Implementação de um filtro de pacotes inteligente para dispositivos de Internet das Coisas

Gabriel Victor C Fernandes[1,2], Pedro H Borges Monici[1,2], César H de Araujo Guibo[1,2]
Gustavo de Carvalho Bertoli[1], Aldri Santos[3], Lourenço Alves Pereira Jr.[1]

[1] Divisão de Ciência da Computação
Instituto Tecnológico de Aeronáutica (ITA) – São José dos Campos, SP – Brazil

[2]Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo (USP) – São Carlos, SP – Brazil

[3]Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG – Brazil

{gabriel.fernandes,pedroh.monici}@ga.ita.br, {bertoli,ljr}@ita.br

cesarguibo@usp.br, aldri@dcc.ufmg.br

Abstract. *Cybersecurity is crucial for digital transformation as many computing assets are exposed on the network. In this context, attackers exploit vulnerabilities and proceed with privilege escalation to perform malicious actions. Despite this, there are few solutions to protect Internet of Things devices. Thus, this article presents the T800 packet filter to provide low resource consumption and packet filtering with advanced algorithms. The results show the efficiency of the T800 via implementation and experimentation through the ESP32 board and the ESP-IDF system. Furthermore, T800 increased the device's computational capacity by excluding unsolicited malicious traffic from the processing pipeline.*

Resumo. *A Segurança Cibernética é uma questão crucial a medida que muitos ativos computacionais ficam expostos na rede. Nesse contexto, os atacantes exploram vulnerabilidades e escalam privilégios para executar ações maliciosas. Essa exposição exige soluções para proteção de dispositivos de Internet das Coisas. Assim, este artigo apresenta o filtro de pacotes T800 capaz de proporcionar baixo consumo computacional e filtragem de pacotes com algoritmos avançados. Os resultados evidenciam a eficiência do T800 por meio de implementação e experimentação através da placa ESP32 e do sistema ESP-IDF. Mais ainda, T800 foi capaz de aumentar a capacidade computacional do dispositivo tendo em vista que o tráfego malicioso é excluído do processamento.*

1. Introdução

A temática de segurança cibernética vem cada vez mais sendo apontada como um dos aspectos estratégicos para a continuidade dos negócios [Klint 2021]. Segundo o Relatório de Riscos Globais de 2021 do World Economic Forum [McLennan 2021], incidentes desta natureza representam um dos grandes desafios pós-pandemia e possuem potencial para causar disrupção econômica, perdas financeiras, tensões geopolíticas e/ou instabilidades sociais. Dessa forma, é importante destacar que segurança cibernética deveria ser parte essencial do ciclo de vida do desenvolvimento de produtos e serviços. É possível observar

que, no passado, ataques cibernéticos eram divulgados em mídias especializadas. No entanto, devido à transformação digital pela qual o mundo passa, esses tipos de incidentes têm aparecido em veículos direcionados ao público geral (e.g., JBS em junho de 2021[1], USA Pipelines em maio de 2021[2], TJ-RS em abril de 2021[3], STJ em novembro de 2020[4], apenas para destacar alguns). Assim, há uma relação entre ataques cibernéticos e impactos em diversas áreas de atividades do setor produtivo [Lobo 2019].

Apesar de observarmos a existência de *Advanced Persistent Threats* (APTs) que direcionam esforços a um sistema em específico, os ataques podem ser parte de campanhas em larga escala com objetivos de orquestração de atividades maliciosas de grande escala [Bertino and Islam 2017]. Nesse sentido, um caso típico trata do comprometimento de recursos computacionais, isto é, computadores, smart-phones, roteadores Wi-Fi, câmeras de monitoramento entre outros, fazendo com que componham uma rede de comando e controle e possam ser ativados conforme a conveniência do atacante conhecidos como *botnets*. Dispositivos de Internet das Coisas (IoT) são alvos comuns pelo fato de possuírem ciclo de atualização e manutenção precários; algo que pode ser observado em *malwares* como Mirai [Chacos 2016] e Mozi [McMillen 2021]. Portanto, ao considerar a política de pouca atualização, a negligência de adoção de um processo de desenvolvimento que inclua segurança como elemento essencial e o avanço da adoção de sistemas computacionais como habilitadores de novas soluções tecnológicas, cada vez mais sistemas de IoT são alvo de atores maliciosos.

Pelo lado do atacante, Cyber Kill Chain é um modelo capaz de descrever os passos necessários de um ataque cibernético [Yadav and Rao 2015], consistindo de 7 fases consecutivas: reconhecimento (*reconnaissance*), armamento, entrega, exploração, instalação, comando e controle, e ação. A Figura 1a ilustra esse encadeamento. Assim sendo, a primeira atividade trata da coleta de informações e identificação dos dispositivos presentes e tem o objetivo de enumerar hardware, sistema operacional, serviços e aplicações para que seja possível explorar as vulnerabilidades existentes (e assim seguir para o próxima fase). Ao observar as estatísticas de incidentes reportadas pelo CERT.br (Figura 1b), as atividades de *scan* têm crescido ao longo do tempo e atualmente correspondem a praticamente 60% dos incidentes reportados. Portanto, identificar ações maliciosas no início é importante para inibir o progresso dos ataques (Figura 1a).

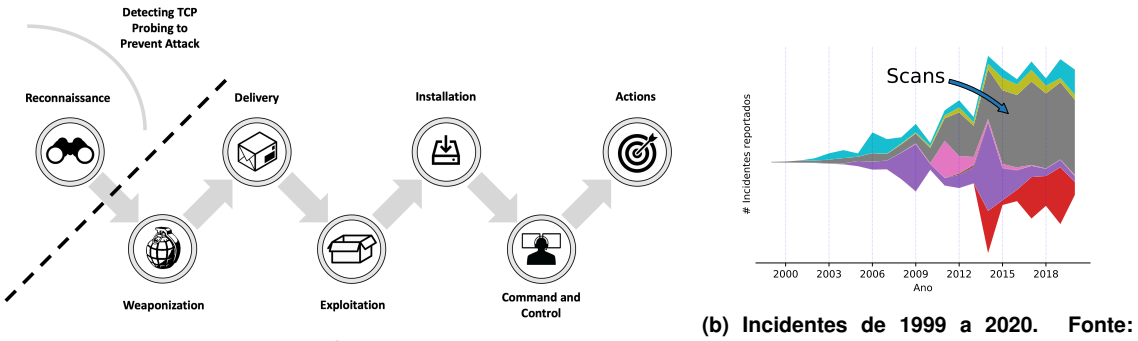
Estudos realizados sobre $7,41$ Petabytes de dados de tráfego, coletados de um grande *backbone* de rede entre 2004 e 2011, apontaram que cerca de $2,1\%$ dos pacotes atribui-se a *scan* [Glatz and Dimitropoulos 2012]. Isso revela a dimensão que esses ataques representam na Internet, cujo tráfego anual tem dobrado a cada dois anos desde 2005, podendo chegar a $2,3$ Zettabytes nos dias atuais, impulsionado também pela implantação do 5G e o crescimento dos dispositivos de Internet das Coisas[Idzikowski et al. 2018, Lee and Lee 2012, Al-Sarawi et al. 2020]. Porém, observa-se que notoriamente há poucas soluções focadas para dispositivos menores; soluções que permitam a implementação de políticas de segurança especializadas e baseadas em *zero-trust*.

[1] https://www.nytimes.com/2021/06/01/business/meat-plant-cyberattack-jbs.html

[2] https://www.bbc.com/news/business-57112371,https://www.nytimes.com/2021/05/10/us/politics/dark-side-hack.html

[3] https://g1.globo.com/rs/rio-grande-do-sul/noticia/2021/04/29/tj-rs-diz-que-sistema-de-informatica-do-tribunal-foi-alvo-de-ataque-hacker-e-muito-grave.ghtml

[4] https://www.cisoadvisor.com.br/stj-comunica-superacao-do-incidente-cibernetico-com-ransomware/



(a) Fases de um ataque cibernético do Cyber Kill Chain (CKC).

(b) Incidentes de 1999 a 2020. Fonte: `cert.br/stats/incidentes`

Figura 1. Na fase Reconnaissance, o atacante varre a rede a procura de um alvo. Atividade essa que soma 59% dos incidentes reportados ao CERT.br. Assim, restringi-las reduz a efetividade dos ataques.

Este trabalho apresenta a proposta de um firewall direcionado a dispositivos de Internet das Coisas (IoT) denominado T800. Esta proposta é inovadora e percebe-se que pouco foi explorado nessa linha, ainda mais quando se considera requisitos influenciados por arquiteturas baseadas em zero confiança (*zero-trust architecture*). Os experimentos demonstram que T800 é capaz de incorporar regras avançadas para filtragem de pacotes com árvores de decisão e *multilayer perceptron* na plataforma ESP32 (FreeRTOS, pilha de protocolos TCP/IP lwIP e SDK ESP-IDF) e TensorFlow. Os resultados indicam eficiência energética e melhoria no consumo de recursos computacionais ao remover, da pilha de protocolos, tráfego malicioso que seria processado pelo dispositivo. Ainda, o estudo de caso escolhido para a validação trata da identificação de pacotes de scan. Logo, um dispositivo com o T800 em execução seria capaz de passar despercebido durante a fase de reconhecimento (*reconnaissance*) para movimentação lateral de um ataque.

O restante do artigo está organizado da seguinte maneira: na Seção 2, são discutidos trabalhos relacionados. Na Seção 3, descrevemos a arquitetura do T800. Na Seção 4, descrevemos o projeto de experimentos. Na Seção 5, reportamos os resultados obtidos e a influência dos fatores. Na Seção 6, conclui-se este estudo e listam-se os trabalhos futuros.

2. Trabalhos Relacionados

Como observado em um domínio de aplicação mais amplo no contexto de redes de computadores, a caracterização, a identificação e a criação de regras para implementação e para filtragem de pacotes são bem estabelecidas. No entanto, encará-las como funções de rede componentizadas, a fim de movê-las do perímetro em direção ao *endpoint* de baixa potência computacional, apresenta dificuldades e restrições. Muito se deve ao fato de esses dispositivos terem funcionalidades limitadas e alta diversidade nos sistemas de software. Ainda, o estudo conduzido por [Viegas et al. 2021] mostrou que as soluções de caracterização dos modelos possuem um tempo de vida limitado, significando que há evidências de que os ataques possuem comportamento variante no tempo. As descobertas indicam uma vida útil de 6 (seis) semanas, com acurácia aceitável de 2 (duas) a 8 (oito) semanas. Logo, funcionalidades que permitem atualização das políticas em função de novos incidentes fazem parte dos requisitos de tais tipos de projetos.

Como apresentado por [Niedermaier et al. 2019], os dispositivos de baixa potência constituem uma parcela importante do cenário industrial. No estudo, é apresen-

tada uma arquitetura para realizar o processo de detecção de ataques de modo distribuído. O foco está no contexto de sistemas SCADA e PLC. Como descrito, há a caracterização clara do comportamento periódico de comunicação entre os dispositivos. Logo, ele possibilita a identificação de anomalias. No entanto, o estudo possui pouca generalidade, exigindo um esforço para implementar novas regras e aplicar em outros contextos.

Direcionados à plataforma Raspberry Pi, [Soe et al. 2020] apresentam uma solução de detecção de atividades maliciosas e provêem resultados que indicam baixa exigência computacional para execução. Muito embora, seja uma solução flexível, ainda assim, a generalização do método de filtragem e um mecanismo de atualização ficaram fora do escopo do trabalho. Isso faz com que o processo de adaptação a novos ataques, revocação de acesso e incorporação de novos dispositivos seja comprometido; portanto, dificulta a implementação de políticas de segurança baseadas em *zero-trust*. Em [Jan et al. 2019], os autores implementam um sistema focado em dispositivos de IoT utilizando SVM (*Support Vector Machine*) como modelo de identificação. Muito embora, o trabalho ganhe maior generalidade, os testes foram realizados em base de dados convencional, sem considerar a implementação em dispositivo real. Redes Neurais podem ser potenciais modelos implementados [Filus et al. 2021], ainda assim, há falta de mecanismo de atualização factível, pois deve-se considerar a natureza evolutiva dos ataques.

No sentido de trazer a implementação de um firewall para dispositivos de IoT, [Gupta et al. 2017, Ben Achballah et al. 2018] apresentam soluções para esse fim. Porém, há pouca flexibilidade para especificação de regras, sem a possibilidade de implementação de técnicas mais avançadas que usam aprendizado de máquina. Destaca-se ausência de um projeto genérico o suficiente para ser acoplado a diferentes sistemas, pois percebe-se uma variedade alta de sistemas operacionais e pilhas de protocolos TCP/IP para implementação. Assim, uma solução que seja capaz de interceptar pacotes em diferentes implementações ainda permanece em aberto. Mais ainda, essa solução deve fazer interface com diferentes sistemas operacionais. Com isso, é possível uma solução adaptável a diferentes sistemas. [Bertoli et al. 2021] apresenta a solução de um arcabouço capaz de descrever a criação de políticas de filtragem de pacotes. Trata-se de uma metodologia que auxilia no processo de caracterização de ataques conhecidos, em que se usa modelos heurísticos para esse fim (instanciados por meio de aprendizado de máquina). Uma contribuição importante trata da implementação do modelo em ambiente real, de modo que contrasta-se os resultados numéricos obtidos com a viabilidade de execução dos modelos e os desempenhos encontrados em sistemas alvos em operação. No entanto, percebe-se a falta de testes em plataformas baixa potência computacional. Nesse sentido, T800 aproveita dessas boas práticas detalhadas pelo trabalho de Bertoli *et. al* (2021) e faz um estudo de projeto, implementação e avaliação de algoritmos avançados em uma ESP32 (320 KiB de memória RAM), apontando desempenho adequado.

Como observado na literatura científica relacionada, muitos trabalhos focam em métodos pontuais com pouca generalidade e muitas vezes sem implementação real. T800 diferencia-se das demais pois apresenta um projeto adaptável a outras plataformas de baixo desempenho computacional, sendo passível de acoplamento a diferentes pilhas de protocolos TCP/IP e sistemas operacionais. Mais ainda, é possível perceber que a abordagem de um firewall para dispositivos de IoT permite que o tráfego malicioso seja descartado e com isso aumente a eficiência em dispositivos com escassez de recursos. Por

fim, demonstra-se a efetividade de implementação de políticas avançadas de filtragem representadas aqui através do porte da biblioteca TensorFlow. Isso habilitou a execução de algoritmos de árvore de decisão e *multilayer perceptron.* Assim, o presente trabalho avança o estado da arte possibilitando que políticas de segurança baseadas em *zero-trust* e direcionadas a filtragem de pacotes sejam viáveis no contexto de IoT.

3. T800 - Filtro de pacotes

o atual trabalho utiliza-se da proposta apresentada por Bertoli *et. al* (2021) e faz um estudo de projeto, implementação e avaliação de algoritmos avançados em um microcontrolador ESP32 (320 KiB de memória RAM), visando um baixo custo computacional devido ao seu direcionamento a dispositivos embarcados como a ESP32. Assim, nosso projeto de um firewall direcionado para dispositivos de Internet das Coisas (*Internet of Things*— IoT) deve seguir requisitos especializados para garantir a viabilidade de execução na plataforma computacional alvo. Esses requisitos envolvem o acoplamento no sistema de software básico e baixo *overhead* quando em execução. Devido à grande exposição de dispositivos de IoT, é necessário que políticas avançadas de segurança sejam realizadas, preferencialmente baseadas em uma arquitetura *zero-trust*.

Nesse contexto, apresenta-se o projeto T800 capaz de servir como componente habilitador para novas aplicações de segurança. A presente proposta é limitada à funcionalidade de filtragem de pacote *stateless*. No entanto, é possível realizar extensões para implementação de regras avançadas que incluem análise *stateful*. Isso posto, o presente trabalho tem como finalidade a implementação de um filtro de pacotes inteligente que pode ser aplicado no contexto de dispositivos embarcados. De maneira mais específica, o filtro foi construído para atuar na ESP-IDF [5], uma framework *open-source* de Internet das Coisas desenvolvida pela *Espressif*, cujo objetivo se dá pela possibilidade de implementar qualquer aplicação genérica para as plataformas com microcontroladores da família ESP32. Portanto, o *T800 Filtro de Pacotes* foi implementado e pensado seguindo tal estrutura de desenvolvimento, de forma que se tornam imprescindíveis resultados que promovam baixo consumo computacional e ao mesmo tempo uma estrutura genérica o suficiente para impedir ataques cibernéticos baseados em assinatura.

3.1. Arquitetura

O componente *T800*, ilustrado na Figura 2, implementa uma triagem do tráfego de rede recebido por um dispositivo ESP32. Isso é feito pela avaliação dos cabeçalhos de pacotes do protocolo IPv4 que permite diferenciar o tráfego como maligno ou benigno. Para isso, a instrumentação é realizada na função responsável por processar os pacotes da pilha de protocolos TCP/IP dentro do componente Lightweight IP (lwIP) na ESP-IDF.

Desta forma, o *T800* foi construído como um novo componente da *framework* ESP-IDF, tornando-se parte da biblioteca padrão. Adicionalmente, foram introduzidas duas novas dependências: *esp-nn*[6], uma biblioteca oficial da Espressif que implementa funções comuns para aprendizado de máquina otimizadas em Assembly; e a *tflite-lib*[7], biblioteca da Google que possibilita implementar modelos de Aprendizado de Máquina

[5]esp-idf: `https://github.com/espressif/esp-idf`
[6]esp-nn: `https://github.com/espressif/esp-nn`
[7]Tensorflow Lite: `https://github.com/tensorflow/tflite-micro`

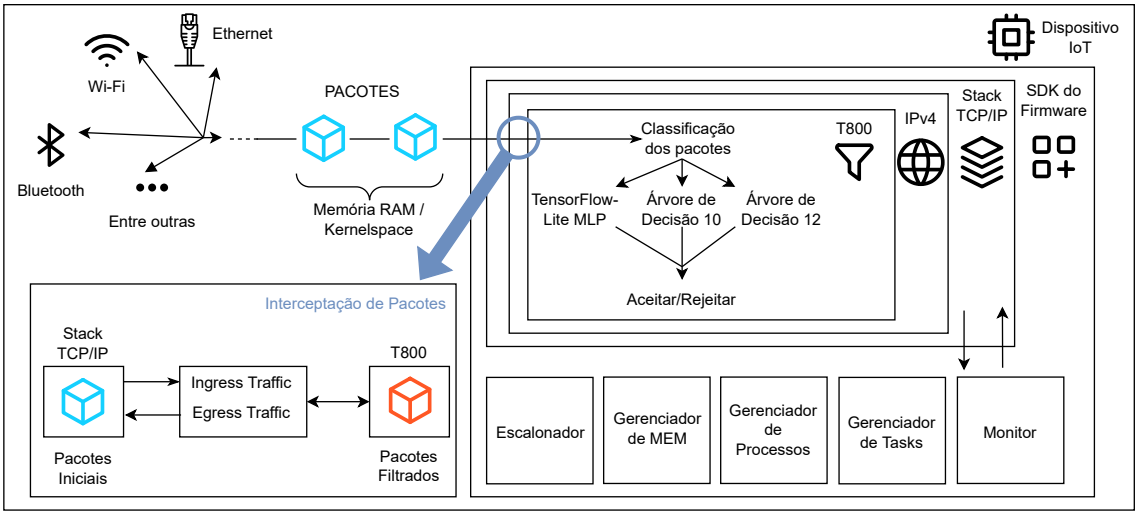


Figura 2. Arquitetura do filtro de pacotes T800.

desenvolvidos na *framework Tensorflow* com otimizações para transpor esses modelos para dispositivos com baixos recursos computacionais como por exemplo quantização dos pesos dos modelos finais.

Durante a execução no sistema embarcado, o *T800* passa por uma etapa de inicialização. Essa etapa de inicialização acontece pela disponibilização de uma estrutura de configuração inicial, incluindo tanto uma função que classifica pacotes, quanto a informação sobre o contexto (estático ou dinâmico) no qual essa função é executada. Esse tipo de contexto é necessário pois o *T800* poderá operar de duas maneiras diferentes, sendo elas, abordagens com ou sem a necessidade de guardar os estados anteriores de fluxos (*stateless* ou *stateful*). Por outro lado, caso essa inicialização não seja concluída, o *T800* não exerce nenhuma influência sobre o sistema. Após a etapa de inicialização, o *T800* é executado, passando a atuar diretamente nas camadas de Rede e de Transporte. A função que utiliza um modelo de aprendizado de máquina para realizar a classificação dos pacotes é escolhida e definida previamente na estrutura de configuração inicial. Sua entrada consiste em dados dos cabeçalhos de TCP e IP, que se encontram presentes nos pacotes recebidos pela ESP32, por meio de uma estrutura chamada de `pbuf` que é a representação de um pacote na stack lwIP [Dunkels 2001] e contém os *headers* para TCP/IP, camada de enlace, *payload* e, quando necessário, uma referência para outros pacotes que possam fazer parte de um encadeamento.

Por fim, o pacote pode ser ou não processado pela stack lwIP. Se o modelo o classificar como malicioso, o `pbuf` é descartado. Caso seja classificado como um pacote não-malicioso, o pacote segue seu processamento no ESP-IDF.

3.2. Políticas de filtragem de pacotes

Uma política de filtragem de pacotes pode ser definida como um conjunto de regras que especificam se um pacote deve receber permissão para continuar o seu tráfego natural dentro de uma camada de protocolos ou deve ser rejeitado. Dessa forma, considerando o quão geral pode ser essa definição, existem inúmeras maneiras de se construir tal política. Sua construção pode englobar desde regras selecionadas por um especialista de domínio, até modelos de aprendizado de máquina não supervisionados.

T800 utiliza o arcabouço (*framework*) denominado **AB-TRAP** (**A**ttack, **B**onafide, **T**rain, **R**ealization and **P**erformance) [Bertoli et al. 2021] que busca a concepção de mecanismos de proteção baseados em aprendizado de máquina capazes de contemplar toda a cadeia de desenvolvimento, desde o estudo da caracterização de operação normal até a implementação e avaliação de desempenho da solução proposta. Desse modo, as soluções para filtragem avançada de pacote testadas no T800 seguiram a metodologia AB-TRAP. Isto é, correspondem ao processo de geração de dados dos ataques e treinamento dos modelos até a implementação na placa ESP32. Então, as políticas de filtragem são dadas por modelos de aprendizado de máquina treinados *offline*, que talvez demandem atualizações periódicas de acordo com as necessidades de cada aplicação.

Nesse contexto, utilizaram-se três diferentes políticas de filtragem que foram selecionadas com a finalidade de testar a viabilidade operacional em uma plataforma baixo custo computacional. Dessa forma, apenas o desempenho computacional dos modelos já treinados foi levado em consideração. O custo incorrido pelo treinamento desses, foi desconsiderado nesse trabalho. Outro ponto foi o de verificar a aplicabilidade do T800 em preservar a capacidade de generalização das regras. Duas políticas foram feitas com estruturas de Árvores de Decisão com profundidades 10 (DT-10) e 12 (DT-12). Enquanto, a última foi construída com um *Multilayer Perceptron* (MLP) que possui duas camadas escondidas com 16 neurônios cada e uma camada de saída com dois neurônios

As Árvores de Decisão foram implementadas através de um encadeamento de estruturas condicionais, que se baseiam nos testes de atributos realizados pelos modelos originais. Essa simplicidade de interpretação e transposição dos modelos treinados para um encadeamento de estruturas condicionais tornam-nas propícias para o contexto de embarcados, pois elas incorrem em pouca utilização de recursos computacionais como processamento e memória. Dessa forma, esses modelos se encaixam bem no objetivo de implementação do componente T800, uma vez que minimizar o custo computacional nesse tipo de sistema é essencial, devido à escassez de recursos.

O *Multilayer Perceptron*, por outro lado, é especificado com uma função de ativação sigmoid para as camadas escondidas, e com uma função de ativação *softmax* para a camada de saída. Tanto sua implementação quanto seu treinamento foram feitos através da *framework Tensorflow* e inverteu-se o modelo final para poder ser utilizado na ESP32 através do *Tensorflow-Lite*. Isso fez com que esse modelo adotasse otimizações de espaço e processamento que são capazes de oferecer um desempenho satisfatório no contexto de implementações de mecanismos de inferência em dispositivos embarcados.

3.3. Conjunto de dados e treinamento dos modelos

Para a etapa de treinamento, as árvores de decisão empregaram a métrica de entropia como critério de divisão. Já a MLP foi treinada por 2000 *epochs* com o otimizador Adam, um *learning rate* de 1.10^{-5} e um *batch size* de 260.

O dataset aplicado para todas essas etapas de treinamento foi obtido pela metodologia **AB-TRAP** [Bertoli et al. 2021] com uma pequena alteração que consistiu na remoção do atributo `tcp.window_size`. Os pacotes de ataque são gerados através de uma *testbed* e de uma coleta do tráfego exclusivamente malicioso. Para isso, foi feito o uso de ferramentas de `TCP scan` como o `Zmap`, `masscan`, `Hping3`, `Unicorn Scan` e `NMap`, o que resultou em 86480 pacotes maliciosos. O comportamento legítimo,

por outro lado, foi obtido através de uma amostragem 103094 pacotes que foi realizada no tráfego que percorreu um enlace de internet que liga os Estados Unidos da América ao Japão, no dia 21 de novembro de 2019 (os fluxos desse enlace são abertos e atualizados periodicamente, ficando disponível como MAWILab [Fontugne et al. 2010]). Após isso, através de uma abordagem de *salting*, os tráfegos malignos e os benignos foram unidos em um único dataset. A partir desse processo foram obtidos os resultados de *F1-score* 0.93, 0.93 e 0.91 para a Árvore de Decisão com profundidade 10, a Árvore de Decisão com profundidade 12 e o *Multilayer perceptron*, respectivamente.

4. Planejamento de experimentos

Esta seção apresenta uma avaliação e uma análise de desempenho do T800. A aferição do desempenho do T800 foi feita através de quatro métricas que estão sumarizadas na Tabela 1. A primeira delas é a taxa de utilização da CPU dos dois núcleos presentes na ESP32. A segunda é a quantidade de memória alocada pelo T800 (somente a memória estática foi medida pois o componente não realiza alocação de memória dinâmica). A terceira é a taxa de recebimento de pacotes pela interface de Wi-Fi da placa. A última é o consumo de energia do dispositivo.

Tais medidas foram selecionadas pelas suas relevâncias no contexto de dispositivos embarcados. Softwares desenvolvidos para esse tipo de sistema costumam atuar em situações onde recursos computacionais são escassos. A ESP32 dispõe de apenas 320 KiB de memória RAM e de um processador com dois núcleos que possui um ciclo de clock de 240 MHz. Então, os possíveis aumentos na taxa de processamento ou na memória alocada do lwIP precisam ser verificados. Além disso, como o T800 atua no processamento dos pacotes recebidos dentro da implementação da pilha de protocolos TCP/IP do lwIP, alterações na taxa de recebimento de pacotes do dispositivo, se existirem, também precisam ser verificadas. Por fim, como alguns computadores embarcados são alimentados por baterias, também é necessário entender o impacto do T800 no consumo energético.

Considerando então essas métricas, uma modelagem de diferentes ambientes de execução foi utilizada para contrastar o funcionamento da ESP32 com e sem o T800 em diferentes situações. Isso foi feito com o planejamento de experimentos descrito na Tabela 2. Para simular diferentes cenários de execução, foram contemplados dois fatores. Um deles é a intensidade de tráfego benigno de rede recebido placa (I) que pode ser de 8Mbps (I0) ou de 16Mbps (I1). O outro é a presença de pacotes maliciosos no tráfego de rede destinado à ESP32 que varia entre os valores ausente (M0) e presente (M1). Dessa forma, um experimento é descrito pela cadeia IxMy.

Tabela 1. Métricas mensuradas na coleta e avaliação do *T800*.

Métrica	Descrição
CPU	Taxa de utilização da CPU (todos os núcleos)
MEM	Taxa de utilização de memória (apenas stack)
REDE	Taxa de utilização de REDE (apenas Wi-Fi)
ENERGIA	Consumo energético em mW

Tabela 2. Características do tráfego nos experimentos.

Fator	Níveis	Código
Intensidade do Tráfego	8Mbps ou 16Mbps	I0, I1
Tráfego Malicioso	Ausente ou Presente	M0, M1

A troca de mensagens entre os dispositivos para a realização dos testes ocorre entre uma ESP32 e uma máquina atacante que estão presentes na mesma rede. Para isso, elas se comunicam por meio do protocolo UDP para gerenciar as configurações de uma conexão TCP que ficará ativa por 180 segundos. No primeiro passo da execução (1), a ESP32 manda uma mensagem para a máquina atacante sinalizando o início do experimento. No segundo (2), o atacante envia um código que pode especificar a política de filtragem do T800 ou pode sinalizar que o componente não deve ser habilitado . Em seguida, no terceiro passo (3), a ESP32 responde comunicando que recebeu as informações necessárias e já realizou a sua configuração inicial. Nesse processo, dois servidores são inicializados na ESP32. O primeiro recebe todo o tráfego TCP da simulação, enquanto o segundo coleta as métricas de desempenho e as envia para a máquina atacante por UDP em períodos de 1 segundo. Após isso, no quarto passo (4), o tráfego de rede é gerado pela máquina atacante. Por fim, depois da coleta de todas as métricas, no quinto passo (5), a máquina atacante manda uma mensagem para a ESP32 que sinaliza o fim do experimento e, no sexto passo (6), a ESP32 envia uma resposta confirmando a finalização da simulação. Destaca-se ainda, que essa abordagem permite que a política de filtragem de pacotes seja alterada em tempo de execução, possibilitando a adaptabilidade da solução.

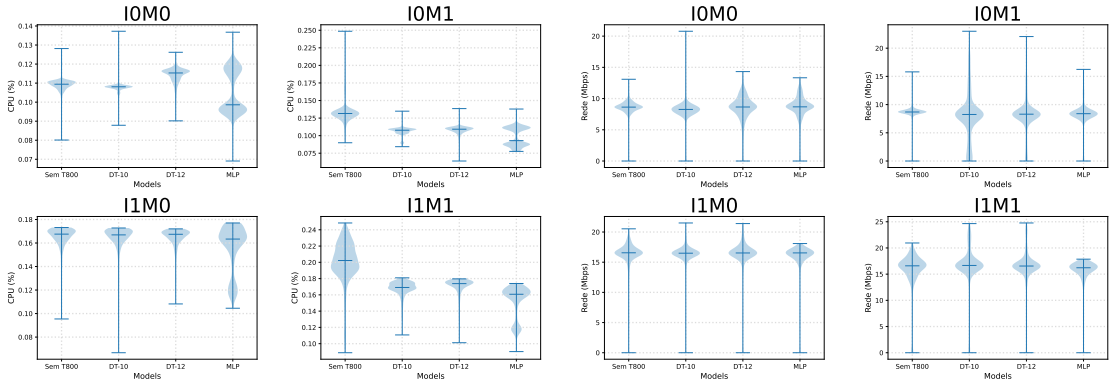
Ainda no contexto dessa implementação, para obter as métricas de desempenho que não envolvem o consumo energético, as funções usadas são disponibilizadas pela ESP32 por meio da interface com o kernel do sistema operacional FreeRTOS. O consumo energético é obtido através da medição dos canais de alimentação da ESP32 com o sensor de corrente e potência INA219[8]. Este sensor transmite as leituras dessas medidas por comunicação I2C para uma placa de leitura (Arduino Nano 33 BLE Sense) que se utiliza de comunicação serial com o computador atacante para registrar essas medições. Em conjunto com as medições de energia, a placa de leitura também monitora um sinal discreto da ESP32 (GPIO5) que é responsável por indicar o início e fim de um experimento, facilitando assim o pós-processamento destes dados. Além disso, os tráfegos normais com as intensidades de 8 Mbps e 16 Mbps são gerados através do *iperf v2.0* e os tráfegos maliciosos são simulados por meio do *nmap* direcionado a execução de ataques de *scanning*.

A partir disso, esse trabalho adotou um planejamento de experimentos fatorial completo. Os experimentos I0M0, I0M1, I1M0 e I1M1 foram executados não só para todas as políticas de filtragem descritas na Seção 3.2, mas também para a ESP32 sem o T800. Além disso, apesar de a métrica de consumo de energia ter sido obtida com apenas uma execução por experimento, todas as outras medidas foram coletadas através da produção de dez replicações para cada experimento. Finalmente, as especificações de

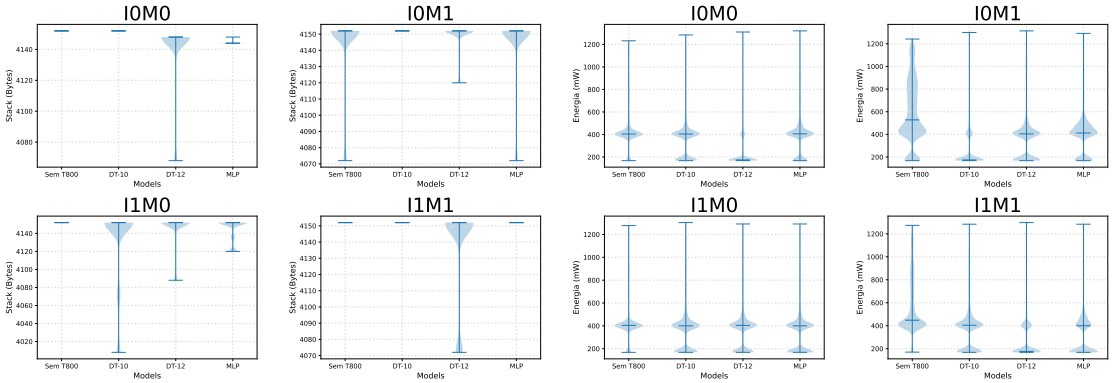
[8]Texas Instrument INA219: `https://www.ti.com/lit/ds/symlink/ina219.pdf`

Tabela 3. Descrição dos hardwares utilizados nos experimentos.

Descrição	CPU (GHz)	Memória	Sistema Operacional
Atacante	Intel i7-8565U-(4.6) x8	16 GiB	Manjaro Linux x86_64
ESP32 DevKit V1	Xtensa-(0.240) x2	320 KiB	FreeRTOS, ESP-IDF
Atacante	Intel i5-3337U-(2.7) x4	6 GiB	Ubuntu 20.04.2 LTS



(a) Utilização da CPU. **(b) Utilização da Rede.**



(c) Utilização de Memória. **(d) Gasto energético.**

Figura 3. Valores de consumo obtidos a partir dos experimentos.

hardware tanto da máquina atacante e da ESP32 utilizadas estão presentes na Tabela 3, junto com as do dispositivo usado para medir o consumo de energia durante as coletas.

5. Análise de desempenho e resultados

Todo o processo descrito na Seção 4 resultou nos valores apresentados na Figura 3. Nela, para cada métrica, agrupam-se os gráficos dos experimentos correspondentes. A representação da métrica ocorre por meio de *violinplots*. Neles, os pontos obtidos em todas as replicações são agrupados em um único gráfico. Dessa forma, o aspecto de série temporal foi ignorado. Para a métrica de energia foram considerados 180 pontos, enquanto para as outras foram considerados 1800 pontos e, assim, foi possível fazer uma análise de confiabilidade com mediana, máximo e mínimo. Por fim, é possível comparar os dados das execuções de diferentes políticas de decisão e das que não utilizam o T800. As barras no centro de cada *violinplot* representam as extremidades (valores máximo e mínimo) e a mediana (*tick* central).

Os dados de taxa de utilização da CPU estão presentes nos gráficos da Figura 3a. Para o experimento I0M0, as medianas dessa métrica estão dentro do intervalo $[0.10, 0.12]$. Para o experimento I0M1, as medianas da medida estão dentro do intervalo $[0.13, 0.90]$. Para o experimento I1M0, as medianas da taxa de utilização da CPU ficaram dentro do intervalo $[0.17, 0.16]$. Por fim, para o experimento I1M1, as medianas se encontram dentro do intervalo $[0.20, 0.16]$. Dessa forma, nota-se que não parece existir uma diferença de desempenho significativa entre as versões que utilizam o T800 com diferentes políticas de decisão e a que não utilizou o componente. Além disso, os dados da métrica de rede estão representados na Figura 3b. Nela, os experimentos com nível I0 apresentam uma mediana de velocidade de processamento de tráfego de 8 Mbps. Enquanto isso, para a mesma medida, as simulações com nível I1 possuem uma mediana de 16 Mbps. Isso mostra que essa métrica permanece próxima do valor da intensidade de tráfego benigno especificada para o experimento.

Ademais, a métrica de memória se encontra ilustrada na figura 3c. As medianas dos seus resultados permanecem constantes com valor 4152 bytes. Além disso, os intervalos interquartis mostram que essa métrica tem uma variação ínfima, na magnitude de dezenas de bytes, o que possibilita considerá-la desprezível. Por fim, a Figura 3d exibe os dados de consumo energético. Nela, a maioria das medianas se encontra no intervalo $[400, 412]$. Porém, em alguns experimentos, sem um padrão específico, a mediana se apresenta ou muito mais baixa (172) ou muito mais alta (528). Observa-se que há indícios de que a quantidade de replicação deve ser maior para a coleta dessa métrica.

5.1. Influência de Fatores

Para analisar a influência dos fatores na variação dos valores da taxa de utilização da CPU, considerou-se a presença do T800 (T) também como um fator do experimento. Ela assumiu, então, os valores presente (T1) ou ausente (T0). Além disso, os níveis foram mapeados para valores discretos de tal forma que $T0 = -1$ e $T1 = 1$ para qualquer fator A. Dessa forma, os experimentos e tal métrica de desempenho passaram a ser representados pela Tabela 4, onde as variáveis $y_{i,j}$ correspondem à média dos valores obtidos na replicação j do experimento i. A partir disso, foi realizada a regressão

$$\mathbf{XQ} = \bar{\mathbf{Y}}$$

que considera um modelo aditivo do sistema sob análise, na qual

$$\mathbf{Q} = \begin{bmatrix} q_0 \\ q_T \\ q_I \\ q_M \\ q_{TI} \\ q_{TM} \\ q_{IM} \\ q_{TIM} \end{bmatrix}, \qquad \mathbf{X} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & T & I & M & TI & TM & IM & TIM \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}, e \qquad \bar{\mathbf{Y}} = \begin{bmatrix} \bar{Y}_1 \\ \bar{Y}_2 \\ \bar{Y}_3 \\ \bar{Y}_4 \\ \bar{Y}_5 \\ \bar{Y}_6 \\ \bar{Y}_7 \\ \bar{Y}_8 \end{bmatrix}.$$

Em que $\mathbf{Q}$ representa o conjunto de parâmetro a serem estimados pelo algoritmo mínimos quadrados, $\mathbf{X}$ o conjunto de preditores correspondente aos fatores e suas interações, e $\bar{\mathbf{Y}}$ a média simples dos valores empíricos gerados pelos experimentos.. Assim, foi possível obter não somente as variações causadas pelos fatores e suas combinações (SS_A), mas também a variação atribuída ao erro experimental (SSE) e a variação total (SST).

$$SSE = \sum_{i=1}^{8} \sum_{j=1}^{1} 0(y_{ij} - \bar{y}_i)^2 \qquad SS_A = 2^3 \ 10 \ q_A$$

Tabela 4. Representação do planejamento fatorial completo com a adição do fator T e níveis discretos.

i	1	T	I	M	TI	TM	IM	TIM	$\mathbf{Y}_{i,1}$	...	$\mathbf{Y}_{i,175}$	$\bar{\mathbf{Y}}_i$
1	1	−1	−1	−1	1	1	1	−1	$y_{1,1}$	...	$y_{1,10}$	$\hat{y}_1$
2	1	−1	−1	1	1	−1	−1	1	$y_{2,1}$	...	$y_{2,10}$	$\hat{y}_2$
3	1	−1	1	−1	−1	1	−1	1	$y_{3,1}$	...	$y_{3,10}$	$\hat{y}_3$
4	1	−1	1	1	−1	−1	1	−1	$y_{4,1}$	...	$y_{4,10}$	$\hat{y}_4$
5	1	1	−1	−1	−1	−1	1	1	$y_{5,1}$	...	$y_{5,10}$	$\hat{y}_5$
6	1	1	−1	1	−1	1	−1	−1	$y_{6,1}$	...	$y_{6,10}$	$\hat{y}_6$
7	1	1	1	−1	1	−1	−1	−1	$y_{7,1}$	...	$y_{7,10}$	$\hat{y}_7$
8	1	1	1	1	1	1	1	1	$y_{8,1}$	...	$y_{8,10}$	$\hat{y}_8$

$$SST = SS_T + SS_I + SS_M + SS_{TI} + SS_{TM} + SS_{IM} + SS_{TIM} + SSE.$$

Por fim, a influência de um fator é dada pela divisão da variação que ele causa pela variação total. Dessa forma, tal análise de influência de fatores foi realizada para cada política de decisão, além de ser executada para os valores de todas as políticas de decisão agregados por uma média. Os resultados obtidos estão dispostos na Tabela 5. A partir dela, pode-se visualizar que o fator predominante no uso da CPU é o tráfego benigno, produzido pelo *iperf* (I). Portanto, a presença ou não de modelos de aprendizado de máquina possui pouca influência na variação da métrica CPU. Com isso, evidencia-se que a inclusão do T800 no sistema da ESP32 em situações de rede segura com tráfego pouco intenso ou em redes vulneráveis de tráfego intenso, não introduz um custo significativo.

Tabela 5. Resultados da análise de influência de fatores para a métrica CPU, em que T representa a presença do T800 na ESP32, I representa o tráfego de rede do *iPerf* e M a presença de tráfego malicioso.

Modelo	T	I	M	TM	TI	MI	TMI	Err
DT-10	0.06	0.77	0.06	0.0	0.05	0.0	0.0	0.06
DT-12	0.04	0.78	0.06	0.0	0.06	0.01	0.0	0.06
MLP	0.13	0.69	0.04	0.0	0.07	0.0	0.0	0.06
Agregados	0.04	0.78	0.08	0.0	0.04	0.0	0.0	0.06

6. Conclusão

Este artigo apresentou o filtro de pacotes T800 destinado a dispositivos de Internet das Coisas (IoT) com baixa potência computacional. A arquitetura proposta é adaptável a outras plataformas devido à instrumentação da pilha de protocolos utilizada, bastando identificar o ponto de interceptação dos pacotes. Ele permite que diferentes políticas de filtragem possam ser inseridas por meio da implementação em conjunto com o sistema operacional em questão. Demonstrou-se a efetividade dessa solução por meio da descrição da implementação com a plataforma de desenvolvimento ESP32 (FreeRTOS, pilha de protocolos TCP/IP lwIP e ESP-IDF). Os experimentos evidenciaram alta eficiência na abordagem escolhida, uma vez que a presença do T800 diminuiu o consumo de recursos

computacionais. A atuação do filtro de pacotes permitiu que o tráfego malicioso fosse descartado e possibilitou um impacto positivo no sistema de baixa potência computacional. Por meio de análise de influência de fatores, as políticas de filtragem baseadas em árvore de decisão e *multilayer perceptron* produziram um *overhead* operacional de 4% , ainda que melhore o desempenho geral do sistema. Portanto, os resultados proporcionam um projeto de *firewall* para plataformas de IoT de baixo consumo computacional.

Como trabalhos futuros, objetiva-se executar políticas de filtragem de pacotes *stateful*—uma vez que o mecanismo encontra-se implementado no T800, porém não testado neste artigo. Outro ponto de experimentação diz respeito a uma abordagem orientada a detecção de anomalias para verificar ataques ainda não caracterizados. Por fim, integrar uma arquitetura baseada em *zero-trust* direcionada a dispositivos de IoT em que o T800 serve como habilitador para tal solução. Todo código utilizado encontra-se no repositório `https://github.com/c2dc/t800-sbrc2022`.

Agradecimentos

Este trabalho tem apoio financeiro do Programa de Pós-graduação em Aplicações Operacionais—PPGAO/ITA, da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) processos #2020/09850-0 e #2021/09416-1, e do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) processo n.o 157021/2021-1.

Referências

Al-Sarawi, S., Anbar, M., Abdullah, R., and Al Hawari, A. B. (2020). Internet of things market analysis forecasts, 2020–2030. In *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, pages 449–453.

Ben Achballah, A., Ben Othman, S., and Ben Saoud, S. (2018). Fw_ip: A flexible and lightweight hardware firewall for noc-based systems. In *2018 International Conference on Advanced Systems and Electric Technologies (IC_ASET)*, pages 261–265.

Bertino, E. and Islam, N. (2017). Botnets and internet of things security. *Computer*, 50(2):76–79.

Bertoli, G. D. C., Júnior, L. A. P., Saotome, O., Dos Santos, A. L., Verri, F. A. N., Marcondes, C. A. C., Barbieri, S., Rodrigues, M. S., and De Oliveira, J. M. P. (2021). An end-to-end framework for machine learning-based network intrusion detection system. *IEEE Access*, 9:106790–106805.

Chacos, B. (2016). Major ddos attack on dyn dns knocks spotify, twitter, github, paypal, and more offline. `https://www.pcworld.com/article/3133847/ddos-attack-on-dyn-knocks-spotify-twitter-github-etsy-and-more-offline.html`. Publicado em 21/10/2016; acessado em 28/02/2022.

Dunkels, A. (2001). Design and implementation of the lwip tcp/ip stack. *Swedish Institute of Computer Science*, 2(77).

Filus, K., Domańska, J., and Gelenbe, E. (2021). Random neural network for lightweight attack detection in the iot. In Calzarossa, M. C., Gelenbe, E., Grochla, K., Lent, R., and Czachórski, T., editors, *Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 79–91, Cham. Springer International Publishing.

Fontugne, R., Borgnat, P., Abry, P., and Fukuda, K. (2010). Mawilab: combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. In *Proceedings of the 6th International COnference*, pages 1–12.

Glatz, E. and Dimitropoulos, X. (2012). Classifying internet one-way traffic. In *Proceedings of the 2012 Internet Measurement Conference*, IMC'12.

Gupta, N., Naik, V., and Sengupta, S. (2017). A firewall for internet of things. In *2017 9th International Conference on Communication Systems and Networks (COMSNETS)*.

Idzikowski, F., Chiaraviglio, L., Liu, W., and van de Beek, J. (2018). Future internet architectures and sustainability: An overview. In *2018 IEEE International Conference on Environmental Engineering (EE)*, pages 1–5.

Jan, S. U., Ahmed, S., Shakhov, V., and Koo, I. (2019). Toward a lightweight intrusion detection system for the internet of things. *IEEE Access*, 7:42450–42471.

Klint, C. (2021). These are the top risks for business in the post-covid world. `https://www.weforum.org/agenda/2021/01/building-resilience-in-the-face-of-dynamic-disruption/`. Publicado em 19/01/2021; acessado em 28/02/2022.

Lee, Y. and Lee, Y. (2012). Toward scalable internet traffic measurement and analysis with hadoop. *SIGCOMM Comput. Commun. Rev.*, 43(1):5–13.

Lobo, S. (2019). Understanding the cost of a cybersecurity attack: The losses organizations face. `https://hub.packtpub.com/understanding-the-cost-of-a-cybersecurity-attack-the-losses-organizations-face/`. Publicado em 31/03/2019; acessado em 28/02/2022.

McLennan, M. (2021). The global risks report 2021 16th edition. `https://www.weforum.org/reports/the-global-risks-report-2021`.

McMillen, D. (2021). Internet of threats: Iot botnets drive surge in network attacks. `https://securityintelligence.com/posts/internet-of-threats-iot-botnets-network-attacks/`. Publicado em 22/04/2021; acessado em 28/02/2022.

Niedermaier, M., Striegel, M., Sauer, F., Merli, D., and Sigl, G. (2019). Efficient intrusion detection on low-performance industrial iot edge node devices.

Soe, Y. N., Feng, Y., Santosa, P. I., Hartanto, R., and Sakurai, K. (2020). Implementing lightweight iot-ids on raspberry pi using correlation-based feature selection and its performance evaluation. In Barolli, L., Takizawa, M., Xhafa, F., and Enokido, T., editors, *Advanced Information Networking and Applications*, Cham. Springer Intl. Publish.

Viegas, E., Santin, A. O., and Abreu Jr, V. (2021). Machine learning intrusion detection in big data era: A multi-objective approach for longer model lifespans. *IEEE Transactions on Network Science and Engineering*, 8(1):366–376.

Yadav, T. and Rao, A. M. (2015). Technical aspects of cyber kill chain. In Abawajy, J. H., Mukherjea, S., Thampi, S. M., and Ruiz-Martínez, A., editors, *Security in Computing and Communications*, pages 438–452, Cham. Springer International Publishing.