AVALIAÇÃO DE PROCESSAMENTO LOCAL BASEADA EM TRAÇOS DE EXECUÇÃO, APLICADA A PROTOCOLOS DE COMUNICAÇÃO E A SENSORES DA INTERNET DAS COISAS.

Rafael Cruz Salles

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientador: Ricardo Cordeiro de Farias

Rio de Janeiro

Maio de 2023

AVALIAÇÃO DE PROCESSAMENTO LOCAL BASEADA EM TRAÇOS DE EXECUÇÃO, APLICADA A PROTOCOLOS DE COMUNICAÇÃO E A SENSORES DA INTERNET DAS COISAS.

Rafael Cruz Salles

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientador: Ricardo Cordeiro de Farias

Aprovada por: Prof. Ricardo Cordeiro de Farias
Prof. Claudio Esperança
Prof. Renato Fontoura de Gusmão Cerqueira
Prof. Karin Koogan Breitman
Prof. Flávia Coimbra Delicato

RIO DE JANEIRO, RJ – BRASIL
MAIO DE 2023

Salles, Rafael Cruz

Avaliação de processamento local baseada em traços de execução, aplicada a protocolos de comunicação e a sensores da Internet das Coisas./Rafael Cruz Salles. – Rio de Janeiro: UFRJ/COPPE, 2023.

XVII, 151 p.: il.; 29, 7cm.

Orientador: Ricardo Cordeiro de Farias

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2023.

Referências Bibliográficas: p. 134 – 148.

1. Avaliação de Desempenho. 2. Internet das Coisas. 3. Kernel. 4. Arquitetura e Sistemas Operacionais. I. Farias, Ricardo Cordeiro de. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

À minha esposa e filha pelo tempo ausente. Aos meus pais por sempre estimularem meu desenvolvimento

Agradecimentos

Um agradecimento a todos meus familiares e amigos que de alguma forma acabaram por ficar em segundo plano durante o desenvolvimento deste trabalho.

Um agradecimento especial para o Professor Ricardo Cordeiro de Farias que aceitou me orientar durante meu doutorado. Muito obrigado pela ajuda, suporte e ensinamentos ao longo destes anos.

Aos membros da Banca, por aceitarem fazer parte de minha trajetória acadêmica, bem como por suas críticas, sempre construtivas e bem-vindas.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

AVALIAÇÃO DE PROCESSAMENTO LOCAL BASEADA EM TRAÇOS DE EXECUÇÃO, APLICADA A PROTOCOLOS DE COMUNICAÇÃO E A SENSORES DA INTERNET DAS COISAS.

Rafael Cruz Salles

Maio/2023

Orientador: Ricardo Cordeiro de Farias

Programa: Engenharia de Sistemas e Computação

A Internet das Coisas (IoT) é um paradigma onde bilhões de objetos comuns com poder de processamento, sensoriamento e comunicação são capazes de atuar ao nosso redor de maneira quase imperceptível.

À medida que novos dispositivos, aplicativos e protocolos de comunicação são propostos para o contexto da IoT, sua avaliação, comparação, ajuste e otimização se tornam cruciais. Tal cenário estimula a pesquisa por novas metodologias de avaliação que contribuam para uma melhor eficiência na utilização dos recursos computacionais disponíveis.

Esta tese propõe um método para avaliação do processamento local de sensores IoT com múltiplos processadores, mediante a análise e classificação de traços de execução obtidos junto ao escalonador de processos.

No âmbito da IoT, onde a comunicação por rede sem fio é predominante, verifica-se na literatura a necessidade de uma melhor avaliação do processamento local realizado por sensores em relação a novos protocolos de comunicação que estão sendo propostos. Neste contexto, a avaliação do processamento local mostra-se fortemente influenciada pelo estado da rede, muitas vezes comprometendo a qualidade dos resultados obtidos.

Como forma de aplicarmos o método de avaliação de processamento local proposto nesta tese a um problema real em aberto, realizamos sua implementação em um sensor IoT de prateleira, com mais de um processador e com sistema operacional de tempo real (FreeRTOS). A implementação realizada é então utilizada para avaliar um protocolo de comunicação segura (TLS) com o objetivo de verificarmos a capacidade de aplicação do método na produção de resultados comparáveis e não influenciados pelo estado da rede.

As principais contribuições desta tese são: uma metodologia para detalhamento e avaliação do processamento local de sensores IoT e a apresentação de que o método proposto, quando aplicado à avaliação de protocolos ou aplicações que se comuniquem em rede, é capaz de produzir resultados mais confiáveis, comparáveis e sem influência do estado da rede.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

EVALUATION OF LOCAL PROCESSING BASED ON EXECUTION TRACES, APPLIED TO COMMUNICATION PROTOCOLS AND SENSORS OF THE INTERNET OF THINGS.

Rafael Cruz Salles

May/2023

Advisor: Ricardo Cordeiro de Farias

Department: Systems Engineering and Computer Science

The Internet of Things (IoT) is a paradigm where billions of common objects with processing, sensing, and communication capabilities are able to act around us almost imperceptibly.

As new devices, applications, and communication protocols are proposed for the context of IoT, their evaluation, comparison, adjustment, and optimization become crucial. Such a scenario stimulates research for new evaluation methodologies that contribute to better efficiency in the use of available computing resources.

This thesis proposes a method for evaluating the local processing of IoT sensors with multiple processors, through the analysis and classification of execution traces obtained from the process scheduler.

In the scope of IoT, where wireless network communication is predominant, there is a need for better evaluation of the local processing performed by sensors in relation to new communication protocols being proposed. In this context, the evaluation of local processing is strongly influenced by the state of the network, often compromising the quality of the results obtained.

As a way to apply the proposed method for local processing evaluation to a real open problem, we implemented it on a shelf IoT sensor with more than one

processor and a real-time operating system (FreeRTOS). The implementation is then used to evaluate a secure communication protocol (TLS) with the aim of verifying the method's ability to produce comparable results that are not influenced by the network state.

The main contributions of this thesis are: a methodology for detailing and evaluating the local processing of IoT sensors and the presentation that the proposed method, when applied to the evaluation of protocols or applications that communicate over a network, is capable of producing more reliable, comparable results without being influenced by the network state.

Sumário

Lista de Abreviaturas

AMQP Advanced Message Queuing Protocol.

CoAP *Constrained Application Protocol*.

CPS Sistemas cyber-físicos.

DNS Domain Name System.

HiTL *Human In The Loop*.

HTTP Hypertext Transfer Protocol.

IEEE Institute of Electrical and Electronics Engineers.

IoE Internet de Todas as Coisas.

IoT Internet das Coisas.

LCD Tela de cristal líquido.

MANET Rede móvel ad hoc.

MQTT *MQ Telemetry Transport*.

OSI Open Systems Interconnection model.

REST Representational State Transfer.

RFID Identificação por rádio frequência.

SoC Sistemas em um único circuito integrado.

TCP/IP Protocolo de controle de transferência / protocolo internet.

VANETs Redes veiculares ad-hoc.

WSN Redes de sensores sem fio.

XMPP *Extensible Message and Presence Protocol*.

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

IoT nos apresenta um futuro onde a Internet aparecerá como uma rede sem limites. Objetos físicos mais comuns, como lâmpadas, passaportes e sapatos, estarão na mesma rede e serão capazes de enviar informações sobre o nosso redor e também de atuar em nosso meio. Como consequência, a Internet passará a dispor, em tempo real, de informações sobre o mundo físico que nos cerca e dos objetos que compõem tal rede, incluindo, muitas vezes, sua geolocalização.

O conceito de IoT surgiu há quase duas décadas. Entretanto, a partir de meados de 2015, verificamos um expressivo aumento no número de grupos de estudo, comitês, fundos de pesquisa e congressos envolvendo setores acadêmicos, empresariais e governamentais. Como consequência, diversas publicações e linhas de pesquisa têm surgido em todo o mundo [4–6]. No Brasil, também se verifica o engajamento de diversos setores, especialmente após a consulta pública apresentada pelo governo em 2017 [7].

Como resultado de tal cenário, estima-se que bilhões de dispositivos serão introduzidos na Internet. O contexto desta tese compreende os dispositivos sensores de IoT, aqueles responsáveis pelo sensoriamento, atuação com o meio e realização de comunicação acerca destes. Dentro deste escopo, um ambiente computacional deve ser capaz de operar ligado a baterias e como tal procura-se eficiência no uso de seus recursos, não só como forma de evitar seu uso desnecessário, mas também como exigência para que esteja apto a operar pelo máximo de tempo possível sem eventual intervenção para recarga ou troca de baterias.

Dois tipos de análises de execução são comumente encontradas na literatura:

estática e dinâmica. O primeiro, se dá sem a real execução do programa, com base em modelos de hardware. O segundo, mediante real execução do programa e coleta de informações.

Esta tese propõe um método para a avaliação dinâmica do processamento local realizado por sensores IoT. Nesse contexto, em cenários onde há possibilidades de escolha (por exemplo, configurações ou códigos distintos), medir o processamento utilizado contribui para determinar aquele que demanda menos processamento. O método proposto se baseia em dados contidos em traços de execução coletados durante o escalonamento de processos.

Em relação às metodologias de avaliação de experimentos comumente empregadas na literatura, os resultados são obtidos por meio da introdução de códigos que permitem a medição do tempo de execução de determinado trecho do programa. No entanto, esse tipo de avaliação não revela, por exemplo, o comportamento da execução dentro do sistema operacional, o que pode levar a resultados influenciados por fatores externos ao experimento sendo avaliado. Em certos casos, por exemplo, um eventual escalonamento de processos em menor tempo poderia reduzir o processamento de rotinas do sistema operacional relacionadas à troca de contexto e, consequentemente, produzir resultados diferentes. Nesse exemplo, seria interessante verificar esse fato.

Para aplicar a metodologia proposta a um problema real, esta tese utiliza-a no problema de avaliação do processamento local utilizado por aplicações ou protocolos em cenários onde ocorra comunicação em rede. À medida que diversos protocolos são propostos, a avaliação do processamento local por eles utilizado torna-se de grande interesse. Nesse cenário, a obtenção de métricas que avaliem o processamento local está sujeita à influência de fatores externos ao dispositivo, decorrentes de mudanças constantes no estado físico da rede sem fio. Tais fatores, quando não tratados corretamente, podem contribuir para a produção de resultados que não refletem o real processamento local demandado pelas aplicações ou protocolos avaliados.

No contexto do problema avaliado, os resultados obtidos através do método desenvolvido possibilitaram a obtenção de avaliações de processamento relativas à comunicação em rede sem a influência das variações inerentes ao uso desta. Mediante o uso do método e ao seu detalhamento provido, pudemos verificar de que forma

tais variações influenciam os resultados obtidos quando não corretamente consideradas, avaliadas, isoladas e retiradas dos resultados finais apresentados para fins de comparação.

As principais contribuições desta tese são:

- Desenvolvimento de um método para avaliação dinâmica e detalhamento do processamento local realizado por dispositivos IoT.
- Aplicação do método proposto ao problema de avaliação do processamento local de aplicações ou protocolos quando estes realizam comunicação em rede. Implementação do método em dispositivo com arquitetura em sistema de chip único e que utiliza sistema operacional em tempo real escalonável.
- Emprego do método proposto na identificação de comportamentos do dispositivo que não atendam às características exigidas ou divulgadas.
- Pesquisa acerca de estudos recentes que tenham como objetivo promover, nas diversas camadas de rede, a diminuição do processamento local realizado por sensores IoT a fim de que realizem comunicação segura.
- Apresentação, baseada em resultados quantitativos, da influência do estado da rede nos resultados encontrados, quando não corretamente tratado. Apresentação de estudos que, devido à impossibilidade do método utilizado, não procederam na verificação de tal influência em seus resultados.

1.1 Organização da Tese

Os Capítulos 2, 3 e 4 apresentam, no âmbito de IoT, conceitos relacionados ao estudo de redes de computadores e arquitetura de computadores e sistemas operacionais (ASO). O leitor com maior familiaridade em tais tópicos pode proceder diretamente ao Capítulo 5.

No Capítulo 2, apresentamos o paradigma de IoT para verificarmos o contexto em que se insere dentro das grandes áreas de estudos tradicionais de computação.

No Capítulo 3, apresentamos revisão conceitual e pesquisa, no âmbito de IoT, referente ao estudo de comunicação de dispositivos em rede. Também nesse capítulo

apresentaremos arquitetura de sistemas distribuídos proposta ao paradigma de IoT para então contextualizarmos esta tese dentro deste modelo. Passaremos, então, ao contexto de comunicação segura, identificando ao logo das diversas camadas OSI estratégias para obtenção desta.

Revisão e pesquisa realizada no âmbito de arquitetura de computadores, sistemas operacionais e IoT serão apresentados no Capítulo 4. Os conceitos nesse apresentados serão utilizados na construção do método de avaliação proposto. Também nesse capítulo apresentamos estudos, referentes a metodologias de análise de processamento, revistos durante a elaboração do método aqui desenvolvido.

No Capítulo 5, apresentamos o método de avaliação desenvolvido nesta tese e o ambiente utilizado para sua implementação de uso. Também são apresentados os detalhes de sua aplicação ao contexto da obtenção de avaliação do processamento referente à comunicação em rede.

Através do emprego do método proposto apresentamos no Capítulo 6, avaliação, no ambiente experimental utilizado, referente à comunicação em rede. Avaliamos comunicação segura realizada, na camada de aplicação, mediante o emprego de TLS. Nesse experimento avaliaram-se, com distintas durezas de segurança e conjuntos criptográficos, as fases de troca de chaves, a autenticação, e a troca de informações. Posteriormente, em outro experimento, avaliou-se a troca de informações sem criptografia. Este como forma de avaliarmos o processamento, caso se utilize somente a segurança provida pela camada de enlace deixando a cargo do próximo nó o processamento referente ao estabelecimento de comunicação segura fim a fim.

Seguimos no Capítulo 7 apresentando trabalhos relacionados.

Por fim, apresentamos nossas conclusões e trabalhos futuros no Capítulo 8.

Capítulo 2

Paradigma IoT

2.1 Introdução

Diversos trabalhos de distintas áreas de estudo se apresentam como relacionados ao tema IoT. Neste capítulo apresentamos uma revisão que nos permita estabelecer uma melhor relação entre estudos no âmbito de IoT e outras áreas de estudo tradicionalmente existentes. O principal objetivo é contextualizarmos nossa pesquisa tanto em relação à literatura direcionada ao tema IoT quanto a seu mapeamento em relação a áreas clássicas de estudos.

2.2 Contextualização.

O termo IoT foi proposto em 1999 pelo empreendedor britânico Kevin Ashton durante uma apresentação realizada a uma empresa [8]. Inicialmente, tratava-se de um conceito onde futuramente tudo estaria conectado à internet. Também por volta de 1999 a tecnologia de Identificação por rádio frequência (RFID) surge como forma de sensoriamento e identificação, a pequenas distâncias (alguns metros), da presença de objetos através do uso de etiquetas de baixo custo. O termo IoT, desde sua primeira menção e em algumas literaturas que se seguiram, costumava aparecer em conjunto com a tecnologia RFID, uma vez que este tipo de tecnologia de identificação ganhava grande empregabilidade. Apesar de, à época, RFID operar a distância de alguns metros sem a necessidade de fios, a transmissão dos dados obtidos por esse e outros sensores ainda se dava, em grande parte, por fios.

No início dos anos 2000, se dá a popularização de tecnologias sem fio. Em [9] os autores fazem um apanhado de diversos conceitos e tecnologias que começam a moldar o conceito de IoT. IoT engloba o conceito de comunicação ubíqua apresentado em 1999 [10], [11] onde dispositivos omnipresentes se comunicariam através de uma rede. Em [11], os autores apresentam as necessidades tecnológicas necessárias à computação ubíqua como sendo: computadores de baixo custo, aplicação específica e uma rede de comunicação.

Além do conceito de computação ubíqua, muitos outros pontos relevantes à área de IoT e afins são reunidos e apresentados em [9], dentre os quais destacamos:

- **Comunicação sem fio**: Por razões de flexibilidade, adaptabilidade e mobilidade será o meio de comunicação dominante em IoT.
- **Euforia** (termo "hype" em inglês): Necessidade de maior pesquisa na área de IoT, a fim de que se identifique que não se trata de uma tecnologia sem relevância.
- **Não é um conceito revolucionário**: trata-se de uma área que provém de uma evolução de estudos e tecnologias.
- **Evolucionário**: um grande número de dispositivos conectados trarão novos problemas e necessidades que farão com que IoT necessite de estudos e inovações que só serão alcançadas a longo prazo.
- **Empregabilidade**: É importante que instituições acadêmicas e indústria mantenham contato de forma a produzirem material que venha a ser empregado em problemas reais.

À época de [9] (anos 2000), verificamos que, do ponto de vista tecnológico dos dispositivos, os principais desafios a serem superados eram, dentre outros, tamanho, custo e eficiência energética. Em relação às redes de comunicação à época, um dos desafios apontados era a inexistência de dispositivos capazes de armazenar alguma implementação do modelo Open Systems Interconnection model (OSI) de 7 Camadas [12]. Os autores colocam, como um desafio, a necessidade de criação de novos protocolos de rede a serem utilizados por tais dispositivos.

Ainda em [9] os autores questionam se algum dia seria possível a existência de um único dispositivo pequeno, de baixo custo também ser capaz de realizar a comunicação e ainda sim possuir eficiência energética.

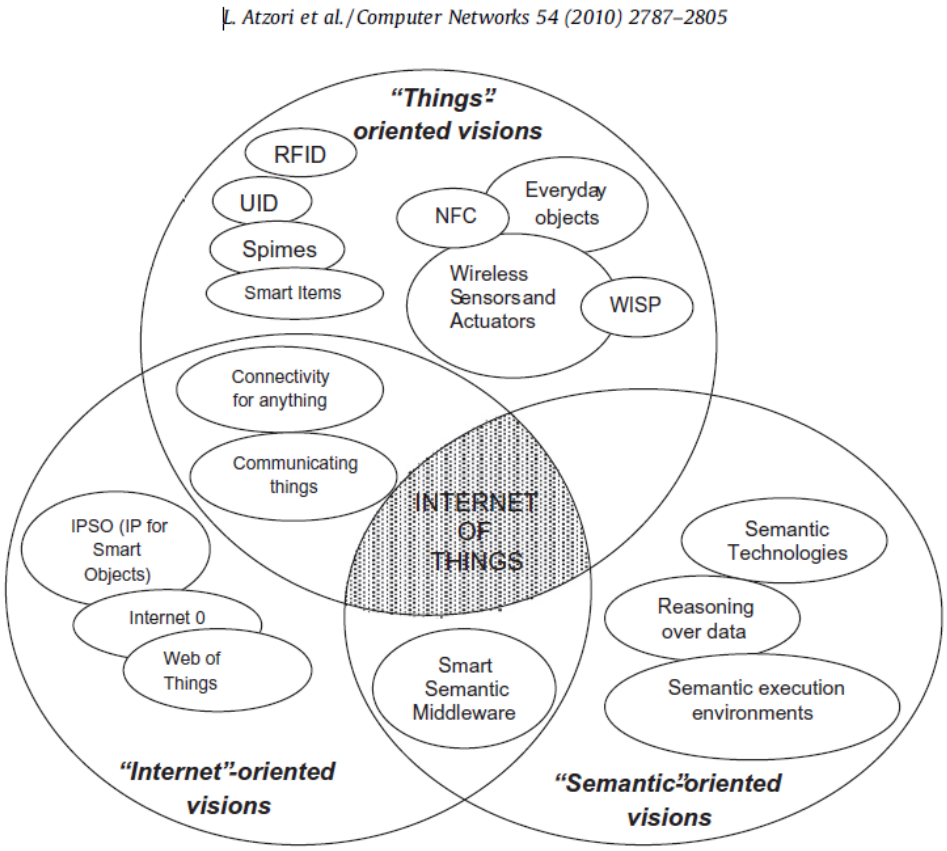


Figura 2.1: IoT a partir de diferentes visões: orientada a Internet, orientada as Coisas (sensores) e orientada a semântica. Extraído de [13]

2.2.1 Classificação dos estudos segundo sua Visão conceitual.

No contexto da IoT, a ideia de visões refere-se às diferentes perspectivas ou abordagens que podem ser adotadas para entender e conceituá-la. As visões representam diferentes maneiras de olhar para a IoT, enfatizando diferentes aspectos ou dimensões do paradigma. Em 2010, os autores de [14] reúnem publicações apresentadas em um congresso de IoT para então, em [13], apresentarem uma definição mais formal para o paradigma IoT. Mediante a análise de diversas publicações os autores de [13] descrevem IoT como um paradigma proveniente da interseção de estudos realizados a partir de três visões distintas [Figura 2.1] : orientada à Internet, orientada às Coisas (sensores) e orientada à semântica. Uma visão orientada à semântica tem como

motivador o alto volume de dados originados por dispositivos IoT e como tal procura métodos apropriados à representação, busca, organização e armazenamento de tais dados. Já em estudos, cuja visão se dá sob a ótica das Coisas, o enfoque surge a partir de "objetos", dispositivos individuais capazes de efetuar sensoriamento, atuar com o meio e enviar e receber informações da Internet. Em uma visão orientada à Internet, verifica-se o enfoque em questões de rede e em eventuais modificações necessárias para que a Internet absorva a entrada dos novos agentes IoT que passarão a utilizá-la.

Verifica-se [14] que em muitas áreas de pesquisa até então tratadas como distintas começavam a ser vistas como parte do paradigma IoT, como por exemplo, Rede móvel ad hoc (MANET), Redes de sensores sem fio (WSN), Redes veiculares ad-hoc (VANETs).

Nos anos seguintes a 2010 diversas publicações começaram a ser apresentadas em torno do tema IoT. A presença do ser humano em redes IoT (*Human In The Loop* (HiTL)) é abordada em [15]. Aplicações IoT que envolvam seres humanos são classificadas pelos autores como: (i) aplicações influenciadas ativamente por pessoas; (ii) aplicações que realizem sensoriamento passivamente em seres humanos e (iii) aplicações híbridas de (i) e (ii). A interação de uma pessoa com uma cadeira de rodas através de monitor Tela de cristal líquido (LCD) para ajuste de postura ou com um controle para se locomover exemplificam aplicações com intervenção direta de humanos (i). Em tal classificação o ser humano tem alguma interface direta influenciando no sistema IoT em questão. Pulseiras que medem batimentos cardíacos ou medem passos são exemplos de dispositivos passivos (ii). Smartphones, em um cenário onde o usuário eventualmente habilita e desabilita tais funções podem ser visto como um híbrido (iii).

Sistemas IoT capazes de atuar no meio com base na análise dos dados sensoreados acrescenta ao estudo de IoT o conceito de Circuito Fechado ou retroalimentado (closed Loop) [16]. Dispositivos que meçam sinais, enviem os dados a uma central e que sinalizem, por sinal sonoro, que determinado paciente deve ir ao médico, exemplificam a retroalimentação. Por sua vez, uma pulseira que simplesmente exiba batimentos em um visor e envie tais informações por e-mail se enquadra na classe chamada Circuito Aberto ou sistema sem retroalimentação. A mera existência de

transmissão das informações não caracteriza retroalimentação. Para que seja considerado um sistema retroalimentado o dispositivo deve atuar com base em análise realizada externamente a ele, considerando-se dados ou ações não presentes no dispositivo.

O contexto do presente trabalho se dá no âmbito de dispositivos IoT retroalimentados, havendo ou não interação ativa com seres humanos. Também se dá no âmbito das visões chamadas de Internet e Coisas. Neste sentido faremos uma revisão, nos capítulos seguintes, dos conceitos que usaremos, procurando organizá-los dentros destas duas Visões.

2.3 Outras denominações

Na literatura observamos que a adição de inteligência e conectividade a dispositivos eletrônicos tem recebido diversos nomes em distintos campos de atuação, tais como, Casas Inteligentes, Construções Inteligentes, Grides Inteligentes, e-Health, Internet de Todas as Coisas (IoE).

Em algumas literaturas, especialmente no contexto acadêmico, o termo Sistemas cyber-físicos (CPS) também é utilizado. Em [17] os autores apresentam sua visão quanto à diferença entre as áreas de estudo conhecidas como WSN, IoT e CPS. Em redes de sensores sem fio não há necessidade de identificação única do objeto que efetua o sensoriamento, sendo sua ênfase na percepção da informação e coleta. IoT, segundo os autores, incorpora o uso de Internet, transmissão confiável e processamento inteligente da informação sensoreada. O conceito de CPS, por sua vez, acrescenta a retroalimentação ao ecossistema de IoT, permitindo que se incorpore a habilidade de controlar o ambiente. Sistemas IoT teriam ênfase no sensoriamento através da Internet enquanto que nos CPS existiria uma análise em tempo real dos dados sensoreados e uma reação baseada em tal análise.

Em [18], no ano de 2008, o autor considerado criador da definição de CPS a descreve como sendo a integração entre computação e processos físicos: "computadores e redes monitorando e controlando processos físicos, normalmente com retroalimentação, onde o processo físico afeta a computação e vice-versa". Já em [19], no ano de 2015, o mesmo autor, juntamente a outros, classifica aplicações IoT em dois

grupos: aplicações coletoras de dados e aplicações em tempo real. No conjunto de aplicações IoT em tempo real os autores enquadram as aplicações "reativas ao ambiente" e analisam o uso de tecnologias na nuvem responsáveis pelo processamento das informações sensoreadas.

Ambos os termos IoT e CPS são muitas vezes descritos e aceitos como complementares e sinônimos [20]. O termo IoT tem sua origem em estudos da Comissão Europeia, enquanto que CPS aparece inicialmente em estudos da Fundação Nacional de Ciências Americana.

Como apresentado, muitas vezes não há uma distinção entre CPS e IoT havendo uma grande superposição entre ambas as denominações. No presente trabalho, optamos por utilizar a designação de IoT por verificarmos que o termo IoT apresenta um número maior de publicações na biblioteca digital da Institute of Electrical and Electronics Engineers (IEEE).

Capítulo 3

Comunicação em rede no âmbito de IoT

3.1 Introdução

No capítulo anterior, introduzimos IoT como uma interseção de estudos oriundos de três grandes áreas, dentre as quais apresentamos a chamada "Visão orientada à Internet". Neste capítulo, abordaremos os conceitos relacionados a tal visão. Começaremos apresentando uma arquitetura de sistemas distribuídos chamada de arquitetura em névoa, onde se encontra a camada de borda que por sua vez é onde se dá nosso estudo.

Na camada de borda, assim como em outras áreas de estudo, há exigência de que a comunicação seja segura. Seguiremos o capítulo apresentando alguns conceitos necessários ao entendimento estabelecimento de comunicação segura

A seguir, abordaremos de que forma estudos no âmbito da camada física se mostram promissores ao estabelecimento de um canal de comunicação totalmente seguro.

Uma vez que segurança provida pela camada de física ainda não é uma realidade, passaremos a apresentar de que forma os dispositivos de borda vêm se comunicando na atualidade. Dentro deste contexto, que se dá na camada de aplicação do modelo OSI, iremos apresentar como a atualização do principal protocolo de segurança utilizado pode impactar os dispositivos de borda.

3.2 Arquitetura IoT no âmbito de sistemas distribuídos

O conceito de IoT apresenta um cenário onde bilhões de dispositivos, capazes de realizar sensoriamento e atuar com o meio, se comunicam através da Internet para realização de tais funções. Em [21] os autores apresentam computação em névoa como uma arquitetura composta por três níveis hierárquicos onde tais dispositivos são classificados como "dispositivos de borda", pertencentes à camada 1 (Figura 3.2). Em [22], encontramos uma definição sobre a camada 1 e os elementos que a compõem: a camada mais baixa, onde se encontram os dispositivos de borda, bem como os dispositivos de interconexão de redes (*gateways*) responsáveis pela comunicação com as demais camadas. Ao longo deste trabalho, ao nos referirmos a dispositivos de borda, ou dispositivos IoT, estaremos fazendo menção aos sensores IoT existentes na camada 1.

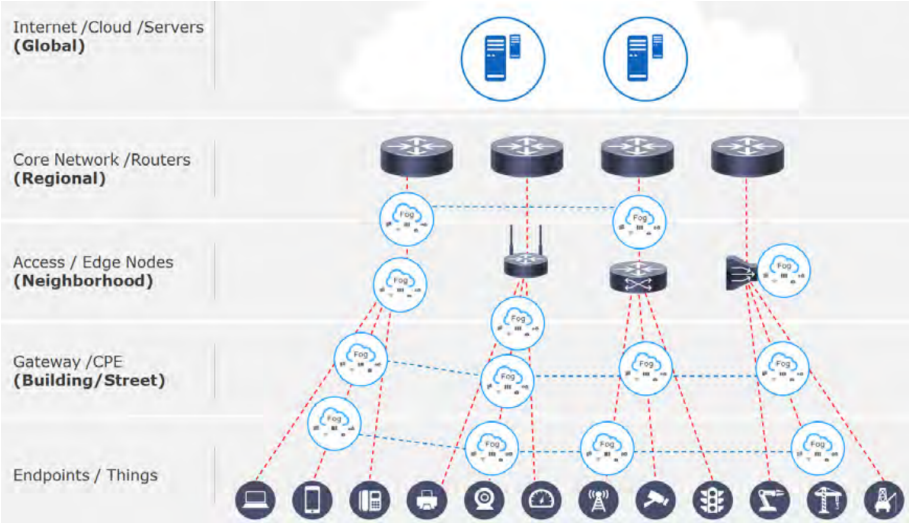


Figura 3.1: Exemplo de Distribuição ao longo da arquitetura em núvem. Extraído de [23].

A computação em névoa, segundo [23], é uma arquitetura que distribui funções de computação, armazenamento, controle e rede mais próximas aos usuários ao longo de um caminho continuum entre a nuvem [24] e dispositivos. A computação em névoa é uma extensão do modelo de computação baseado em nuvem tradicional, onde as implementações da arquitetura podem residir em várias camadas da topologia de rede. No entanto, todos os benefícios da nuvem devem ser preservados com essas extensões para a névoa, incluindo a containerização, virtualização, orquestração,

gerenciabilidade e eficiência. Em muitos casos, a computação em névoa trabalha com a nuvem, como ilustrado na figura 3.1.

Implantações em névoa são realizadas em grande escala e em pequena escala, dependendo do cenário que está sendo abordado. O número de camadas em uma implantação na núvem será ditado pelos requisitos do cenário, incluindo:

- Quantidade e tipo de trabalho necessário por cada camada.
- Capacidades dos nós em cada camada.
- Número de sensores.
- Latência entre nós e latência entre sensores e atuação.
- Confiabilidade/disponibilidade dos nós.

Os nós da núvem podem estar interligados para formar uma malha que forneça balanceamento de carga, resiliência, tolerância a falhas, compartilhamento de dados e minimização da comunicação com a nuvem. Em termos de arquitetura distribuída, isso requer que os nós tenham a capacidade de se comunicar lateralmente e verticalmente dentro da hierarquia da nuvem. O nó também deve ser capaz de descobrir, confiar e utilizar os serviços de outro nó a fim de prover confiabilidade, disponibilidade e capacidade de manutenção de serviço.

Os níveis hierárquicos (camadas) são descritos em [21] como sendo:

- **Camada 1 - Borda:** Nível mais abaixo na arquitetura em névoa. Composta por uma grande quantidade de dispositivos responsáveis pelo sensoriamento e atuação com o meio. Gateways para comunicação com a Internet também pertencem a essa camada.
- **Camada 2 - Névoa:** Nível intermediário, chamado de camada de computação em névoa. Composta por dispositivos que possuam inteligência e poder de armazenamento superiores as da camada 1. Capaz de determinar a necessidade de atuação em tempo real e acionar dispositivos da camada de borda.
- **Camada 3 - Nuvem:** Última camada, com maior poder computacional. Composta por servidores e data centers com capacidade de atender e produzir dados com base nas informações provenientes da camada 2.

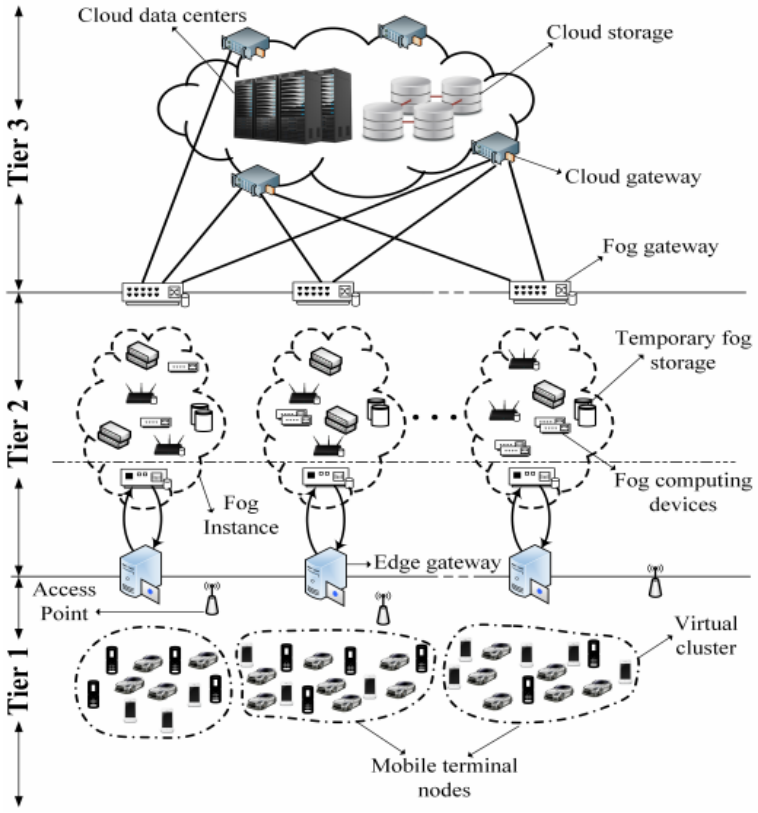


Figura 3.2: Arquitetura proposta para computação em névoa. Extraído de [21]

BONOMI *et al.* [25] apresentam casos de uso da camada em névoa, descrevendo sua integração com as camadas da borda e nuvem. Dentre os casos, os autores apresentam o de um parque eólico onde se encontram diversas turbinas, cujo gerenciamento depende da velocidade do vento incidente sobre cada região onde encontram-se os aerogeradores. Na presença de vento em velocidade excessiva tais equipamentos podem sofrer avarias (ex: sobrecarga elétrica). Abaixo de um certo limite, a energia produzida não compensa sua ativação.

A figura 3.3 ilustra a aplicação da arquitetura em névoa em função da latência e informações exigidas por subsistemas hipotéticos. No caso do campo eólico, dados gerados a partir de dispositivos IoT produzem grande quantidade de informações em tempo real. Dentre as informações, encontra-se a velocidade do vento, utilizada pelo subsistema de proteção para determinação da necessidade de desligamento ou religamento dos aerogeradores (atuação com o meio). Além de tais funções, tal subsistema realiza uma consolidação dos dados (ex: velocidade máxima e mínima em determinado intervalo de tempo) e as transfere, com intervalos da ordem de segundos ou minutos, ao subsistema de dados históricos. Análises que tenham como

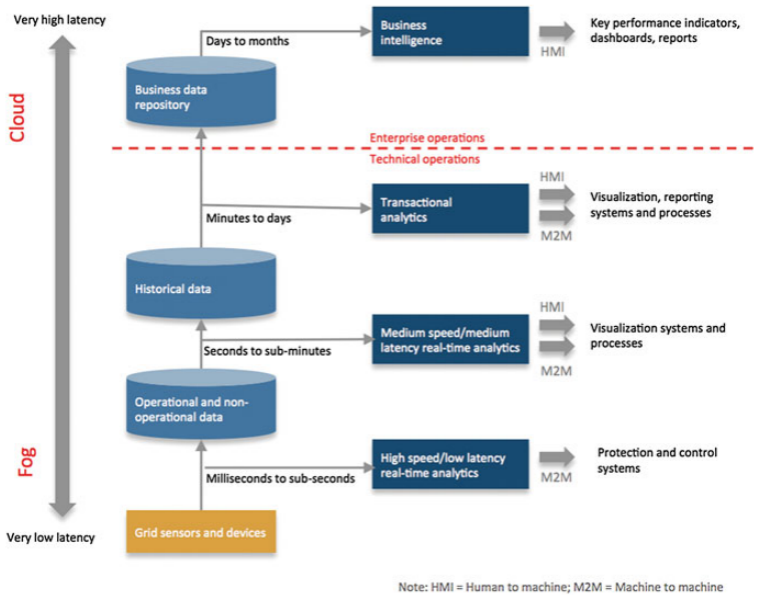


Figura 3.3: Arquitetura em névoa: Exemplo de subsistemas, latências aceitas e condensação da informação. Extraído de [25]

características a exigência de alto poder computacional e a possibilidade de geração em intervalo de tempo muito superior (dias) são realizadas pela camada da nuvem. Previsões de desgaste dos equipamentos e métricas de performance seriam realizadas na camada de nuvem.

3.3 Comunicação na camada de borda do modelo em nuvem

Na camada de borda encontram-se dispositivos com distintas tecnologias de comunicação cada qual adequada a certos tipos de aplicação. A comunicação sem fio é a predominante no âmbito de IoT por razões de flexibilidade, adaptabilidade e mobilidade [9] e por esse motivo daremos enfoque a tal meio.

O modelo OSI e o protocolo TCP/IP, estudados em TANENBAUM e WETHERALL [26] e KUROSE e ROSS [27], são amplamente encontrados na literatura referente ao estudo de redes de computadores. Na tabela 3.1 ilustramos seus respectivos modelos conceituais juntamente com a terminologia que faremos referência durante a tese.

Também destacamos que, no âmbito de IoT, o protocolo TCP/IP está presenta na lista de Frameworks atualmente desenvolvidos e utilizados, assim como é o pro-

tocolo suportado pelos principais padrões de conectividade, em epecial quando é exigida comunicação segura fim a fim, desde a aplicação existente no sensor até o seu destino [28].

Modelo OSI	**TCP/IP**
Aplicação	Aplicação
Apresentação	
Sessão	
Transporte	Transporte
Rede	Internet
Enlace	Host/Rede/MAC
Física	

Tabela 3.1: Modelos OSI e TCP/IP

3.3.1 Camada física no contexto de comunicação sem fio

A determinação da tecnologia sem fio a ser utilizada envolve o entendimento das características que as distinguem, bem como dos domínios de aplicações que se propõem a atender. Um mapeamento de aplicações em domínios, ou classes a que pertencem, é apresentado em [29, 30]. Em [29], os autores apresentam aplicações pertencentes ao domínio de Cidades Inteligentes, Saúde, Casas Inteligentes e Sistemas Inteligentes de Transporte. Similarmente, [30], os autores apresentam 12 domínios, dos quais alguns podem ser mapeados para os mesmos de [29]. Como exemplo, [29] identifica que aplicações de Casas Inteligentes, para monitoramento da qualidade do ar, toleram atrasos de 5 minutos, são atualizadas a cada 30 minutos e transmitem pouca informação. Já, em [30], os autores classificam, no domínio Casas Inteligentes, que cobertura possui relevância média enquanto que taxa de transmissão, custo e baixo consumo possuem alta relevância. Independente do nome dado a cada domínio, os trabalhos acima apresentam características a serem observadas a cada tecnologia de comunicação aplicável. As principais características interrelacionadas e identificadas nesses trabalhos como sendo determinantes são: consumo energético, cobertura, taxa de transmissão e custo, as quais apresentaremos brevemente a relação a seguir.

Consumo energético, cobertura e taxas

Em [30], os autores apresentam, no âmbito de IoT, as tecnologias sem fio predominantes em 2020 em função de seus respectivos alcances. Tais tecnologias se referem às camadas físicas e de enlace do modelo OSI. Abaixo um resumo das tecnologias exibidas na figura 3.4:

- **Proximidade** Dados transmitidos mediante aproximação. Destina-se a aplicações que necessitem a identificação de presença física de um dispositivo de borda. A comunicação se dá diretamente entre dois nós. Para eventual comunicação com a Internet, outra tecnologia é utilizada em conjunto.
- **WPAN - Rede pessoal sem fio** Destina-se a redes com as principais características: baixa taxa de transmissão e curta distância.
- **WLAN - Rede de área local sem fio**. Alta taxa de transferência e alcance limitado a centenas de metros. Aplicada a redes locais. Utiliza-se em grande parte do protocolo WiFi (802.11).
- **WNAN - Redes de área de vizinhança sem fio**. Extensão das redes WLAN no que diz respeito ao alcance. Alta taxa de transferência limitada e conexão direta com a Internet.
- **WWAN - Rede de longa distância sem fio**

No âmbito da camada física do modelo OSI, apresentamos estudo realizado para que haja entendimento acerca de sua influência no consumo energético. Em [31], os autores fazem um estudo sobre tecnologias de redes sem fio, onde as 4 primeiras encontram-se presentes na figura 3.4: Bluetooth (IEEE 802.15.1), ZigBee (802.15.4), Wi-Fi (802.11a/b/g), GSM/GPRS (850-900 DCS PCS), Wi-Max (802.16), UWB (802.15.3). Nele os autores apresentam referencial teórico aplicado à comparação destas tecnologias segundo as seguintes métricas: tamanho da rede (número de nós), tempo de transmissão, potência de transmissão e cobertura, consumo energético, taxa de erro de bit (BER) e eficiência de codificação de dados. Tais métricas se baseiam em parâmetros fixos que cada tecnologia possui, tais como: frequência de operação, taxa máxima de transmissão, alcance máximo, potência nominal e

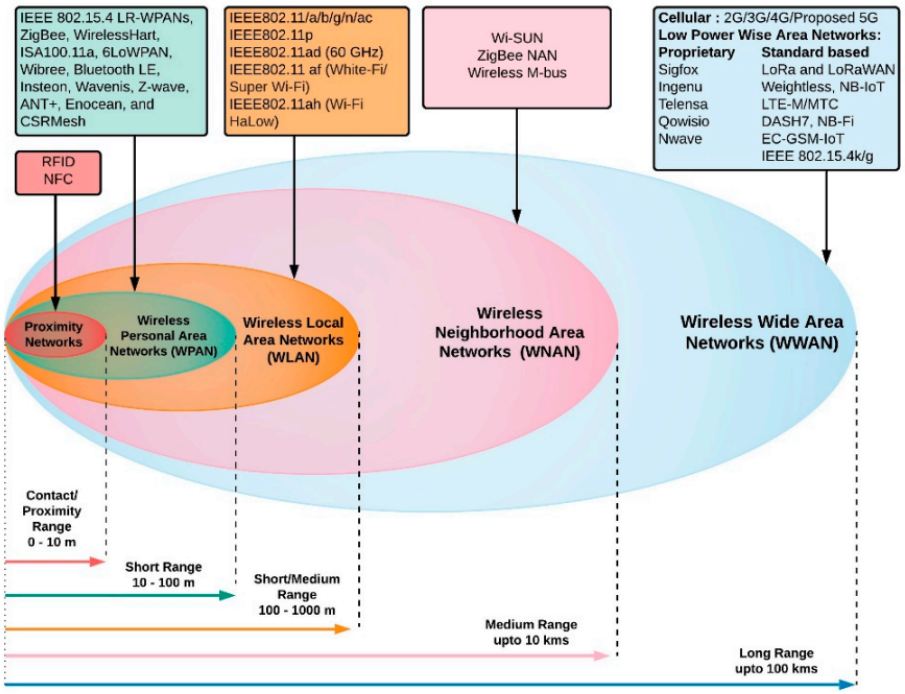


Figura 3.4: Tecnologias sem fio comumente aplicadas a IoT. Classificação em relação à distância. Extraído de [30]

Tecnologia	**Taxa de transmissão**	**Distância**	**Consumo (mW) ***	**Consumo Normalizado (mJ/Mb) ***
Bluetooth	720Kb/s	10m	50	100
UWB	110Mb/s	102m	750	10
Zigbee	250Kb/s	1000m	40	300
Wi-FI	54Mb/s	100m	700	10
Wi-Max	35-70 Mb/s	49km	1020	10
GSM/GPRS	168Kb/s	35km	1010	6300

* De Transmissão

Tabela 3.2: Proximidade

tipo de modulação. Um levantamento de 16 parâmetros é apresentado para cada tecnologia avaliada, sendo dois são mapeados diretamente para os anteriormente descritos como necessários em [29, 30]: cobertura e taxa de transmissão. Os mesmos são apresentados na tabela 3.2.

O estudo [31] prossegue com a realização de uma avaliação experimental de um *chip* de cada tecnologia. Dois dos resultados apresentados no referido estudo são apresentados nas duas últimas colunas da tabela 3.2. Verifica-se que embora o consumo energético das tecnologias Wi-Max, GSM/GPRS, Wi-FI seja consideravelmente superior quando comparado a Bluetooh e ZigBee, o mesmo não ocorre em

relação ao consumo normalizado (mJ/Megabits). Neste caso, observa-se que Wi-Fi, UWB e Wi-Max possuem melhor eficiência energética visto que se propõem a operar com maiores taxas de transmissão e a maiores distâncias. Já GSM/GPRS possui a pior relação, haja vista que sua taxa de transmissão é muito pequena mesmo tendo uma cobertura superior. Verifica-se que o consumo energético possui forte relação com a taxa de transmissão e distância entre receptor e transmissor. Embora a escolha da tecnologia de comunicação envolva diversos outros critérios da aplicação, deixamos como referência [31, 32], para melhor compreensão da relação entre os parâmetros físicos de cada tecnologia e seu potencial reflexo no consumo energético. Uma vez que seja definida a tecnologia mais adequada à aplicação, seguiremos apresentando o estabelecimento de comunicação segura.

Efeitos da interferência na avaliação de aplicações

Nesta tese, a fim de avaliarmos o processamento local realizado por sensores IoT quando em comunicação, torna-se necessário considerarmos a interferência existente no meio sem fio e seus reflexos nas medições obtidas. A seguir apresentamos pesquisas relacionadas à interferência existente em tal meio, visando contextualizar o tema e remeter o leitor a um aprofundamento sobre o assunto.

Em [33], é apresentado um estudo no contexto de extração de conhecimento em redes sem fio e em larga escala. Neste trabalho os autores discorrem a respeito de interferências às quais as redes sem fio estão sujeitas e a seus reflexos no desempenho da comunicação obtida. A fim de exemplificar as fontes de interferência, os autores apresentam os equipamentos envolvidos na comunicação sem fio e indicam que os pontos de acesso que constroem a infraestrutura de rede geralmente são fixos e conectados à estrutura de rede com fio. A mobilidade de usuários é apontada com uma causa para a flutuação no nível de ruído ao qual os pontos de acesso estão sujeitos visto que interfere no ambiente de rádio, atuando como barreiras à propagação de micro-ondas. Além disso, o estudo aponta como motivos de interferência, em redes em larga escala, o fato de novos pontos de acesso serem ligados e desligados a qualquer momento; o fato de alguns pontos de acesso serem móveis devido à proliferação da Internet celular por meio do compartilhamento Wi-Fi; e à mudanças constantes no ambiente, por exemplo, pela abertura e fechamento de portas. Destacam

também que a degradação, resultante de interferências, acaba por ocasionar quedas de pacotes, retransmissões e instabilidades de link que por consequência ocasionam comportamento inconsitente por parte dos protocolos utilizados.

Em [34] os autores avaliam os efeitos da interferência, em abiente sem fio, aplicado à comunicação de sensores. Os autores quantificam o percentual de recebimento de pacotes, através de experimento onde um transmissor envia 2000 mensagens a um receptor localizado a 12 pés de distância. Uma fonte de interferência é posicionada a distâncias de 15, 65, 115 e 170 pés em relação ao receptor. Mede-se então o percentual de recebimento, quando a fonte de interferência opera em redes Wifi 802.11b e 802.11g. Para as referidas distâncias, os percentuais de recebimento encontrados foram de aproximadamente 2%, 8%, 60% e 65% para a rede 802.11b, e de aproximadamente 17%, 50%, 100% e 100% para a rede 802.11g. Em relação ao atraso introduzido, os autores indicaram que obtiveram aumentos de 13% (802.11g) e de 40% (802.11b) na latência.

No estudo realizado em [35], os autores investigaram o efeito da interferência para propor um protocolo de controle de topologia de múltiplos saltos e múltiplos canais para redes de sensores sem fio. Para isso, levam em consideração a interferência causada por redes Wi-Fi em operação na vizinhança. Eles observaram que a maioria dos protocolos propostos não se comporta como previsto quando submetidos a ambientes de rádio reais. Para ilustrar essa interferência, eles realizaram experimentos com oito pares de nós, cada um sintonizado em uma frequência separada. Os transmissores foram colocados em uma linha a uma distância de 1,5m de seus respectivos receptores, a 5 metros do chão. Para os nós que também estavam sob a influência de interferência Wi-Fi, eles apresentam que a taxa de sucesso na recepção de pacotes variou, a depender da configuração, potência e canal, de 99% 75%. Eles mostraram que uma das principais causas de mau desempenho é a interferência, que resulta em perda de pacotes, retransmissões, instabilidade de links e comportamento inconsistente do protocolo.

Utilizaremos o termo "interferência no estado da rede" para se referir aos efeitos que as constantes mudanças no canal sem fio causam nos resultados das medições. Isso inclui mudanças no atraso estimado, perda de pacotes e retransmissões que venham a ocorrer em qualquer repetição de determinado experimento.

Ao longo deste trabalho, quantificaremos a significância da interferência nas avaliações realizadas nesta tese como forma de apresentar a relevância de se utilizar uma metodologia que produza resultados comparáveis e confiáveis. Na seção trabalhos relacionados, apresentaremos como as atuais metodologias de avaliação experimental, que envolvam comunicação em rede, assumem condições não realistas e desconsideram a infuência da interferência em seus resultados.

3.4 Comunicação segura

O objetivo desta Seção é o de apresentarmos revisão referente à comunicação de informações de forma segura. Iniciaremos com uma revisão dos conceitos que utilizaremos, em especial os relacionados à criptografia. Embora a presente tese se proponha a atuar nas camadas de transporte e aplicação do modelo TCP/IP, apresentaremos, após os conceitos iniciais, revisão referente à obtenção de segurança mediante implementação nas camadas inferiores do modelo OSI. O objetivo de realizarmos estudo nestas camadas vem do desejo de verificarmos possíveis soluções que retirem das camadas superiores a responsabilidade de realizar computação referente à transmissão segura. Neste sentido, o que se verifica é que as camadas superiores muitas vezes implementam mecanismos de comunicação segura sem se aproveitar de eventual segurança já implementada nas camadas inferiores e, por este motivo, realizaremos uma revisão também nas camadas inferiores.

3.4.1 Conceitos de criptografia

Comunicação segura envolve, dentre outros conceitos, os de confidencialidade, integridade, autenticidade, autenticação e não repúdio. Aqui apresentaremos alguns conceitos necessários com base nas definições apresentadas em MENEZES *et al.* [36].

Confidencialidade tem como objetivo garantir que a informação original sendo transmitida só será vista por quem for autorizado. A integridade dos dados visa garantir que a informação recebida não foi alterada e a autenticidade tem como objetivo permitir a verificação da integridade e da autoria da mensagem. Autenticação tem por objetivo a identificação de que a outra parte é realmente aquela a que se deseja estabelecer comunicação. Não repúdio refere-se à técnica empregada como

forma de que o autor de determinada mensagem não possa negar sua autoria.

Técnicas criptográficas são comumente utilizadas no contexto de comunicação segura. Tais técnicas se valem do conceito de dureza computacional: o desconhecimento de técnica capaz de reverter, em tempo hábil, a criptografia aplicada.

Abaixo encontram-se definições relacionadas ao estudo de criptografia no contexto em que as iremos utilizar ao longo deste trabalho.

- **Criptografia:** Estudo de técnicas matemáticas relacionadas a aspectos da segurança da informação tais como: confidencialidade, integridade da informação, identificação e autenticidade.

  Nesse trabalho se refere ao emprego de tais técnicas por parte de dois participantes que desejam se comunicar através de uma rede sem fio, passível da presença de agentes interceptadores que não devem conseguir ter acesso a mensagem original transmitida.
  Ao utilizarmos "cifrar" nos referimos ao uso de algum algoritmo de criptografia aplicado à informação a ser transmitida de forma que esta só possa ser recuperada ("decifrada") mediante o emprego do mesmo algoritmo.
  Consideraremos a comunicação entre as duas partes como sendo confidencial quando aplicada criptografia, entretanto, destacamos a existência de estudos relacionados a técnicas capazes de quebrar a confidencialidade da comunicação criptografada, como o apresentado em [37, 38]. Igualmente destacamos a existência de estudos relacionados à extração de chaves criptográficas gravadas em dispositivos. Uma vez que no contexto de IoT nem sempre é possível se controlar o acesso físico a estes, técnicas que envolvam alterações no hardware podem ser empregadas [37, 39], a fim de se obter acesso a tais chaves.

  As técnicas criptográficas que abordaremos se dividem em dois grupos [36]: simétrica e assimétrica, apresentadas adiante.

- **Dureza Computacional:** Esforço computacional necessário a fim de que se decifre mensagem criptografada, sem a posse da chave criptográfica utilizada para sua cifragem. As soluções computacionais empregadas no estabelecimento de comunicação segura muitas vezes se utilizam de ambos os grupos citados (criptografia simétrica e assimétrica) a fim de proporcionar a dureza compu-

tacional estabelecida como sendo suficiente para garantia de confidencialidade [40].

- **Funções Hash:** São funções computacionalmente eficientes que mapeiam dados de tamanho arbitrário para valores binários de tamanho fixo [36]. Uma das propriedades de tais funções é a resistência a colisão: baixíssima probabilidade de que, mediante entradas distintas, obtenha-se o mesmo resultado da função.

- **Criptografia simétrica:** Refere-se ao uso de mensagem criptografada gerada a partir de uma chave conhecida por ambas as partes. O transmissor A envia ao receptor B mensagem criptografada usando chave k e B só consegue descriptografá-la se possuir a chave k. A certeza de que a mensagem só pode ser lida por quem tem a chave é o que garante a confidencialidade do envio. Uma vez que A envie a mensagem criptografada torna-se necessário determinar se B é um receptor legítimo. O fato de não saber se o receptor é legítimo não traz prejuízo à confidencialidade do envio, uma vez que se considera que um falso receptor não conseguirá descriptografá-la sem a chave.

  Um dos problemas relacionados ao uso de criptografia simétrica é a distribuição segura desta chave [36], portanto exige-se o emprego de algum outro método seguro de distribuição.

  Em relação ao esforço computacional exigido, o uso de criptografia simétrica é menor que o associado ao uso de criptografia assimétrica (apresentada a seguir). Em um cenário onde se garanta que só as partes autorizadas possuem a chave, o emprego de algoritmos criptográficos simétricos adequados [40] é considerado suficiente, a fim de se garantir confidencialidade.

- **Criptografia assimétrica:** Refere-se ao uso de criptografia baseada em duas chaves, uma pública distribuída a qualquer um e outra privada que deve ser mantida somente com uma das partes. Como propriedade o algoritmo assimétrico garante que uma mensagem cifrada com a chave pública só pode ser decifrada pelo possuidor da chave privada e que uma mensagem assinada com a chave privada pode ser decifrada por qualquer um que tenha a chave pública.

  Duas aplicações comuns deste tipo de criptografia são: a encriptação com chave pública e a assinatura digital.

Na encriptação com chave pública, somente o receptor possui a chave privada. O transmissor usa a chave pública para cifragem com a certeza de que só o receptor conseguirá descifrá-la, garantindo-se assim a confidencialidade desta transmissão.

Na assinatura digital a mensagem a ser recebida não é cifrada. Ela é transmitida, de forma não cifrada, juntamente com dados extras, chamados de assinatura. Quem transmite a mensagem também gera um código *hash* desta e o cifra com sua chave privada, gerando assim a assinatura anexada a mensagem e enviada. O receptor conseguirá decifrar a assinatura recebida com a chave pública e obterá o *hash*, tendo como garantia o fato de que somente o possuidor da chave privada pode tê-lo gerado. O receptor, a fim de validar a autoria, realiza a computação do *hash* da mensagem recebida e a comparação com o *hash* anteriormente decifrado da assinatura. A mensagem só é considerada válida se o resultado da comparação indicar que são iguais. Com tal mecanismo o receptor tem a garantia de que a mensagem não foi alterada (do contrário o *hash* calculado para a mensagem não seria igual ao *hash* decifrado) e de que foi gerada pelo transmissor legítimo (o *hash* cifrado só pode ter sido computado pelo detentor da chave privada). Neste caso também observa-se que o esforço computacional relacionado a criptografia só ocorre em relação à cifragem e decifragem do *hash* e não em relação à mensagem inteira.

O esforço computacional decorrente do uso de criptografia assimétrica é superior quando comparado ao uso de criptografia simétrica. Como exemplo, o estudo [41] aponta que o uso de algoritmo assimétrico baseado em curvas elípticas é de 100 a 1000 vezes mais lento que o algorítimo simétrico AES [42], quando utilizado em certo dispositivo, o que demandou, segundo o referido trabalho, um aumento do consumo energético com a mesma ordem de magnitude.

Uma motivação à adoção do uso de criptografia assimétrica se dá pelo fato de que somente uma das partes deverá guardar seguramente a chave privada. Em um cenário onde diversos dispositivos de borda enviem mensagem a um mesmo agente, somente este deverá implementar a segurança necessária a fim de que a chave privada permaneça segura.

- **Segurança futura perfeita (PFS - Perfect Forward Secrecy):** Refere-se à garantia de que o comprometimento de determinada chave mestra (ou chave de longa duração) não comprometerá a comunicação passada. Se o protocolo adotado garantir segurança futura perfeita e um adversário obtiver acesso a uma chave de cifragem utilizada ele não conseguirá, a partir desta chave obtida, decifrar outras mensagens cifradas. Uma forma comumente empregada afim de se garantir PFS envolve a utilização de chaves aleatórias a cada sessão. Um eventual comprometimento futuro de uma chave de sessão não permite ao adversário descifrar a comunicação referente a outras sessões.

- **Desafio (Challenge-response):** Refere-se à técnica empregada para verificação de que a outra parte possui determinada chave. Abaixo exemplificaremos através do uso de números gerados aleatoriamente, entretanto, a depender da aplicação, outra informação pode ser utilizada como, por exemplo, um valor referente à data e hora do sistema como forma de prevenção de repetição de mensagens [36].

  No caso de criptografia simétrica, ambas as partes conhecem a chave k, entretanto o envio da mesma garantiria a outra parte (ainda não identificada) e a possíveis interceptadores seu conhecimento. A, para provar a B que possui a chave, recebe de B um número aleatório $[r]$ e então criptografa tal número usando k. Como resultado A envia $[E_k(r_b)]$ para B. Os dados transmitidos são apresentados entre colchetes.

$$A \leftarrow B \qquad [r_b] \qquad (1)$$

$$A \rightarrow B \qquad [E_k(r_b)] \qquad (2)$$

  Ao descifrar o recebido em (2), usando a chave k, B obtém r'. A identidade de A é comprovada por B se $r' = r_b$. A identificação de B para A se dá de maneira análoga.

  No caso de criptografia assimétrica, A conhece a chave pública de B. Para reconhecer B como legítimo receptor, A gera um número aleatório r_a e o envia

cifrado a B usando a chave pública.

$$A \rightarrow B \qquad [E_{pub_b}(r_a)] \qquad (1)$$

$$A \leftarrow B \qquad r\text{'} \qquad (2)$$

Ao receber r', A o compara com o r_a que enviou de forma cifrada. A identidade de B é comprovada por A se $r\text{'} = r_a$ uma vez que somente B conseguira decifrar r_a e devolver o r' correto.

Apresentados os conceitos necessários, seguiremos utilizando-os.

3.4.2 Segurança provida nas camadas física e de enlace

A comunicação sem fio, predominante na camada de borda [9], demanda o emprego de mecanismos que garantam a identidade dos dispositivos envolvidos e que impeçam um agente não autorizado de participar em tal troca de informações. Como exemplos de participação indesejada temos: interceptação de informações, introdução de mensagens e interrupção da troca de informações.

Em [43], os autores apresentam um estudo no âmbito das camadas física, enlace, rede, transporte e aplicação do modelo OSI. Para cada camada são apresentados os principais tipos de ataques e exemplos de suas aplicações a protocolos comumente utilizados por essas. Os autores classificam os requisitos de segurança aplicados a comunicação sem fio como sendo relacionados a: Autenticação de mensagens, Identificação, Confidencialidade, Integridade e Disponibilidade. Aqui não trataremos do último.

Em um dos primeiros levantamentos dos desafios a serem endereçados pela rede 6G para que esta venha a substituir a recém chegada 5G [44], os autores defendem que possivelmente o mecanismo mais forte de segurança será provido pela camada física. Os autores identificam que isso será possível mediante o emprego de técnicas de análise do canal (para que este garanta confidencialidade na distribuição de chaves para duas partes) e autenticação de mensagens provida por alguma estratégia que permita a camada física realizar a extração de assinatura única. Iniciaremos apresentando possíveis estratégias nesse contexto.

Em [45, 46], os autores apresentam uma proposta para estabelecimento de confidencialidade no âmbito da cama física (PLS - Physical layer security) e de estudos relacionados à teoria da informação. Mediante análise do estado do canal de comunicação determina-se a presença de um intruso a fim de que não se proceda a troca de informação. A aplicação de tais abordagens são limitadas no contexto de comunicação sem fio, conforme apresentado em [43]. Entrada e saída de nós, movimentações dos mesmos e mudanças na qualidade do canal em tal ambiente ($fading\ effect$) demandam uma constante reavaliação do mesmo para que se garanta confidencialidade durante toda transmissão. Em cenários com muitas mensagens essa constante reavaliação estaria associada a um maior consumo energético e a introdução de atrasos.

Já, em [47], os autores estudam a obtenção de confidencialidade mediante a utilização das mesmas técnicas de análise de canal acima citadas (WYNER [45], LEUNG-YAN-CHEONG e HELLMAN [46]), para fins de troca de chave criptográfica SKG (*Secret key extraction*). Uma vez que ambas as partes troquem seguramente a chave, o restante da comunicação passa a se dar de forma cifrada não exigindo constante reavaliação do canal.

O emprego de soluções em camadas superiores à física para realização de autenticação é comum. Como exemplo, após a entrada de senha em uma determinada aplicação, o endereço de enlace do dispositivo pode identificá-lo como autorizado na rede, entretanto, tal abordagem poderia ser comprometida caso um dispositivo clonasse tal endereço [43].

No estudo apresentado anteriormente [47], onde a troca de chaves simétricas ocorreu através da análise do canal de transmissão, observa-se que a camada física promoveu confidencialidade antes da identificação. Neste cenário, mediante o emprego de chaves criptográficas salvas nos dispositivos, pode-se realizar um desafio a fim de se verificar se B é realmente o equipamento da organização a que pertence. O mesmo pode ocorrer por parte de B, a fim de identificar se A é um dispositivo autorizado.

Abordagens que propõem alternativa à identificação mediante a presença de chaves criptográficas armazenadas nos dispositivos de borda são apresentadas em [44, 48, 49]. Tais trabalhos têm como motivação o fato de que tais dispositivos estão

sujeitos a ataques físicos com o propósito de extração da chave criptográfica responsável por prover identificação e/ou confidencialidade. No caso de o dispositivo atacado utilizar criptografia assimétrica, a obtenção de sua chave privada comprometeria a identificação do mesmo. Já no caso de utilização de chave simétrica, ambas as partes estariam comprometidas. Outra motivação desses estudos é propor identificação sem o emprego de criptografia assimétrica, visto que esta demanda maior poder computacional dos dispositivos o que acaba por exigir maior consumo energético, conforme anteriormente apresentado.

O mecanismo proposto anteriormente em [44, 48, 49], baseia-se no conceito de funções físicas não clonáveis (PUF) [50]: funções encapsuladas no hardware do dispositivo de borda capazes de responder a desafios mediante uma avaliação que, embora rápida, é de difícil caracterização. Tais funções são apresentadas como capazes de prover identificação única similar a impressões digitais humanas [49] e utilizam-se da premissa de que o circuito de cada dispositivo é único devido a variações do processo de produção do silício.

Os autores de [48] apresentam uma revisão sobre a aplicação de PUF à identificação de dispositivos da camada de borda. Cada um destes dispositivos, antes de se autenticar, deve passar por um primeiro estágio de inscrição, onde recebe um identificador único e lhe são apresentados diversos desafios onde se computa cada resposta em seu PUF. Esse conjunto de informações é, então, armazenado em um agente autenticador. A fim de realizar a verificação da identidade do dispositivo, o agente autenticador seleciona um desses desafios armazenados, envia para o dispositivo e compara sua resposta. A fim de evitar interceptações e reuso dessas chaves, as mesmas são descartadas a cada uso. Por fim, os autores apresentam os desafios à utilização de tal mecanismo, dentre os quais, destaca-se a produção de resultados distintos a um mesmo desafio em decorrência degradação gradual dos transistores CMOS.

Em [49], os autores apresentam uma implementação para identificação sem necessidade de hardware específico, aplicável a qualquer dispositivo que possua relógio oscilador e conversor analógico digital, como é o caso de grande parte dos dispositivos de borda. A implementação proposta também se baseia no conceito de PUF e na etapa de inscrição de cada dispositivo em um sistema autenticador. Os autores

aplicam a solução apresentada a um total de 50 dispositivos de borda distribuídos entre três modelos de distintos fabricantes e microcontroladores. Os dispositivos foram submetidos a variações de 15 a 45 graus Celsius e a variações na voltagem de alimentação (2,7 a 3,3V). Dentre as 500 mil autenticações realizadas ao longo de 1 mês, o experimento obteve 100% de acurácia. Os autores discorrem sobre o impacto do envelhecimento aplicado ao uso do modelo de PUF proposto e indicam que somente após alguns anos de uso haverá necessidade de nova coleta de respostas a desafios.

Embora mecanismos de identificação e confidencialidade providos pela camada de rede se mostrem promissores, sua ampla utilização ainda precisa ser aprofundada e não se verifica sua atual adoção nas tecnologias sem fio apresentadas anteriormente. Procederemos, então, a discussão das atuais técnicas utilizadas para obtenção de comunicação segura dos dispositivos realizadas na camada de enlace.

3.4.3 Segurança provida na camada de enlace e de rede

Em relação ao estabelecimento de comunicação segura entre o dispositivo e o nó seguinte, verifica-se que tal segurança já é implementada, na camada de enlace, pelos diversos protocolos de comunicação sem fio utilizados como aqueles que apresentamos anteriormente 3.4. Em [31], os autores apresentam como os métodos os protocolos Bluetooth, UWB, Zigbee, WiFi, WiMax e GSM implementam tal segurança. Verifica-se o estabelecimento de comunicação segura mediante o emprego de criptografia, em grande parte com base no algoritmo AES. Como exemplo, as redes WiFi [51] especificam padrões de segurança para que os dispositivos estabeleçam conexão com seus respectivos roteadores. A autenticação dos dispositivos pode de dar mediante senha compartilhada ou autenticação realizada de forma centralizada em algum servidor responsável por esta validação. Em relação ao uso de senhas compartilhadas, verifica-se, em [52, 53], a existência de métodos com capacidade de obtenção da senha, bem como capazes de causar degradação da qualidade da rede mesmo em configurações que utilizem o mais recente protocolo WPA2. Já em relação ao uso de autenticação mediante o emprego do protocolo WPA2-enterprise, os autores de [53] identificam que seus testes de penetração apontaram que, quando em combinação com o uso de certificados, mostrou-se impossível a obtenção de acesso

a tais redes sem o respectivo certificado. Os autores então apontam o acesso físico aos dispositivos para fins de obtenção de tais certificados como sendo o maior risco a tais redes. A integridade é implementada mediante o emprego de função hash com base em campos quadros como o endereço MAC. O receptor verifica o hash recebido, chamado de MIC (Message Integrity Code), como forma de verificação da integridade.

Ao utilizarmos um canal inseguro de comunicação, como é o caso da Internet, uma das formas possíveis de se obter um canal de comunicação seguro até o destino se dá, no âmbito da camada de rede, mediante o estabelecimento de redes virtuais privadas [54]. Uma vez que a comunicação entre os dispositivos de borda e o nó de saída (roteador) seja seguro, tal canal poderia ser utilizado retirando-se do dispositivo o processamento necessário à criação deste canal.

Embora o estabelecimento de uma rede privada seja possível em certos cenários, em outros, como os em que os dispositivos de borda se utilizam diretamente das redes metropolitanas implementadas por empresas de telefonia, as camadas de enlace e rede costumam ser providas e administradas somente por tais empresas. O que se verifica é que em tais cenários as redes utilizadas não proporcionam a seus usuários algum tipo de suporte ao estabelecimento de redes privadas. Para estes casos os dispositivos acabam por implementar segurança em suas respectivas aplicações. Trazendo tal cenário para o âmbito da camada de borda, torna-se necessária a busca por soluções onde se possa garantir segurança e se retirar o processamento relativo a comunicação segura fim a fim dos dispositivos IoT.

Propostas que incorporem a descentralização do gerenciamento do tráfego de comunicação em redes metropolitanas 5G são apresentadas em [55, 56]. Nestes apresenta-se modelo chamado de "fatiamento de serviços de rede" proposto a redes 5G. O modelo se baseia no conceito de fatia: conjunto de funções de rede implementadas virtualmente. Tais fatias então executariam, de forma isolada, em dispositivos da rede 5G pertencentes ao fornecedor do serviço. Em relação à direta comunicação dos dispositivos com as antenas, o modelo então permitiria uma transferência de dados, de forma segura, somente à fatia a que se destinam ficando o prestador do serviço limitado a informações da quantidade de dados trafegada para que possa proceder eventual comercialização do serviço. Os administradores responsáveis pela configu-

ração de cada fatia seriam os contratantes do serviço. Uma das funções virtuais de rede possíveis seria o estabelecimento de redes privadas. Os autores prosseguem então apresentando os desafios de tal modelo dentre os quais destacamos: realização do isolamento do processamento de cada fatia, distribuição e sincronização das funções virtuais nas diversas células (antenas), e bem como infraestrutura necessária para obtenção de desempenho (qualidade do serviço) satisfatória em cada célula.

3.4.4 Segurança provida na camada de aplicação

Nessa Seção, abordaremos comunicação segura no âmbito da camada de aplicação do modelo TCP/IP (ou entre a camada de aplicação e a de transporte). Neste âmbito, o protocolo estudado foi o TLS, conforme apresentaremos.

Utilização do TLS

TLS, segundo definido em [57], é um protocolo, cujo principal objetivo é prover privacidade e integridade de dados entre duas aplicações que se comunicam. TLS independe da aplicação que fará seu uso, mas utiliza-se de algum protocolo que garanta estabelecimento de conexão e entrega da informação (ex: TCP). Em sua variação, DTLS [58], não há garantia de ordenação ou entrega da informação. Sua implementação se baseia na existência de algum protocolo sem garantia de entrega (Ex: IP) ficando a cargo da aplicação determinar eventuais perdas ou não ordenamento das informações recebidas. TLS e DTLS, segundo o modelo OSI, encontram-se acima da camada de rede e abaixo da camada de aplicação.

Em [59], são apresentados alguns dos principais protocolos, pertencentes à camada de aplicação do modelo OSI, utilizados por dispositivos da camada de borda para realização de troca de mensagens: MQTT, CoAP, AMQP e HTTP. No referido trabalho são apresentadas as principais características de cada um destes protocolos e apresentam-se cenários aos quais se indica o emprego de cada um. Em relação à troca de mensagens com segurança, verifica-se em [59] que todos os protocolos dão suporte à utilização de TLS como forma de estabelecimento de conexão segura.

A relevância no emprego do protocolo TLS na camada de apresentação também se verifica em [60] e [61]. No primeiro, os autores realizam uma análise comparativa de 3 dos principais provedores de serviços na nuvem que ofereçam suporte a MQTT.

Verifica-se que todos exigem o emprego de TLS na identificação dos dispositivos da camada de borda. No segundo, apresenta-se o percentual de utilização de TLS em conjunto com o protocolo HTTP, com base em dados de navegação gerados por usuários que compartilham suas estatísticas. Verifica-se neste que, em 01/11/2020, mais de 80% dos acessos HTTP se deram de maneira segura, utilizando-se do protocolo TLS.

Em [62], é apresentado um estudo sobre a última versão do protocolo TLS, 1.3, lançada em novembro de 2018 [63]. O estudo se dá a partir de conexões a sites que utilizam TLS, realizadas até novembro de 2019. Verificou-se que 27% dos sites analisados davam suporte à última versão. Os autores realizam um agrupamento dos sites em quinze categorias, onde a que apresenta maior percentual (47%) é a de sites relacionados a blogs, enquanto, que a de menor percentual é a de sites governamentais (3%). A diferença nos respectivos percentuais deu-se quase que na totalidade em função da existência de suporte até a versão 1.2 (menos de 1% só suportava versões inferiores a esta). Verifica-se, portanto, que até a data do estudo, cerca de 70% ainda utilizava a versão 1.2.

No âmbito dos sensores da camada de borda não identificamos estudo recente que apresente informações representativas acerca da adoção da versão 1.3, entretanto, esta tende a ser mais tardia visto que implica na atualização de um universo muito distinto de bibliotecas, sistemas operacionais e firmwares. Uma eventual obrigatoriedade de adoção da nova versão tende a acelerar tal atualização. Verifica-se em [64] que o NIST, órgão regulador responsável pelos padrões de troca de informações dos Estados Unidos, determinou que até o primeiro dia de 2024 todos os servidores e clientes que utilizem TLS, no âmbito governamental ou que a este se comuniquem, devem suportar e dar preferência à versão 1.3 embora ainda possam suportar a 1.2.

Características do TLS

No presente trabalho abordaremos aspectos referentes às versões 1.2 e 1.3 do protocolo TLS. A primeira, por ser a mais utilizada e por ser a versão existente no ambiente que avaliaremos. A segunda, por ser a mais atual e poder, futuramente, ser avaliada com a metodologia que apresentaremos. Quando necessário mencionaremos explicitamente a versão caso alguma informação só se aplique à mesma.

Em relação aos conceitos iniciais apresentados, TLS 1.3 [63] especifica o objetivo do protocolo como sendo: criar um canal seguro entre duas partes que se comunicam. O canal seguro estabelecido atende às seguintes propriedades:

- **Autenticação:** O lado do servidor é sempre autenticado e o cliente é opcionalmente autenticado. Autenticação pode ocorrer via o uso de criptografia assimétrica ou simétrica com chave compartilhada.
- **Confidencialidade:** Os dados trocados só são visíveis às duas partes envolvidas (cliente/servidor)
- **Integridade:** Os dados trocados no canal estabelecido não podem ser modificados por um eventual atacante sem que seja detectada sua mudança.

Em relação à autenticação, não estaremos interessados neste trabalho nos métodos baseados em chave compartilhada, somente no uso de criptografia assimétrica através de certificados digitais.

O protocolo TLS é composto de duas camadas (denominadas protocolos segundo sua especificação): o protocolo Handshake (TLS Handshake Protocol) e o protocolo de Registro (TLS Record Protocol).

O protocolo Handhsake é responsável pela autenticação das partes, pela negociação dos algoritmos criptográficos que serão utilizados e pelo estabelecimento de uma chave privada utilizada pelo protocolo de registro. Uma vez que o Handshake seja realizado com sucesso e sejam utilizados padrões de criptografia considerados computacionalmente seguros para os padrões atuais, as seguintes propriedades são ditas como garantidas: (i) uma ou ambas as partes foram autenticadas; (ii) a negociação da chave simétrica de sessão (efêmera) se deu de forma segura (nenhum interceptador conseguiu obtê-la) e (iii) estabelecimento de canal seguro.

O protocolo de registro (*Record*) é utilizado para troca de dados após o estabelecimento de canal seguro. Este garante que a conexão é privada, mediante o emprego de criptografia simétrica, e de confiança: a integridade é checada mediante o emprego de algoritmos específicos como funções *hash* [36]).

Observa-se que o Handshake, sem o emprego de chaves simétricas compartilhadas, faz uso de técnicas criptográficas mais seguras (assimétrica), enquanto, que a comunicação dos dados, realizada pelo protocolo de registro, utiliza criptografia

simétrica, menos custosa computacionalmente. O primeiro é empregado somente como forma de permitir a utilização do segundo. A informação acerca de quais algoritmos serão utilizados dá-se o nome de conjunto ou *Suite* criptográfico. Cada conjunto do TLS 1.2 especifica: (i) algoritmo usado para troca de chaves, (ii) algoritmo de autenticação, (iii) algoritmo de criptografia dos dados (simétrica), (iv) algoritmo hash para verificação dos dados trocados. Os dois primeiros são usados na fase de Handshake e os últimos dois durante as subsequentes trocas de informações usando o protocolo de registro.

Como exemplo, se durante o Handshake da versão TLS 1.2 for selecionado o conjunto *TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA*, o Handshake procederá com troca de chaves assimétricas usando ECDHE, autenticação usando ECDSA e, após o Handshake, os protocolos de registro usarão AES 256 CBC para cifragem e SHA para verificação de integridade. Um outro exemplo de conjunto TLS 1.2 é *TLS_RSA_WITH_AES_256_CCM_8*, onde neste caso RSA é usado tanto para a troca de chaves assimétricas quanto para a autenticação.

TLS 1.2 data de 2008 e foi elaborado tendo como premissa a compatibilidade com diversos conjuntos criptográficos anteriores. Em sua especificação [57], verificam-se cerca de 30 conjuntos, sendo que outros foram sendo adicionados à versão 1.2 [65]. Incluindo-se versões antigas de conjuntos, TLS 1.2 chegou a ter mais de 300 opções.

Troca de chaves e segurança futura perfeita PFS

Troca de chaves refere-se aos procedimentos realizados durante o Handshake com o objetivo de que ao final as partes obtenham, de forma segura, uma chave criptográfica que será usada para realizar criptografia simétrica nos dados a serem dados trocados.

Em relação à troca de chaves, muitos dos conjuntos criptográficos suportados pela versão TLS 1.2 não garantem segurança futura perfeita (PFS). Como exemplo, durante uma troca de chaves baseada em RSA, a chave simétrica (ou de sessão) é gerada pelo cliente, criptografada com a chave pública enviada pelo servidor e transmitida ao mesmo. Como somente o servidor possui a chave privada, mantém-se o sigilo da chave de sessão. Neste mecanismo, um adversário que grave toda a comunicação e no futuro consiga a chave privada poderá obter a chave de sessão

(a partir desta mesma gravação) e assim ter acesso a todas as mensagens trocadas nesta.

Como alternativa, ECDH é um exemplo de algoritmo suportado por TLS 1.2 que provê segurança futura perfeita. Para isso estabelece chaves de sessão sem a utilização das chaves publica/privada do Handshake.

ECDH se baseia no uso de curvas elípticas para fins criptográficos, proposto por Neal Koblitz [66] e Victor S. Miller [67]. Pontos em uma curva elíptica possuem a seguinte propriedade:

$$(a * G) * b = (b * G) * a$$

Tal propriedade da curva permite que se estabeleçam chaves públicas de ambas as partes, (a*G) ou (b*G), que estas sejam trocadas em um canal inseguro e que mesmo assim se estabeleça uma chave segura. Os seguintes passos exemplificam a troca de chaves efêmeras (usadas em uma única sessão) usando ECHD (ECDHE) [66]

1. A e B definem qual curva irão usar. Uma vez acordada a curva, está definido, dentre outros valores, o valor do chamado ponto base G.

2. A gera sua chave privada aleatóriamente e sua chave pública:
   $\{\ Priv_a,\ Pub_a = Priv_a * G\ \}$

3. B faz o mesmo. Seu par de chaves será:
   $\{\ Priv_b,\ Pub_b = Priv_b * G\ \}$

4. A e B trocam suas chaves públicas em canal inseguro.

5. A calcula chaveSessao = Pub_b * $Priv_a$

6. B calcula chaveSessao = Pub_a * $Priv_b$

É de interesse observarmos que a chave efêmera compartilhada é gerada a cada sessão (ou trocada ao longo desta) entretanto não é transmitida. Sem possuir a chave de sessão a dureza computacional do algoritmo garante que não será possível a um adversário recuperar as mensagens originais a partir da gravação das mensagens cifradas. Desta forma garante-se segurança futura perfeita. A chave de sessão também é conhecida como chave compartilhada ou parâmetro Z.

Embora as partes tenham estabelecido uma chave que somente ambas conheçam, existe a necessidade de autenticarmos uma ou ambas as partes. Como exemplo, um cliente ao se conectar ao servidor deseja saber se não está se comunicando com um falso agente.

Outro ponto aqui destacado refere-se à escolha da curva elíptica a ser utilizada na troca de chaves. Cinco possibilidades são adotadas em TLS 1.3. Em [68, 69], os autores apontam a identificação de possíveis vulnerabilidades em 3 destas. Como alternativa a tais curvas, a curva x25519 [70], de domínio público, é adotada pela versão 1.3 e possui 128 bits de segurança, atendendo aos níveis de dureza exigidos até 2030 e anteriormente apresentados.

Autenticação

A autenticação comumente utilizada em TLS se dá com base no procedimento de assinatura digital descrito anteriormente na seção conceitos. Tal procedimento garante que a mensagem recebida não pode ter sido alterada e que foi gerada por quem a assinou.

Um eventual cliente que receba a chave pública de um servidor a recebe juntamente com uma assinatura digital gerada por um terceiro, conhecido como autoridade certificadora. O cliente possui previamente a chave pública deste terceiro, de forma que a utiliza para verificar sua assinatura segundo os procedimentos de verificação de assinatura digital. Fica, portanto, a cargo da autoridade geradora, a emissão da assinatura da chave fornecida pelo servidor.

Uma vez que a chave pública do servidor seja verificada, procede-se ao fluxo para trocas de chaves de sessão se este for o mecanismo utilizado.

Embora uma troca de mensagem exclusiva para realização da validação do certificado possa ser enviada, evita-se a troca de informações meramente para este fim. Como exemplo, TLS 1.2 especifica [65] que se for utilizado ECDHE como mecanismo de troca de chaves e um certificado RSA como autenticação (como no *Suite TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384*), a autenticação se dá segundo os seguintes passos:

1. A chave pública efêmera *EDCH* do servidor será enviada na mensagem chamada de $ServerKeyExchange$. A autenticação se dará mediante o uso do

certificado RSA. O servidor envia o certificado, sua chave pública efêmera e uma assinatura da mensagem $ServerKeyExchange$ gerada com a chave privada do certificado RSA, que só ele tem.

2. O cliente, ao receber a mensagem valida o certificado nela contido. Se válido, utiliza o certificado recebido (chave pública) RSA para avaliar a assinatura também recebida. O resultado da avaliação só será positivo se o servidor for aquele que enviou a mensagem (possuir a chave privada do certificado RSA). Neste caso, após autenticar o servidor, o cliente aceita a chave pública efêmera enviada pelo servidor.

3. O cliente, a fim de concluir o procedimento de troca de chaves de sessão (êfemeras), envia sua chave pública efêmera *EDCH* em mensagem $ClientKeyExchange$, somente se a autenticação do certificado e da avaliação da assinatura se mostraram corretos.

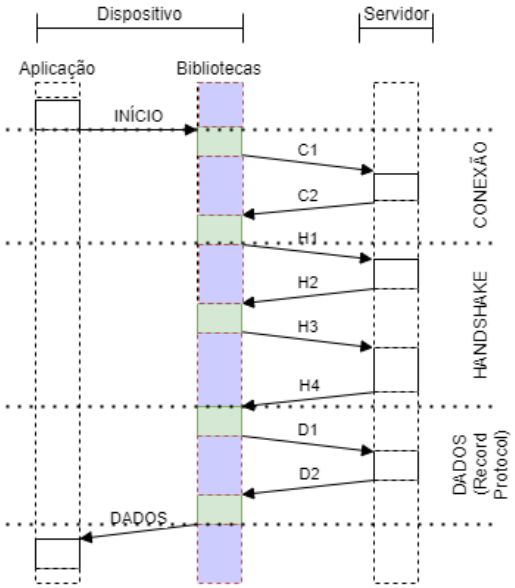
4. Cliente e servidor passam a utilizar a chave de sessão conhecida por ambos.

Importante observar que o mecanismo de autenticação optou por aproveitar-se da necessidade de transferência das chaves públicas efêmeras para também realizar autenticação entretanto os mecanismos de autenticação e de troca de chaves são procedimentos distintos.

Mensagens: Handshake TLS 1.2

Apresentamos na figura 3.5 um exemplo de comunicação baseada no protocolo TLS 1.2. Na figura verifica-se um mapeamento das mensagens TLS e dos possíveis pacotes TCP trocados. A conexão TCP é estabelecida mediante a troca de pacotes C1 e C2 com marcadores TCP SYN e ACK. As mensagens TLS referentes ao Handshake (H1 até H5) geram pacotes TCP contendo dados e que seguem acompanhadas de confirmações ACK. As mensagens H1 até H5 foram diretamente apresentadas como pacotes TCP, entretanto, um mesmo pacote pode incluir mais de uma mensagem TLS. Após o Handshake, são apresentados os pacotes D1 e D2, cada qual contendo mensagem do tipo Registro (TLS Record Protocol). A figura também exemplifica, em verde, processamento útil e ociosidade da CPU, enquanto espera por dados (representada em cor púrpura).

Abaixo descrevemos o conteúdo de interesse ao estudo, contido nas principais mensagens TLS apresentadas na coluna *PAYLOAD TLS 1.2*) da figura 3.5.



PACOTES	TCP	PAYLOAD:TLS 1.2
C1	SYN	
C2	SYN/ACK	
H1	ACK, dados	ClientHello
H2	*ACK, dados	ServerHello,Certificate, ServerKeyExchange, ServerHelloDone
H3	*ACK, dados	ClientKeyExchange, ClientFinished
H4	*ACK, dados	ServerFinished
D1	ACK, dados	Dados cifrados (TLS Record Protocol)
D2	ACK, dados	Dados cifrados (TLS Record Protocol)

* Pacote TCP pode conter mais de uma mensagem TLS ou Payload TLS pode gerar mais de um pacote

Figura 3.5: Pacotes TCP durante comunicação TLS 1.2

- **ClientHello:** Cliente envia, dentre outras informações, quais conjuntos criptográficos ele suporta.

- **Após:** Servidor informa qual conjunto, dentre os propostos inicialmente pelo cliente, ambos irão utilizar.

- **ServerKeyExchange:** A depender do algoritmo de troca de chaves utilizado, essa mensagem deve ser enviada. Ao utilizar ECDHE esta mensagem conterá a chave pública efêmera do servidor e o certificado. A mensagem será assinada [65] com a chave privada do certificado utilizado (EX: RSA ou ECDSA).

- **ClientKeyExchange:** Conterá a chave pública efêmera do cliente. Ao utilizar ECDHE como mecanismo de troca de chaves, a chave simétrica inicial está definida e esta mensagem não conterá nenhuma informação referente à chave simétrica.

  Caso a troca de chaves use RSA (que não prove segurança perfeita), o cliente gera uma chave simétrica aleatória. ClientKeyExchange irá conter a cifra

desta chave, gerada usando-se a chave pública do servidor. Eventual adversário que gravasse essa troca de mensagem poderia decifrar as mensagens, se futuramente conseguisse acesso à chave privada do certificado RSA do servidor.

TLS 1.3: Mudanças em relação a versão anterior

A versão 1.3 [63] aprimora o protocolo TLS tanto em relação ao nível de segurança propiciado, quanto em relação ao número de mensagens trocadas durante o $Handshake$.

Enquanto que na versão anterior eram trocadas 4 mensagens ($2xRTT$), na versão 1.3 são necessárias apenas 2 mensagens ($1xRTT$). O cliente ao enviar a mensagem $ClientHello$ também informa os conjuntos criptográficos suportados, entretanto, ele já gera e envia na mesma mensagem os parâmetros necessários (ex: chave pública) para um ou mais conjuntos propostos. O servidor dá preferencia a utilizar o conjunto já proposto e envia sua chave e certificado na mensagem $ServerHello$, evitando, assim, o envio de mais uma mensagem e sua confirmação.

Em relação ao conjunto criptográfico negociado durante o Hanshake, a versão TLS 1.3 [63], deixa de suportar os conjuntos antigos e passa a adotar somente os padrões criptográficos considerados seguros à época de sua criação. Em relação à troca de chaves, quando não operando com chaves compartilhadas, a nova versão passa a aceitar somente os protocolos que garantam segurança futura perfeita.

O mecanismo de troca de chaves deixa de fazer parte da nomenclatura do conjunto criptográfico e passa a ser identificado como um parâmetro (do tipo $NamedGroup$ [63]) dentro das mensagens trocadas ($ClientHello$ e $ServerHello$). Desta forma, os conjuntos criptográficos na versão 1.3 possuem uma nomenclatura que identifica somente o método de autenticação e o de cifragem de dados utilizado para Registros (TLS Record Protocol).

Em relação à autenticação através de protocolos assimétricos, a versão 1.3 passa a só aceitar seguintes algoritmos: RSA, ECDSA e EdDSA. O mecanismo de autenticação, assim como o de troca de chaves, deixa de fazer parte da nomenclatura do conjunto criptográfico e passa a ser identificado como um parâmetro ($SignatureScheme$). Para realização da autenticação, no $ClientHello$ é especificada uma lista de padrões de certificados aceitos. Cada padrão ($SignatureScheme$)

informa os parâmetros do algoritmo (se houver) e a função hash de validação aceita para o certificado. Ao responder, o servidor envia seu certificado e uma mensagem do tipo $CertificateVerify$. Essa mensagem contém o *SignatureScheme* selecionado da lista proposta e a assinatura digital, criada com base no *SignatureScheme* selecionado. A assinatura é realizada a partir de um valor resultante da concatenação de todas as mensagens enviadas até então durante o handhshake. O hash resultante dessa concatenação é então assinado e enviado.

A cifragem das informações trocadas entre as aplicações se dá, assim como na versão anterior, mediante mensagens TLS de Registro (TLS Record Protocol). Também emprega-se criptografia simétrica, entretanto, só são aceitos algoritmos que empreguem *criptografia autenticada com dados associados - AEAD* [71]. Como exibido na figura 3.6,tal método de cifragem foi escolhido por proporcionar, além de confidencialidade do texto cifrado, a garantia de integridade e autenticidade tanto do texto cifrado quanto também de dados não cifrados enviados juntamente a este. Como exemplo, uma mensagem *AEAD* pode conter um cabeçalho não cifrado juntamente com os dados cifrados, sendo possível se utilizar do cabeçalho para uma pré-validação sem que haja necessidade de decifrar antecipadamente toda a mensagem.

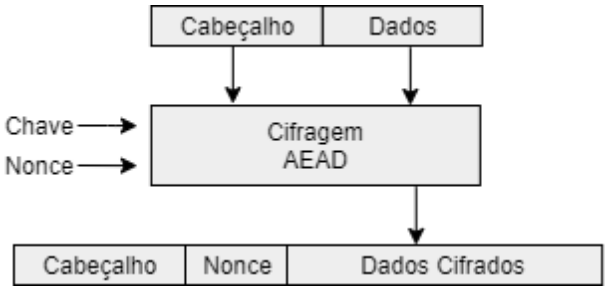


Figura 3.6: Cifragem AEAD

Como mencionado, os métodos de troca de chaves e o de autenticação passaram, na versão 1.3, a ser informados mediante parâmetros específicos, de forma que nesta versão um conjunto refere-se somente à cifra simétrica utilizada nas mensagens TLS do tipo Registro (TLS Record Protocol). Esta versão só possui cinco possibilidades de conjuntos: *TLS_AES_128_GCM_SHA256, TLS_AES_256_GCM_SHA384, TLS_CHACHA20_POLY1305_SHA256 ,TLS_AES_128_CCM_SHA256,TLS_-AES_128_CCM_8_SHA256*. Como exemplo, esta última opção especifica o uso

do algoritmo *AEAD AES_128_CCM_8* [72] e da função *hash* SHA256 [73].

TLS 1.3: Dureza computacional

O TLS emprega três sistemas criptográficos distintos (troca de chaves, autenticação, troca de informações), a fim de que se estabeleça um canal seguro. A escolha de cada um dos três determinará a dureza computacional aplicada a cada etapa.

A versão TLS 1.3 só permite a realização de troca de chaves públicas mediante o emprego de algoritmos que garantam segurança futura perfeita. Os dois grupos de sistemas criptográficos suportados para troca de chaves são: DHE [74] e ECDHE [66]. Para cada um desses dois grupos são aceitos cinco opções de curvas elípticas ECDH ou cinco opções de grupos finitos DHE. A escolha da curva ou do grupo finito determina seus parâmetros, dentre os quais o tamanho da chave.

Em relação aos certificados aceitos para autenticação, a versão TLS 1.3 suporta RSA [75] e os que se utilizam de curvas elípticas (ECDSA [76] e EdDSA [77]), também com distintos tamanhos de chaves. O mesmo ocorre para o protocolo de registro, responsável por garantir a troca de informações de forma privada. Este se utiliza de criptografia simétrica, onde os algoritmos aceitos são: AES_128, AES_-256 e CHACHA20_POLY1305, cada qual com um tamanho de chave distinto.

Verifica-se, portanto, que o uso de criptografia simétrica ou assimétrica se dá por meio de chaves criptográficas de tamanho variável.

Dois algoritmos possuem dureza computacional similar se o tempo necessário para "quebrá-los" (ou determinar a chave utilizada) é aproximadamente o mesmo [78]. A avaliação da dureza de um algoritmo criptográfico em termos do tamanho de chave utilizado tipicamente se dá mediante a avaliação do processamento realizado para avaliação de todas as chaves possíveis. Apresentados textos cifrados e seus correspondentes textos não cifrados, um algoritmo que possua X bits de segurança levaria $2^{X-1}T$ unidades de tempo para descobrir a chave. T é o tempo para cifrar o texto não cifrado e compará-lo ao seu correspondente texto cifrado.

A avaliação da dureza depende de diversos fatores, como a capacidade computacional empregada pelo adversário e o eventual surgimento de novas tecnologias, tal como os computadores quânticos.

Regularmente o NIST, órgão dos Estados Unidos responsável por direcionar seu

Dureza (bits de segurança)	Simétrico	DH ou RSA (tamanho chave)	ECC (tamanho chave)
80		1024	160-233
112		2048	224-255
128	AES-128	3072	256-383
192	AES-192	7680	384-511
256	AES-256	15360	512+

Tabela 3.3: Dureza de algoritmos criptográficos e tamanhos de chave

país no que diz respeito a questões de segurança cibernética, realiza avaliações e apresenta publicações que procuram identificar algoritmos de criptografia e seus respectivos parâmetros tidos como seguros diante dos atuais padrões computacionais existentes. Em [78], do ano de 2020, é apresentada a tabela 3.3 contendo alguns dos algoritmos citados onde se verifica a relação entre tamanho de chave utilizado e bits de segurança (dureza) proporcionada. Também neste documento especifica-se cronograma contendo prazos e durezas que devem ser suportadas. Verifica-se também neste, no que diz respeito a estabelecimento de canal seguro, que até 2030 a dureza mínima aceitável é de 112 bits de segurança e, que a partir de 2030, deverá ser de 128 bits.

A seleção de um algoritmo com maior tamanho de chave implicará em uma maior transmissão de informações entre as partes. Como apresentado na troca de mensagens da figura 3.5, um desses dados é o certificado digital. Uma vez que o certificado possua tamanho maior que o do pacote TCP, mais de um pacote será necessário além de processamento extra decorrente de eventual ordenação ou retransmissão destes.

Como discutiremos durante o trabalho, além de questões relacionadas à transmissão, a escolha do algoritmo de cifragem e do tamanho de chave empregado também influenciará no processamento realizado.

3.5 Conclusões do capítulo

Iniciamos o capítulo apresentando uma arquitetura de sistemas distribuídos que vem sendo amplamente pesquisada, afim de contextualizarmos no âmbito de IoT onde atuaremos. Como tal, identificamos se tratar da camada de borda.

A garantia de comunicação segura é uma exigência comum a quase todos os sistemas computacionais de aplicação real. Apresentamos estudos no sentido de

obtenção de comunicação segura em redes sem fio nas camadas físicas e de enlace. Na existência desta, o processamento referente ao estabelecimento de comunicação segura poderia ser reduzido ou retirado da camada de aplicação. Em não ocorrendo cabe a esta garantir o estabelecimento de canal seguro fim a fim.

Uma vez que ocorra na camada de aplicação, um protocolo amplamente utilizado é o TLS. Este passou por mudanças e sua nova implementação exige maior dureza computacional e consequente atualização dos dispositivos de borda, caso desejem adotá-las. Novas atualizações neste sentido exigirão avaliações, a fim de que se verifique também o seu impacto no processamento e se determinem, dentre outros, seu ponto de operação ideal.

Prosseguiremos no sentido de realizar análise do processamento local relizado por sensores IoT referente à comunicação segura em dois cenários: (i) com o auxílio da aplicação, mediante o uso de TLS com padrões criptográficos suportados em sua última versão, e (ii) sem o auxílio da camada de aplicação, mediante simples estabelecimento de canal de comunicação, considerando que as camadas inferiores proverão a segurança.

Capítulo 4

Arquitetura e Sistemas Operacionais no âmbito de IoT

4.1 Introdução

No capítulo anterior, introduzimos IoT como uma interseção de estudos oriundos de três grandes áreas, dentre as quais, apresentamos a chamada "Visão orientada às Coisas". Estudos orientados a tal visão encontram-se, em parte, relacionados a estudos relacionados à arquitetura de computadores e sistemas operacionais.

Neste capítulo, apresentaremos uma breve revisão sobre a arquitetura computacional indicada no âmbito de estudos de IoT.

Seguimos então revisando conceitos clássicos de sistemas operacionais relacionados ao estudo. Prosseguimos apresentando características que distinguem um sistema operacional em tempo real (RTOS) dos demais e também aquelas a se considerar quando este for do tipo embarcado.

Apresentadas as revisões acerca de arquitetura de computadores e de sistemas operacionais pertinentes, prosseguiremos neste capítulo abordando revisão relacionada a metodologias para obtenção de avaliações de desempenho.

4.2 Arquitetura computacional

4.2.1 Classificação em relação a seus recursos

Dispositivos IoT podem ser classificados como sendo ricos ou limitados a depender dos recursos tecnológicos que dispõem. Em [79], os autores definem dispositivos ricos em recursos como aqueles que dispõem de capacidade de comunicação a distâncias maiores, bem como capacidade de persistir dados.

Em [80], dispositivos ricos em recursos são definidos como sendo os que possuem hardware e software que implementem a pilha de protocolos Protocolo de controle de transferência / protocolo internet (TCP/IP). Os autores também identificam *gateways* específicos e centros de processamento em nuvem como outros atores pertencentes ao ecossistema IoT.

Uma distinção entre dispositivos limitados e dispositivos ricos em recursos é apresentada em [81]. Aqui também são classificados como ricos em recursos os dispositivos que possuem hardware e software que implementem a pilha TCP/IP. Por possuírem tal mecanismo de comunicação diversos protocolos e bibliotecas encontradas na literatura, tais como *Constrained Application Protocol* (CoAP), Representational State Transfer (REST), *Extensible Message and Presence Protocol* (XMPP), Advanced Message Queuing Protocol (AMQP) e *MQ Telemetry Transport* (MQTT), podem ser adotados diretamente por esses dispositivos. Por outro lado, os dispositivos com recursos limitados dependem de outros atores do ecosistema IoT para que possam adotar qualquer um dos padrões acima citados que competem como candidatos entre si na posição de padrão universal a ser adotado na área de IoT.

A expansão dos dispositivos IoT, segundo seu conceito inicial [82], onde futuramente tudo estaria conectado à Internet, certamente é influenciada pela necessidade ou não de novos equipamentos, a fim de que se estabeleça conexão com a Internet. Dispositivos ricos em recursos possuem capacidade de se conectar à Internet sem a necessidade de desenvolvimento e utilização de gateways específicos a IoT.

Inúmeras propostas de novas arquiteturas para a Internet, de forma a se obter mais flexibilidade, escalabilidade e segurança acabam não sendo adotadas por demandarem mudanças na atual infra estrutura. Como consequência, verificamos que os primeiros protocolos da Internet (por exemplo, Domain Name System (DNS),

TCP/IP, Hypertext Transfer Protocol (HTTP)) permanecem sendo utilizados e adaptados, de forma a adquirirem novas características sem a necessidade de uma mudança profunda na atual infraestrutura da rede [83].

Na figura 4.1, extraída de [81], verificamos que a utilização de dispositivos limitados em recursos acaba por demandar equipamentos específicos responsáveis pela comunicação e tradução para determinado padrão adotado.

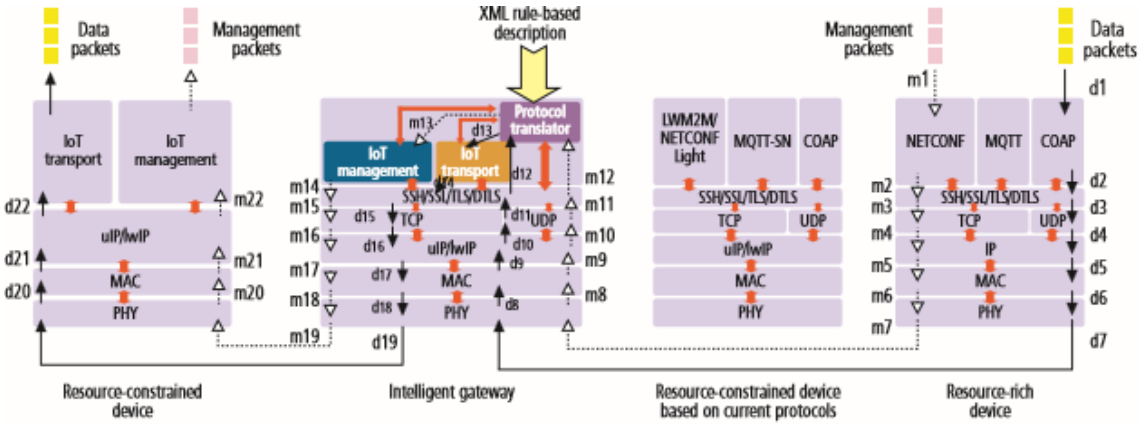


Figura 4.1: Dispositivos IoT ricos ou limitados quanto a seus recursos e à necessidade de gateways a fim de conectá-los na Internet. Extraído de [81]

Seguiremos apresentando de que forma os dispositivos IoT vêm sendo implementados a fim de ser caracterizarem como ricos em recursos.

4.2.2 Sistema em um único chip

Observamos que, por volta de 2015, dispositivos conhecidos como Sistemas em um único circuito integrado (SoC) começaram a agregar importantes características: menor consumo energético, baixo custo, miniaturização e capacidade de comunicação direta com a Internet. Plataformas como Raspberry Pi [84] e Intel Gallileo [85] aparecem como plataformas inicialmente utilizadas em IoT [86], entretanto, agregam diversos outros componentes eletrônicos independentes, a fim de disponibilizar as funcionalidades necessárias. Como consequência, possuem valor mais elevado e consumo energético superior.

Em FLYNN e LUK [1] os autores definem uma arquitetura de Sistemas em Chip (SoC) como um conjunto de processadores, memórias e interconexões dimensionado para determinado domínio aplicação.

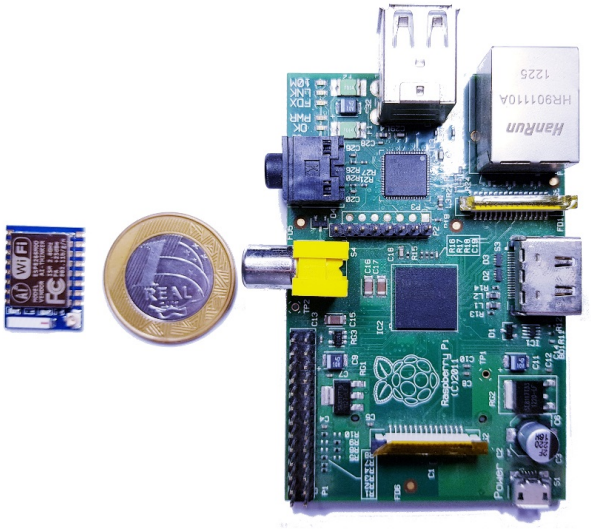


Figura 4.2: Exemplo de miniaturização dos controladores IoT. À esquerda um exemplo de SoC. À direita um exemplo dos até então utilizados. Ao centro uma moeda para demonstração da miniaturização

Como exemplo, um SoC voltado para *smartphones* poderá inclui mais de um processador para o processamento das aplicações e um processador gráfico. Como exemplo de circuitos analógicos e customizados, também poderia incluir conversores analógicos digitais e outros circuitos necessários à leitura de temperatura. Outros componentes de sistema, tais como interfaces de comunicação sem fio ou processadores criptográficos, são exemplos de elementos por vezes incorporados a um SoC.

O que distingue um SoC de um computador de uso geral ou das plataformas de desenvolvimento IoT como como Raspberry Pi é seu dimensionamento voltado à determinada aplicação. A aplicação é previamente conhecida e determina a escolha dos elementos que irão compor o SoC.

Uma decisão fundamental durante o processo de criação de um SoC é se componentes do sistema serão implementados em hardware ou software. Mediante uma análise de que rotinas demandam muito processamento, procura-se implementá-las em hardware, se assim for possível. Tipicamente, observa-se que uma implementação em hardware é muito específica, inflexível a outras utilizações e de maior complexidade em relação à sua construção e testes, quando comparada à implementação em software. Por outro lado, uma boa implementação em hardware é mais rápida e demanda menor consumo energético quando também comparada à sua implementação por software rodando em um processador de uso geral.

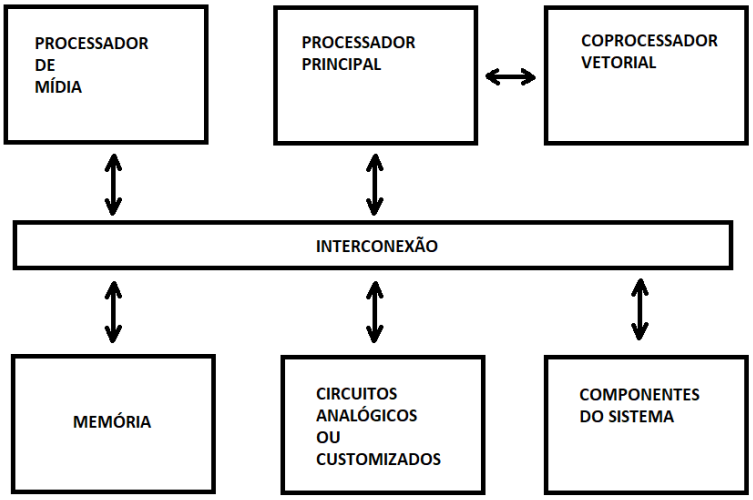


Figura 4.3: Modelo básico de SoC apresentado em [1]

Em [87], verifica-se um estudo relativo à implementação de um processador criptográfico em hardware destinado à computação do algorítimo de curvas elípticas anteriormente apresentado. Verifica-se no trabalho que além das questões de performance, a necessidade de avaliação do espaço físico ocupado torna-se extremamente relevante.

Indicamos [1] como referência aos diversos âmbitos de estudo envolvidos na projeção de sistemas em um único chip.

No âmbito de IoT, verifica-se que a ampla adoção da arquitetura SoC devido à sua miniaturização, baixo custo e consumo. Tendo como características principais o sensoriamento, a atuação e a capacidade de comunicação segura em rede sem fio, verifica-se que um SoC apto a operar em tal âmbito pode ser encontrado com um custo inferior a cinco dólares e um tamanho inferior a $4cm^2$ [88].

4.3 Sistemas operacionais multitarefas

Passamos aqui a abordar estudo relacionado a sistemas operacionais no contexto de IoT. Uma vez que iremos atuar no escalonamento, apresentamos os conceitos relevantes e em grande parte necessários ao entendimento do método de avaliação que iremos propor.

Uma tarefa (ou processo) representa uma abstração de um programa em execução. Um processo contém o código do programa e dados.

Em um sistema de tarefa única apenas um programa é executado por vez. So-

mente após sua conclusão outro programa passa a executar.

Sistemas operacionais multitarefas têm como objetivo maximizar o uso de seus recursos computacionais existentes através da alternância entre processos aptos a executar. Além da maximização do uso da CPU e demais recursos, tais sistemas também proporcionam ao usuário a impressão de que diversas tarefas estão sendo executadas simultaneamente, embora na realidade apenas uma esteja sendo executada, em cada processador existente, em determinado instante.

Sistemas operacionais multitarefas podem ser preemptivos ou colaborativos (não preemptivos). No caso dos colaborativos, uma tarefa tem a garantia de que executará até sua conclusão ou até que voluntariamente ceda sua execução. Caso não o faça, tomará a CPU utilizada toda para si.

No caso de sistemas operacionais preemptivos um processo pode ser retirado de execução mediante critério adotado pelo sistema operacional.

Neste trabalho estaremos interessados em sistemas operacionais multitarefas e preemptivos aos quais iremos nos referir simplesmente como sistema operacional a menos que explicitamente descrito como de outro tipo.

4.3.1 Chamadas do sistema

Um sistema operacional tem como objetivo gerenciar o uso de recursos de maneira eficiente. Para isso provê bibliotecas, contendo funções responsáveis por coordenar o uso de tais recursos.

Processos tipicamente são construídos utilizando-se de bibliotecas que, direta ou indiretamente, se utilizam de bibliotecas fornecidas pelo sistema operacional para a realização de parte de suas rotinas. Como exemplo, citamos operações relacionadas à alocação de memória e relacionadas à entrada e saída de dados: acesso a disco, comunicação serial, comunicação em rede, comunicação entre processos.

Bibliotecas do sistema operacional proporcionam, do ponto de vista de quem as utiliza, abstração de características de hardware e software necessárias não só à correta execução mas também, do ponto de vista do sistema, necessárias à acomodação do processo dentro do sistema operacional.

Suponhamos que um processo necessite acessar o disco. As bibliotecas do sistema operacional, além de realizar a comunicação com o hardware, determinam se o

processo necessitará aguardar até que a operação seja concluída. Do ponto de vista do programador só ocorre uma leitura ou escrita em disco entretanto, do ponto de vista do sistema, determina-se que o processo não continuará sua execução até que o dado esteja disponível. Caso o processo tente acessar diretamente o disco sem conhecimento do sistema operacional (por exemplo através de manipulação direta do dispositivo), este não poderá acomodá-lo de acordo com sua política de seleção de tarefas em execução.

Durante o restante desse trabalho, a menos que destacado o contrário, iremos nos referir a tarefas ou processos como aqueles que se utilizam das bibliotecas que o sistema operacional oferece ao desenvolvedor como forma de coordenar o acesso às estruturas internas do sistema operacional (APIs do sistema). Ou seja, tarefas que, por exemplo, irão recorrer às chamadas do sistema operacional ao realizarem uma operação de entrada e saída. Consequentemente, se o sistema operacional assim determinar, o processo será bloqueado até a conclusão da operação. Neste cenário, tais tarefas não se utilizam de chamadas internas do kernel, àquelas não destinadas ao desenvolvedor final.

Quanto nos referirmos à tarefas implementadas pelo fabricante, ou nativas, estas podem ou não se utilizar de chamadas internas do kernel, chamadas estas cujo uso não deve ser feito por desenvolvedores de aplicações regulares.

4.3.2 Escalonador de processos

Em um sistema multitarefas cabe ao escalonador definir qual processo estará em execução em cada processador disponível.

A alternância entre processos deve garantir que um processo retome sua execução de onde parou. Chamamos de contexto o conjunto de informações necessárias para que um processo retome sua execução. O conjunto de informações necessárias engloba, dentre outros, os valores dos registradores que um programa utilize. Suponha que um processo possua o valor de uma variável em um registrador de uso geral e um valor referente a qual trecho de código está executando (IP/PC *Instruction Pointer* ou *program counter*). Ao serem retiradas de execução tais informações devem ser armazenadas para que, ao retornarem à execução, possam ser novamente atribuídas a seus respectivos registradores, garantindo assim que a

execução se proceda no mesmo estado em que foram interrompidas.

Além das informações para retomada de execução, outras informações acerca de cada processo são necessárias para que se proceda à alternância de processos. Chamamos de PCB, bloco de controle de processo, ao conjunto de informações acerca de cada processo. Além das informações que chamamos de contexto, podem-se encontrar no PCB outras informações como seu estado (ex: em execução ou não), nome e código identificador do processo dentro do sistema.

Em sistemas operacionais multitarefas preemptivos, tipicamente um processo deixará de executar quando ocorre uma interrupção (ex.: temporizador indicando esgotamento da fatia de tempo alocada ao processo), quando termina ou quando chama uma operação que o coloque em espera (ex: operações de entrada e saída, solicitações *wait* ou *sleep*). Ao detectar a necessidade de troca de um processo em execução o escalonador precisa determinar qual será o processo que irá sucedê-lo e proceder à troca de contexto, que, resumidamente, consiste em salvar o estado atual do processo em execução (ou exclusão do mesmo da lista de processos aptos a executar) e reestabelecimento do estado que o novo processo a executar detinha antes de ser interrompido.

Critério de Seleção de tarefa a ser executada

O escalonador é a rotina do sistema operacional que tem como função principal implementar a política de seleção da próxima tarefa a ser executada.

A depender da aplicação do sistema operacional, o algoritmo de seleção de tarefas deve atender a critérios, tais como: maximização do número de processos atendidos por intervalo de tempo, maximização do percentual de utilização da CPU, minimização do tempo de espera. Dessa forma diversos algoritmos de escalonamento são propostos na literatura. Abaixo apresentamos um resumos de algoritmos comumente utilizados.

- **First Come, First Served ($FCFS$):** Processos são atendidos por ordem de chegada. Quando um processo se torna apto à execução, ele entra no fim de uma fila de processos. O escalonador seleciona o primeiro da fila quando necessita escolher o próximo processo a executar.

- **Por Prioridade (*Priority Scheduling*):** Processos são atendidos segundo uma prioridade atribuída a cada um. O sistema seleciona, dentre os aptos a executar, aquele com mais prioridade. Em caso de empate, outro critério (por exemplo $FCFS$) é adotado.

- **Round-Robin:** Destinado a arquiteturas que disponham de um temporizador. Uma unidade de tempo, chamada de quantum, determina o tempo máximo que um processo pode executar. Quando um processo se torna apto a execução, ele entra no fim de uma fila de processos. O escalonador seleciona o primeiro da fila e limita sua execução a um quantum. Passado esse tempo ele é retirado e retorna ao final da fila.

No caso do $FCFS$, verifica-se sua limitação quanto ao tempo de espera. Independente do tempo que fique em execução todos os processos ocupam um mesmo lugar na fila e não há preempção. Suponhamos que em determinado momento estejam na fila os processos P1 e P2 com tempos de utilização T1=20 e T2=2 respectivamente. P1 já está apto a executar, portanto, P2 irá aguardar T2=20. O tempo médio de espera para atender as duas tarefas seria TME=10. Suponha agora o caso em que P2 preceda P1 na fila. P2 é imediatamente atendido, enquanto que P1 aguarda T2=2. Neste caso, o tempo médio de espera cairia para TME=1. Nesse caso, verificamos que não há um critério que procure equalizar o tempo de espera.

Num escalonamento prioritário deve-se definir o critério a ser utilizado. O sistema pode internamente determinar algum critério mensurável (ex: número de entradas em execução) ou pode utilizar algum critério externo (ex: atribuído diretamente pelo programador). Um problema aqui é que pode ocorrer espera infinita (starvation) caso sempre existam processos de maior prioridade na fila, levando um processo menos prioritário a nunca executar.

Em relação ao momento em que um processo pode ser escalonado, escalonadores prioritários se dividem em dois tipos [89]: orientados a eventos e orientados ao quantum. Os orientados ao quantum realizam escalonamento, dentre outras situações, quando o tempo de execução do processo atinge o quantum estabelecido. Já os orientados a eventos iniciam a rotina de escalonamento de um processo assim que ocorram eventos, tais como chamada à uma operação que o coloque em espera, término de execução ou troca de prioridade. Importante observar que um sistema pode

ser orientado a evento e realizar escalonamento decorrente de evento relacionado ao término de um quantum.

Um ponto crucial no algoritmo Round Robin é a determinação do quantum. Com um quantum muito grande ele se assemelha ao $FCFS$. Já com um quantum muito pequeno teríamos um excesso de trocas de contexto, o que acabaria por destinar demasiada utilização da CPU às rotinas de escalonamento e não à execução de processos.

A combinação de mais de um algoritmo por vezes é utilizada. Um escalonador que utilize Round Robin com prioridades por exemplo.

Em alguns sistemas operacionais é permitido que se determine que um processo só pode executar em determinado processador. Em tais sistemas, o escalonador também deve levar em consideração tal característica durante a elaboração da estratégia de escalonamento adotada.

Processamento associado ao escalonamento

O escalonamento de processos demanda processamento o que acaba por consumir parte da capacidade computacional disponível à execução de processos.

Exemplos de processamento inerentes às funções de escalonamento:

- **Quantum:** Associado ao tratamento da interrupção causada pelo temporizador responsável pela preempção de tarefas.
- **Escolha de processos:** Associado à execução dos algoritmos responsáveis pela escolha do novo processo que irá executar.
- **Troca de contexto:** Associado ao armazenamento e ao reestabelecimento dos contextos referentes ao processo que deixa de executar e ao novo processo que será executado.
- **Alternância de processadores:** Associado à troca de uma tarefa de um processador para outro. O escalonador pode, por exemplo, manter filas separadas por processador e necessitar migra a tarefa de uma fila para outra.

Além do processamento dispendido pelo sistema operacional na realização do escalonamento, a preempção também pode causar menor eficiência em relação ao

hardware utilizado. Ineficiência do uso da memória cache é um exemplo de degradação que ocorre em decorrência do escalonamento [90].

4.3.3 Interrupções

Ao sistema operacional cabe a tarefa de receber notificações assíncronas de agentes externos e realizar o devido processamento. Quando tais sinalizações sejam decorrentes da execução de um programa (ex: indicação de divisão por zero), chamamos de interrupções de software. Uma interface de rede que receba dados ou um disco que complete uma operação de leitura ou escrita exemplificam interrupções de hardware. A fim de identificá-las e tratá-las, hardware e sistema operacional precisam acordar os mecanismos que utilizarão.

Uma forma simples de notificação de hardware pode ser a disponibilização por parte do processador de um registrador que indique a presença de uma notificação. Neste caso o sistema operacional eventualmente consulta tal registrador (*pooling*), faz o devido processamento e altera o valor do registrador para indicar que a notificação foi consumida.

Outro formato de notificação se dá através do uso de interrupções (ou desvio de execução) do processamento atualmente em execução em um processador para uma rotina específica do sistema operacional. Para tal o hardware provê uma lista de "locais" (endereços de memória) relacionados à cada notificação que poderá emitir. O sistema operacional, conhecendo tal lista, implementa as rotinas destinadas a cada notificação possível e coloca o endereço de cada rotina em seu respectivo "local" do hardware. Existindo uma notificação, o hardware transfere a execução de determinado processador para o endereço da rotina acordada. Cabe ao sistema operacional realizar o necessário para que, após a execução da rotina, o processo anteriormente possa retomar a execução de onde parou. O "local" especificado pelo hardware depende do dispositivo utilizado, sendo comum o uso de registradores ou endereços virtuais de memória que internamente são mapeados para o dispositivos e não para a memória RAM ou ROM.

Havendo o suporte do hardware para o uso de interrupções, o sistema operacional deixa de realizar processamento referente a consultas frequentes a registradores, a fim de verificar a existência ou não tais notificações. O processamento envolvido

passa a se dar somente na presença de uma notificação.

BRANDENBURG [89] apresenta classes de interrupções, das quais destacamos as seguintes:

- **De dispositivos (DI):** Destinadas ao tratamento de dispositivos externos, evitando assim a necessidade constante verificação (*pooling*). Como exemplo, temos a indicação de que um botão foi pressionado.
- **De tempo(TI):** Destinadas à execução periódica de rotinas em com suporte por parte do hardware. Um processador pode ter um único relógio associado a um contador de ciclos e oferecer 3 registradores programáveis que causem interrupções quando o contador atingir o valor programado. Tais interrupções podem ser associadas à troca de tarefas por parte do escalonador.
- **Comunicação entre processadores (IPI):** Associado à troca de informações entre processadores. Pode ser utilizada, por exemplo, quando um processador P1 queira informar à P2 que P2 deve trocar de tarefa. Neste caso, uma interrupção é gerada em P2, de forma que P2 passa a executar a rotina destinada a tratar tal interrupção.

4.3.4 Exclusão mútua

A troca dos processos em execução introduz uma nova classe de problemas quando mais de um processo compartilha um mesmo recurso. Suponhamos dois processos, P1 e P2, que leiam um valor no mesmo endereço de memória e realizem seu incremento, de forma a sabermos quantas vezes determinado trecho foi executado. Uma execução de P1 e uma execução de P2 poderiam produzir valor diferente de 2, caso P1 e/ou P2 fosse(m) interrompido(s) entre a leitura e a escrita em memória (figura 4.4).

Em um sistema com um único núcleo tal concorrência por recursos poderia ser evitada mediante a garantia de que não haja interrupção do processo em execução até sua conclusão. Já em sistemas com mais de um núcleo, a total interrupção da

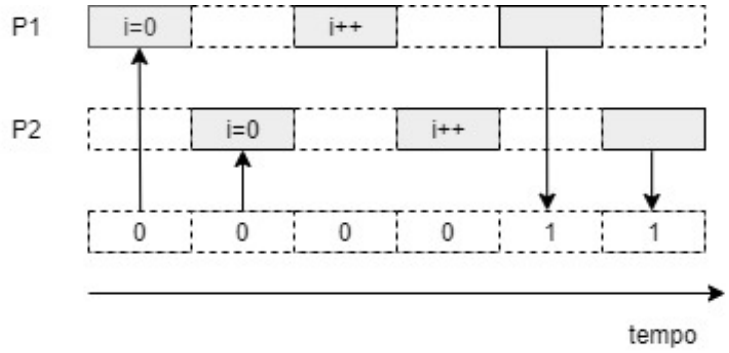


Figura 4.4: Processos P1 e P2 manipulando uma mesma variável sem exclusão mútua

execução de processos em todos os dos demais processadores introduziria demasiada ineficiência em relação ao uso das CPUs.

Diante de cenários como este necessitamos de mecanismo que impeça a manipulação de forma indiscriminada de recursos compartilhados por dois ou mais processos em execução. A trechos de código onde se deseja garantir acesso a um único processo chamamos de **região crítica**.

Dá se o nome de exclusão mútua a técnicas empregadas na garantia de acesso exclusivo a regiões críticas. Mediante seu correto emprego, se um processo estiver na região crítica, os demais serão "excluídos" de executar regiões críticas que também utilizem os mesmos recursos até que o acesso a mesma seja liberado.

Um algoritmo que se proponha a garantia a exclusão mútua deve satisfazer a três propriedades:

Exclusão mútua: Se um processo está na região crítica, nenhum outro processo estará em execução na mesma região.

Progresso: Se um processo desejar entrar na região crítica, eventualmente algum processo entrará.

Ausência de esperas ilimitadas (*starvation*): Se um processo desejar entrar na região crítica, eventualmente ele entrará.

Auxílio do hardware

A exclusão mútua pode ser implementada com o auxílio do hardware ou totalmente por software. Em geral, quando implementadas com o auxílio de instruções de hardware, operações atômicas (indivisíveis) são utilizadas.

No código apresentado em 4.1, verifica-se, em processadores Intel, o uso da instrução atômica XCHG [91]. Inicialmente, atribui-se o valor 0 à variável *lockvar*. Na linha 8, o valor do registrador EAX é alterado para 1. A operação atômica realizada na linha 9 troca o valor de EAX com o da variável *lockvar*. Se dois ou mais processos tentarem realizar a instrução $XCHG$, somente um executará por vez. Com essa construção, o resultado de EAX na linha 10 só será zero a um único processo por vez, cabendo aos demais retornarem a função *Spin_Lock* (linha 11), a fim de repetirem o teste.

Listing 4.1: Exlusão mútua por hardware. ISA Intel

```
1
2 Spin_Lock:
3     CMP lockvar, 0 ;Check if lock is free
4     JE Get_Lock
5     PAUSE ;Short delay
6     JMP Spin_Lock
7 Get_Lock:
8     MOV EAX, 1
9     XCHG EAX, lockvar ;Try to get lock
10    CMP EAX, 0 ;Test if successful
11    JNE Spin_Lock
12 Critical_Section:
13    <critical section code>
14    MOV lockvar, 0
15 ...
16 Continue:
```

Similarmente, temos a instrução atômica S32C1I [92] que, em determinados processadores $Xtensa$, permite que um valor só seja armazenado em memória caso seu conteúdo atual seja igual a um valor fornecido (compara e troca). O retorno de tal operação é o valor que efetivamente permaneceu em memória. Quando o resultado dessa operação é diferente do valor esperado está determinado que outro processo está atuando na região crítica.

Em ambos os exemplos a espera por acesso resultante da necessidade de exclusão mútua pode vir seguida de uma espera ocupada ou de uma preempção a fim de que outros processos passem a utilizar a CPU enquanto se aguarda a liberação da região crítica. No código apresentado em 4.1, o sistema operacional poderia, em substituição as linhas 6 e 14, implementar uma preempção a fim de retomar a execução do processo somente após chamada especifica que indique conclusão da

operação.

O auxílio que o hardware pode oferecer à exclusão mútua proporciona uma melhor eficiência no que diz respeito à realização das operações de busca em memória e comparação de valores, entretanto, a exclusão pode ser totalmente implementada por software, conforme abordamos mais à frente.

Exclusão mútua com preempção

Independente do mecanismo de exclusão implementado ser por software ou com auxílio de hardware, quando um processo não pode acessar determinada região crítica deve-se determinar o que fará enquanto aguarda a liberação de acesso a tal região. Em sistemas operacionais preemptivos a troca do processo em execução surge como alternativa à espera sem execução de processamento útil (espera ocupada). Uma vez que determinada a necessidade de espera, pode-se realizar uma preempção do processo que está aguardando. Uma vez preemptado agora se faz necessário um mecanismo que indique a liberação de acesso à tal recurso.

Apresentamos duas abordagens a implementação de algoritmos de exclusão mútua: através de espera ocupada e com troca de contexto auxiliada pelo sistema operacional.

Exclusão mútua com auxílio do sistema operacional

Sistemas operacionais preemptivos costumam oferecer um conjunto de operações que dê suporte à exclusão mútua. Tal fornecimento é importante não só do ponto de vista de organização e reutilização de códigos comumente utilizados mas também com o objetivo de incorporar a exclusão mútua às rotinas de escalonamento realizadas pelo sistema operacional.

Sem recorrer a bibliotecas do sistema operacional específicas à exclusão mútua, um conjunto de processos pode implementar seu próprio mecanismo e ceder sua execução ao detectar necessidade de espera por liberação de acesso à região crítica. Além da reimplementação de códigos que uma biblioteca poderia prover, ainda haveria necessidade de se determinar quando nova checagem de liberação de acesso se faria necessária. Uma estratégia poderia ser a de liberar o processamento de outras tarefas por um intervalo de tempo (a ser determinado) e, então, proceder nova che-

cagem. Verifica-se, portanto, que, embora a detecção da necessidade de espera seja relativamente trivial, o mesmo não é verdade na determinação do momento em que haverá liberação da região crítica.

Ao delegarmos ao sistema operacional tanto a tarefa de restringir quanto a de liberar o acesso a regiões críticas, este poderá coordenar a necessidade de espera (preempção) e a de liberação para execução em consonância com demais critérios de escalonamento adotados, evitando esperas desnecessárias.

Uma abstração presente em sistemas operacionais preemptivos é o semáforo, um tipo especial de variável, onde se pode aplicar duas operações: incremento e decremento de seu valor. Durante a verificação ou alteração de um semáforo, o sistema operacional não permite que a operação seja interrompida, ou seja, só é realizada de forma atômica. Determinado processo, ao se utilizar de um semáforo, é posto em espera caso o valor deste seja menor que zero. Caso contrário, o semáforo tem seu valor decrementado e o processo continua sua execução dentro da região crítica. Ao concluir sua execução dentro da região crítica, o processo em questão deve incrementar o semáforo para que nesse momento o sistema operacional possa liberar a execução de outro(s) processo(s) que por ventura aguarde(m) tal liberação.

Através de sucessivos incrementos a um semáforo, podemos permitir que o sistema operacional libere diversos processos a execução. Enquanto não for atingido o valor zero, processos que passarem por um semáforo estarão aptos a executar. Em um caso especial conhecido como mutex ou semáforo binário, somente o valor zero pode ser assumido, ou seja, um único processo executa por vez. Por vezes chamadas distintas são fornecidas a semáforos e mutexes.

Importante observar o papel do sistema operacional na implementação de abstrações como semáforo ou *mutex*, uma vez que ele é o responsável por garantir a atomicidade das operações de incremento e decremento, bem como realizar o devido bloqueio ou desbloqueio das tarefas fazendo uso de regiões críticas. Outro aspecto que aqui damos destaque diz respeito a preempção. A menos que citado o contrário, ao nos referirmos a semáforos, assumiremos a realização de preempção do processo em execução em caso de espera.

Exclusão mútua com espera ocupada

A exclusão mútua com espera ocupada pode ser utilizada como uma alternativa ao uso de mecanismos fornecidos pelo sistema operacional ou em casos que o sistema julgue mais eficiente aguardar ao invés de colocar o processo em espera. Suponhamos que seja feita uma chamada ao sistema operacional de leitura de um sensor com um tempo limite de resposta muito curto. O sistema pode determinar ser mais vantajoso realizar a espera ocupada do que realizar a troca de contexto.

O uso de espera ocupada nos será de interesse mais à frente, de forma que vamos aqui abordá-la através do algoritmo *Filter*, uma extensão do algoritmo de Peterson estendido a N processadores. Tal algoritmo, apresentado em 4.2, atende às propriedades necessárias à exclusão mútua citada anteriormente, entretanto, como verificamos na linha 11, não atende ao critério de justiça ($fairness$), uma vez os processos são atendidos com base no processador a que estão alocados (conforme o laço da linha 7) e não com base na ordem de chegada.

Outro problema associado a tal algoritmo advém do fato que os compiladores promovem otimizações (ex: *loop unrolling*) com a intenção de otimizar o processamento envolvido, o que pode alterar a execução realizada. Portanto, sua utilização exige que sejam tomados cuidados extras.

Atendendo-se as considerações apresentadas, o algoritmo nos é de interesse devido à possibilidade de mantermos os processos em execução, ou seja, sem que haja preempção. Através do laço existente na linha 11 e mediante o uso de variáveis voláteis (linhas 3 e 4), podemos instruir o compilador a realizar consulta a tais variáveis a cada iteração, mantendo o processador em tal procedimento até que seu valor se altere. Ainda sem preempção, o laço da linha 11 também nos apresenta uma abordagem onde cada processador se mantém em posição conhecida em relação aos trechos de código que serão executados em seguida. Futuramente, utilizaremos abordagem semelhante, não para garantir exclusão mútua, mas sim para realizar uma execução coordenada de trechos de código em cada processador.

Listing 4.2: Exlusão mútua por software. Algorítimo de Peterson para N processadores

```
1 class PetersonN implements Lock {
2
```

```
3   volatile bool level[N];
4   volatile int  victim[N-1];
5
6   public void acquire(int i) {
7     for (int j = 1; j < N; j++) {
8       level[i]  = j;
9       victim[j] = i;
10      //Espera ocupada
11      while ((exists k != i) level[k] >= j) && victim[j] == i);
12    }
13  public void release(int i) {
14    level[i] = 0;
15  }
16 }
```

4.4 Sistemas operacionais em tempo real - RTOS

Em computação em tempo real a "corretude" de um programa se baseia não apenas em seu resultado lógico, mas no tempo de execução. O tempo de execução é crítico e não apenas um fator de performance. Em [93] o autor apresenta duas classificações para sistemas em tempo real:

Soft real-time: o tempo de execução é crítico, entretanto, atrasos apenas degradam a qualidade da resposta do sistema. Sistemas de transmissão ao vivo e sistema de medição por satélite são exemplos dessa categoria.

Hard real-time: o tempo de execução é absolutamente crítico. Uma operação que não cumpra o tempo especificado é inútil. Como exemplo, temos sistemas de controle de tráfego aéreo e de sistemas de freios automotivos.

Sistemas operacionais de tempo-real não garantem, mas procuram auxiliar para que processos consigam executar atendendo ao tempo limite especificado. Nesse contexto, a latência de interrupção se apresenta como um fator crucial que diz respeito ao quão rápido o sistema é capaz de atender a interrupções. Um alto tempo de atendimento a interrupções atrasará a execução (e o escalonamento) de tarefas degradando sua performance.

4.4.1 Sistemas Operacionais em Tempo Real Embarcados

Sistemas operacionais utilizados em computadores pessoais, tais como, Android, Linux ou iOS se propõem atender um escopo de aplicações de amplo espectro. Como tal, têm uma alta complexidade, alta demanda por memória e não costumam atender a demandas em tempo real. Alguns dispositivos IoT disponibilizam ambiente ou bibliotecas próprias (bare hardware) que tornam a utilização de memória ($footprint$) mínima. Algumas soluções compatíveis a grupos de dispositivos possuem demanda de memória entre 5KB e 50KB (por exemplo, TinyOS [94], Yottos[95]). Outros sistemas operacionais demandam mais, fugindo do escopo de IoT (Ex: Android [96] demandando a partir de 0.5GB).

A implementação de um RTOS em um sistema IoT deve também atender à necessidade de adaptação do sistema para que este possa executar em um ambiente com recursos computacionais mais escassos. Em geral, o sistema mantém apenas funcionalidades extremamente essenciais para seu funcionamento.

Tomemos, como exemplo, a unidade de proteção a memória existente em processadores ARM. Tal unidade, acessível através de conjunto de instruções específicos [97], permite o estabelecimento de restrições de acesso a regiões de memória. Tais restrições permitem a criação de regras de acesso (de leitura e/ou escrita) destinadas a determinado processo ou conjunto de processos. O hardware, em caso de violação, chama função do sistema responsável por tratá-la. Sua utilização pode permitir por exemplo, uma separação entre o espaço de memória do *kernel* e o do usuário. Muitos sistemas embarcados, em favor da redução do processamento envolvido, não implementam nenhuma proteção de espaços de memória mesmo havendo recursos de hardware dando suporte a tal função. Como consequência, cabe ao desenvolvedor tomar os cuidados necessários.

Outro exemplo de minimização da utilização de recursos é a retirada, durante a compilação, de bibliotecas não utilizadas. Abre-se mão da possibilidade de execução de processos ou programas não existentes durante a compilação em favor da produção de um código resultante ($firmware$) de menor tamanho.

Além de procurar minimizar o uso de recursos, sistemas operacionais multitarefas dessa categoria costumam prover um conjunto de mecanismos de comunicação entre tarefas (mensagens) e de acesso exclusivo a memória (mutex, semáforos). Tais me-

canismos por vezes são fornecidos em um mesmo sistema com diferentes chamadas extremamente otimizadas a determinado uso. Como exemplo, o sistema FreeRTOS possui um mecanismo para comunicação entre tarefas, onde somente uma única tarefa pode aguardar mensagem [98]. O desbloqueio dessa tarefa, através desse mecanismo, é 45% mais rápida e usa menos memória do que uma comunicação realizada por meio da implementação de semáforo binário também existente em tal sistema. Além da limitação de envio a uma única tarefa, o emissor da notificação também não poderá ficar bloqueado aguardando o envio. Embora esse tipo de otimização seja muito específico, ele exemplifica a tentativa do sistema operacional em prover o máximo de eficiência do uso de recursos a casos onde seja possível utilizar-se das otimizações oferecidas.

4.5 Metodologias comumente empregadas na avaliação de desempenho

Sistemas em tempo real são compostos por aplicações e pelo sistema operacional. A obtenção de estimativas temporais de desempenho em emprego real deve ser obtida levando-se em consideração não só a execução da aplicação, mas também a interdependência existente entre o sistema operacional, a aplicação e o hardware que está executando estes dois.

Dois tipos de análises de execução são comumente encontradas na literatura: estática e dinâmica. O primeiro, se dá sem a real execução do programa, com base em modelos de hardware. O segundo, mediante real execução do programa e coleta de informações. Em [99], é realizado um estudo sobre tais métodos de avaliação de desempenho, seus desafios e erros comuns. Em relação à análise estática, os autores identificam os principais desafios como pertencentes a quatro grupos. O primeiro, chamado de acurácia baixo nível, diz respeito às dificuldades de obtenção do modelo de hardware associadas à inconsistência nos modelos temporais divulgados pelos fabricantes, dificuldades de modelagem da interação entre componentes de diversos fabricantes e à não divulgação de modelos precisos a fim de evitar acesso, por parte da concorrência, a informações consideradas estratégicas por esses. Outra classe de desafios apresentada diz respeito a sistemas como múltiplos núcleos: modelos

precisos de hardware que levem em consideração contenções ocasionadas pelo acesso a recursos compartilhados. As outras duas classes de desafios apresentadas, em [99], são identificadas como acurácia alto nível e certificação. A primeira, diz respeito às dificuldades na obtenção de uma modelagem da execução do programa a ser analisado. A segunda tem relação com o alto custo de obtenção de uma certificação baseada em uma análise estática.

Em relação à análise dinâmica, destacamos os seguintes desafios dentre, os apresentados pelos autores:

- **Condições de testes:** A introdução de medições pode alterar o funcionamento original se necessitar de instruções custosas.
- **Variáveis de teste**: Dificuldades na avaliação onde o espaço de entradas possíveis for muito grande. A realização de todos os casos pode ser inviável e a não realização de todos pode produzir resultados que não expressam uma real execução.
- **Medição de tempo**: Medir precisamente o número de ciclos tende a ser um problema complexo.
- **Múltiplos núcleos**: Tarefas interagem de maneira não óbvia com diversos hardwares simultaneamente, produzindo resultados com grande variação.

Em [100], os autores apresentam uma proposta para avaliação de sistemas RTOS baseada em avaliações com base em métricas refinadas ou de grão fino (*fine-grained benchmark*). Em tal proposta, os resultados se baseiam em experimentos que mensuram o tempo utilizado na realização de rotinas específicas responsáveis pelos seguintes serviços do sistema operacional: (i)resposta a eventos externos, (ii)comunicação entre tarefas e compartilhamento de recursos, (iii) transferência de dados entre tarefas. Tal proposta de avaliação é utilizada em [2], onde os autores realizam avaliação dinâmica dos principais componentes de um sistema RTOS, com o enfoque na comparação de sistemas RTOS utilizados em dispositivos IoT. Dentre as oito avaliações realizadas, encontram-se : (i) tempo para troca de tarefa, (ii) tempo de obtenção e liberação de semáforos, (iii) tempo de transferência de um semáforo a outra tarefa,(iv) tempo de envio e recebimento de uma mensagem a si mesma, (v) tempo de

obtenção e liberação de bloco de memória. A avaliações são realizadas em uma placa NXP/Freescale FRDM-K64F [**?** , FRDMK64F] em cinco sistemas operacionais.

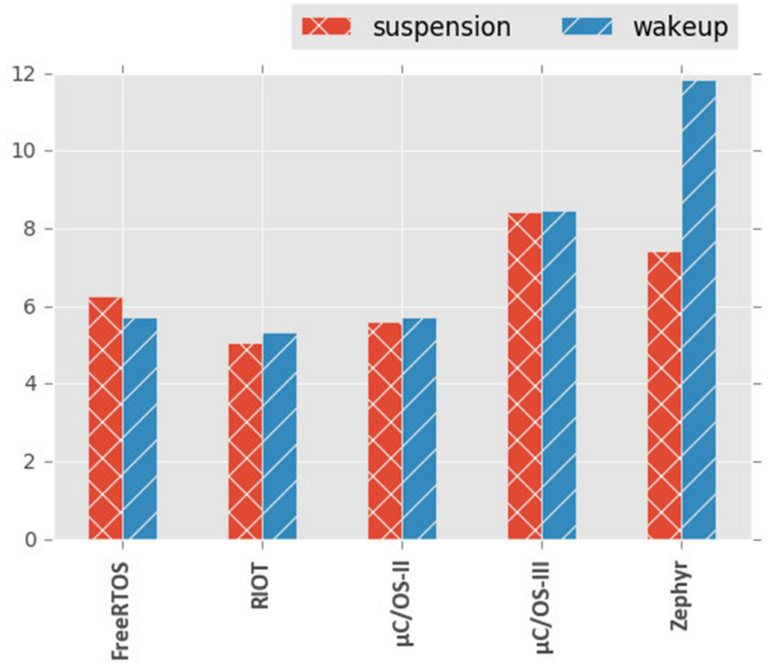


Figura 4.5: Resultado de avaliação realizada e extraída de [2]. O tempo (em microsegundos, no eixo vertical) refere-se ao procedimento para medição referente a troca de tarefa.

Para cada tipo de avaliação os autores descrevem uma série de procedimentos de forma a validar a relevância do experimento e a acurácia da medição. Como exemplo, na figura 4.5, estão os resultados apresentados para o experimento que avalia o tempo para troca de tarefas assim descrito: "A tarefa A tem prioridade maior que a tarefa B. Assim que ela é acordada, ela é posta para dormir e é realizada uma troca de contexto para a tarefa B que então acorda a tarefa A. O tempo para troca entre A e B é medido".

O exemplo acima ilustra que, por mais que uma avaliação deste tipo produza resultados comparáveis, a utilização de tais resultados na avaliação real de determinado programa ou biblioteca exigiria, por exemplo, um mapeamento de quantas trocas de contexto são realizadas. Em um cenário multiprocessado, onde outras tarefas estejam em execução, tal mapeamento se mostra ainda mais desafiador. A utilização destes resultados serviria como uma forma de comparação entre os sistemas avaliados, mas seria de difícil aplicação à avaliação do comportamento de uma aplicação.

Outro método de avaliação de desempenho amplamente empregado se baseia na

medição do tempo de execução de sucessivas repetições de um trecho de código. No entanto, essa abordagem pode não ser suficiente para compreender em mais detalhes o desempenho do sistema, uma vez que não leva em consideração, por exemplo, a interação entre diferentes componentes do sistema e a presença de outros processos em execução. Além disso, fatores externos podem afetar a precisão e a consistência dos resultados, como o estado atual da rede de comunicação. Por fim, essa análise não fornece informações sobre a utilização de recursos compartilhados pelo sistema operacional, o que impede ajustes para melhor utilização dos processadores e outros recursos. Para otimizar o desempenho do sistema, é possível se realizar ajustar diferentes parâmetros como, por exemplo, o quantum do escalonador. Neste caso, a determinação de um valor de quantum adequado à determinada aplicação pode contribuir para a diminuição de tentativas de escalonamento desnecessárias, acarretanto melhor emprego da CPU na realização de computação útil.

4.6 Necessidade de metodologia de avaliação de processamento quando da ocorrência de comunicação de rede

Aqui destacamos artigos que corroboram a necessidade por uma metodologia de avaliação de processamento local de sendores IoT, quando da ocorrência de comunicação de rede. O objetivo é darmos destaque à necessidade da metodologia que apresentaremos nesta tese.

Em [101] os autores realizam estudo e comparação de 18 métodos de avaliação propostos (benchmarks) no âmbito de IoT. Embora a rede desempenhe um papel fundamental no desempenho, apenas dois benchmarks abordam esse aspecto. O primeiro tem como objetivo avaliar o overhead introduzido pela virtualização realizada em servidores de borda. O segundo mede a latência de comunicação de um dispositivo para a nuvem da Amazon. Os autores apontaram que pesquisas adicionais em benchmarks são necessárias para (i) quantificar o desempenho de aceleradores e redes; (ii) identificar limitações nas abordagens atuais de benchmarking em testes reais de dispositivos ricos em recursos (aqueles que possuem as capacidades de

hardware e software para suportar comunicação em rede [81]); e (iii) investigar os efeitos de mudanças, como atualizações de bibliotecas de sistemas operacionais.

Em [102], os autores apresentaram o estudo em relação aos pilares funcionais de IoT e suas aplicações emergentes a fim de motivar acadêmicos e pesquisadores a desenvolverem aplicativos de IoT em tempo real, energeticamente eficientes, escaláveis, confiáveis e seguros. A fim de que se obtenha melhor desempenho os autores identificam que milhões de dispositivos trocam informações usando diferentes padrões de comunicação e que a interoperabilidade entre eles é um problema significativo. Nesse contexto, eles apontaram que: (i) é necessário estudar os protocolos de camada de aplicação da IoT em diferentes ambientes (dentre eles os dispositivos rico em recursos) com diferentes cargas e diversidade de dados; (ii) desenvolver um mecanismo de autenticação forte e leve ainda é um desafio; e (iii) testar e validar o desempenho desses algoritmos incorporando-os aos protocolos de camada de aplicação da IoT.

Outro desafio, cuja metodologia de avaliação contribui para seu entendimento e na busca por soluções, é a energia requerida por esses bilhões de dispositivos [103, 104]. Muitas aplicações seguem um padrão: (i) os dados são adquiridos, (ii) processados e (iii) as informações são enviadas por meio de um canal sem fio. Esse processo se repete, e o ciclo de trabalho é fundamental: quanto menor for, o que pode ser alcançado reduzindo o tempo ativo ou aumentando os períodos de inatividade, menor será a energia média necessária [105].

4.7 Conclusões do capítulo

Iniciamos apresentando o conceito de sistemas ricos em recursos: aqueles que possuem hardware e software capazes de se comunicar com a Internet. Como tal são os vislumbrados nos conceitos iniciais de IoT.

Prosseguimos apresentando sistemas em chip único como sendo a arquitetura que vem tendo destaque no âmbito de IoT devido ao fato de poder atender à demanda por sistemas ricos em recursos com melhor eficiência energética e menores dimensões físicas.

Os conceitos de sistemas operacionais em tempo real embarcados foram então apresentados. Sistemas com tais características se inserem no contexto daqueles

utilizados na arquitetura de computadores baseadas em sistemas de chip único.

Os conceitos relacionados a sistemas operacionais serão utilizados na elaboração do método de avaliação. Iremos implementá-lo no escalonamento, utilizar espera ocupada como forma de contenção de CPU e exclusão mútua em um cenário muito especial, onde não há preempção. Além disso, nos utilizaremos do fato do sistema utilizar escalonamento prioritário e Round Robin para determinamos um formato de obtenção da avaliação que introduza processamento extra mínimo e mensurável. Também avaliaremos a memória utilizada visto que este recurso é limitado na arquitetura em questão,

Após a revisão de sistemas operacionais apresentamos metodologias utilizadas na avaliação de processamento para determinarmos aquela adotada no trabalho: análise dinâmica de ajuste fino. Para tal, torna-se necessário verificarmos os ajustes que nos permitirão classificar o processamento associado a distintos grupos de processamento.

Seguiremos agora no sentido de apresentar o método e os grupos de interesse que serão aplicados.

Capítulo 5

Metodologia proposta para avaliação de processamento local

Neste capítulo, propomos uma metodologia para avaliação de desempenho de sensores IoT, baseada em trações de execução. Tal método foi desenvolvido em um ambiente experimental e utilizado para realizarmos avaliação do processamento local referente à comunicação de dados do dispositivo utilizado.

Prosseguiremos descrevendo o ambiente experimental e seguiremos apresentando o método de avaliação proposto. Durante a apresentação, o ambiente selecionado será utilizado a fim de esclarecermos o método.

Posteriormente, o método será resumido como forma de extrapolarmos sua possível utilização em outros ambientes.

5.1 Metodologia de pesquisa: quantitativa experimental

A metodologia de pesquisa adotada nesta tese é do tipo pesquisa quantitativa experimental. Como tal, apresentaremos ao longo deste capítulo o contexto da pesquisa, a definição das variáveis a serem estudadas, um plano experimental que permita manipular essas variáveis de forma controlada, a coleta de dados quantitativos e a análise estatística dos dados para verificar as hipóteses propostas.

5.2 Contexto da pesquisa

Em um computador pessoal, verifica-se a possibilidade de escolha de componentes e sistemas operacionais a serem utilizados. Já nos atuais dispositivos da camada de borda, verifica-se a adoção de sistemas em um único chip contendo componentes, tais como, conversores analógicos digitais, interfaces de rede e portas lógicas. O ecossistema principal de desenvolvimento destes dispositivos é aquele disponibilizado pelo fabricante e encontra-se em ambiente onde parte do código é fechado, em especial os referentes a *drivers* ou trechos que o fabricante julgue ser estratégico manter somente sob seu conhecimento. Verifica-se, portanto, que, tanto o hardware, como uma parte do sistema operacional são praticamente imutáveis, cabendo ao seu "usuário" utilizar-se somente do disponibilizado pelo fabricante, de forma conjunta.

Diante de um ecossistema com tais restrições, se mostrou de interesse a obtenção de método de avaliação de desempenho que englobe, em conjunto, tanto o hardware quanto o software "mínimo" de código fornecidos.

A fim de que possamos realizar uma análise mais profunda e conclusiva, tornou-se necessário a elaboração de um método que propicie um detalhamento de trechos de execução. Como exemplo, o processamento associado a um sensoriamento pode ser decomposto em processamento associado a (i) funções relacionadas a solicitação de leitura da porta, (ii) sistema operacional para troca de contexto enquanto aguarda leitura, (iii) processamento não útil enquanto aguarda leitura, (iv) retomada do processo após o dado do sensor estar disponível e (v) conclusão do processo. Em uma transmissão, poderia ser decomposta em processamento relativo a (i) solicitação de transmissão, (ii) espera por respostas e (iii) processamento dos dados recebidos.

Prosseguiremos apresentando o Ambiente experimental a fim de detalharmos como tais aspectos se apresentam em um caso concreto.

5.3 Ambiente Experimental

A seleção do ambiente experimental teve como premissa a escolha de um sistema em chip único SoC que incorpore comunicação em rede sem fio, que seja de baixo custo e fácil aquisição, que possua sistema operacional amplamente utilizado pela comunidade e cujo código fonte seja em grande parte aberto. Prosseguiremos apre-

sentando estudos que se deram no mesmo ambiente, como forma de corroborar o seu emprego nesta tese e de familiarizar o leitor acerca do ambiente escolhido.

5.3.1 Sistema em um único chip utilizado

Em [106], os autores apresentam estudo sobre a utilização do SoC ESP32 para desenvolvimento de sistemas voltados para IoT. Os autores identificam que tal ambiente possui todas as premissas desejáveis a um ambiente experimental. Em suas conclusões, estimam que o ESP32 terá um papel de destaque no desenvolvimento de sistemas IoT justamente por atender a tais premissas. Similarmente outros trabalhos como o apresentado em [107], acabam por ter conclusões similares quanto às características ESP32, o que o corrobora a ideia de o utilizarmos em nosso ambiente experimental.

Ainda em relação ao ESP32, uma avaliação de dispositivos IoT é apresentada em [108], mediante a classificação destes em quatro grupos de gerações, sendo a mais recente descrita como a de "microcontroladores indicados ao uso em aplicações IoT". No estudo os autores realizam uma análise de performance de 13 dispositivos de todas as classes. Para avaliações de operações com pontos flutuantes utilizam o pacote Linpack DP [109] e para o de operações com inteiros utilizam o pacote Dhraystone 2.1 [110]. Também são realizadas avaliações, com código desenvolvido pelos autores, das velocidades de leitura e escrita das portas lógicas. Em relação às portas lógicas, os autores concluem que a nova geração de SoCs apresenta claro desempenho superior em relação à gerações anteriores. No estudo ESP32 e seu predecessor (ESP8266) são apontados como pertencentes ao grupo referente à geração mais recente. ESP32, em relação a todos os dispositivos, obteve a quarta melhor avaliação com 176MIPS para o Dhrystone e 2805MIPS para o Linpack DP. Em referência a tal estudo, a seleção do ESP32 para o uso na presente tese se deu com base no fato de que, além de possuir desempenho satifatório e ser referido na literatura, é um SoC que possui as características anteriomente identificadas como necessárias à ampla adoção de IoT: baixo custo, dimensões reduzidas para os padrões atuais, implementa todo necessário à comunicação sem fio - "rico em recursos", fácil acesso à compra.

Em 5.1, apresentamos o diagrama de blocos do ESP32. De interesse direto ao

presente trabalho, destacamos a existência de aceleradores criptográficos, e baixo consumo quando comparado a sistemas que não sejam SoC [111].

O ESP32 opera em frequências de 80,160 e 240 MHz. Sem o modem/rádio ligado, o consumo apresentado pelo fabricante é de 27 a 44mA para 80Mhz, 27 a 44mA para 160Mhz e 30 a 68mA para 240Mhz. O consumo com o modem ligado é de 180mA (quando transmitindo com uma potência de transmissão de 14dBm) e durante a recepção é de 100mA. Há ainda um modo de operação onde as CPUs podem ser pausadas e reativadas mediante eventos externos, como por exemplo chegada de pacotes de rede na placa de rede ou outras interrupções de hardware. Para este caso o consumo é de 0,8mA.

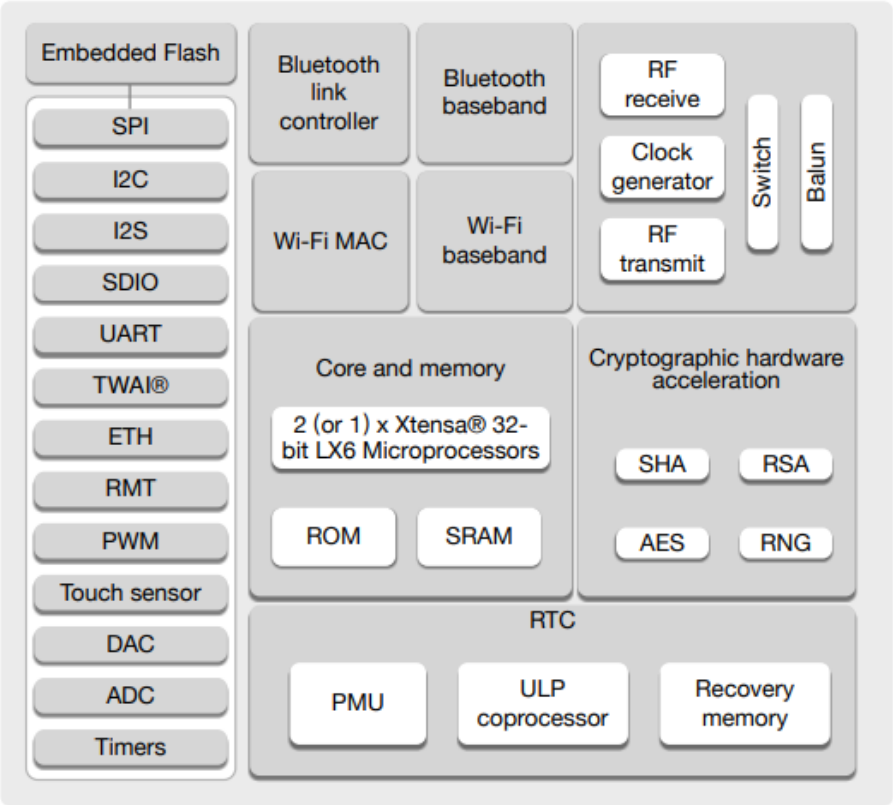


Figura 5.1: Diagrama de blocos ESP32. Extraído de [112]

No presente trabalho foi utilizado o módulo Esp-wroom-32 [113], que possui um chip ESP32 (Chip ESP32-D0WDQ6) acrescido de uma memória flash externa 4MB. Suas dimensões são de 25x18 milímetros.

5.3.2 Sistema Operacional FreeRTOS

FreeRTOS [98] é um sistema operacional em tempo real, preemptivo, de código aberto, cujo desenvolvimento tem como um dos objetivos sua portabilidade para diversos microcontroladores. Por utilizar poucos recursos para realizar suas funções e ser muito pequeno (9KB), já foi portado para mais de 40 arquiteturas de microcontroladores e mais de 15 conjuntos de ferramentas de compilação (toolchain).

Seu código se divide em dois conjuntos: a camada principal e a camada de hardware a ser portada. Na figura 5.2, apresentamos o conjunto de bibliotecas a serem portadas por cada fabricante.

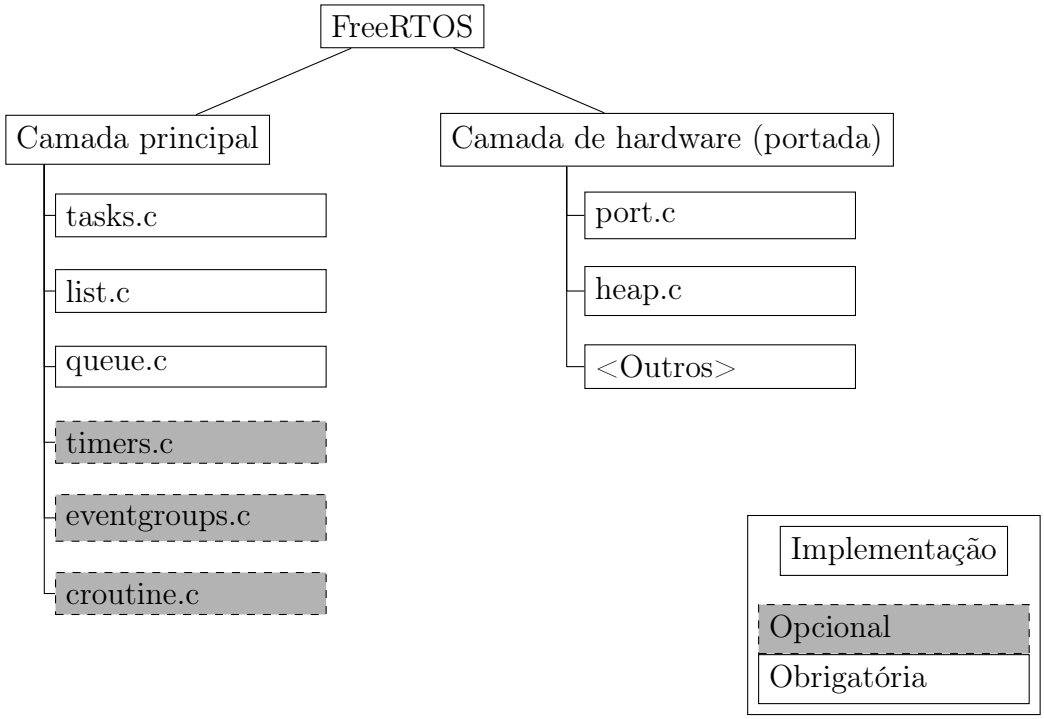


Figura 5.2: Camada principal. Implementações referentes a arquitetura portada devem ser implementadas nesses arquivos.

A implementação para uma determinada arquitetura se dá através da codificação das funções existentes em cada arquivo da camada de hardware. Na camada principal estão as funcionalidades expostas pelo *kernel*, comuns a todas as arquiteturas.

Dentre os recursos disponibilizados pelo *kernel*, destacamos: operação preemptiva, atribuições de prioridade a processos, filas de intercomunicação de processos, semáforos (binários, contadores, recursivos).

Sistema Operacional ESP-IDF - FreeRTOS

A versão utilizada do sistema ESP-IDF - FreeRTOS foi a 4.1.0 [114]. Trata-se da versão FreeRTOS v10.2.0 portada para o SOC ESP32. Destacamos que embora baseie-se no FreeRTOS algumas mudanças foram introduzidas pelo fabricante especificamente para este dispositivo [115]. Como exemplo, no sistema ESP-IDF a suspensão do escalonador (impedimento de que haja preempção) ocorre individualmente em cada processador, ou seja, a chamada à função $vTaskSuspendAll$ só impede o escalonamento referente ao processador utilizado no momento da chamada.

Outro ponto de atenção é que o escalonador *Round Robin* do sistema ESP-IDF não respeita a ordem de chegada de tarefas de mesma prioridade. Ou seja, sem os devidos cuidados (ex.:bloqueio) uma tarefa pode nunca ser posta em execução, não atendendo à propriedade descrita quando falamos anteriormente de exclusão mútua (ausência de esperas ilimitadas).

5.3.3 Ambiente de desenvolvimento

Utilizou-se o editor de códigos Visual Studio Code [116] juntamente com o plugin PlatformIO [117]. Tal plugin disponibiliza o ferramental para diversos dispositivos IoT dentre os quais o ESP32. O ambiente de desenvolvimento é multiplataforma (Linux, Windows e macOS) e gratuito.

Utilizou-se um sistema de banco de dados MariaDB [118], versão 10.2, onde foram armazenados os resultados experimentais.

Uma versão Linux CentOS 7 [119] foi utilizado como servidor. Nele utilizou-se um servidor Apache [120] 2.4.6, com suporte à PHP [121] (versão 7.2). Sua utilização na coleta dos dados experimentais será descrita à frente.

Os certificados utilizados para avaliações de comunicação segura com o uso de TLS foram gerados no servidor utilizando-se do OpenSSL versão 1.1.

5.3.4 Ambiente de comunicação sem fio

Foi utilizado um roteador Wifi TP-Link AC 1350 Archer C60 se comunicando com os dispositivos ESP32 mediante utilização de do protocolo 802.11n [122] com mecanismo de segurança WPA2-PSK [51].

5.4 Apresentação da Metodologia

O método proposto nesta tese destina-se à avaliação de processamento local realizado por sensores IoT. O método proposto realiza análise dinâmica [99] baseada em métricas refinadas [100], ambos anteriomente apresentados. Diferentemente do proposto em [99], onde a escolha das métricas refinadas se dá em função da mensuração das principais funcionalidades do sistema operacional, aqui estamos interessados em avaliar o processamento associado à realização das funções elementares realizadas por sensores IoT: sensoriamento, atuação e comunicação, com especial interesse na comunicação, visto que esta nos dá uma gama de possibilidades de escolha com reflexos diretos no processamento associado a cada.

Para realizarmos a análise dinâmica, apresentaremos a mecânica adotada para criação de traço de execução em sensores IoT com mais de um processador. O traço conterá uma lista sequencial de execução, onde cada elemento da lista informa o a tarefa executada, o processador utilizado, o número de ciclos que utilizou e o número de ciclos que o escalonador utilizou para selecioná-la.

As métricas refinadas são obtidas mediante a introdução de marcas (ou marcadores). Uma vez introduzidos, os elementos futuros do traço de execução estarão associados à última marca criada.

As marcas devem ser introduzidas por dois tipos distintos de usuários, que chamaremos de usuário final e implementador.

Ao usuário final compete inserir marcas em seu código, analisando o contexto da avaliação desejada. A tais marcas, quando necessário, iremos utilizar a nomeclatura marca de nível 1. Nesta tese, analisamos o reflexo do estado da rede em relação à execução do protocolo TLS, a fim de apresentar o uso do método em um problema real. Como exemplo, neste cenário, o usuário final insere marcas em seu código regular, antes da chamada de conexão, antes da chamada que inicia o handshake, antes de enviar dados, antes de solicitar dados, antes da chamada que aguarda dados após uma solicitação por estes. As métricas são inseridas pelo usuário final mediante a chamada oferecida pelo método proposto nesta tese, sem que ele precise ter conhecimento sobre o funcionamento interno. O usuário final precisa basicamente conhecer a função de inserir marcas, a de iniciar gravação de marcas, a de parar a gravação de marcas e a de enviar as marcas coletadas para o servidor que realizará

posterior análise.

O implementador é responsável por disponibilizar a versão compilada para que um usuário final a utilize na análise de um ou mais problemas específicos. O implementador determina locais estratégicos para a inserção de marcas, de modo que o traço gerado pelo usuário final seja incrementado com marcações extras. A tais marcas, quando necessário, iremos utilizar a nomeclatura marca de nível 2. Nesta tese, onde ilustramos o problema da interferência do estado da rede, as marcas são introduzidas automaticamente quando ocorre a transmissão de algum dado na rede. Ou seja, se o usuário inserir a marca "Handshake", as marcas introduzidas pelo implementador poderiam ser: "EnvioPacote1", "EnvioPacote2", "EnvioPacoteN". Ou também "RecebimentoPacote1", "RecebimentoPacote2", "RecebimentoPacoteN". Além disso, para esse problema em análise, disponibiliza-se contadores de pacotes para que o usuário final possa avaliar o número de pacotes enviados, o número de pacotes recebidos e o número de tentativas de conexão enviadas. Dessa forma, o usuário final pode avaliar, em alto nível, eventuais retransmissões ou reconexões mediante a avaliação desses contadores.

Por fim, o usuário final irá obter um traço com marcações detalhando o processamento. Ainda em relação ao exemplo acima, o traço obtido conteria marcações do tipo "Handshake,EnvioPacote1"e "Handshake,RecebimentoPacote1". Dessa forma, o usuário final pode, por exemplo, computar o processamento referente a todos os elementos que foram adicionados ao traço após a marcação "Handshake,EnvioPacote1"e antes da marcação "Handshake,RecebimentoPacote1".

Mediante a utilização dessas marcações, estabelece-se, a cada marcação introduzida, um grupo (ou agrupamento) de elementos do traço vinculados a tal marcação.

5.4.1 Coleta das informações de um traço de execução

Apresentamos uma visão macro dos passos executados durante a coleta. O dispositivo IoT (ESP32) e o servidor a que fazemos referência são os apresentados no ambiente experimental.

Foram geradas compilações distintas para diferentes valores de quantum e cada compilação foi enviada ao ESP32 para geração dos traços. Tal procedimento se fez

necessário uma vez que o quantum só pode ser alterado durante a compilação, ou seja, o sistema não permite uma troca de quantum durante sua execução.

Os passos realizados para coleta de traços, ilustrados na figura 5.3, são os seguintes:

1. O ESP32 se conecta ao servidor e informa seu quantum.

2. O servidor verifica os testes que já realizou para aquele quantum, e determina a próxima avaliação a realizar. A avaliação corresponde a qual função o dispositivo deve avaliar, a quantas repetições deve executar e seus parâmetros. Em situações onde uma avaliação tenha sido interrompida, o número de repetições informado corresponde somente à quantidade que falta repetir. A função a ser avaliada e o número de testes é então retornado ao ESP32 ou, caso não haja mais avaliações a realizar, o servidor informa ao dispositivo tal fato.

3. O ESP32 obtém a função de teste a realizar, o número de repetições e os parâmetros (clock, porta a se conectar e o tamanho de pacotes a receber). Caso o servidor indique que não há mais teste a realizar, o dispositivo permanece sem executar nada.

4. O ESP32 inicia a execução da função de testes.

5. A função de testes realiza sua configuração inicial que não varia a cada teste. Como exemplo, ajusta sua frequência caso esta não seja a atual. Neste momento, os objetos que poderiam ser alocados uma única vez pela aplicação são iniciados e eventual processamento não fará parte do traço. A função de testes passa então a executar cada repetição do bloco de instruções da função a ser avaliada.

6. No inicio de cada bloco a função que inicia a coleta é chamada garantindo que as medições só contenham processamento a partir daquele ponto. Ao final da execução do bloco a coleta é interrompida.

7. Uma avaliação do resultado da execução do bloco é feita. Se houve erro segundo o critério do experimento (exemplo: reconexão ou retransmissão de pacotes) a repetição é descartada. Caso contrário, o conjunto de informações

coletadas (traço de execução) é transmitida ao servidor. O ESP32 então apaga de sua memória os resultados daquela avaliação e, se necessário, realiza outra repetição.

8. Após executar todas as repetições necessárias o passo 1 volta a se repetir.

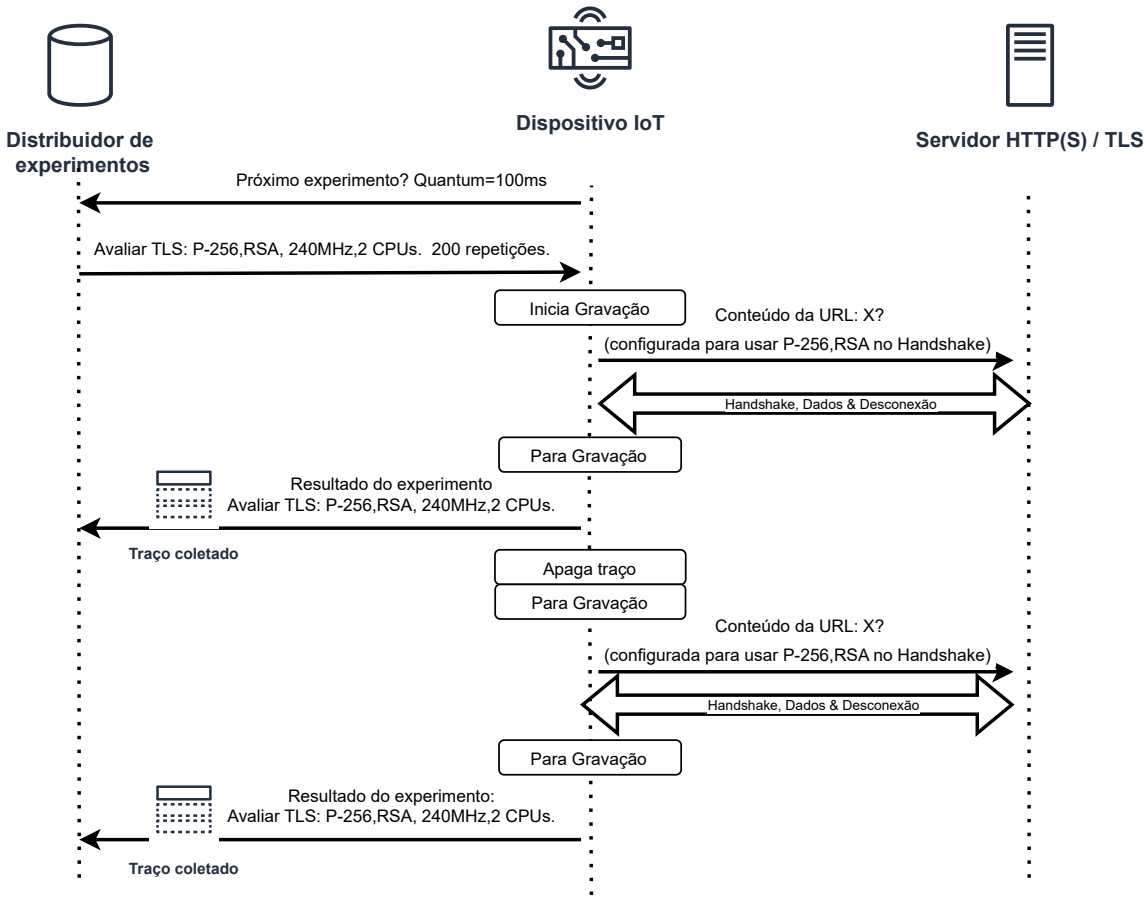


Figura 5.3: Exemplo de coleta de traços. Dispositivo solicita a um distribuidor o experimento a realizar e o número de repetições necessárias. Os traços coletados a cada repetição são enviados ao distribuidos para futura análise.

As funções a serem avaliadas são criadas mediante o uso de macros, como a função apresentada em 5.1. Verifica-se que, desta forma, não há a introdução de chamadas a funções durante a repetição, uma vez que a macro é substituída por um laço evitando-se, assim, o processamento extra, como o decorrente da necessidade de se gravar registradores ao se chamar uma função. As macros são as responsáveis pela criação do código final resultante dos passos acima descritos.

Apresentaremos a seguir mais detalhes referentes a cada traço gerado.

Listing 5.1: Definição de uma tarefa a ser avaliada. Uso de macros

```
1 EVLTR_SET_AS_TASK( ev_net_http_socket_connect, EVLTR_pre(
2
3 ////////////////
4 Inicializacao (Passo 5)
5 ////////////////
6
7 ), EVLTR_run(
8
9 ////////////////
10 Trecho a avaliar (Passo 6.Deve iniciar e terminar a coleta)
11 ////////////////
12
13 ),EVLTR_pos(
14 ////////////////
15 Desalocar recursos iniciais
16 ////////////////
17 ));
```

5.4.2 Exemplo de traço de execução

Nessa Seção, iniciaremos apresentando um exemplo do resultado referente a uma única avaliação de determinado experimento. Refere-se, portanto, a um traço de execução produzido no passo 6 anteriormente descrito. Este exemplo nos guiará no entendimento do método proposto. Ao longo do capítulo detalharemos as informações nele apresentado.

Na figura 6.2, visualizamos o resultado obtido após uma única repetição de determinado experimento. Tal resultado refere-se à avaliação de execução de uma tarefa (*ev_net_mdebtls*) criada, à nível de usuário, responsável por conectar-se, de forma segura, a um servidor para transferência e obtenção de informações. Um exemplo de sua utilização, seria a transmissão de um resultado de um conversor analógico e a obtenção de confirmação de que o mesmo foi recebido.

No topo da figura 6.2, encontram-se os nomes de todas as tarefas executadas. Embora a avaliação seja da tarefa *ev_net_mdebtls*, sua total execução se deu mediante a execução dela e de todas as demais apresentadas.

No gráfico, em escala logarítmica, encontra-se no eixo Y o número de ciclos

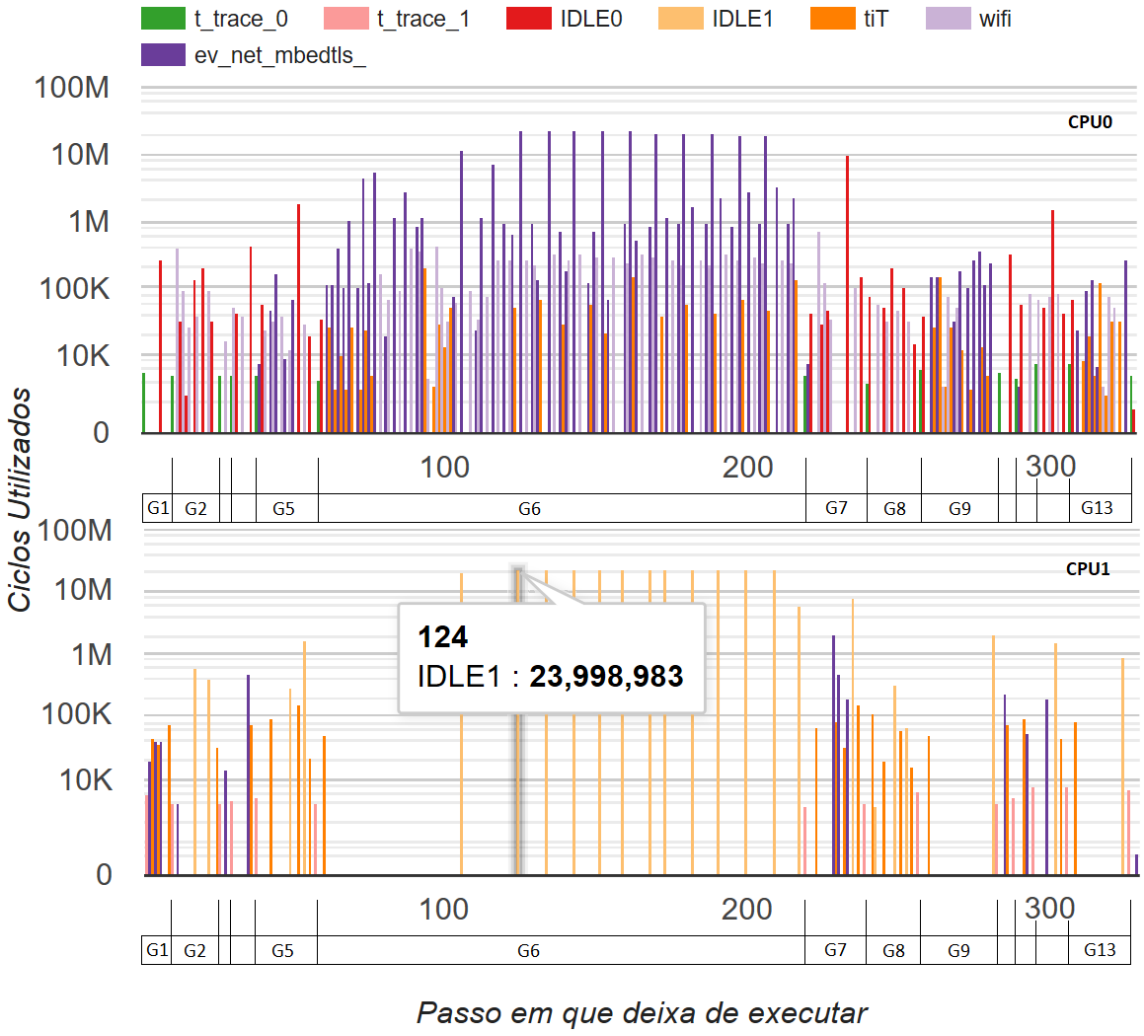


Figura 5.4: Traço coletado referente ao processamento local ocorrido durante uma conexão TLS. Clock 240MHZ, 2 processadores. Conjunto criptográfico TLS-ECDHE-RSA-WITH-AES-256-GCM-SHA384. Troca de chaves: Curve25519 (256 bits). Certificado RSA de 2048 bits. Tamanho dos dados recebidos: 400 bytes

utilizados por cada tarefa a cada vez que o escalonador a retira de execução (o que chamamos de "passo", visto no eixo X). Para melhor entendimento, os passos foram separados em dois gráficos, cada qual correspondente à CPU designada pelo escalonador para sua execução.

Destacamos que o exemplo se inicia pela execução de t_trace_0 na CPU0 e de t_trace_1 na CPU1. Posteriormente, CPU0 e CPU1 executaram, respectivamente, $IDLE0$ e $ev\_net\_mdebtls$. O último passo foi o 328, onde $ev\_net\_mdebtls$ se utilizou de 661 ciclos e a o experimento foi concluído.

Abaixo de cada gráfico em 6.2 também estão presentes o que as marcas introduzidas (G1 à G11). Cada marca possui um passo inicial e um final, sem sobreposição. Sua utilização será aprofundada mais adiante.

5.4.3 Tarefas do sistema operacional

Conforme descrito anteriormente, o sistema FreeRTOS provê um sistema operacional mínimo. Parte do sistema, a depender do ecossistema em questão, implementa certas funcionalidades do sistema através de tarefas independentes. Neste cenário, tanto tarefas do sistema como o programa principal são escalonados pelo sistema operacional. Verifica-se, no traço de execução apresentado na figura 6.2, que, além da tarefa avaliada, outras 6 foram postas em execução, das quais 4 se referem a tarefas providas pelo sistema operacional:

- **IDLE0 e IDLE1**: Tarefas postas em execução quando a CPU está ociosa (nenhuma outra tarefa está apta a executar). Mediante a aferição dos ciclos utilizados por tais tarefas, poderemos observar percentuais de ociosidade de cada CPU a cada experimento.

  Tais tarefas possuem prioridade mínima, ou seja, se uma outra tarefa de maior prioridade estiver apta a executar elas sairão de execução.

  Verifica-se, no segundo gráfico em 6.2, que, no passo 124 a tarefa utilizou se de 24 milhões, de ciclos assim como nos 10 passos subsequentes nesta mesma CPU. Observa-se, através dessa sequência de trechos de execução de $IDLE1$, que tais tarefas se comportam como qualquer outra, ou seja, quando esgotada sua fatia de tempo (quantum) ocorre uma procura por tarefa de igual prioridade.

Uma vez que outra não foi encontrada a mesma volta a executar. Uma vez que nesta sequência $IDLE1$ executou por 24×10^6 ciclos, verifica-se o quantum utilizado neste experimento.

- **wifi**: Tarefa responsável pela interface do sistema operacional com o dispositivo de rádio existente no SOC. De código fechado, atua como o driver e implementa, dentre outras funções, as necessárias ao estabelecimento de conexão do dispositivo com o roteador (ponto de acesso wifi). Pode-se observar na figura que tal tarefa é posta em execução quando ocorre envio ou recebimento de dados e também, em intervalos fixos, a fim de que sejam trocadas mensagens entre o dispositivo e o roteador com o objetivo de se manter ativa a conexão estabelecida entre ambos.

- **tiT**: Tarefa responsável pela implementação dos protocolos TCP/IP. No dispositivo utilizado tal tarefa baseia-se na biblioteca LWIP. Através da observação de resultados de experimentos, verifica-se que uma vez que os dados transmitidos ou recebidos através da pilha TCP/IP também passam pela tarefa wifi uma vez que essa é responsável pela camada física do dispositivo de rede sem fio (Wifi)

5.4.4 Obtenção do traço de execução

O traço de execução é composto pelo conjunto de informações de todas as tarefas escalonadas desde o início até a conclusão de uma repetição de dado experimento. O monitoramento da entrada e da retirada de execução de cada tarefa se utilizou de duas funções existentes nas APIs do sistema operacional, que são ativadas durante o processo de escalonamento. Tais funções são executadas no contexto da tarefa e do processador responsável por executá-la, com a garantia de que: (i) qualquer tarefa executada passará antes pela função de entrada e ao final pela de saída, (ii) durante a passagem pela função de entrada e de saída o processador utilizado é o mesmo e (iii) a chamada a tais funções se dão no contexto de um serviço de interrupção, não havendo portanto, interrupção da execução das mesmas.

Entrada em execução

Uma vez que o escalonador escolha a tarefa que irá ocupar determinada CPU, armazenamos as informações necessárias a fim de que, ao final da execução desta tarefa, as informações acerca de sua exeução sejam computadas e adicionadas ao traço. Uma tarefa, quando iniciada, adquire um identificador único (instância). Aqui utilizaremos instância e tarefa indistintamente, a menos que, no contexto apresentado, seja necessário distingui-las.

Apresentamos em 5.2 um resumo do código e das estruturas de dados utilizadas para computar informações de cada tarefa existente no traço. A estrutura *task_trace_task_node* representa cada instância que foi alguma vez executada durante o experimento. Independente do número de vezes que determinada instância é posta em execução, durante todo traço uma mesma instância possuirá uma única estrutura *task_trace_task_node* a ela associada. Uma instância não pode estar executando simultaneamente em mais de uma CPU, de forma que a cada entrada em execução armazenamos o valor atual do contador de ciclos da respectiva CPU em *cpu_cycle_last_switched_in*.

A fim de evitarmos a introdução de processamento extra, um vetor contendo informações de instâncias é pré-alocado antes de iniciarmos a gravação as medições. Se ele não fosse pré alocado, haveria um processamento extra, referente a alocação de memória para cada elemento do traço (alocação dinâmica). Cada posição no vetor corresponde ao identificador único (task id) de cada instância. Tal identificador é gerado pelo sistema operacional cada vez que uma instância é criada (identificador da instância da tarefa em execução). O tamanho do vetor $TASK\_TRACE\_MAX\_TASK\_INDEX$ deve ser capaz de comportar as instâncias não só das tarefas que desejamos medir, mas também das eventuais tarefas criadas pelo sistema operacional. Ao iniciarmos a medição de um novo experimento, tal vetor é reiniciado atribuindo-se zero ao valor de *initiated* de cada um de seus elementos. Quando uma instância é posta em execução verifica-se, na posição de número correspondente ao id da instância da tarefa, se o elemento foi iniciado. Caso não tenha sido, o nome da tarefa é copiado.

A função *task_trace_switched_in* é a informada à API do sistema como sendo a função a ser chamada antes que cada tarefa seja posta em execução. A determinação

do momento em que a tarefa é posta em execução (ou retirada) se dá mediante uma chamada de leitura a um contador de ciclos existente em cada processador (*task_trace_get_cycle_count*). A chamada é realizada em *assembler* de forma a minimizar processamento decorrente da introdução de medições. A cada ciclo o contador é incrementado e sua capacidade é de 32 bits sendo, portanto, necessário um tratamento, na retirada de execução, para o caso em que seu valor tenha sido reiniciado.

O valor do contador de ciclos é armazenado somente ao final da chamada à função *task_trace_switched_in* a fim de que a contagem de ciclos gastos pela instância da tarefa se inicie somente após qualquer outro processamento decorrente da medição, como, por exemplo, a inicialização da instância da tarefa no vetor de tarefas.

Destacamos que para a máxima frequência de 240MHz esse contador se reinicia a cerca de 17 segundos, não sendo problema utilizá-lo na avaliação, uma vez que o quantum máximo utilizado nos experimentos foi de 300ms. Tal contador é o mesmo utilizado pelo sistema operacional internamente para realização de escalonamento.

Listing 5.2: Função executada por qualquer tarefa antes de iniciar ou retomar execução

```
1  typedef struct task_trace_task_node {
2      unsigned int cpu_cycle_last_switched_in;
3      char ignore;
4      char is_running;
5      char task_name[CONFIG_FREERTOS_MAX_TASK_NAME_LEN];
6      char initiated;
7      // contador a ser utilizado quando
8      // nova marca criada
9      unsigned int cpu_cycle_to_use_next_switched_out;
10     char use_this_cpu_cycle_when_switched_out;
11 } task_trace_stTaskInfo, *task_trace_ptTaskInfo;
12
13 #define TASK_TRACE_MAX_TASK_INDEX 100
14
15 task_trace_stTaskInfo
16     task_trace_stTaskTrace_vector[TASK_TRACE_MAX_TASK_INDEX];
17
18 void task_trace_switched_in() {
```

```
19  int task_idx = TASK_TRACE_CURRENT_TASK_IDX_ON_CPU();
20  task_trace_ptTaskInfo node = &task_trace_stTaskTrace_vector[task_idx];
21  // Inicia estrutura caso seja 1a vez que a tarefa entra em execucao
22  node->cpu_cycle_last_switched_in = task_trace_get_cycle_count();
23 }
24
25 static inline uint32_t task_trace_get_cycle_count()  {
26     uint32_t __ccount;
27     __asm__ __volatile__("rsr.ccount %0" : "=a"(__ccount));
28     return __ccount;
29 }
```

Retirada de execução

Uma tarefa, ao ser retirada de execução, produz informações que serão coletadas conforme código exibido em 5.3. A cada retirada de execução uma estrutura *task_trace_stat* é preenchida. Neste momento armazena-se o identificador da tarefa retirada, a CPU utilizada, o número de ciclos utilizados nesta fatia de execução, o número de ciclos que o escalonador utilizou para seleção da tarefa e a marca a que pertence.

A fim de que o processamento introduzido seja mínimo, o vetor *task_trace_stStat_vector*, de estruturas *task_trace_stat*, é pré-alocado antes do início da coleta de qualquer medição. O índice *task_trace_stStat_vector_curr_idx* é zerado a cada novo experimento e indica a posição do vetor de estruturas que será alimentada na próxima retirada, evitando-se assim qualquer alocação dinâmica. O tamanho deste vetor (tamanho máximo de retiradas de execução permitidas por experimento) é determinado por $TASK\_TRACE\_MAX\_DETAILS$.

A cada retirada de execução o valor de *task_trace_stStat_vector_curr_idx* é incrementado. Como cada processador realiza a interrupção de escalonamento individualmente (com base em seu respectivo contador de ciclos), o incremento do índice foi protegido de acesso simultâneo.

Importante observar que a execução da função de saída se dá, quando for o caso, após o término da fatia de tempo (quantum) e que a introdução de processamento

nesta função não diminui seu tempo de execução. O sistema ajusta o valor do contador de interrupções do escalonador em trecho posterior, imediatamente antes da tarefa ser posta em execução.

A função *task_trace_switched_out* é chamada imediatamente antes do escalonador retirar uma tarefa de execução. Observa-se em sua implementação que o armazenamento temporário do contador de ciclos é realizado em primeiro lugar. Desta forma os ciclos utilizados pela tarefa são calculados através da diferença do valor do contador avaliado no início da função *task_trace_switched_out* e de seu valor anteriormente armazenado ao final da função *task_trace_switched_in*, de forma a desconsiderar eventual processamento introduzido por estas funções. Observa-se também no código apresentado o tratamento para o caso de o contador de ciclos ter sido reiniciado. Neste caso, seu valor será inferior ao valor inicial, indicando que a contagem passou por zero (reiniciou).

Os ciclos utilizados pelo escalonador a fim de selecionar a tarefa, que agora foi retirada de execução, são computados através da diferença entre o contador de ciclos no momento em que realizou, no processador em questão, a retirada da tarefa anterior (*task_trace_last_switch_out*[*core*]) e o valor do contador no momento da entrada da tarefa agora sendo retirada.

Listing 5.3: Função executada por qualquer tarefa antes de ser retirada de execução

```
1
2  typedef struct task_trace_stat {
3      char task_idx;
4      unsigned int cycles_running;
5      unsigned int cycles_scheduler;
6      char cpu;
7  } task_trace_stStat, *task_trace_ptStat;
8
9
10 #define TASK_TRACE_MAX_DETAILS 3500
11
12 task_trace_stStat task_trace_stStat_vector[TASK_TRACE_MAX_DETAILS];
13 int task_trace_stStat_vector_curr_idx = 0;
14 unsigned int task_trace_last_switch_out[portNUM_PROCESSORS] = {0};
15
```

```
16
17 void task_trace_switched_out() {
18       unsigned int stop = task_trace_get_cycle_count();
19       int task_idx = TASK_TRACE_CURRENT_TASK_IDX_ON_CPU();
20       task_trace_ptTaskInfo node =
21              &task_trace_stTaskTrace_vector[task_idx];
22
23       unsigned int cycles = 0;
24       if (stop >= node->cpu_cycle_last_switched_in) {
25          cycles = stop + 0x00000000 - node->cpu_cycle_last_switched_in;
26       } else {
27          cycles = stop +(0xFFFFFFFF - node->cpu_cycle_last_switched_in);
28       }
29       int detail_idx = 0;
30       taskENTER_CRITICAL_ISR(&task_trace_mutex);
31        detail_idx = task_trace_stStat_vector_curr_idx;
32        task_trace_stStat_vector_curr_idx++;
33       taskEXIT_CRITICAL_ISR(&task_trace_mutex);
34
35
36       task_trace_ptStat detail = &task_trace_stStat_vector[detail_idx];
37       detail->task_idx = task_idx;
38       detail->cycles_running = cycles;
39       detail->cpu = core;
40
41
42       task_trace_last_switch_out[core] = task_trace_get_cycle_count();
43 }
```

Traço coletado

Na tabela 5.1, apresentamos um trecho do traço de execução referente ao exemplo apresentado neste capítulo.

A coluna passo refere-se a um contador que indica, em ordem cronológica, a retirada da tarefa de execução, bem como a CPU que ocupava. Observa-se na listagem: os ciclos utilizados pelas tarefas do sistema operacional anteriormente descritas e pela tarefa sendo avaliada (*ev_net_mdebtls*). Observa-se também: (i)

Tabela 5.1: Trecho do traço referentes ao exemplo (figura 6.2)

passo	id. marca	cpu	ciclos executando	ciclos para escalonar tarefa	nome da tarefa
1	8	1	6.882	0	t_trace_1
2	8	0	6.563	0	t_trace_0
3	8	1	16.938	1.282	ev_net_mbedtls_
4	8	1	30.336	823	tiT
5	8	1	33.275	1.054	ev_net_mbedtls_
6	8	1	21.115	666	tiT
7	8	1	30.305	1.054	ev_net_mbedtls_
8	8	0	205.556	1.464	IDLE0
9	8	1	66.183	666	tiT
10	8	1	4.397	741	t_trace_1
11	9	0	76.8	763	t_trace_0
12	9	1	5.301	1.203	ev_net_mbedtls_
13	9	0	359.666	1.147	wifi
...	...	...	....	....	....
328	20	0	5.074	666	tiT
329	20	1	549.776	1.218	IDLE1
330	20	0	259.217	1.054	ev_net_mbedtls_
331	20	1	7.672	728	t_trace_1
332	21	0	5.463	741	t_trace_0
333	21	0	1.569	994	IDLE0
334	21	1	661	1.462	ev_net_mbedtls_

a existência de uma marca associada a cada execução e (ii) duas tarefas (t_trace_0 e t_trace_1) que são executadas a cada vez que uma marca é criada.

5.4.5 Obtenção de métricas refinadas mediante introdução de marcas

Para que possamos analisar o comportamento do resultado de cada experimento em relação às métricas refinadas, torna-se necessário avaliarmos trechos do traço. Trata-se da capacidade de atribuirmos uma marca, mediante introdução de código específico, ao qual cada tarefa retirada de execução passará a estar associada. Seu uso mais simples se dá no início da coleta de informações: nenhum dado é gravado até que ocorra a criação da primeira marca. Tal característica garante, por exemplo, que processamento decorrente de inicializações que possam se dar unicamente não seja computado a cada execução do experimento. Na tabela 5.2, se encontram as marcas do exemplo apresentado (Figura 6.2).

Verifica-se na tabela que as marcas referentes ao *Handshake* realizado pelo protocolo TLS 1.2 ocorreram entre os passos 31 e 292. As descrições referentes às colunas Nível 1 são introduzidas, pelo usuário final, no código a ser analisado. As

Marca	Nível 1	Nível 2	Passo Inicial	Passo Final
G1	Conexão	Inicialização	1	10
G2	Conexão	T_O_SYN-000	11	26
G3	Conexão	T_I_SYN-000	27	30
G4	Handshake	Inicialização	31	38
G5	Handshake	T_O_DATA-000	39	56
G6	Handshake	T_I_DATA-000	57	222
G7	Handshake	T_O_DATA-001	223	243
G8	Handshake	T_O_DATA-002	244	266
G9	Handshake	T_I_DATA-001	267	292
G10	Envio de dados	Inicialização	293	297
G11	Envio de dados	T_O_DATA-003	298	304
G12	Recebimento de dados	Inicialização	305	319
G13	Recebimento de dados	T_I_DATA-002	320	331
G14	Término		332	334

Tabela 5.2: Marcas referentes ao exemplo (figura 6.2)

descrições existentes na coluna Nível 2 foram adicionadas automaticamente (introduzidas pelo "implementador como descrito anteriormente). As de Nível 2 estão vinculadas a marca de Nível 1 anteriormente criada. Uma apresentação da estratégia na escolha dos locais de introdução de novas marcas será apresentada mais à frente nesse capítulo. Por hora, destacamos que a marca $G5$ da tabela 5.2 refere-se a todo processamento realizado durante o $Handshake$, após sua inicialização (iniciada no passo 39) e até que a tarefa tit solicitasse à tarefa $wifi$ o envio do pacote TCP contendo dados. Neste exemplo, o índice de tal pacote de dados, no contexto do experimento, era 000 e por isso o nome dado a marca foi T_O_DATA_000 representando TCP, Saída (O), dados com índice 000 (primeira saída de dados TCP). Similarmente G9 indica que os passos 267 à 292 referem-se ao processamento ocorrido após o G8 até o recebimento do pacote de dados, cujo índice interno é 001 (segundo pacote recebido).

Sincronização de processadores mediante introdução de tarefas t_trace

As marcas são introduzidas mediante chamada que, dentre outras rotinas, retira de execução cada tarefa presente em cada CPU. Mediante a retirada de execução, o processamento realizado por determinada tarefa, desde o último momento que foi posta em execução, ficará vinculado à marca até então existente. Futuramente, após sua retomada, seu processamento estará vinculado à marca criada. Também faz-se necessário não contabilizar os ciclos associados ao processamento introduzido, referente à criação de nova marca, à tarefa sendo executada. Do contrário, os valores

aferidos conteriam processamento não existente anteriormente.

A fim de que a introdução de nova marca se realize conforme descrito, durante a inicialização global da ferramenta, foram criadas, para cada processador, tarefas de prioridade máxima cujo nome dado se inicia por *t_trace*. Uma vez criadas, tais tarefas ficam suspensas aguardando uma notificação para que sejam passíveis de execução. A prioridade máxima tem como objetivo garantir que sejam postas em execução quando ocorrer o próximo escalonamento.

No exemplo apresentado na figura 6.2, essas tarefas são *t_trace_0* e *t_trace_1* e verifica-se que estão vinculadas a seus respectivos processadores. Ao se chamar a função que inicia nova marca, esta toma para si um semáforo, a fim de impedir que seja novamente executada antes de sua completude. Em seguida, tal função envia uma mensagem a cada *t_trace* para que sejam consideradas aptas a executar. A chamada que inicia nova marca é executada no contexto da tarefa que a chamou.

Em 5.4 apresentamos um esboço do código executado por cada *t_trace*.

Listing 5.4: Código referente a criação de tarefas *t_trace*

```
1
2 TaskHandle_t task_trace_ptask_handles[portNUM_PROCESSORS];
3 volatile char task_trace_ptask_busywait_step[portNUM_PROCESSORS] = {0};
4
5 void task_trace_init_core_task( void * pvParameters ) {
6     int i;
7     while(1) {
8         ulTaskNotifyTake( pdTRUE,portMAX_DELAY );
9         task_trace_ptask_busywait_step[xPortGetCoreID()]=1;
10        for(i=0;i<portNUM_PROCESSORS;i++)
11            while(task_trace_ptask_busywait_step[i]<1);
12        if (xPortGetCoreID()==0) {
13            task_trace_stats_parse();
14            task_trace_ptask_busywait_step[xPortGetCoreID()]=2;
15        } else {
16            task_trace_ptask_busywait_step[xPortGetCoreID()]=2;
17        }
18        for(i=0;i<portNUM_PROCESSORS;i++)
19            while(task_trace_ptask_busywait_step[i]<2);
20        if (xPortGetCoreID()==0)
21            xSemaphoreGive(task_trace_barr_wait_pinned_finish);
```

```
22        }
23  }
```

Observa-se que, similarmente ao código de Peterson 4.2, são utilizadas esperas ocupadas, neste caso não como um mecanismo de proteção a variáveis, mas sim como um mecanismo de sincronização de execução entre processadores. Cada tarefa t_trace recebe uma notificação ($ulTaskNotifyTake$) e fica aguardando até que todas as demais t_trace também a recebam e cheguem ao passo 1 (linha 11). Após este laço e por terem prioridade máxima, nenhuma outra tarefa que não uma t_trace estará executando em qualquer processador. Uma delas (a do processador com índice 0) realiza qualquer processamento necessário ($task\_trace\_stats\_parse()$) enquanto as demais aguardam até que todas cheguem no passo 2. Todas as tarefas então atingem o passo 3 para finalizarem. Importante observar que: (i) depois que todas atingirem o passo 1, somente $t\_trace[CPU\_NUM]$ ocupará as CPUS, (ii) todo processamento realizado por $task\_trace\_stats\_parse()$ ficará vinculado à tarefa t_trace_0. Um exemplo da sincronização entre processadores proposta, aplicada à introdução de nova marcas, é apresentada na figura 5.5.

5.5 Metodologia aplicada a avaliações na ocorrência de comunicação em rede

Atrasos, perdas de pacotes e eventuais retransmissões decorrentes da comunicação em rede influenciam os resultados experimentais. Uma avaliação que envolva comunicação em rede e que se baseie exclusivamente em medição de tempo decorrido entre o incio da execução e seu final está sujeita a tais interferências e, a depender da estratégia adotada, serão desprezadas, ignorados ou estimadas.

Um dos objetivos dos experimentos realizados utilizando o método proposto nesta tese é a produção de resultados não influenciados pelo estado da rede. A estratégia adotada inicia-se pela separação do processamento associado ao envio realizado pela camada de enlace (modelo OSI) daquele associado a camadas superiores.

Os experimentos realizados nesta tese, quando referentes à avaliação do processamento decorrente da comunicação em rede, utilizaram-se do protocolo TCP, de

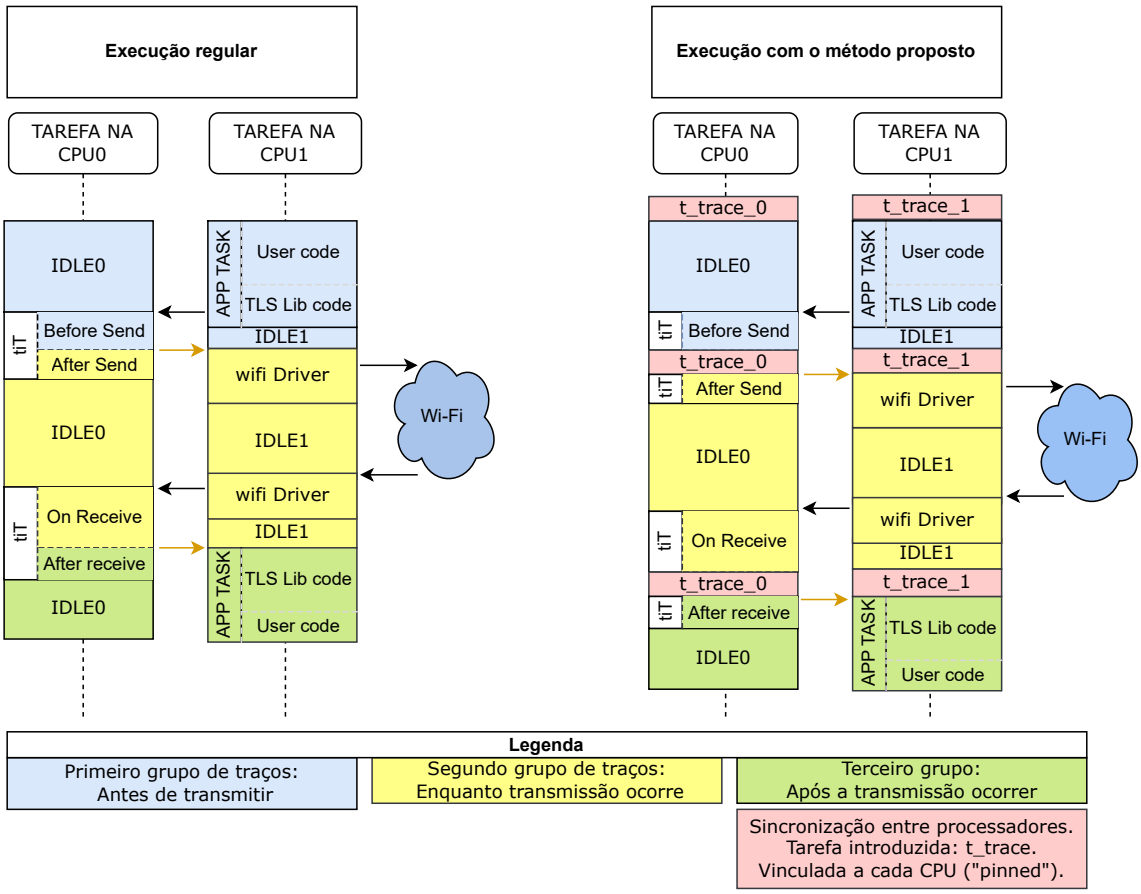


Figura 5.5: Exemplo da sincronização entre processadores proposta, aplicada à introdução de nova marca. Observa-se o fatiamento do processamento original a fim de que se distribua o processamento realizado entre as métricas refinadas.

forma que associamos um contador a cada marcador (TCP FLAG) de interesse. A cada vez que um pacote com um marcador TCP de interesse é gerado ou recebido, é criado nova marca. Como resultado as marcas introduzidas nos permitem identificar o processamento associado a cada tarefa até que haja envio ou recebimento de pacote contendo marcador TCP específico.

A opção pela criação das métricas refinadas utilizando-se do protocolo TCP se deu pelo fato de que o código fonte referente a drivers, como os *drivers* da interface de rede, muitas vezes são negados pelo fabricante, como é o caso do ambiente experimental utilizado nesta tese. Cabe ressaltar que poderiam ser feitas no *driver*, caso seu código seja acessível ou haja algum outro mecanismo para isso como APIs da camada física de rede.

Na figura 5.6, apresentamos um exemplo do processamento realizado para execução de uma requisição por dados, seguida por espera decorrente da transmissão até que se obtenha a resposta. No exemplo, verifica-se que a execução se deu mediante a execução das tarefas referentes à aplicação, ao TCP (tiT) e à comunicação em rede (processo wifi).

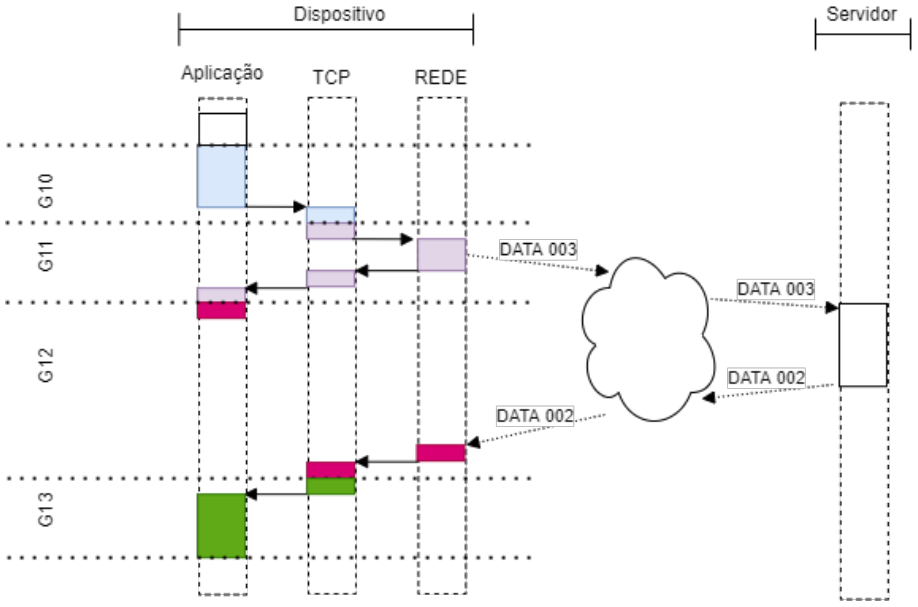


Figura 5.6: Em G10 a aplicação realiza a requisição. A biblioteca TCP, ao criar o pacote, inicia G11, concluindo o processamento associado à requisição. G12 compreende qualquer processamento associado à espera e G13 o processamento referente a leitura da resposta. Para um canal seguro, G10 está associado a cifragem e G12 à decifragem de informações

Ainda em relação à figura 5.6, a marca G11 refere-se a todo processamento

realizado para o envio do pacote TCP contendo a solicitação por dados (requisição). Imediatamente após G11, iniciamos uma nova marca referente à solicitação de que a aplicação leia os dados (G12) provenientes da rede. Tal requisição só será atendida após a chegada do pacote que efetivamente contém dados, o que ocasiona o início da marca G13.

Em relação à G12, a simulação de um atraso de rede, mediante a introdução de uma instrução de pausa no servidor da figura 5.6, ocasionaria um aumento no traço de execução associado à G12. A depender da latência da rede, esta também poderia impactar consideravelmente a avaliação dos resultados obtidos. Mediante o emprego do método proposto, G12 identificará no traço tais situações. A influência de tais situações sobre a qualidade do resultado final será novamente abordada à frente, durante a apresentação dos resultados experimentais.

5.5.1 Identificação de retransmissão

A avaliação proposta visa à obtenção de resultados não influenciados por efeitos decorrentes de retransmissões de rede. Sem prejuízo na utilização do método proposto, em cenários onde ocorram retransmissões os resultados aqui encontrados servirão de base para futuras avaliações que se proponham a considerar tais efeitos e eventualmente procurem, por exemplo, obter uma melhor qualidade na infra estrutura de rede utilizada. Neste contexto, a partir da obtenção dos resultados em um cenário sem retransmissões e mediante a avaliação de retransmissões médias que possam ocorrer, o processamento associado a estas poderia ser estimado com base nos dados aqui apresentados.

Na tabela anteriormente apresentada (5.2), a coluna Nível 2 nos permite observar de que forma podemos identificar eventuais retransmissões. Nela verifica-se que a marca G2 possui identificador T_O_SYN_000, onde T_O representa o envio de um pacote TCP contendo marcador SYN cujo indíce é zero (000). De maneira análoga, a marca G13 possui identificação T_I_DATA_002, onde T_I identifica a recepção de um pacote TCP contendo dados cujo índice é 2.

O procedimento adotado para descarte de experimentos, onde tenha ocorrido retransmissão se dá mediante a utilização de contadores de pacotes TCP reiniciados a cada experimento. Tais contadores são apresentados em 5.5 (enumeradas em

lwip_hookeval_getstats_types). Similarmente às APIs disponíveis pelo escalonador, a biblioteca LWIP nos permite a introdução de funções, neste caso executadas antes do envio e recepção de pacotes TCP.

Listing 5.5: Contadores de pacotes TCP

```
1  enum lwip_hookeval_getstats_types {
2          SEND_SYN, SEND_FIN, SEND_DATA,
3          SEND_ACK_NODATA, RECEIVE_SYN,
4          RECEIVE_FIN, RECEIVE_DATA,
5          RECEIVE_ACK_NODATA, END_MARK
6          };
7
8  uint32_t lwip_hookeval_getstats_counter[END_MARK];
9
10 #define LWIP_HOOKEVAL_INC_COUNTER(type)
11         lwip_hookeval_getstats_counter[type] =
12             (lwip_hookeval_getstats_counter[type]+1)
13
14 err_t lwip_hookeval_tcp_inpacket(struct tcp_pcb* pcb
15         , struct tcp_hdr *hdr, u16_t optlen
16         , u16_t opt1len, u8_t *opt2, struct pbuf *p)
17 {
18     u8_t flags = TCPH_FLAGS(hdr);
19      if((flags & TCP_FIN)) {
20             task_trace_new_stats("T_I_FIN"
21                 ,(lwip_hookeval_getstats_counter[RECEIVE_FIN]),1,0,0);
22             LWIP_HOOKEVAL_INC_COUNTER(RECEIVE_FIN);
23     } else if((flags & TCP_SYN)) {
24             task_trace_new_stats("T_I_SYN"
25                 ,(lwip_hookeval_getstats_counter[RECEIVE_SYN]),1,0,0);
26             LWIP_HOOKEVAL_INC_COUNTER(RECEIVE_SYN);
27     } else if(p->len>0) {
28             task_trace_new_stats("T_I_DATA"
29                 ,lwip_hookeval_getstats_counter[RECEIVE_DATA],1,1,0);
30             LWIP_HOOKEVAL_INC_COUNTER(RECEIVE_DATA);
31     } else if( (p->len==sizeof(struct tcp_hdr)) && (flags & TCP_ACK)) {
32         LWIP_HOOKEVAL_INC_COUNTER(RECEIVE_ACK_NODATA);
33     }
34     return ERR_OK;
```

```
35 }
36 u32_t * lwip_hookeval_tcp_out_add_tcpopts(struct pbuf *p
37     , struct tcp_hdr *hdr, struct tcp_pcb *pcb, u32_t *opts) {
38      u8_t flags = TCPH_FLAGS(hdr);
39     if((flags & TCP_FIN)) {
40         task_trace_new_stats("T_O_FIN"
41             ,lwip_hookeval_getstats_counter[SEND_FIN],1,0,0);
42         LWIP_HOOKEVAL_INC_COUNTER(SEND_FIN);
43     } else if((flags & TCP_SYN)) {
44         task_trace_new_stats("T_O_SYN"
45             ,lwip_hookeval_getstats_counter[SEND_SYN],1,0,0);
46         LWIP_HOOKEVAL_INC_COUNTER(SEND_SYN);
47     } else if(p->len > sizeof(struct tcp_hdr)) {
48         task_trace_new_stats("T_O_DATA"
49             ,lwip_hookeval_getstats_counter[SEND_DATA],1,1,0);
50         LWIP_HOOKEVAL_INC_COUNTER(SEND_DATA);
51     } else if( (p->len==sizeof(struct tcp_hdr)) && (flags & TCP_ACK)) {
52         LWIP_HOOKEVAL_INC_COUNTER(SEND_ACK_NODATA);
53     }
54     return opts;
55 }
```

Os contadores TCP são reiniciados antes que se inicie a medição. Cada vez que um pacote TCP é recebido, ou antes de ser enviado ocorre, uma análise do mesmo, a fim de que o respectivo contador seja incrementado e nova marca seja iniciada (funções *lwip_hookeval_tcp_inpacket* e *lwip_hookeval_tcp_out_add_tcpopts*).

Ao final da geração do traço os valores de cada contador são comparados com valores predefinidos. A determinação dos valores a serem comparados depende do experimento a ser avaliado.

Em experimentos onde utilizou-se TLS, a determinação do número de pacotes TCP transmitidos depende do conjunto criptográfico utilizado. A depender do tamanho do certificado utilizado, a autenticação (*Hansdhake*) poderá utilizar-se de mais de um segmento TCP, conforme descrito na tabela 3.3. Verifica-se nesta que um certificado RSA com dureza 128 possuirá tamanho de 3072 bytes, de forma que será necessária a transmissão de um segmento a mais quando comparada com os protocolos ECC ou com o RSA utilizando-se de dureza inferior. O descarte do expe-

rimento, após o $Hansdhake$, ocorrerá caso o valor do contador $RECEIVE\_DATA$ não possua o valor esperado. Tal valor depende do tamanho do certificado e por esse motivo é parametrizado a depender da autenticação.

No presente trabalho as avaliações relacionadas à troca de dados em rede utilizaram-se de pacotes TCP de tamanhos predefinidos, onde cada requisição ou resposta possui tamanho inferior ao tamanho máximo de um segmento TCP (mss). Quando a troca de dados foi avaliada utilizando-se TLS, considerou-se que o TLS Record Protocol também cabe em um único segmento TCP. Tal abordagem não compromete eventual avaliação de transmissão de dados em quantidade superior à máxima avaliada uma vez que tais transmissões se darão mediante a divisão e transmissão de um ou mais segmentos cujo tamanho será igual ou inferior ao tamanho máximo de segmento.

Por fim, eventuais retransmissões de pacotes de conexão também invalidam o experimento e são avaliadas através dos contadores $RECEIVE\_SYN$ e $SEND\_SYN$ que deverão possuir apenas um incremento ao longo de cada experimento.

5.6 Métricas propostas

5.6.1 Processamento por CPU

Uma vez que o traço de execução de um experimento seja concluído no dispositivo, o traço é persistido para posterior análise. No presente trabalho, tal análise se deu mediante o uso de um sistema de banco de dados ($SGBD$).

A partir do traço de execução de cada experimento passamos a avaliar, em relação a cada marca, os ciclos utilizados por cada CPU. Uma vez que cada passo refere-se ao momento em que a tarefa foi retirada de execução, iniciamos a análise mediante a computação da soma acumulada de processamento (referente a execução e ao escalonamento) em cada CPU para que possamos obter o intervalo de execução de cada tarefa em seu respectivo processador.

Como observado na tabela 5.3, as tarefas auxiliares introduzidas (t_trace_0 e t_trace_1) delimitam o início e fim de cada marca de execução. Verifica-se, no passo 240 da CPU 1, que a tarefa tiT ao deixar de executar deu lugar à tarefa

*t_trace_*1. Portanto, a tarefa tiT foi a responsável por iniciar o mecanismo de adição de nova marca. de forma que, após sua retirada de execução, *t_trace_*1 ocupe seu processador até que se conclua a introdução da nova marca. Observa-se também que no passo 241 IDLE0 deixou de ocupar a CPU 0 para ser sucedida pela tarefa *t_trace_*0 no próximo passo desta mesma CPU (não exibido na tabela).

Tabela 5.3: Processamento por CPU referente ao Handshake, após o envio do primeiro pacote (G7 da tabela 5.2). As coluna inicio e fim indicam o intervalo de execução de cada processo, em cada CPU

cpu0						**cpu1**					
	ciclos utilizados			**ciclo**			ciclos utilizados			**ciclo**	
#	exec.	escal.	tarefa	**inicial**	**final**	#	exec.	escal.	tarefa	**inicial**	**final**
222	8569	826	t_trace0	-	-	225	270505	857	tiT	1	271362
223	7307	1105	*	1	8412	231	958198	1143	*	271363	1230703
224	48958	1121	IDLE0	8413	58491	232	1156487	651	*	1230704	2387841
226	823939	703	wifi	58492	883133	233	91258	666	tiT	2387842	2479765
227	34371	1464	IDLE0	883134	918968	234	500790	1054	*	2479766	2981609
228	121241	680	wifi	918969	1040889	235	34822	666	tiT	2981610	3017097
229	76773	1465	IDLE0	1040890	1119127	236	187892	1054	*	3017098	3206043
230	37106	680	wifi	1119128	1156913	238	7907945	820	IDLE1	3206044	11114808
237	9871615	1464	IDLE0	1156914	11029992	240	167096	905	tiT	11114809	**11282809**
239	87309	680	wifi	11029993	11117981	242	4762	752	t_trace1	-	-
241	164965	1276	IDLE0	11117982	**11284222**						
#: índice do passo *: ev_net_mbedtls_											

Os números de ciclos utilizados por determinada marca apresentam-se destacados na tabela 5.3, nos passos 241 e 240. Para tal exemplo, verifica-se uma diferença de 1413 ciclos, ou cerca de 0,0126% do total de ciclos computados em cada CPU. Tais valores são computados individualmente para cada marca. A aferição de tais valores localmente, em referência a cada marca, nos permite evitar a propagação, para as marcas subsequentes, da discrepância existente nos valores encontrados para cada CPU.

5.6.2 Processamento útil

Anteriormente descrevemos como atrasos na rede e perdas de pacotes podem influenciar os resultados obtidos nas medições e propusemos a identificação de tais interferências mediante a atribuição de marcas específicas para tais casos.

Outro tipo de atraso que estamos interessados referem-se aqueles introduzidos mediante o emprego de temporizadores. Um exemplo é o controle exercido pela

interface de rede, a fim de que a transmissão ocorra à taxa específica. Outro exemplo, é o controle de congestionamento utilizado pelo TCP.

O uso de temporizadores sem espera ocupada acaba por deixar a CPU ociosa, caso não haja nenhum outro processo apto a executar. Ao observarmos a tabela 5.3, verfica-se que a média de ciclos utilizados pela marca (coluna fim) foi de 11.283.515 ciclos. Ao descartarmos os ciclos ociosos e os das tarefas t_trace, verifica-se que a CPU 0 utilizou 1.080.750 ciclos, enquanto que a 1 utilizou 3.374.044, o que representa, respectivamente, um uso de 9,6% e 29,9% para cada CPU.

5.6.3 Processamento útil na presença de paralelismo

Para o caso de experimentos onde o dispositivo utilize mais de uma CPU, desejamos quantificar o processamento paralelo. Uma vez disponíveis os intervalos de execução, conforme exemplificamos na tabela 5.3, apresentaremos a estratégia utilizada na determinação da quantidade de ciclos utilizados na presença de paralelismo. Devido ao alto número de dados envolvidos, tornou-se necessário uma implementação computacionalmente eficiente que será aqui descrita.

Inicialmente, excluímos da lista de intervalos as informações referentes aos processos IDLE e t_trace e ordenamos tal lista pelo ciclo de início, independente do processador utilizado. Uma lista exemplo juntamente com colunas auxiliares à descrição dos passos é apresentada na tabela 5.4.

O objetivo da implementação é passarmos pela lista ordenada acumulando, a cada linha percorrida, o intervalo com ocorrência de processamento que devemos permanecer analisando nas linhas subsequentes. A cada nova linha a ser analisada acumula-se eventual processamento paralelo (coluna ciclos em paralelo) e realiza-se o ajuste do intervalo a ser futuramente verificado. Como exemplo, na linha 2 o processamento da referida tarefa, executada na CPU 0, ocorreu entre os ciclos 1 e 8142, enquanto que o intervalo com ocorrência de processamento a ser avaliado era (1 ; 271.362). Neste momento determinou-se que 8.411 ciclos executaram dentro deste intervalo (em paralelo). O intervalo a ser avaliado passou então a ser (8.412 ; 271.362). É importante observar que o ajuste do intervalo deve considerar a possibilidade de que uma tarefa anteriormente iniciada pode ser concluída posteriormente à finalização de uma tarefa subsequente como verifica-se na tarefa da linha 4, iniciada

Tabela 5.4: Processamento por CPU referente ao Handshake, após o envio do primeiro pacote (G7 da tabela 5.2). As coluna inicio e fim indicam o intervalo de execução de cada processo, em cada CPU

			ciclos			ciclo		**intervalo com ocorrênca de processamento**		
Linha	#	cpu	execução	escal.	tarefa	incial	final	início	fim	**ciclos em paralelo**
1	225	1	270505	857	tiT	1	271.362	1	271.362	0
2	223	0	7307	1105	*	1	8.412	8.412	271.362	8.411
3	226	0	823939	703	wifi	58.492	883.133	271.362	883.133	212.870
4	231	1	958198	1143	*	271.363	1.230.703	883.133	1.230.703	611.770
5	228	0	121241	680	wifi	918.969	1.040.889	1.040.889	1.230.703	121.920
6	230	0	37106	680	wifi	1.119.128	1.156.913	1.156.913	1.230.703	37.785
7	232	1	1156487	651	*	1.230.704	2.387.841	1.230.704	2.387.841	0
8	233	1	91258	666	tiT	2.387.842	2.479.765	2.387.842	2.479.765	0
9	234	1	500790	1054	*	2.479.766	2.981.609	2.479.766	2.981.609	0
10	235	1	34822	666	tiT	2.981.610	3.017.097	2.981.610	3.017.097	0
11	236	1	187892	1054	*	3.017.098	3.206.043	3.017.098	3.206.043	0
12	239	0	87309	680	wifi	11.029.993	11.117.981	11.029.993	11.117.981	0
13	240	1	167096	905	tiT	11.114.809	11.282.809	11.117.981	11.282.809	3.172

#: índice do passo
*: ev_net_mbedtls_: tarefa avaliada

antes da de linha 5, mas finalizada posteriormente a esta.

Em relação à computação útil da tabela 5.4, verifica-se, mediante soma das respectivas células da tabela, que a CPU 0 utilizou 1.080.750 ciclos, enquanto que a 1 utilizou 3.374.044, dos quais 995.928 ciclos foram utilizados em paralelo.

5.6.4 Determinação do valor de quantum adequado

A importância de que cada tarefa seja escalonada em conformidade com o valor do quantum estabelecido em sistemas em tempo real é fundamental para garantir o cumprimento das propriedades de tempo real do sistema. O quantum é o intervalo de tempo máximo permitido para uma tarefa ser executada antes que ela seja interrompida e outra tarefa comece a ser executada.

Se uma tarefa em um sistema em tempo real for executada por mais tempo do que o quantum estabelecido, outras tarefas podem ser afetadas. Isso pode levar a atrasos na execução de outras tarefas, resultando em falhas no cumprimento dos prazos de execução.

A fim de verificar se tal propriedade é atendida e qual o valor mínimo que poderia ser atribuído ao quantum, analisamos os traços produzidos e verificamos o tempo máximo utilizado por cada tarefa, a cada vez que é posta a executar.

A determinação deste valor mínimo de quantum é de interesse não só para verificação do atendimento ao quantum establecido mas também para que se realize avaliação de novo valor mais adequado à aplicação em execução, visando diminuir processamento relativo à tentativas de escalonamento que por ventura não sejam necessárias.

5.7 Resumo do método apresentado

Embora o método apresentado tenha sido discutido com base no ambiente de desenvolvimento utilizado, sua implementação em outros ambientes é de interesse. Concluída a apresentação do método utilizado, podemos resumir suas características da seguinte forma:

1. Determinação de uma metodolgia adequada à quantificação do processamento local, considerando a necessidade de se quantificar o *overhead* introduzido.

2. Metodologia baseada na coleta, durante o escalonamento, de traços de execução.

3. Enfoque na minimização do processamento extra introduzido, evitando-se a utilização, dentre outros, de alocações dinâmicas em memória ou qualquer comunicação (serial, rede,etc.) durante a avaliação.

4. Identificação de uma forma de mensuração dos ciclos não utilizados.

5. Capacidade de criação das metricas refinadas aplicáveis a classes de problemas, para que, mediante execução regular do código, o usuário possa obter maior nível de visualização (granularidade) das métrica obtidas.

6. Introdução de um mecanismo de sincronização entre processadores a fim de que o processamento seja corretamente distribuido entre as marcas (métricas refinadas).

7. Determinação da estratégia para introdução de métricas refinadas no momento em que haja comunicação em rede.

8. Análise aplicada a um problema: processamento relacionado à espera por comunicação de dados na rede para posterior descarte e produção de resultados não influenciados pelo estado desta.

No presente trabalho as etapas acima foram implementadas no ambiente utilizado entretanto, com as devidas considerações, nada impede que sejam portadas a outros sistemas. Em especial, para outras diversas plataformas que se utilizem o FreeRTOS ou a biblioteca LWIP, estima-se que o método apresentado poderá ser implementado sem maiores modificações.

Capítulo 6

Avaliação Experimental

6.1 Introdução

O objetivo deste capítulo é o de apresentar os resultados obtidos, mediante o emprego do método proposto, para a avaliação da comunicação de dados em (i) um canal com segurança fim a fim, mediante o uso do TLS 1.2, e em (ii) um canal sem segurança fim a fim implementada à nivel da aplicação.

Além de apresentar os resultados, também se deseja apresentar a utilização do método proposto na busca por possíveis otimizações. Neste sentido, procurou-se explorar cenários reais de emprego do método para entendimento mais profundo acerca de resultados encontrados ao longo da experimentação.

6.1.1 Confiança dos resultados apresentados

Os resultados apresentados foram obtidos com base em um número de repetições de determinado experimento. Suas execuções foram tratadas como variáveis aleatórias independentes com distribuição normal. Os resultados apresentados são médias, $\bar{x}$, com um intervalo de confiança (IC) de 95% ($z = 1{,}96$), de uma experimento com n amostras, onde o valor de cada experimento é x_i.

$$IC = \bar{x} \pm z \cdot \frac{\sigma}{\sqrt{n}} \qquad \sigma = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n}}$$

Para facilitarmos a apresentação dos resultados, utilizaremos a seguinte notação:

$$\Delta_{IC} = z \cdot \frac{\sigma}{\sqrt{n}} \qquad\qquad \%\Delta_{IC} = \frac{\Delta_{IC}}{\bar{x}}$$

Desta forma, ao apresentarmos a média, $\bar{x}$, o valor expresso por $\mathbf{\Delta}_{IC}$ representa o "raio" do intervalo de confiança. Já $\%\mathbf{\Delta}_{IC}$ indica o quanto, percentualmente, este raio representa em relação ao valor médio encontrado. Quanto menor $\%\mathbf{\Delta}_{IC}$, maior a confiança de que a repetição de um experimento obterá valor próximo ao médio $\bar{x}$.

6.1.2 Quantificação da influência do estado da rede nos resultados

Anteriormente argumentamos que eventuais atrasos na comunicação influenciam os resultados apresentados. Como forma de verificarmos tal influência, apresentamos na tabela 6.1 os resultados obtidos quando a análise é feita sem o descarte de nenhum trecho do traço. Nesta tabela apresentamos os 15 primeiros trechos de traços vinculados à marcas com maior intervalo de confiança referentes à avaliação do processamento local realizado mediante a utilização do protocolo TLS 1.2.

O percentual apresentado na última coluna da tabela 6.1, $\%\mathbf{\Delta}_{IC}$, refer-se à marca estipulada pelo usuário final (marcas nível 1), sem levarmos em consideração as marcas internas adicionados automaticamente (nível 2). Para estas a tabela nos mostra, na linha 1, que o resultado obtido para a fase de Conexão, quando não descartamos nenhuma marca interna, possui um $\mathbf{\Delta}_{IC}$ ("raio de confiança") que equivale a 73,56% do valor médio dos ciclos utilizados na fase de conexão. O total de ciclos apresentado refere-se a todas as tarefas, ou seja, ciclos úteis e ociosos.

A fim de aprofundarmos nos resultados apresentados, observamos as marcas internas que compõem os três primeiros resultados da tabela 6.1. As correspondentes marcas internas de maior valor são apresentados na tabela 6.2.

Observa-se que o alto $\%\mathbf{\Delta}_{IC}$ ocorre em marcas internas onde estamos aguardando o recebimento de dados ou de estabelecimento de conexão (chegada de pacote contendo SYN) e, portanto, sujeitos à influência do estado da rede. No método prosto nesta tese, a correta avaliação das marcas internas que estão sujeitas à influência

#1	MHz	#2	Marca	#3	Conjunto Criptográfico	Média de ciclos ($/10^3$)	Δ_{IC} ($/10^3$)	$\%\Delta_{IC}$
200	160	1	Conexão	256	ECDSA,AES-128-GCM-SHA256	2.198,40	1.617,25	73,56
200	160	2	Recebimento	112	ECDSA,AES-256-GCM-SHA384	2.287,38	636,33	27,82
200	240	1	Conexão	128	RSA,AES-128-GCM-SHA256	2.291,11	561,56	24,51
200	240	1	Recebimento	128	ECDSA,AES-128-CCM-8	2.994,13	648,60	21,66
200	240	2	Recebimento	128	RSA,AES-128-GCM-SHA256	2.892,18	604,44	20,90
200	80	1	Recebimento	112	ECDSA,AES-256-GCM-SHA384	1.119,15	215,04	19,21
200	240	1	Recebimento	256	ECDSA,AES-128-GCM-SHA256	3.402,39	566,27	16,64
200	160	1	Recebimento	128	RSA,AES-128-GCM-SHA256	1.974,61	312,87	15,84
200	240	2	Recebimento	112	ECDSA,AES-256-GCM-SHA384	2.951,16	455,69	15,44
200	240	1	Recebimento	112	ECDSA,AES-256-GCM-SHA384	2.794,86	365,58	13,08
200	80	2	Recebimento	128	RSA,AES-128-GCM-SHA256	1.143,41	142,33	12,45
200	160	1	Recebimento	192	ECDSA,AES-128-CCM	1.874,97	228,89	12,21
200	240	1	Recebimento	128	RSA,AES-128-GCM-SHA256	2.689,87	324,52	12,06
200	160	2	Recebimento	128	RSA,AES-128-GCM-SHA256	1.916,14	225,05	11,74
200	80	1	Recebimento	192	ECDSA,AES-128-CCM	1.057,84	111,61	10,55

#1: Número de repetições (amostras), #2: Número de CPUS, #3: Dureza da autenticação

Tabela 6.1: Avaliação do intervalo de confiança de trechos de traços vinculados à marcas de experimentos. Sem descarte de trechos de traços vinculados à marcas influenciadas pela rede.

#1	MHz	#2	Marca Interna (Nível 2)	#3	Conjunto Criptográfico	Total de de ciclos ($/10^3$)	Δ_{IC} ($/10^3$)	$\%\Delta_{IC}$
200	160	1	Conexão Após envio do SYN	256	ECDSA,AES-128-GCM-SHA256	1.802,14	1.617,09	89,73
200	160	2	Recebimento Após bloqueio aguardando dados	112	ECDSA,AES-256-GCM-SHA384	1.858,36	635,74	34,21
200	240	1	Conexão Após envio do SYN	128	RSA,AES-128-GCM-SHA256	1.755,54	561,64	31,99

#1: amostras, #2: cpus, #3: dureza da autenticação

Tabela 6.2: Avaliação do intervalo de confiança em relação ao total de ciclos associados trechos de traços vinculados à marcas internas.

do estado rede deve ser realizada, para que não influenciem os resultados obtidos. A fim de concluírmos a apresentação a influência do estado da rede nos resultados obtidos, apresentamos, na tabela 6.3, os resultados encontrados através do método proposto e mediante o descarte de tais marcas internas.

#1	MHz	#2	Marca	#3	Conjunto Criptográfico	Total de ciclos ($/10^3$)	Δ_{IC} ($/10^3$)	$\%\Delta_{IC}$
200	240	2	Recebimento	128	ECDSA, AES-128-CCM-8	516,32	13,09	2,54
200	160	2	Recebimento	112	RSA, AES-256-GCM-SHA384	445,06	10,73	2,41
...	...	...	...	...	...	...	...	...
200	160	2	Recebimento	112	ECDSA, AES-256-GCM-SHA384	429,02	8,23	1,92
...	...	...	...	...	...	...	...	...
200	240	2	Conexão	128	RSA, AES-128-GCM-SHA256	276,03	4,21	1,52
...	...	...	...	...	...	...	...	...
200	**160**	1	**Conexão**	256	ECDSA, AES-128-GCM-SHA256	396,26	4,87	**1,23**
...	...	...	...	...	...	...	...	...
200	240	1	Handshake	256	ECDSA, AES-128-GCM-SHA256	496.477,36	1.310,04	0,26

#1: amostras, #2: cpus, #3: dureza da autenticação

Tabela 6.3: Avaliação do intervalo de confiança em relação ao total de ciclos de trechos de traços vinculados à marcas de experimentos. Após descarte de trechos de traços vinculados à marcas internas que sofrem influência da rede.

A tabela apresenta os valores ordenados pelo intervalo de confiança (valores intermediários foram suprimidos para melhor exibição). Observa-se que o $\%\mathbf{\Delta}_{IC}$, passou a variar entre 2,54% e 0,26%. Ao compararmos o primeiro marca da tabela sem descarte (6.1), verifica-se que o $\%\mathbf{\Delta}_{IC}$ referente à marca Conexão caiu de 73,56% para 1,23%, no caso onde se utilizou 1 processador e frequência de 160MHz.

Sendo assim, observa-se que se não removermos o processamento referente à marcas onde esperamos por comunicação, introduzimos grande imprecisão às métricas referentes ao processamento local demandado. Tal observação corrobora o fato de que tais marcas nos permitem identificar, no traço, trechos de execução influenciados pelo estado da rede.

6.2 Avaliações realizadas utilizando o método proposto

Passamos agora a apresentar os experimentos realizados e os resultados obtidos mediante o emprego do método proposto. O primeiro experimento, avaliará a comunicação segura usando TLS 1.2. O segundo, avaliará a comunicação sem segurança

fim a fim, utilizando-se para isso de uma conexão socket.

6.2.1 Comunicação segura com TLS

O objetivo desse experimento é avaliarmos, no ambiente utilizado, o processamento local realizado para alguns dos conjuntos criptográficos disponíveis. Foram selecionados aqueles cujos conjuntos criptográficos são suportados na versão 1.3 do TLS.

A experimentação se dá mediante a criação de uma tarefa que será avaliada pela ferramenta desenvolvida. A tarefa foi criada utilizando-se da macro anteriormente apresentada 5.1.

Inicialmente, identificamos as marcas e marcas internas criadas e aqueles que descartaremos. O resultado de tal identificação encontra-se na tabela 6.2.

Marca	**Nível 1**	**Nível 2**	**REDE**	**Resumo do processamento**
G1	Conexão	Inicialização		Inicialização
G2	Conexão	T_O_SYN-000	SIM	Após envio de solicitação de abertura da porta (TCP SYN) Aguardando conexão.
G3	Conexão	T_I_SYN-000		Após Comunicação TCP estabelecida
G4	Handshake	Inicialização		Preparando envio de TLS ClientHello
G5	Handshake	T_O_DATA-000	SIM	Após envio de ClientHello Aguardando dados.
G6	Handshake	T_I_DATA-000		Após receber TLS ServerHello, Certificate,ServerKeyExchange, ServerHelloDone Processamento associado à troca de chaves e autenticação
G7	Handshake	T_O_DATA-001		Após enviar ClientExchange.
G8	Handshake	T_O_DATA-002	SIM	Após enviar ClientFinish. Aguardando ServerFinished.
G9	Handshake	T_I_DATA-001		Após receber ServerFinished
G10	Envio	Inicialização		TLS Record Protocol: Cifragem da requisição
G11	Envio	T_O_DATA-003		Processamento após envio requisição
G12	Receb.	Inicialização	SIM	Aplicação solicita leitura da resposta. Bloqueia execução. Aguardando dados.
G13	Receb.	T_I_DATA-002		TLS Record Protocol: Decifragem da resposta a requisição

Tabela 6.4: Determinação do número de pacotes referentes ao exemplo (figura 6.2). A coluna Rede indica que processamento desta marca está associado à espera por dados da rede.

Para chegarmos à tabela 6.2, a transmissão foi analisada mediante a execução do experimento e o uso de uma ferramenta de análise de pacotes (TCPDUMP [123]) executada no servidor. Visualmente, inspecionou-se, com esta ferramenta, os pacotes

trocados, de forma a se determinar quantos seriam a cada fase. Através da inspeção, pudemos verificar um caso onde não ocorra retransmissão e determinar os tipos e quantidade de pacotes trocados.

Feita a identificação das marcas internas de descarte, apresentaremos os resultados mediante o descarte de tais marcas internas. Iniciaremos com a análise do Hanshake e demais marcas. A fase de conexão será apresentada juntamente com os resultados referentes ao estabelecimento de conexão sem segurança fim a fim.

Resultados da fase de Handshake

O objetivo do experimento foi o de avaliarmos a fase de Handshake utilizando um mesmo algoritmo de 128 bits para troca de chaves e variarmos o algoritmo de autenticação, a fim de avaliarmos o processamento destes. Os algoritmos de troca de chaves utilizados foram o Curve25519 [3] o P-256 [124] ambos com 128 bits de dureza. Ambos são suportados pela especificação do protocolo TLS 1.3 [63]. A seleção deveu-se ao fato de serem de domínio público e a possuírem dureza de 128 bits.

Dureza	**algoritmo[124]**
112	secp224r1
128	prime256v1
192	secp384r1
256	secp521r1

Tabela 6.6: Algoritmos baseados em curvas elípticas e utilizados na avaliação da fase de autenticação.

Os algoritmos de autenticação que usam curvas elípticas selecionados para avaliação, suportados e apresentados na especificação do TLS 1.3 [63], encontram-se na tabela 6.6. Em relação ao RSA, este foi avaliado com dureza 112 e 128 bits (tamanho de 3072 bytes). Acima destes valores de dureza, o certificado possuiria tamanho acima de 7680 bytes, o que não é suportado pelo dispositivo.

Durante o Handshake o dispositivo obtém o certificado correspondente a autenticação sendo avaliada (ECDSA ou RSA) e realiza a verificação de assinatura do mesmo. O certificado correspondente a cada método de autenticação foi assinado com um certificado de mesmo algoritmo e dureza 128 bits.

Os resultados apresentados para a fase de Handshake compreenderão, portanto, o processamento relativo à troca de chaves, verificação do certificado e autenticação.

Iniciamos a análise da fase Handshake, apresentando, na tabela 6.7 ,para o uso de curva Curve25519 [3]. Observa-se, para o caso em questão, que a ordem de grandeza do Handshake é mil vezes superior quando comparada ao processamento associado ao Recebimento. O tempo obtido, calculado dividindo-se a média total de ciclos pelo clock, é de cerca de 6,75 segundos, enquanto que as demais marcas possuem tempo inferiores a 7 mili segundos.

Marca	**Dureza**	**Criptografia (autenticação,troca de informações)**	**Total de ciclos** ($/\mathbf{10^3}$)	$\%\mathbf{\Delta}_{IC}$	**Tempo (ms)**
Conexão	256	ECDSA	134,68	0,91	1,68
Handshake	256	ECDSA	539.929,70	0,35	6.749,12
Requisição	256	ECDSA,AES-128-GCM-SHA256	178,86	1,44	2,24
Recebimento	256	ECDSA,AES-128-GCM-SHA256	487,27	1,31	6,09

Tabela 6.7: Avaliação do estabelecimento de sessão TLS por marca. 80MHz, 2 processadores. 200 amostras. Curva da troca de chaves: Curve25519 [3]

Na tabela 6.8, apresentamos o processamento referente à mesma fase do mesmo experimento, agora dividido por marcas internas que a compõe. Nesta tabela, observa-se que o maior processamento do Handshake se dá após o recebimento do ServerHello. Tal resultado era esperado, uma vez que nesta etapa o dispositivo deve realizar todas as operações necessárias, a fim de que possa enviar resposta com os parâmetros criptográficos necessários ao estabelecimento do canal seguro.

Uma vez determinado que a marca interna referente ao ServerHello possui processamento com ordem de grandeza superior a cem vezes em relação aos demais, passaremos a analisá-lo mediante variação do número de processadores e da dureza da criptografia utilizada. Os resultados são apresentados na tabela 6.9.

Marca interna (após)	**Dureza**	**Criptografia (autenticação)**	**Total de ciclos** ($/\mathbf{10^3}$)	$\%\mathbf{\Delta}_{IC}$	**Tempo (s)**
inicialização criado:ClientHello	256	ECDSA	229,05	0,50	0,00
receber: ServerHello	256	ECDSA	534.922,49	0,35	6,69
envio:ClientExchange	256	ECDSA	3.828,30	0,69	0,05
receber:ServerFinished	256	ECDSA	950,72	1,19	0,01

Tabela 6.8: Avaliação do estabelecimento de sessão TLS por marca (interna). 80MHz, 2 processadores. 200 amostras.

Verifica-se na última coluna da tabela 6.9, o percentual de acréscimo no tempo total de processamento mediante o uso de duas CPUs em relação ao uso de apenas uma. Em nenhum caso com 2 processadores o tempo total foi menor em relação

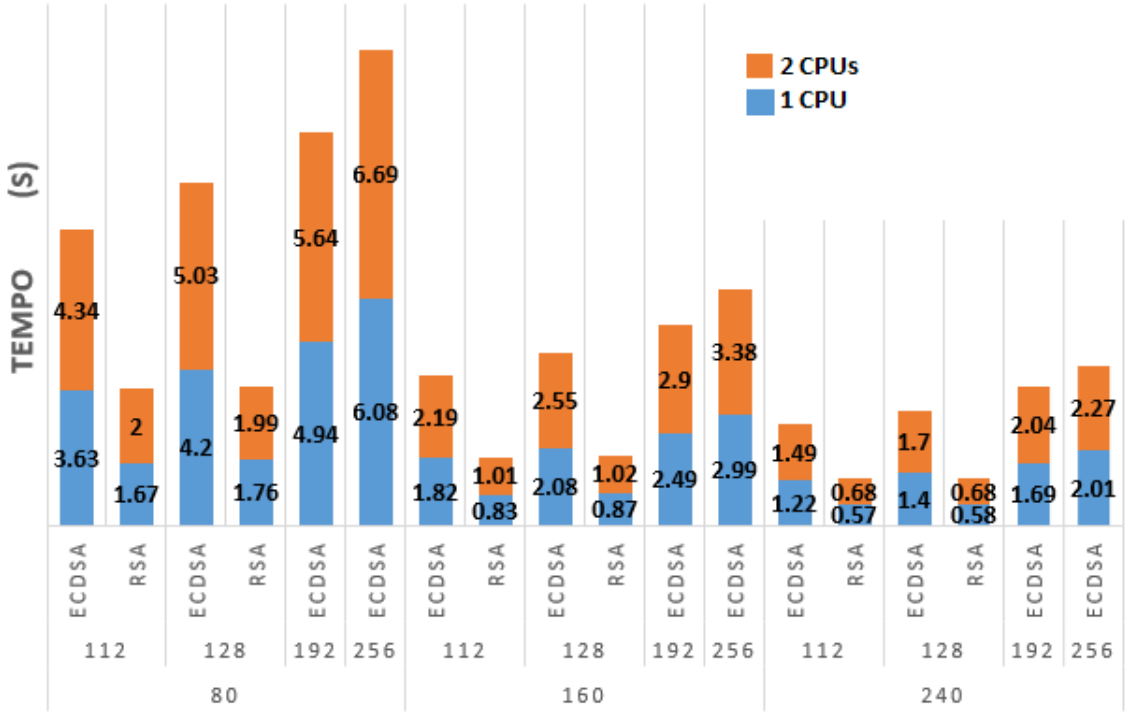


Figura 6.1: Avaliação do estabelecimento de sessão TLS. Após recebimento de ServerHello. 200 amostras. Curva da troca de chaves: Curve25519.

à configuração com 1 só CPU. Uma melhor caracterização de tais resultados será apresentada à frente, quando apresentarmos a métrica referente à avaliação de computação paralela.

Também observa-se que para os casos onde RSA e ECDSA são opções de autenticação com uma mesma dureza, o uso de RSA obteve melhor desempenho em todas as avaliações.

Observa-se que o aumento na frequência produziu-se um ganho proporcional. Como exemplo, ao operarmos a 80MHz com ECDSA obtemos o resultado de 6,69ms. Ao triplicarmos para 240MHz, o tempo caiu para aproximadamente 1/3 (2,27s a tabela). Discutiremos em outra seção essa linearidade e o motivo da maior investigação realizada.

Uma análise, utilizando-se agora de troca de chaves com a curva P-256, é apresentada em 6.10. Seu desempenho foi pior tanto para uma CPU quanto para duas de forma que apenas apresentamos os dados referentes ao último caso. O que se pode verificar então, é que em relação a troca de chaves usando 128 bits de dureza, o dispositivo apresenta melhores resultados usando a curva apresentada em 6.9.

Concluída a apresentação dos dados referentes ao estabelecimento de sessões,

			1 CPU			2 CPUS			
#1	**#2**	**MHz**	**Total de ciclos ($/10^3$)**	**$\%\Delta_{IC}$**	**Tempo (s)**	**Total de ciclos ($/10^3$)**	**$\%\Delta_{IC}$**	**Tempo (s)**	**#3**
112	ECDSA	80	290.772,76	0,40	**3,63**	347.016,46	0,50	**4,34**	**19,34**
112		160	291.240,37	0,32	**1,82**	350.287,95	0,42	**2,19**	**20,27**
112		240	292.736,44	0,28	**1,22**	356.581,99	0,49	**1,49**	**21,81**
112	RSA	80	133.816,28	0,58	**1,67**	159.716,08	0,42	**2,00**	**19,35**
112		160	133.192,77	0,46	**0,83**	162.353,23	0,42	**1,01**	**21,89**
112		240	135.656,39	0,37	**0,57**	163.739,08	0,48	**0,68**	**20,70**
128	ECDSA	80	335.863,70	0,37	**4,20**	402.264,33	0,50	**5,03**	**19,77**
128		160	332.603,91	0,33	**2,08**	407.792,84	0,56	**2,55**	**22,61**
128		240	335.231,22	0,29	**1,40**	408.224,44	0,48	**1,70**	**21,77**
128	RSA	80	140.505,14	0,56	**1,76**	159.515,28	0,60	**1,99**	**13,53**
128		160	139.460,12	0,48	**0,87**	162.753,47	0,54	**1,02**	**16,70**
128		240	138.915,62	0,48	**0,58**	162.113,21	0,53	**0,68**	**16,70**
192	ECDSA	80	395.120,04	0,39	**4,94**	450.877,04	0,43	**5,64**	**14,11**
192		160	397.611,03	0,32	**2,49**	464.457,98	0,46	**2,90**	**16,81**
192		240	405.492,11	0,29	**1,69**	489.444,76	0,37	**2,04**	**20,70**
256		80	486.334,77	0,33	**6,08**	534.922,49	0,35	**6,69**	**9,99**
256		160	478.264,92	0,31	**2,99**	541.512,34	0,39	**3,38**	**13,22**
256		240	483.097,17	0,27	**2,01**	543.712,81	0,41	**2,27**	**12,55**

#1: Dureza, #2: Criptografia(autenticação), #3: % de variação com 2 CPUs.

Tabela 6.9: Avaliação do estabelecimento de sessão TLS. Após recebimento de ServerHello. 200 amostras. Curva da troca de chaves: Curve25519 [3]

			2 CPUS		
#1	**#2**	**MHz**	**Total de ciclos ($/10^3$)**	**$\%\Delta_{IC}$**	**Tempo (s)**
112	ECDSA	80	392.038,98	0,35	**4,90**
112		160	443.461,58	0,34	**2,77**
112		240	488.013,00	0,35	**2,03**
112	RSA	80	198.374,51	0,39	**2,48**
112		160	243.311,39	0,41	**1,52**
112		240	286.928,08	0,29	**1,20**
128	ECDSA	80	484.326,41	0,41	**6,05**
128		160	565.110,44	0,33	**3,53**
128		240	657.758,12	0,32	**2,74**
128	RSA	80	199.237,21	0,53	**2,49**
128		160	242.758,40	0,42	**1,52**
128		240	284.882,12	0,45	**1,19**
192	ECDSA	80	489.097,83	0,32	**6,11**
192		160	538.498,69	0,38	**3,37**
192		240	592.707,42	0,30	**2,47**
256	ECDSA	80	588.958,59	0,37	**7,36**
256		160	639.128,60	0,32	**3,99**
256		240	693.630,68	0,30	**2,89**

#1: Dureza, #2: Criptografia(autenticação).

Tabela 6.10: Avaliação do estabelecimento de sessão TLS. Após recebimento de ServerHello. 200 amostras. Curva da troca de chaves: P-256. [3]

passamos a avaliar o reuso de sessões TLS. Neste caso, o Handshake utiliza um ticket referente à última sessão estabelecida e procura reutilizar a autenticação previamente

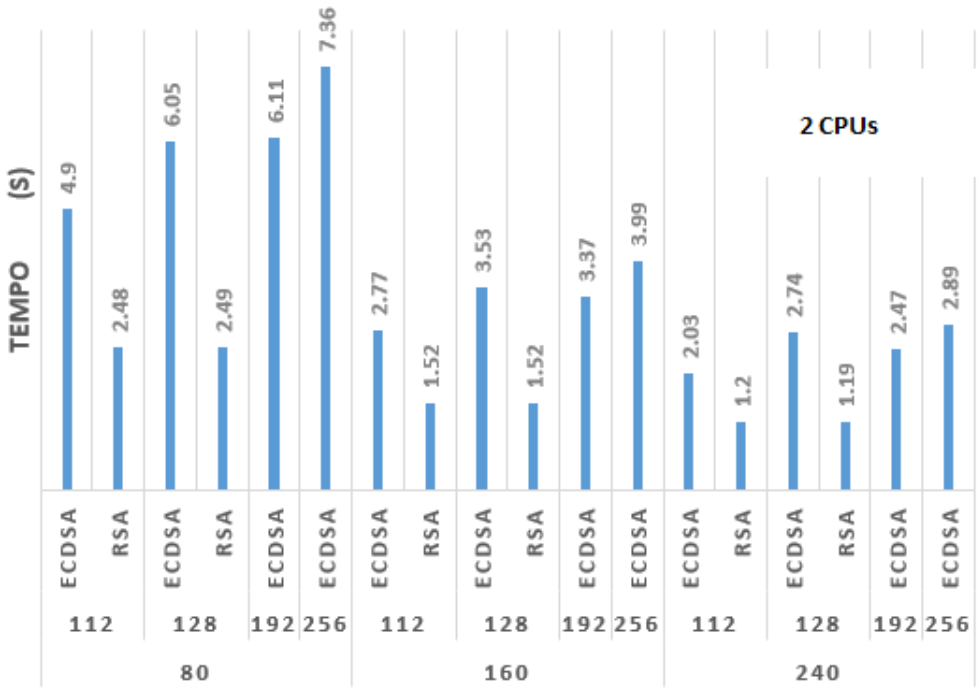


Figura 6.2: Avaliação do estabelecimento de sessão TLS. Curva da troca de chaves: P-256.

realizada. Caso o servidor considere o ticket válido segundo seus critérios (como por exemplo dentro de um limite de tempo máximo), nova autenticação não se faz necessária.

Na tabela 6.11, apresentamos os valores referentes ao caso com maior processamento. Os demais possuem valores dentro do intervalo de confiança apresentado (todos inferiores a 0,2s), de forma que optamos por suprimi-los.

			1 CPU			2 CPUS			
#1	**#2**	**MHz**	**Total de ciclos ($/10^3$)**	**%Δ_{IC}**	**Tempo (s)**	**Total de ciclos ($/10^3$)**	**%Δ_{IC}**	**Tempo (s)**	**#3**
256	ECDSA	80	1.320,79	0,69	0,02	1.412,92	0,52	0,02	6,97
256	ECDSA	160	1.629,43	0,66	0,01	1.790,02	0,42	0,01	9,86
256	ECDSA	240	1.999,76	0,74	0,01	2.132,37	0,58	0,01	6,63

#1: Dureza, #2: Criptografia(autenticação), #3 % de variação com 2 CPUs.

Tabela 6.11: Processamento após recebimento de ServerHello mediante reestabelecimento de sessão TLS. 300 amostras.

Resultados da fase de requisição de dados

A requisição realizada pelo dispositivo ao servidor possuí tamanho total de 200 bytes. Tal tamanho foi arbitrado como suficiente para que o dispositivo envie um sensoriamento realizado ou uma requisição por eventuais atuações que deva realizar. Os valores se apresentam na tabela 6.13. O objetivo é observamos o comportamento do processamento referente a cada conjunto criptográfico avaliado no que diz respeito à requisição por dados, após a fase de Handshake. Neste momento passa-se a utilizar criptografia simétrica.

		1 CPU		**2 CPUS**	
MHz	**AES**	**ms**	$\%\Delta_{IC}$	**ms**	$\%\Delta_{IC}$
80	128-CCM	**1,41**	2,62	**1,53**	1,49
160		**0,93**	2,68	**0,99**	1,81
240		**0,78**	3,03	**0,81**	1,97
80	128-CCM-8	**1,39**	2,28	**1,54**	1,75
160		**0,93**	2,78	**0,98**	1,65
240		**0,78**	3,13	**0,81**	2,21
80	128-GCM-SHA256	**1,36**	1,53	**1,39**	1,00
160		**0,93**	1,89	**0,93**	1,48
240		**0,78**	1,61	**0,77**	1,80
80	256-GCM-SHA384	**1,37**	1,28	**1,41**	0,87
160		**0,94**	1,80	**0,94**	1,28
240		**0,78**	1,89	**0,77**	1,62

Tabela 6.13: Transmissão em canal seguro. Tamanho do pacote 200 bytes. 300 amostras para cada configuração

Observa-se que o aumento no clock não produz um ganho proporcional. Como exemplo, ao operarmos a 80MHz com 256-GCM-SHA384 obtemos 5,9ms para transmitir 1200 bytes. entretanto, ao dobrarmos a frequência o tempo não caiu pela metade. Em relação à utilização de duas CPUS, não verificou-se diminuição do tempo. Já em relação à dureza, verifica-se que o uso de criptografia simétrica mais segura acabou por produzir resultados similares aos de menor dureza, justificando-se, portanto, o uso do conjunto que utilize criptografia simétrica mais segura.

Resultados da fase de recebimento de dados

Observa-se que o uso da criptografia simétrica mais forte produziu melhores resultados ou resultados dentro do intervalo de confiança quando comparados a suites mais fracos.

Observa-se que o aumento no clock não produz um ganho proporcional. Como exemplo, ao operarmos a 80MHz com 256-GCM-SHA384, obtemos 5,9ms para trans-

		1 CPU						2 CPUS					
		Bytes						Bytes					
		400		800		1200		400		800		1200	
MHz	**AES**	**ms**	#	**ms**	#	**ms**	#	**ms**	#	**ms**	#	**ms**	#
80	128-CCM	**4,14**	1,95	**5,43**	3,66	**6,72**	3,26	**4,44**	1,73	**5,96**	3,38	**7,23**	2,72
160		**2,61**	1,27	**3,04**	1,88	**3,55**	1,91	**2,70**	1,38	**3,39**	1,65	**3,99**	0,98
240		**2,17**	1,40	**2,45**	2,01	**2,77**	1,68	**2,18**	1,54	**2,64**	1,34	**3,07**	1,68
80	128-CCM-8	**4,08**	1,82	**5,11**	2,97	**6,48**	3,58	**4,36**	1,58	**5,93**	3,36	**7,27**	3,00
160		**2,63**	1,51	**3,06**	2,18	**3,56**	1,84	**2,69**	1,18	**3,37**	1,50	**4,04**	2,04
240		**2,17**	1,40	**2,40**	0,71	**2,73**	1,19	**2,18**	1,70	**2,64**	1,47	**3,06**	1,24
80	128-GCM-SHA256	**4,71**	1,87	**5,15**	2,57	**6,35**	2,46	**4,90**	1,77	**5,55**	2,15	**6,60**	2,00
160		**3,11**	2,04	**3,00**	1,22	**3,47**	1,47	**3,18**	1,89	**3,28**	1,15	**3,81**	1,07
240		**2,64**	2,20	**2,40**	0,93	**2,71**	1,09	**2,63**	2,29	**2,64**	1,61	**2,97**	1,29
80	256-GCM-SHA384	**4,11**	1,28	**4,82**	0,88	**5,90**	1,28	**4,36**	1,19	**5,50**	1,80	**6,51**	1,39
160		**2,66**	0,87	**3,11**	1,92	**3,58**	1,65	**2,75**	1,05	**3,31**	0,98	**3,94**	1,41
240		**2,23**	1,04	**2,43**	1,39	**2,78**	1,43	**2,26**	1,13	**2,67**	1,02	**3,07**	1,72

#:%$\mathbf{\Delta}_{IC}$

Tabela 6.14: Recebimento em canal seguro. 300 amostras para cada configuração.

mitir 1200 bytes. entretanto ao dobrarmos a frequência o tempo (5,5ms) não caiu pela metade. Conclui-se que não houve aproveitamento representativo da outra CPU existente, indicando que a fase de recebimento, para o experimento avaliado, não é implementada de forma a se utilizar de mais de uma CPU.

Já em relação à dureza, verifica-se que o uso de conjunto criptográfico mais seguro acabou por produzir resultados melhores ou dentro do intervalo de confiança encontrado quando comparado aos de menor dureza, justificando-se, portanto, o uso do conjunto mais seguro.

Linearidade do processamento na fase Handshake

Ao iniciarmos a avaliação do Handshake desejávamos verificar, através de perturbações no processamento, eventual alteração dos resultados obtidos.

Intuitivamente, acreditávamos que interrompendo a execução da tarefa mais vezes poderíamos, por exemplo, causar um atraso na comunicação entre o processador e o acelerador criptográfico. Após a algumas tentativas de diminuição do quantum ou introdução de uma tarefa que causasse perturbações, os valores obtidos se mostravam sempre próximos aos da tabela 6.9 para este experimento. Nesta verificou-se que ao triplicarmos de 80Mhz para 240MHz, o tempo caiu para aproximadamente 1/3 (2,27s a tabela). Tal comportamento se manteve independente das perturbações realizadas.

A fim de identificarmos o motivo, passamos então a investigar eventual ocorrên-

cia de quebra de respeito ao valor de quantum atribuído ao sistema. Por padrão esse valor é de 10ms para o sistema FreeRTOS e também para a versão portada para o ESP32. Para isso, avaliamos a ocorrência de utilização de fatia de tempo superior ao quantum na fase da marca interna de maior processamento (após receber ServerHello).

Na tabela 6.15, verifica-se a quantidade média de vezes e o valor máximo de ciclos que chegaram a ser utilizados por estas para os casos de 80MHz e 240Mhz. Tais valores superam os da configuração FreeRTOS/ESP32 padrão de 10ms e correspondem à 800.000 ou 2.400.000 ciclos para as frequências de 80 ou 240 Mhz respectivamente. A média, dentre os ciclos que ultrapassaram o quantum se mostrou próxima ao valor máximo apresentado (diferença inferior a 5%). Para demais níveis de segurança e frequência de 160MHz os valores ficaram entre os valores apresentados e optamos por não apresentá-los visto que não atrapalham as conclusões.

Mediante inspeção dos traços individuais de cada repetição, o que se verificou foi que a tarefa que utilizava tais ciclos era a referente ao experimento (ev_net_-mbedtls) e que havendo 2 CPUs, o outro processador não deixava de escalonar suas tarefas dentro do quantum estabelecido.

Posteriormente a esta detecção, as avaliações foram realizadas para um quantum de 100ms, visto que o processamento superava o quantum em 10 vezes. Como consequência, verificou-se que o processamento máximo durante a operação de Handshake passou a ser efetuado respeitando-se o quantum somente quando este possuir tempo superior a 100ms.

MHz	**cpus**	**Handshake**	**Dureza**	**#1**	**%Δ_{IC}**	**Máximo de ciclos**
80	1	ECDSA	256	113,88	0,81	7.671.788
80	2		256	54,26	3,44	7.668.224
240	1		256	39,43	1,76	23.074.924
240	2		256	23,05	4,67	23.065.512
80	1	RSA	112	32,16	1,91	7.755.725
80	2		112	14,20	7,48	7.667.297
240	1		112	12,32	2,16	23.025.050
240	2		112	7,35	6,99	23.061.037

#1: Número médio que vezes que ultrapassa o quantum.

Tabela 6.15: Ocorrências de processamento realizado em tempo superior ao quantum padrão do sistema (10ms). Após receber ServerHello.

6.2.2 Comunicação sem criptografia fim a fim

Neste experimento avaliamos o processamento associado à comunicação em um cenário onde o estabelecimento de comunicação segura entre origem e destino não realize na camada de aplicação. Este cenário pode ser descrito como aquele em que há garantia de comunicação segura entre o dispositivo IoT e o nó seguinte, como é o caso do ambiente wifi.

O código utilizado para avaliação deste cenário é apresentado em 6.1. Trata-se de uma conexão estabelecida mediante o uso da biblioteca sockets. Apresentamos o trecho deste código por ser curto e ilustrar a compreensão da implementação da avaliação.

Listing 6.1: Código: Comunicação sem criptografia fim a fim

```
1     ev_sock = socket(ev_http_res->ai_family,
2         ev_http_res->ai_socktype, 0);
3
4     task_trace_new_stats("CONNECT_START",0,0,1,0);
5     if(connect(ev_sock,ev_http_res->ai_addr,
6                 ev_http_res->ai_addrlen) != 0)    {
7                 evltr_last_error = errno;
8     }
9     task_trace_new_stats("REQ_DATA1_START",0,0,1,0);
10    if (write(ev_sock, url, url_len) < 0) evltr_last_error = 88;
11    task_trace_new_stats("RECV_DATA1_START",0,0,1,0);
12    int len = recv(ev_sock, rx_buffer, sizeof(rx_buffer) - 1, 0);
13    task_trace_new_stats("RECV_DATA1_END",0,0,1,0);
14    // ultimo par metro : finaliza coleta
15    task_trace_new_stats("CLOSE_START",0,0,0,1);
16    // valida contadores para descarte se houver retransmissao
17     if( LWIP_HOOKEVAL_GETLASTMEASURE_IDX(SEND_SYN)!=0
18         || LWIP_HOOKEVAL_GETLASTMEASURE_IDX(SEND_FIN)!=0
19         || LWIP_HOOKEVAL_GETLASTMEASURE_IDX(SEND_DATA)!=0
20         || LWIP_HOOKEVAL_GETLASTMEASURE_IDX(RECEIVE_SYN)!=0
21         || LWIP_HOOKEVAL_GETLASTMEASURE_IDX(RECEIVE_DATA)!=0
22    )
23    {
24        evltr_last_error=1;
25    }
```

As marcas introduzidas no código avaliado (6.1) juntamente com os marcas internas geradas em decorrência dos pacotes TCP transmitidos produzem a tabela de marcas exibida em 6.16. A coluna rede com valor 'SIM' indica que o processamento associado a tais marcas estará ocorrendo, enquanto o dispositivo aguarda envio ou recebimento de pacotes TCP e será, portanto, descartado.

Resultados da fase de requisição de dados

Esta avaliação se deu de forma similar à realizada quando avaliamos a comunicação segura e estão na tabela 6.17. A requisição realizada pelo dispositivo ao servidor possui tamanho total de 200bytes. Os resultados encontrados apresentaram um

	Nível 1	Nível 2	REDE	Descrição do processamento
G1	CONNECT_START	-		Início das medições
G2	CONNECT_START	T_O_SYN-000	SIM	Após envio de solicitação de abertura da porta (TCP SYN) Aguardando conexão.
G3	CONNECT_START	T_I_SYN-000		Após Comunicação TCP estabelecida
G4	REQ_DATA1_START	-		Início da solicitação
G5	REQ_DATA1_START	T_O_DATA-000		Após envio de requisição por dados rede.
G6	RECV_DATA1_START	-	SIM	Aplicação solicita dados Aguardando dados
G7	RECV_DATA1_START	T_I_DATA-000		Após receber dados.
G8	RECV_DATA1_END	-		Indica final de coleta

Tabela 6.16: Marcas introduzidas e número de pacotes para o experimento sem criptografia fim a fim.

	1 CPU		2 CPUS	
MHz	**ms**	#	**ms**	#
80	**0,67**	1,52	**0,47**	1,32
160	**0,48**	1,77	**0,39**	1,61
240	**0,41**	2,05	**0,35**	1,42

#:%$\mathbf{\Delta}_{IC}$

Tabela 6.17: Processamento referente a requisição de dados. Tamanho do pacote 200 bytes.

pequeno ganho quando se utiliza duas CPUS. Este melhor desempenho deve-se ao processamento referente à transferência dos dados da requisição entre a tarefa avaliada e as tarefas wifi e tiT. Uma vez que este processamento de transferência ocorre antes do envio do pacote à rede e o tempo total trechos de traços vinculados a essa marca é pequeno, o paralelismo desta transferência acaba por gerar um ganho no caso de 2 CPUs.

Resultados da fase de recebimento de dados

A avaliação do recebimento de dados se deu de forma similar à realizada anteriormente quando avaliamos a comunicação segura. Os resultados encontrados estão na tabela 6.18.

Verifica-se que o caso com 1 CPU obteve melhor desempenho e que o envio de pacotes maiores possui desempenho similar aos de menor tamanho.

	1 CPU						2 CPUS					
	Bytes						Bytes					
	400		800		1200		400		800		1200	
MHz	**ms**	#	**ms**	#	**ms**	#	**ms**	#	**ms**	#	**ms**	#
80	**0,72**	1,32	**0,71**	1,65	**0,75**	1,24	**0,79**	1,13	**0,80**	1,40	**0,80**	1,19
160	**0,47**	1,42	**0,51**	1,53	**0,48**	1,47	**0,64**	1,50	**0,62**	0,94	**0,62**	1,13
240	**0,42**	1,68	**0,44**	1,91	**0,42**	1,67	**0,55**	1,43	**0,57**	1,92	**0,56**	0,81

#:%$\mathbf{\Delta}_{IC}$

Tabela 6.18: Recebimento de dados. Experimento sem criptografia fim a fim.

6.2.3 Computação disponível e paralela

Aqui passamos a analisar a utilização das CPUs e iniciamos verificando a disponibilidade de processamento. Para isso, computamos, para cada marca, o número total de ciclos. No caso de experimentos com duas CPUs, o numero total de ciclos é total de ciclos de uma das CPUs apenas. Também computamos o número de ciclos úteis como sendo os utilizados por todas as tarefas exceto as IDLE e t_trace. Por fim, quando utilizou-se dois processadores, computamos o número de ciclos utilizados em paralelo, conforme descrito em 5.4. Definimos então Disponibilidade como sendo:

$$Disponibilidade = (Ciclos_{Uteis} - Ciclos_{UteisEmParalelo})/Total_{Ciclos}$$

Para o caso em que se utilizou somente um processador o percentual de disponibilidade foi de zero para ambos os experimentos, exceto para um único marca interna. Para o caso em que se utilizou dois processadores, o percentual de disponibilidade, para todos os experimentos, ficou entre 50% e 52% exceto, também, para um única marca interna. Em relação à da tabela 6.9 apresentada anteriormente, onde verificou-se que o desempenho com 1 CPU sempre superou o com 2, a indicação de que ouve disponibilidade superior a 50% nos indica que, para o experimento, não ouve aproveitamento representativo da segunda CPU. Neste caso, o escalonamento realizado e as trocas de contexto introduzidas consumiram processamento útil, justificando assim o melhor desempenho para o caso de 1 CPU.

À exceção desta marca interna, que abordaremos a seguir, verifica-se que no caso de uma CPU, esta permaneceu totalmente utilizada, ao passo que para o caso de 2 CPUs, a ociosidade foi alta e tal fato será abordado à frente.

A marca interna que, independente do número de processadores, apresentou

valores diferentes dos acima em relação à disponibilidade, foi o referente ao processamento realizado após o envio da mensagem ClientExchange. Tal marca contém processamento até que ClientFinish seja enviado. Seus resultados são apresentados na tabela 6.19.

			1 CPU		2 CPUS		
			ms	**#1**	**ms**	**#1**	**#2**
112	ECDSA	80	48,00	31,62	48,88	69,87	16,14
112		160	47,52	56,98	47,58	80,89	20,11
112		240	48,16	65,78	47,61	84,76	23,42
112	RSA	80	48,16	30,89	47,70	68,62	14,38
112		160	47,11	56,32	47,09	80,59	19,38
112		240	47,40	65,09	47,60	84,73	23,01
128	ECDSA	80	48,60	60,34	47,78	84,60	28,14
128		160	47,63	71,44	47,05	89,36	32,52
128		240	47,40	74,96	47,42	90,97	35,65
128	RSA	80	48,25	58,71	47,78	84,10	27,77
128		160	47,94	70,22	47,31	89,05	32,05
128		240	47,57	74,21	47,04	90,39	34,15
192	ECDSA	80	48,83	60,28	47,64	84,67	27,75
192		160	47,38	71,00	47,54	89,36	33,19
192		240	48,08	75,31	47,78	90,99	35,23
256		80	48,64	59,27	47,86	84,36	27,74
256		160	47,99	70,38	47,06	89,06	32,23
256		240	48,04	74,69	47,98	90,62	34,60

#1: % de disponibilidade em relação ao tempo médio
#2: % de paralelismo relativo ao processamento útil

Tabela 6.19: Disponibilidade de CPU e percentual de paralelismo. Marca interna referente ao processamento após o envio da mensagem ClientExchange. Intervalo de confiança, relativo em relação à média, abaixo de 3% para todos os resultados.

O que se verifica para a marca interna em questão é que seu tempo foi de aproximadamente 48ms para todos os casos. Tal valor indica que após realizar o envio de ClientExchange e antes do envio do ClientFinish houve alguma espera introduzida entre estas marcas. Como não houve outro pacote TCP enviado (do contrário, outra marca seria criado), concluímos que possivelmente este tempo foi deliberadamente introduzido. Portanto, para o caso de 1 CPU, este seria a única marca onde algum outro processamento realizado em paralelo à comunicação não acabaria por disputar o uso da CPU durante a realização desta (ou é claro nas marcas internas descartadas, onde estamos aguardando transmissão).

Já mediante o uso de 2 CPUS, embora o percentual de disponibilidade seja alto, torna-se de interesse verificarmos o caso, onde todas as CPUs estavam ocupadas, visto que não se poderá utilizar nenhuma CPU a menos que alguma outra tarefa deixe de ser executada.

O paralelismo foi verificado, utilizando-se de todas as CPUS existentes no dispositivo (2), com base na avaliação de ciclos em paralelo descrita em 5.4. Tal

O percentual de computação paralela foi então calculado em relação ao processamento útil e seus resultados foram verificados.

$$CompParalela = Ciclos_{UteisEmParalelo}/Ciclos_{Uteis}$$

Para o caso de comunicação segura verificou-se um percentual de paralelismo, para a requisição, abaixo de 3%. Para a fase de Handshake, após receber ServerHello (de maior processamento) este valor é abaixo de 1%. Para o recebimento com clock de 80Mhz, o percentual ficou entre 4,15% e 4,53% independente do tamanho do pacote recebido. Para 160 e 240Mhz, o recebimento ficou entre 2,18% e 2,85%.

Para o caso de comunicação sem segurança fim a fim, verificou-se um percentual de paralelismo, entre 12,78% e 13,91% para o recebimento com clock de 80MHz e entre 8,17% e 9,88% para 160 e 240Mhz.

Para as demais marcas internas, os resultados do percentual de paralelismo não se mostraram conclusivos.

Os percentuais de disponibilidade e paralelismo encontrados poderiam ser explorados na eventualidade de uma aplicação desejar utilizar-se do processamento disponível e serviram de base para apresentar de que forma a metodologia apresentada poderia contribuir na investigação de seus valores.

6.2.4 Processamento introduzido pelo método de avaliação

Aqui avalaiamos o processamento extra introduzido pelo método proposto.

Conforme descrito anteriormente, as tarefas t_trace são criadas durante a inicialização do método, antes do início da gravação do traço.

Após o início da gravação, as tarefas t_trace são ativadas ao se criar uma marca, de forma que o restante do traço passe a estar vinculado à marca criada. Aqui avaliamos a execução de t_trace e o resultado é apresentado na tabela 6.20.

amostras	**Ciclos**	**%Δ_{IC}**	Ciclos (máximo)
200.000	2.298	5,01	3.753

Tabela 6.20: Avaliação do processamento introduzido pelas tarefas t_trace.

Outro processamento introduzido pelo método refere-se à chamada realizada para criação de marcas. Cada chamada é responsável por enviar mensagens a cada t_trace introduzindo um processamento extra associado ao respectivo processo que a chama. Afim de avaliarmos tal processamento, foi criada uma tarefa responsável pela criação de 3000 marcas. Essa tarefa foi avaliada e o processamento associado a criação de uma marca encontra-se na tabela 6.21.

cpus	**amostras**	**Ciclos**	**%Δ_{IC}**	**Ciclos (máximo)**
1	200	1.525	3,01	1.957
2	200	1.837	2,42	2.127

Tabela 6.21: Avaliação do processamento referente à criação de novas marcas.

6.2.5 Resultados da fase de Conexão

Por fim, apresentamos o processamento associado ao estabelecimento de conexão. Uma vez que tal fase reflete o estabelecimento de conexão TCP tanto para o caso com comunicação segura fim a fim quanto para o caso sem segurança fim a fim, optamos por apresentá-los em conjunto. Tais medições refletem o processamento executado pela biblioteca sockets e pela biblioteca tls antes de efetuarem o envio do pacote TCP contendo SYN.

Canal seguro	**clock**	**ms**	**%Δ_{IC}**	**ms**	**%Δ_{IC}**
Não	80	**2,19**	1,22	**0,60**	2,12
	160	**1,55**	1,46	**0,48**	2,53
	240	**1,40**	1,61	**0,43**	2,47
SIM	80	**3,24**	0,78	**1,66**	1,02
	160	**2,49**	0,79	**1,27**	1,03
	240	**2,24**	0,80	**1,14**	0,90

Tabela 6.22: Processamento referente ao estabelecimento de conexão.

Verifica-se aqui o processamento extra inerente

6.2.6 Uso de memória

A fim de armazenarmos o traço, tornou-se necessário um dimensionamento correto do uso de memória, do contrário, os resultados teriam que ser persistidos durante o experimento, causando processamento não existente que comprometesse os resultados.

Inicialmente, alocou-se um vetor com o número máximo de escalonamentos que um traço pode conter. Para cada escalonamento ocorrido armazenou-se o índice do processo (1 byte), o número de ciclos utilizados durante sua execução (4 bytes), o número de ciclos utilizados para seu escalonamento (4 bytes) e qual cpu utilizou (1 byte).

Para os experimentos apresentados, até 3.000 medições por traço foi suficiente.

Para cada marca criada armazenou-se um índice referente ao identificador do experimento junto ao banco de dados (4 bytes), um byte informando que se trata de uma marca do usuário ou marca interna, um ponteiro para seu nome (4 bytes), um ponteiro para a próxima marca da lista (4 bytes) e o passo que se inicia (índice do vetor de escalonamentos) com 4 bytes. O número máximo de marcas utilizados no experimento foi 6.

A persistência nos nossos experimentos se deu através do envio ao servidor e após a coleta do traço, sem que este processamento ocorresse durante a fase de gravação. Em outros cenários onde haja, por exemplo, menos disponibilidade de memória, uma outra estratégia pode ser adotada como a interrupção, persistência, retorno do índice do vetor de escalonamentos a zero e retomada da gravação. Neste sentido o processamento associado a tais operações se dá no contexto da tarefa t_trace, não sendo, portanto, atribuído a nenhum outro processo.

6.3 Discussão dos Resultados

6.3.1 Do método proposto

Um dos principais dificultadores em relação ao método proposto é a sua implementação. Muitos recursos existentes em outros ambientes de desenvolvimento para sistemas não embarcados, como os para computador de âmbito pessoal ou para sistemas Android, não fazem parte do ambiente destinado ao desenvolvimento IoT ou não nos permite atuar no Kernel. Como tal, neste cenário o desenvolvimento demanda tempo muito superior a outras plataformas e constantes compilações e avaliações diretas no dispositivo e não em simuladores.

Já em relação aos resultados produzidos, uma vez implementado o método se mostrou não só plenamente capaz de computar o processamento do problema estu-

dado, como também de avaliar questões que não seriam possíveis sem o detalhamento do traço de execução proposto por esta tese.

Na tentativa de avaliarmos o processamento dos aceleradores utilizados pelo sistema em chip único para realização das operações de criptografia, procuramos introduzir perturbações na transferência de dados entre o processador principal e tais aceleradores. Traços com operações a distintas frequências e com quantuns de diferentes valores foram obtidos e analisados. Tais perturbações continuaram a não afetar as conclusões acerca dos resultados de processamento. Mediante uma inspeção mais detalhada fornecida pelo método utilizado, pudemos determinar que, em relação ao uso de métodos mais seguros de criptografia, o sistema avaliado deixou de atender a propriedade de previsibilidade temporal exigida de sistemas em tempo real. Embora o sistema avaliado se proponha a operar com um quantum de 10ms, verificou-se, que a fim de realizar tais operações criptográficas, ele na realidade deixa de proceder, por diversas vezes, escalonamento por até 100ms. Em cenários onde se exija que outros processos rodem em tempo inferior a este, tal informação pode desqualificá-lo, uma vez que durante a comunicação tais processos não irão executar.

O método proposto também nos permitiu de maneira satisfatória quantificar a computação disponível durante as avaliações realizadas. Em relação a estes, foram apresentados os valores médios apresentados e seus respectivos intervalos de confiança. Verificou-se que o intervalo de confiança do método proposto ficou abaixo de 3% em relação ao valor médio, com 95% de confiança.

Já em relação ao percentual de paralelismo quando utilizado duas CPUs, este estudo precisa ser aprofundado. Variações no contador de ciclos de cada CPU, bem como variações no número de ciclos referentes às operações que determinam o momento em que cada processo foi posto em execução causam imprecisões. Neste sentido, a determinação de que dois ou mais processos realmente executaram no mesmo momento está sujeita a um descasamento que influencia os resultados. Tal descasamento sofre maior influência, a cada trecho do traço de execução, em função do número de escalonamentos realizados e do tempo que a tarefa executa.

6.3.2 Dos resultados encontrados

Em relação ao estabelecimento de canal de comunicação e transferência de informação através deste, pudemos quantificar cientificamente o processamento associado a tais operações.

Quanto ao estabelecimento de canal seguro realizado pela aplicação, avaliamos, com base em implementação do protocolo TLS 1.2 disponibilizado pelo fabricante, os conjuntos criptográficos que poderão ser utilizados em sua versão 1.3. Como resultado produzimos diversas avaliações referentes a troca de chaves, autenticação e troca de informações que servem de referência a outros estudos.

Em relação à fase de troca de chaves, avaliamos o uso de dois algoritmos que TLS 1.3 especifica que devem ser aceitos, o Curve25519 [3] e o P-256 [124], ambos com dureza de 128 bits.

Para a fase de autenticação avaliamos sete possibilidades. Cinco algorítimos baseados em curvas elípticas com dureza de 112 a 256 bits e o RSA com 112 e 128 bits.

Tomando como base a dureza computacional desejada após 2030, que é de 128 bits de segurança, analisaremos aqui parte dos resultados encontrados como forma de explicitarmos sua utilização na determinação de pontos de operação do dispositivo em questão.

O resultado para a dureza de 128 bits (tanto para a autenticação quanto para a troca de chaves) nos mostrou que a utilização do algoritmo Curve25519 quando se utiliza autenticação em conjunto com o de curvas elípticas possui melhor

Para o caso em que curvas elípticas (prime256v1) foi utilizado como autenticação, compararemos aqui o Curve25519 [3] e o P-256 [124]. Operando a 240MHz e com duas CPUS, reduz-se o processamento em mais de 60%, visto que os resultados foram 1,70s contra 2,74 do segundo. Como o primeiro se mostrou superior, observamos, então, seu comportamento quando se utiliza apenas 1 CPU. Verificou-se então que o melhor desempenho foi utilizando-se apenas uma, obtendo-se um ganho de cerca de 21% (1,7s com duas CPUS, contra 1,4s com uma só). Por fim, se for possível a escolha de qualquer um dos dois algoritmos e do número de CPUs, o ganho obtido mediante a avaliação foi de 95%. Para menores frequências este ganho se mostrou linear. Se operado a 80Mhz o dispositivo levará cerca de 4,2 segundos com a mesma escolha,

demandando, portanto, praticamente o mesmo número de ciclos de operação.

Passamos então a comparar a autenticação do RSA (128 bits) e o de curvas elípticas de igual dureza (prime256v1) mediante o emprego do Curve25519 [3] como mecanismo de troca de chaves. Também o desempenho ao se utilizar uma única CPU foi melhor. Para 240Mhz seu tempo foi de 0,58s e também se observou que operando a 80Mhz o tempo aumentou por volta de 3x.

Em um cenário de escolha, dentre todas as configurações possíveis com 128 bits de dureza, mediante o uso de Curve25519 para troca de chaves, RSA para autenticação e uma única CPU, obteve-se o melhor resultado de 0,58s. O ganho obtido, resultante da avaliação realizada, foi de cerca de 4,8x (4724%) frente ao tempo anterior apresentado de 2,74s na pior configuração possível, com duas CPUS.

O uso de sessões TLS também foi verificado como forma de se evitar o Handshake, embora este dependa de suporte de ambas as partes. Independente do conjunto criptográfico utilizado, os tempos são inferiores a 0,1s para o caso de operações à 240MHz e 0,2s para 80MHz. Verificou-se menor processamento quando operando com uma CPU e com 80Mhz.

Em relação a troca de dados, avaliamos quatro conjuntos baseados em cifragem com o emprego dos algoritmos AES de 128 e 256 bits. Observou-se melhor desempenho mediante utilização de apenas 1 CPU. Aqui o que se verificou é que o emprego de maior conjunto criptográfico, quando operado a frequência de 240MHz, o processamento é praticamente o mesmo, não sendo portanto, justificável o emprego de menor dureza com base em processamento.

Quanto à comunicação em rede sem o emprego de criptografia por parte da aplicação, como esperado, o processamento associado é muito menor. Verificou-se que referente ao recebimento de dados, utilizando-se criptografia simétrica, demandou cerca de 6x mais processamento para o caso de 240Mhz.

Capítulo 7

Trabalhos Relacionados

Neste capítulo iremos apresentar trabalhos relacionados à avaliação do processamento realizado por sensores IoT implementados em dispositivos que possuam arquitetura de sistema em um único chip. Em relação às visões de estudo apresentadas, os estudos que apresentaremos se enquadram como àqueles cuja abordagem é oriunda da chamada visão a partir da semântica, possuindo interseção com a visão orientada à redes. O método proposto nesa tese, cuja visão é orientada pela chamada visão a partir das coisas, visa contribuir no sentido de que futuras pesquisas possam se inserir na área central da interseção destas três visões, agregando, para tal, métricas mais apuradas em relação ao processamento local executado.

Em [125], os autores apresentam estudo a fim de identificar a existência de dispositivos capazes de atender a necessidade de IoT no que diz respeito à capacidade de processamento, baixo custo e realização de comunicação segura. São avaliados 3 dispositivos, entre eles, o ESP32, que, segundo suas conclusões, são capazes de atender, de maneira satisfatória, às necessidades de comunicação com tal característica. São realizadas avaliações utilizando e comparados algoritmos de curvas elípticas e o RSA. A avaliação se dá mediante repetidas operações de experimentos distintos de geração e verificação de assinaturas. Com base em seus resultados, os autores argumentam que a performance do RSA no ESP32 é superior ao ECC e atribuem tal resultado ao fato do dispositivo possuir um acelerador específico para realização de operações RSA. Embora a avaliação especificamente das funções de assinatura e verificação nos dê um bom entendimento da realização exclusiva dessas, a realização do handshake completo em cenário real envolve processamento do sistema opera-

cional e outros processamentos da biblioteca utilizada. O presente trabalho difere deste por apresentar resultados decorrentes de uma avaliação completa da utilização desses protocolos no estabelecimento de comunicação segura e não somente da avaliação de operações específicas.

Em [126], os autores avaliam a comunicação utilizando curvas elípticas e RSA mediante utilização do dispositivo ESP32. Os dispositivos são avaliados no contexto de arquitetura em névoa onde a comunicação ocorre em dois cenários. No primeiro, um dispositivo de maior poder computacional se conecta a um servidor implementado no ESP32. Neste caso o servidor provê o certificado e não necessita, portanto, fazer a validação deste. Para este caso, os resultados indicam, para 128 bits de segurança, 1,1 requisição por segundo para ECDSA (0,9s), contra 0,4 requisições/s usando RSA (2,5s). Baseados nisso apontam que ECDSA se mostrou mais adequado a tal cenário.

No segundo cenário, um dispositivo ESP32 se conecta a outro. Neste caso um deles atua como cliente, necessitando assim fazer a validação. Para este caso, os autores encontram 0,17 requisições/s (5,88s) para ECDSA e 0,13 requisições/s para RSA (7,7s). Os autores, então, argumentam que devido a atrasos de rede e processamento realizado pelos dispositivos os resultados foram afetados. Passam, então, a observar a troca de pacotes, com ferramenta de análise de troca de pacotes introduzida em outro dispositivo da rede, como método de estimação. Passam, então, a estimar o atraso de conexão (envio/recebimento do SYN) e o tempo do cliente para enviar ClientExchange após envio, pelo servidor, do ServerHelloDone. Em relação ao atraso do ClientExchange, este é justamente o atraso decorrente do processamento do Handshake que inicialmente os autores desejavam avaliar e acabaram avaliando externamente, sujeitos a influências de atraso na rede. Além de SYN e ClientExchange terem sido estimados externamente, o resultado foi apresentado como sendo constante quando podem variar consideravelmente a cada comunicação e impactar os resultados conforme apresentamos quando avaliamos a confiança dos nossos resultados. Por fim, para este experimento, os autores concluem que os resultados são insatisfatórios devido a atrasos na rede e que tanto RSA quando ECDSA apresentam resultados muito próximos, com ligeiro melhor desempenho de ECDSA.

O presente trabalho difere desse por considerar que o cenário onde o ESP32 atua

como servidor é muito mais limitado devido ao fato de ser muito mais complexo, com a atual estrutura da Internet, se criar todo um caminho para que um dispositivo da névoa inicie a conexão com o nó (dispositivo de borda). Já no segundo cenário, os autores recaem no problema investigado nesta teste: a obtenção de resultados comparáveis, sem a influência da transmissão em rede. Os autores concluem que, baseados em seus resultados, no dispositivo ESP32 ECC tem melhor performance que RSA. Tal conclusão difere das conclusões que obtivemos assim como as apresentadas anteriormente por [125].

Em [127], os autores avaliam a capacidade de utilização do dispositivo ESP32 com câmera acoplada em aprendizado de máquina. Para isso implementam no dispositivo o reconhecimento de escrita manual das letras "x" e "o"através de algoritmo de classificação baseado em regressão logística, cujos pesos necessários à avaliação são computados na nuvem e enviados ao dispositivo. O dispositivo de borda então fica responsável por realizar o processamento local referente à análise de imagens da câmera, evitando-se assim transferir as imagens para a nuvem. A avaliação do experimento se dá mediante cálculo do tempo utilizado para realização de cada avaliação em diversas resoluções de câmera. São apresentados os tempos de obtenção da imagem da câmera e processamento (redimensionamento e reconhecimento) utilizando-se dois tipos de memória RAM. Para um dos casos o valor obtido foi inferior a 1ms para obtenção da foto e 12ms para processamento. Esta tese complementa o trabalho de [127], uma vez que uma avaliação do escalonamento realizado, em especial, a variação do quantum padrão de 10ms, poderia retirar processamento referente a escalonamento desnecessário e dedicá-lo à resolução do problema.

Em [128], os autores avaliam a comunicação utilizando curvas elípticas e RSA mediante utilização do dispositivo Cypress CYW43907. Similarmente ao nosso trabalho, identificam a necessidade de obtenção de métricas não influenciadas pelo estado da rede. Para a obtenção dos dados os autores conseguem isolar momentos de comunicação com a rede através de acionamento de portas lógicas para interrupção da coleta que é realizada em dispositivo externo. São coletadas externamente informações do tempo de execução e consumo. Os autores concluem que no seu ambiente o processamento mediante o emprego de curvas elípticas demanda mais tempo e energia que os quando se utiliza RSA com o mesmo nível de dureza. Embora

os resultados sejam de interesse e não influenciados pelo estado na rede, o método proposto se difere do nosso, visto que o método proposto pelos autores limita a avaliação de processamento ao tempo de execução e não aponta o comportamento que levou a tal. Como exemplo, não seria possível a determinação de que o sistema não respeitou o quantum estabelecido ou a verificação do quanto de CPU esteve disponível.

Estudo	**Avaliação de processamento**	**Fonte dos resultados**	**Comunicação em rede**	**Considera influência do estado da rede**	**Apresenta métrica de confiança dos resultados**	**Fontes de imprecisão**	**Exemplo de contribuição mediante aplicação do método proposto nesta tese. Realizar avaliação do processamento:**
[125]	Conjuntos criptográficos do TLS em ESP32	Temporizador.	Sim	Não	Não	Estado da rede. Distribuição física dos equipamentos. Retransmissões. Processamento introduzido ao medir tempo.	- sem influência do estado da rede. - introduzido pelo método de avaliação.
[126]	Conjuntos criptográficos do TLS em ESP32	Temporizador.	Sim	Não	Não	Estado da rede. Distribuição física dos equipamentos. Retransmissões. Processamento introduzido ao medir tempo.	- sem influência do estado da rede. - introduzido pelo métdo de avaliação.
[127]	Aprendizado de máquina. Avaliação de reconhecimento de imagens da câmera em ESP32	Temporizador.	Não	N/A	Sim		- paralelelo ou com somente 1 CPU. - com diferentes valores de quantum, visando menor escalonamento. - da comunicação com a câmera, mediante introdução de métrica nível 2.
[128]	Processamento de conjuntos criptográficos do TLS em SoC Cypress CYW43907 (RTOS)	Tempo total. Instrumentação.	Sim	Sim	Sim	dos equipamentos.	- introduzido pelo métdo de avaliação. - individual da bilblioteca TLS, do SO e do TCP/IP.
ESTA TESE	Conjuntos criptográficos do TLS em ESP32	Traços de execução	Sim	Sim	Sim	NA	NA

Tabela 7.1: Estudos. Diferenças e contribuições desta tese a estes estudos.

Capítulo 8

Conclusões e Trabalhos Futuros

A presente tese apresenta um método para avaliação de desempenho baseada na análise de traços de execução realizada a partir da coleta de dados durante o processo de escalonamento realizado pelo sistema operacional.

Como forma de verificarmos a relevância do tema, realizou-se pesquisa na qual analisamos e apresentamos estudos que se realizaram no mesmo contexto que o desta tese. Apresentamos os modelos de nuvem e névoa e identificamos nestes onde este trabalho se insere dentro da grande área de estudos relacionados à comunicação de computadores em rede. Recorremos, então, mais uma vez à literatura especializada e trouxemos conceitos clássicos mais abrangentes relacionados à teoria de redes aplicados ao nosso ambiente de comunicação sem fio.

Em relação a sistemas operacionais, pesquisou-se o contexto do presente trabalho dentro da área de estudos relacionados a estes, em especial aqueles que operam de forma preemptiva e em tempo real. Apresentamos as limitações impostas a tal ambiente e a arquitetura de computadores de sistemas em um único chip.

Uma vez delimitado o escopo do presente trabalho dentro das grandes áreas de pesquisa em sistemas operacionais e comunicação de computadores, passamos então, a apresentar o método proposto à avaliação de desempenho. O método foi ,então, ajustado para a produção de avaliações que envolvam comunicação em rede sem fio. Neste contexto apresentamos como o estado da rede influencia os resultados obtidos.

A partir da aplicação do método proposto obtivemos resultados referentes à comunicação segura provida pela aplicação. Neste contexto quantificamos como o a influência do estado da rede nos resultados obtidos. Os resultados obtidos nos per-

mitiram não só a determinação dos processamentos referentes ao estabelecimento de comunicação, mas também uma análise do uso dos recursos existentes no dispositivo avaliado. No âmbito do processamento realizado para a comunicação verificou-se disponibilidade de processamento que possa ser utilizado, bem como determinou-se inconsistências do dispositivo em relação à obediência do quantum estabelecido às tarefas.

O método foi comparado com o empregado na literatura para o caso da análise de comunicação em rede. Verificou-se que a obtenção de resultados baseados em medições externas de tempo podem produzir resultados influenciados pelo estado da rede a ponto de gerarem resultados inconclusivos. Verificamos que quando os efeitos são devidamente tratados, outros aspectos importantes não são avaliados por tal abordagem, como por exemplo a utilização de recursos. Para outros casos onde se deseje analisar processamento em tal ambiente, também apresentamos na literatura estudos que poderiam se beneficiar do método de avaliação aqui proposto.

Ao término do desenvolvimento da metodologia proposta, surgiram pontos que merecem destaque e que aqui deixamos registrados. Entre estes:

- Novas versões dos sistema operacional utilizado foram disponibilizadas.
- Outra biblioteca de rede, nativa do FreeRTOS, foi disponibilizada.
- Outros dispositivos e outros sistemas operacionais estão sendo retratados na literatura.
- Dada a quantidade de dispositivos sendo introduzidos, a busca por reutilização de canais seguros, estabelecidos nas camadas física e de enlace, vem ganhando destaque e pode vir a trazer benefícios reais no sentido de tirar o processamento realizado à nível da aplicação.
- Constante identificação, na literatura recente, da necessidade de adoção de metodologia adequada à comparação dos protocolos de comunicação sendo propostos para IoT, em relação ao processamento local realizado pelos sensores.

Referências Bibliográficas

[1] FLYNN, M. J., LUK, W. *Computer System Design - System-on-Chip*. John Wiley & Sons, Inc., jul. 2011. doi: 10.1002/9781118009925. Disponível em: <https://doi.org/10.1002/9781118009925>.

[2] BELLEZA, R. R., PIGNATON, E. "Performance study of real-time operating systems for internet of things devices", *IET Software*, v. 12, n. 3, pp. 176–182, jun. 2018. doi: 10.1049/iet-sen.2017.0048. Disponível em: <https://doi.org/10.1049/iet-sen.2017.0048>.

[3] BERNSTEIN, D. J. "Curve25519: New Diffie-Hellman Speed Records". In: *Public Key Cryptography - PKC 2006*, Springer Berlin Heidelberg, pp. 207–228, 2006. doi: 10.1007/11745853_14. Disponível em: <https://doi.org/10.1007/11745853_14>.

[4] COMISSÃO EUROPÉIA. "Comissão Européia - Internet das Coisas". Disponível em: <https://digital-strategy.ec.europa.eu/en/policies/next-generation-internet-things>. Acesso em: 2023-01-15.

[5] FEDERAÇÃO DAS INDUSTRIAS DO ESTADO DE SÃO PAULO. "Internet das Coisas como alavanca de inovação". Disponível em: <http://www.fiesp.com.br/noticias/internet-das-coisas-como-alavanca-de-inovacao/>. Acesso em: 2018-10-15.

[6] BANCO NACIONAL DE DESENVOLVIMENTO. "Estudo "Internet das Coisas: um plano de ação para o Brasil"". Disponível em: <https://www.bndes.gov.br/wps/portal/site/home/conhecimento/pesquisaedados/estudos/estudo-internet-das-coisas-iot/estudo-internet-das-coisas-um-plano-de-acao-para-o-brasil>. Acesso em: 2023-01-15.

[7] GOVERNO BRASILEIRO. "Consulta Pública Plano Nacional de IoT". Disponível em: <http://www.participa.br/cpiot/itens-da-consulta>. Acesso em: 2023-01-15.

[8] ASHTON, K. “That "Internet of Things"thing”, *RFiD Journal*, 2009. Disponível em: <http://www.rfidjournal.com/articles/view?4986>. Acesso em: 2023-01-10.

[9] BUCKLEY, J. “FROM RFID TO THE INTERNET OF THINGS. Pervasive networked systems - Final Report”, *Conference organised by DG Information Society and Media - Networks and Communication Technologies Directorate*, 2006. Disponível em: <http://cordis.europa.eu/pub/ist/docs/ka4/au_conf670306_buckley_en.pdf>. Acesso em: 2017-10-10.

[10] WEISER, M., GOLD, R., BROWN, J. S. “The origins of ubiquitous computing research at PARC in the late 1980s”, *IBM Systems Journal*, v. 38, n. 4, pp. 693–696, 1999. doi: 10.1147/sj.384.0693. Disponível em: <https://doi.org/10.114/sj.384.0693>.

[11] WEISER, M. “The computer for the 21st century”, *ACM SIGMOBILE Mobile Computing and Communications Review*, v. 3, n. 3, pp. 3–11, jul 1999. doi: 10.1145/329124.329126. Disponível em: <https://doi.org/10.1145/329124.329126>.

[12] DAY, J., ZIMMERMANN, H. “The OSI reference model”, *Proceedings of the IEEE*, v. 71, n. 12, pp. 1334–1340, 1983. doi: 10.1109/proc.1983.12775. Disponível em: <https://doi.org/10.1109/proc.1983.12775>.

[13] ATZORI, L., IERA, A., MORABITO, G. “The Internet of Things: A survey”, *Computer Networks*, v. 54, n. 15, pp. 2787–2805, oct 2010. doi: 10.1016/j.comnet.2010.05.010. Disponível em: <https://doi.org/10.1016/j.comnet.2010.05.010>.

[14] Giusto, D., Iera, A., Morabito, G., et al. (Eds.). *The Internet of Things*. Springer New York, 2010. doi: 10.1007/978-1-4419-1674-7. Disponível em: <https://doi.org/10.1007/978-1-4419-1674-7>.

[15] MUNIR, S., STANKOVIC, J. A., LIANG, C.-J. M., et al. “Cyber Physical System Challenges for Human-in-the-Loop Control.” In: *Feedback Computing*, 2013.

[16] DOHR, A., MODRE-OPSRIAN, R., DROBICS, M., et al. “The Internet of Things for Ambient Assisted Living”. In: *2010 Seventh International Conference on Information Technology: New Generations*. IEEE, 2010. doi: 10.1109/itng.2010.104. Disponível em: <https://doi.org/10.1109/itng.2010.104>.

[17] LIU, Y., PENG, Y., WANG, B., et al. "Review on cyber-physical systems", *IEEE/CAA Journal of Automatica Sinica*, v. 4, n. 1, pp. 27–40, jan 2017. doi: 10.1109/jas.2017.7510349. Disponível em: <https://doi.org/10.1109/jas.2017.7510349>.

[18] LEE, E. A. "Cyber Physical Systems: Design Challenges". In: *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*. IEEE, may 2008. doi: 10.1109/isorc.2008.25. Disponível em: <https://doi.org/10.1109/isorc.2008.25>.

[19] ZHANG, B., MOR, N., KOLB, J., et al. "The Cloud is Not Enough: Saving IoT from the Cloud". In: *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, Santa Clara, CA, 2015. USENIX Association. Disponível em: <https://www.usenix.org/conference/hotcloud15/workshop-program/presentation/zhang>.

[20] NUNES, D. S. S., ZHANG, P., SILVA, J. S. "A Survey on Human-in-the-Loop Applications Towards an Internet of All", *IEEE Communications Surveys & Tutorials*, v. 17, n. 2, pp. 944–965, 2015. doi: 10.1109/comst.2015.2398816. Disponível em: <https://doi.org/10.1109/comst.2015.2398816>.

[21] SARKAR, S., CHATTERJEE, S., MISRA, S. "Assessment of the Suitability of Fog Computing in the Context of Internet of Things", *IEEE Transactions on Cloud Computing*, v. 6, n. 1, pp. 46–59, 2018. doi: 10.1109/TCC.2015.2485206.

[22] DASTJERDI, A., GUPTA, H., CALHEIROS, R., et al. "Fog Computing: principles, architectures, and applications". In: *Internet of Things*, Elsevier, pp. 61–75, 2016. doi: 10.1016/b978-0-12-805395-9.00004-6. Disponível em: <https://doi.org/10.1016/b978-0-12-805395-9.00004-6>.

[23] CONSORTIUM, O. "OpenFog Reference Architecture for Fog Computing". 2017. Disponível em: <https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf>.

[24] ZHANG, Q., CHENG, L., BOUTABA, R. "Cloud computing: state-of-the-art and research challenges", *Journal of Internet Services and Applications*, v. 1, n. 1, pp. 7–18, 2010. doi: 10.1007/s13174-010-0007-6. Disponível em: <https://doi.org/10.1007/s13174-010-0007-6>.

[25] BONOMI, F., MILITO, R., NATARAJAN, P., et al. "Fog Computing: A Platform for Internet of Things and Analytics". In: *Big Data and Internet*

of Things: A Roadmap for Smart Environments, Springer International Publishing, pp. 169–186, 2014. doi: 10.1007/978-3-319-05029-4_7. Disponível em: <https://doi.org/10.1007/978-3-319-05029-4_7>.

[26] TANENBAUM, A. S., WETHERALL, D. *Computer Networks*. Prentice Hall, 2011.

[27] KUROSE, J. F., ROSS, K. W. *Computer Networking: A Top-Down Approach*. Addison-Wesley Publishing Company, 2009.

[28] CONSORTIUM, I. I. "The Industrial Internet of Things Connectivity Framework". 2022. Disponível em: <https://www.iiconsortium.org/wp-content/uploads/sites/2/2022/06/IIoT-Connectivity-Framework-2022-06-08.pdf>.

[29] AKPAKWU, G. A., SILVA, B. J., HANCKE, G. P., et al. "A Survey on 5G Networks for the Internet of Things: Communication Technologies and Challenges", *IEEE Access*, v. 6, pp. 3619–3647, 2018. doi: 10.1109/ACCESS.2017.2779844.

[30] CHAUDHARI, B. S., ZENNARO, M., BORKAR, S. "LPWAN Technologies: Emerging Application Characteristics, Requirements, and Design Considerations", *Future Internet*, v. 12, n. 3, pp. 46, mar. 2020. doi: 10.3390/fi12030046. Disponível em: <https://doi.org/10.3390/fi12030046>.

[31] CHAKKOR, S., CHEIKH, E., MOSTAFA, B., et al. "Comparative Performance Analysis of Wireless Communication Protocols for Intelligent Sensors and Their Applications", *International Journal of Advanced Computer Science and Applications*, v. 5, 09 2014. doi: 10.14569/IJACSA.2014.050413.

[32] SODHRO, A. H., CHEN, L., SEKHARI, A., et al. "Energy efficiency comparison between data rate control and transmission power control algorithms for wireless body sensor networks", *International Journal of Distributed Sensor Networks*, v. 14, n. 1, jan. 2018. doi: 10.1177/1550147717750030. Disponível em: <https://hal.archives-ouvertes.fr/hal-01788366>.

[33] MEDEIROS, D. S. V., NETO, H. N. C., LOPEZ, M. A., et al. "A survey on data analysis on large-Scale wireless networks: online stream processing, trends, and challenges". v. 11. Springer Science and Business Media LLC, out. 2020. doi: 10.1186/s13174-020-00127-2.

[34] LIANG, C.-J. M., PRIYANTHA, N. B., LIU, J., et al. "Surviving Wi-Fi Interference in Low Power ZigBee Networks". In: *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, p. 309–322, New York, NY, USA, 2010. Association for Computing Machinery. ISBN: 9781450303446. doi: 10.1145/1869983.1870014. Disponível em: <https://doi.org/10.1145/1869983.1870014>.

[35] AHMED, N., SALIL, S. K., JHA, S. "Mitigating the effect of interference in Wireless Sensor Networks". In: *IEEE Local Computer Network Conference*, pp. 160–167, 2010. doi: 10.1109/LCN.2010.5735690.

[36] MENEZES, A. J., VANSTONE, S. A., OORSCHOT, P. C. V. *Handbook of Applied Cryptography*. 1st ed. USA, CRC Press, Inc., 1996. ISBN: 0849385237.

[37] KIM, C. H. "Differential Fault Analysis against AES-192 and AES-256 with Minimal Faults". In: *2010 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 3–9, 2010. doi: 10.1109/FDTC.2010.10.

[38] THEOBALD, T. "How to Break Shamir's Asymmetric Basis". In: *Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '95, p. 136–147, Berlin, Heidelberg, 1995. Springer-Verlag. ISBN: 3540602216.

[39] YANG, B., WU, K., KARRI, R. "Secure Scan: A Design-for-Test Architecture for Crypto Chips", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 25, n. 10, pp. 2287–2293, 2006. doi: 10.1109/TCAD.2005.862745.

[40] POLK, T., MCKAY, K., CHOKHANI, S. *Guidelines for the selection, configuration, and use of transport layer security (TLS) implementations*. Relatório técnico, abr. 2014. Disponível em: <https://doi.org/10.6028/nist.sp.800-52r1>.

[41] EISENBARTH, T., KUMAR, S., PAAR, C., et al. "A Survey of Lightweight-Cryptography Implementations", *IEEE Design Test of Computers*, v. 24, n. 6, pp. 522–533, 2007. doi: 10.1109/MDT.2007.178.

[42] DAEMEN, J., RIJMEN, V. "The Block Cipher Rijndael". In: *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 277–284, 2000. doi: 10.1007/10721064_26. Disponível em: <https://doi.org/10.1007/10721064_26>.

[43] ZOU, Y., ZHU, J., WANG, X., et al. "A Survey on Wireless Security: Technical Challenges, Recent Advances, and Future Trends", *Proceedings of the IEEE*, v. 104, n. 9, pp. 1727–1765, 2016. doi: 10.1109/JPROC.2016.2558521.

[44] LATVA-AHO, M., LEPPÄNEN, K., CLAZZER, F., et al. "Key drivers and research challenges for 6G ubiquitous wireless intelligence", 2020.

[45] WYNER, A. D. "The wire-tap channel", *The Bell System Technical Journal*, v. 54, n. 8, pp. 1355–1387, 1975. doi: 10.1002/j.1538-7305.1975.tb02040.x.

[46] LEUNG-YAN-CHEONG, S., HELLMAN, M. "The Gaussian wire-tap channel", *Information Theory, IEEE Transactions on*, v. 24, pp. 451 – 456, 08 1978. doi: 10.1109/TIT.1978.1055917.

[47] ZHANG, J., DUONG, T. Q., MARSHALL, A., et al. "Key Generation From Wireless Channels: A Review", *IEEE Access*, v. 4, pp. 614–626, 2016. doi: 10.1109/ACCESS.2016.2521718.

[48] HALAK, B., ZWOLINSKI, M., MISPAN, M. S. "Overview of PUF-based hardware security solutions for the internet of things". In: *2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 1–4, 2016. doi: 10.1109/MWSCAS.2016.7870046.

[49] HALAK, B., ZWOLINSKI, M., MISPAN, M. S. "Overview of PUF-based hardware security solutions for the internet of things". In: *2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 1–4, 2016. doi: 10.1109/MWSCAS.2016.7870046.

[50] GASSEND, B., CLARKE, D., VAN DIJK, M., et al. "Silicon Physical Random Functions". In: *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, p. 148–160, New York, NY, USA, 2002. Association for Computing Machinery. ISBN: 1581136129. doi: 10.1145/586110.586132. Disponível em: <`https://doi.org/10.1145/586110.586132`>.

[51] SOCIETY, I. C. "IEEE 802.11i-2004 - IEEE Standard for information technology-Telecommunications and information exchange between systems-Local and metropolitan area". . Disponível em: <`https://standards.ieee.org/standard/802_11i-2004.html`>. Acesso em: 2019-03-15.

[52] SHELDON, F. T., WEBER, J. M., YOO, S., et al. "The Insecurity of Wireless Networks", *IEEE Security Privacy*, v. 10, n. 4, pp. 54–61, 2012. doi: 10.1109/MSP.2012.60.

[53] RADIVILOVA, T., HASSAN, H. A. "Test for penetration in Wi-Fi network: Attacks on WPA2-PSK and WPA2-enterprise". In: *2017 International Conference on Information and Telecommunication Technologies and Radio Electronics (UkrMiCo)*, pp. 1–4, 2017. doi: 10.1109/UkrMiCo.2017.8095429.

[54] KUMAR, A., RASTOGI, R., SILBERSCHATZ, A., et al. "Algorithms for provisioning virtual private networks in the hose model", *IEEE/ACM Transactions on Networking*, v. 10, n. 4, pp. 565–578, 2002. doi: 10.1109/TNET.2002.802141.

[55] KOTULSKI, Z., NOWAK, T., SEPCZUK, M., et al. "On end-to-end approach for slice isolation in 5G networks. Fundamental challenges". In: *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pp. 783–792, 2017. doi: 10.15439/2017F228.

[56] HAGA, S., ESMAEILY, A., KRALEVSKA, K., et al. "5G Network Slice Isolation with WireGuard and Open Source MANO: A VPNaaS Proof-of-Concept". In: *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 181–187, 2020. doi: 10.1109/NFV-SDN50289.2020.9289900.

[57] "The Transport Layer Security (TLS) Protocol Version 1.2". Disponível em: <`https://tools.ietf.org/html/rfc5246`>.

[58] RESCORLA, E., MODADUGU, N. "Datagram Transport Layer Security Version 1.2". Disponível em: <`https://tools.ietf.org/html/rfc6347`>.

[59] NAIK, N. "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP". In: *2017 IEEE International Systems Engineering Symposium (ISSE)*, pp. 1–7, 2017. doi: 10.1109/SysEng.2017.8088251.

[60] PIERLEONI, P., CONCETTI, R., BELLI, A., et al. "Amazon, Google and Microsoft Solutions for IoT: Architectures and a Performance Comparison", *IEEE Access*, v. 8, pp. 5455–5470, 2020. doi: 10.1109/ACCESS.2019.2961511.

[61] GOOGLE. "Google Transparency Report". Disponível em: <https://transparencyreport.google.com/https/overview?hl=en>. Acesso em: 2023-01-05.

[62] HOLZ, R., HILLER, J., AMANN, J., et al. "Tracking the Deployment of TLS 1.3 on the Web: A Story of Experimentation and Centralization", *SIGCOMM Comput. Commun. Rev.*, v. 50, n. 3, pp. 3–15, jul. 2020. ISSN: 0146-4833. doi: 10.1145/3411740.3411742. Disponível em: <https://doi.org/10.1145/3411740.3411742>.

[63] RESCORLA, E. *The Transport Layer Security (TLS) Protocol Version 1.3*. Relatório técnico, ago. 2018. Disponível em: <https://doi.org/10.17487/rfc8446>.

[64] MCKAY, K. A., COOPER, D. A. *Guidelines for the selection, configuration, and use of Transport Layer Security (TLS) implementations*. N. NIST SP 800-52r2. Aug 2019. doi: 10.6028/NIST.SP.800-52r2. Disponível em: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf>.

[65] JOSEFSSON, S., NIR, Y. "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier". Disponível em: <https://tools.ietf.org/html/rfc8422>.

[66] KOBLITZ, N. "Elliptic curve cryptosystems", *Mathematics of Computation*, v. 48, n. 177, pp. 203–203, jan. 1987. doi: 10.1090/s0025-5718-1987-0866109-5. Disponível em: <https://doi.org/10.1090/s0025-5718-1987-0866109-5>.

[67] MILLER, V. S. "Use of Elliptic Curves in Cryptography". In: *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 417–426. doi: 10.1007/3-540-39799-x_31. Disponível em: <https://doi.org/10.1007/3-540-39799-x_31>.

[68] ZINZINDOHOUÉ, J. K., BARTZIA, E.-I., BHARGAVAN, K. "A Verified Extensible Library of Elliptic Curves". In: *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pp. 296–309, 2016. doi: 10.1109/CSF.2016.28.

[69] SIMION, E. "Entropy and Randomness: From Analogic to Quantum World", *IEEE Access*, v. 8, pp. 74553–74561, 2020. doi: 10.1109/ACCESS.2020.2988658.

[70] DONG, J., ZHENG, F., CHENG, J., et al. "Towards High-performance X25519/448 Key Agreement in General Purpose GPUs". In: *2018 IEEE Conference on Communications and Network Security (CNS)*, pp. 1–9, 2018. doi: 10.1109/CNS.2018.8433161.

[71] MCGREW, D. *An Interface and Algorithms for Authenticated Encryption.* Relatório técnico, jan. 2008. Disponível em: <https://doi.org/10.17487/rfc5116>.

[72] MCGREW, D., BAILEY, D. *AES-CCM Cipher Suites for Transport Layer Security (TLS).* Relatório técnico, jul. 2012. Disponível em: <https://doi.org/10.17487/rfc6655>.

[73] EASTLAKE, D., HANSEN, T. *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF).* Relatório técnico, maio 2011. Disponível em: <https://doi.org/10.17487/rfc6234>.

[74] GILLMOR, D. *Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS).* Relatório técnico, ago. 2016. Disponível em: <https://doi.org/10.17487/rfc7919>.

[75] RIVEST, R. L., SHAMIR, A., ADLEMAN, L. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", *Commun. ACM*, v. 21, n. 2, pp. 120–126, fev. 1978. ISSN: 0001-0782. doi: 10.1145/359340.359342. Disponível em: <https://doi.org/10.1145/359340.359342>.

[76] JOHNSON, D., MENEZES, A., VANSTONE, S. "The Elliptic Curve Digital Signature Algorithm (ECDSA)", *International Journal of Information Security*, v. 1, n. 1, pp. 36–63, ago. 2001. doi: 10.1007/s102070100002. Disponível em: <https://doi.org/10.1007/s102070100002>.

[77] BERNSTEIN, D. J., DUIF, N., LANGE, T., et al. "High-speed high-security signatures", *Journal of Cryptographic Engineering*, v. 2, n. 2, pp. 77–89, ago. 2012. doi: 10.1007/s13389-012-0027-1. Disponível em: <https://doi.org/10.1007/s13389-012-0027-1>.

[78] BARKER, E. *Recommendation for key management.* Relatório técnico, maio 2020. Disponível em: <https://doi.org/10.6028/nist.sp.800-57pt1r5>.

[79] HU, W., BULUSU, N., JHA, S. "A communication paradigm for hybrid sensor/actuator networks". In: *2004 IEEE 15th International Symposium on Personal, Indoor and Mobile Radio Communications (IEEE Cat.*

No.04TH8754), v. 1, pp. 201–205 Vol.1, Sept 2004. doi: 10.1109/PIMRC.2004.1370864.

[80] DELICATO, F. C., PIRES, P. F., BATISTA, T. "The Resource Management Challenge in IoT". In: *SpringerBriefs in Computer Science*, Springer International Publishing, pp. 7–18, 2017. doi: 10.1007/978-3-319-54247-8_2. Disponível em: <https://link.springer.com/chapter/10.1007/978-3-319-54247-8_2>.

[81] AL-FUQAHA, A., KHREISHAH, A., GUIZANI, M., et al. "Toward better horizontal integration among IoT services", *IEEE Communications Magazine*, v. 53, n. 9, pp. 72–79, September 2015. ISSN: 0163-6804. doi: 10.1109/MCOM.2015.7263375.

[82] ASHTON, K. "That "Internet of Things"thing", *RFiD Journal*, 2009. Disponível em: <http://www.rfidjournal.com/articles/view?4986>. Acesso em: 2022-10-10.

[83] POPA, L., GHODSI, A., STOICA, I. "HTTP as the narrow waist of the future internet". In: *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks - Hotnets '10*. ACM Press, 2010. doi: 10.1145/1868447.1868453. Disponível em: <https://doi.org/10.1145%2F1868447.1868453>.

[84] RASPBERRY PI FOUNDATION PI FOUNDATION. "Raspberry Pi". Disponível em: <https://www.raspberrypi.org/>. Acesso em: 2023-02-15.

[85] CORPORATION, I. "Intel® Galileo Board". Disponível em: <https://ark.intel.com/content/www/us/en/ark/products/78919/intel-galileo-board.html>. Acesso em: 2023-02-15.

[86] SALLES, R. C. *Avaliação de Capacidade e Consumo de Energia de Rede Móvel Ad Hoc Centrada em Interesse*. Tese de Mestrado, Universidade Federal do Rio de Janeiro, 2014.

[87] ALRIMEIH, H., RAKHMATOV, D. "Fast and Flexible Hardware Support for ECC Over Multiple Standard Prime Fields", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, v. 22, n. 12, pp. 2661–2674, 2014. doi: 10.1109/TVLSI.2013.2294649.

[88] DIGIKEY. "ESP32-WROOM-32E price". 2021. Disponível em: <https://www.digikey.com/en/products/detail/espressif-systems/ESP32-WROOM-32E-(4MB)/11613148>.

[89] BRANDENBURG, B. B. *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*. Tese de Doutorado, 2011.

[90] ALTMEYER, S., BURGUIÈRE, C. M. "Cache-related preemption delay via useful cache blocks: Survey and redefinition", *Journal of Systems Architecture*, v. 57, n. 7, pp. 707–719, ago. 2011. doi: 10.1016/j.sysarc.2010.08.006. Disponível em: <https://doi.org/10.1016/j.sysarc.2010.08.006>.

[91] INTEL CORPORATION. "Intel 64 and IA-32 Architectures Software Developer's Manual". Disponível em: <https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-instruction-set-reference-manual
pdf>. Acesso em: 2019-11-15.

[92] TENSILICA, I. "Xtensa Instruction Set Architecture (ISA) Reference Manual". Disponível em: <https://usermanual.wiki/Document/Xtensa2020ASSEMBLER20GUIDE.1231659642/help>. Acesso em: 2019-11-15.

[93] BURNS, A. "Scheduling hard real-time systems: a review", *Software Engineering Journal*, v. 6, n. 3, pp. 116–128, 1991. doi: 10.1049/sej.1991.0015.

[94] TINYOS. "TinyOS Home Page". Disponível em: <http://www.tinyos.net/>. Acesso em: 2023-01-15.

[95] CHANG, M., CROSBY, J., VINCENT, H. "Yottos Operating System Connecting Low-Power Devices with High-Level Programming". In: *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, SenSys '14, p. 336–337, New York, NY, USA, 2014. Association for Computing Machinery. ISBN: 9781450331432. doi: 10.1145/2668332.2668360. Disponível em: <https://doi.org/10.1145/2668332.2668360>.

[96] ANDROID. "Android Home Page". Disponível em: <https://www.android.com/>. Acesso em: 2023-01-15.

[97] ARM. "Documentation – Arm Developer". 2020. Disponível em: <https://developer.arm.com/documentation/ddi0337/e/memory-protection-unit/about-the-mpu>.

[98] FREERTOS. "RTOS Task Notifications". 2020. Disponível em: <https://www.freertos.org/RTOS-task-notifications.html>.

[99] ABELLA, J., HERNANDEZ, C., QUIÑONES, E., et al. "WCET analysis methods: Pitfalls and challenges on their trustworthiness". In: *10th IEEE*

International Symposium on Industrial Embedded Systems (SIES), pp. 1–10, 2015. doi: 10.1109/SIES.2015.7185039.

[100] GARCIA-MARTINEZ, A., CONDE, J. F., VINA, A. "A comprehensive approach in performance evaluation for modern real-time operating systems". In: *Proceedings of EUROMICRO 96. 22nd Euromicro Conference. Beyond 2000: Hardware and Software Design Strategies*, pp. 61–68, 1996. doi: 10.1109/EURMIC.1996.546366.

[101] VARGHESE, B., WANG, N., BERMBACH, D., et al. "A Survey on Edge Performance Benchmarking". v. 54, New York, NY, USA, apr 2021. Association for Computing Machinery. doi: 10.1145/3444692.

[102] SWAMY, S. N., KOTA, S. R. "An Empirical Study on System Level Aspects of Internet of Things (IoT)". v. 8, pp. 188082–188134, 2020. doi: 10.1109/ACCESS.2020.3029847.

[103] JAYAKUMAR, H., RAHA, A., KIM, Y., et al. "Energy-efficient system design for IoT devices". In: *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 298–301, 2016. doi: 10.1109/ASPDAC.2016.7428027.

[104] HALABI, T., BELLAICHE, M., FUNG, B. C. M. "Towards Adaptive Cybersecurity for Green IoT". In: *2022 IEEE International Conference on Internet of Things and Intelligence Systems (IoTaIS)*, pp. 64–69, 2022. doi: 10.1109/IoTaIS56727.2022.9975990.

[105] MARTINEZ, B., MONTÓN, M., VILAJOSANA, I., et al. "The Power of Models: Modeling Power Consumption for IoT Devices". v. 15, pp. 5777–5789, 2015. doi: 10.1109/JSEN.2015.2445094.

[106] MAIER, A., SHARP, A., VAGAPOV, Y. "Comparative analysis and practical implementation of the ESP32 microcontroller module for the internet of things". In: *2017 Internet Technologies and Applications (ITA)*, pp. 143–148, 2017. doi: 10.1109/ITECHA.2017.8101926.

[107] MEKKI, K., BAJIC, E., CHAXEL, F., et al. "Concept and Hardware Considerations for Product-Service System Achievement in Internet of Things". In: *2019 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS)*, pp. 1–6, 2019. doi: 10.1109/WITS.2019.8723755.

[108] IVKOVIĆ, J., IVKOVIĆ, J. "Analysis of the performance of the new generation of 32-bit Microcontrollers for IoT and Big Data Application". In: *ICIST 2017 - 7th International Conference on Information Society and Techology, Kopaonik, Serbia*, 03 2017.

[109] IVKOVIĆ, J. "Linpack Benchmark C Port". 2021. Disponível em: <https://os.mbed.com/users/JovanEps/code/Linpack/>.

[110] IVKOVIĆ, J. "DHRYSTONE Benchmark C Port". 2021. Disponível em: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/freertos-smp.html>.

[111] LTD., R. P. T. "Raspberry Pi 4 Model B". 2021. Disponível em: <https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711/rpi_DATA_2711_1p0_preliminary.pdf>.

[112] FREERTOS. "ESP32 Series Datasheet". 2021. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>.

[113] ESPRESSIF. "ESP32 WROOM 32". 2021. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf>.

[114] ESPRESSIF. "Espressif IoT Development Framework v4.1". 2021. Disponível em: <https://github.com/espressif/esp-idf/tree/release/v4.1>.

[115] ESPRESSIF. "ESP-IDF FreeRTOS SMP Changes". 2021. Disponível em: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/freertos-smp.html>.

[116] MICROSOFT. "Visual Studio Code". 2021. Disponível em: <https://code.visualstudio.com/>.

[117] PLATFORMIO. "PlatformIO". 2021. Disponível em: <https://platformio.org/>.

[118] DB, M. "Maria DB". 2021. Disponível em: <https://mariadb.org/>.

[119] CENTOS. "CentOS". 2021. Disponível em: <https://centos.org/>.

[120] FOUNDATION, A. "Apache". 2021. Disponível em: <https://apache.org/>.

[121] FOUNDATION, P. "PHP". 2021. Disponível em: <https://php.org/>.

[122] SOCIETY, I. C. "IEEE 802.11n-2009 - IEEE Standard for Information technology". . Disponível em: <https://standards.ieee.org/standard/802_11n-2009.html>. Acesso em: 2021-03-10.

[123] GROUP, T. T. "TCPDUMP/LIBCAP". 2021. Disponível em: <https://tcpdump.org/>.

[124] SOCIETY, T. I. "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)". 2006. Disponível em: <https://tools.ietf.org/html/rfc4492>.

[125] PEARSON, B., LUO, L., ZHANG, Y., et al. "On Misconception of Hardware and Cost in IoT Security and Privacy". In: *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pp. 1–7, 2019. doi: 10.1109/ICC.2019.8761062.

[126] SUÁREZ-ALBELA, M., FRAGA-LAMAS, P., FERNÁNDEZ-CARAMÉS, T. "A Practical Evaluation on RSA and ECC-Based Cipher Suites for IoT High-Security Energy-Efficient Fog and Mist Computing Devices", *Sensors*, v. 18, n. 11, pp. 3868, nov. 2018. doi: 10.3390/s18113868. Disponível em: <https://doi.org/10.3390/s18113868>.

[127] DOKIC, K. "Microcontrollers on the Edge – Is ESP32 with Camera Ready for Machine Learning?" In: *Image and Signal Processing*, pp. 213–220, Cham, 2020. Springer International Publishing. ISBN: 978-3-030-51935-3. Disponível em: <https://link.springer.com/chapter/10.1007/978-3-030-51935-3_23>.

[128] GEREZ, A. H., KAMARAJ, K., NOFAL, R., et al. "Energy and Processing Demand Analysis of TLS Protocol in Internet of Things Applications". In: *2018 IEEE International Workshop on Signal Processing Systems (SiPS)*, pp. 312–317, 2018. doi: 10.1109/SiPS.2018.8598334.

[129] YU, W., LIANG, F., HE, X., et al. "A Survey on the Edge Computing for the Internet of Things", *IEEE Access*, v. 6, pp. 6900–6919, 2018. doi: 10.1109/ACCESS.2017.2778504.

[130] MITEV, M., CHORTI, A., REED, M., et al. "Authenticated secret key generation in delay-constrained wireless systems", *EURASIP Journal on Wireless Communications and Networking*, v. 2020, n. 1, jun. 2020. doi: 10.1186/s13638-020-01742-0. Disponível em: <https://doi.org/10.1186/s13638-020-01742-0>.

[131] DELVAUX, J., PEETERS, R., GU, D., et al. "A Survey on Lightweight Entity Authentication with Strong PUFs", *ACM Comput. Surv.*, v. 48, n. 2, out. 2015. ISSN: 0360-0300. doi: 10.1145/2818186. Disponível em: <https://doi.org/10.1145/2818186>.

[132] INFO, C. S. "Cipher Suite Info". 2020. Disponível em: <https://ciphersuite.info/cs/?tls=all&singlepage=true>.

[133] KOPPERMANN, P., DE SANTIS, F., HEYSZL, J., et al. "X25519 Hardware Implementation for Low-Latency Applications". In: *2016 Euromicro Conference on Digital System Design (DSD)*, pp. 99–106, 2016. doi: 10.1109/DSD.2016.65.

[134] YU, W., LIANG, F., HE, X., et al. "A Survey on the Edge Computing for the Internet of Things", *IEEE Access*, v. 6, pp. 6900–6919, 2018. doi: 10.1109/ACCESS.2017.2778504.

[135] FLYNN, M. J. "Some Computer Organizations and Their Effectiveness", *IEEE Transactions on Computers*, v. C-21, n. 9, pp. 948–960, 1972. doi: 10.1109/TC.1972.5009071.

[136] SILBERSCHATZ, A. *Operating system concepts*. Hoboken, NJ, J. Wiley & Sons, 2009. ISBN: 0470128720.

[137] TANENBAUM, A. *Modern operating systems*. Upper Saddle River, N.J, Pearson Prentice Hall, 2008. ISBN: 0136006639.

[138] AL-FUQAHA, A., GUIZANI, M., MOHAMMADI, M., et al. "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications", *IEEE Communications Surveys Tutorials*, v. 17, n. 4, pp. 2347–2376, 2015. doi: 10.1109/COMST.2015.2444095.

[139] SEMICONDUCTORS, N. "Freedom Development Platform for Kinetis® K64, K63, and K24 MCUs". Disponível em: <https://www.nxp.com/design/development-boards/freedom-development-boards/mcu-boards/freedom-development-platform-for-kinetis-k64-k63-and-k24-mcus:FRDM-K64F>. Acesso em: 2023-01-15.

[140] SALLES, R., FARIAS, R. "TLS Protocol Analysis Using IoTST - An IoT Benchmark Based on Scheduler Traces", *Sensors*, v. 23, n. 5, 2023. ISSN: 1424-8220. doi: 10.3390/s23052538.

[141] SHI, W., CAO, J., ZHANG, Q., et al. “Edge Computing: Vision and Challenges”. v. 3, pp. 637–646, 2016. doi: 10.1109/JIOT.2016.2579198.

Apêndice A

Artigo Publicado

- **Salles, R.; Farias, R. TLS Protocol Analysis Using IoTST—An IoT Benchmark Based on Scheduler Traces. [140]**

  Abstract: The Internet of Things (IoT) envisions billions of everyday objects sharing information. As new devices, applications and communication protocols are proposed for the IoT context, their evaluation, comparison, tuning and optimization become crucial and raise the need for a proper benchmark. While edge computing aims to provide network efficiency by distributed computing, this article moves towards sensor nodes in order to explore efficiency in the local processing performed by IoT devices. We present IoTST, a benchmark based on per-processor synchronized stack traces with the isolation and precise determination of the introduced overhead. It produces comparable detailed results and assists in determining the configuration that has the best processing operating point so that energy efficiency can also be considered. On benchmarking applications which involve network communication, the results can be influenced by the constant changes that occur in the state of the network. In order to circumvent such problems, different considerations or assumptions were used in the generalization experiments and the comparison to similar studies. To present IoTST usage on a real problem, we implemented it on a commercial off the-shelf (COTS) device and benchmarked a communication protocol, producing comparable results that are unaffected by the current network state. We evaluated different Transport-Layer Security (TLS) 1.3 handshake cipher suites at different frequencies and with various numbers of

cores. Among other results, we could determine that the selection of a specific suite (Curve25519 and RSA) can improve the computation latency by up to four times over the worst suite candidate (P-256 and ECDSA), while both providing the same security level (128 bits).