UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

GABRIEL AUGUSTO VEIGA RODRIGUES

ANÁLISE DE DESEMPENHO DO BEAGLEBONE BLACK UTILIZANDO O PROTOCOLO MQTT

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2019

GABRIEL AUGUSTO VEIGA RODRIGUES

ANÁLISE DE DESEMPENHO DO BEAGLEBONE BLACK UTILIZANDO O PROTOCOLO MQTT

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Augusto Foronda

PONTA GROSSA

2019

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa
Diretoria de Graduação e Educação Profissional
Departamento Acadêmico de Informática
Bacharelado em Ciência da Computação

TERMO DE APROVAÇÃO

ANÁLISE DE DESEMPENHO DO BEAGLEBONE BLACK UTILIZANDO O PROTOCOLO MQTT

por

GABRIEL AUGUSTO VEIGA RODRIGUES

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 06 de junho de 2019 como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação. Os candidatos foram arguidos pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. Dr. Augusto Foronda
Orientador

_____
Prof. Dr. Richard Duarte Ribeiro
Membro titular

_____
Prof. Dr. André Pinz Borges
Membro titular

_____
Prof. MSc. Geraldo Ranthum
Responsável pelo Trabalho de Conclusão de Curso

_____
Prof. Dr. Saulo Jorge Beltrão de Queiroz
Coordenador do curso

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso -

AGRADECIMENTOS

Um dos desafios da minha vida está sendo vencido: me graduar em uma universidade pública. A Universidade Tecnológica Federal do Paraná foi primordial para meu futuro, pois foi quem me ensinou o percurso do sucesso, e me ensinou a transparência da vida. Espero que os demais alunos dessa universidade tenham a mesma experiência que vivenciei. Nesta trajetória, diversas pessoas foram essenciais para alcançar esse objetivo, as quais não devem ser esquecidas.

Primeiramente, agradeço ao meu irmão, o qual não se encontra mais entre nós, por me incentivar a seguir na área que estou me graduando e que sempre acreditou no meu potencial, fazendo minha fraqueza ser praticamente nula.

Também agradeço à minha mãe Sueli, que nos anos da minha graduação passou por momentos dolorosos da vida, nunca deixou de me apoiar e se esforçar para que o desafio de me graduar em uma universidade pública não falhasse.

À Isabelle, por aguentar todo desespero e fraqueza que surgiram nesse caminho. Obrigado por me apoiar e fortalecer a minha autoconfiança quando estava acabando.

Por fim, gostaria de deixar registrado o meu agradecimento ao meu orientador Augusto Foronda, por toda dedicação e confiança depositada em mim.

RESUMO

RODRIGUES, Gabriel Augusto Veiga. **Análise de Desempenho do BeagleBone Black utilizando o protocolo MQTT**. 2019. 31f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2019.

Este trabalho buscou analisar o desempenho de um *hardware* de baixo custo, o BeagleBone Black, utilizando-se do protocolo MQTT. O ponto de partida foi a necessidade de se preservar a confiabilidade nas redes de Internet das Coisas, visto que sua ausência ocasiona diversos problemas. Levando isso em consideração, realizou-se três testes, com o máximo de 6000 *publishers*, averiguando três aspectos do *hardware*: consumo de CPU, consumo de memória e total de mensagens enviadas sem sucesso. Dos resultados obtidos, notou-se uma alta utilização de processamento de CPU, porém obtiveram-se ótimos resultados na entrega das mensagens, na qual não houve falhas. Assim, é possível concluir que, no âmbito dos cenários definidos neste trabalho, a escolha do BeagleBone Black é uma opção adequada para o uso do protocolo MQTT.

Palavras-chave: *Internet of Things*. MQTT. BeagleBone Black. Desempenho.

ABSTRACT

RODRIGUES, Gabriel Augusto Veiga. **Performance Analysis of BeagleBone Black using the MQTT protocol**. 2019. 31p. Work of Conclusion Course (Bachelor's Degree in Computer Science) - Federal University of Technology - Paraná. Ponta Grossa, 2018.

This work sought to analyze the low budget hardware performance of BeagleBone Black microcontroller, using the MQTT protocol. The starting point was the need of preserving the confiability of IoT networks, taking in consideration the lack of it can cause different relevant problems. Having this in mind, it was realized three different tests, with a max of 6000 publishers, checking three hardware specifications: cpu usage, memory usage and total of failed to send messages. From the results obtained, it was noted a high use of cpu processing, however there was great results on messaging delivery, in which there was not a single fail. Then, it is possible to conclude that, in the different scenarios defined in this work, the choice of BeagleBone Black is proper for MQTT protocol use.

Keywords: Internet of Things. MQTT. BeagleBone Black. Performance.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS

CoAP	*Constrained Application Protocol*
CPU	*Central Processing Unit*
IoT	*Internet of Things*
JDK	*Java Development Kit*
MBytes	*Megabytes*
MIT	*Massachusetts Institute of Technology*
MQTT	*Message Queuing Telemetry Transport*
RFID	*Radio-Frequency Identification*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
WAMP	*Web Application Messaging Protocol*

SUMÁRIO

1 INTRODUÇÃO

A principal finalidade da Internet das Coisas (do inglês, *Internet of Things*, IoT) é interligar objetos realizando sua interação e cooperação mútua com vista a alcançar um fim comum (SINGH, 2014). Contudo, nem todos esses objetos são computacionais, de forma que, para haver a conexão entre eles, é necessário o uso de sensores e atuadores.

Estima-se que o valor potencial econômico mundial em IoT gira em torno de um total de 3,9 a 11,1 trilhões de dólares alcançando entre 25 a 50 bilhões de dispositivos até 2025. As principais áreas para esse valor potencial econômico são fábricas, saúde e agricultura. Somente nessas áreas, o valor potencial pode chegar a 3,7 trilhões em 2025, ou seja, um terço do valor total esperado (MANYIKA, *et al*., 2015).

A possibilidade de conectar uma grande quantidade de diferentes dispositivos é um dos pontos fortes do uso do IoT porém para existir uma conexão funcional entre eles é necessário o uso de um *hardware* chamado *gateway*. O *gateway* é responsável por organizar todo o tráfego de informações em uma rede, ou seja, receber um dado e organizá-lo para entregar ao destino de forma segura e eficiente (FERREIRA, *et al*., 2017).

Existem inúmeras plataformas de *hardware* para solução de *gateway* em uma rede IoT, sendo mais utilizado nesse cenário os *hardwares* de baixo custo, tais como o Arduino, Raspberry Pi e BeagleBone.

O desempenho do *gateway* é uma característica fundamental em IoT pois caso o *hardware* não suporte a solução, grandes problemas podem ser agravados como inconsistências de dados, lentidão na transmissão de dados, colisão de pacotes e tempo de espera de resposta elevado. Considerando a importância do desempenho do *gateway*, neste trabalho foi utilizado o BeagleBone Black.

Dentre os diversos protocolos de comunicações existentes para IoT, o trabalho adotou o MQTT (*Message Queuing Telemetry Transport*). Este protocolo foi criado em 1999 pela IBM com a ideia inicial de vincular sensores em *pipelines* de petróleo a satélites. Realizando uma integração com o protocolo TCP/IP (*Transmission Control Protocol/ Internet Protocol)*, o MQTT utiliza um padrão de transmissão de mensagens chamado o *Publisher-Subscriber* (publicador-assinante) o qual tem um intermediário chamado *broker* (BANKS, 2014). A escolha do MQTT foi

definida em razão de sua simplicidade e flexibilidade de implementação além do crescimento de sua utilização no mercado, por exemplo, no Facebook onde é utilizado no sistema de *instant messaging,* e na Amazon, na qual é um dos protocolos padrões da plataforma AWS IoT (TORRES, 2016).

Desta forma, o propósito desse trabalho será analisar o desempenho de um *hardware* de baixo custo, no caso o BeagleBone Black, em cenários de grande número de *publishers* utilizando o protocolo MQTT.

1.1 OBJETIVOS DO TRABALHO

A seguir são descritos o objetivo geral e os objetivos específicos para a realização do trabalho.

1.1.1 *Objetivo Geral*

Analisar o desempenho do BeagleBone Black operando como um *gateway* para IoT utilizando o protocolo MQTT.

1.1.2 *Objetivos Específicos*

- Implementar o *broker* Mosquitto no *hardware*;
- Construir cenários de testes;
- Extrair os resultados: uso de CPU, consumo de memória e total de mensagens enviadas sem sucesso;
- Analisar o desempenho do *hardware* em cada cenário de teste;
- Apresentar uma comparação entre os cenários de teste e identificar em qual cenário o *hardware* apresentou melhor desempenho.

1.2 JUSTIFICATIVA

A importância da análise de desempenho de um *hardware* de baixo custo decorre da necessidade de se criar um cenário de confiabilidade para realização de futuras implementações em redes IoT com o uso de *hardwares* de baixo custo como um *gateway* que responde a milhares de requisições de leituras realizadas simultaneamente, como sensor público (temperatura, poluição ou presença), e que pode ser acessado via *web* ou *app*.

1.3 ORGANIZAÇÃO DO TRABALHO

O desenvolvimento deste trabalho está organizado em cinco capítulos que apresentarão o seguinte conteúdo:

- Capítulo 1: Introdução - Definição dos principais conceitos utilizados como base do estudo desenvolvido.
- Capítulo 2: Referencial teórico - História e definição de Internet das Coisas. Definição da arquitetura de uma rede de Internet das Coisas e explicação das suas camadas. Definição e explicação do protocolo MQTT.
- Capítulo 3: Desenvolvimento - Instrumentos utilizados para a realização dos testes, tais como *hardwares* e *softwares*. Configuração inicial do ambiente de teste. Cenários de testes.
- Capítulo 4: Resultado – Apresentação e analise dos resultados obtidos, divididos em três categorias: Uso de CPU, consumo de memória e total de mensagens enviadas sem sucesso.
- Capítulo 5: Conclusão.

2 REFERENCIAL TEÓRICO

Este capítulo apresentará definições teóricas necessárias para a compreensão do trabalho desenvolvido. Será explanada a definição e a estrutura de Internet das Coisas, ressaltando o *hardware* Beaglebone Black utilizado como *gateway.* Posteriormente, serão apresentados o protocolo MQTT e o *broker* Mosquitto, itens essenciais dos testes realizados.

2.1 DEFINIÇÃO DE INTERNET DAS COISAS

Em 1999, em um grupo de pesquisadores da universidade MIT (*Massachusetts Institute of Technology*), um participante chamado Kevin Ashton, criou o termo *Internet of Things*. Kevin, fundador do Auto-ID Center, com o intuito de chamar atenção dos executivos da empresa Procter&Gamble, utilizou o termo como título de uma apresentação (ASHTON, 2010).

A ideia central do IoT é permitir a conexão eficiente e segura na troca de dados entre dispositivos e aplicações do mundo real, a fim que eles tomem decisões ou invoquem ações fornecendo serviços avançados (FAN, 2010).

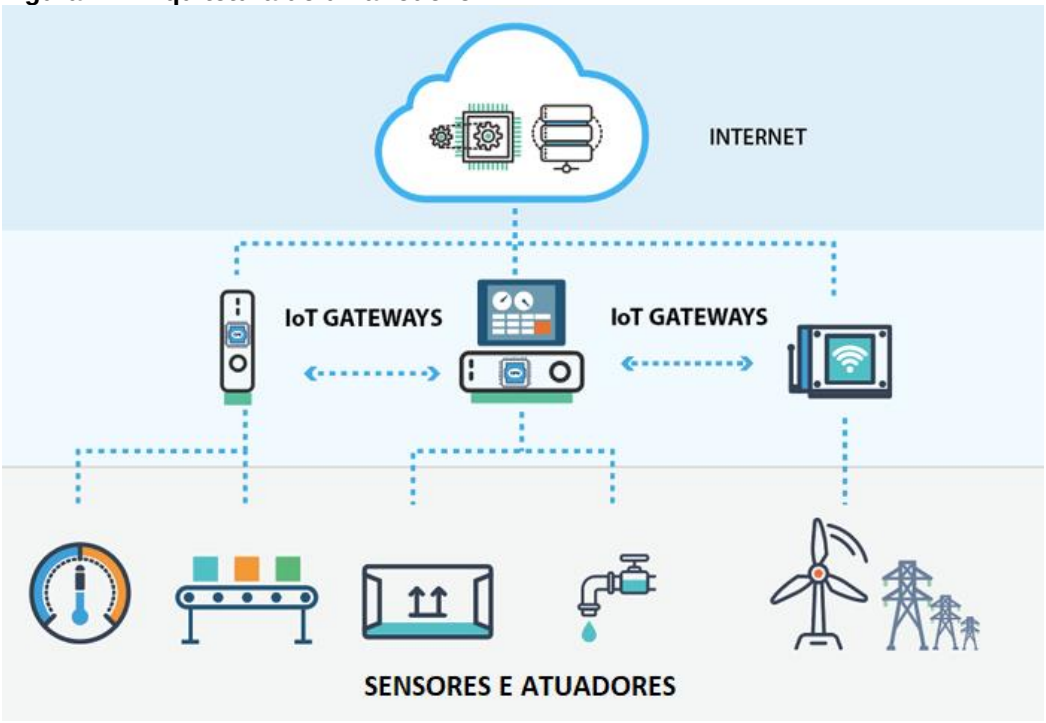
Foi em 2010 que Internet das Coisas se transformou em grande potência por consequência da popularização dos *smartphones* e *tablets.* A quantidade de dispositivos conectados no mundo era aproximadamente de 12,5 bilhões, sendo que a população humana mundial era de 6,8 bilhões, ou seja, a quantidade de dispositivos conectados na Internet passou a quantidade de pessoas no mundo (EVANS, 2011).

2.2 ARQUITETURA DE INTERNET DAS COISAS

Ainda não foi definida uma arquitetura padrão de IoT pois existem sistemas específicos, os quais foram projetados para operarem exclusivamente em uma aplicação criando dificuldade em propor uma interoperabilidade (MODOFF, 2014).

Porém, existe uma arquitetura genérica que as redes IoT seguem. Essa arquitetura, como mostra a Figura 1, é composta por três principais camadas: sensores e atuadores, *gateways* e plataforma de processamento.

Figura 1 – Arquitetura de uma rede IoT



Fonte: (OAS - Open Automation Software, 2010)

2.2.1 *Sensores e Atuadores*

Esses componentes são um dos pilares de um sistema IoT pois são responsáveis por capturar e agir com as informações do cenário ao qual está implantado. Os quais consistem em diversos *hardwares*, tais como sistemas embarcados, sensores de diversos formatos e etiquetas RFID (*Radio-Frequency Identification*) (ZARGHAMI, 2013).

Um dos principais desafios do IoT é o gerenciamento de energia dos sensores e atuadores pois trocar a bateria de milhões de componentes é totalmente inviável. Então, existem diversos estudos a fim de criar métodos para que esses componentes sejam autossustentáveis (EVANS, 2011).

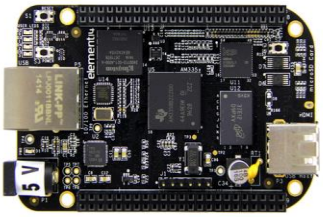
2.2.2 *Gateway*

Para que exista uma organização no fluxo de dados nas redes IoT, envolvendo uma grande quantidade e variedade de sensores e atuadores, torna-se indispensável o uso de um *hardware* chamado *gateway*.

Esse *hardware* é responsável por receber dados e organizá-los para entregar aos destinos de forma segura e eficiente. Ele deve atender a requisitos como: baixo custo, fácil extensibilidade, bom desempenho e suporte a camada de aplicativo (TORRES, 2016).

O desempenho do *gateway* é uma característica fundamental, pois caso o *hardware* não suporte a solução, grandes problemas podem ser agravados como inconsistências de dados, lentidão na transmissão de dados, colisão de pacotes e tempo de espera de resposta elevado. Considerando a importância dessa característica, nesse trabalho será utilizado o BeagleBone Black pois se trata de um *hardware* de baixo custo com uma vantagem de desempenho comparado com seus concorrentes, como mostrado na figura 2, disponíveis no mercado atual (2019).

Figura 2 – *Hardwares* de baixo custo



Nome	Arduino Uno	Raspberry Pi	BleagleBone
Modelo	R3	B	Black
Tamanho	2.95x2.10"	3.37"x2.125"	3.402"x2.098"
Processador	ATMega 328	ARM11	ARM CortexA8
Clock	16MHz	700MHz	1GHz
RAM	2KB	256MB	512MB
Flash	32KB	Somente microSD	4GB + microSD
Ethernet	N/A	10/100	10/100

Fonte: Autoria própria

Conforme a Figura 2, o BeagleBone modelo Black possui mais memória RAM, tendo 512 MBytes, e um processador com maior clock, 1GHZ, que seus concorrentes.

2.2.3 *Plataforma de Processamento*

As redes IoT não conseguem ter capacidade computacional necessária para oferecer seus serviços em todos os locais. Isso ocorre em virtude dos sensores e atuadores possuírem recursos de processamento computacionais limitados. A fim de resolver esse problema, surgiram as plataformas em nuvem tais como Amazon Web

Services (AMAZON, 2006), Google Cloud (GOOGLE, 2008) e Microsoft Azure (MICROSOFT, 2010).

Essas plataformas disponibilizam seus serviços de uma forma que fiquem acessíveis a partir de qualquer lugar aplicando todas as restrições necessárias fornecendo recursos de processamento, armazenamento e gerência dos dados (ZHOU, 2018).

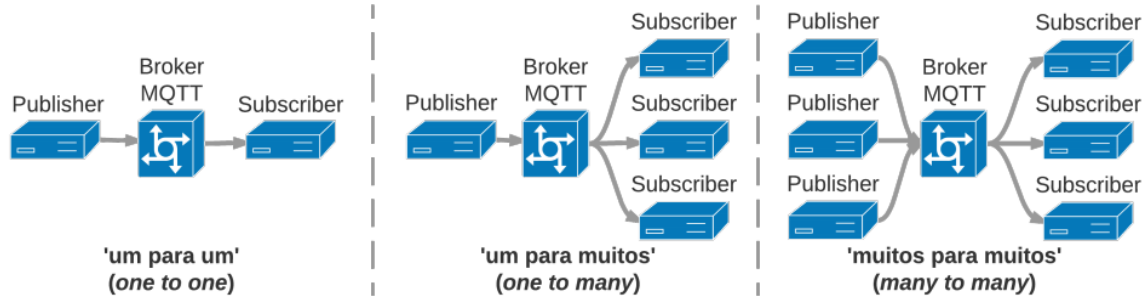
A inclusão dessas plataformas gerou uma forte tendência no desenvolvimento de soluções para redes IoT, sendo primordial para a sua evolução (BOTTA, 2014).

2.3 PROTOCOLO DE COMUNICAÇÃO

Os protocolos de comunicação entre os dispositivos da rede devem se defrontar com os principais problemas: baixa largura de banda, alta latência e instabilidade na comunicação (ROTTA, 2017).

Dentre os diversos protocolos existentes para IoT, tais como MQTT, CoAP (*Constrained Application Protocol*) e WAMP (*Web Application Messaging Protocol*), o trabalho adotou o MQTT. Este protocolo foi criado em 1999 pela IBM e utiliza um padrão de transmissão de mensagens, o *Publisher-Subscriber* (publicador-assinante) tendo um intermediário nessa transmissão chamado *broker* (BANKS, 2014). Uma vantagem de utilização desse intermediário é que somente a origem (publicador) precisa saber o endereço do *broker* permitindo três métodos de envio como mostra a Figura 3 (TORRES, 2016).

Figura 3 – Métodos de envio de dados suportados pelo MQTT



Fonte: (Torres, 2016)

No primeiro método, *one to one,* há um único *publisher*, o qual envia dados para somente um *subscriber*. Já o segundo método, *one to many*, possui um único *publisher* que envia mensagens para muitos *subscribers*. Por fim, no último método, *many to many*, existem diversos *publishers* que enviam mensagens para muitos *subscribers*.

2.3.1 *Broker*

O *broker* representa o centro da comunicação de qualquer protocolo que utiliza o padrão *Publisher-Subscriber*. A sua principal função é receber as mensagens, filtrá-las e entregá-las aos respectivos destinos, além do gerenciamento de conexões dos clientes principalmente em relação à segurança (HIVEMQ, 2017).

Para esse trabalho será utilizado o *broker* Mosquitto. Mosquitto, que é um *broker* de código-fonte aberto e faz parte do Eclipse IoT Working Group, que consiste em um grupo de companhias que investem e promovem projetos *open source* voltados para IoT. A grande vantagem do Mosquitto aos demais *brokers* é sua execução leve e eficiente em decorrência da sua construção em linguagem C, sendo adequado para uso em todos os dispositivos, desde computadores baixa potência até servidores completos. Desta forma, o *broker* Mosquitto é o mais viável para ser utilizado nas aplicações de IoT utilizando microcontroladores, sistemas embarcados, sensores de baixa potência ou dispositivos móveis.

2.4 TRABALHOS RELACIONADOS

Um trabalho semelhante ao aqui apresentado foi realizado por Torres (2016), nele se faz uma análise do desempenho de um *hardware* de baixo custo, Raspberry Pi 2 Modelo B, utilizando o protocolo MQTT operando em diversos *brokers* - como mostra a Tabela 1.

O cenário adotado, no referido trabalho, foi um *publisher* para muitos *subscribers* (*one-to-many*), operando com o limite máximo de dez mil *subscribers*. Diferentemente, este trabalho utilizou muitos *publishers* para um *subscriber* (*many-to-one*), adotando o máximo de seis mil *publishers*.

Tabela 1 – *Brokers* analisados

Broker	Versão	Linguagem	Implementação	Extras
Mosquitto[9]	1.4.8	C	MQTT 3.1.1	–
Apache ActiveMQ Apollo[10]	1.7.1	Java	MQTT 3.1	Suporte a STOMP, AMQP, MQTT, OpenWire
Mosca[11]	1.1.2	Javascript (node.js)	MQTT 3.1.1	Sem suporte a QoS 2
eMQTT[12]	0.17.0-beta	Erlang	MQTT 3.1.1	Foco em clusterização
Ponte[13]	0.0.16	Javascript (node.js)	Não informado	Realiza ponte entre HTTP (REST), MQTT e CoAP.

Fonte: Torres (2016)

A semelhança entre os trabalhos se observa nas métricas utilizadas para analisar o desempenho do *gateway*, quais foram, o uso de CPU, o consumo de memória e o total de falhas no envio de mensagens.

3 DESENVOLVIMENTO

Nesse capítulo serão apresentados todos os instrumentos utilizados para a realização dos testes, objeto do presente trabalho, tais como *hardwares* e *softwares* utilizados, e configuração inicial do ambiente de teste. Por fim, serão expostos os cenários de testes.

3.1 AMBIENTE DE TESTE

3.1.1 *Hardwares*

Para a realização dos testes foram utilizados como *hardware* o BeagleBone Black e um computador pessoal *desktop*. O BeagleBone Black (especificações na Tabela 1) foi utilizado como *gateway*, processando todo o protocolo MQTT além do *software* collectl, responsável pela coleta de diversas métricas de desempenho.

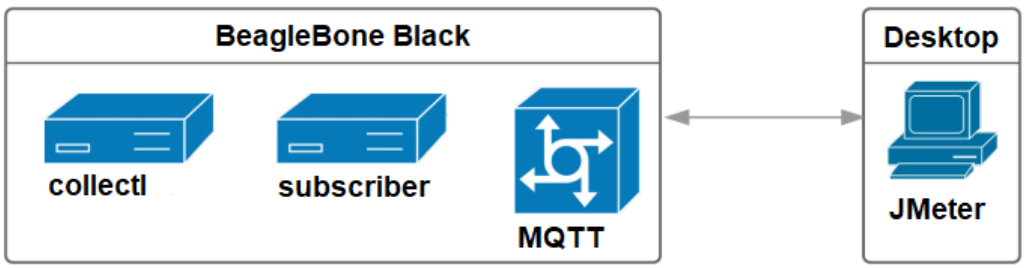
Tabela 2 – Especificações do BeagleBone Black

Processador	AM3358 1GHz ARM Cortex-A8	**Sistema Operacional**	Debian Linux 7.11
Memória RAM	512MB DDR3L 800MHZ	**Rede**	Ethernet 10/100 RJ45
Memória Flash	4GB 8-bit eMMC	**Alimentação**	MicroUSB 5V/1.8A

Fonte: BeagleBone Black System Reference Manual Revision C.1 – Gerald Coley (2014)

No computador pessoal *desktop* foi executado o *software* Apache JMeter, o qual faz uma simulação de carga progressiva.

Figura 4 – Ambiente de teste



Fonte: Autoria própria

3.1.2 *Softwares*

Também foram utilizados *softwares* para a execução dos testes, sendo essencial a compreensão da função de cada um destas ferramentas.

- Mosquitto[1]: inclui um conjunto de códigos de interpretação do *broker*, ou seja, o funcionamento do Mosquitto depende desse pacote;
- Collectl[2]: é uma ferramenta de monitoramento de desempenho do sistema, portanto ela será responsável por coletar os dados de desempenho durante os testes;
- Apache JMeter[3]: é um *software* de código aberto projetado para simular uma carga pesada em um servidor, grupo de servidores, rede ou objeto para testar sua força ou para analisar o desempenho geral sob diferentes tipos de carga. Originalmente foi projetado para testar aplicativos da *Web*, contudo, por ser uma ferramenta gratuita e de código aberto, foi possível desenvolver melhorias, surgindo *plugins* para gerar outros relatórios além dos já oferecidos, bem como outras aplicações além dos aplicativos *Web;*
- *Plugin* "MQTT *Protocol Support"*[4]: é o *plugin* necessário para haver reconhecimento do protocolo MQTT pelo *software* Apache JMeter;
- *Plugin "Stepping Thread Group"*[5]*:* é um *plugin* que oferece uma abordagem simplificada para configurar o planejamento de carga progressiva com o objetivo de manter o nível de simultaneidade.

[1] https://mosquitto.org/ - Acesso em 11/05/2019.
[2] http://collectl.sourceforge.net/ - Acesso em 11/05/2019.
[3] https://jmeter.apache.org/ - Acesso em 11/05/2019.
[4] https://github.com/emqx/mqtt-jmeter - Acesso em 11/05/2019.
[5] https://jmeter-plugins.org/wiki/SteppingThreadGroup/ - Acesso em 11/05/2019.

3.2 CONFIGURAÇÃO DO AMBIENTE

3.2.1 *Mosquitto*

Para a execução do *broker* Mosquitto foi instalado, primeiramente, o pacote essencial “*mosquito”*, pois é onde se encontra a biblioteca principal do Mosquitto, sem a qual não é possível compreender qualquer comando do *broker*. Conjuntamente foi instalado o pacote “*mosquitto clients*”, que fornece um simples método de execução de mensagem utilizando apenas um comando, “mosquitto_pub” para *publisher* ou “mosquitto_sub” para *subcriber*.

3.2.2 *Apache JMeter*

Como o Apache JMeter é uma ferramenta com linguagem base Java, necessita que seja instalado o *Java Development Kit* (JDK), o qual é uma ferramenta que inclui o indispensável para desenvolver e executar aplicações em Java. Além disso, dentro do Apache JMeter é preciso instalar os dois *plugins*, “*MQTT Protocol Support*” e “*Stepping Thread Group*”.

3.2.3 *Collectl*

Foi utilizada a mesma configuração do collectl em todos os testes para a realização de coleta de dados de desempenho. Neste padrão adotado foi executado o comando “collectl –scmt”. O parâmetro “scmt”, utilizado, é o que indica quais as fontes de dados o collectl deve coletar. Neste contexto, cumpre esclarecer o significado de cada argumento que compõe o parâmetro:

- s (*Statistic*): indica que não serão coletadas as fontes padrões e que serão indicadas quais coletar logo após esse argumento;
- c (*CPU*): coleta todas as mudanças dinâmicas do estado de uma CPU;
- m (*Memory*): corresponde às informações detalhadas sobre a utilização da memória;
- t (*TCP*): analisa o tráfego da rede verificando o sucesso ou falha na entrega das mensagens.

3.3 TESTES

Para analisar o desempenho do *hardware* foram realizados três testes. O nível de carga de trabalho progressiva foi o mesmo para todos, sendo a quantidade máxima de 6000 *publishers*. A única diferença entre os testes foi o tempo de carregamento até o nível máximo de carga de trabalho e o tempo que essa iria permanecer ativa.

Em todos os testes o comando utilizado no BeagleBone Black foi o "*mosquitto_sub*", sendo que cada teste foi realizado cinco vezes com a reinicialização do *gateway* e do computador em todos eles.

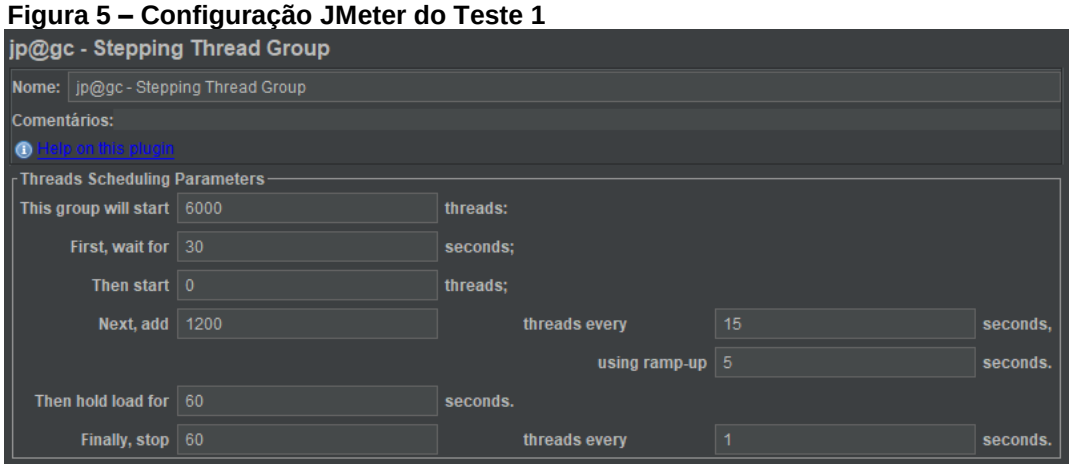
3.3.1 *Teste 1*

A finalidade do primeiro teste foi alcançar o nível máximo de carga no menor tempo. Para tanto, o teste seguiu as seguintes etapas:

1. Iniciar coleta de desempenho;
2. Aguardar 30 segundos;
3. Iniciar com 0 conexões e aumentar 1200 conexões a cada 15 segundos;
4. Ao atingir a carga máxima de 6 mil conexões, mantê-las durante 60 segundos;
5. Encerrar 60 conexões a cada 1 segundo;
6. Encerrar teste.

Para alcançar esse objetivo, o JMeter com o plugin "Stepping Thread Group" foi configurado conforme as etapas acima transcrita:

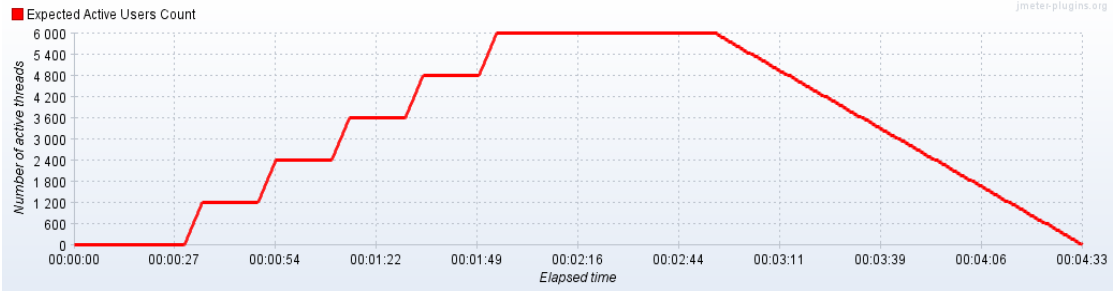
Figura 5 – Configuração JMeter do Teste 1



Fonte: Autoria própria

O tempo total de execução do teste foi de 4 minutos e 33 segundos, como é possível observar na Figura 6.

Figura 6 – Gráfico de tempo de execução do teste 1



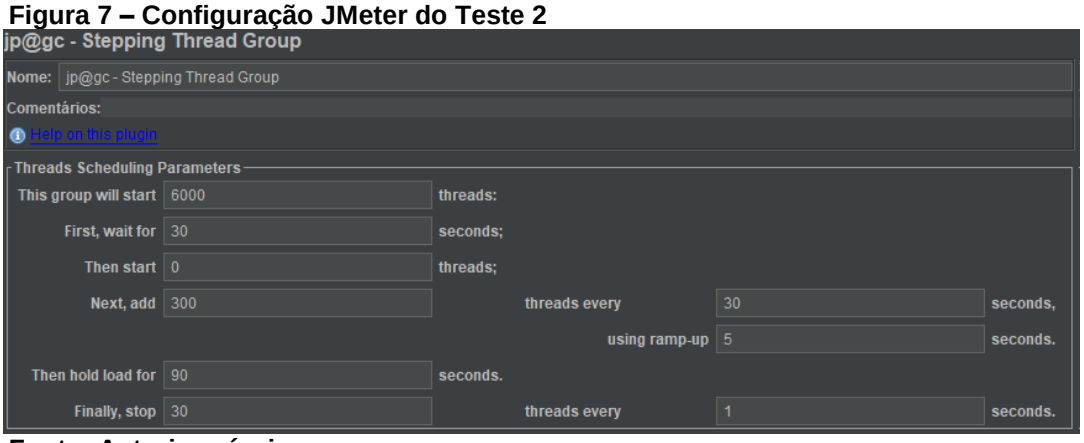
Fonte: Autoria própria

3.3.2 *Teste 2*

No segundo teste a intenção foi simular um uso normal do *gateway* mantendo um balanço no total e tempo de conexões. Para isso, o teste seguiu as seguintes etapas:

1. Iniciar coleta de desempenho;
2. Aguardar 30 segundos;
3. Iniciar com 0 conexões e aumentar 300 conexões a cada 30 segundos;
4. Ao atingir a carga máxima de 6 mil conexões, mantê-las durante 90 segundos;
5. Encerrar 30 conexões a cada 1 segundo;
6. Encerrar teste.

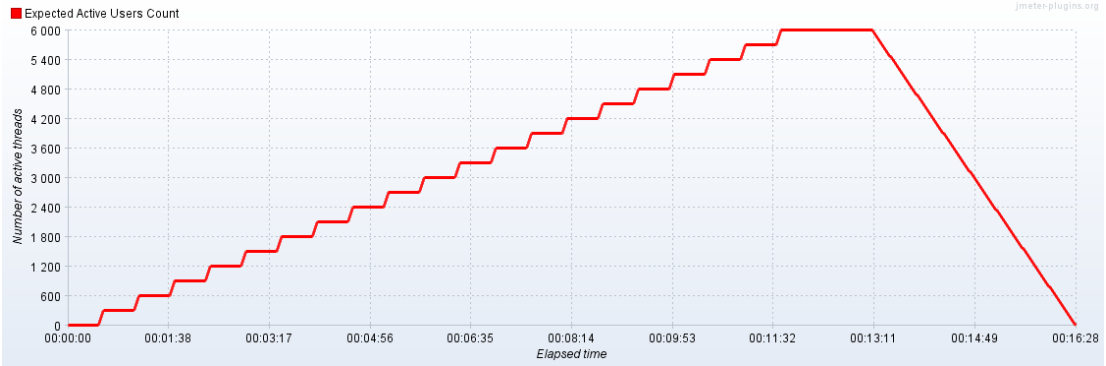
Figura 7 – Configuração JMeter do Teste 2



Fonte: Autoria própria

O tempo total de execução do teste foi de 16 minutos e 28 segundos, como é possível observar na Figura 8.

Figura 8 – Gráfico de tempo de execução do teste 2



Fonte: Autoria própria

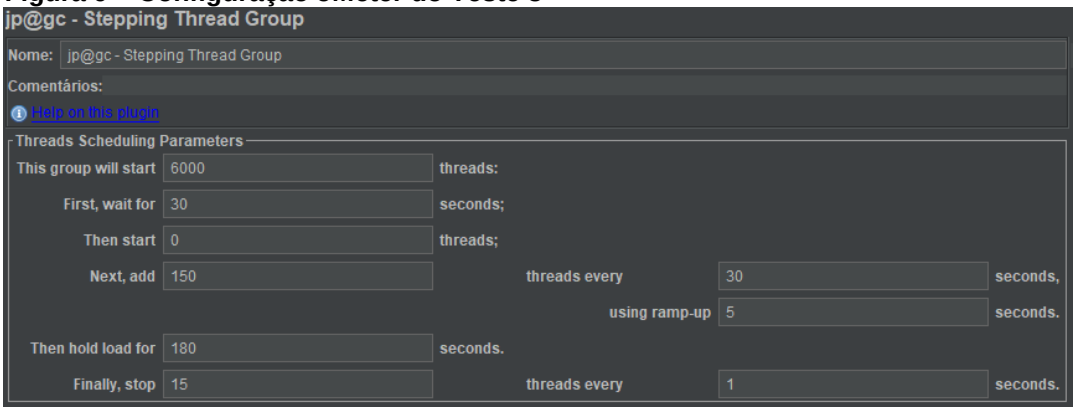
3.3.3 *Teste 3*

Com a finalidade de um teste de larga escala de tempo, tanto para alcançar a carga máxima como mantê-la ativas, o terceiro teste seguiu as seguintes etapas:

1. Iniciar coleta de desempenho;
2. Aguardar 30 segundos;
3. Iniciar com 0 conexões e aumentar 150 conexões a cada 30 segundos;
4. Ao atingir a carga máxima de 6 mil conexões, mantê-las durante 180 segundos;
5. Encerrar 30 conexões a cada 1 segundo;
6. Encerrar teste.

Para execução do terceiro teste, o JMeter com o plugin “Stepping Thread Group” foi configurado da seguinte maneira:

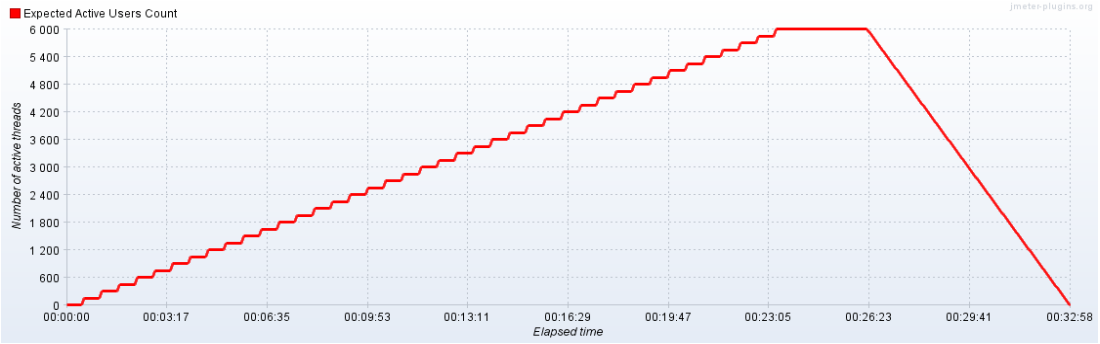
Figura 9 – Configuração JMeter do Teste 3



Fonte: Autoria própria

O tempo total de execução do teste foi de 32 minutos e 58 segundos, como é possível observar na Figura 10.

Figura 10 – Gráfico de tempo de execução do teste 3



Fonte: Autoria própria

4 RESULTADOS

Os resultados foram obtidos por meio da coleta de dados de desempenho realizado pelo collectl. Para cada teste foi gerado um log específico, o qual foi divido nas três categorias já tratadas neste trabalho, quais sejam, CPU (processador), *Memory* (memória) e TCP (rede). Os resultados obtidos em cada segundo do teste foram dispostos nas linhas do log gerado, como mostra a figura abaixo.

Figura 11 – Log gerado pelo collectl

```
#<--------CPU--------><-----------Memory-----------><------------TCP------------>
#cpu sys inter  ctxsw Free Buff Cach Inac Slab  Map PureAcks HPAcks   Loss FTrans
   5   0    50     52 312M  17M  67M  61M  14M 100M        1      0      0      0
   5   0    47     54 312M  17M  67M  61M  14M 100M        1      0      0      0
   5   0    62     60 312M  17M  67M  61M  14M 100M        1      0      0      0
   5  10   354    399 312M  17M  67M  61M  14M 100M        0      1      0      0
   6   1    50     55 312M  17M  67M  61M  14M 100M        1      0      0      0
   5   0    51     59 312M  17M  67M  61M  14M 100M        1      0      0      0
   5   1    44     49 312M  17M  67M  61M  14M 100M        1      0      0      0
   5   0    55     59 312M  17M  67M  61M  14M 100M        1      0      0      0
  15   5   128    158 312M  17M  67M  61M  14M 100M        1      0      0      0
   4   0    48     56 312M  17M  67M  61M  14M 100M        1      0      0      0
   5   0    51     56 312M  17M  67M  61M  14M 100M        1      0      0      0
   5   0    54     65 312M  17M  67M  61M  14M 100M        0      1      0      0
   5   1    55     48 312M  17M  67M  61M  14M 100M        1      0      0      0
```

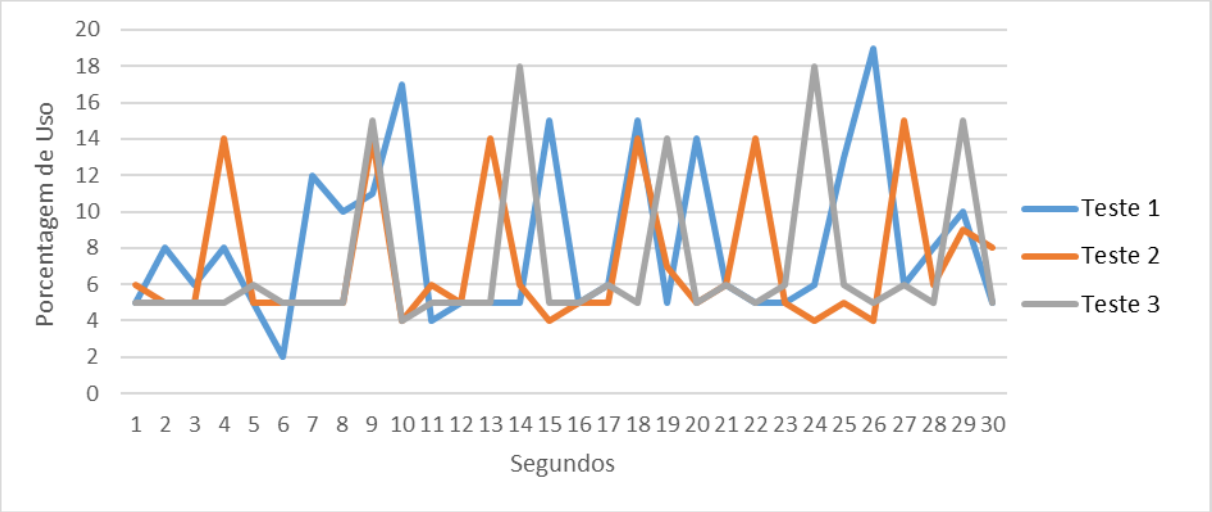
Fonte: Autoria própria

Como visto, em razão dos inúmeros dados, os logs se apresentaram extensos e com certa complexidade. Assim, para que fosse possível uma melhor compreensão do resultado, foram gerados gráficos a partir dos resultados obtidos.

4.1 PROCESSADOR

Logo no começo dos testes, o processador apresentou variação na sua carga de processamento, como mostra Figura 12. Destaque-se que, neste primeiro momento, o BeagleBone ainda não estava recebendo os pacotes enviados pelo *publisher*, mas somente executando o *collectl* e o *mosquitto clientes*.

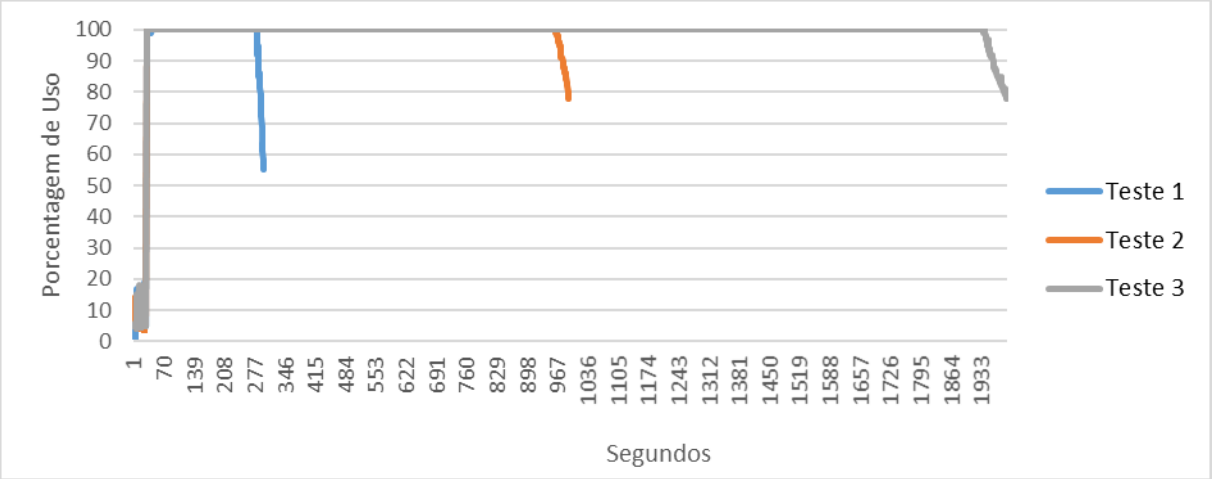
Figura 12 – Log início do uso do processador



Fonte: Autoria própria

Após o recebimento do primeiro pacote, a carga de processamento deslocou-se para o valor máximo e, posteriormente, ainda que permanecesse recebendo pacotes, a carga máxima se manteve.

Figura 13 – Log geral do uso do processador



Fonte: Autoria própria

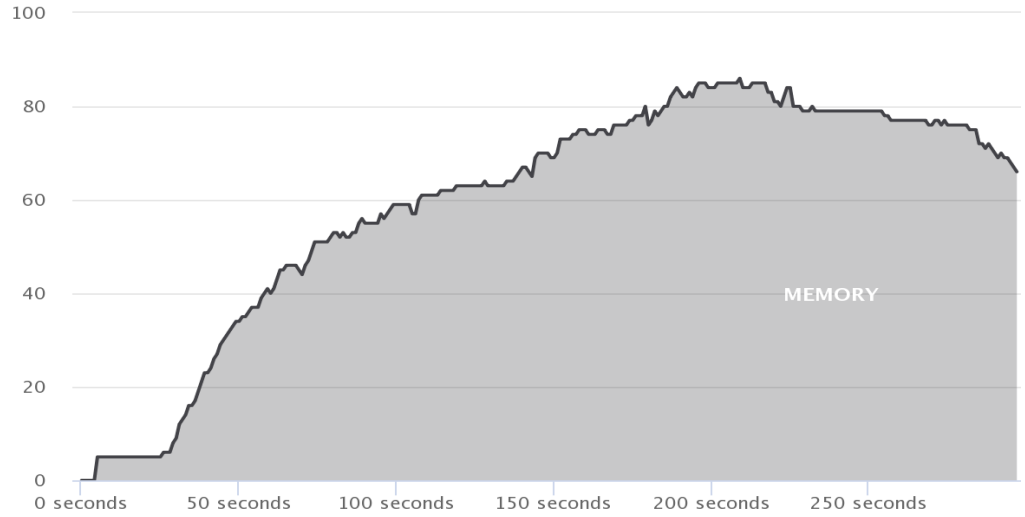
A carga de processamento somente voltou a decair no momento em que se deu início ao encerramento das conexões.

4.2 MEMÓRIA

Nos testes realizados, a memória alcançou o pico de uso entre 180 a 200 segundos de execução, não ultrapassando 100 MBytes de uso.

No primeiro teste, após 5 segundos teve um imperceptível aumento de consumo de memória alcançando 5 MBytes. Esse consumo aumentou somente após o início de recebimento das mensagens dos *publishers,* sendo o consumo máximo de 87 MBytes, como mostra a figura abaixo.

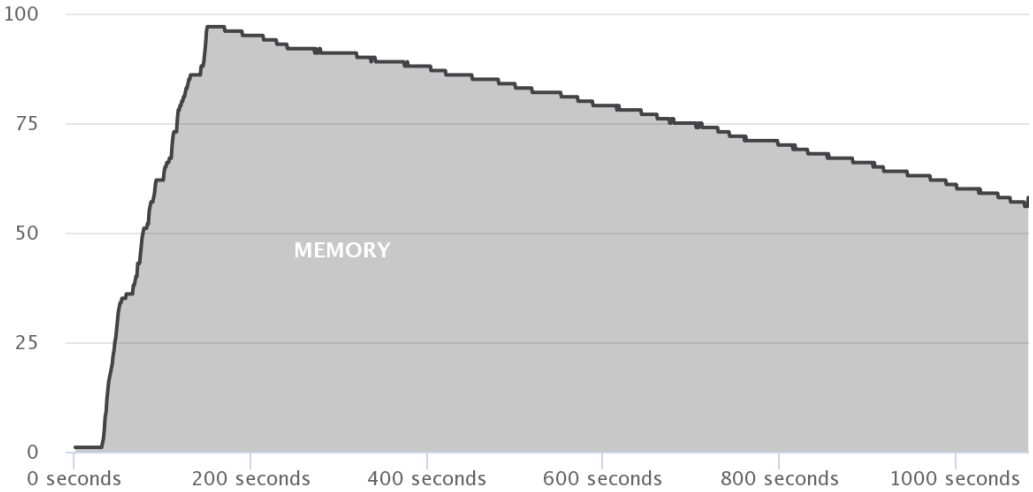
Figura 14 – Consumo de Memória Teste 1



Fonte: Autoria própria

Já no segundo teste, o consumo de memória teve seu máximo de 97 MBytes alcançado entre 151 a 170 segundos. Após esse período de tempo, o consumo começou a diminuir até 58 MBytes, mesmo com o encerramento de todas as conexões.

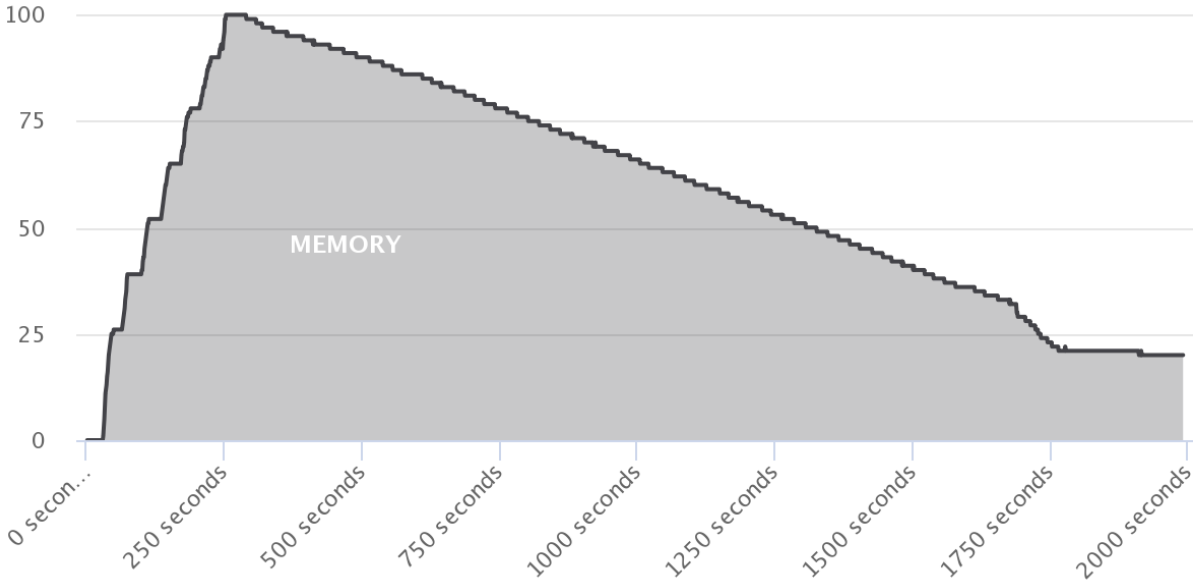
Figura 15 – Consumo de Memória Teste 2



Fonte: Autoria própria

Por fim, o último teste teve um crescimento gradual de consumo de memória, vez que a cada 30 segundos aumentava média de 14 MBytes. O seu maior consumo foi atingindo aos 253 segundos de testes, o qual foi 100 MBytes. Então, lentamente o consumo foi decaindo, e quando chegou a 29 minutos e 14 segundos de teste estabilizou em 20 MBytes de consumo mesmo com o encerramento das conexões.

Figura 16 – Consumo de Memória Teste 3



Fonte: Autoria própria

4.3 REDE

Por fim, no tocante à rede, em todos os testes, esta não demonstra qualquer perca de pacotes em seu tráfego. Diante deste quadro, não se presumiu a necessidade de transformar as informações obtidas em gráficos.

5 CONCLUSÃO

Devido a diversidade de benefícios e o forte crescimento nas redes IoT, a confiabilidade e o forte desempenho nesses cenários são indispensáveis para se criar uma implementação segura, rápida e de baixo custo. Levando em consideração essa necessidade, o presente trabalho buscou analisar o desempenho de um *hardware* de baixo custo, o BeagleBone Black, utilizando um protocolo de comunicação que atendesse a esses requisitos, o MQTT.

Da análise dos resultados dos testes, foi possível concluir que o *hardware* escolhido atendeu a todas as expectativas nos três cenários testados, os quais empregaram a quantidade máxima de 6000 *publishers*, diferenciando o tempo para alcançar essa quantidade.

No quesito confiabilidade, destacou-se o tráfego de mensagens na rede, que foram entregues todos os pacotes durante os três testes. Este é um ponto crucial, pois se existir a vazão de pacotes na troca de mensagens em uma rede, pode ocasionar inconsistências de dados, lentidão na transmissão de dados, colisão de pacotes e tempo de espera de resposta elevado.

Em desempenho, percebeu-se que o CPU ficou em seu uso total durante todas as trocas de mensagens, situação que poderia ocasionar problemas no tráfego. Todavia, no caso do cenário observado no trabalho, a memória estava disponível para auxiliar a CPU.

A memória ficou ociosa até o CPU necessitar de mais processamento, atingindo no máximo 100 MBytes dos seus 512 MBytes, situação que, todavia, permaneceu por poucos segundos. Mesmo com o risco da CPU estar trabalhando no seu limite, a memória RAM não ultrapassou 20% (vinte por cento) de uso, criando-se um cenário de grande desempenho.

Tomando por base os resultados obtidos nos cenários propostos, foi possível concluir que o BeagleBone Black é uma ótima escolha como *gateway* em uma rede IoT utilizando protocolo MQTT.

Embora o presente trabalho tenha respondido as dúvidas inicialmente formuladas acerca do uso do BeagleBone Black como gateway de rede de IoT, outras questões ainda precisam ser esclarecidas como a possibilidade de ampliar a quantidade de *publishers* a fim de viabilizar o uso do BeagleBone Black em maiores

e diferentes cenários, a existência de melhores configurações que auxiliem na obtenção de um melhor desempenho buscando uma folga no uso de CPU e/ou memória, o desempenho de outros *hardwares* de baixo custo nos cenários propostos por este trabalho, bem como o desempenho do BeagleBone Black nos cenários propostos pelo trabalho, mas utilizando um protocolo diferente.

REFERÊNCIAS

AMAZON. **AWS IoT: Serviços de IoT para soluções industriais, comerciais e de consume**. 2016. Disponível em: <https://aws.amazon.com/iot/>. Acesso em: 11 mai. 2019.

ASHTON, K. That "Internet of Things" Thing. **RFID Journal**, 22 jun. 2009. Disponível em: <http://www.rfidjournal.com/articles/view?4986>. Acesso em: 11 mai. 2019.

BANKS A. GUPTA, R. MQTT Version 3.1.1. **OASIS Standard.** 29 out. 2014. Disponível em: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>. Acesso em: 11 mai. 2019.

BOTTA, A. DONATO, D. W. PERSICO, V. **On the integration of cloud computing and internet of things**. In: Future Internet of Things and Cloud (FiCloud), 2., 2014, Barcelona. p. 23–30.

EVANS, D. A Internet das Coisas: **Como a próxima evolução da Internet está mudando tudo**, abril 2011. Cisco Internet Business Solutions Group (IBSG). Disponível em: < https://www.cisco.com/c/dam/global/pt_br/assets/executives/pdf/internet_of_things_iot_ibsg_0411final.pdf>. Acesso em: 11 mai. 2019.

FAN, T. CHEN, Y. **A scheme of data management in the Internet of Things**. In: IEEE International Conference on Network Infrastructure and Digital Content, 2., 2010, Beijing. 2010. p. 110 – 114.

FERREIRA, I. V.; et al. **Proposta de um modelo para a aplicação da internet das coisas industrial**. In: XIII Simpósio Brasileiro de Automação Inteligente, 1., 2017, Porto Alegre. PUCRS: Porto Alegre, 2017, p. 1468-1473.

GOOGLE. **Google Cloud Platform.** 2008. Disponível em: <https://cloud.google.com/>. Acesso em: 11 mai. 2019.

HIVEMQ. MQTT Essentials Part 3: **Client, Broker and Connection Establishment**, 22 jan. 2015. Disponível em: <http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment>. Acesso em: 11 mai. 2019.

MANYIKA, J.; et al. **The Internet of Things:** Mapping the value beyond the hype. McKinsey Global Institute, 2015.

MICROSOFT. **Microsoft Azure**. 2010. Disponível em: <https://azure.microsoft.com>. Acesso em: 11 mai. 2019.

MODOFF, M. BHAGAVATH, V. CLIFTON, K. The Internet of Things: Not quite the Jetsons yet, but places to look. **Deutsche Bank - Markets Research**. 6 mai. 2014. Disponível em: <https://www.deutschebank.nl/nl/docs/The_internet_of_things.pdf>. Acesso em: 11 mai. 2019.

ROTTA, G. CHARÃO, A. DANTAS, M. **Um estudo sobre protocolos de Comunicação para ambientes de Internet das Coisas**. 2017. Departamento de Informática e Estatistica (INE) - UFSC. Departamento de linguagens e Sistemas de Computação (DLSC) - UFSM.

SINGH, D. TRIPATHI, G. JARA, J. A. **A survey of Internet-of-Things: Future vision, architecture, challenges and services**. In: IEEE World Forum on Internet of Things (WF-IoT), 2014, Seoul. Seoul Olympic Parktel Hotel, 2014. p. 287–292.

TORRES, B. A. ROCHA, R. A. SOUZA, N. J. **Análise de Desempenho de Brokers MQTT em Sistema de Baixo Custo**, Fortaleza, 2016. Grupo de Redes de Computadores, Engenharia de Software e Sistemas (GREat) - Universidade Federal do Ceará.

WESTERBERG, H. TORNQVIST, S. **Tablespoon - real-time system metric monitoring for Karamel**. 2016. 50f. Trabalho de Conclusão de Curso (Especilização) - KTH Royal Institute of Technology, Estocolmo.

ZARGHAMI, S. **Middleware for Internet of Things.** 2013. 87f. Dissertação (Mestrado) - University of Twente, Enschede, 2013.

ZHOU, B. BUYYA, R. Augmentation Techniques for Mobile Cloud Computing: A Taxonomy, Survey, and Future Directions. **ACM Computing Surveys (CSUR)**, Nova Iorque, v. 51, n. 13, abr. 2018.