UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS CRATEÚS

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

WELLINGTON SOARES ARAÚJO

ESTUDO COMPARATIVO DA UTILIZAÇÃO DE PROTOCOLOS DA CAMADA DE TRANSPORTE E APLICAÇÃO EM AMBIENTES DE MOBILE CLOUD COMPUTING

CRATEÚS - CEARÁ

2022

WELLINGTON SOARES ARAÚJO

ESTUDO COMPARATIVO DA UTILIZAÇÃO DE PROTOCOLOS DA CAMADA DE TRANSPORTE E APLICAÇÃO EM AMBIENTES DE MOBILE CLOUD COMPUTING

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus Crateús da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Me. Francisco Anderson de Almada Gomes.

Coorientador: Prof. Me. Filipe Fernandes dos Santos Brasil de Matos.

CRATEÚS - CEARÁ

2022

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S657e Soares Araújo, Wellington.
Estudo comparativo da utilização de protocolos da camada de transporte e aplicação em ambientes de mobile cloud computing / Wellington Soares Araújo. – 2023.
66 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, 3, Crateús, 2023.
Orientação: Prof. Me. Francisco Anderson de Almada Gomes.
Coorientação: Prof. Me. Filipe Fernandes dos Santos Brasil de Matos.

1. Mobile Cloud Computing. 2. Offloading. 3. Device-to-Device. 4. Camada de Transporte. 5. Camada de Aplicação. I. Título.

CDD

WELLINGTON SOARES ARAÚJO

ESTUDO COMPARATIVO DA UTILIZAÇÃO DE PROTOCOLOS DA CAMADA DE TRANSPORTE E APLICAÇÃO EM AMBIENTES DE MOBILE CLOUD COMPUTING

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus Crateús da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Aprovada em:

BANCA EXAMINADORA

Prof. Me. Francisco Anderson de Almada Gomes (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Me. Filipe Fernandes dos Santos Brasil de Matos (Coorientador)
Universidade Federal do Ceará (UFC)

Prof. Me. Maurício Moreira Neto
Centro Universitário Christus (UniChristus)

Ma. Vitória Regina Nicolau Silvestre
Fiserv Brasil

Dedico este trabalho aos meus pais, demais familiares e amigos.

AGRADECIMENTOS

Primeiramente a Deus, por ser sempre o meu ponto de apoio quando me senti perdido durante essa caminhada.

Aos meus pais, Enoque Soares e Ana Lúcia, por sempre acreditar em mim e me lembrar por diversas vezes que tudo valeria a pena.

Ao meus irmãos, Raquel, Paulinha, Rodrigo e Pedro, pelo incentivo durante essa caminhada.

A minha esposa, Brena, que esteve sempre do meu lado mesmo nos momentos mais difíceis dessa caminhada.

Ao meu orientador, professor Anderson Almada e meu coorientador, professor Filipe Fernandes, por aceitar e me guiar durante a realização deste trabalho, por sempre está disponível para conversar e me aconselhar independentemente do horário.

A Universidade Federal do Ceará, pela oportunidade de fazer a graduação e ser minha casa durante todo esse período.

A todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

“Em um mundo em que mudanças estão ocorrendo rapidamente, a única estratégia que terá garantia de fracasso é a de não correr riscos.”

(Mark Zuckerberg)

RESUMO

Com o crescimento da tecnologia da informação e o surgimento da internet, os dispositivos móveis (por exemplo (e.g.), *tablets* e *smartphones*) vêm se tornando essenciais para a vivência humana, pelo fato de serem dispositivos que podem se conectar á *internet* de qualquer lugar apenas utilizando a rede de telefonia celular ou redes *Wi-Fi*. Por outro lado, as aplicações móveis vêm exigindo cada vez mais recursos desses dispositivos, o que os forçam a adotar estratégias para suprir essa demanda. Algumas estratégias foram criadas, como a *Mobile Cloud Computing* (MCC), que utilizam diversas abordagens, como o *offloading* para migrar recursos do dispositivo móvel como dados e processamento para a Nuvem. Porém, para a realização do *offloading* faz-se necessário um estabelecimento de comunicação entre o dispositivo móvel e o ambiente remoto, que em casos de desastres naturais como terremotos e *tsunamis* podem comprometer essa comunicação, fazendo-se necessário o uso de outros meios de comunicação, como a abordagem *Device-to-Device* (D2D). Tendo em vista a necessidade da comunicação D2D, com o intuito de garantir a interação entre os usuários em consequência de um desastre natural e apresentado o uso do *offloading* para realização de migração de tarefas, este trabalho tem como proposta analisar o desempenho de protocolos de comunicação da camada de *Transporte* e *Aplicação* no ambiente de MCC, para entender o impacto causado pelo uso dos mesmos e quais desses protocolos melhoram a experiência do usuário baseada no desempenho. Como resultado final, foi possível concluir que de uma maneira geral o protocolo *Transmission Control Protocol* (TCP) teve um melhor desempenho em relação os demais, tendo sua taxa média de influência no processo total de *offloading* de até 39% enquanto outros protocolos como o *Reliable UDP* (RUDP) teve sua taxa média de até 97%. Já realizando uma análise por camadas, foi percebido que na camada de *Aplicação*, o protocolo *Hypertext Transfer Protocol* (HTTP) teve um melhor desempenho com menor influência e consequentemente menor tempo de *download* e *upload* em comparação com o *MQ Telemetry Transport* (MQTT), e na camada de *Trasnporte*, confirmando o que foi citado anteriormente, o protocolo TCP se mostrou uma melhor opção em frente ao *RUDP*.

Palavras-chave: *Mobile Cloud Computing*. *Offloading*. *Device-to-Device*. Dispositivos Móveis, Camada de Transporte. Camada de Aplicação

ABSTRACT

With the growth of Information Technology and the emergence of the Internet, mobile devices (e.g., *tablets and smartphones*) become essential tools for human lives, because this type of devices can connect into network anywhere through a Mobile Network Service or a Wi-Fi network. However, mobile applications require more and more resources from those devices, which forces them to adopt strategies to supply that demand. Some strategies we have been created, like the MCC, which uses some approaches like *offloading* to migrate resources like data and process to the Cloud. But, in order to execute, offloading *offloading* needs to establish a communication between the mobile device and remote environment, which in cases like natural disasters or attacks can interrupt that communication, making it necessary to use other ways to communicate, like the D2D approach. Keeping in mind the necessity to use the D2D approach, to ensure the interaction between users when happens a natural disaster and presented the *offloading* to migrate the tasks, this paper has as propose to analyze the communication protocol performance into *Transport* Layer and *Application* Layer in *MCC* environment, to understand the impact made by the usage of those protocols, and determinate which protocols improve the user experience based on performance. As a final result, it was possible to conclude that in general the TCP had a better performance compared to the others, having its average influence rate in the total offloading process of 39% while other protocols such as RUDP had its average rate of 97%. Performing an analysis by layers, it was noticed that in the *Application* layer, the HTTP performed better with less influence and consequently less download and upload time compared to the MQTT, and in the *Transport* layer, confirming what was mentioned previously, the TCP protocol proved to be a better option over RUDP.

Keywords: Mobile Cloud Computing. Offloading, Device-to-Device. Mobile Devices. Transport Layer. Application Layer

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

CAOS	*Context Acquisition and Offloading System*
D2D	*Device-to-Device*
DoD	*Departament of Defense*
e.g.	por exemplo
FCC	*Function-Centric Computating*
GCS	*Ground Control Station*
GUI	*Graphic User Interface*
HTTP	*Hypertext Transfer Protocol*
i.e.	Isto é
IEEE	*Institute of Electrical and Electronics Engineers*
IoT	*Internet of Things*
MCC	*Mobile Cloud Computing*
MQTT	*MQ Telemetry Transport*
OSI	*Open Systems Interconnection*
QUIC	*Quick UDP Internet Connections*
RUDP	*Reliable UDP*
SWP	*Sliding Window Protocol*
TCP	*Transmission Control Protocol*
UAV	*Unmanned Aerial Vehicles*
UDP	*User Datagram Protocol*
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locator*

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização

Diante de um mundo cada dia mais conectado onde todos se comunicam através da *Internet*, os dispositivos móveis (*e.g.*, *tablets* e *smartphones*) vieram como uma forma de manter as pessoas conectadas a esse mundo independentemente do local, se tornando assim objetos essenciais no dia-a-dia das pessoas (CISCO, 2017). Com isso, o aumento excessivo do uso de dispositivos móveis trouxe consigo o grande consumo de aplicações móveis, seja elas voltadas para entretenimento, finanças, saúde ou estudo (FERNANDO *et al.*, 2013).

Aplicações móveis podem auxiliar de várias formas, como dando acesso as notícias do dia de acordo com o gosto do usuário, ou até mesmo facilitando o pagamento de contas com apenas alguns cliques. Uma pesquisa realizada em Juniper (2021), mostra que no ano de 2023 terá um valor de 49 bilhões de transações utilizando dispositivos móveis, isso é um aumento de 93% com relação ao ano de 2021.

1.2 Justificativa

Mesmo com o avanço contínuo da tecnologia e a melhoria significativa dos dispositivos móveis ao longo dos anos, esses ainda contém limitações (*e.g.*, processamento e bateria) que os colocam como dispositivos com recursos limitados (FERNANDO *et al.*, 2013).

Dado o aumento da demanda por aplicações móveis, isso provoca também um aumento nos recursos exigidos por elas para entregar uma boa experiência ao usuário, de forma mais aprimorada (QURESHI *et al.*, 2011). Para suprir essa demanda, algumas estratégias foram criadas, como a *Mobile Cloud Computing* (MCC), que tem como objetivo utilizar dos recursos da Nuvem para aumentar o poder computacional e/ou armazenamento dos dispositivos móveis, consequentemente, melhorar a performance das aplicações que são executadas nesses aparelhos.

O uso da MCC pode trazer vários benefícios ao que diz respeito a expansão de recursos computacionais, uma vez que dá a possibilidade de migrar dados de um dispositivo móvel para a Nuvem (SHIRAZ *et al.*, 2013). Uma estratégia abordada dentro da MCC é o *offloading*, descrito por Kumar *et al.* (2013) como um procedimento onde acontece a migração

do recursos computacionais de um dispositivo móvel para Nuvem, com o intuito de expandir os recursos como: processamento e memória, além de reduzir o consumo de energia que seria exigido para executar esse processo localmente no dispositivo.

Para a realização do *offloading*, é necessário um estabelecimento de comunicação entre o dispositivo móvel e o ambiente remoto, que irá executar as tarefas requisitadas. Se caso o local de execução seja uma Nuvem pública, é necessário a utilização da Internet para alcançar esse ambiente. Em casos de desastres naturais, como o terremoto que ocorreu no Japão no ano de 2011, que acabou com a utilização da Internet dos dispositivos móveis, ocasionando um problema de comunicação[1], é necessário o uso de outros meios de comunicação, como a abordagem *Device-to-Device* D2D, que é definido como uma comunicação direta entre dois dispositivos móveis sem intermédio de uma rede central. Além disto, ambas as abordagens, tanto utilizando recursos da Nuvem como utilizando a abordagem D2D, é utilizado um conjunto de protocolos que padroniza a comunicação, chamado *TCP/IP*.

Tendo em vista a necessidade da comunicação D2D em ambientes como o citado anteriormente e com o intuito de garantir a interação entre os usuários em consequência de um desastre natural, assim como a possibilidade de realização de tarefas das aplicações móveis por meio do *offloading* para poupar recursos, esse trabalho tem como proposta analisar o desempenho de protocolos de comunicação da camada de *Transporte* e *Aplicação* no ambiente de MCC, para avaliar quais desses protocolos melhoram a experiência do usuário baseada no desempenho.

1.3 Objetivos

1.3.1 Objetivo Geral

Avaliar os impactos causados pela utilização de diferentes protocolos da Camada de *Transporte* e *Aplicação* na realização de *offloading* no ambiente D2D.

1.3.2 Objetivos Específicos

- Modelar e implementar um módulo de comunicação expansível dos protocolos para a realização do *offloading*;

[1] https://super.abril.com.br/ideias/catastrofes-que-podem-acabar-com-o-mundo/

- Analisar protocolos das camadas de *Transporte* e *Aplicação* no processo de *offloading*;
- Realizar uma análise comparativa do tempo de processamento do *offloading* com diferentes protocolos;
- Avaliar o tempo gasto dos protocolos escolhidos na realização de *Upload* e *Download* de dados;

1.4 Organização do Trabalho

Este trabalho está organizado em 6 (seis) capítulos. Inicialmente foi dado uma introdução ao tema e contextualização do ambiente onde o problema está inserido, a motivação desse trabalho e seus objetivos.

O Capítulo 2 aborda a fundamentação teórica desse trabalho, falando sobre assuntos como Computação Móvel, *Cloud Computing*, *Mobile Cloud Computing*, *offloading* e Comunicação Sem Fio com foco na Camada de Transporte e Aplicação.

O Capítulo 3 apresenta os trabalhos relacionados e que tem proposta semelhantes com esse trabalho, focando em diferentes protocolos de comunicação.

O Capítulo 4 apresenta a proposta do trabalho, mostrando a metodologia adotada, os protocolos escolhidos para análise, a arquitetura da solução e detalhes de implementação.

O Capítulo 5 apresenta os testes executados para validar a solução proposta e verificar os resultados encontrados para a avaliação dos protocolos de comunicação escolhidos.

Por fim, o último capítulo apresenta a conclusão do trabalho, bem como os trabalhos futuros para a melhoria do trabalho e futuras pesquisas.

2 FUNDAMENTAÇÃO TEÓRICA

Este Capítulo apresenta os conceitos e definições relacionados às áreas e subáreas que fundamentam este trabalho. A Seção 2.1 trata sobre a Computação Móvel. A Seção 2.2 apresenta os conceitos básicos relacionados a Computação em Nuvem. Já a Seção 2.3 apresenta os conceitos principais em torno do tema principal desse trabalho que é a *Mobile Cloud Computing*, mais especificamente sobre o conceito de *offloading*. Por fim, os protocolos de comunicação sem fio é apresentado na Seção 2.4.

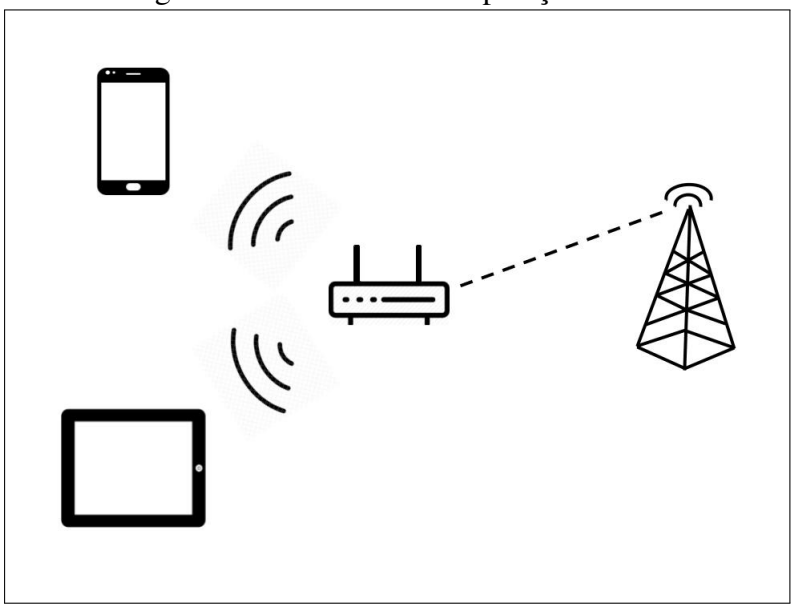
2.1 Computação Móvel

A *Computação Móvel* surge como uma quarta revolução na computação antecedida pelos grandes centros e processamento de dados da década de sessenta, o surgimento dos terminais nos setenta, e as redes de computadores na década de oitenta (MATEUS; LOUREIRO, 1998). A computação móvel pode ser definida como um novo paradigma computacional que nasce da união da infraestrutura de comunicação sem fio e dispositivos portáteis de computação, que dão aos usuários a possibilidade de acessar informações e colaboram com outras pessoas independentemente da sua localização (GUPTA, 2008).

Com o avanço na tecnologia e áreas afins, os meios de comunicação computacionais que eram fixos passaram para computadores móveis e portáteis, que conseguem trocar mensagens e se comunicar mesmo estando em movimento contínuo (FORMAN; ZAHORJAN, 1994). A Figura 1 demostra o funcionamento de uma rede utilizando o paradigma de Computação Móvel, onde o dispositivo móvel pode ter acesso às informações a partir de redes de telefonia celular (*e.g.*, *3G, 4G e 5G*) ou redes *Wi-Fi*.

Mesmo sabendo que a computação móvel trouxe diversos benefícios, não se pode negar que existem limitações, pois com o passar do tempo as demandas de processamento aumentaram significativamente em relação à evolução do paradigma. Gupta (2008) afirma que as limitações são de energia, tempo de processamento e o tamanho da memória. Sendo colocado alguns aspectos sobre a computação móvel, a próxima Seção aborda um tema chamado de *Computação em Nuvem* que objetiva aumentar os recursos dos sistemas quando necessário.

Figura 1 – Modelo de Computação Móvel



Fonte – Silva (2019)

2.2 Computação em Nuvem

Segundo Wang *et al.* (2010), a *Computação em Nuvem* é um conjunto de serviços disponíveis na rede, provendo escalabilidade, *QoS* (ou Qualidade de Serviço) garantida, geralmente personalizada, infraestrutura barata e sob demanda. Esse paradigma vem ganhando grande resplendor na área da Computação, por dar possibilidade aos usuários moverem seus arquivos e aplicações para a Nuvem e acessá-los de maneira fácil e rápida, se tiver acesso à Internet.

Somula e Sasikala (2018) citam cinco características importantes da computação em Nuvem, que são:

(i) Autoatendimento sob demanda - que diz respeito a possibilidade do usuário requisitar serviços diretamente da Nuvem sem a necessidade de interação com o provedor de serviços;

(ii) Amplo acesso à rede - o cliente pode acessar serviços a qualquer momento de qualquer lugar através de dispositivos móveis, como: *laptops, smartphones e tablets*;

(iii) Agrupamento de recursos - múltiplos usuários podem acessar recursos computacionais (*e.g*, armazenamento, memória, largura de banda e capacidade de processamento) sem a necessidade de saber onde esses recursos estão armazenados;

(iv) Elasticidade rápida - baseado na demanda dos clientes, a Nuvem pode aumentar ou diminuir os seus recursos, fazendo o usuário pensar que os recursos são ilimitados e escaláveis a qualquer momento; e

(v) Serviço medido - controla a utilização de serviços de forma automática, fazendo com que o valor pago seja baseado na quantidade de recursos usados, mantendo a transparência tanto para o provedor de serviços quanto para o consumidor.

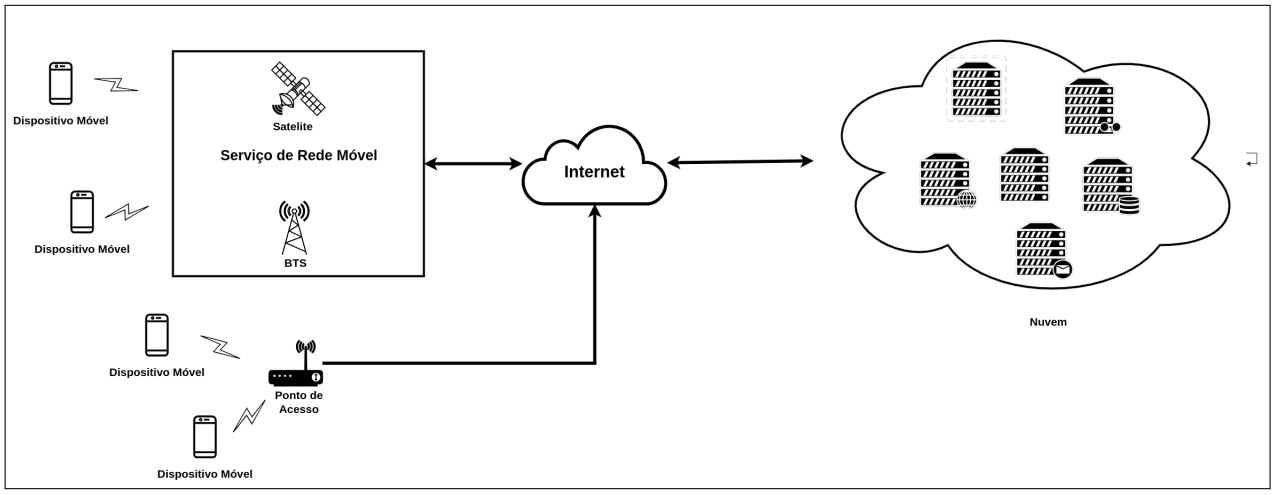
Proveniente da junção entre a computação móvel e a computação em Nuvem, nasce a *Mobile Cloud Computing*, que será abordada na seção a seguir.

2.3 *Mobile Cloud Computing*

Para Khan Mazliza Othman *et al.* (2013), a MCC se diz respeito a integração entre a computação em Nuvem e a computação móvel para criar dispositivos com mais disponibilidade de recursos (*e.g*, poder computacional, memória, armazenamento, energia e consciência de contexto).

A MCC está presente em diversos ambientes, desde aplicativos móveis para uso cotidiano como o aplicativo de *e-mail* da *Google Gmail for Mobile*[1] ou até serviços de armazenamento de arquivos como o *iCloud* da *Apple*[2] (FERNANDO *et al.*, 2013). A Figura 2 mostra detalhadamente como funciona a arquitetura de acesso à serviços em Nuvem nos dispositivos móveis no ambiente de MCC. O dispositivo móvel pode se conectar à Internet através das redes de telefonia móvel (*e.g. 3G, 4G e 5G*) ou redes sem fio (redes *Wi-Fi*), para assim ter acesso aos serviços prestado pela Nuvem.

Figura 2 – Arquitetura da Computação Móvel no ambiente de MCC



Fonte – Adaptado de Khan Mazliza Othman *et al.* (2013)

[1] <https://www.google.com/mobile/mail/>
[2] <http://www.apple.com/icloud/>

No contexto de MCC, exitem diversas estrategias para abordar esse paradigma. Como exemplo, pode-se citar o *offloading*, que será abordado na próxima seção desse trabalho.

2.3.1 Offloading

Segundo Santos *et al.* (2017), o *offloading* (descarregamento) diz respeito a um procedimento onde acontece a migração de recursos do dispositivo móvel para a Nuvem, afim de aumentar o poder computacional deste dispositivo, além de diminuir o consumo energético do mesmo. Para Gomes *et al.* (2017), o *offloading* é proposto com o principal intuito de economizar bateria do dispositivo, melhorar o desempenho das aplicações móveis, bem como melhorar também a colaboração entre os usuários das aplicações móveis.

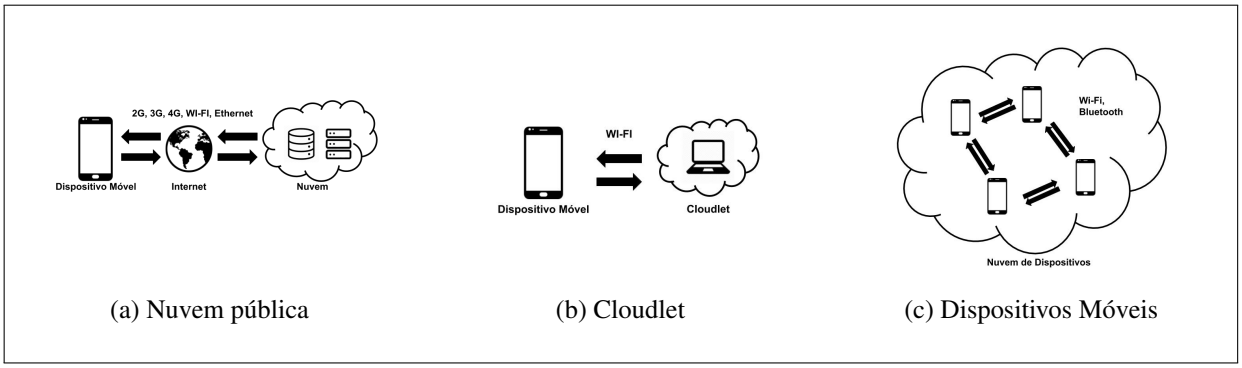
Para Fernando *et al.* (2013), existem basicamente dois tipos de *offloading*. A primeira é o ***offloading* de dados**, onde os dados são migrados do dispositivo móvel para a Nuvem, e então esses dados são armazenados. Isso faz com que o armazenamento local do dispositivo seja preservado, assim aumentando a capacidade do mesmo. A segunda é o ***offloading* de processamento**, que consiste de migrar o processamento do dispositivo móvel para a Nuvem, e uma vez que finalizar, o resultado desse processamento é enviado de volta para o dispositivo.

Fernando *et al.* (2013) e Kumar *et al.* (2013) também apontam várias questões que devem ser discutidas e observadas antes de utilizar o *offloading*, como: (i) **Por que realizá-lo?**; (ii) **Onde realizá-lo?**; (iii) **Quando realizá-lo?**; (iv) **Fazer *offloading* de quê?** e; (v) **Como executá-lo?**.

O *offloading* pode ser executado em três possíveis locais: uma Nuvem pública, um *cloudlet* e outro dispositivo móvel. Na Nuvem pública, os dispositivos móveis podem utilizar redes de telefonia celular (*e.g.*, 2G, 3G e 4G) e redes *Wi-Fi* para ter acesso a serviços e realizar o *offloading*, como mostrado na Figura 3 (a). O *cloudlet* foi proposto por Satyanarayanan *et al.* (2009), que diz respeito a máquinas disponíveis na mesma rede compartilhada com os usuários de aplicações móveis, com o principal intuito de aproximar os serviços de *offloading* dos usuários, conforme mostrado na Figura 3 (b). A vantagem encontrada nessa abordagem é que as redes locais, geralmente, possuem velocidades maiores e latências menores em relação aos servidores hospedados em infraestruturas mais remotas, e comumente acessados por redes celulares (COSTA *et al.*, 2015). Por fim, o *offloading* pode ser realizado em outro dispositivo móvel, nessa

abordagem os dispositivos móveis se conectam normalmente a uma rede *peer-to-peer* com o intuito de resolver uma tarefa em comum a todos com os outros dispositivos (FERNANDO *et al.*, 2013). Essa abordagem por ser melhor observada na Figura 3 (c).

Figura 3 – Locais de execução do *offloading*



(a) Nuvem pública (b) Cloudlet (c) Dispositivos Móveis

Fonte – Silva (2019)

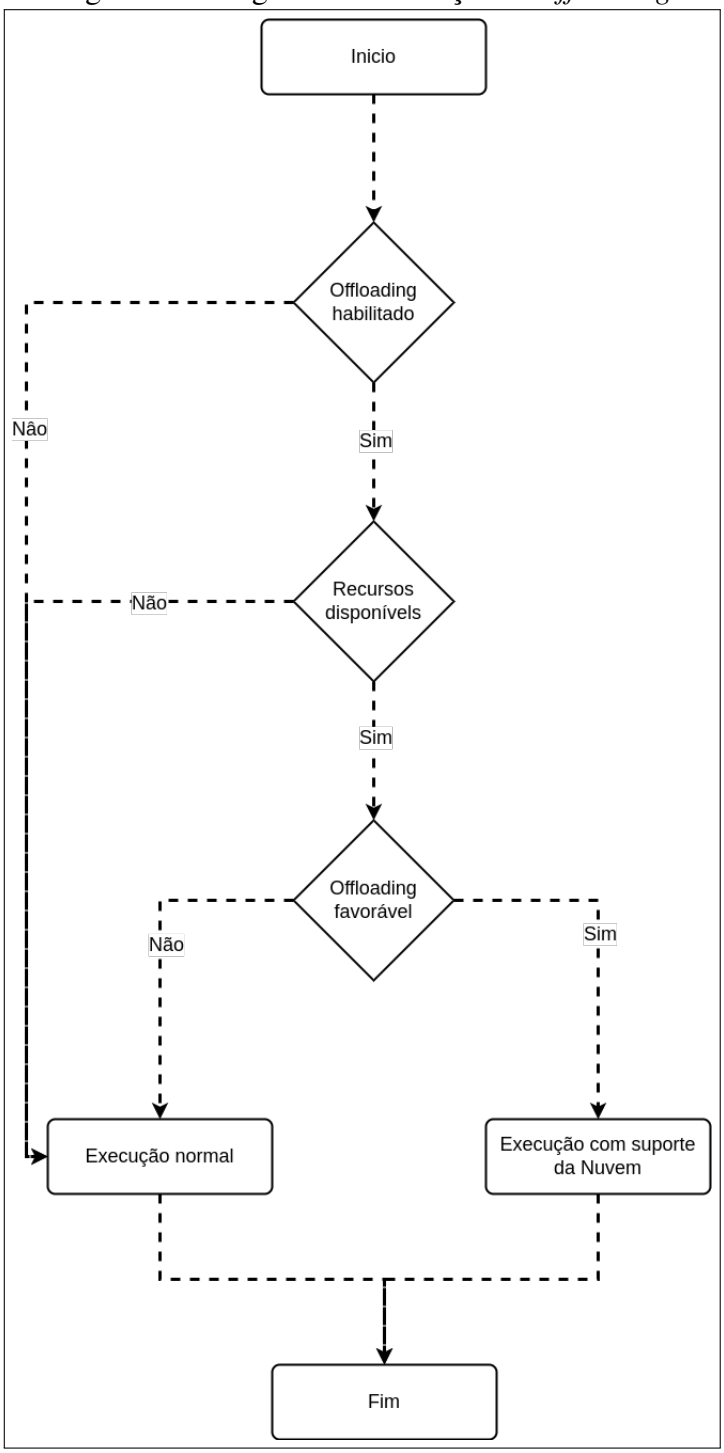
A decisão de realizar ou não o *offloading* deve ser baseada em critério que confirma se o processo realmente irá trazer benefícios notórios para o dispositivo móvel. Estes critérios podem ser bastante relativos e pode mudar dependendo do contexto da aplicação móvel. Para Somula e Sasikala (2018), a decisão é baseada em tempo de execução, utilizando informações em tempo real como a capacidade da conexão *Wi-Fi*. Já para Kharbanda *et al.* (2012), a decisão é baseada no quanto de consumo de bateria é proporcionado ao dispositivo móvel.

Khan Mazliza Othman *et al.* (2013) apresentam um diagrama de execução onde mostra o fluxo para a execução de *offloading*. O fluxo inicia com a execução da aplicação, onde em seguida é verificado a permissão de *offloading*. Caso o aplicativo esteja permitido para o *offloading*, a aplicação então verifica a conexão com os recursos da Nuvem e anota os recursos disponíveis. Após isso é decidido se o processo de *offloading* é favorável ou não. Se sim, o *offloading* é realizado com sucesso, caso não a aplicação refaz todos os passos novamente. A Figura 4 mostra a representação dos passos descritos acima.

2.4 Comunicação Sem Fio - Protocolos

No início do desenvolvimento de redes de computadores, todos os esforços eram destinados à evolução do hardware, tendo como principal objetivo a descentralização e substituição do então dominante “centro de computação”, que diz respeito a abordagem onde um único computador atende a todas as demandas de uma organização Tanenbaum (2011).

Figura 4 – Diagrama de execução do *offloading*



Fonte – Adaptado de Khan Mazliza Othman *et al.* (2013)

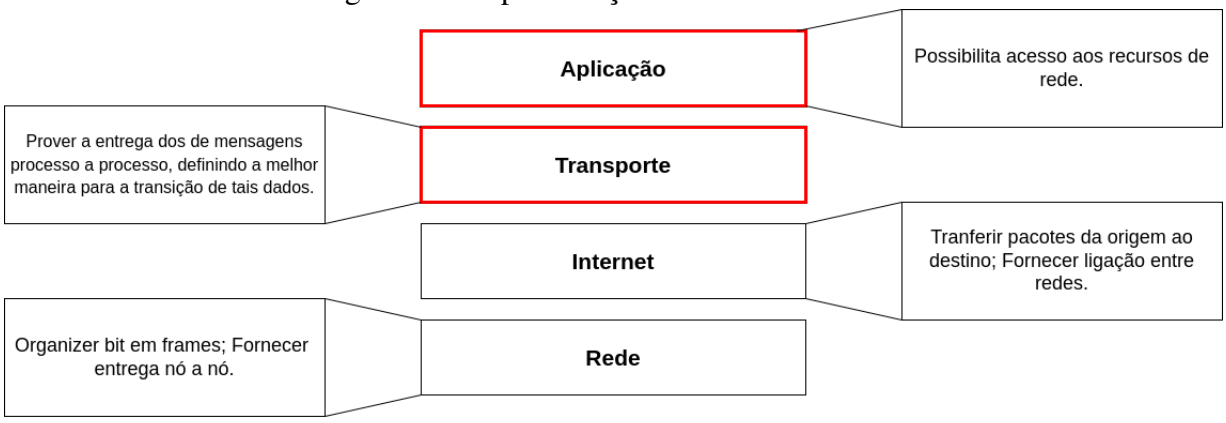
Após conseguir tal feito, os esforços então foram voltados para a estruturação de uma arquitetura de softwares de rede, com o intuito de que todos os computadores interconectados pudessem se comunicar, compartilhar informações, recursos e atribuir tarefas, utilizando camadas que se comunicariam por meio de protocolos (TANENBAUM, 2011).

Seguindo essa linha de raciocínio, a ISO criou, no ano de 1970, o padrão *Open Systems Interconnection* (OSI) que contém 7 (sete) camadas interligadas: **Física**, **Enlace**, **Rede**, **Transporte**, **Sessão**, **Apresentação** e **Aplicação** (MENDES, 2007). Cada uma delas é designada para um trabalho especifico e uma complementa a função da outra.

Além do modelo OSI, foi preciso criar um conjunto de protocolos para ser usado nesse modelo, de forma que padronizasse a comunicação entre diversas plataformas, possibilitando assim a comunicação entre as mesmas. Mishra (2017) explica que junto com a definição do padrão OSI, o *Departament of Defense* (DoD) estabeleceu o modelo de referência *TCP/IP* para esse fim.

Mishra (2017) cita que diferente do modelo OSI, a representação do modelo *TCP/IP* contém apenas 4 (quatro) camadas: **Rede, Internet, Transporte** e **Aplicação**, onde para cada uma camada, existe um protocolo diferente. A Figura 5 mostra uma representação detalhada do modelo *TCP/IP*, destacando as camadas de *Transporte* e *Aplicação*, as quais são o foco do presente trabalho.

Figura 5 – Representação do modelo *TCP/IP*



Fonte – Próprio autor

2.4.1 Transporte

Para Forouzan (2008), a camada de *Transporte* é responsável pela garantia de entrega das mensagens entre processos, onde o processo é um programa que está sendo executado em um *host*[3]. Sendo assim, o dever da camada de *Transporte* é garantir a entrega da mensagem ao destinatário, de forma intacta e na ordem correta do envio. Dentre as características e responsabilidades da camada de *Transporte* estão:

Endereçamento do ponto de acesso ao serviço - Geralmente, computadores executam mais de um serviço ou programa simultaneamente. Por essa razão, faz-se necessário um mecanismo que permita a entrega das mensagens para o processo específico onde a aplicação está executando. Desse modo, o protocolo da camada de *Transporte* deve disponibilizar um mecanismo que permita identificar os processos responsáveis pela execução da aplicação nos computadores remetente e destinatário;

Segmentação e remontagem - No processo de transporte da mensagem, ela é dividida em segmentos transmissíveis, e, para cada segmento, é atribuído um número de sequência. Desse modo, a camada de *Transporte* deve prover mecanismos que permita "remontar" a mensagem quando ela chega ao destino, além de identificar eventuais pacotes perdidos e meios alternativos de obtê-los;

Controle de conexão - Existem dois tipos: **Orientada à conexão**, onde antes da transmissão dos pacotes é estabelecido uma conexão entre a máquina remetente e a destinatário, de forma que a conexão só é encerrada quando todos os pacotes são entregues; e **Não orientada à conexão**, que trata cada segmento da mensagem como independente, entregando-as de forma independente ao destinatário;

Controle de fluxo - Caso no meio da transmissão perceba que a velocidade com que os dados são enviados pelo remetente está sendo maior que a velocidade de recepção do destinatário, ela impõe um mecanismo de controle de fluxo para evitar gargalos na comunicação; e

Controle de erros - A camada de *Transporte* se certifica de que todos os segmentos da mensagem serão entregues ao destinatário, sem que os dados sejam corrompidos, perdidos ou duplicados.

Existem diversos protocolos presentes na camada de *Transporte*, em que cada um apresenta uma abordagem diferente, assim como seu proposito e forma de funcionamento. Dito

[3] Uma máquina ou dispositivo que pode ser usada para executar programas e aplicativos

isso, alguns serão destacados a seguir.

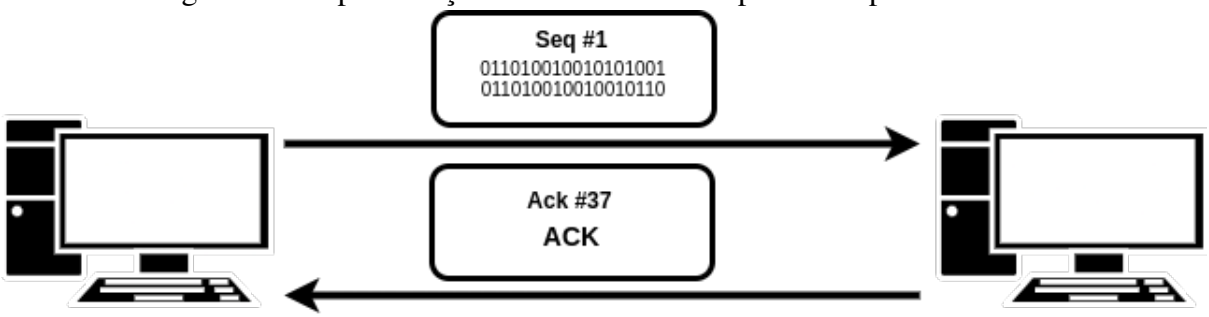
2.4.2 TCP

Como um dos principais protocolos da camada de *Transporte*, o TCP é um protocolo orientado a conexão, além de conter mecanismos que implementa uma entrega confiável dos dados, evitando perda de dados durante o envio.

Como dito anteriormente, o protocolo *TCP* é orientado a conexão, necessitando que uma conexão seja estabelecida antes do envio dos dados. Tal conexão utiliza um mecanismo chamado *Three Way Handshake*, descrito na *RFC 793*[4] do protocolo.

Para a entrega confiável, o protocolo adota duas medidas. A primeira, mostrada na Figura 6, o servidor envia uma confirmação com a *flag ACK*, sempre que receber o pacote enviado pelo cliente. Com isto, na segunda medida, apresentada na Figura 7, sempre que o cliente enviar um pacote, ele inicia um cronômetro interno, e caso este cronômetro acabe e o servidor não tenha enviado a mensagem de confirmação, o cliente entende que o pacote foi perdido e o reenvia.

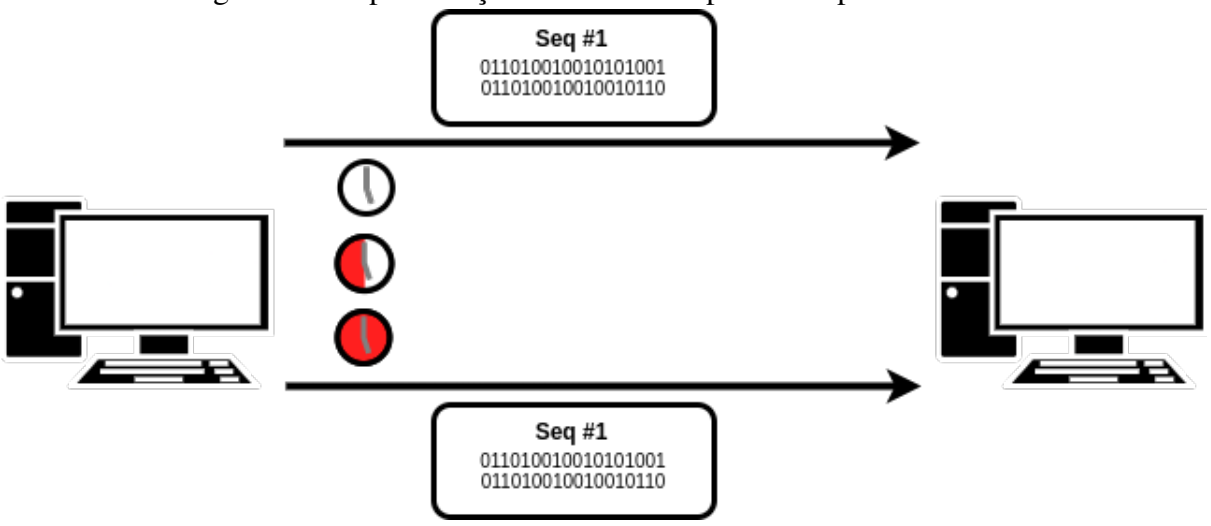
Figura 6 – Representação de recebimento de pacote no protocolo TCP



Fonte – Adaptado de Academy (2020)

[4] <https://www.ietf.org/rfc/rfc793.txt>

Figura 7 – Representação de reenvio de pacote no protocolo TCP



Fonte – Adaptado de Academy (2020)

O protocolo TCP é recomendado para casos onde a perda de informações não é tolerada, fazendo assim com que ele seja muito utilizado em servidor de envio de *e-mails*, mensageiria, e arquivos.

2.4.3 UDP

Como uma proposta de protocolo com características diferentes do TCP, o *User Datagram Protocol* (UDP) além de ser um protocolo não orientado a conexão, também não existem mecanismos que garantem a entrega de pacotes, fazendo com que seja um protocolo mais simples.

Descrita na *RFC 768*[5], a não orientação a conexão (ou *connectless*), faz com que o UDP, não exija uma conexão estabelecida antes do envio dos dados. O que acontece é que, na comunicação entre *hosts* utilizando o protocolo UDP, o servidor está sempre esperando por uma mensagem de algum cliente, e, por outro lado, o cliente apenas necessita saber o endereço *IP* e a porta utilizada por este servidor.

Além deste ponto, como dito anteriormente, o UDP não realiza o gerenciamento de perda de pacotes. Então, caso durante o envio de pacotes, algum deles não chegue ao servidor, o cliente não saberá, e nem retransmitirá o mesmo. A falta destes recursos faz com que o UDP seja um protocolo mais rápido, pois por não se preocupar em estabelecer uma conexão e fazer o reenvio dos pacotes, o protocolo se torna mais ágil. O uso do protocolo UDP está muito ligado a

[5] <https://www.ietf.org/rfc/rfc768.txt>

casos onde a perda de pacotes não é um grande problema (*e.g.*, *streaming* de video e de áudio).

2.4.4 RUDP

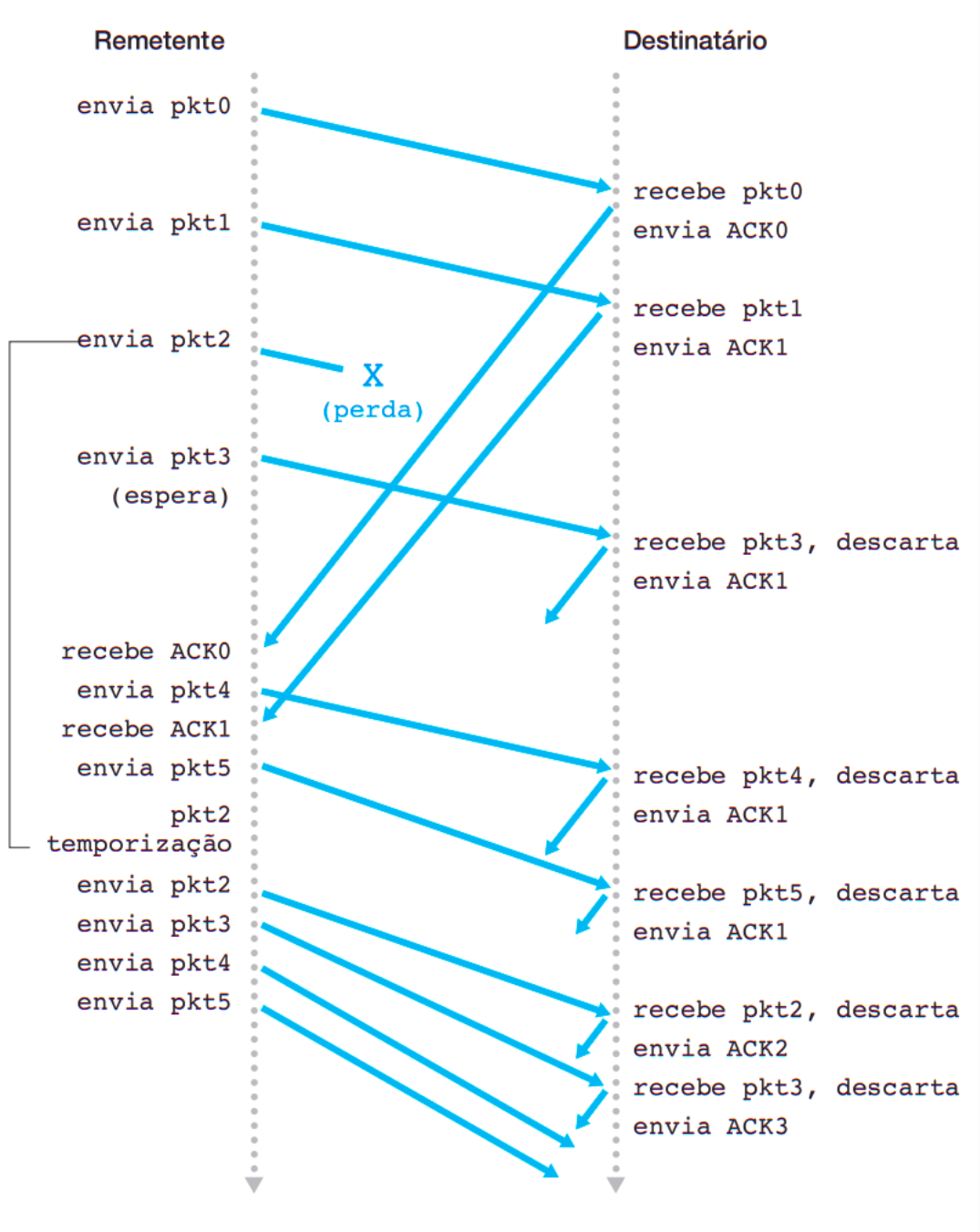
Como dito anteriormente, por mais que o protocolo UDP possua uma grande vantagem sobre o TCP em termos de rapidez no envio dos pacotes, um grande problema enfrentado por ele é a não confiabilidade na entrega dos mesmos, que, por outro lado, está presente no TCP. Pensando nisso, o protocolo RUDP apresentado em uma *RFC Draft*[6] é uma boa alternativa, pois combina o ponto positivo de ambos, ou seja, ele utiliza como base o protocolo UDP, garantindo uma maior rapidez na entrega dos pacotes, e adiciona uma camada que implementa a confiabilidade na entrega dos mesmos, fazendo assim um protocolo que promete uma entrega confiável dos dados, com maior rapidez.

Para tal feito, o protocolo RUDP utiliza um segundo protocolo de transferência de dados chamado SWP (no português, Protocolo de Janela Deslizante), focado na confiabilidade e segurança do envio. O SWP, segundo Kurose (2010), funciona em segmentos, o que quer dizer que o protocolo divide o dado a ser enviado em partes menores, de mesmo tamanho, e os enviam através da rede. Além da divisão em segmentos, é dado um número de sequência para cada pacote, para que o receptor, através do SWP, consiga manter a ordem dos pacotes, e descartar os duplicados, caso existam.

Kurose (2010) complementa que, o SWP permite com que o remetente envie múltiplos pacotes sem a necessidade de esperar por uma confirmação de recebimento dos pacotes um a um por parte do receptor. Porém, a quantidade de pacotes enviados ao mesmo tempo é determinado por um número máximo permitido, chamado *Window Size* (ou Tamanho da Janela). Para entender melhor como funciona o envio dos dados através do SWP, a Figura 8 demostra o mecanismo de envio do mesmo.

[6] <https://datatracker.ietf.org/doc/html/draft-ietf-sigtran-reliable-udp-00.txt>

Figura 8 – Representação de envio de pacote no protocolo SWP



Fonte – Kurose (2010)

No exemplo acima, *Window Size* é definido como 4. Por conta disso, no inicio é enviado os pacotes de 0 a 3, e por conta do tamanho da janela definida, o remetente deve esperar a confirmação de pelo menos 1 pacote para continuar o envio, esta confirmação é dada pelo envio de uma mensagem por parte do destinatário, denominada *ACK*, que acompanha o número do pacote recebido. E, para cada *ACK* recebido (*e.g.*, *ACK0* e *ACK1*), a janela desliza para frente para que o remetente possa enviar os próximos pacotes (*e.g.*, *pkt4* e *pkt5*, no exemplo acima).

Pode-se observar, que no lado do destinatário o pacote 2 é perdido, com isso, os pacotes 3, 4 e 5 foram descartados por estarem fora de ordem. Por fim, quando o temporizador do pacote 2 acaba e o remetente não tem recebido o *ACK* do mesmo, ele reenvia todos os pacotes da janela a partir do último *ACK* recebido (e.g., o *ACK1* da Figura 8).

Além da confiabilidade de entrega, o RUDP provê prevenção de erros através do uso do *Checksum*. O *Checksum* nada mais é que um mecanismo de verificação do dado para checar se um pacote foi corrompido ou não durante o trajeto.

O protocolo RUDP se apresenta como uma boa alternativa, que une pontos positivos do TCP e UDP. Por isto, pode ser usado tanto para casos onde a perda de informações não é tolerada, como para casos onde se necessita uma maior rapidez no envio dos dados.

2.4.5 Aplicação

A camada de *Aplicação* permite com que o usuário (*Isto é (i.e.)* humano ou *software*) tenha acesso à rede. Ela é responsável por prover uma interface ao usuário que possibilita suporte à serviços como *e-mail*, transferência e acesso a arquivos remotos e gerenciamento de dados em banco de dados distribuídos (FOROUZAN, 2008). Alguns exemplos de aplicações da camada de *Aplicação* são:

Terminal de rede virtual - Um terminal de rede virtual se diz respeito a um *software* que simula um terminal físico de um *host* remoto, fazendo com que outro *host* possa ter acesso remoto, e executar comandos como, por exemplo, fazer o *login* remotamente;

Transferência, acesso e gerenciamento de arquivos - Diz respeito a possibilidade de um usuário conseguir ter acesso remoto a arquivos de um *host*, podendo assim visualizar, alterar e mandar arquivos do *host* local para o *host* remoto e vice-versa;

Serviços de correio eletrônico - A aplicação prove uma base para armazenamento de *e-mails* bem como o seu encaminhamento; e

Serviços de diretório - Essa aplicação dá a possibilidade de acesso a banco de dados distribuídos e acesso às informações do mesmo.

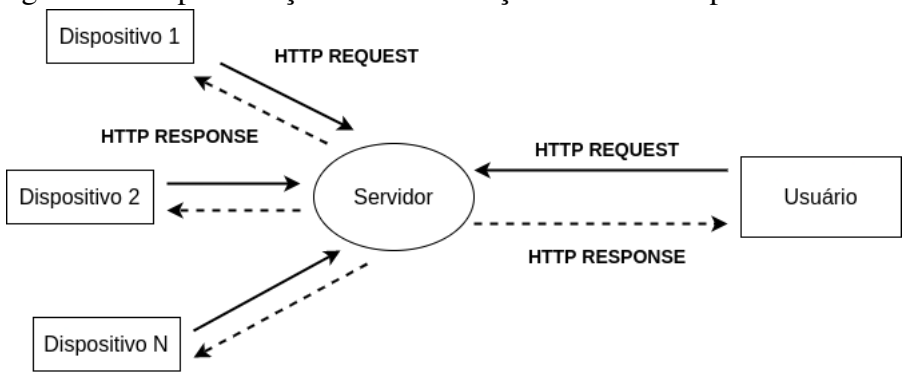
Assim como na camada de *Transporte*, existem diversos protocolos da camada de *Aplicação*, onde cada um apresenta uma abordagem diferente, assim como seu proposito. Dito isso, nas próximas serão descritos os dois protocolos de camada de Aplicação adotados nesse

trabalho: o HTTP e o MQTT.

2.4.6 HTTP

HTTP é um protocolo traducional na *Web*, criado pela W3C (*World Wide Web*) e usado para troca de dados estruturados. Trata-se de um protocolo que utiliza o modelo computacional **Cliente-Servidor**, onde o cliente (*e.g.* navegador *web*) requisita ao servidor recursos Web através de uma mensagem HTTP e o servidor os retorna em forma de dados estruturados, como uma página *HTML*[7]. Para mostrar melhor o funcionamento do protocolo HTTP, a Figura 9 traz um exemplo de comunicação padrão utilizando este protocolo.

Figura 9 – Representação de comunicação utilizando o protocolo HTTP



Fonte – Adaptado de Yokotani (2016a)

Pode-se ver que, como dito anteriormente, o protocolo HTTP utiliza o padrão **Cliente-Servidor**, onde na Figura 9 mostra a representação de vários dispositivos requisitando o servidor através de uma solicitação chamada *HTTP Request*, e resultando em um *HTTP Response*, que são feitas através do *TCP/IP*, ou seja, o HTTP é executado sobre o TCP, que traz consigo algumas características deste protocolo, como entrega confiável e orientação a conexão.

A comunicação utilizando o protocolo HTTP acontece da seguinte forma: O cliente se conecta ao servidor utilizando uma *Uniform Resource Identifier* (URI) que, geralmente, é uma *Uniform Resource Locator* (URL) contendo o endereço do servidor e um **caminho** (e.g., http://enderecoserividor**/foo/bar/**).

Após isto, uma conexão entre o cliente e o servidor é estabelecida, e uma ação é feita dependendo do método de solicitação que o usuário enviou. Existem alguns métodos disponíveis

[7] HyperText Markup Language, uma linguagem de marcação para criação de páginas *web*

no protocolo HTTP, onde cada uma deles contém um propósito. Dos métodos de solicitações presentes no HTTP, pode-se destacar:

GET - Método utilizado para requisitar recursos do servidor, como páginas *Web* ou dados estruturados como um objeto *JSON*;

POST - Método usado para transferir dados para o servidor, geralmente utilizado para solicitar o armazenamento de dados. Este método permite o envio de um corpo contendo um *payload*.

PUT - Similar ao método *POST*, este método também permite envio de um corpo para o servidor. Porém, o objetivo deste método é solicitar alteração de algum dado no servidor, e caso não exista nenhum dado semelhante ao solicitado pelo método, então este é armazenado.

DELETE - Método utilizado para remover um dado no servidor.

PATCH - Este método, semelhante ao método *PUT*, também solicita alteração em um dado no servidor. Porém, a alteração pode ser feita de forma parcial, sem necessitar alterar todo o dado de uma só vez.

OPTIONS - O servidor pode ser personalizado para aceitar somente determinados métodos. Dito isto, este método retorna para o cliente quais métodos o servidor está apto a responder.

O protocolo HTTP é bastante utilizado para comunicação entre dispositivos através da internet, além de ser comumente caracterizado como o protocolo padrão do modelo *TCP/IP*. Por isto, pode ser utilizado para muitos tipos de aplicações, sejam elas em hospedagens de *sites*, ou até mesmo comunicação entre dispositivos *Internet of Things* (IoT).

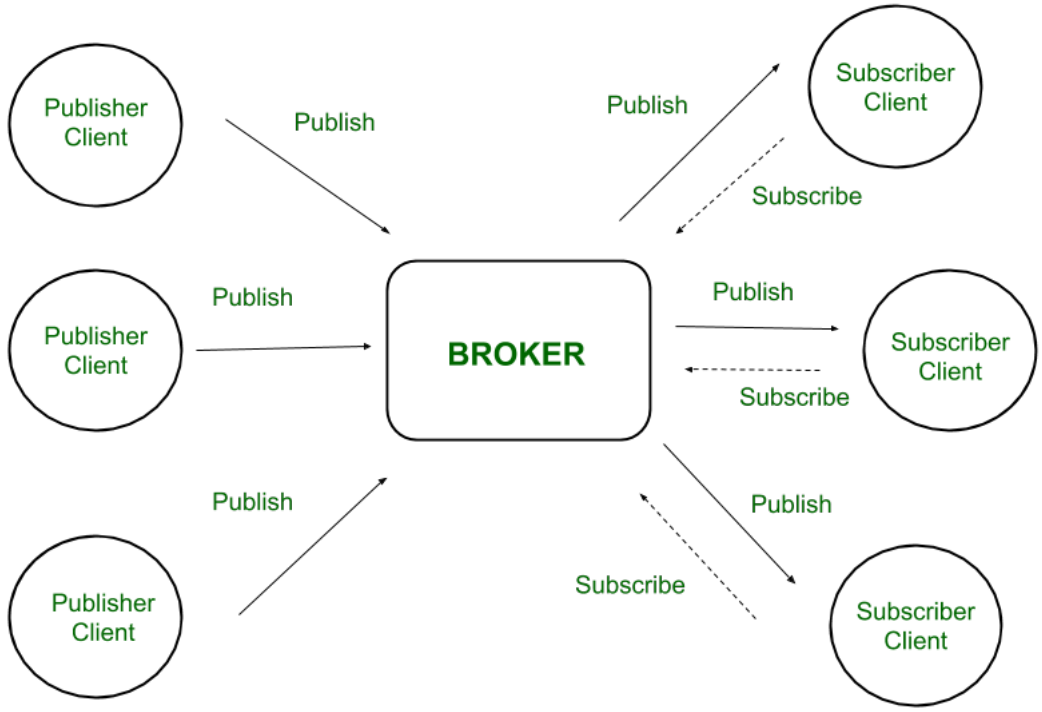
2.4.7 MQTT

o MQTT, criado e mantido pela *IBM*, é um protocolo voltado para comunicação entre dispositivos com pouca capacidade de processamento, e por conta disto, o MQTT é um protocolo muito usado no ambiente de IoT (YOKOTANI, 2016b). Outra característica interessante sobre este protocolo, é o uso de uma arquitetura chamada *Publisher-Subscriber*, que funciona de forma um pouco diferente da *Cliente-Servidor*.

Para exemplificar melhor como a arquitetura *Publisher-Subscriber* funciona, a Figura 10 descreve um ambiente com vários dispositivos se comunicando através do protocolo MQTT.

Pode-se notar que, diferente da arquitetura *Cliente-Servidor*, a *Publisher-Subscriber*

Figura 10 – Representação da arquitetura *Publisher-Subscriber* presente no protocolo MQTT



Fonte – Geeks (2021)

possui uma entidade central chamada *Broker*, responsável por armazenar e distribuir as mensagens enviadas pelo *Publisher*. O *Publisher* é a entidade responsável por enviar mensagens para serem consumidas posteriormente. É importante destacar que a mensagem enviada contém um formato padrão, contendo a seguinte forma: o **Tópico**, sendo um cabeçalho que indica o ponto para qual o *Broker* deve distribuir a mensagem; e o **Corpo**, que armazena o conteúdo da mensagem em si.

O *Subscriber*, por sua vez, é a entidade que recebe as mensagens enviadas pelo *Publisher*. Para receber tais mensagens, o *Subscriber* deve fazer a assinatura de um tópico junto ao *Broker*, para então receber todas as mensagens enviadas para o tópico que este está assinado. Na comunicação entre dispositivos, um dispositivo pode ser tanto um *Publisher*, como um *Subscriber*, podendo também se inscrever e receber mensagens de mais de um tópico. Como dito anteriormente, pela sua característica de baixo consumo de *hardware*, o protocolo MQTT é muito utilizado no meio IoT, trazendo vantagens em termo suporte a dispositivos com pouco processamento, e aplicações que fogem da arquitetura *Cliente-Servidor*.

No MQTT, os tópicos lembram o conceito de URI, onde existem níveis separados

por barra. Com isso, os dispositivos podem criar tópicos com diversos níveis, com analogia semelhante a uma URL de um *site*. A Tabela 1 expressa um exemplo hipotético de um conjunto de salas com sensores de temperatura, onde cada sala é identificada por um *ID* e contém diversos sensores, também identificados por *IDs*.

Tabela 1 – Representação de uso de tópicos no protocolo MQTT

Identificador	Estrutura	Exemplo
Dinâmico	sala/IDSala/sensor/IDSensor/temperatura	sala/1/sensor/5/temperatura
+	sala/IDSala ou +/sensor/IDSensor ou +/temperatura	sala/1/sensor/+/temperatura
#	sala ou #/IDSala ou #/sensor ou #/IDSensor ou #/temperatura	sala/1/sensor/#
$	$SYS/broker/clientes/total	$SYS/broker/clientes/total

Fonte – Próprio autor

Uso dinâmico - Os tópicos no MQTT podem conter uma forma dinâmica, onde se pode passar parâmetros para obter informações personalizadas. No exemplo da Tabela 1 podemos observar que no mesmo tópico, os dispositivos podem solicitar informações de diferentes salas, e diferentes sensores apenas mudando os parâmetros de *ID*.;

+ - O uso do + é agregado a possibilidade de poder agrupar resultados de diferentes parâmetros do mesmo tópico. No exemplo da Tabela 1, utilizando o +, pode-se obter informações de todos os sensores presentes na sala 1, de forma agrupada e sem necessidade de subscrição em todos os sensores individualmente.

- Semelhante ao uso do +, o # possibilita obter informações agrupadas com uma única subscrição, a diferença é que enquanto o + somente pode ser usado para substituir um parâmetro dinâmico, o # pode ser substituído em qualquer nível do tópico, e o dispositivo receberá informações de qualquer nível existente abaixo #.

**$** - Considerado como um tipo especial, o *$* é utilizado somente de modo interno dentro do *broker* e são utilizadas somente para disponibilizar informações relevantes sobre o mesmo.

2.5 Considerações finais

Nesse Capítulo, foram abordados os temas da proposta desse trabalho, como a *Computação Móvel*, a *Computação em Nuvem*, a *Mobile Cloud Computing*, bem como a técnica de *offloading*. Além disso, foi apresentado sobre a comunicação, camadas, bem como exemplos de protocolos utilizados nas mesmas. No próximo capítulo, será abordado trabalhos relacionados com o trabalho proposto.

3 TRABALHOS RELACIONADOS

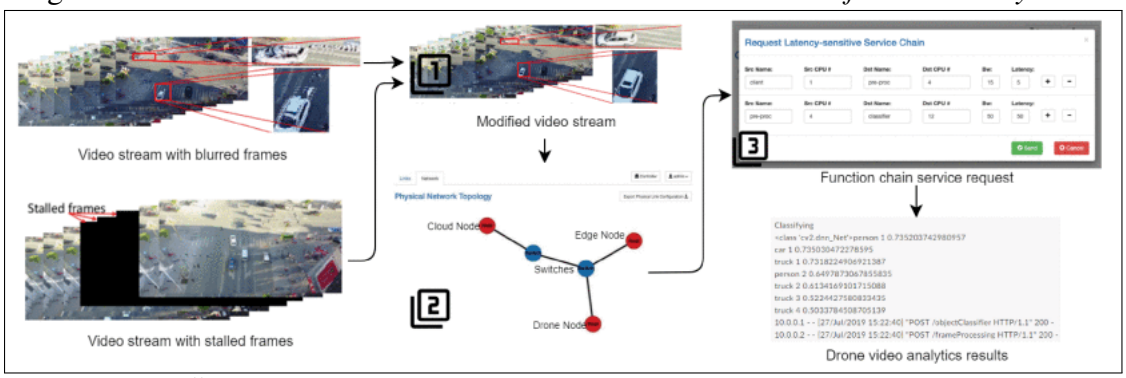
Com o objetivo de apresentar trabalhos relacionados à presente proposta, os trabalhos abordados neste capítulo, retirados dos repositórios *Institute of Electrical and Electronics Engineers* (IEEE), envolvem infraestruturas de software que proveem *offloading* entre dispositivos móveis.

3.1 DyCOCo: A Dynamic Computation Offloading and Control Framework for Drone Video Analytics

Qu Songjie Wang (2019) propôs um trabalho intitulado *DyCOCo*, que é um *framework* de controle e *offloading* dinâmico para análises de vídeo de *Drone*, cuja proposta é apresentar um *framework* para tomada de decisão dinâmica no processo de *offloading* entre o UAV e o *Ground Control Station* (GCS) utilizando uma arquitetura chamada *Function-Centric Computating* (FCC)[1] para minimizar problemas como distorção e desfoque dos *frames*, assim melhorando a experiência do usuário acerca do processo inteligente de análise do video transmitido.

Para um transmissão de vídeo em alta resolução e de forma que minimize as deficiências, esse *framework* toma decisões dinâmicas sobre a alteração dos protocolos de rede (i.e., TCP / HTTP, UDP / RTP, QUIC) para a transmissão de dados, bem como a resolução do video transmitido. Na Figura 11, pode-se ver uma ilustração dos processos tomado pelo *framework* para analisar video capturado por UAV.

Figura 11 – Passos da análise do video de um UAV utilizando o *framework DyCOCo*



Fonte – Qu Songjie Wang (2019)

[1] Aplicativos de decomposição em funções de microsserviço que podem ser implantadas em recursos de borda em conjunto com as principais plataformas de nuvem

No fluxo mostrado acima os passos são: **(1)** o *framework* detecta falhas na captura do video (i.e., distorção, desfoque) e o modulo de controle para modificar o *streaming* de video; **(2)** Após isso, é feita a configuração da topologia de rede física, configurando assim os pontos onde poderão alocar os recursos (e.g., Nuvem, *Edge*); e **(3)** é feito uma solicitação de alocação de recursos em tempo real, de acordo com as politicas de controle.

Com o intuito de realizar a demostração do *framework* na prática, foi utilizado um cenário contendo vídeos da base de dados *VisDrone*[2] para simular a captura de imagens de um UAV em 4 (quatro) diferentes resoluções 1080p, 720p, 480p e 360p, com o objetivo de identificar objetos em movimento (e.g., carros, caminhões e pedestres). A Figura 12(a) apresenta o pipeline de execução do *framework*.

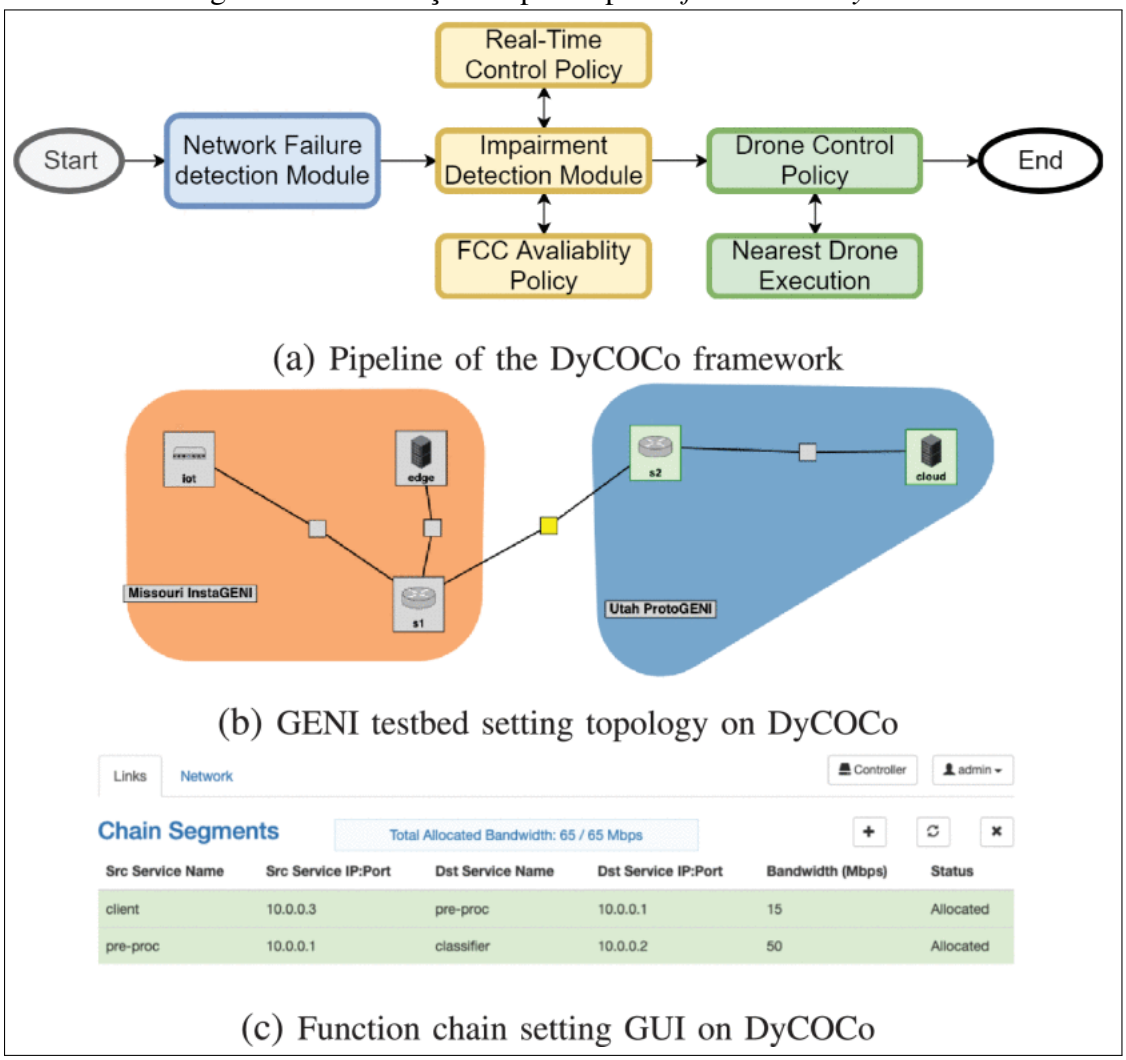
Para simular a topologia de rede (i.e., UAV, GCS e Nuvem), o autor utilizou uma infraestrutura de sistemas distribuídos voltada para ensino e pesquisa chamada *GENI*[3], mais especificamente o *Missouri InstaGENI*[4] como servidor de borda, e o *Utah ProtoGENI*[5] como servidor em Nuvem, ambos conectados por um *OpenFlow vSwitch*[6], como mostrado na Figura 12(b).

Por fim, utilizando o *Graphic User Interface* (GUI) do *framework*, o autor colocou os parâmetros para cada função (e.g., latência e largura de banda requeridas) na aplicação de análise de vídeo. Então, o autor demostrou a eficiência do *framework* no poder de decisão para identificar qual a topologia de rede era mais adequada no momento para a alocação de recursos e processo de *offloading* utilizando como abordagem:(1) a mudança do controle da câmera para obter vídeos em alta resolução ou (2) a mudança de protocolos de rede sem a perca de performance de captura. Os resultados da demostração do *framework* podem ser vistos na Figura 12(c).

O trabalho citado acima é de extrema relevância para a base deste trabalho aqui dissertado. Pois, assim como a nossa proposta, ele apresenta um *framework* que em sua parte decide quais protocolos utilizar para uma comunicação eficiente e com o proposito de minimizar

[2] http://aiskyeye.com/
[3] https://www.geni.net/
[4] https://groups.geni.net/geni/wiki/GeniAggregate/MissouriInstaGENI
[5] https://www.flux.utah.edu/project/pgeni
[6] Uma implementação *open-source* voltado para sistemas distribuidos com o intuito de controlar o direcionamento e distribuição para vário servidores, além de possuir suporte a diversos protocolos de gerenciamento padrão

Figura 12 – Ilustração do protótipo do *framework DyCOCo*



Fonte – Qu Songjie Wang (2019)

os impactos negativos na experiência do usuário. Como ponto negativo do trabalho, foi sentido a falta de uma conclusão mais detalhada, pois uma vez que o trabalho acima tem a proposta de mostrar uma melhor maneira de executar uma tarefa, foi sentido falta de comparativos de resultado nesse trabalho.

3.2 *A Framework of Mobile cloudlet Centers based on the Use of Mobile Devices as cloudlets*

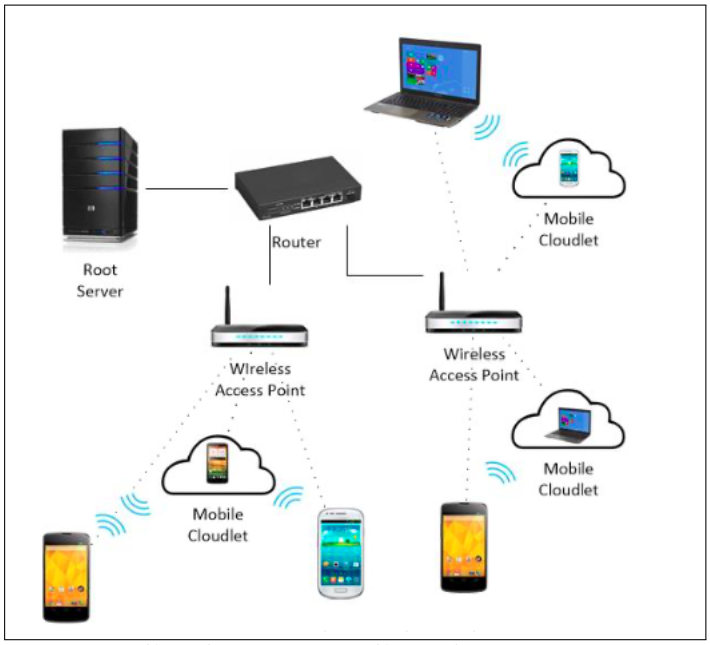
Serviços baseado em Nuvem são cada vez mais usados nos dias atuais, porém o uso dessa abordagem geralmente envolve problemas de alta latência na rede. Para resolver esse problema, foi implantado a ideia de "nuvens locais", denominadas de *cloudlets*, que segundo Satyanarayanan (2009), pode ser definida como "nuvens menores" que geralmente atendem aos usuários próximos e se destacam na transferência de conteúdo e tarefas de dispositivos móveis.

Os *cloudlets* geralmente são máquinas *desktop* poderosas funcionando constantemente, que provê serviços e softwares para usuários próximos.

Artail Karim Frenn e Artail (2015) apresenta um trabalho cuja proposta é implantar a ideia de *cloudlets* utilizando dispositivos móveis, pois segundo os autores os dispositivos móveis por mais que tenham uma capacidade inferior de recursos comparado a uma máquina *desktop*, acaba sendo um meio mais onipresente e acessível.

Para fazer tal feito, Artail Karim Frenn e Artail (2015) usa uma arquitetura de *cloudlets* representado pela Figura 13, que pode ser dividido em duas principais partes. Os *cloudlets* podem ser qualquer dispositivo que deseja compartilhar os seus recursos, como *smartphones* e *laptops*. E o *root server*, que é responsável por gerenciar todos os *cloudlets*, bem como descobrir novos dispositivos que desejam se tornar *cloudlets*.

Figura 13 – *General Network Topology*



Fonte – Artail Karim Frenn e Artail (2015)

O funcionamento dessa arquitetura acontece da seguinte forma, quando um dispositivo que está na mesma rede que do *Root server* deseja disponibilizar os seus recursos para que outros dispositivos façam o *offloading*, esse dispositivo deve anunciar seu interesse e informações, como quais recursos ele deseja disponibilizar e sua capacidade de bateria. Então, o *Root server* o registra como um *cloudlet* e atribui para ele um identificador. Por outro lado, quando um dispositivo deseja utilizar algum recurso dos *cloudlets*, o dispositivo envia um pedido para o

Root server e o mesmo se encarrega de identificar qual o melhor *cloudlet* para disponibilizar seus recursos aquele dispositivo.

Os autores usaram uma plataforma chamada CloudSim para realizar os experimentos, segundo (CALHEIROS R. RANJAN *et al.*, 2011), é um software amplamente utilizado para modelagem e simulação de sistemas de computação em nuvem. Inclui interfaces para máquinas virtuais, memória, armazenamento e largura de banda, bem como a capacidade de monitorar o estado do sistema e provisionamento de *hosts* para máquinas virtuais. Como ponto negativo, a utilização do *Root server* caracteriza um ponto único de falha, e caso venha a acontecer algo com esse dispositivo, pode comprometer toda a rede de *cloudlets* de uma só vez.

3.3 *Choice of Effective Messaging Protocols for IoT Systems: MQTT, CoAP, AMQP and HTTP*

Naik (2017) propõe um estudo entre alguns protocolos utilizados no ambiente de IoT (i.e., *MQTT, CoAP, AMQP e HTTP*), utilizando alguns critérios e apontando prós e contras de cada protocolo, a fim de ajudar o usuário a escolher o protocolo de uma forma mais clara, alinhando o intuito do uso do protocolo, com a objetividade do mesmo. Para tal feito, Naik (2017) dividiu a sua análise em duas partes principais. Na primeira parte, é possível entender diferenças entre os protocolos em questão que pode ajudar na distinção e escolha de um melhor protocolo para cada caso. Como, por exemplo, o tamanho da mensagem, que a depender da solução que será feita, está pode necessitar de um tamanho mínimo maior que 256 MB, o que impossibilitaria o uso do protocolo MQTT.

Na segunda parte, o autor realizou uma análise relativa dos protocolos, a fim de compará-los em relação aos seus prós e contras. Para isto, Naik (2017) os adicionou em uma espécie de gráfico, com dois critérios e uma medição qualitativa que vai de *pouco* a **muito**.

Foi possível observar que o protocolo HTTP suporta maior tamanho de mensagem, porém isso acarreta um maior *overhead* dos dados, além disso, pode-se ver que este protocolo também contém um maior consumo de energia, maior consumo de *hardware*, maior latência, e maior uso de banda. Enquanto, por outro lado, por mais que o protocolo *CoAP* contenha o menor tamanho de mensagem, ele apresenta menor *overhead*, além de menor consumo de energia, menor consumo de *hardware*, menor consumo latência, e menor consumo de banda.

Pode-se ver que além das limitações citadas acima, o protocolo HTTP apresenta também uma menor confiabilidade em termos de qualidade de serviço, e um baixo uso no ambiente de IoT. Onde, por outro lado, o protocolo MQTT se mostra muito usado em ambientes IoT, e com muita confiabilidade.

Embora o autor tenha realizado uma ótima análise dos protocolos aqui citados, foi percebido um ponto em específico que o presente trabalho não analisou a fundo. O próprio autor confirma que as fontes utilizadas como base para gerar a segunda etapa da análise, foi baseada em outra literaturas e em ambientes com componentes estáticos, ou seja, não sei foi realizado testes práticos, ponto que poderia melhorar a análise e trazer maior confiabilidade das informações prestadas. Dito isto, podemos concluir que o trabalho contém informações relevantes para o presente trabalho, que podem agregar e ajudar a entender todo o processo do mesmo, mas que os resultados obtidos pelo autor, poderiam ter consigo testes reais utilizando os protocolos.

3.4 *Comparison with HTTP and MQTT on Required Network Resources for IoT*

Yokotani (2016a) traz uma discussão sobre o protocolo MQTT como um candidato para uso em comunicação entre dispositivos de IoT, bem como uma comparação com o protocolo HTTP, bastante utilizado em comunicação entre dispositivos. Além disto, o autor também apresenta uma melhoria no protocolo MQTT, afim de melhorar sua performance.

Para isto, o autor explica detalhes de como funciona a comunicação em ambos os protocolos, adicionando imagens e representações do fluxo de comunicação. Em especial para o MQTT, o autor explica um pouco mais detalhado o funcionamento da arquitetura *Publisher-Subscriber*. Para fazer o estudo comparativo, Yokotani (2016a) utiliza de duas métricas. Na primeira, o autor faz a comparação entre consumo de banda/*overhead* e quantidade de dispositivos interagindo. E na segunda, o autor mede o consumo de banda/*overhead* tamanho do *payload* enviado.

Para ambos os testes, foi percebido que o HTTP teve um maior consumo de banda e *overhead* em comparação com o MQTT. Porém, foi percebido também que o protocolo HTTP começa a ter menos *overhead* que o MQTT, quando o mesmo começa a ter tópicos com mais de 680 *bytes*.

Com isto, o autor propôs uma melhoria ao MQTT que promove uma compressão dos tópicos afim de diminuir o seu tamanho e resolver parcialmente o problema de *overhead*. Para tal melhoria, Yokotani (2016a) propôs o tópico, uma vez que enviado para o servidor, fosse armazenado e associado a um identificador, que por sua vez seria enviado para os dispositivos, e assim sempre que um dispositivo precisar enviar aquele tópico novamente, ao invés de enviar o tópico em si, o dispositivo enviaria o identificador e o servidor ficaria responsável por associar o identificador ao tópico em questão.

O autor trouxe em seu trabalho uma análise muito importante a cerca do problema de comunicação entre dispositivos IoT, colocando a teste protocolos que são usados neste meio, e ainda propondo melhorias para um dos mesmos (i.e., MQTT). Contudo, sentiu-se falta de um teste prático da melhoria levantada pelo autor, que poderia confirmar a eficacia da mesma. Dito isto, foi concluído que este trabalho pode contribui no ponto de vista de comparação entre protocolos, uma vez que o autor realizou uma análise com diferentes pontos de vista e utilizando protocolos relevante para a proposta final deste trabalho. Porém a melhoria posposta pelo autor, poderia ter acompanhado testes práticos para comprovar sua eficácia.

3.5 Comparativo Entre os Trabalhos

Para o comparativo entre os trabalhos relacionados, foram levantadas características relevantes desses trabalhos em relação a pesquisa proposta. As características consideradas para a análise são as seguintes:

- **Protocolos de transporte:** essa característica diz respeito a informar se foram utilizados protocolos da camada de *Transporte*, e se sim, quais.
- **Protocolos de aplicação:** refere-se a informar se também foram utilizados protocolos da camada de *Aplicação*, e caso positivo, quais.
- **Número de dispositivos:** essa característica refere-se se quantidade de dispositivos utilizados na solução.
- **Tipo de ambiente:** essa característica diz respeito a que tipo de ambiente foi utilizado na solução (e.g., real, virtual).
- **Plataforma:** essa característica diz respeito a plataforma utilizada na solução (e.g., *Cloudlet*, D2D, Hibrida).

A Tabela 2 detalha o comparativo entre os trabalhos relacionados e as características elencadas.

Tabela 2 – Comparativo Entre os Trabalhos Relacionados

Trabalho	Protocolos de transporte	Protocolos de aplicação	Número de dispositivos	Tipo de ambiente	Plataforma
Qu Songjie Wang	TCP, UDP	HTTP, *RTP*, QUIC	2	Real	Nuvem, *Egde*
Artail et al.	Não especificado	Não especificado	Não especificado	Simulado	*Cloudlet*
Naik	Não especificado	MQTT, *CoAP*,*AMQP* HTTP	Não especificado	Nenhum	IoT
Yokotani	Não especificado	MQTT, HTTP	10, 100, 1000	Simulado	IoT
Naik	Não especificado	MQTT, *CoAP*,*AMQP* HTTP	Não especificado	Nenhum	IoT
Yokotani	Não especificado	MQTT, HTTP	10, 100, 1000	Simulado	IoT
Presente Trabalho	**TCP, RUDP**	**MQTT, HTTP**	**2**	**Real**	**D2D**

Fonte – Próprio autor

3.6 Considerações Finais

Esse Capítulo apresentou os trabalhos relacionados, que envolvem infraestruturas para prover um processo de *offloading* entre dispositivos móveis utilizando diferentes abordagens e utilizando a abordagem de escolha de protocolos para melhorar o desempenho da tarefa. A seguir, será apresentado a proposta desse trabalho.

4 PROPOSTA

Nos Capítulos anteriores foram mostrados conceitos introdutórios importantes para a compreensão desse trabalho, desde fundamentos acerca da Computação Móvel e Computação em Nuvem, essenciais para entendimento do ambiente onde esse trabalho se encaixa, até conceitos sobre MCC, *Offloading* e Comunicação sem Fio, com foco especial nas camadas de *Aplicação* e *Transporte* do padrão TCP/IP e em alguns dos protocolos que as representam. Além disso, foram apresentados trabalhos relacionados a área de MCC no processo de *offloading* computacional, mas que apesar de proporem a utilização de protocolos fora do convencional (e.g., MQTT e QUIC), não se realizou uma comparação entre os protocolos utilizados, ou justificativa de o porquê usá-los.

Diante disto, é importante entender se existe uma diferença de desempenho no processo de *offloading* entre dispositivos móveis, a depender do protocolo escolhido para a comunicação no processo. Assim, o presente trabalho consiste no desenvolvimento de um módulo expansível de adição de protocolos de comunicação e uma comparação de desempenho do *offloading* computacional quando realizado através dos protocolos TCP e RUDP da Camada de *Transporte* e HTTP e MQTT da Camada de *Aplicação*, todos componentes da pilha TCP/IP, em ambiente de MCC. As próximas seções apresentam a metodologia e arquitetura utilizada pelo presente trabalho.

4.1 Metodologia

Inicialmente, foi realizado um levantamento sobre os principais protocolos de cada camada e, empiricamente, alguns deles foram selecionados para compor a proposta do trabalho. Os testes empíricos consistiram em três principais etapas.

Como primeira etapa, após realizar um apanhado de todos os protocolos de cada camada escolhida, foi realizado uma pesquisa sobre compatibilidade de cada um deles em ambientes *mobile*, mais especificamente na plataforma *Android* (plataforma base utilizada nessa pesquisa).

Já na segunda etapa, dos protocolos filtrados na etapa anterior, verificou-se a presença de alguma biblioteca, nativa ou de terceiros, que facilita a implementação das mesmas, em

específico, bibliotecas escritas nas linguagens *Java* ou *Kotlin*. Com o intuito de maximizar os resultados, buscou-se, para cada protocolo, pelo menos uma biblioteca e aqueles que não tinham nenhuma biblioteca disponível, foram descartados.

Na terceira etapa, analisaram-se as bibliotecas encontradas na etapa acima, com o intuito de verificar se as mesma implementam o protocolo tal qual a *RFC*, de forma funcional, e se seria viável adotá-las no presente trabalho. Para isto, foram realizados provas de conceitos para cada protocolo, utilizando cada biblioteca encontrada, e escolhido as com melhores resultados em termos de simplicidade de implementação e número de colaborações no repositório do Github[1].

A quarta etapa consistiu na modelagem da solução desenvolvida, definindo a arquitetura e interfaces do módulo de comunicação expansível dos protocolos para a realização do *offloading*.

A quinta etapa consistiu na modelagem e desenvolvimento de uma aplicação móvel que funciona como um *benchmarking* para avaliar o poder de processamento das aplicações.

A sexta etapa consistiu na implementação dos protocolos escolhidos e disponibilização ao usuário, por meio da aplicação, onde o mesmo poderá escolher qual protocolo utilizar para realizar o processo de comunicação.

Por fim, foi realizado um conjunto de testes de desempenho, a fim de concluir qual o real impacto da escolha do protocolo, e entender como isso pode melhorar ou piorar a operação de *offloading* entre dispositivos móveis a depender do protocolo utilizado.

4.2 Protocolos Escolhidos

A Tabela 3 resume os protocolos escolhidos, bem como as bibliotecas escolhidas para cada um. Para a camada de *Aplicação* foram escolhidos os protocolos HTTP e MQTT. O HTTP foi escolhido por ser o principal protocolo da camada no modelo *TCP/IP*. O protocolo MQTT foi escolhido porque ele apresenta um baixo consumo de energia, de largura de banda e de recursos de computacionais durante o seu funcionamento, características compatíveis com as necessidades dos dispositivos móveis.

Para a camada de *Transporte* foram selecionados os protocolos TCP e RUDP. O

[1] https://github.com/

TCP foi escolhido pelos benefícios que ele proporciona as aplicações que o utilizam: garantia de entrega do pacote, controle de fluxo e tratamento de erro durante o transporte dos dados. Além disso, ele também foi escolhido porque, assim como o HTTP, ele é comumente apontado com o protocolo de referência dessa camada. Já RUDP foi escolhido, pois, segundo Thammadi (2009), se trata de um protocolo que apresenta bom desempenho na entrega de pacotes e gera baixo *overhead* na rede. Assim, espera-se que sua adoção proporcione ao *offloading* um desempenho semelhante ao do protocolo TCP, mas economizando largura de banda da rede sem fio.

Tabela 3 – Protocolos escolhidos para o trabalho

Nome	Camada	Biblioteca	Referência
HTTP	Aplicação	*HttpURLConnection*	https://docs.oracle.com/javase/8/docs/api/java/net/HttpURLConnection.html.
MQTT	Aplicação	*Eclipse Paho Java MQTT* e *Moquette*	https://github.com/eclipse/paho.mqtt.java https://github.com/andsel/moquette
RUDP	Transporte	*RUDP*	https://github.com/GermanCoding/RUDP
TCP	Transporte	*HttpURLConnection*	https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html

Fonte – Próprio autor

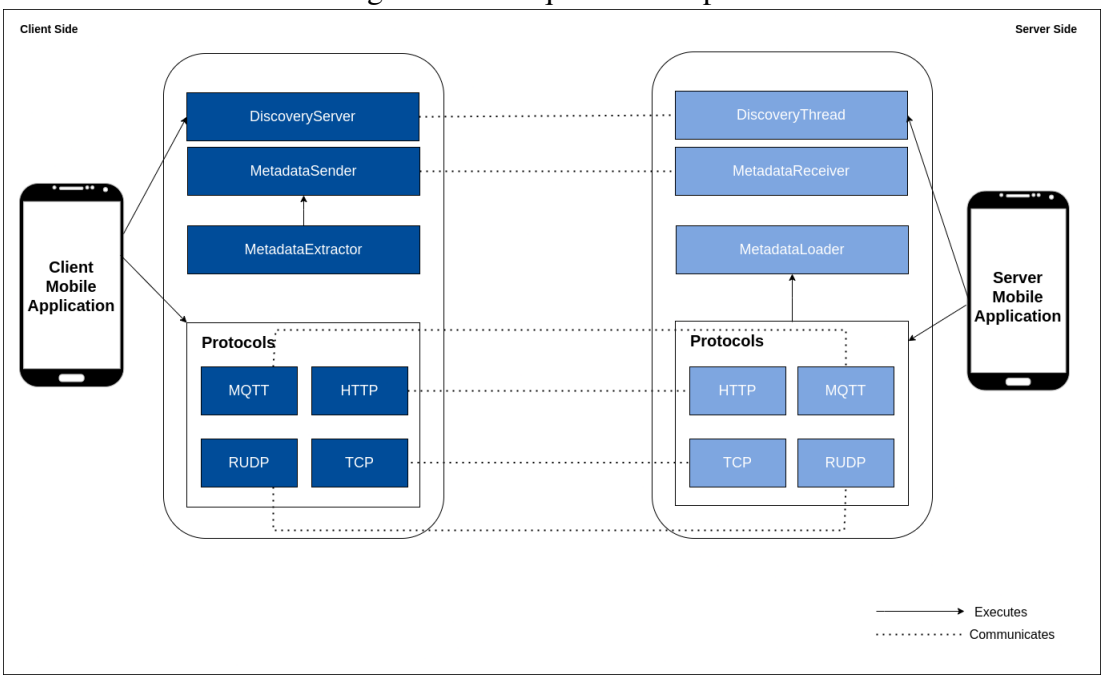
Tendo em mente todo o processo de escolha e seleção dos procolos e suas bibliotecas, a seguir, será mostrado de forma mais detalhada a arquitetura da proposta, bem como cada parte que a compõe.

4.3 Arquitetura

Para alcançar o objetivo da presente pesquisa, foi-se necessário o desenvolvimento de um módulo de comunicação capaz de interagir diretamente com protocolos das camadas de *Aplicação* e *Transporte*. Esse módulo abstrai a interação entre o cliente e o servidor durante o *offloading* computacional. O módulo foi construído usando recursos disponíveis na linguagem Java e na plataforma Android. Inicialmente, o módulo foi implementado para interagir com os protocolos presentes na Tabela 3, através de suas respectivas bibliotecas. A Figura 14 mostra a arquitetura do módulo desenvolvido neste trabalho.

A implementação se divide em quatro serviços principais implementados em ambos os lados (*i.e., cliente e servidor*), que se comunicam entre si para possibilitar o processo de

Figura 14 – Arquitetura Proposta



Fonte – Próprio autor

offloading computacional. A seguir, uma breve explicação sobre cada serviço, e como eles funcionam internamente.

4.3.1 Serviço de Descoberta

No lado do cliente, o módulo chamado *DiscoveryServer* possibilita a busca na rede por um servidor apto a atender ao *offloading* realizado pelo cliente. Para isso, o *DiscoveryServer*, dispara, em *multicast*, uma mensagem pré-definida a todas as máquinas disponíveis na rede, e aguarda por uma resposta vinda de algum servidor disponível. Neste caso, como o cliente ainda não escolheu o protocolo a ser utilizado durante o *offloading*, optou-se pelo TCP para intermediar essa etapa, pois, como dito na Seção 4.1, ele é apontado como um protocolo de referência do modelo *TCP/IP*.

Já na parte do servidor, há um módulo intitulado *DiscoveryThread*, executado em segundo plano, responsável por manter o servidor visível para possíveis clientes. Quando a mensagem pré-definida do *DiscoveryServer* é recebida, o servidor identifica que algum cliente está em busca de um servidor para executar o processo de *offloading* e responde à requisição com uma segunda mensagem pré-definida que sinaliza a disponibilidade dele para tal processo. Tal mensagem leva consigo informações sobre o servidor, como o seu *IP* para confirmação.

4.3.2 Serviço de Extração e Carregamento de Metadados

Para que o servidor consiga executar o processamento enviado pelo cliente, se faz necessário que ele tenha conhecimento e acesso a uma definição e representação dos métodos á serem executados.

Para isto, foi implementado, no lado do cliente, um módulo chamado *MetadataExtractor*, responsável pela extração dos metadados que contém o código-fonte executável e todas as informações sobre os métodos que serão migrados para o servidor. Utilizando uma técnica desenvolvida pelo próprio autor para a extração dos arquivos compilados da aplicação em tempo de execução, estes dados são extraídos e armazenados na memória interna do *smartphone* em um arquivo com extensão própria, intitulada *.caos*. Está extensão foi criada, pois, no processo de extração convencional dos metadados, percebeu-se que, no *payload* final, existiam muitas informações extras que não seriam necessários para a finalidade desta pesquisa, causando uma carga útil desnecessariamente maior.

Enquanto isso, no lado do servidor foi implementado um módulo denominado *MetadataLoader*, designado a carregar os métodos a serem executados. Utilizando a técnica *Reflection* disponível na linguagem Java, os metadados presentes no arquivo *.caos* podem ser carregados em tempo de execução, assim possibilitando a aplicação servidora de ter acesso aos métodos e, com isso, executá-los.

4.3.3 Serviço Migração de Metadados

Antes que a aplicação do lado do servidor consiga carregar, por meio do arquivo *.caos*, os métodos a serem executados, o mesmo precisa ser enviado pelo cliente. Para isso, no lado cliente foi criado o módulo *MetadataSender*, responsável por pegar o arquivo extraído pelo *MetadataExtractor*, e enviá-lo para o servidor por meio da rede. Assim como no processo de descoberta, para este envio, utilizou-se o protocolo TCP por padrão.

No lado servidor, o módulo chamado *MetadataReceiver* foi implementado com o intuito de receber o arquivo de metadados enviado pelo *MetadataSender*, e o armazená-los na memória interna do dispositivo servidor, assim o deixando disponível para ser usado pelo *MetadataLoader* posteriormente.

4.3.4 Pilha de Protocolos

Todo o processo de adição de novos protocolos a aplicação é explicado no Apêndice A. O desenvolvedor precisa definir algumas configurações para integrar o protocolo a aplicação, utilizando de interfaces e definições estabelecidas pelo módulo.

4.4 Considerações finais

Nesse Capítulo foi apresentado a proposta do presente trabalho, que busca realizar uma análise comparativa entre protocolos da camada de *Transporte* e *Aplicação* no processo de *offloading* em ambiente de MCC. Para isto, foi desenvolvido um módulo expansível que implementa os protocolos destas camadas, bem como possibilita, a nível de código, a implementação de novos protocolos por meio de uso de interfaces e definições presentes no presente módulo.

5 RESULTADOS

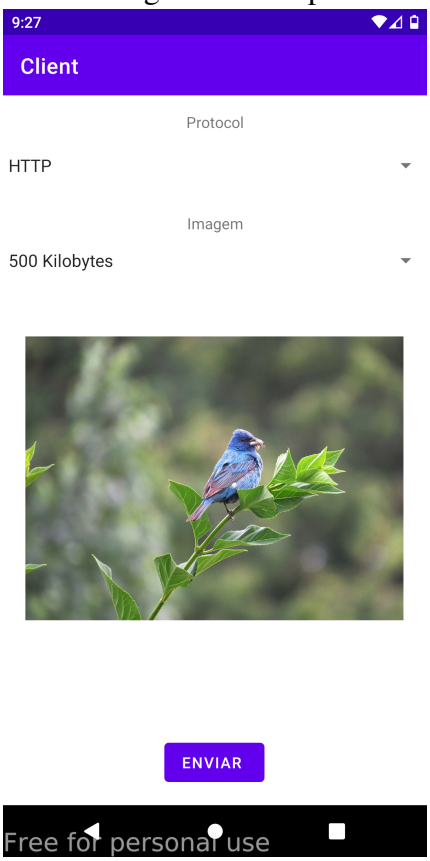
Este Capítulo apresenta os testes realizados e os resultados obtidos através deles. Para a realização dos testes, foi necessário o desenvolvimento de uma nova aplicação intitulada de *ImaBench* que será detalhada na Seção 5.1. A Seção 5.2 apresentará uma descrição do ambiente montado para a execução dos testes. A Seção 5.3 apresentará uma descrição do experimento realizado, mostrando as métricas utilizadas para medição, bem como as imagens e os passos utilizados durante os testes. A Seção 5.4 apresentará e analisará os resultados obtidos com os testes realizados.

5.1 Aplicação

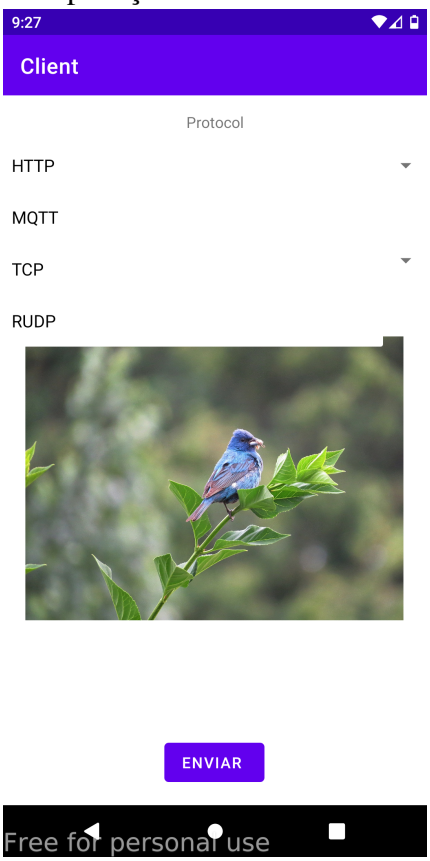
ImaBench é uma aplicação móvel do tipo *BenchMarking*, ou seja, que se utiliza de operações computacionais complexas para medir o desempenho delas em relação ao tempo de processamento necessário para processá-las. Com o suporte aos protocolos indicados na Tabela 3, a aplicação possibilita aplicar um filtro Preto e Branco em uma imagem previamente estabelecida utilizando um protocolo definido pelo usuário em tempo de execução. Para a aplicação do filtro, foi usada uma implementação desenvolvida pelo autor do trabalho, que consiste na análise de cada pixel da imagem para determinar a sua nova cor dependendo da sua tonalidade padrão.

A Figura 15 apresenta capturas de tela da aplicação *ImaBench*. Ao inicializar o aplicativo, o usuário visualizará a tela apresentada na Figura 15.a. Durante a interação com o aplicativo, inicialmente, o usuário tem a opção de escolher qual protocolo será utilizado no processo de *offloading*: HTTP, MQTT, TCP ou RUDP (Figura 15.b), bem como a imagem que será utilizada no processo, onde cada imagem representa um tamanho de *payload*: 500 *Kilobytes*, 1 *Megabyte*, 2 *Megabytes*, 4 *Megabytes*, ou 8 *Megabytes* (Figura 15.c). Feito isso, o usuário deve clicar no botão ENVIAR para que o processo de aplicação do filtro seja migrado e executado para o servidor e a imagem filtrada seja retornada como resultado do *offloading* (Figura 15.d).

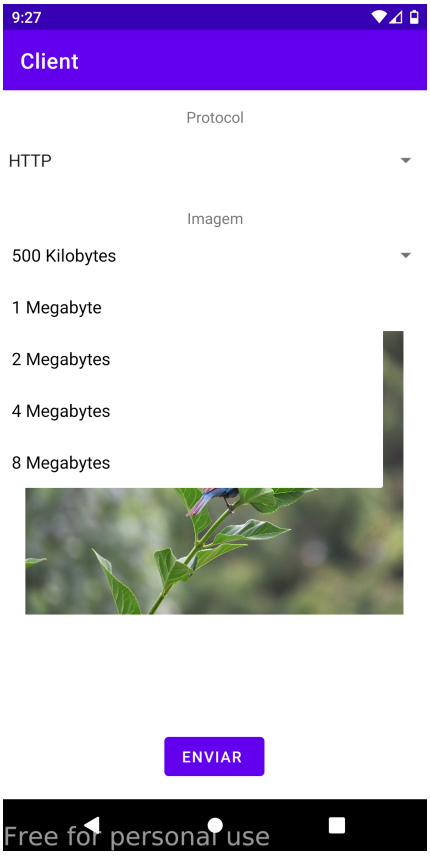
Figura 15 – Capturas de tela da aplicação ImaBench



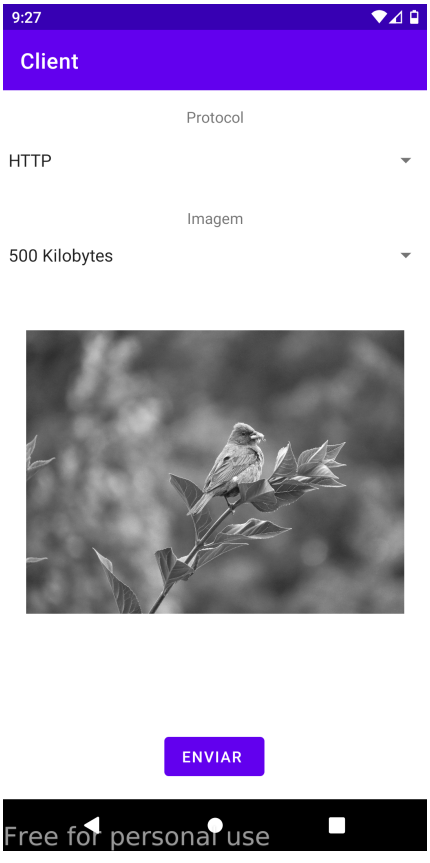
(a) Tela inicial



(b) Tela mostrando os protocolos disponíveis



(c) Tela mostrando as imagens disponíveis



(d) Tela mostrando o resultado

Fonte – Próprio autor

5.2 Descrição do Ambiente de Execução

Para o experimento foi utilizado dois *smartphones* e um notebook auxiliar. O primeiro *smartphone* (S1), utilizado como cliente, possui as seguintes características:

- Xiaomi Mi 8 Lite
- Processador de 64 *bits* Qualcomm SDM660 Snapdragon 660 (14 nm) 2.2Ghz Octa-core
- 4GB de memória RAM
- 64GB de memória interna
- Android 10 (Google API 29)

O *smartphone* (S2), utilizado como servidor, tem as seguintes configurações:

- Xiaomi Redmi Note 7
- Processador de 64 *bits* Qualcomm SDM660 Snapdragon 660 (14 nm) 2.2Ghz Octa-core
- 4GB de memória RAM
- 64GB de memória interna
- Android 10 (Google API 29)

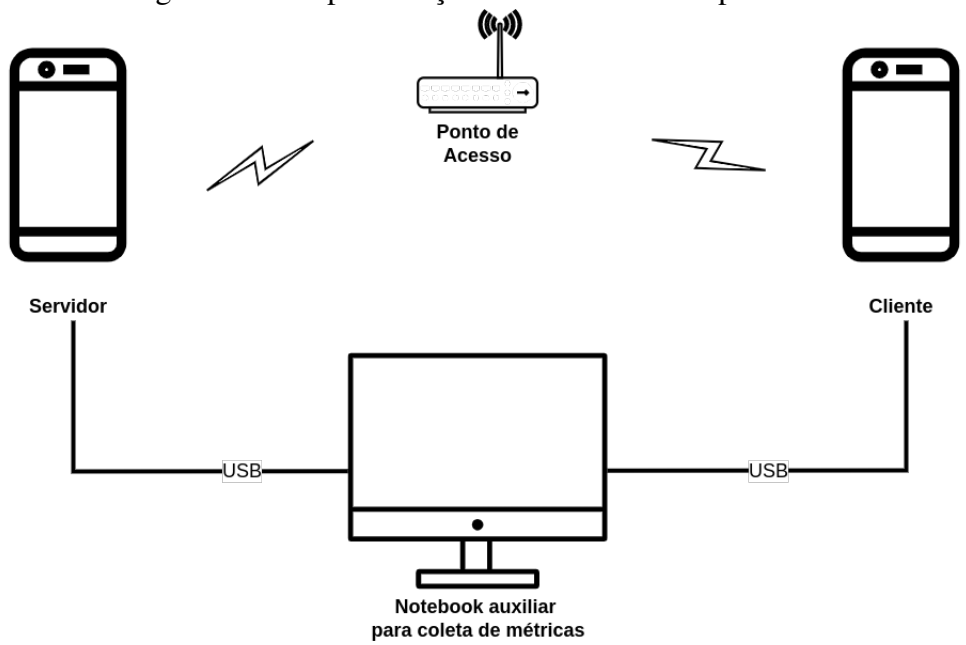
O *notebook* (N1), utilizado como coletor de métricas, tem as seguintes configurações:

- Notebook Samsung Book
- Processador de 64 *bits* core i7 i7-1165G7 11th geração @ 2.80GHz
- 8GB de memória RAM
- Sistema Linux (Pop!_OS 22.04 LTS)

Com o intuito de minimizar ao máximo inferências externas, também foi utilizado um Ponto de Acesso, onde somente estes dispositivos mencionados acima estavam conectados. Além disto, como o experimento não necessitava de acesso direto à internet, o Ponto de Acesso estava desconectado e utilizando somente o recurso de rede local. Para exemplificar melhor o ambiente do experimento, a Figura 16 aborda uma representação do cenário de testes.

Pode-se notar que ambos os *smartphones* estão conectados ao *notebook* utilizando uma conexão *USB*. Isto se fez necessário para, assim como a escolha do Ponto de Acesso, minimizar ao máximo interferências e algum possível *delay* na coleta das métricas.

Figura 16 – Representação do ambiente de experimento



Fonte – Próprio autor

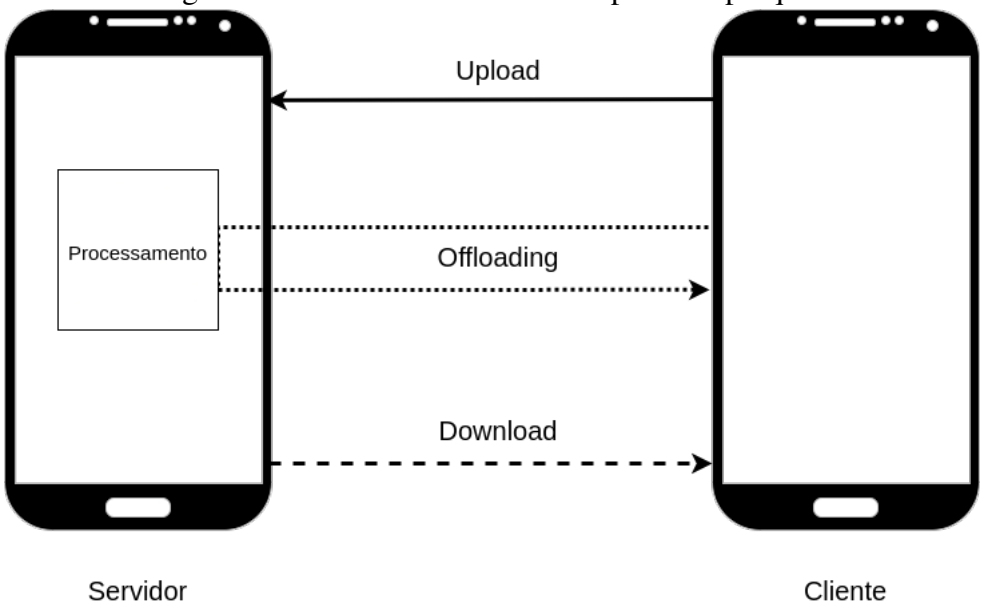
5.3 Descrição do Experimento

O experimento foi realizado a partir da utilização da aplicação *ImaBench* com o intuito de medir o desempenho dos protocolos no processo de *offloading*. Para o presente trabalho, foi feito a medição do tempo gasto com a comunicação (*e.g., upload e download*) e com a execução do método que aplica o filtro Preto e Branco na foto pré-definida. A medição do tempo de execução foi considerada, pois ela também influência no tempo total do *offloading*.

O tempo total do *offloading* foi calculado a partir da diferença entre o instante em que a requisição é realizada pela aplicação cliente até o momento em que ela recebe o resultado da execução do servidor. O tempo de *upload* foi calculado levando em consideração o instante em que a aplicação do cliente realiza a requisição até o momento que a aplicação do servidor recebe a solicitação. Já o tempo de *download* foi calculado usando o raciocínio oposto, isto é, considerando o instante em que o servidor envia o resultado da execução, até o momento em que a aplicação do cliente recebe a resposta. Por fim, o tempo de execução foi medido usando o momento em que o servidor recebe a requisição do cliente até o momento após a execução do método solicitado. Todas as métricas foram medidas em milissegundos (ms). A Figura 17 exemplifica de forma visual toda estas métricas.

Para a realização dos testes, foram escolhidos 4 imagens com diferentes tamanhos

Figura 17 – Métricas de análise da presente pesquisa



Fonte – Próprio autor

para aplicação do filtro Preto e Branco pelo aplicativo *ImaBench*. A escolha das imagens aconteceu de forma empírica de forma que o tamanho das mesmas dobram para cenário dos testes, com imagem inicial de 545,199 *bytes* (cerca de 500 *Kilobytes*), até imagem com 8,281,153 *bytes* (8 *Megabytes*).Em outras palavras, o teste consiste em uma imagem com 5 tamanhos distintos: 500 *Kilobytes*, 1 *Megabyte*, 2 *Megabytes*, 4 *Megabytes*, e 8 *Megabytes*, e para cada cenário de teste, uma imagem é utilizada e executada por cada um dos protocolos previamente escolhidos e implementados no *ImaBench*.

Conforme o Teorema Central do Limite (TCL) (FISCHER, 2010), ao ter uma amostra suficientemente grande, a distribuição de probabilidade da média amostral pode ser aproximada por uma distribuição normal. Essa aproximação é válida a partir de 30 médias amostrais. Assim, tanto para cada uma das imagens citadas a cima, quanto para cada um dos protocolos implementados (i.e., HTTP, MQTT, TCP, e RUDP), os experimentos foram realizados 30 vezes e os dados coletados foram adicionados em uma planilha para posterior análise dos resultados.

5.4 Resultados Obtidos

Após toda a coleta dos dados, eles foram armazenados em uma planilha e foi realizada uma análise acerca dos valores obtidos a partir dos testes. Como abordado na Seção anterior, os testes foram repetidos 30 vezes para cada imagem e protocolo escolhido, para cada

uma das vezes que os cenários foram executados, foi feito a coleta de todas as métricas citadas anteriormente (i.e., tempo de *upload*, tempo de processamento, e tempo de *download*). Na análise dos dados foi considerado um intervalo de confiança de 95%, e utilizado o *Software* Gnumeric[1] para a organização dos mesmo.

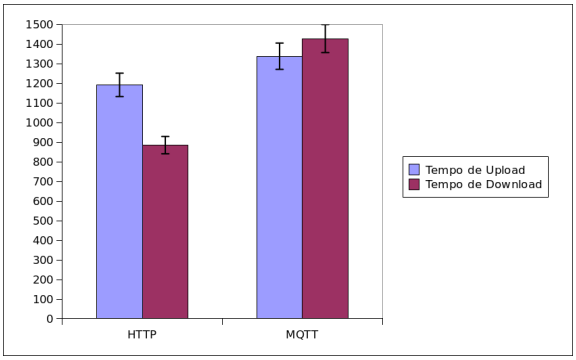
Realizando uma análise por camada, as Figuras 18 e 19 apresentam gráficos contendo o tempo gasto no processo de *upload* e *download* dos protocolos implementado da camada de *Aplicação* e *Transporte*, respectivamente.

Observa-se na Figura 18 que o protocolo HTTP se mostra, em todos os cenários, com um tempo menor e mais consistente em comparação com o protocolo MQTT. Como reforço, percebe-se através dos gráficos 18a e 18b que o protocolo MQTT se mostra inconsistente, tendo um tempo de *download* maior que o tempo de *upload* nestes dois primeiros cenários, e se reverte como mostrado nos gráficos 18c 18d e 18e. Portanto, pode-se confirmar que analisando do ponto de vista da camada de *Aplicação*, o protocolo *HTTP* se mostra uma melhor opção de uso no processo de *offloading*.

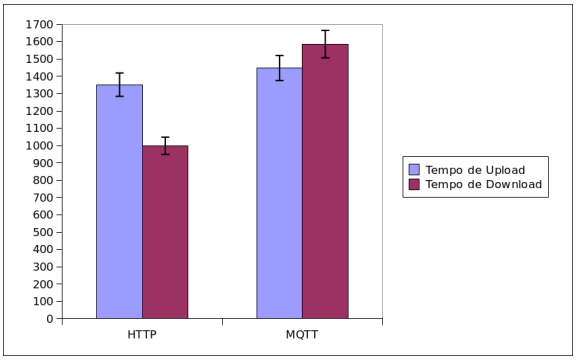
Já analisando a partir da camada de *Transporte*, confirme os gráficos presentes na Figura 19, pode-se notar uma diferença bem visível entre os protocolos TCP e *RUDP*. Onde, de forma bastante significativa, o protocolo TCP se mostra com bem menos tempo e bem mais consistente. Portanto, conclui-se que tendo como analise a camada de *Transporte*, o protocolo TCP é uma melhor escolha frente ao RUDP no processo de *offloading*.

[1] <http://www.gnumeric.org/>

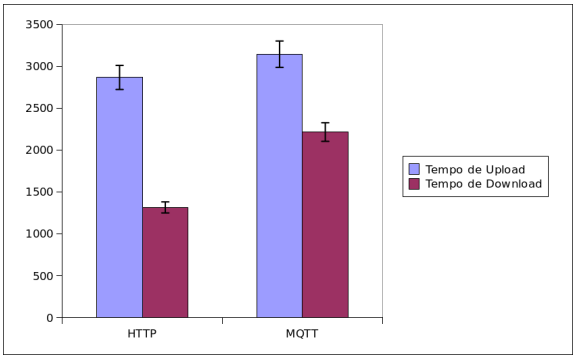
Figura 18 – Tempo em milissegundos (ms) gasto com processo de *upload* e *download* (camada de *Aplicação*)



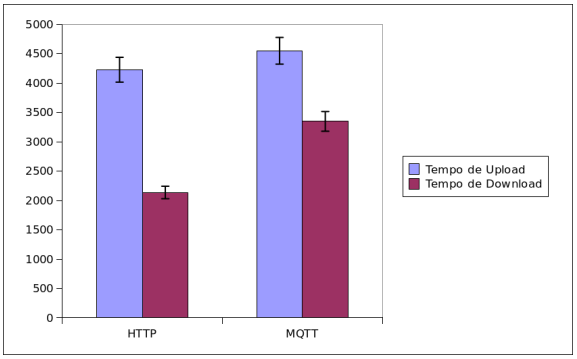
(a) Imagem com 500KB



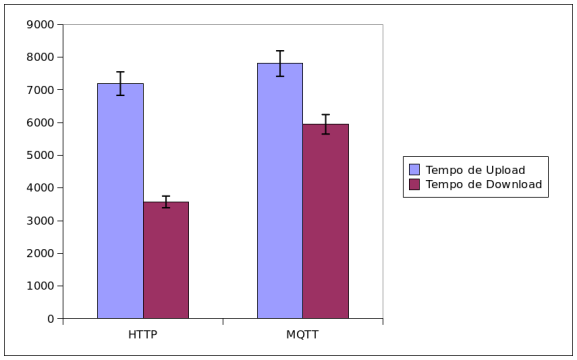
(b) Imagem com 1MB



(c) Imagem com 2MB



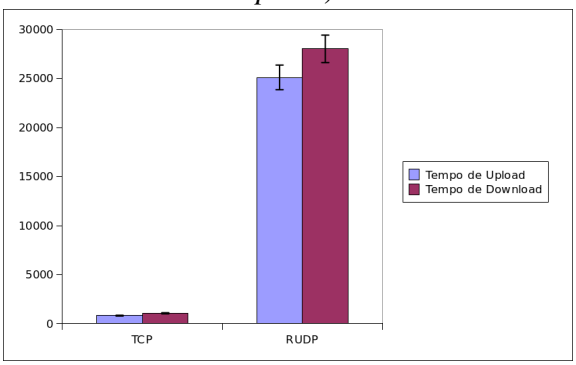
(d) Imagem com 4MB



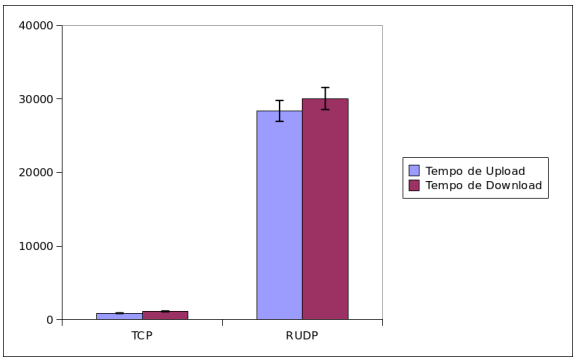
(e) Imagem com 8MB

Fonte – Próprio autor

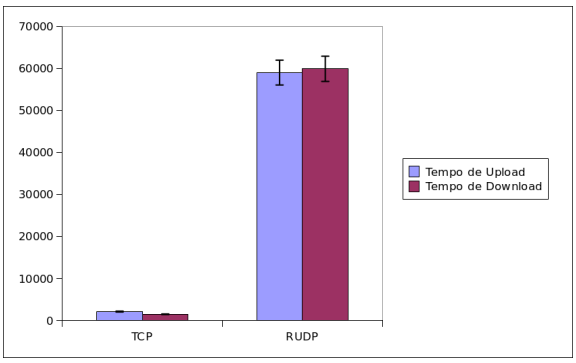
Figura 19 – Tempo em milissegundos (ms) gasto com processo de *upload* e *download* (camada de *Transporte*)



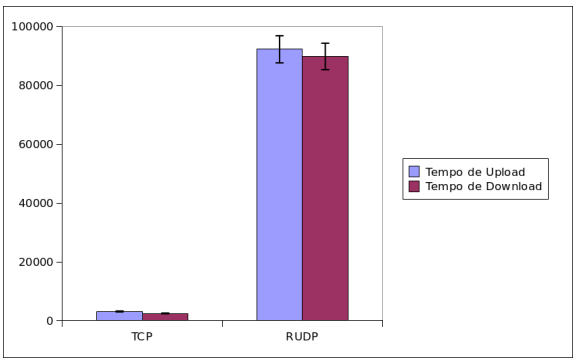
(a) Imagem com 500KB



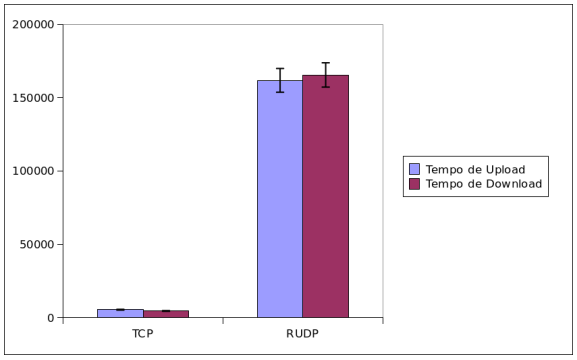
(b) Imagem com 1MB



(c) Imagem com 2MB



(d) Imagem com 4MB



(e) Imagem com 8MB

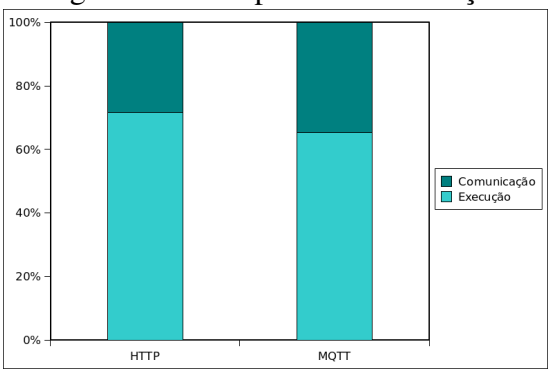
Fonte – Próprio autor

Novamente, em uma análise ao nível de camada, as Figuras 20 e 21 mostram o tempo gasto entre comunicação e execução durante o processo de *offloading* utilizando os protocolos da camada de *Aplicação* e *Transporte*, nesta ordem.

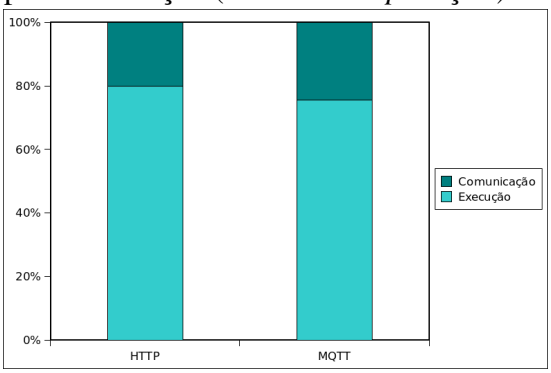
É importante ressaltar que o tempo de comunicação é constituído do somatório entre o tempo de *download* e o tempo de *upload*, assim como o tempo de execução é considerado como o tempo necessário para a execução do método solicitado no processo de *offloading*.

Estes gráficos abaixo mostrado tem o intuito principal de mostrar a relação de impacto que cada protocolo causa no processo de *offloading*, mostrando a relação em porcentagem do quanto o tempo de comunicação interfere no processo total, comparado com o tempo de execução.

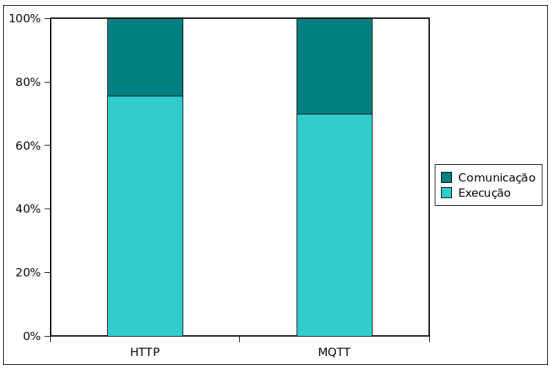
Figura 20 – Tempo de Comunicação x Tempo de Execução (camada de *Aplicação)*



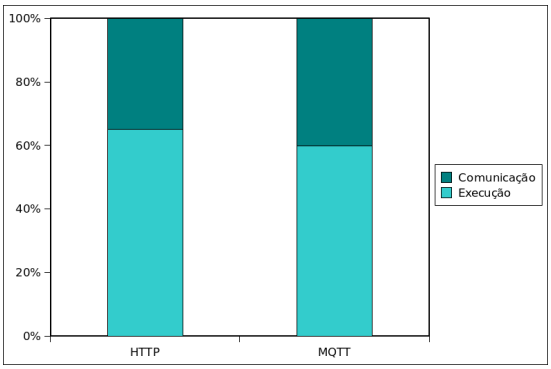
(a) Imagem com 500KB



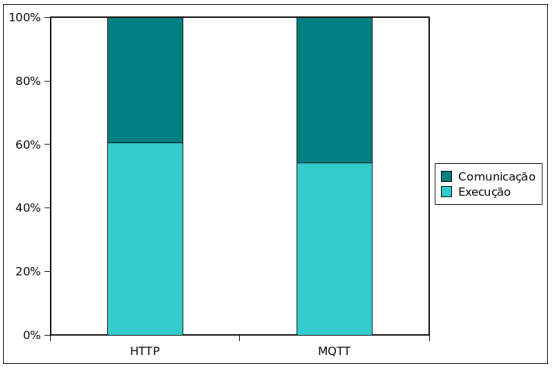
(b) Imagem com 1MB.png



(c) Imagem com 2MB.png



(d) Imagem com 4MB.png

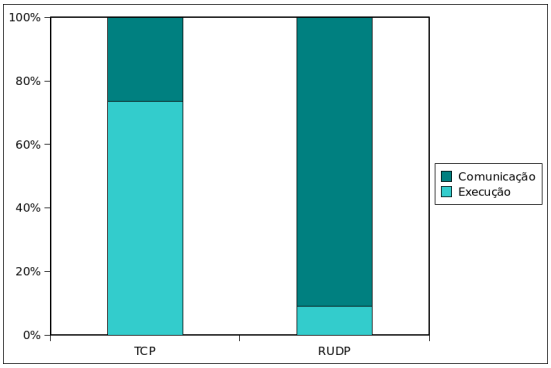


(e) Imagem com 8MB.png

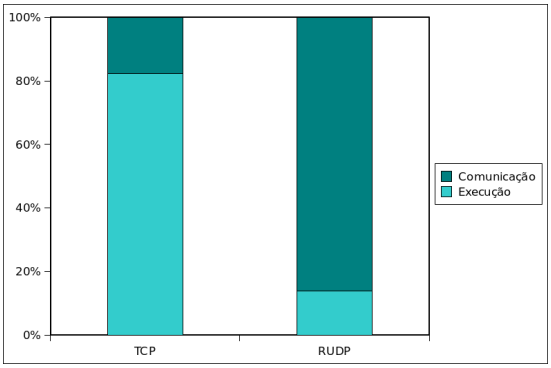
Fonte – Próprio autor

Com base na Figura 20 é possível reafirmar o que foi analisado nos gráficos da Figura 18, onde foi visto que o protocolo HTTP retem uma menor influência (que varia de 20% a 39%) no processo de *offloading* que o protocolo MQTT (com influênia entre 22% e 42%), onde este comportamento se repete em todos os cenários, assim reafirmando que o protocolo *HTTP* é

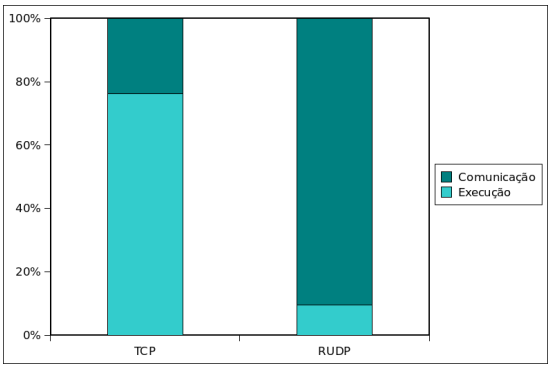
Figura 21 – Tempo de Comunicação x Tempo de Execução (camada de *Transporte)*



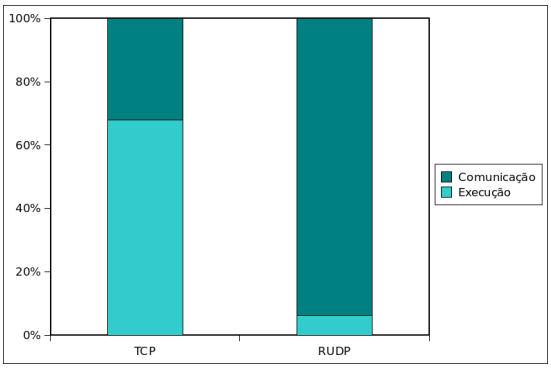
(a) Imagem com 500KB



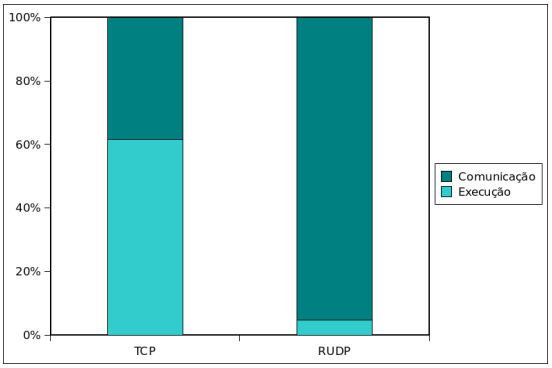
(b) Imagem com 1MB.png



(c) Imagem com 2MB.png



(d) Imagem com 4MB.png



(e) Imagem com 8MB.png

Fonte – Próprio autor

uma melhor opção no processo de *offloading* em comparação com o *MQTT*.

Por fim, a Figura 21 também reafirma o que foi analisado nos gráficos da Figura 19, onde o protocolo TCP se mostra tendo uma influência muito menor (com valores que variam entre 18% e 39%) em comparação ao RUDP (com valores que variam entre 83% e 97%). Portanto, do ponto de vista da camada de *Transporte*, o protocolo TCP se mostrou uma melhor opção.

5.5 Considerações Finais

Esse Capítulo apresentou uma aplicação móvel no modelo cliente-servidor que utiliza do contexto de processamento de imagem e implementa os protocolos HTTP, MQTT, TCP e RUDP. Foram realizados testes com o intuito de analisar o impacto de cada um dos protocolos no processo de *offloading*, assim como fazer uma comparação entre os mesmos no quesito desempenho.

A análise dos resultados foi realizada utilizando comparações de tempo em relação ao processo de *offloading* utilizando imagens com diferentes tamanhos, onde para cada imagem o processo de teste aconteceu para cada um dos protocolos. Com relação ao tempo, foi notado que os protocolos que tiveram um melhor desempenhos foram o HTTP, MQTT e TCP, onde o TCP apresentou se mostrou como o melhor entre os três no balanço entre *upload* e *download*. Já, por outro lado, o protocolo RUDP se mostrou ser menos eficiente, pois as demostrações de tempo do mesmo se mostraram muito maiores que os demais. Em outros gráficos utilizando como comparação o tempo de comunicação e o tempo de execução os resultados foram semelhantes, enquanto o TCP se mostrou o protocolo com menor influência no processo de *offloading* com valores em torno de 39% para o cenário com imagem de 8 *Megabytes*, o RUDP se mostrou o mais influente, com valores em torno de 97% para o mesmo cenário.

6 CONCLUSÃO E TRABALHOS FUTUROS

Este Capítulo contém um resumo do que foi discutido e desenvolvido neste trabalho de conclusão de curso. A Seção 6.1 apresenta os resultados alcançados utilizando a proposta, enquanto na Seção 6.2 é apresentada as principais limitações encontradas na solução durante o mesmo. Por fim, a última Seção apresenta os trabalhos futuros.

6.1 Resultados Alcançados

O presente trabalho apresentou um estudo comparativo entre protocolos da camada de transporte e aplicação na utilização do *offloading*. A metodologia utilizada na realização deste trabalho destacou as etapas necessárias para a concretização do mesmo, iniciando pelo estudo bibliográfico para a escolha dos protocolos HTTP, MQTT, TCP e RUDP. Após isso, foi feito a definição da arquitetura a ser utilizada na implementação da proposta, além das definições de métricas utilizadas para medir o impacto dos protocolos no processo de *offloading* em ambiente MCC.

O presente trabalho apresentou a implementação de uma aplicação cliente-servidor para processamento de imagem que implementa os protocolos citados anteriormente, e possibilita a escolha de um deles a ser utilizado me tempo de execução do *offloading*. Além isso, a implementação inclui um módulo que possibilita a desenvolvedores implementar outros protocolos, abrindo assim a possibilidade de desenvolvedores da plataforma *Android* utilizar os recursos do *offloading* com outros protocolos de sua escolha.

Por fim, o capítulo de Resultados apresentou a realização de todos os testes, assim como a coleta e análise dos dados obtidos pelos mesmos. Durante a análise, foi possível identificar que o protocolo TCP se comportou de melhor forma em todos os cenários, mantendo um tempo de comunicação bem menor e com menos influência no processo de *offloading* que os demais.

Por outro lado, foi identificado que o protocolo RUDP, cuja proposta era ter uma entrega de pacotes semelhante ao do TCP e com menos *overhead*, se mostrou o pior protocolo em todos os cenários, com tempo de envio e recebimento muito superior em comparação aos demais protocolos.

Com uma análise dividida por camada, onde se pode identificar que analisando a partir da camada de *Aplicação*, foi notado que o protocolo HTTP oferece um menor tempo entre tempo de *upload* e *download*, bem como uma menor influência no processo de *offloading* em comparação com o protocolo MQTT. Já analisando a camada de *Transporte*, foi identificado que o protocolo TCP deteve um menor tempo e menor influência em comparação ao *RUDP*.

Por fim, foi-se concluído que existe sim um impacto positivo ou negativo no tempo de *offloading* a depender do protocolo escolhido, e que esse impacto não se concentra em uma só camada, pois a escolha do protocolo de cada camada pode causar um impacto diferente. Assim, abrem-se possibilidades para estudo de combinação de protocolos de outras camadas para obter um melhor resultado no processo de *offloading* em ambientes de MCC.

6.2 Limitações Encontradas

Durante a execução do presente trabalho foram encontradas algumas dificuldades que acabaram por limitando em algumas execuções, o qual foram inicialmente propostas no trabalho de conclusão 1. A primeira limitação encontrada foi acerca do protocolo QUIC citado como um dos protocolos que seria utilizado na proposta do trabalho, que por conta de ser um protocolo relativamente novo, foi encontrado pouco material prático, ou bibliotecas que auxiliassem na implementação do mesmo. Outra limitação encontrada foi a implementação da citada proposta deste trabalho no ambiente do *Context Acquisition and Offloading System* (CAOS) D2D, que por razões de tempo e conhecimento do presente autor, a implementação não foi realizada sendo removida do escopo deste trabalho.

6.3 Trabalhos Futuros

Como trabalhos futuros, propõem-se algumas melhorias no trabalho, tais como:

- Realizar uma análise do impacto causado por outras camadas;
- Realizar um estudo cujo propósito é combinar protocolos de outras camadas a fim de obter um menor tempo de *offloading*.
- Implementar a proposta deste trabalho junto ao CAOS para possibilitar a escolha do protocolo a ser utilizado em tempo de execução no *framework*;

REFERÊNCIAS

ACADEMY, K. **Transmission Control Protocol (TCP)**. 2020. <https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:the-internet/xcae6f4a7ff015e7d:transporting-packets/a/transmission-control-protocol--tcp>. Accessed: 2022-10-16.

ARTAIL KARIM FRENN, H. S. A.; ARTAIL, H. A framework of mobile cloudlet centers based on the use of mobile devices as cloudlets. In: . [S.l.: s.n.], 2015.

CALHEIROS R. RANJAN, A. B. R. *et al.* A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. In: . [S.l.: s.n.], 2011.

CISCO, C. V. N. I. Global mobile data traffic forecast update, 2016–2021. **white paper**, 2017.

COSTA, P. B.; REGO, P. A. L.; ROCHA, L. S.; TRINTA, F. A. M.; SOUZA, J. N. de. Mpos: A multiplatform offloading system. In: **Proceedings of the 30th Annual ACM Symposium on Applied Computing**. New York, NY, USA: ACM, 2015. (SAC '15), p. 577–584. ISBN 978-1-4503-3196-8. Disponível em: <http://doi.acm.org/10.1145/2695664.2695945>.

FERNANDO, N.; LOKE, S. W.; RAHAYU, W. Mobile cloud computing: A survey. **Future generation computer systems**, Elsevier, v. 29, n. 1, p. 84–106, 2013.

FISCHER, H. **A history of the central limit theorem: From classical to modern probability theory**. [S.l.]: Springer Science & Business Media, 2010.

FORMAN, G. H.; ZAHORJAN, J. The challenges of mobile computing. **Computer**, IEEE, v. 27, n. 4, p. 38–47, 1994.

FOROUZAN, B. A. Comunicação de dados e redes de computadores. 4ª ed. In: . [S.l.: s.n.], 2008.

GEEKS, G. F. **Introduction of Message Queue Telemetry Transport Protocol (MQTT)**. 2021. <https://www.geeksforgeeks.org/introduction-of-message-queue-telemetry-transport-protocol-mqtt/>. Accessed: 2022-10-28.

GOMES, F. A.; REGO, P. A.; ROCHA, L.; SOUZA, J. N. de; TRINTA, F. Caos: A context acquisition and offloading system. In: IEEE. **2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)**. [S.l.], 2017. v. 1, p. 957–966.

GUPTA, A. K. Challenges of mobile computing. In: **Proceedings of 2nd National Conference on Challenges and Opportunities in Information Technology (COIT-2008). Mandi Gobindgarh, India: RIMT-IET**. [S.l.: s.n.], 2008. p. 86–90.

JUNIPER, R. **MOBILE CONTACTLESS PAYMENT TRANSACTION VOLUMES TO GROW BY 92% GLOBALLY BY 2023, OUTPACING CARD VOLUME GROWTH**. 2021. <https://www.juniperresearch.com/pressreleases/mobile-contactless-payment-transaction-volumes>. Accessed: 2022-03-09.

KHAN MAZLIZA OTHMAN, S. A. M. Atta ur R. *et al.* A survey of mobile cloud computing application models. IEEE, v. 16, p. 393 – 413, 2013.

KHARBANDA, H.; KRISHNAN, M.; CAMPBELL, R. H. Synergy: A middleware for energy conservation in mobile devices. In: IEEE. **2012 IEEE International Conference on Cluster Computing**. [S.l.], 2012. p. 54–62.

KUMAR, K.; LIU, J.; LU, Y.-H.; BHARGAVA, B. A survey of computation offloading for mobile systems. **Mobile Networks and Applications**, Springer, v. 18, n. 1, p. 129–140, 2013.

KUROSE, J. F. **Redes de computadores e a internet: uma abordagem top-down**. [S.l.]: Pearson, 2010.

MATEUS, G. R.; LOUREIRO, A. A. F. Introdução à computação móvel. DCC/IM, COPPE/UFRJ, 1998.

MENDES, D. R. Redes de computadores: Teoria e prática. In: . [S.l.: s.n.], 2007.

MISHRA, S. The tcp/ip model: An overview. 2017.

NAIK, N. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. 2017.

QU SONGJIE WANG, P. C. C. Dycoco: A dynamic computation offloading and control framework for drone video analytics. In: . [S.l.: s.n.], 2019.

QURESHI, S. S.; AHMAD, T.; RAFIQUE, K. *et al.* Mobile cloud computing as future for mobile applications-implementation methods and challenging issues. In: IEEE. **2011 IEEE International Conference on Cloud Computing and Intelligence Systems**. [S.l.], 2011. p. 467–471.

SANTOS, G. B. dos; REGO, P. A.; TRINTA, F. Uma proposta de solução para offloading de métodos entre dispositivos móveis. In: SBC. **Anais Estendidos do XXIII Simpósio Brasileiro de Sistemas Multimídia e Web**. [S.l.], 2017. p. 76–81.

SATYANARAYANAN, M.; BAHL, P.; CACERES, R.; DAVIES, N. The case for vm-based cloudlets in mobile computing. **Pervasive Computing, IEEE**, v. 8, n. 4, p. 14–23, Oct 2009. ISSN 1536-1268.

SATYANARAYANAN, P. B. The case for vm-based cloudlets in mobile computing. In: . [S.l.: s.n.], 2009.

SHIRAZ, M.; GANI, A.; KHOKHAR, R.; BUYYA, R. A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing. **Communications Surveys Tutorials, IEEE**, v. 15, n. 3, p. 1294–1313, Third 2013. ISSN 1553-877X.

SILVA, W. R. da. Estudo comparativo do impacto da segurança em ambientes de mobile cloud computing. In: . [S.l.: s.n.], 2019.

SOMULA, R. S.; SASIKALA, R. A survey on mobile cloud computing: mobile computing+ cloud computing (mcc= mc+ cc). **Scalable Computing: Practice and Experience**, v. 19, n. 4, p. 309–337, 2018.

TANENBAUM, A. S. Redes de computadores. 5ª ed. In: . [S.l.: s.n.], 2011.

THAMMADI, A. Reliable user datagram protocol (rudp). In: . [S.l.: s.n.], 2009.

WANG, L.; LASZEWSKI, G. V.; YOUNGE, A.; HE, X.; KUNZE, M.; TAO, J.; FU, C. Cloud computing: a perspective study. **New Generation Computing**, Springer, v. 28, n. 2, p. 137–146, 2010.

YOKOTANI, Y. S. T. Comparison with http and mqtt on required network resources for iot. 2016.

YOKOTANI, Y. S. T. Comparison with http and mqtt on required network resources for iot. **Procedia Computer Science**, 2016.

APÊNDICE A – PROCESSO DE CONFIGURAÇÃO

Este apêndice descreve todo o processo de adição de novos protocolos. O desenvolvedor precisa definir algumas configurações para integrar o protocolo a aplicação, utilizando de interfaces e definições estabelecidas pelo módulo. Para o desenvolvimento de novos protocolos, o desenvolvedor deve seguir os seguintes passos:

No lado do cliente - Uma nova classe Java deve ser criada para a implementação do novo protocolo. Após s criação da mesma, esta classe deve implementar uma interface do módulo denominada *ProtocolService*. Está interface carregará consigo a assinatura dos métodos necessários para a implementação do processo de *offloading*, métodos estes que devem ser sobrescritos pelo desenvolvedor do código, utilizando as próprias definições do protocolo escolhido. O seguinte bloco de código demostra este passo para um novo protocolo chamado *MyOwnProtocol*:

```
1  package foo.bar.customprotocols;
2  package org.customprotocol.OwnProtocol;
3
4  import br.ufc.great.caos.service.protocol.client.model.
       services.ProtocolService;
5  import br.ufc.great.caos.service.protocol.core.offload.
       InvocableMethod;
6
7  public class MyOwnProtocol implements ProtocolService {
8      @Override
9      public boolean init() {
10         return OwnProtocol.init();
11     }
12
13     @Override
14     public boolean connect(String ip, Integer port) {
15         return OwnProtocol.connect(ip, port);
16     }
17
18     @Override
```

```
19      public boolean disconnect() {
20          return OwnProtocol.disconnect();
21      }
22
23      @Override
24      public Object executeOffload(InvocableMethod method) {
25          return OwnProtocol.send(method);
26      }
27
28      @Override
29      public String isInstanceOf() {
30          return return "OwnProtocol";;
31      }
32  }
```

Listagem 1 – Implementação dos métodos - Cliente

Após implementação devida dos métodos conforme a interface *ProtocolService*, na classe principal da aplicação, uma instância do protocolo deve ser criada, bem como uma instância do cliente do presente módulo. Após isto, o novo protocolo deve ser passado para a instância do cliente do módulo, inferindo que este protocolo será agora utilizado pelo cliente como meio de comunicação no processo de *offloading*. O seguinte bloco de código, representa este passo para o protocolo *MyOwnProtocol*.

```
1   public class MainActivity extends AppCompatActivity {
2
3       @Override
4       protected void onCreate(Bundle savedInstanceState) {
5
6           ProtocolService protocol = new MyOwnProtocol();
7           Client client = new Client(protocol);
8
9
10      }
11  };
```

Listagem 2 – Adição de novo protocolo no módulo - Cliente

Com isso, métodos disponíveis na classe cliente como *connect()*, *desconnect()*, e *execute-Offload* podem ser chamados utilizando o protocolo *MyOwnProtocol*.

No lado do servidor - Semelhante ao lado do cliente, uma classe Java deve ser criada para a implementação do protocolo, bem como implementar a classe *ProtocolService* do lado do servidor deve ser feito para utilizar a assinatura dos métodos necessários no processo de comunicação. A seguir, o bloco de código irá exemplificar como esta implementação acontece.

```
1  package foo.bar.customprotocols;
2  package org.customprotocol.OwnProtocol;
3
4  import br.ufc.great.caos.service.protocol.server.model.
       services.ProtocolService;
5
6  public class MyOwnProtocol implements ProtocolService {
7      @Override
8      public boolean init() {
9          return OwnProtocol.init();
10     }
11
12     @Override
13     public boolean connect(String ip, Integer port) {
14         return OwnProtocol.connect(ip, port);
15     }
16
17     @Override
18     public boolean disconnect() {
19         return OwnProtocol.disconnect();
20     }
21
```

```
22      @Override
23      public String isInstanceOf() {
24          return return "OwnProtocol";;
25      }
26  }
```

Listagem 3 – Implementação dos métodos - Servidor

Após a implementação dos métodos presentes na interface *ProtocolService* do servidor, na classe principal da aplicação, uma instância do protocolo deve ser criada, assim como uma instância do servidor do módulo. E por fim, deve-se passar o novo protocolo como parâmetro, bem como o contexto da aplicação (que será utilizado internamente pelo módulo), para então o indicar que o servidor utilizará o novo protocolo para comunicação no processo de *offloading*. Novamente, o seguinte bloco de código exemplificará esta implementação para o protocolo *MyOwnProtocol*.

```
1  public class MainActivity extends AppCompatActivity {
2
3      @Override
4      protected void onCreate(Bundle savedInstanceState) {
5
6          ProtocolService protocol = new MyOwnProtocol();
7          Server server = new Server(protocol,
               getApplicationContext());
8
9      }
10 };
```

Listagem 4 – Adição de novo protocolo no módulo - Servidor

Assim como no cliente, métodos disponíveis na classe servidor como *connect()*, e *disconnect()* podem ser chamados utilizando o protocolo *MyOwnProtocol*.