report on c2 agility. 2013.

ALBERTS D. S. HAYES, R. E. **Understanding Command and Control**. [S.l.]: CCRP, 2006.

AZMOODEH, A.; DEHGHANTANHA, A.; CHOO, K. R. The internet of battle things. **IEEE Transactions on Sustainable Computing**, v. 4, 2018.

B., M. N.; CHARIGLIONE, L. The potential of information centric networking in two illustrative use scenarios: mobile video delivery and network management in disaster situations. **E-LETTER**, 2013.

BACCELLI, E. et al. Information centric networking in the iot: Experiments with ndn in the wild. **Proceedings of the 1st ACM Conference on Information-Centric Networking**, p. 77–86, jun. 2014. Available from Internet: <http: //doi.acm.org/10.1145/2660129.2660144>. Accessed in: 11 mai. 2021.

FONTES, R. R. et al. Mininet-wifi: Emulatingsoftware-defined wireless networks. 2015.

HAYAMIZU, Y.; MATSUZONO, K.; ASAEDA, H. Real-time video streaming using ceforesim: Simulator to the real world. Nov 2020.

KARMAKAR, K. K. et al. Sdn-enabled secure iot architecture. **IEEE Internet of Things Journal**, 2020.

KOTT, A.; ALBERTS, D. S. How do you command an army of intelligent things? **Computer**, v. 50, n. 12, p. 96–100, 2017.

KOTT, A.; SWAMI, A.; WEST, B. J. The internet of battle things. **Computer**, v. 49, n. 12, p. 70–75, 2016.

LEAL, G. M. et al. Empowering command and control though a combination of information-centric networking and software defined networking. **IEE Communications MAgazine**, 2019.

MASTORAKIS, S.; AFANASYEV, A.; ZHANG, L. On the evolution of ndnsim: an open-source simulator for ndn experimentation. **ACM SIGCOMM Computer Communication Review**, p. 19,33, Jul 2017.

MOLL, P.; THEUERMANN, S.; HELLWAGNER, H. Wireless network emulation for research on information-centric networking. Oct 2018.

NUNES, B. A. A. et al. A survey of software-defined networking: Past, present, and future of programmable networks. **IEEE Communications Surveys Tutorials**, v. 16, n. 3, p. 1617–1634, 2014.

OH, S. Y.; LAU, D.; GERLA, M. Content centric networking in tactical and emergency manets. **IFIP Wireless Days**, n. 1, p. 1–5, oct. 2010.

POULARAKIS, K.; IOSIFIDIS, G.; TASSIULAS, L. Sdn-enabled tactical ad hoc networks: Extending programmable control to the edge. **IEEE Communications Magazine**, v. 56, p. 132–138, 2018.

SAKET. **What is SDN and How does it Work?** 2018. Available from Internet: <https://www.it4nextgen.com/sdn-software-defined-networking/>.

SIRACUSANO, G. et al. A framework for experimenting icn over sdn solutions using physical and virtual testbeds. **Computer Networks**, v. 134, p. 245–259, 2018.

TAN, L.; WANG, N. Future internet: The internet of things. 2010.

V., T. J. et al. Evaluating cros-ndn: a comparative performance analysis of a controller-based routing scheme for named-data networking. **Journal of Internet Services and Applications**, v. 10, n. 20, 2019.

WICKBOLDT, J. A. et al. Software-defined networking: management requirements and challenges. **IEEE Communications Magazine**, v. 53, n. 1, p. 278–285, jan. 2015.

YAQUB, M. A.; H., A. S.; KIM, D. Chapter 2 information-centric networks (icn). 2017. Available from Internet: <https://www.semanticscholar.org/paper/Chapter-2-Information-Centric-Networks-(-ICN-)-Yaqub-Ahmed/3fdf506b1df73feb3a8d59cff8c50c301d91ec74>.

ZHANG, L. et al. Named data networking. **ACM SIGCOMM Computer Communication Review**, v. 44, p. 66, 73, 2014.

APÊNDICE A: Exemploe de inicialização de uma rede simples no MininetWifi e NFD

```
#!/usr/bin/python

'This example creates a simple network topology with 1 AP and 2 station

import sys

from functools import partial

from mininet.log import setLogLevel, info
from mininet.node import RemoteController
from mn_wifi.node import Station
from mn_wifi.cli import CLI
from mn_wifi.net import Mininet_wifi


def topology():

    privateDirs = [ ( '/var/log', '/tmp/%(name)s/var/log' ),
                    ( '/var/run', '/tmp/%(name)s/var/run' ),
                    ( '/run', '/tmp/%(name)s/run' ),
                      '/var/mn' ]
    station = partial( Station,
                       privateDirs=privateDirs )
    "Create a network."
    net = Mininet_wifi(station=station)

    info("*** Creating nodes\n")
    sta_arg = dict()
    ap_arg = dict()
    if '-v' in sys.argv:
        sta_arg = {'nvif': 2}
```

```
else:
    # isolate_clientes: Client isolation can be used to prevent
    # low-level bridging of frames between associated stations
    # in the BSS.
    # By default, this bridging is allowed.
    # OpenFlow rules are required to allow communication among node
    ap_arg = {'client_isolation': True}


ap1 = net.addAccessPoint('ap1', protocols='OpenFlow13',
    ssid="simpletopo", mode="g", channel="5", position='20,20,0',
    **ap_arg)

ap2 = net.addAccessPoint('ap2', protocols='OpenFlow13',
    ssid="simpletopo2", mode="g", channel="11", position='80,20,0',
    **ap_arg)

ap3 = net.addAccessPoint('ap3', protocols='OpenFlow13',
    ssid="simpletopo3", mode="g", channel="11", position='80,80,0',
    **ap_arg)

ap4 = net.addAccessPoint('ap4', protocols='OpenFlow13',
    ssid="simpletopo4", mode="g", channel="11", position='20,80,0',
    **ap_arg)

sta1 = net.addStation('sta1', position='10,10,0', **sta_arg)
sta2 = net.addStation('sta2', position='90,10,0', )
sta3 = net.addStation('sta3', position='90,90,0', )
sta4 = net.addStation('sta4', position='10,90,0', )

c0 = net.addController('c0', controller=RemoteController,
    ip='127.0.0.1', port=6653)

info("*** Configuring propagation model\n")
```

```
    net.setPropagationModel(model='logDistance', exp=4.5)

    info("*** Configuring wifi nodes\n")
    net.configureWifiNodes()

    info("*** Associating Stations\n")
    net.addLink(ap1, ap2)
    net.addLink(ap2, ap3)
    net.addLink(ap3, ap4)

    net.addLink(sta1, ap1)
    net.addLink(sta2, ap2)
    net.addLink(sta3, ap3)
    net.addLink(sta4, ap4)

    if '-p' not in sys.argv:
        net.plotGraph(max_x=100, max_y=100)

    info("*** Starting network\n")
    net.build()
    c0.start()
    ap1.start([c0])
    ap2.start([c0])
    ap3.start([c0])
    ap4.start([c0])

    if '-v' not in sys.argv:
        ap1.cmd('ovs-ofctl add-flow ap1 "priority=0,arp,in_port=1,'
                'actions=output:in_port,normal"')
        ap1.cmd('ovs-ofctl add-flow ap1 "priority=0,icmp,in_port=1,'
                'actions=output:in_port,normal"')
        ap1.cmd('ovs-ofctl add-flow ap1 "priority=0,udp,in_port=1,'
                'actions=output:in_port,normal"')
        ap1.cmd('ovs-ofctl add-flow ap1 "priority=0,tcp,in_port=1,'
```

```
            'actions=output:in_port,normal"')

info("*** Starting NFD processes\n")
nfd1 = sta1.popen("nfd")
nfd2 = sta2.popen("nfd")
nfd3 = sta3.popen("nfd")
nfd4 = sta4.popen("nfd")

info("*** Creating faces and routes in sta1\n")
sta1.cmd("nfdc face create udp://10.0.0.2")
info(sta1.cmd("nfdc route add /sta2 udp://10.0.0.2"))
info(sta1.cmd("nfdc route add /sta3 udp://10.0.0.2"))
info(sta1.cmd("nfdc route add /sta4 udp://10.0.0.2"))

info("*** Creating faces and routes in sta2\n")
sta2.cmd("nfdc face create udp://10.0.0.3")
sta2.cmd("nfdc face create udp://10.0.0.1")
sta2.cmd("nfdc route add /sta3 udp://10.0.0.3")
sta2.cmd("nfdc route add /sta4 udp://10.0.0.3")

info("*** Creating faces and routes in sta3\n")
sta3.cmd("nfdc face create udp://10.0.0.4")
sta3.cmd("nfdc face create udp://10.0.0.2")
sta3.cmd("nfdc route add /sta4 udp://10.0.0.4")

info("*** Running CLI\n")
CLI(net)

info("*** Stopping NFD\n")
nfd1.kill()
nfd2.kill()
nfd3.kill()
nfd4.kill()
```

```
    info("*** Stopping network\n")
    net.stop()


if __name__ == '__main__':
    setLogLevel('info')
    topology()
```

APÊNDICE B: Criação de filas no TC

```
tc qdisc del dev ap1-wlan1 root
tc qdisc add dev ap1-wlan1 root handle 1: htb default 1 direct_qlen 100
tc class add dev ap1-wlan1 classid 1:fffe htb rate 54mbit ceil 54mbit \
    burst 1500b cburst 1500b
tc class add dev ap1-wlan1 classid 1:fa parent 1:fffe htb rate 12Kbit \
    ceil 2Mbit burst 512b cburst 512b
# cria filtro para classificar tráfego através do skb_mark (250 = 0xfa)
tc filter add dev ap1-wlan1 parent 1: prio 1 protocol ip handle 0xfa \
    fw flowid 1:fa action ok

# Adiciona flow com limite de 4mbps para tráfego 10.0.0.3 -> 10.0.0.1
# (Tag 250 = 0xfa)
ovs-ofctl -O OpenFlow13 add-flow ap1 "table=0,priority=10,ip," + \
    "nw_src=10.0.0.3,nw_dst=10.0.0.1," + \
    "actions=set_field:250->pkt_mark,goto_table:1"
```

APÊNDICE C: Criação das rotas NFD

```
def createNfdRoutes(self):

   lstHostLinks = self.abstractApsFromLinks()
   # Create faces to all neighbouring hosts
   for pStation in self.net.stations:
      for topoLink in lstHostLinks:
         strDest = topoLink.connectsTo(pStation.name)
         pDestHost = Topology.findNodeByName(strDest,
            self.net.stations)
         if (pDestHost):
            pStation.cmd('nfdc face create udp://' + pDestHost.IP())

   # Create routes between faces
   # Create NX graph for the network topology
   pGraph = nx.Graph()
   for pLink in lstHostLinks:
      nWeight = 0
      if ('delay' in pLink.kwargs):
         strDelay = pLink.kwargs['delay']
         nPos = strDelay.find('ms')
         if (nPos > 0):
            # Remove ms suffix
            nWeight = int(strDelay[0:nPos])
         else:
            nWeight = int(strDelay)
      pGraph.add_edge(pLink.strNode1, pLink.strNode2, weight=nWeight)

   # Create routes starting on each host
   for pStart in self.net.stations:
      for pEnd in self.net.stations:
         if (pStart != pEnd):
            lstPath = nx.shortest_path(pGraph, pStart.name,
```

```
    pEnd.name, weight='weight')
if (len(lstPath) >= 2):
    strNextHost = lstPath[1]
    pNextHost = Topology.findNodeByName(strNextHost,
        self.net.stations)
else:
    #  There should be at lease 2 hosts in this route
    raise Exception
```