UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS DE TELECOMUNICAÇÕES

FLAVIO HENRIQUE LEAL RIBEIRO

DETECÇÃO DE DISPOSITIVOS WI-FI UTILIZANDO RASPBERRY PI

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2018

FLAVIO HENRIQUE LEAL RIBEIRO

DETECÇÃO DE DISPOSITIVOS WI-FI UTILIZANDO RASPBERRY PI

Trabalho de Conclusão de Curso de Graduação, apresentado ao Curso Superior de Tecnologia em Sistemas de Telecomunicações, do Departamento Acadêmico de Eletrônica, da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof.ª Dra. Tânia Lucia Monteiro

CURITIBA
2018

TERMO DE APROVAÇÃO

FLAVIO HENRIQUE LEAL RIBEIRO

DETECÇÃO DE DISPOSITIVOS WI-FI UTILIZANDO RASPBERRY PI

Este trabalho de conclusão de curso foi apresentado no dia 10 de dezembro de 2018, como requisito parcial para obtenção do título de Tecnólogo em Sistemas de Telecomunicações, outorgado pela Universidade Tecnológica Federal do Paraná. O aluno foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. Dr. Danillo Leal Belmonte
Coordenador de Curso
Departamento Acadêmico de Eletrônica

_____
Prof. Me. Sérgio Moribe
Responsável pela Atividade de Trabalho de Conclusão de Curso
Departamento Acadêmico de Eletrônica

BANCA EXAMINADORA

_____
Prof. Dr. Kleber Kendy Horikawa Nabas
UTFPR

_____
Prof. Me. Nelson Garcia De Paula
UTFPR

_____
Prof.ª Dra. Tania Lucia Monteiro
Orientadora - UTFPR

“A Folha de Aprovação assinada encontra-se na Coordenação do Curso”

Dedico este trabalho a minha família e todos que me apoiaram em seu desenvolvimento.

AGRADECIMENTO(S)

Agradeço aos meus colegas de trabalho Edislau, Júlio, Estevão e Anderson pelo suporte e fornecimento de materiais, equipamentos e cabos que me auxiliaram no desenvolvimento prático do projeto. Agradeço também a empresa Oi pelo tempo que pude dedicar a este trabalho durante minha jornada diária de trabalho. À professora Tânia Monteiro pelo apoio como orientadora. E também a meus familiares pelos incentivos e constante apoio.

“Quando algo é importante o suficiente, você o faz mesmo que as probabilidades não estejam a seu favor.”

(Elon Musk)

RESUMO

RIBEIRO, Flavio H. L. **Detecção de dispositivos Wi-Fi utilizando Raspberry Pi.** 2018. 56 f. Trabalho de Conclusão de Curso (Curso Superior de Tecnologia em Sistemas de Telecomunicações), Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2018.

Com o crescente número de dispositivos sem fio a necessidade de ferramentas que ajudem a gerenciar, mensurar ou simplesmente detectar tais dispositivos se faz cada vez mais necessária. Este projeto visa criar um equipamento que detecte qualquer dispositivo dentro da sua área de alcance que possua uma placa de acesso à rede sem fio padrão IEEE 802.11 ativa, trabalhando na faixa de 2,4 GHz, e registrar o horário que isto ocorreu pela primeira e pela última vez, mesmo que ele não tente estabelecer uma conexão, utilizando a ferramenta TCPDUMP, shell scripts e Python.

Palavras chave: Raspberry. WI-FI. Monitoramento de rede. Wireless. Python.

ABSTRACT

RIBEIRO, Flavio H. L. **Wi-fi device detector using Raspberry Pi.** 2018. 56 f. Trabalho de Conclusão de Curso (Curso Superior de Tecnologia em Sistemas de Telecomunicações), Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2018.

With the increasing number of wireless devices, the necessity for tools that help manage, measure or simply detect those devices is becoming increasingly necessary. This project aims to create an equipment that detects any device inside its range that have an active IEEE 802.11 standard network interface, working on the frequency of 2.4 GHz, and log the first and last time that it happen, even if it did not try to establish a connection, using the capture tool TCPDUMP, shell scripts and Python.

Keywords: Raspberry. WI-FI. Network monitoring. Wireless. Python.

LISTA DE ILUSTRAÇÔES

LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS

AP	Access Point
API	Application Programming Interface
ARM	Advanced RISC Machine
BLE	Bluetooth Low Energy
BSD	Berkeley Software Distribution
BSSID	Basic Service Set Identifier
BYOD	Bring Your Own Device
CPU	Central Processing Unit
CYOD	Choose Your Own Device
DA	Destination Address
DHCP	Dynamic Host Configuration Protocol
DSSS	Direct Sequence Spread Spectrum
FDMA	Frequency Division Multiple Access
FHSS	Frequency-Hopping Spread Spectrum
GNU	GNU's not Unix
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of things
IP	Internet Protocol
ISM	Internet, Scientific and Medical
LAN	Local Area Network
LLC	Logical Link Control
LSB	Linux Standard Base
MAC	Media Access Control
OFDM	Orthogonal Frequency-Division Multiplexing
OSI	Open Systems Interconnection
OTG	On The Go
OUI	Organizationally Unique Identifier
PHY	Physical layer
RAM	Random-Access Memory
RISC	Reduced Instruction Set Computing
RNID	Remote Network Interface Driver
SA	Source Address
SD	Secure Digital
SSH	Secure Shell
STA	Station
TI	Tecnologia da Informação
TSFT	Timing Synchronization Function Timer
USB	Universal Serial Bus
Wi-Fi	Wireless Fidelity
WLAN	Wireless Local Area Network

SUMÁRIO

1 INTRODUÇÃO

Com o crescimento e popularização de dispositivos “*smart*“ como smartphones, tablets, televisores etc. e o grande estudo e evolução da Internet das Coisas (IoT), cada dia mais equipamentos trazem funções de conexão sem fio. “Há hoje mais dispositivos Wi-Fi em uso do que pessoas na Terra, e mais da metade do tráfego da internet mundial trafega por meio de Wi-Fi” (WI-FI ALLIANCE, s.d.).

Redes sem fio estão se tornando o padrão de acesso a redes privadas. Em redes corporativas, com a ascensão das políticas de *Bring Your Own Device* (BYOD), onde funcionários podem levar seus próprios dispositivos (notebooks, tablets, smartphones) para o ambiente de trabalho, criar um ambiente monitorado é de extrema importância para segurança da informação.

Com a miniaturização destes dispositivos, monitorar o fluxo de aparelhos autorizados em ambientes particulares se tornou muito difícil. Um ponto de acesso Wi-Fi convencional já possui métodos de identificar e registrar os aparelhos que se conectam ou tentam se conectar a ele. Porém tal aparelho deve tentar se conectar a este ponto de acesso para que a entrada seja feita no registro.

Este projeto visa criar um equipamento que detecte qualquer dispositivo dentro da sua área de alcance, que possua uma placa de acesso à rede sem fio que siga o padrão do Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) 802.11 ativa, trabalhando na faixa de 2,4 GHz, e registrar o horário que isto ocorreu pela primeira, quando entrou na rede, e pela última vez, quando saiu do alcance, mesmo que ele não tente estabelecer uma conexão.

1.1 TEMA

As redes de computadores já são uma necessidade para o funcionamento da sociedade moderna a anos “Mesmo uma rede local básica pode economizar tempo, dinheiro e muito trabalho de sua empresa” (ALLBUSINESS, 2010). A necessidade de mobilidade e a facilidade de instalação levou ao desenvolvimento de redes sem fio e a criação de padrões que foram adotados por diversas empresas. Nos últimos anos o

crescimento do número de dispositivos sem fio aumentou consideravelmente e a Internet das coisas promete aumentar ainda mais. Assim, novas formas de gerência e segurança de rede precisam ser estudadas para que locais privados controlem quem pode estar interceptando seus dados. Ferramentas para mensurar e assim gerar análises também precisam se adaptar a esta nova realidade, que traz a possibilidade de novos tipos de análises estatísticas.

1.2 PROBLEMA

A falta de uma conexão física faz a identificação e posicionamento de dispositivos sem fio muito difícil. Este trabalho visa uma solução simples e de baixo custo e baixo consumo de energia que identifique se algum dispositivo está em seu alcance. Fornecerá também a resolução dos nomes dos endereços de *Media Access Control* (MAC) para uma possível identificação de equipamento. Esta identificação pode ser utilizada para geração de estatísticas e também identificação de quando determinado equipamento entrou ou saiu de uma rede, facilitando também possíveis buscas por equipamentos perdidos.

1.3 OBJETIVOS

Nesta seção são definidos os objetivos gerais e específicos deste projeto.

1.3.1 Geral

Implementação de um sistema embarcado utilizando um Raspberry Pi Zero W, tcpdump, bash script e Python que detecte a presença de dispositivos com placa de acesso a rede sem fio padrão 802.11 trabalhando na faixa de 2,4 GHz em sua área de alcance.

1.3.2 Objetivos específicos

- Configurar a placa de rede de um Raspberry Pi Zero W em modo de monitoramento;
- Configurar o tcpdump para que capture somente os quadros ethernet de controle;
- Gerar um bash script que alterne entre todos os 11 canais da faixa de 2,4 GHz;
- Criar um código em Python que faça a leitura dos pacotes capturados e gere o log de saída, com o MAC e a data e hora de captura;

1.4 JUSTIFICATIVA

Atualmente, com a popularização do smartphone, um grande número de pessoas carrega em seus bolsos dispositivos que possuem placas de acesso sem fio padrão IEEE 802.11. A Internet das coisas promete trazer ainda mais dispositivos para essa população já muito grande. Estes avanços levaram as empresas a trazer novas formas de mobilizar seu trabalho. Estratégias de mobilidade como a de *Bring Your Own Device* (BYOD), onde o funcionário pode trabalhar de um aparelho próprio, ou *Choose Your Own Device* (CYOD), onde a empresa apresenta ao funcionário algumas opções de aparelhos que ele pode escolher, trazem novas preocupações para a segurança de TI empresarial. Formas de controlar estes acessos à determinadas áreas da empresa se fazem necessários.

Estimar o horário de entrada e saída de determinados dispositivos na proximidade também pode ajudar a localizar, seja por meio de câmeras ou testemunhas, dispositivos perdidos ou furtados.

Estratégias de marketing e análises estatísticas também podem tirar proveito da informação a respeito do número de consumidores de determinadas marcas que circulam por certas áreas. Como por exemplo as lixeiras inteligentes feitas pela empresa Renew, de Londres, que registram a movimentação do público para analisar preferências de movimentação e compra (GOODIN, 2013). Ou a empresa de transporte de Londres, *Transport for London* (TfL), que utilizou técnicas de coleta de

endereços MAC para verificar o trafego de entrada e saída de passageiros nas estações de metrô de Londres (CORFIELD, 2016).

Ao usuário mais com conhecimento técnico, sabe-se da possibilidade de alterar o endereço MAC de seu aparelho, porém à uma grande maioria de usuários o próprio termo MAC é desconhecido. Para aumentar também a privacidade de seus usuários, algumas empresas utilizam processos de randomização de MAC, onde o MAC do dispositivo é alterado cada vez que muda de rede. Porém há registro de que mesmo hoje, poucos fabricantes utilizam esta técnica (CLABURN, 2017) e mesmo esta técnica não está a prova de falhas (VANHOEF, 2016).

Assim, ressalta-se que este trabalho fornece apenas uma análise inicial e, visto o leque de possibilidades demonstrado acima, superficial e aberta para trabalhos futuros, trabalhando com uma realidade que está em constante mudança.

1.5 METODOLOGIA

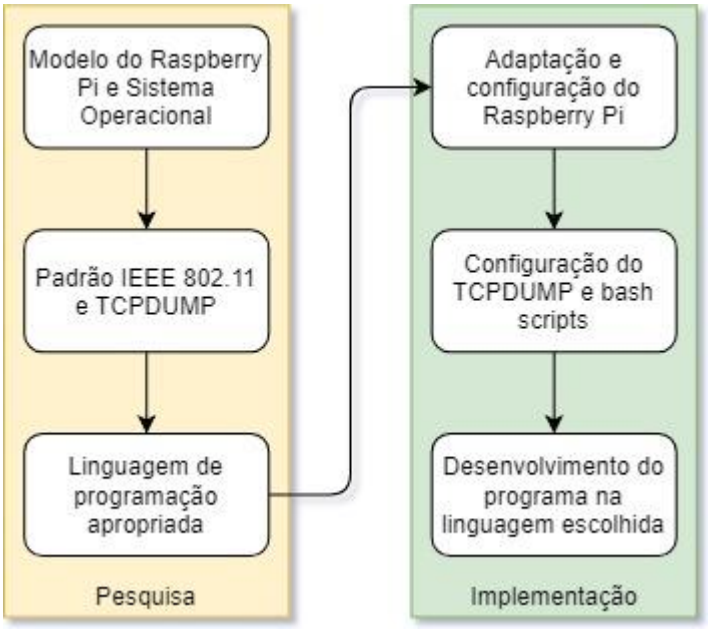
A execução do projeto foi dívida em duas fases: pesquisa e implementação. A pesquisa iniciou-se com o modelo do Raspberry apropriado visando o baixo consumo de energia e o alcance necessário para o bom funcionamento do dispositivo. Nesta etapa, também foram analisados os acessórios necessários para implementação, como adaptadores e cabos necessários. O sistema operacional selecionado foi o Raspbian, um sistema operacional livre baseado na distribuição Debian e otimizado para o hardware do Raspberry (RASPBIAN, s.d.). Foram estudadas as modificações necessárias para os propósitos do projeto.

A pesquisa continua com a compreensão do padrão IEEE 802.11 visando filtrar e analisar os quadros de descoberta de rede. Para a captura de pacotes foi utilizado o software tcpdump, um analisador de pacotes por linha de comando (TCPDUMP, 2010). Nesta etapa, foi estudado o funcionamento e configurações do programa. Após, foi selecionada a linguagem de programação apropriada para a interpretação do arquivo de saída gerado pelo tcpdump, criação de registros e a exportação dos mesmos.

A fase de implementação começa com a compra do hardware e do software, caso seja necessário. Será então configurado o sistema operacional para que atenda

às necessidades do projeto e o analisador de protocolos. Segue, então, a implementação do software. Com o dispositivo e o software devidamente configurados, será feito um primeiro protótipo que será utilizado para os primeiros testes, com poucos dispositivos. Com os resultados destes testes, as correções necessárias serão aplicadas para a execução de testes de estresse. A última parte consiste nos ajustes finais baseados em todos os testes. A figura 1 mostra o diagrama de blocos do projeto.

Figura 1 - Diagrama de blocos do projeto



Fonte: Autoria própria

2 FUNDAMENTAÇÃO TEÓRICA

Até a década de 90, a Internet só era disponível para um número restrito de usuários. No entanto, a diminuição dos custos de equipamentos e o incentivo do governo e de empresas acabou disponibilizando o acesso a milhares de cidadãos, além das diferentes alternativas e formas de conexão introduzidas quando as necessidades dos usuários foram apresentadas (MARQUES, 2006). Além das exigências de maior desempenho e velocidade de tráfego de dados, surgiu-se a necessidade de uma tecnologia que dispensasse o uso de fios para a troca de informações entre computadores: a tecnologia wireless (MARQUES, 2006).

No final de 2010, as assinaturas de telefone celular eram 4 vezes maiores que às de linha de telefone (MSHVIDOBADZE, 2012). A tecnologia 1G foi a primeira desenvolvida (1980) providenciando serviços de voz com base em técnicas analógicas de transmissão de rádio usando a tecnologia *Frequency Division Multiple Access* (FDMA), que trazia limitações no uso de canais (MSHVIDOBADZE, 2012). A segunda geração de telefonia móvel foi caracterizada pela digitalização e compressão da fala, enquanto o 3G passou a oferecer chamadas de vídeo, uso de dados, chamadas de voz por rede etc., já que oferece maior capacidade devido à eficiência de espectro (MSHVIDOBADZE, 2012). Em 2015, as redes de comunicação móvel evoluíram para o 4G. O trafego de dados no futuro (2020) será 1000 vezes maior que o de 2010 por causa do aumento da popularidade dos smartphones, trazendo conexões de dispositivos e de máquinas móveis por wireless (ADACHI, 2016).

O smartphone foi introduzido ao mercado como uma nova tendência de telefone celular, oferecendo mais do que a tecnologia tradicional de mandar mensagens e fazer ligações: fotos, jogos, vídeos, aplicativos em geral e principalmente a oportunidade de se conectar à Internet, pela rede wireless. Pesquisas mostram que 42% dos americanos usam smartphone e que a necessidade de se ter um celular com acesso à rede (navegar, acessar aplicações, fazer downloads) aumentou em mais de 50%, introduzindo redes de alta velocidade e maior disponibilidade de rede Wi-Fi em lugares públicos (SARWAR, 2013). Porém, um dos impactos negativos do crescente uso do smartphone é a segurança de rede: “furos” de segurança podem ser explorados e softwares maliciosos já foram encontrados em vários aparelhos de diferentes vendedores. (SARWAR, 2013).

Os mecanismos de segurança do padrão 802.11 (rede Wi-Fi) devem ter duas funções: limitar o usa da rede para dispositivos autorizados e dispor de privacidade de dados (MULLINER, 2006). No entanto, a mobilidade dos aparelhos aumenta os riscos de roubo de dados (usuários e senhas) e um dos ataques mais comuns quando o smartphone está conectado a uma rede wireless é a obtenção de quem é o dono do celular pelo MAC (MULLINER, 2006). Entretanto, o objetivo básico de uma rede é assegurar que os dados sejam compartilhados com segurança e de maneira confiável (PINHEIRO, 2003). Uma rede de computadores precisa de protocolos para a coordenação entre todos os seus níveis (AMARAL, 2012).

O modelo *Open Systems Interconnection* (OSI) surgiu nos anos 70 com 7 camadas (física, enlace de dados, rede, transporte, sessão, apresentação e aplicação). “O objetivo do modelo OSI é mostrar que é possível facilitar a comunicação entre sistemas diferentes, sem exigir alterações na lógica do hardware e do software subjacentes” (FOROUZAN, 2009, p.17). A primeira camada do modelo OSI, a camada física, fornece as características mecânicas e funcionais para estabelecer uma conexão física de transmissão de bits entre os sistemas ou equipamentos, estabelece as características da interface entre os aparelhos e o meio de transmissão, define esse meio, a codificação dos bits em sinais elétricos ou ópticos, a duração de um bit (taxa de dados), faz a sincronização em nível de bit do emissor e do receptor, determina a topologia física da rede (malha, estrela, anel, barramento) e o modo de transmissão (simplex, half-duplex, full-duplex) (FOROUZAN, 2008).

A segunda camada do modelo OSI é a camada de enlace de dados. Ela é responsável pela divisão do fluxo de bits que vêm da camada 3 (camada de rede) em frames (quadros de gerenciamento), a transferência destes de um hop para outro, adiciona cabeçalhos ao frame para que o emissor/receptor possa identificá-lo, pode implementar um controle de fluxo se os dados transmitidos pelo emissor tem velocidade maior do que os recebidos no receptor, faz controle de erros (detecta e transmite novamente frames danificados ou perdidos) e controle de acesso para dispositivos em um link (FOROUZAN, 2008). Wi-Fi e Ethernet são redes de múltiplo acesso de enlace de dados, por exemplo.

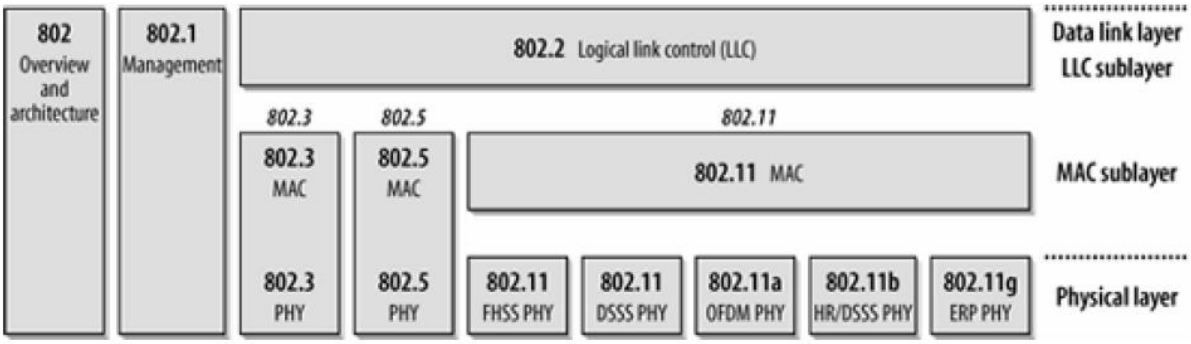
A camada de rede, a terceira do modelo OSI, providencia formatos de pacotes que podem usar diferentes tipos de links de rede para estabelecer uma conexão, além de possuir um esquema de endereçamento para escolher onde o pacote vai quando enviado de um dispositivo final para outro (FALL, 2012). A camada de transporte

permite que os dados sejam transportados da origem ao destino, sem erros e na sequência correta, independente da tecnologia ou topologia usadas nas camadas 1,2 e 3 (FRANCISCATTO et al, 2014). A camada de sessão representa interações entre aplicações, faz o *checkpointing* (salvar arquivos enquanto ainda estão sendo processados) e pode iniciar e recomeçar conexões, enquanto a camada de aplicação é responsável por formatar conversações e codificações para informações (FALL, 2012). Então, é na camada de aplicação que se encontram os aplicativos dos usuários ou dos sistemas, em que esta camada providencia a comunicação entre eles (FRANCISCATTO et al, 2014).

Apesar das origens da comunicação em rede sem fio poderem ser traçadas até a década de 1970, foi a retificação do padrão IEEE 802.11 em 1997 e o subsequente desenvolvimento de certificações de interoperabilidade pela Wi-Fi Alliance que tornou essa uma das tecnologias de mais rápido crescimento do início do século 21 (RACKLEY, 2007, p.1). O padrão IEEE 802.11 define um controle de acesso ao meio (MAC) e as especificações de diversas camadas físicas para conectividade sem fio de estações fixas, portáteis ou móveis dentro de uma rede local (IEEE, 2016, p.122).

Como membro da família de padrões IEEE 802 para redes locais e metropolitanas, o padrão 802.11 interage com a arquitetura, gerenciamento e comunicação entre redes do 802.1 e com a camada de controle de enlace lógico (LLC) do 802.2 (PERAHIA; STACEY, 2008, p.3). Assim, é necessário que o 802.11 apareça para as camadas superiores (LLC) da mesma forma que um equipamento com fio padrão 802, exigindo assim que o padrão 802.11 trabalhe com a mobilidade das estações (STA) dentro da subcamada MAC (IEEE, 2016, p.184). A figura 2 mostra a família de padrões IEEE 802 e sua relação com as camadas do modelo OSI.

Figura 2 - Família 802 e as camadas do modelo OSI.



Fonte: GAST, 2005.

O 802.11 define diversos padrões para a camada física, sendo os mais comuns o *Frequency Hopping Spread Spectrun* (FHSS), o *Direct Sequence Spread Spectrum* (DSSS) e o *Orthogonal Frequency Division Multiplexing* (OFDM) para as frequências de 2,4 GHz, utilizada pelos padrões 802.11b e 802.11g, e 5 GHz, utilizada pelo padrão 802.11a (BULHMAN, CABIANCA, 2016). Estas frequências fazem parte das bandas denominadas *Internet, Scientific and Medical* (ISM) que são bandas utilizadas pela indústria, processos científicos ou por equipamentos médicos (GAST, 2005). O espectro de 2,4 GHz possui 11 ou 14 canais dependendo do país, enquanto o espectro de 5 GHz possuía originalmente 8 canais, porém atualizações no padrão permitiram este espectro a chegar em até 24 canais (MURRAY, DIXON, KOZINIEC, 2007).

Para as estações se comunicarem em uma rede sem fio, inicialmente elas devem encontrar outras estações ou pontos de acesso (AP) e isto é feito através de uma varredura que pode ser passiva ou ativa (BULHMAN; CABIANCA, 2016). A varredura passiva é feita fazendo com que a estação aguarde a transmissão de um quadro *beacon* que é enviado em *broadcast* pelos pontos de acesso a cada 100 ms, exigindo assim que a estação aguarde pelo menos 100 ms em cada canal (MURRAY; DIXON; KOZINIEC, 2007). A varredura ativa faz com que a estação envie um quadro de *probe request* para cada canal e então aguardando um *probe response* por um determinado tempo, caso não tenha resposta o canal é considerado vazio e a varredura passa para o próximo, do contrário a estação reconhece que há um AP neste canal e permanece nele (MURRAY, DIXON, KOZINIEC, 2007).

O Raspberry Pi é um computador do tamanho de um cartão de crédito que pode ser conectado a uma TV e um teclado e utilizado como um computador pessoal

convencional (PC) (RASPBERRY PI, 2017). O Raspian é um sistema operacional otimizado para o Raspberry Pi, criado por um pequeno grupo de desenvolvedores: é ele que faz com que os programas do Raspberry Pi funcionem, além de iniciar com mais de 35000 pacotes de Raspbian (RASPBIAN, 2017).

3 DESENVOLVIMENTO E ANÁLISE DOS RESULTADOS

3.1 PESQUISA

3.1.1 Raspberry Pi e Raspbian

Este projeto, desde em sua concepção, visava criar um sistema embarcado. Para tanto, visto que o foco do projeto não era o desenvolvimento de hardware, a utilização de um computador de placa única com o menor tamanho e menor custo possível foi planejado. Por mais que o mercado disponibilize diversas opções de computador que se encaixam nessas alternativas, nenhum possui o suporte que o Raspberry Pi possui, em especial no Brasil.

Com a marca já escolhida, faltou então determinar dentre os modelos de Raspberry Pi, qual seria o mais adequado. Os modelos mais encontrados no mercado brasileiro no final de 2017 foram: Raspberry Pi 3 Model B (Figura 3) e o Raspberry Pi Zero W (Figura 4). Alguns modelos mais antigos ainda eram encontrados, mas a fim de utilizar um sistema com suas atualizações em dia, foram comparados somente os dois modelos citados.

Figura 3 - Raspberry Pi 3 Modelo B

Fonte: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

Figura 4 - Raspberry Pi Zero W

Fonte: https://www.raspberrypi.org/products/raspberry-pi-zero-w/

As principais características consideradas foram conectividade, processamento, consumo de energia e preço. Memória e outras funcionalidades não foram tão relevantes, pois mesmo o modelo com menos memória, o Zero W, que contém apenas 512 MB de Random Access Memory (RAM), já possuía mais que o suficiente para o projeto.

No quesito conectividade, o modelo 3 B possui uma flexibilidade muito maior. Possuindo 4 portas *Universal Serial Bus* (USB), uma interface Fast Eternet, Bluetooh Low Energy (BLE) e *Wireless Local Area Network* (WLAN) contra apenas uma entrada micro USB, WLAN e BLE do modelo Zero W. Para este projeto, já que em seu planejamento ele visa ser um sistema embarcado, a conectividade apresentada pelo modelo Zero W é suficiente.

Em se tratando de processamento, o modelo 3 B também traz maior poder. Possui um processador 64bit de quatro núcleos de 1.2 GHz contra o processador núcleo único de 1 GHz do Zero W. No entanto este maior poder de processamento também traz um maior gasto de energia. Conforme testes feitos por Eames (2017), em estado ocioso, o Pi 3 B consome quase o dobro de corrente que o Zero W (Figura 5).

Figura 5 - Corrente utilizada pelo Raspberry Pi (em 5,19 V)

	Zero	Zero W	A+	A	B+	B	Pi2B	Pi3B
	/mA	/mA	/mA	/mA	/mA	/mA	/mA	/mA
Idling	100	120	100	140	200	360	230	230
Loading LXDE	140	160	130	190	230	400	310	310
Watch 1080p Video	140	170	140	200	240	420	290	290
Shoot 1080p Video	240	230	230	320	330	480	350	350

Fonte: EAMES, 2017

O script a ser executado, sem a resolução de nomes dos fabricantes, assunto que será tratado na subseção do script, não utiliza muito processamento. O poder extra do modelo 3 B poderia servir à expansão de projetos futuros, mas para o escopo atual, o processamento do modelo Zero W é suficiente.

Por fim, ao se considerar o preço, o modelo Zero W tem a vantagem, custando aqui no Brasil aproximadamente R$ 100,00, enquanto o modelo 3 B custa praticamente o dobro, saindo em uma faixa de R$ 195,00. Estes preços têm como base o distribuidor oficial aqui no Brasil do Raspberry Pi, a loja FilipeFlop, em novembro de 2017. Sendo assim o modelo escolhido foi o Raspberry Pi Zero W.

Em se tratando de sistema operacional, salvo possíveis pesquisas específicas, o mais recomendado para os computadores Raspberry Pi é o Raspbian. O Raspbian é uma versão não oficial do Debian otimizada para trabalhar com o conjunto de instruções *Advanced RISC Machine* (ARM) "*hard-float*" utilizado pelos computadores Raspiberry Pi, o que fornece um ganho de desempenho através do uso de instruções avançadas do processador ARMv6 do Raspberry Pi (RASPBIAN, 2017).

O site oficial da Raspberry Pi Foundation, fabricante do computador, fornecia duas alternativas em 2017: o Raspbian Stretch Desktop e o Raspbian Stretch Lite. A versão Desktop traz uma interface gráfica, além de alguns programas recomendados pela Raspberry Pi Foundation para pesquisa e aprendizado. A versão Lite não possui interface gráfica e é uma versão mais limpa, trazendo apenas o necessário para o sistema operacional utilizar as funções mais comuns do computador, como drivers de comunicação. Para a implementação do projeto, utilizaremos a versão lite por não existir uma interface de usuário final planejada para o sistema. Visto também que estamos utilizando um modelo de menor poder computacional, a versão Lite é a recomendada já que não traz aplicações desnecessárias.

3.1.2 Tcpdump e frames de gerência do padrão IEEE

O tcpdump é um poderoso analisador de pacotes por linha de comando (TCPDUMP, 2010) distribuído sob a licença *Berkeley Software Distribution* (BSD). Ele captura pacotes que trafegam na rede que equivalham a uma expressão usada de filtro, podendo ser especificada a interface a ser observada entre outras opções. O tcpdump é o melhor analisador de tráfego em modo texto que existe. Ele é baseado na *libcap*, uma poderosa API para a captura de pacotes de redes durante seu tráfego. (MOTA FILHO, 2013, p 37)

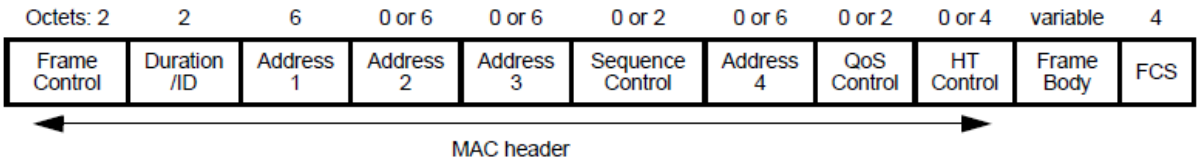
Foram então estudadas estas configurações para verificar o que seria necessário para capturar as informações dos quadros de gerência do padrão 802.11 que o projeto exige.

Ao capturar um pacote, a opção –e do tcpdump faz com que ele registre também dados do cabeçalho da camada de enlace em cada pacote capturado, permitindo assim o registro de endereços MAC. Foi utilizada também a opção –i para indicar qual interface deveria ser monitorada, neste caso a interface wireless, e as opções –U para imprimir os pacotes somente após terem sido capturados por completo e –tttt para imprimir a data em um formato mais completo, que inclua data e hora.

Para finalizar a configuração do tcpdump então restou apenas a expressão a ser utilizada, e para isso foi necessária uma análise de como os quadros de gerência do protocolo 802.11 são identificados.

Um frame conforme o padrão 802.11 possui o formato demonstrado na figura 6.

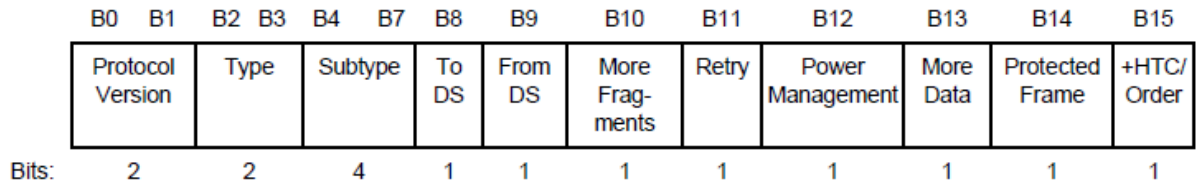
Figura 6 - Quadro padrão IEEE 802.11



Fonte: IEEE, 2016

O subcampo de controle (*Frame Control*) segue o padrão demonstrado na figura 7.

Figura 7 - Campo *Frame Control* de um quadro 802.11



Fonte: IEEE, 2016

Focando somente aos campos relevantes à pesquisa, observou-se que os frames são identificados pelos bits 2 a 6 de cada quadro, o tipo (*Type*) e o subtipo (*Subtype*). Conforme a documentação do padrão 802.11, os frames de gerência são identificados pelos bits 00. Sendo assim os quadros que nos interessam são os que possuem o tipo 00. Entre os quadros de gerência, três subtipos nos interessam: o *beacon*, o *probe request* e o *probe response*.

Beacon é o quadro de anúncio enviado pelos pontos de acesso para divulgar sua rede. Estes quadros são identificados pelo subtipo 1000. *Probe request* são os quadros enviados pelas estações a procura de um ponto de acesso. São identificados pelo subtipo 0100. E por fim temos os quadros de *probe response*, que são os quadros enviados pelos pontos de acesso em resposta aos quadros de *probe request* das estações. Estes são identificados pelo subtipo 0101.

O tcpdump separa os dados de cada pacote ou, neste caso, de cada quadro capturado de forma que fique mais fácil a filtragem dos dados necessários. Verificando então que ele cria um vetor para os bytes que representam um quadro ethernet padrão 802.11 chamado wlan, podemos pegar facilmente o primeiro byte, posição 0 do vetor, que ele conterá o tipo, juntamente com o subtipo. Neste byte também estará a versão do protocolo (campo *Protocol version* na figura 6). A versão atual do protocolo é a 00 e outros valores para estes bits estão reservados para o futuro (IEEE, 2016). Os bytes relevantes que devem ser capturados são então aqueles que possuem a combinação da versão com o tipo, todos em 0, mais o subtipo.

Deve-se ficar atendo também à ordenação dos bytes que o padrão 802.11 segue na hora de transmitir a informação pela rede. A ordenação utilizada é a *little-endian* onde os bytes de menor valor são transferidos primeiro. Assim, o primeiro byte a ser recebido contém o subtipo, seguido do byte que trará o tipo e a versão. Para

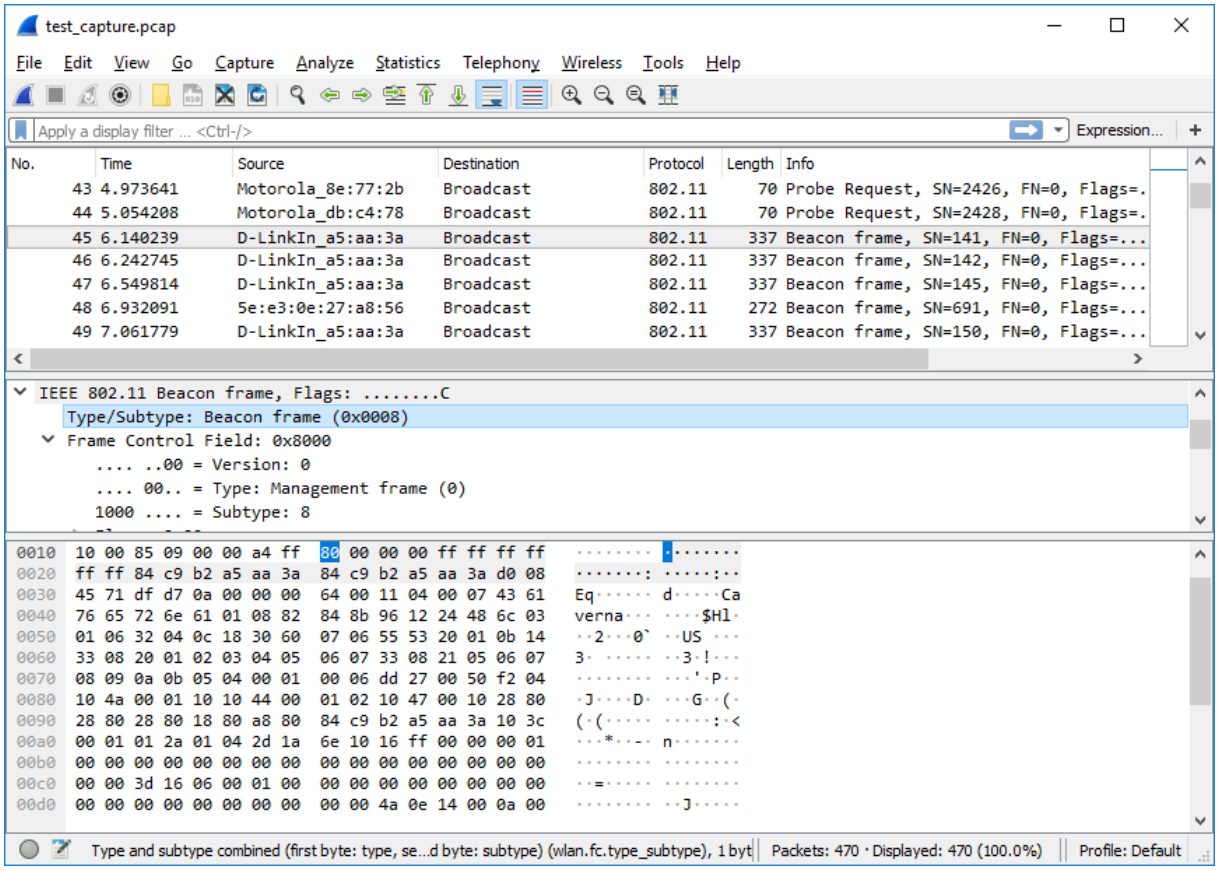
facilitar a visualização os valores são tratados no formato hexadecimal, resultando na seguinte tabela:

Tabela 1 - Versão, tipo e subtipo de um quadro 802.11

Subtipo	Binário	Hexadecimal	Little-endian binário	Little-endian hexadecimal
Beacon	0000 1000	0x08	1000 0000	0x80
Probe request	0000 0100	0x04	0100 0000	0x40
Probe response	0000 0101	0x05	0101 0000	0x50

Com estes dados então foi feito o filtro necessário para a captura dos quadros relevantes utilizando uma operação lógica ou (em inglês *or*), resultando na expressão: wlan[0]=0x80 or wlan[0]=0x40 or wlan[0]=0x50. A figura 8 mostra um exemplo de quadro *beacon* capturado e visto no programa Wireshark.

Figura 8 - Exemplo de quadro de beacon.



Fonte: Autoria própria.

3.1.3 Definindo a linguagem de programação

Para a definição da linguagem de programação apropriada, foi levantado um algoritmo básico do que o script deverá fazer:

- Ler pacote capturado pelo tcpdump (em formado de texto);
- Extrair informação exigida (data, horário e MAC de origem);
- Gravar informação em modo texto com formatação apropriada para leitura.

Como as informações estarão todas em modo texto, uma linguagem que trabalhe bem e possua bibliotecas criadas para trabalhar com variáveis do tipo cadeia de caracteres (*strings*) seria a mais adequada. Visto que não há operações muito complexas a serem feitas, por mais que o número de operações por segundo vá ser alto, uma linguagem interpretada poderia ser utilizada. Assim, levando-se em consideração que o Raspbian vem por padrão com suporte à linguagem Python, esta foi a linguagem escolhida, por sua simplicidade e eficiência na hora de programar.

3.2 IMPLEMENTAÇÃO

3.2.1 Preparando o Raspberry

A instalação do sistema operacional, adquirido diretamente da página do fabricante do Raspberry Pi, foi instalado em um cartão de memória micro SD (do inglês *Secure Digital*) de 8 GB utilizando o software Etcher, um aplicativo gratuito para gravação de imagens em cartões de memória e pen drives USB. Algumas modificações foram necessárias para se iniciar o Raspberry sem a utilização de um monitor (em inglês *headless start*). Para isso foi criado um arquivo, em um computador desktop utilizando Windows, chamado wpa_supplicant.conf e nele foi configurada uma rede sem fio que ajudaria na configuração inicial do Raspberry. O conteúdo deste arquivo pode ser observado no quadro 1. Este arquivo foi então movido para a pasta boot do cartão de memória. Também foi criado um arquivo em branco chamado ssh nesta mesma pasta para sinalizar o Raspberry a habilitar o acesso via *Secure Shell* (SSH) quando ligado pela primeira vez.

Quadro 1 - Conteúdo do arquivo wpa_supplicant.conf

```
network={
        ssid="AndroidAP"
        psk="TCCsistel2018"
}
```

Como ponto de acesso foi utilizado meu aparelho de telefone celular, um Samsung Galaxy S5. Através da lista de dispositivos conectados ao ponto de acesso era possível verificar o IP que o Raspberry recebeu do servidor *Dynamic Host Configuration Protocol* (DHCP) do celular. No entanto esta etapa se fez desnecessária visto que o *hostname* padrão do Raspberry foi divulgado na rede.

Era preciso definir ainda um ambiente de desenvolvimento para o projeto. Para isso foi adquirido um Raspberry Pi 3 Model B e instalado a versão Desktop do Raspbian nele. Para facilitar a compreensão, este computador será doravante denominado “pi 3” enquanto o computador que servirá de base para o projeto será denominado “pi zero”. Após feitas as configurações padrões de acesso à internet no pi 3, um simples comando, *ssh pi@raspberrypi* nos permitiu conectar ao pi zero.

A instalação do programa tcpdump foi extremamente simples, exigindo apenas o comando *sudo apt-get install tcpdump* para instala-lo diretamente dos repositórios padrões do Raspbian.

A próxima etapa seria a configuração da placa de rede sem fio do pi zero em modo de monitoramento. Uma placa de rede sem fio verifica todo o tráfego dentro de um determinado canal da faixa de frequência que é configurada a trabalhar e coleta somente aqueles quadros que lhe são pertinentes. O modo de monitoramento faz com que a placa capture todos os quadros trafegando em seu canal, não exigindo nenhuma associação ou autenticação. Importante não confundir este modo, que existe somente para o meio de transmissão sem fio, com o modo promíscuo, que também se trata de uma forma de capturar todos os pacotes em uma rede, mas para isso exige que se esteja conectado diretamente, no caso de rede cabeada, ou associado a um ponto de acesso no caso da rede sem fio.

Para colocar a placa de rede sem fio do pi zero em modo de monitoramento foi preciso configurar uma nova forma de acessar o pi zero pois o ssh estava sendo feito através da placa em questão. Em determinadas configurações é possível utilizar a placa de rede sem fio em modo monitoramento para também funcionar de forma

normal, mas neste caso ela ficaria presa a funcionar em somente um dos 11 canais que pode trabalhar, como o projeto pretende varrer os 11 canais, esta configuração a seguir se fez essencial pois o adaptador de rede ficaria inutilizável para conexões comuns.

Uma alternativa para acessar o pi zero sem monitor foi configurá-lo como um *gadget* em modo *On-The-Go* (OTG). Neste modo é possível configurar a porta micro USB do Raspberry para se comportar como um dispositivo OTG, como por exemplo um adaptador de rede USB, que receba não somente dados mas também alimentação de energia com uma única conexão USB. Assim, ao conectar o pi zero em outro computador ele poderia ser reconhecido como um adaptador de rede, permitindo uma conexão ssh pela porta USB.

Para isso foram alteradas algumas linhas em dois arquivos contidos na pasta boot do cartão de memória. Primeiro foi adicionada uma linha no final do arquivo config.txt com dtoverlay=dwc2, que especifica o driver USB a ser utilizado. Então, no arquivo cmdline.txt, adicionamos em sua única linha de configuração (muito importante que este arquivo contenha apenas uma linha) as seguintes instruções: modules-load=dwc2,g_ether g_ether.host_addr=02:46:c0:a8:0d:02 g_ether.dev_addr=02:46:c0:a8:0d:01. Estas instruções especificam, em ordem, para carregar o modulo ethernet do driver USB, um endereço MAC para a interface de rede criada no host, neste caso o pi zero, e um endereço MAC para o host que se conectar a ele pela USB. Estes endereços MAC são aleatórios para adaptadores de rede USB e esta etapa foi essencial para garantir depois que eles receberiam os *Internet Protocols* (IPs) apropriados que permitiriam a conexão ssh.

Para continuar a configuração do pi zero de forma que funcionasse como um dispositivo OTG, precisamos configurar o IP que será associado a esta rede. Para não depender do dispositivo que será conectado a ele, um servidor DHCP teria que ser configurado no próprio pi zero. Esta conexão pela USB também seria a única forma possível de conectar o pi zero a internet, assim era necessário que o IP do dispositivo fosse sempre o mesmo para que fosse possível criar uma rota padrão (*gateway*) para ele. Para isso foi utilizado o servidor DHCP ISC. Nele foi configurada a rede 192.168.100.0/30 que distribuiria apenas dois IPs, 192.168.100.1 e 192.168.100.2. Como o IP do pi zero seria configurado como estático no IP 192.168.100.1, o IP distribuído pelo servidor DHCP seria sempre o outro da rede /30 mencionada, 192.168.100.2. Para garantir também que o IP seria distribuído de forma certa caso

houvesse alguma desconexão abrupta, o que poderia confundir o servidor DHCP em pensar que aquele IP já estava sendo utilizado, foi fixado também o IP de final 2 no MAC configurado para o dispositivo conectado na USB com o pi zero no arquivo cmdline.txt. Assim, o arquivo de configuração do servidor DHCP /etc/dhcp/dhcpd.conf ficou como demonstrado no quadro 2.

Quadro 2 - Configuração final do arquivo dhcpd.conf

```
ddns-update-style none;
subnet 192.168.100.0 netmask 255.255.255.252 {
        range 192.168.100.2 192.168.100.2;
}
host garden {
        hardware ethernet 02:46:c0:a8:0d:02;
        fixed-address 192.168.100.2;
}
```

E, conforme mencionado, o IP do pi zero seria fixo, e para isso teve que ser configurado em um arquivo próprio para o funcionamento do modo *gadget*: /etc/network/interfaces.d/usbgadget. A configuração deste arquivo pode ser observada no quadro 3.

Quadro 3 - Configuração do arquivo usbgadget

```
allow-hotplug usb0
iface usb0 inet static
        address 192.168.100.1
        netmask 255.255.255.252
```

Por fim, nega-se a interface no arquivo de DHCP padrão do Raspbian para evitar conflitos. Isto foi feito adicionando a linha denyinterfaces usb0 no arquivo /etc/dhcpcd.conf.

Com estas configurações o acesso ao pi zero poderia ser feito em qualquer computador com suporte a adaptadores de rede USB apenas enviando uma conexão ssh ao IP fixo definido anteriormente e com o usuário padrão do Raspbian: *ssh pi@192.168.100.1*. Um passo adicional foi necessário para se trabalhar em máquinas utilizando o sistema operacional Windows. Nestas foi preciso instalar uma versão específica do driver proprietário da Microsoft RNID (do inglês *Remote Network Interface Driver*). Após esta etapa, qualquer programa que permita estabelecer uma conexão ssh funcionaria, no caso deste projeto foi utilizado o Putty.

Coma conexão estabelecida através da USB, a placa wireless ficou disponível para o modo monitoramento. E foi nesta etapa que o projeto atingiu uma complicada

barreira. O driver padrão do Raspberry Pi, contido no Raspbian, não possui suporte ao modo de monitoramento. Para isso foi necessário encontrar um driver alternativo para o chip bcm43430a1 fabricado pela Broadcom Corp. utilizado pelos modelos Raspbery Pi 3 Model B e Raspberry Pi Zero W. Nesta busca encontramos o Nexmon, um framework para modificação de firmwares para chips WiFi Broadcom/Cypress que permite você crie seus próprios *"firmware patches*", como por exemplo, para habilitar o modo de monitoração (SCHULZ, WEGEMER, HOLLICK, 2017).

A partir das instruções do próprio Nexmon foi criado então um patch para o firmware do chip wireless do Raspberry Pi que permitia o uso do modo de monitoramento. Este firmware substituía o padrão, possuindo diversos riscos de danos ao hardware do equipamento, no entanto a instalação, seguindo as instruções cuidadosamente, obteve sucesso. Apenas come este firmware já foi possível criar a interface de monitoramento, no entanto outro problema surgiu ao efetuar a troca do canal sendo monitorado. A interface que estava sendo criada não habilitava esta mudança. Para resolver este problema recorreu-se ao conjunto de ferramentas de software Aircrack-ng. O Aircrack-ng é um conjunto completo de ferramentas para avaliar a segurança de redes WiFi. Com ele foi possível, através de um simples comando, criar a interface de monitoração que permitiu a troca em tempo real do canal monitorado.

3.2.2 Bash scripts

Bash é o *shell* (interface de acesso ao sistema operacional) desenvolvido pelo GNU Project, um projeto de desenvolvimento de um sistema operacional 100% livre. Hoje ele é distribuído como a interface de linha de comando padrão na maioria das distribuições Linux, incluindo o Raspbian.

Para automatizar determinados processos foram desenvolvidos então alguns *shell* scripts, que são sequencias de comandos ao sistema operacional interpretados por um *shell*, como o bash. Estes scripts seriam executados na inicialização ou ficariam permanentemente executando em segundo plano. No total foram feitos três scripts: um para criação da interface de monitoramento, um para constante troca de canal da placa wireless e um para transformar o programa em Python em um *daemon*, um serviço em segundo plano executado na inicialização. Estes scripts encontram-se nos apêndices deste trabalho.

O script que inicia o modo de monitoração é bem simples. Os scripts de inicialização devem seguir a *Linux Standard Base* (LSB), que pode ser traduzido como Base Padrão do Linux, que é um projeto da Linux Foundation para padronização da estrutura de software das distribuições Linux. Seguindo como base como são feitos os scripts para o Debian (DEBIAN, 2014), foi feito um script que executa apenas dois comandos na inicialização: *airmon-ng* start wlan0, para inicializar o modo de monitoramento da interface wlan0, e o comando *channel-change*, que é o script criado para trocar constantemente os canais da interface mencionada.

O script para a troca de canais da interface de rede sem fio não precisou ser criado como um script de inicialização visto que o script anterior já poderia inicia-lo desta forma. Logo ele seguiu os padrões de um *shell* script simples, possuindo apenas um laço de repetição infinito, que rodaria o programa até ser interrompido, e um laço de repetição estilo *for* que variava de 1 a 11. Esta variação era então aplicada no comando *iwconfig wlan0mon channel "$i"*, onde i é o número do canal. Optou-se por dedicar mais tempo do monitoramento nos canais 1, 6 e 11 pois estes são os canais mais utilizados. Estes canais são monitorados por 4 segundos enquanto os demais por apenas 2.

Para o script que faz o programa em Python rodar na inicialização como um *daemon*, por se tratar de um script mais complexo, uma solução foi procurada na Internet que auxiliasse a criação. A solução encontrada foi feita e distribuída pelo engenheiro da Universidade de Southampton Stephen Philips (PHILIPS, 2013). Seguindo seu padrão foi necessário alterar apenas os caminhos e nomes dos programas.

Com a placa wireless entrando em monitoramento na inicialização, seus canais alterando constantemente e os scripts prontos, restou somente o desenvolvimento do programa em Python.

3.2.3 Desenvolvimento do programa em Python

Com o sistema já configurado, o desenvolvimento de um programa que unisse e automatizasse tudo foi a última etapa. Inicialmente foram estabelecidas as principais funções do programa:

- Ler os pacotes capturados pelo tcpdump e extrair a informação do MAC, data e hora deles;

- Gerar um arquivo de saída com uma lista dos dispositivos capturados;
- Iniciar e interromper o tcpdump, controlando o tempo que fica ativo para que os arquivos de captura não fiquem muito grandes;
- Detectar caso o usuário conecte um dispositivo USB e gerar uma lista atualizada naquele momento.

Visto que o tcpdump gera o arquivo que serve como entrada de dados do programa, esta foi a primeira parte a ser desenvolvida. Para iniciar o tcpdump foi utilizado um módulo chamado *subprocess*. Com este módulo é possível iniciar subprocessos, redirecionando a saída deles, seja para a tela ou para um arquivo, e enviar sinais para eles, como de interrupção. O tcpdump foi então iniciado com as opções já mencionadas e redirecionando sua saída para um arquivo de texto nomeado *live_capture.txt* que fica em uma subpasta chamada *dumps* na pasta gerada pelo programa, *wifi_device_logger*, dentro da pasta do usuário padrão do Raspbian. Duas funções foram feitas, uma para iniciar o tcpdump e outra para interromper. A função que interrompe foi necessária para a finalização devida do programa e também para que o arquivo que salva a captura em tempo real fosse copiado e renomeado indicando a data da captura. Optou-se por reiniciar o tcpdump a cada hora pois em locais de maior movimento o arquivo de captura poderia ficar muito grande, dificultando a otimização do programa.

Conforme o arquivo de captura fosse gerado, ele precisaria ser lido em tempo real. Para isso uma função como a do comando *tail* no Linux, que acompanha o final de um arquivo conforme é escrito, deveria ser implementada. Para isto foi utilizado um *generator*, uma função que se comporta como um “iterador” (adaptado do inglês *iterator*), isto é, uma função que retorna uma iteração para um laço de repetição. A função faz isso através do comando *yield* no lugar do padrão *return* que as funções padrões utilizam para retornar um valor. O *yield* retorna o fluxo do programa para onde a função foi chamada, mas não tira da memória as variáveis locais da função e seus valores, podendo continuar de onde parou quando chamada novamente para a próxima iteração.

Para este projeto, a iteração é gerada cada vez que uma linha é escrita no arquivo de saída gerado pelo tcpdump. Cada quadro capturado pelo tcpdump é delimitado por uma quebra de linha, um exemplo de quadro capturado pode ser visto no quadro 4 abaixo. A função *generator* aguarda uma linha ser escrita, verificando a cada 0,1 segundo, e ao verificar que foi, a função, chamada *follow*, retorna a nova

linha para ser interpretada pelo programa e permanece aguardando a próxima iteração sem modificar o local do ponteiro de posição no arquivo.

Quadro 4 - Frame de Probe Request capturado.

2018-11-20 05:11:34.259737 10702424795us B 2437 MHz -76dBm signal BSSID:Broadcast DA:Broadcast SA:f4:f5:24:26:ba:4d (oui Unknown) Probe Request () [1.0 2.0 5.5 11.0 Mbit]

A análise e extração dos dados da linha retornada foi feita com o auxílio de um módulo de data e hora, *datetime*, que transforma uma dada *string* em uma variável de formado próprio com as operações básicas, como adição e subtração das datas, já implementadas. Como o tcpdump separa todas os campos do quadro capturado por um espaço, a extração do MAC foi feita utilizando uma simples função de *split*, que divide uma *string* em várias dado um delimitador, neste caso o espaço.

Para armazenar cada dispositivo capturado foi criada uma classe *Device* que guarda o MAC do dispositivo e a primeira e a última vez que ele foi visto. Foi criada para ela uma função de *timeout* que faz a diferença entre estes últimos valores citados e verifica se já faz mais que um determinado número de segundos que o dispositivo não é visto na área de alcance. O valor escolhido para a versão final foi de 10 minutos. Valores inferiores faziam determinados dispositivos gerar muitas entradas na lista.

Uma vez capturado, cada dispositivo é armazenado em memória em uma lista. Esta lista é verificada periodicamente por dispositivos que já não se encontram no alcance. Essa verificação é feita a cada 10 segundos para não gerar um excesso do uso do processador caso haja muitos aparelhos próximos. Ao verificar que uma entrada da lista que não enviou mais nenhum quadro a mais de 10 minutos, esta entrada é retirada da lista e salva em um arquivo seguindo o padrão visto no quadro 5, que tomou como base o frame exposto no quadro 4.

Quadro 5 - Exemplo de dispositivo capturado.

MAC: f4:f5:24:26:ba:4d | First Seen: 2018-11-20 05:00:49 | Last Seen: 2018-11-20 05:11:50

A separação utilizando uma barra vertical foi feita para fácil exportação para programas como o Microsoft Excel ou o Libreoffice Calc. Esta informação permanece na memória RAM até que o usuário conecte um dispositivo na porta USB ou o processo seja encerrado. Levando a próxima etapa que seria a detecção deste dispositivo na porta USB.

Como a porta USB está configurada para se comportar como um *gadget* OTG de interface de rede, a verificação foi feita conferindo dois arquivos contidos na pasta /sys/class/net/usb0/. Estes arquivos são o *carrier*, que guarda o status da conexão desta interface de rede, isto é, se há alguma outra interface de rede conectada nela, e o *carrier_changes*, que guarda o número de vezes que houve uma mudança na portadora da conexão. Assim foi possível detectar a conexão e desconexão de dispositivos de rede na porta USB sem confundir com qualquer outro dispositivo que pudesse ser conectado nela, como um teclado por exemplo. Esta verificação também é feita a cada 10 segundos, junto com a verificação de *timeout* dos dispositivos na lista em memória. O programa feito encontra-se nos apêndices deste trabalho.

Um programa adicional foi feito para a resolução do fabricante do MAC do quadro capturado. Inicialmente planejava-se fazer esta tradução em tempo real, porém isso se demonstrou demais para o processador do Raspberry Pi Zero W. Este programa utiliza uma lista “limpa” feita a partir da lista oficialmente fornecida pelo IEEE de fabricantes de adaptadores de rede. Esta lista foi feita através de uma “limpeza” da lista oficial para agilizar a pesquisa. No quadro 6 pode ser visto um exemplo de uma entrada da lista oficial contra uma entrada na lista gerada.

Quadro 6 - Entrada de fabricante na lista oficial do IEEE contra lista gerada para uso no projeto.

Oficial IEEE		**Versão gerada**
3C-D9-2B (hex) 3CD92B (base 16)	Hewlett Packard Hewlett Packard 11445 Compaq Center Drive Houston 77070 US	3c:d9:2b Hewlett Packard

O programa pode ser chamado pelo comando *resolve_oui* e fornecendo como parâmetro de entrada o arquivo a ser traduzido. Ele gera então dois arquivos de saída: um com todas as linhas do arquivo fornecido como entrada e um que exclui dispositivos repetidos, gerando uma lista mais limpa. O quadro 7 mostra o resultado gerado, baseado na entrada do quadro 6.

Quadro 7 - Arquivo de saída com nome do fabricante.

MAC: f4:f5:24:26:ba:4d | First Seen: 2018-11-20 05:00:49 | Last Seen: 2018-11-20 05:11:50 | Organization: Motorola Mobility LLC, a Lenovo Company

Esta etapa conclui o desenvolvimento do projeto. Todos os códigos de scripts e programas escritos podem ser verificados nos apêndices deste trabalho.

3.3 RESULTADOS

A coleta de dados foi feita em dois locais para testes. Dentro do prédio do Palácio das Telecomunicações, localizado na Avenida Manoel Ribas número 115 em Curitiba, doravante denominado local 1, e em minha própria residência, local 2. A ideia foi comparar estes dois lugares pois o primeiro apresenta um tráfego muito elevado de dispositivos enquanto o segundo, sendo um ambiente residencial, um tráfego bem inferior.

Durante os testes surgiu a dúvida a respeito do número de dispositivos capturados, que estava muito elevado até mesmo para um local movimentado como o local 1. Este número, em apenas algumas horas de captura, chegava a mais de 2000 dispositivos que muitas vezes permaneciam apenas por um ou dois segundos no alcance. Muitas vezes um determinado MAC enviava um único quadro e não se comunicava mais após isso. O padrão verificado foi que uma porcentagem bem alta destes MACs eram de um mesmo fabricante, registrado como Motorola Mobility LLC, a Lenovo Company, com a identificação de MAC f4:f5:24 alterando somente os bytes relacionados ao dispositivo. A conclusão chegada veio de Vanhoef (2016), onde foi possível verificar que algumas empresas estariam aleatorizando o endereço MAC de seus clientes para tentar proteger sua privacidade. Conforme o próprio artigo cita, apesar da funcionalidade estar disponível no sistema operacional Android, somente alguns fabricantes estaria utilizando. E com esta análise, percebemos que somente alguns modelos de celulares Motorola utilizam aqui no Brasil. Testes mais controlados seriam necessários para identificar tais modelos.

A captura no local 2 foi sempre dentro do esperado, nunca passando de 50 dispositivos com padrões esperados, onde ficavam no alcance por alguns minutos antes de desaparecer.

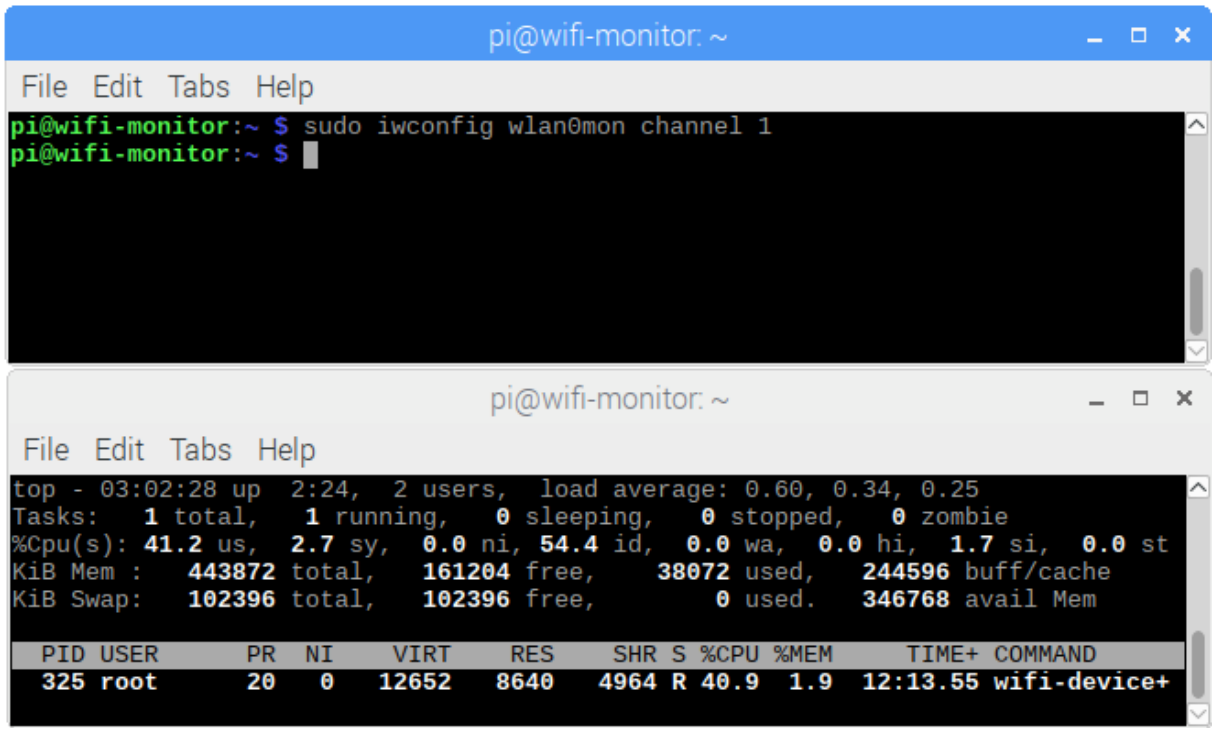
O uso do processador foi mais alto que o esperado. Em local movimentados, a monitoração dos canais 1, 6 e 11 faziam o uso do processamento chegar a mais de 50%. Ainda assim, com a constante mudança de canal a utilização média ficou baixa, ficando em média 8% no local 1 e menos de 3% no local 2. A figura 9 mostra a média do uso de processamento dos 3 processos gerados por este projeto em uma captura de 1 hora no local 1.

Figura 9 - Média de uso da CPU no local 1.

Fonte: autoria própria.

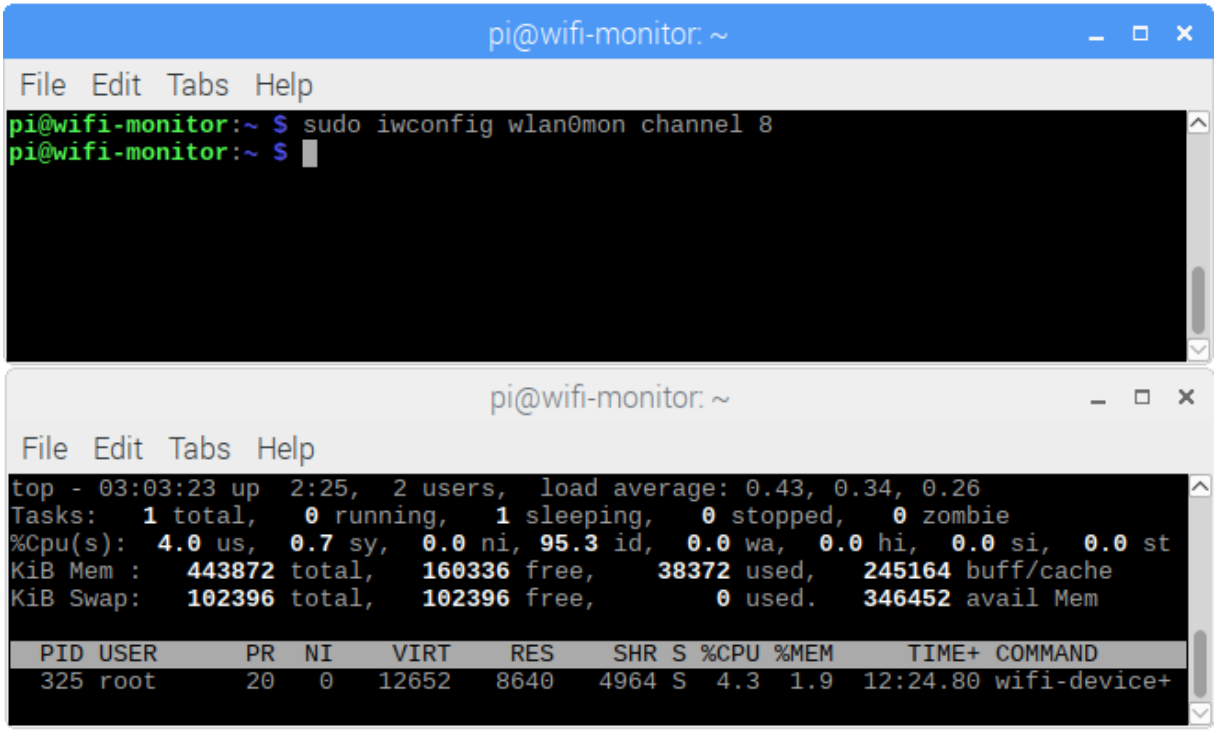
As figuras 10 e 11 demonstram o uso do processador após interromper o script que altera constantemente os canais. Para comparação foi então fixado o canal 1, que conforme demonstrado na figura 12, capturada com o auxílio do aplicativo Wifi Analyzer, estava congestionado de redes, e em seguida a captura foi fixada no canal 8, um canal intermediário pouco utilizado.

Figura 10 - Uso da CPU capturando no canal 1.



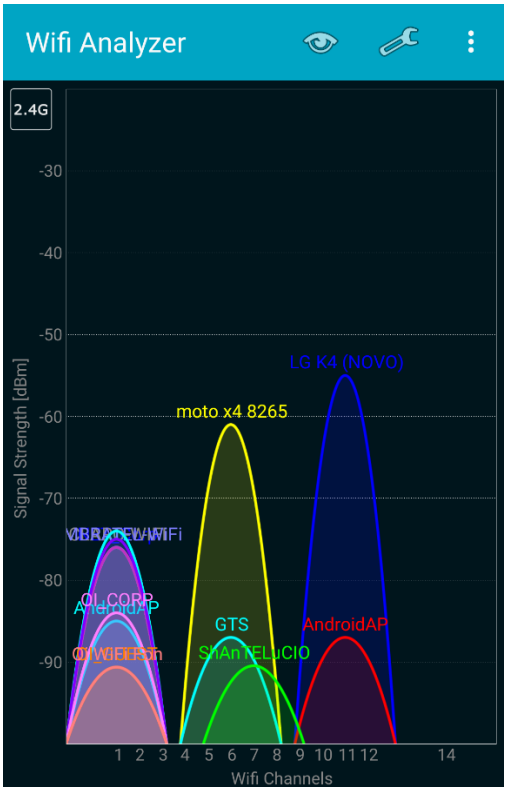
Fonte: autoria própria.

Figura 11 - Uso da CPU capturando no canal 8.



Fonte: autoria própria.

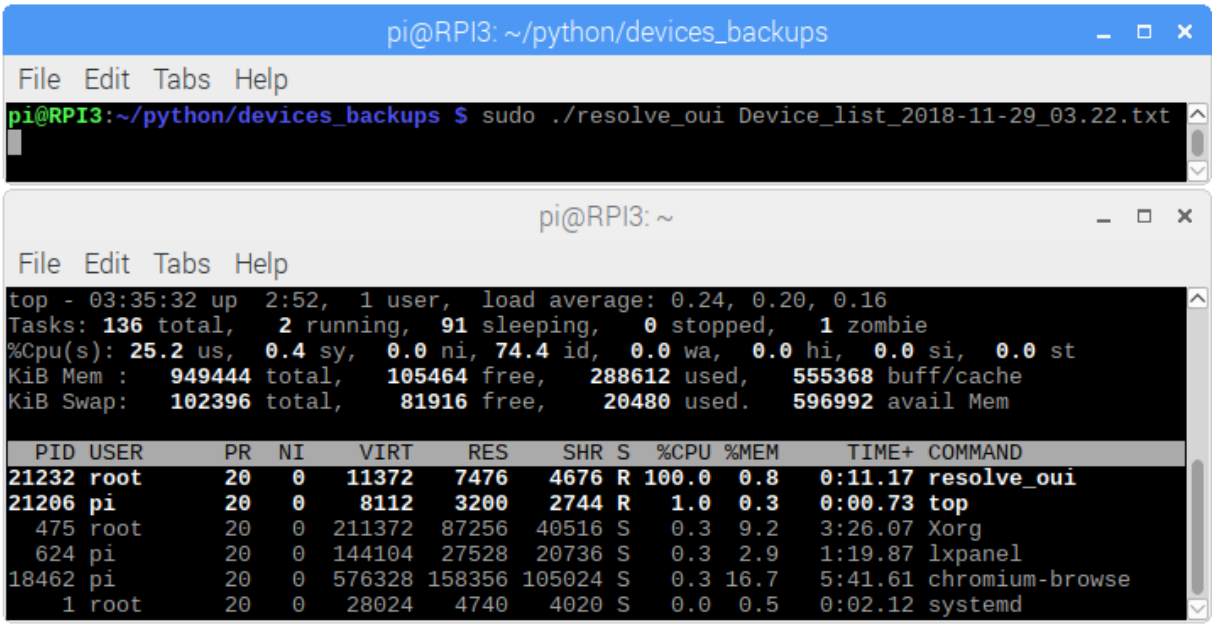
Figura 12 - Pontos de acesso no local 1.



Fonte: autoria própria.

Os testes com a resolução de nomes simplesmente demonstraram que, pelo menos utilizando a busca padrão por texto, ela não seria possível em tempo real. Ao buscar os nomes dos fabricantes de uma lista de 500 dispositivos o programa utilizava 100% do processamento por vários minutos no Raspiberry Pi Zero W e mesmo no Raspberry Pi 3 Model B, apesar de levar menos tempo, ainda assim chegava a um minuto de uso constante de todo o poder do processador. A busca poderia ser consideravelmente otimizada utilizando algoritmos de busca e ordenação mais eficientes. A solução adotada foi a de remover esta função e executa-la somente sob demanda, a figura 13 demonstra o uso da unidade central de processamento (CPU, do inglês *Central Processing Unit*) com o script em execução no Raspberry Pi 3 Model B.

Figura 13 - Uso da CPU com programa resolve_oui em execução.



Fonte: autoria própria.

4 CONSIDERAÇÔES FINAIS E TRABALHOS FUTUROS

4.1 CONCLUSÃO

A proposta original do projeto trazia uma ideia mais direcionada a segurança da informação. No entanto o que foi observado durante as pesquisas foi seu potencial para levantamentos comportamentais através de análises da movimentação dos dispositivos ao redor. O potencial para uma camada bem superficial de segurança ainda existe, funcionando talvez apenas como um detector, mas com os próprios fabricantes começando a aleatorizar seus endereços MAC, estudos como os feitos por Vanhoef (2016) serão necessários para verificar se estas práticas se tornarão padrões.

Com relação ao comportamento do Raspberry Pi Zero W, o que se pôde observar é que o programa teria de ser otimizado para atender uma área de elevado tráfego, como um shopping center. Isso seria possível caso o programa desenvolvido fosse desenvolvido utilizando uma linguagem compilada ao invés de interpretada. Coletar também apenas os pacotes de *Probe Request* também poderia trazer melhores resultados caso o interesse seja de capturar apenas dispositivos descobrindo rede, e não servindo de ponto de acesso para uma. O processo de resolução de nomes dos fabricantes poderia ser otimizado ordenando a lista fornecida pelo IEEE e utilizando algoritmos de busca mais eficientes. Neste trabalho a busca ficou como fator secundário e foi utilizada somente uma busca de texto simples, que possui tempo de execução muito alto.

Outra solução para o problema de processamento seria a atualização para um modelo mais forte. Isso aumentaria o custo do projeto e o gasto de energia, porém traria também uma variedade imensa de possibilidades no que se trata de comunicação. Os modelos da série Raspberry Pi 3 possuem 4 portas USB e uma porta Ethernet, além da placa Wi-Fi e Bluetooth Low Energy (BLE). Além disso, no início deste ano foi lançado o modelo B+ que, além de outras atualizações, substituiu o chip wireless pelo modelo Cypress CYW43455 que traz um combo de Wi-Fi padrão 802.11ac e Bluetooth, trazendo assim o suporte a redes Wi-Fi na faixa de 5 GHz para o Raspberry.

4.2 TRABALHOS FUTUROS

Um estudo da utilização de algoritmos para aleatorizar o endereço MAC das interfaces de rede pode trazer resultados interessantes e, juntamente com um estudo de padrões de preenchimento dos campos de dados dos quadros de gerenciamento do padrão 802.11 podem ajudar a obter mais informações do dispositivo que enviou o quadro, como por exemplo o sistema operacional.

A implementação de uma comunicação entre mais de um dispositivo de captura traz a possibilidade de estudos para traçar padrões de movimentação ou de aglomerações.

A integração de um banco de dados pode melhorar a eficiência do armazenamento, permitindo assim registrar mais informações e facilitando a integração com serviços de nuvem, que poderiam ser acessados através de módulos de transmissão de dados móveis 3G/4G.

O mesmo padrão seguido por este projeto também pode ser expandido para detecção de outras tecnologias sem fio como o Bluetooth.

REFERÊNCIAS

ADACHI, Fumiyuki. **Wireless Network Evolution Toward 5G Network.** 2016, 3f. Pesquisa (Organização de Pesquisa de Comunicação Elétrica), Universidade de Tohoku, Sendai. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7718227> Acesso em: 20 maio 2017.

ALLBUSINESS Editors. **WHY you need a LAN?** 2010. Disponível em: <https://www.allbusiness.com/why-you-need-a-lan-403-1.html> Acesso em: 29 nov. 2018

AMARAL, Allan F. F. **Redes de Computadores**. Colatina: e-Tec, 2012.

BOUVETTE, Thomas d'Otreppe de. **Aircrack-ng**. 2009. Disponível em: < https://www.aircrack-ng.org/> Acesso em: 01 dez. 2018.

BULHMAN, Haroldo J. CABIANCA Luis A. **Redes LAN/MAN Wireless II: Funcionamento do Padrão 802.11**. Disponível em: <http://www.teleco.com.br/tutoriais/tutorialrwlanman2/default.asp> Acesso em: 21 de jun. 2017.

CLABURN, Thomas. **MAC randomization: A massive failure that leaves iPhones, Android mobes open to tracking**. 2017. Disponível em: <https://www.theregister.co.uk/2017/03/10/mac_address_randomization/> Acesso em: 22 nov. 2018.

CORFIELD, Gareth. **TfL to track Tube users in stations by their MAC adresses**. 2016. Disponível em: <https://www.theregister.co.uk/2016/11/17/tfl_to_track_tube_users_by_wifi_device_mac_address/> Acesso em: 22 nov. 2018.

DEBIAN wiki. **LSBInitScripts**. 2014. Disponível em: <https://wiki.debian.org/LSBInitScripts> Acesso em: 01/12/2018.

EAMES, Alex. **How much power does Pi Zero W use?** 2017. Disponível em: <https://raspi.tv/2017/how-much-power-does-pi-zero-w-use> Acesso em: 28 nov. 2018.

FALL, Kevin R. TCP/IP Illustrated, Volume 1. New Jersey: Pearson Education, Inc, 2012.

FOROUZAN, Behrouz A. Comunicação Dados e Redes de Computadores. 3. ed. São Paulo: Amgh Editora, 2008.

FOROUZAN, Behrouz A. Protocolo Tcp/Ip. 3. ed. São Paulo: Amgh Editora, 2009.

FRANCISCATTO, Roberto et al. **Redes de Computadores**. Ed. Frederico Westphalen: e-Tec, 2014.

GAST, Matthew S. **802.11® Wireless Networks The Definitive Guide**. 2nd ed. Sebastopol: O'Reilly Media, 2005. 656 p.

GNU Bash. **Free Software Foundation**. 2007. Disponível em: < https://www.gnu.org/software/bash/> Acesso em: 01 dez. 2018.

GOODIN, Dan. **No, this isn't a scene from Minority Report. This trash can is stalking you.** 2013. Disponível em: <https://arstechnica.com/information-technology/2013/08/no-this-isnt-a-scene-from-minority-report-this-trash-can-is-stalking-you/> Acesso em 22 nov. 2018.

IEEE Computer Society - **IEEE Standard for Information and technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific Requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications**. 2016. Nova Iorque. Disponível em: <http://standards.ieee.org/about/get/802/802.11.html> Acesso em: 21 jun. 2017.

MARQUES, Francisco. **A regulação do acesso wireless Internet no Brasil.** 2006, 24 f. Dissertação (Doutorado em Comunicação e Cultura Contemporânea), Universidade Federal da Bahia, Ondina. Disponível em: <http://repositorio.ufc.br/bitstream/riufc/673/1/2006_art20fpjamarquesEptic.pdf> Acesso em: 14 maio. 2017.

MOTA FILHO, João Eriberto. **Análise de tráfego em redes TCP/IP: utilize tcpdump na análise de tráfegos em qualquer sistema operacional**. 1. ed. São Paulo: Novatec, 2013. 416 p.

MSHVIDOBADZE, Tinatin. **Evolution Mobile Wireless Communication And Lte Networks.** 2012, 7f. Dissertação (Doutorado em Ciência), Universidade da Georgia, Athens. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6398495> Acesso em 14 maio. 2017.

MULLINER, Collin R. **Security of Smart Phones.** 2006, 119f. Dissertação (Mestrado de Ciência em Ciência da Computação), Universidade da Califórnia, Santa Barbara. Disponível em <https://mulliner.org/mobilesecurity/2006_mulliner_MSThesis.pdf> Acesso em 5 junho. 2017

MURRAY, David. DIXON, Michael. KOZINIEC, Terry. **Scanning Delays in 802.11 Networks**. Disponível em: <http://ieeexplore.ieee.org/document/4343430/> Acesso em: 21 jun. 2017.

PINHEIRO, José Maurício. **Guia completo de cabeamento de redes**. Rio de Janeiro: Campus, 2003.

PERAHIA, Eldad. STACEY, Robert. **Next Generation Wireless LANs: Throughput, Robustness, and Reliability in 802.11n**. Cambridge: Cambridge University Press, 2008. 416 p.

PHILLIPS, Stephen. **Getting a Python script to run in the background (as a service) on boot**. 2013. Disponível em: <http://blog.scphillips.com/posts/2013/07/getting-a-python-script-to-run-in-the-background-as-a-service-on-boot/> Acesso em: 01 dez. 2018.

RACKLEY, Steve. **Wireless Networking Technology: From Principles to Successful Implementation**. Oxford: Elsevier, 2007. 416 p.

RASPBIAN**. Raspberry Pi Foundation.** S.d. Disponível em: <https://www.raspbian.org> Acesso em: 26 mai. 2017

SARWAR, Muhammad. SOOMRO, Tariq R. **Impact of Smartphone's on Society**. 2013, 11f. Dissertação, Universidade de Ciência e Tecnologia, All Ain. Disponível em: <https://pdfs.semanticscholar.org/2c28/0b6a690442a97a571e09b2404e2d21720db4.pdf> Acesso em 1. jun. 2018.

SCHULZ, Matthias. WEGEMER, Daniel. HOLLICK, Matthias. **Nexmon: The C-based Firmware Patching Framework**. 2017. Disponível em: < https://nexmon.org> Acesso em: 01 dez. 2018.

TCPDUMP & Libcap**. The Tcpdump team**. 2010. Disponível em: <http://www.tcpdump.org> Acesso em: 26 mai. 2018.

VANHOEF, Mathy et al. **Why MAC Address Randomization is not Enough: An Analysis of Wi-Fi Network Discovery Mechanisms**. 2016. Disponível em: < http://papers.mathyvanhoef.com/asiaccs2016.pdf> Acesso em: 22 nov. 2018.

WI-FI ALLIANCE. **Wi-Fi Alliance**. S.d. Disponível em: < https://www.wi-fi.org/who-we-are> Acesso em: 24 mai. 2018.

APÊNDICE A – Script create-mon

```
#! /bin/sh
# /etc/init.d/create_mon

### BEGIN INIT INFO
# Provides:          create_mon
# Required-Start:    $remote_fs $syslog
# Required-Stop:
# Default-Start:     2 3 4 5
# Default-Stop:
# Short-Description: Starts monitoring mode on wlan0 and channel-change script  at boot.
### END INIT INFO

# If you want a command to always run, put it here

# Carry out specific functions when asked to by the system
case "$1" in
  start)
    echo "Creating mon0"
    # run application you want to start
    airmon-ng start wlan0
    channel-change
    ;;
esac

exit 0
```

APÊNDICE B – Script channel_change

```
#!/bin/bash

while true
do
  for i in {1..11}
  do
    iwconfig wlan0mon channel "$i"
    if [ $i -eq 1 ] || [ $i -eq 6 ] || [ $i -eq 11 ]
    then
      sleep 4
    else
      sleep 2
    fi
  done
done
```

APÊNDICE C – Script wifi-device-logger-init

```
#!/bin/sh

### BEGIN INIT INFO
# Provides:          wi-fi-device-logger
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Monitor and log MAC devices in range
# Description:       Uses interface wlan0mon, a wireless interface in monitor mode, to capture control
802.11 frames and keep a log of the MAC devices who send them. Save the logs in the pi user folder.
### END INIT INFO

# Change the next 3 lines to suit where you install your script and what you want to call it
DIR=/usr/local/bin
DAEMON=$DIR/wifi-device-logger
DAEMON_NAME=wifi-device-logger

# Add any command line options for your daemon here
DAEMON_OPTS=""

# This next line determines what user the script runs as.
# Root generally not recommended but necessary if you are using the Raspberry Pi GPIO from
Python.
DAEMON_USER=root

# The process ID of the script when it runs is stored here:
PIDFILE=/var/run/$DAEMON_NAME.pid

. /lib/lsb/init-functions

do_start () {
    log_daemon_msg "Starting system $DAEMON_NAME daemon"
    start-stop-daemon --start --background --pidfile $PIDFILE --make-pidfile --user $DAEMON_USER --
chuid $DAEMON_USER --startas $DAEMON -- $DAEMON_OPTS
    log_end_msg $?
}
do_stop () {
    log_daemon_msg "Stopping system $DAEMON_NAME daemon"
    start-stop-daemon --stop --pidfile $PIDFILE --retry 10
    log_end_msg $?
}

case "$1" in

    start|stop)
        do_${1}
        ;;

    restart|reload|force-reload)
        do_stop
        do_start
        ;;

    status)
        status_of_proc "$DAEMON_NAME" "$DAEMON" && exit 0 || exit $?
        ;;
```

```
    *)
        echo "Usage: /etc/init.d/$DAEMON_NAME {start|stop|restart|status}"
        exit 1
        ;;

esac
exit 0
```

APÊNDICE D – Programa em python wifi-device-logger

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
"""Script para o registro do MAC de dispositivos wireless capturados
pela interface de rede em modo de monitoramento wlan0mon utilizando o
tcpdump"""

from datetime import datetime
import time
import subprocess
import os
import errno
import shutil
import signal


class Device():
    """Classe para registro dos dispositivos"""

    def __init__(self, mac="", fseen=""):
        self.__mac = mac
        self.__fseen = fseen
        self.__lseen = fseen

    @property
    def mac(self):
        """Retorna o MAC do dispositivo"""
        return self.__mac

    @property
    def fseen(self):
        """Retornar a primeira vez que o dispositivo foi visto (datetime)"""
        return self.__fseen

    @property
    def lseen(self):
        """Retornar a ultima vez que o dispositivo foi visto (datetime)"""
        return self.__lseen

    def __str__(self):
        """Formata o dispositivo para impressao no arquivo"""
        return ("MAC: " + self.__mac + " | First Seen: " + str(self.__fseen)
                + " | Last Seen: " + str(self.__lseen))

    def __eq__(self, other):
        """Compara os dispositivos pelo MAC"""
        if isinstance(other, Device):
            return self.__mac == other.mac
        return False

    def seen(self, value):
        """Atualiza a ultima vez que o dispositivo foi visto (datetime)"""
        if isinstance(value, datetime):
            if self.__lseen < value:
                self.__lseen = value
                return True
            return False
        print("Not datetime format!")
```

```
        return False

    def timeout(self):
        """Verifica se ja faz mais que 5 minutos desde a ultima vez que o
        dispositivo foi visto"""
        if (datetime.now() - self.lseen).total_seconds() > 600:
            return True
        return False


class CleanKiller():
    """Classe para tratar sinalizacao de saida"""
    kill_now = False
    def __init__(self):
        signal.signal(signal.SIGINT, self.clean_exit)
        signal.signal(signal.SIGTERM, self.clean_exit)

    def clean_exit(self, signum, frame):
        """Seta verificacao de saida"""
        self.kill_now = True


def get_frame(line):
    """Funcao para o parsing da linha do frame capturado"""

    if not line.strip():
        return None

    w_frame = []
    try:  #Verifica erros dados devido a linha nao estar impressa por completo
        w_frame.append(datetime.strptime(line[0:19], "%Y-%m-%d %H:%M:%S"))
        source_addr = line.split(" SA:", 1)[1]
        source_addr = source_addr.split(" ", 1)[0]
        if len(source_addr) < 17:
            raise ValueError
        w_frame.append(source_addr)
    except (ValueError, IndexError):
        pass
    else:
        return w_frame


def check_dev_timeout(ram_dev_list, path):
    """Verifica timeout e grava dispositivos no arquivo de saida"""
    if not ram_dev_list:
        return

    device_file_path = path + "live_device_list.txt"
    with open(device_file_path, 'a') as outputfile:
        for dev in ram_dev_list[:]:
            if dev.timeout():
                outputfile.write(str(dev) + "\n")
                ram_dev_list.remove(dev)

def save_dev_list(ram_dev_list, path):
    """Grava todos os dispositivos em RAM no arquivo, sem alterar a ram"""
    if not ram_dev_list:
        return

    device_file_path = path + "live_device_list.txt"
```

```
    timestr = time.strftime("%Y-%m-%d_%H.%M")
    filename = "Device_list_" + timestr + ".txt"
    output_file_path = path + filename

    try:
        shutil.copy(device_file_path, output_file_path)

        with open(output_file_path, 'a') as outputfile:
            for dev in ram_dev_list:
                outputfile.write(str(dev) + "\n")

    except IOError:
        pass



def flush_dev_list(ram_dev_list, path):
    """Grava todos os dispositivos em RAM no arquivo"""
    if not ram_dev_list:
        return

    device_file_path = path + "live_device_list.txt"
    with open(device_file_path, 'a') as outputfile:
        for dev in ram_dev_list[:]:
            outputfile.write(str(dev) + "\n")
            ram_dev_list.remove(dev)

    timestr = time.strftime("%Y-%m-%d_%H.%M")
    filename = "Device_list_" + timestr + ".txt"
    output_file_path = path + filename

    try:
        shutil.move(device_file_path, output_file_path)
    except IOError:
        pass


def tcpdump_start(file_path):
    """Inicia o tcpdump, gerando arquivo live_capture.txt"""
    with open(file_path, 'w') as capturefile:
        tcp_args = ['tcpdump', '-i', 'wlan0mon', '-e', '-tttt', '-U',
                    'wlan[0]=0x80', 'or', 'wlan[0]=0x40', 'or', 'wlan[0]=0x50']
        tcpd_process = subprocess.Popen(tcp_args, stdout=capturefile,
                                        stderr=open(os.devnull, 'w'))

    return tcpd_process


def tcpdump_stop(tcpd_process, file_path):
    """Para tcpdump e move o arquivo live_capture"""

    tcpd_process.send_signal(subprocess.signal.SIGTERM)
    tcpd_process.communicate()

    timestr = time.strftime("%Y-%m-%d_%H.%M")
    file_dir = os.path.dirname(file_path)
    dump_filename = "Capture_" + timestr + ".txt"
    final_dump_filename = os.path.join(file_dir, dump_filename)
```

```
    try:
        shutil.move(file_path, final_dump_filename)
    except IOError:
        pass


def follow(thefile):
    """Funcao para o acompanhamento da captura conforme e alimentado"""
    thefile.seek(0, 2)
    while True:
        line = thefile.readline()
        if not line:
            time.sleep(0.1) #Aguarda uma nova linha no arquivo sendo acompanhado
            continue
        yield line


def check_usb(last_c_change):
    """Verifica se uma interface de rede foi conectada na USB"""
    with open('/sys/class/net/usb0/carrier_changes', 'r') as cc_file:
        c_change = cc_file.readline().strip()
        if last_c_change != c_change:
            with open('/sys/class/net/usb0/carrier', 'r') as c_file:
                if c_file.read(1) == '1':
                    return True, c_change
                else:
                    return False, last_c_change
        else:
            return False, last_c_change


def main():
    """Início do script caso diretamente executado"""

    killer = CleanKiller()

    save_dir = "/home/pi/wifi-device-logger/" #Pasta para salvar os logs
    rel_path = "dumps/live_capture.txt"
    capture_file_path = save_dir + rel_path

    if not os.path.exists(os.path.dirname(capture_file_path)):
        try:
            os.makedirs(os.path.dirname(capture_file_path))
        except OSError as exc:
            if exc.errno != errno.EEXIST:
                raise

    try:
        with open('/sys/class/net/usb0/carrier_changes', 'r') as cc_file:
            carrier_changes = cc_file.readline().strip()

        dev_list = []
        last_dev_check = datetime.now()

        while True:

            if killer.kill_now:
                raise SystemExit

            tcpd_process = tcpdump_start(capture_file_path)
```

```
            tcpdump_timer = datetime.now()

            capturefile = open(capture_file_path, 'r')
            loglines = follow(capturefile)

            for c_line in loglines:
                if killer.kill_now:
                    raise SystemExit
                c_frame = get_frame(c_line)
                if c_frame:
                    c_dev = Device(c_frame[1], c_frame[0])
                    if c_dev in dev_list:
                        dev_list[dev_list.index(c_dev)].seen(c_dev.lseen)
                    else:
                        dev_list.append(c_dev)

                if (datetime.now() - last_dev_check).total_seconds() > 10:
                    if dev_list:
                        check_dev_timeout(dev_list, save_dir)
                    last_dev_check = datetime.now()
                    usb = check_usb(carrier_changes)
                    if usb[0] == True:
                        carrier_changes = usb[1]
                        save_dev_list(dev_list, save_dir)

                if (datetime.now() - tcpdump_timer).total_seconds() >= 3600:
                    break

            capturefile.close()
            loglines = None
            tcpdump_stop(tcpd_process, capture_file_path)

        except (KeyboardInterrupt, SystemExit):
            if not capturefile.closed:
                capturefile.close()
            tcpdump_stop(tcpd_process, capture_file_path)
            flush_dev_list(dev_list, save_dir)


if __name__ == '__main__':
    main()
```

APÊNDICE E – Programa em python resolve_oui

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
"""
Abre arquivo de dispositivos e resolve os fabricantes gerando um arquivo
completo e um sem duplicados
"""

import sys
import os
import stat

def parse_dev_list(file):
    """Transforma arquivo de dispositivos em uma lista"""

    dev_list = []
    dev = []

    for line in file:
        split_aux = line.split("MAC: ")[1].split(" | ")[0]
        dev.append(split_aux)
        split_aux = line.split("First Seen: ")[1].split(" | ")[0]
        dev.append(split_aux)
        split_aux = line.split("Last Seen: ")[1]
        dev.append(split_aux.rstrip())
        dev_list.append(dev)
        dev = []

    return dev_list

def resolve_oui(dev_list):
    """Procura no arquivo de OUIs o nome da fabricante do MAC"""

    if not dev_list:
        return None

    for dev in dev_list:
        oui = dev[0][0:8]
        with open("oui/clean_oui.txt", encoding="utf8") as oui_file:
            for line in oui_file:
                if oui in line:
                    dev.append(line.split(" ", 1)[1].rstrip())
                    break
    return dev_list

def remove_duplicates(complete_dev_list):
    """Gera uma lista sem duplicados"""
    clean_dev_list = []
    seen = set()

    for dev in complete_dev_list:
        if dev[0] not in seen:
            clean_dev_list.append(dev)
            seen.add(dev[0])
        else:
            for dev2 in clean_dev_list:
                if dev2[0] == dev[0]:
                    dev2[2] = dev[2]
```

```
    return clean_dev_list

def main():
    """Início do programa caso diretamente executado"""
    if len(sys.argv) > 1:
        with open(sys.argv[1], 'r') as devfile:
            dev_list = parse_dev_list(devfile)
            dev_list = resolve_oui(dev_list)

            output_file_name = "Complete_" + os.path.basename(devfile.name)

            with open(output_file_name, 'w') as output_file:
                for dev in dev_list:
                    if len(dev) < 4:
                        output_file.write("MAC: " + dev[0] + " | First Seen: " +
                                          dev[1] + " | Last Seen: " + dev[2] + "\n")
                    else:
                        output_file.write("MAC: " + dev[0] + " | First Seen: " +
                                          dev[1] + " | Last Seen: " + dev[2] +
                                          " | Organization: " + dev[3] + "\n")
                output_file.write("\nTotal: " + str(len(dev_list)))

            os.chmod(output_file_name, stat.S_IRUSR | stat.S_IRGRP | stat.S_IROTH)
            dev_list = remove_duplicates(dev_list)
            output_file_name = "Clean_" + os.path.basename(devfile.name)

            with open(output_file_name, 'w') as output_file:
                for dev in dev_list:
                    if len(dev) < 4:
                        output_file.write("MAC: " + dev[0] + " | First Seen: " +
                                          dev[1] + " | Last Seen: " + dev[2] + "\n")
                    else:
                        output_file.write("MAC: " + dev[0] + " | First Seen: " +
                                          dev[1] + " | Last Seen: " + dev[2] +
                                          " | Organization: " + dev[3] + "\n")
                output_file.write("\nTotal (unique MACs): " +
                                  str(len(dev_list)))

            os.chmod(output_file_name, stat.S_IRUSR | stat.S_IRGRP | stat.S_IROTH)

        os.remove(sys.argv[1])
    else:
        print("\nForneca o nome do arquivo com a lista de dispositivos gerada " +
              "pelo script wifi_device_logger como parametro (arg[1]).\n" +
              "Uso correto: ./resolve_oui [file]\n")


if __name__ == '__main__':
    main()
```

APÊNDICE F – Programa em python oui_cleanner

```
with open("oui.txt", 'r', encoding="utf8") as input_file:
    with open("clean_oui.txt", 'w', encoding="utf8") as output_file:
        for line in input_file:
            if "(hex)" in line:
                out_line = line.split("   (hex)\t\t")
                out_line[0] = out_line[0].replace('-', ':')
                output_file.write(out_line[0].lower() + " " + out_line[1])
```