UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ALISSON MARCELO

Análise de desempenho da camada de segurança de aplicações DPWS

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Carlos Eduardo Pereira
Orientador

Prof. Dr. João Cesar Netto
Co-orientador

Porto Alegre, abril de 2013

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Marcelo, Alisson

Análise de desempenho da camada de segurança de aplicações DPWS / Alisson Marcelo. – Porto Alegre: PPGC da UFRGS, 2013.

78 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2013. Orientador: Carlos Eduardo Pereira; Coorientador: João Cesar Netto.

1. Web Services. 2. Sistemas Embarcados. 3. DPWS. 4. HTTPS. 5. Segurança. 6. Compressão. I. Pereira, Carlos Eduardo. II. Netto, João Cesar. III. Título.

AGRADECIMENTOS

Agradeço à minha esposa Valéria, pelo apoio incondicional neste trabalho, à minha família pelo suporte dado em todos os dias da minha vida, em especial ao meu irmão Audrei pela consultoria estatística dado ao longo do desenvolvimento do trabalho, e aos amigos por entender os momentos em que estive ausente. Agradeço às empresas nas quais trabalhei, pelo apoio dado nos momentos em que precisei cumprir os compromissos acadêmicos. Agradeço também aos orientadores e demais mestres, pelo conhecimento transmitido desde o início da graduação. E por fim, tão importante quanto os demais, o agradecimento ao povo brasileiro que acredita e financia a Universidade pública como forma de desenvolver o conhecimento científico no nosso país.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

3DES	Triple Data Encryption Standard
AES	Advanced Encryption Standard
ARM	Advanced RISC Machine
Corba	Common Object Request Broker Architecture
DMTF	Distributed Management Task Force
DPWS	Devices Profile for Web Services
ERP	Enterprise resource planning
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ITEA	Information Technology for European Advancement
J2SE	Java 2 Platform, Standard Edition
JavaME	Java Platform, Micro Edition
MD5	Message-Digest algorithm 5
OASIS	Organization for the Advancement of Structured Information Standards
REST	Representational state transfer
SHA1	Secure Hash Algorithm 1
SIRENA	Service Infrastructure for Real-time Embedded Networked Devices
SNMP	Simple Network Management Protocol
SOA	Service-oriented architecture
SODA	Service Oriented Device & Delivery Architecture
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UPnP	Universal Plug and Play
UDP	User Datagram Protocol
uDPWS	Micro DPWS

W3C World Wide Web Consortium

WS4D Web Services for Devices

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A disseminação dos dispositivos embarcados por diferentes domínios de aplicação torna necessário que a interação entre eles dispense ao máximo a intervenção humana. Diversos protocolos e tecnologias foram criados para a atingir este objetivo. Uma destas tecnologias é o Device Profile for Web Services (DPWS), que tem tido relativa aceitação tanto em ambientes industriais quanto residenciais. A expressividade e flexibilidade do DPWS tem permitido integrar dispositivos com recursos reduzidos diretamente com os sistema de gerenciamento das organizações. Com a diversidade dos dispositivos disponíveis e o crescimento das aplicações de redes sem fio, torna-se importante que os dados possam ser protegidos para manter a sua integridade e sigilo. Em aplicações como controle de processos, aquisição de dados estratégicos, medição de consumo de energia, entre outras, os prejuízos causados por interferências indevidas justificam a aplicação de estratégias de proteção dos dados, mesmo com o custo que elas podem gerar. Neste trabalho, são estudados os impactos da união da flexibilidade e expressividade do DPWS com a criptografia e compressão, fornecendo subsídios para que seja avaliado o custo de cada combinação destas soluções para definir quais níveis de proteção e otimização são mais adequados a cada cenário em que o DPWS possa ser aplicado. Ao fim do estudo realizado, demonstra-se o custo da criptografia na proteção dos dados, avaliando-se diferentes suítes de cifragem nas quais os algoritmos RC4 e o AES se mostram como opções viáveis em ambientes embarcados apesar do custo inerente ao processo de criptografia. Outro ponto importante é a comprovação estatística de que a compressão dos dados propicia um ganho de desempenho com redução de banda ocupada capaz de compensar o custo computacional da sua aplicação. Estas análises comprovam que o uso do tráfego seguro não pode ser aplicado a todos os casos devido às suas exigências de recursos. Em certas aplicações é possível achar um ponto intermediário onde se garanta apenas a autenticidade dos dados, deixando a criptografia completa para os casos mais críticos. Por fim, a conclusão à qual se chega é que o DPWS pode ser um substituto para as opções atuais de gerenciamento, sobre as quais tem como principal vantagem a capacidade de crescer em representatividade e complexidade de acordo com os recursos computacionais disponíveis.

Palavras-chave: Web Services, Sistemas Embarcados, DPWS, HTTPS, Segurança, Compressão.

Performance Analysis of Security Layer of DPWS application

ABSTRACT

The growth of use of embedded devices in several application domains makes necessary the reduction of human interaction for the devices control and has motivated the development of new technologies to achieve this goal. The Device Profile for Web Services (DPWS) is one of such technologies. It has been adopted on both industrial and home applications due to its flexibility, scalability and interoperability that allows the direct integration between deeply embedded devices and enterprise management systems. The remote device management requires a high level of data protection that could guarantee the integrity and confidentiality needed by industrial applications. In applications such as process control, strategic data acquisition, power consumption measurement, the possible harm caused by undue interference justify the use of data protection strategies despite the increasing of cost added by security technologies. This paper presents the results of a statistically proven study on the security layer applications of DPWS embedded devices using compressed data. The main objective of this work is to provide the background required to DPWS secure devices design, estimating the impact of many combinations of compression and security algorithms such as gzip, RC4 and AES. It proves that these technologies fit in embedded devices applications when data security is a strong requirement. Additionally, it is shown that performance degradation caused by data compression is compensated by the reduction of the amount of data to be encrypted. Best results were obtained when the compressed data fit in one Ethernet packet. Finally, is concluded that DPWS can replace the current alternatives to process management, having the advantages of scalability direct connection between deeply embedded devices and enterprise management systems.

Keywords: Web Services, Embedded Systems, DPWS, HTTPS, Security, Compression.

1 INTRODUÇÃO

A integração crescente entre os sistemas de automação e as redes corporativas das empresas nos últimos anos tem aberto novas possibilidades de pesquisa sobre as tecnologias utilizadas para realizá-la. O uso dos Web Services para a monitoração, controle e gerenciamento dos dispositivos é uma delas. A necessidade de configuração remota dos dispositivos existente em algumas aplicações motivou o estudo realizado neste trabalho, o qual analisa os efeitos da adição das camadas de criptografia e compressão em dispositivos de monitoração e controle utilizando o Devices Profile for Web Services (DPWS).

1.1 Motivação

O desenvolvimento tecnológico constante da área de sistemas embarcados tem reduzido o tamanho e o custo dos dispositivos e permitido que sejam desenvolvidas novas aplicações para eles. A possibilidade de integração direta dos componentes dos sistemas de automação às redes de dados de propósito geral existentes é uma delas. O protocolo IP, padrão nas redes de computadores e Internet, tem se destacado também no cenário da automação pela sua grande disseminação, simplicidade de uso e baixo custo dos equipamentos. Ele possibilita a integração dos dispositivos nas redes corporativas sem a necessidade de intermediários usando protocolos de mais alto nível como o UDP e o TCP. Os sistemas de automação anteriormente existentes não são adequados às novas necessidades do mercado, pois em muitos casos utilizam protocolos proprietários de comunicação e configuração que os isolam de dispositivos de outros fabricantes, tornando necessário o uso de diversos sistemas de gerenciamento. A disseminação da computação ubíqua e a crescente integração de tecnologia no cotidiano das pessoas torna mais evidente a necessidade de interoperabilidade e independência de plataforma. Do ponto de vista dos usuários, o ideal é que exista a convergência dos diferentes equipamentos de automação para um sistema de controle único e de alto nível que permita a passagem das diretivas de funcionamento aos dispositivos, a partir das quais eles executem suas tarefas de forma autônoma.

A aplicação deste novo paradigma nos sistemas de automação, exige que sejam observadas as necessidades de cada tipo de automação. Enquanto a automação industrial é essencial para a sobrevivência das empresas nos mercados dinâmicos e competitivos em que atuam, a automação predial e residencial, precisa ter facilidade de uso, fornecer economia, segurança e conforto para os seus usuários, além de estar atualizada em relação aos anseios dos usuários, como a utilização de dispositivos portáteis como interface com os dispositivos gerenciados. Em ambos os casos, os requisitos de segurança na comunicação, acesso aos dados e no controle dos dispositivos, devem garantir a integridade dos processos monitorados, sendo essenciais para a continuidade do funcionamento dos

equipamentos monitorados.

Os sistemas de produção tem se modificado da produção em massa para a produção em lotes menores e mais personalizados, exigindo que parte da inteligência do processo esteja nos dispositivos permitindo que sejam rapidamente reconfigurados, afim de obter máxima disponibilidade na execução dos processos. Essa rápida resposta permite o atendimento de requisitos técnicos e comerciais permite que os produtos fabricados possam atingir o mercado no momento correto, maximizando o retorno obtido. A aplicação de novas tecnologias no controle dos dispositivos de automação permite a adição de inteligência ao processo e o aumento da interoperabilidade entre os diferentes dispositivos. Apesar das vantagens obtidas com a inovação, o paradigma de gerenciamento utilizado deve ser compatível com os sistemas legados, pois não é possível remodelar completamente uma planta industrial em um curto espaço de tempo, sendo necessário que as novas tecnologias não gerem impactos negativos nos processos existentes e permitam a sua substituição gradual. Isto é obtido através de interfaces de comunicação bem definidas mantendo baixo o acoplamento na integração entre sistemas novos e os legados, facilitando a substituição dos sistemas legados e a entrada de novos dispositivos para aumentar o tamanho do sistema.

A necessidade que os sistemas de automação industrial tem de melhoria continua encontra como obstáculo a grande quantidade de variáveis envolvidas no controle dos processos, impedindo a otimização manual dos mesmos. Os sistemas de produção atuais necessitam de soluções que cresçam de acordo com a demanda e possam ser distribuídas nas diversas plantas industriais. Atualmente, grande parte das máquinas industriais necessita de reinstalação ou reconfiguração individual para realizar ajustes no processo. O uso de um controle automático que conecte estas máquinas em rede, permite uma resposta mais rápida a falhas e uma adaptação mais rápida da produção às necessidades atuais. O uso de protocolos de comunicação que forneçam capacidade de descoberta automática e notificação de eventos facilita a entrada em operação e o diagnóstico.

Com o crescimento dos sistemas é necessário que a automação ocorra também no gerenciamento com a delegação de certas responsabilidades aos dispositivos. Este modelo de gerenciamento distribuído aumenta a complexidade de implementação, mas permite a adaptação mais rápida às modificações de requisitos. Os dispositivos que antes eram elementos passivos se tornam elementos gerenciadores dos seus equipamentos, contribuindo para a otimização dos processos e repassando informações de forma assíncrona aos sistemas de gerenciamento hierarquicamente superiores, reduzindo o fluxo de dados na rede e o tempo de resposta às variações no processo.

Além do protocolo IP mencionado anteriormente, existem outras tecnologias que também tem tido destaque na automação industrial. O uso das redes de sensores sem fio para o controle de elementos simples do processo tem crescido, pois elas representam uma nova classe de dispositivos que trabalham cooperativamente para se integrar aos sistemas administrativos das empresas. Estas redes têm sido usadas em muitos cenários de aplicação, como monitoração de habitat, de motores industriais, de estruturas, de pontes, de vulcões, prevenção de incêndios florestais, monitoração hospitalar, controle de atuadores, dentre outras. A integração das redes de sensores diretamente com as redes de dados corporativas tem sido possível graças ao desenvolvimento do protocolo 6LowPAN (MONTENEGRO et al., 2007), que é uma especificação de protocolo IPv6 para dispositivos com restrições de consumo e banda sem a necessidade de *gateways* ou outros intermediários.

Para que seja possível integrar dispositivos de diversos níveis e complexidade é necessária a adoção de soluções que se adaptem ao tamanho destes equipamentos, que podem

ser desde servidores de alto desempenho até os referidos sensores sem fio com recursos muito reduzidos. É necessária uma tecnologia que permita ao dispositivo operar de forma autônoma, autoconfigurável e dinâmica, através do uso de uma camada de mais alto nível que atenda estes requisitos. Das tecnologias em voga atualmente destacam-se as arquiteturas orientadas a serviços por meio dos Web Services definidos pelo W3C (W3C, 2004a). Estes serviços são independentes de plataforma e baseados nos padrões da Internet, destacando-se pela interoperabilidade e pela sua grande utilização em diversos cenários. Com a arquitetura orientada a serviços (SOA) (VALIPOUR et al., 2009) é possível a um dispositivo em uma rede de sensores se comunicar com o sistemas de produção ou o ERP da empresa sem a necessidade de tradução de mensagens.

Os Web Services são baseados em especificações denominadas WS-* que por serem bastante genéricas e totalmente independentes de protocolos de transporte tem tamanho inadequado à aplicação direta em dispositivos como os sensores sem fio. Para ultrapassar a barreira de consumo de recursos e efetivamente integrar os dispositivos como serviços nas redes corporativas, foi desenvolvido o perfil de uso dos Web Services chamado DPWS (Device Profile for Web Services) (NIXON et al., 2009). O DPWS possui as vantagens da arquitetura de serviços aliadas à economia de recursos e outras melhorias inseridas pela especificação para adequá-lo ao uso em ambientes de gerenciamento e automação. O interesse pelo DPWS tem crescido a partir da sua especificação pela OASIS em 2009 e devido ao apoio de grandes empresas ligadas à área de automação e computação.

Este trabalho apresenta um estudo a respeito do desempenho do DPWS utilizando uma camada de segurança para aplicações em que a confidencialidade dos dados e a autenticação dos usuários é necessária para garantir a continuidade das tarefas desempenhadas e a integridade dos equipamentos controlados. As necessidades dos sistemas de automação industrial são a base das decisões tomadas no decorrer do trabalho, principalmente no que diz respeito aos casos de teste, mas de forma geral as soluções encontradas podem ser facilmente aplicadas a outros domínios de aplicação.

1.2 Visão geral do SOA e Web Services

Os Web Services têm se mostrado a tecnologia mais promissora no fornecimento de interoperabilidade entre sistemas com independência de plataforma, sendo usados também no suporte a sistemas legados e na comunicação destes com sistemas mais modernos formando plataformas heterogêneas de comunicação. Esse suporte os torna adequados à integração entre os elementos de "chão-de-fábrica"e os sistemas corporativos e comerciais das empresas, fornecendo grande flexibilidade, agilidade e reconfigurabilidade aos sistemas de produção. Nesse novo paradigma de utilização da tecnologia, os dispositivos são vistos como blocos funcionais que podem ser combinados de forma a otimizar os processos relacionados.

Com o uso dos Web Services a visualização do estado atual dos equipamentos ultrapassa as barreiras administrativas, podendo os dispositivos ser acessados pelo sistema de produção da empresa, para adequar a produção às necessidades de personalização dos pedidos pendentes ou acessados remotamente por clientes que possuem contratos com garantia de entrega, nos quais as fábricas dos fornecedores funcionam como extensão das suas linhas de produção, sendo influenciadas pelas alterações nos pedidos. Sendo assim, um cliente pode acompanhar remotamente a produção da sua encomenda e redefinir a ordem de entrega para se adequar às suas necessidades. Além disso, a aquisição de dados auxilia no planejamento de expansões na planta e no remanejamento de tarefas de

produção, reduzindo ociosidade que na indústria gera lucratividade menor.

Ainda há barreiras a serem transpostas para possibilitar o uso dos Web Services na automação, como o conservadorismo de alguns setores da indústria, que relutam em aceitar novos paradigmas porque acreditam que as tecnologias em uso são suficientes, dificultando a implantação de inovações. Todas as melhorias nos processos produtivos devem levar em conta dois fatores muito importantes, a integração dos sistemas legados e o custo da inovação, pois a sua adoção causará impacto na competitividade da empresa perante o mercado. Tal impacto deve ser compensado pelo ganho de produtividade fornecido pela tecnologia adotada. Além disso, é necessário ter em mente que as indústrias tem um processo de produção muito dinâmico, onde muitos produtos nascem e morrem frequentemente, que necessita de reprogramação constante dos equipamentos produtivos.

Além da questão do consumo de recursos, os Web Services carecem de algumas funcionalidades como descoberta automática, autodescrição, notificação de eventos para se adequarem ao uso na monitoração em ambientes de automação. A existência destas lacunas nas especificações existentes dos Web Services foi um dos motivos para o desenvolvimento do DPWS.

O DPWS é um perfil de uso dos Web Services, ou seja, uma série de regras que definem quais itens das especificações devem ser implementados e como isto deve ser feito a fim de reduzir a quantidade de recursos utilizados na implementação. Ele tem sido utilizado com sucesso em algumas áreas como automação industrial e residencial, redes de sensores sem fio, sistemas automotivos e em dispositivos que necessitem de comunicação em rede *plug-and-play* para gerenciamento, controle e monitoração. O DPWS utiliza como base as especificações abertas da Internet, como é feito nos Web Services, associando a elas mecanismos de localização automática e envio e recebimento de notificações de eventos. Devido ao tamanho reduzido e por ser baseado nos Web Services, o DPWS possibilita que os dispositivos se comuniquem diretamente com os ambientes de gerenciamento das empresas podendo ser compostos para realizar tarefas mais complexas. O mecanismo de descoberta automática do DPWS dispensa o uso de um terceiro elemento na rede para armazenar as descrições dos serviços, como o UDDI utilizado no padrão Web Services. O DPWS é totalmente distribuído, eliminando pontos únicos de falha existentes e outras soluções, tornando a solução mais robusta. Cada dispositivo DPWS é diretamente acessível por todos os elementos da rede.

1.3 Objetivo do trabalho

Este trabalho tem como objetivo principal avaliar o desempenho e consumo de recursos do perfil DPWS utilizando tráfego seguro. Como objetivos adicionais foram definidos: o fornecimento de subsídios para a definição de requisitos, especificação, implementação e embarcação do código em dispositivos. Para atingir este objetivo, foi realizada a implementação de um serviço DPWS ao qual fora adicionadas camadas de compressão e de criptografia dos dados, a qual permitiu a realização de experimentos que focaram na comparação de resultados entre as implementações com e sem a habilitação destas camadas de forma a mensurar o impacto de sua aplicação em um ambiente embarcado, no caso uma plataforma ARM9. Para validar os resultados obtidos e permitir a mensuração dos efeitos da aplicação de cada fator nos diferentes cenários de aplicação possíveis para esta implementação foi utilizada a metodologia de análise estatística denominada planejamento de experimentos (MONTGOMERY, 2008) por permitir a simplificação dos experimentos realizados e ser focada em otimização de processos.

No que se refere à implementação de segurança em dispositivos utilizando DPWS ainda há uma lacuna de trabalhos que avaliem os impactos do uso do perfil de Web Services em sistemas embarcados. Os sistemas seguros utilizando DPWS tem inúmeras aplicações possíveis. Algumas delas são listadas abaixo:

- Envio de novas rotinas, *scripts* e blocos executáveis para os serviços hospedados.
- Comunicação entre dispositivos de domínios administrativos diferentes.
- Sensores sem fio em ambientes abertos controlando elementos críticos.
- Redes de dispositivos distribuídas em ambientes com pouca segurança física.
- Controle de dispositivos através da Internet.

1.4 Organização do trabalho

Após o final deste capítulo introdutório, o trabalho está organizado como segue: o capítulo 2 apresenta uma introdução às arquiteturas orientadas a serviços e as alternativas de uso em dispositivos. No capítulo 3 é apresentado o estado da arte no uso do DPWS e os trabalhos relacionados. No capítulo 4 são mostradas algumas alternativas ao uso do DPWS e suas diferenças em relação a ele. O capítulo 5 mostra as tecnologias que dão suporte ao DPWS. O capítulo 6 apresenta as decisões tomadas no desenvolvimento do projeto do ambiente de testes e o capítulo 7 resume as principais características que motivaram a escolha das tecnologias que dão suporte à implementação. A metodologia dos experimentos é apresentada no capítulo 8. Os resultados obtidos serão apresentados e analisados no capítulo 9. O capítulo 10 apresenta as conclusões do trabalho e indica possibilidades de trabalhos futuros.

2 ARQUITETURAS ORIENTADAS A SERVIÇO

Com o crescimento do uso das redes de computadores nos últimos anos, o paradigma das arquiteturas orientadas a serviços (SOA) (VALIPOUR et al., 2009) tem sido objeto de trabalho de muitos grupos de pesquisa em diversas áreas da Ciência da Computação, dada a quantidade de oportunidades de estudo e aplicação existentes nesta área. A principal motivação do uso de serviços é o crescimento da comunicação entre sistemas de domínios administrativos diferentes. Os primeiros casos de utilização global de serviços foram os sistemas de consultas e compras *on-line*, tendo como característica a comunicação entre domínios desconhecidos entre si no qual a interoperabilidade e a proteção entre os dois sistemas é essencial, pois o provedor do serviço não pode presumir nada sobre o comportamento dos clientes e deve se proteger de clientes mal-intencionados. Na figura 2.1(PAPAZOGLOU; HEUVEL, 2007), podem ser vistos os principais papéis nas arquiteturas orientadas a serviços:o provedor, o requisitante (ou cliente) e o *broker* (ou registro). Os dois primeiros são essenciais à arquitetura, sendo o terceiro necessário para que clientes localizem novos provedores de serviços.

O conceito de serviço é definido de inúmeras formas por diversos trabalhos (HURWITZ et al., 2009) (W3C, 2004a) (PAPAZOGLOU; HEUVEL, 2007). De uma forma simplificada, define-se que a arquitetura orientada a serviços é um conjunto de princípios arquiteturais para construir sistemas autônomos interoperáveis entre si, independentes de linguagem e com baixíssimo acoplamento.

O uso dos serviços permite que cada entidade do sistema seja vista pelos demais como um provedor de serviços que possui chamadas padronizadas para se comunicar com os outros elementos da rede e para coordenar atividades conjuntas. Na definição de serviços não é fixado o tipo de tecnologia a ser utilizada para a interconexão entre eles, mas

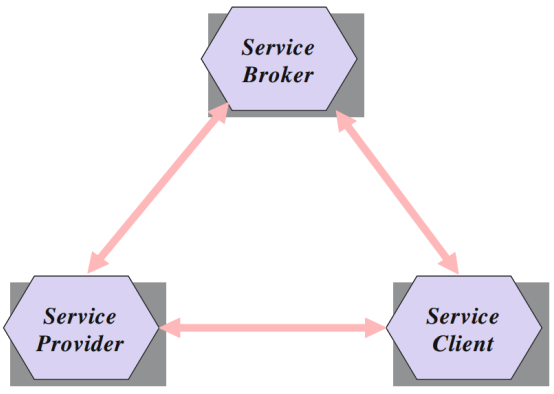

Figura 2.1: Principais papéis na arquitetura orientada a serviços.

isso deve ser feito em aplicações reais. Dentre as tecnologias que dão suporte à implementação de serviços se destacam os Web Services definidos pelo W3C (NEWCOMER, 2002) (BOOTH et al., 2004) (SUDA, 2003) (CERAMI, 2002), comumente chamados Web Services, e os REST Web Services (Representational State Transfer) (RICHARDSON; RUBY, 2007a) (CHINTHAKA, 2007) (RICHARDSON; RUBY, 2007b) (SILVA, 2008). A escolha dentre as duas tecnologias depende de questões como desempenho, armazenamento de estado e dependência do protocolo de transporte.

Uma característica importante dos serviços do ponto de vista dos usuários é que a função desempenhada por cada serviço deve ser facilmente entendida pelas pessoas envolvidas na sua especificação. Alguns exemplo de serviços são o preenchimento de um formulário, a compra de algum item, o acionamento de uma lâmpada, a consulta de uma lista de contatos, etc.

Os serviços são baseados em princípios que buscam criar aplicações tão flexíveis quanto possível.

Além da interoperabilidade entre sistemas diferentes, os serviços possuem algumas características adicionais que proporcionam ganhos na sua aplicação (WIDJAJA; BUXMANN, 2009) (THURAISINGHAM, 2010), sendo algumas delas dependentes da tecnologia utilizada para a implementação:

- Visão homogênea: o uso dos serviços fornece uma visão homogênea de sistemas heterogêneos, como por exemplo, o acionamento de válvulas pode ser visto como serviço, independente do fabricante da mesma.
- Dinamicidade: os serviços são anunciados quando os dispositivos entram na rede, sendo assim, os demais componentes percebem a presença de novos serviços sempre que eles se tornam ativos.
- Autonomia: os serviços são independentes e estruturalmente desacoplados.
- Autodescrição e descoberta: cada dispositivo pode consultar quais capacidades os outros dispositivos possuem, usando a noção de autodescrição inerente ao SOA. Ao entrar em uma rede pode anunciar quais as funcionalidades disponibiliza.
- Encapsulamento: uma vez que os serviços são auto descritos, todos os detalhes da implementação do serviço são ocultados dos seus clientes, os quais conhecem apenas as suas interfaces de comunicação e as mensagens necessárias para invocá-las. A comunicação por mensagens padronizadas reduz significativamente o acoplamento entre os elementos deste tipo de comunicação.
- Notificações: podem ser estabelecidos valores limite para certas grandezas medidas pelo dispositivo, como por exemplo, um sensor de temperatura que envia notificações ao ultrapassar o limite de operação em um processo.
- Responsabilidades distribuídas: parte das decisões sobre o processo controlado é tomada no dispositivo a partir de diretivas definidas previamente pelo sistema de gerenciamento central.

As características dos serviços, como descoberta e autodescrição, fornecem a eles capacidades chamadas *plug-and-play*, ou seja, que possam ser automaticamente reconhecidos e utilizados na redes nas quais ingressam, mesmo com a sua heterogeneidade de hardware e software. Na definição da arquitetura orientada a serviços, em ambientes extremamente distribuídos, existe a presença dos serviços de registro de Web Services, os quais são consultados pelos clientes sobre a localização dos serviços.

Para atender às requisições dos usuários, o serviço de um dispositivo pode invocar ser-

viços de outros dispostivos aumentando a sua expressividade do ponto de vista do usuário. Por exemplo, um serviço de compras *on-line* pode consultar serviços de código postal para validar os dados fornecidos pelo usuário. Um serviço que controle a iluminação ambiente de um determinado espaço, pode interagir com os serviços de cada lâmpada, ajustando a luminosidade das que possuem luminosidade variável e desligando as restantes. Em outros ambientes, um serviço correspondente à automação de um processo complexo pode invocar serviços específicos que monitoram e controlam componentes mais simples como sensores e atuadores. Esta combinação dos serviços para prover serviços mais complexos é realizada utilizando composição de serviços (orquestração ou coreografia) (PELTZ, 2003), isto permite que a interação entre os sistemas possa ser definida externamente, através de linguagens bem definidas. Isto fornece uma nova metáfora de definição de sistemas complexos, a qual possui uma visão distribuída e modularizada em sistemas autônomos interagindo entre si (ZEEB et al., 2010a).

Os princípios de baixo acoplamento e vinculação tardia são fatores cruciais para o sucesso dos serviços nas redes de computadores convencionais. O crescimento do paradigma SOA tem auxiliado na pesquisa de arquiteturas e paradigmas modernos de controle distribuído. Muitas pesquisas nesta área têm sido direcionadas para comércio eletrônico e interações entre corporações. Devido a esse sucesso, os serviços se apresentam como alternativa para comunicação em outros domínios como redes de automação e redes de sensoreamento. Estes novos domínios para a utilização dos serviços vem sendo constantemente objeto de pesquisas nestas áreas. Outra aplicação dos serviços é o seu uso em sistemas multi-agentes.

Este trabalho trata da utilização da arquitetura orientada a serviços no ambiente de dispositivos que podem ser usados na área de automação residencial ou industrial. A utilização dos serviços permite que os dispositivos sejam visíveis de qualquer ponto do sistema desde outros dispositivos até os níveis gerenciais. A arquitetura orientada a serviços é uma solução consolidada nos níveis de negócio das empresas e por isso já vem sendo há algum tempo estudada para a aplicação na área de automação. A heterogeneidade dos sistemas de automação aumenta a complexidade e exige a existência da interoperabilidade entre os seus componentes. O uso dos serviços em sistemas de automação permite uma adaptação mais rápida às mudanças que podem ocorrer no fluxo do processo. Devido ao seu baixo acoplamento, eles possibilitam a atualização independente dos serviços desde que as interfaces não sejam alteradas.

Em automação residencial, os serviços ajudam a interligar equipamentos de diferentes fabricantes, permitindo que eles possam ser controlados pelo mesmo sistema de gerenciamento. Na área industrial pode ser utilizado na integração equipamentos da infraestrutura corporativa, permitindo o uso das informações obtidas diretamente dos equipamentos nas decisões gerenciais.

O desenvolvimento de serviços leves para a utilização em automação tem sido motivado pelo fato dos sistemas de automação poderem se beneficiar das características dos serviços mencionadas anteriormente. Os sistemas de automação atuais são heterogêneos, formados por equipamentos de diversos fabricantes, com diferentes componentes e necessitam alcançar tanto objetivos locais, como metas diárias de produção, quanto globais, onde as empresas precisam desenvolver produtos cada vez mais adaptados às necessidades de cada cliente. Desta forma, os sistemas de manufatura modernos tem migrado para a forma descentralizada de controle, na qual as unidades de manufatura são cada vez mais inteligentes, tolerantes a falhas e adaptáveis às constantes modificações nas características dos processos produtivos geradas pela dinamicidade do mercado.

Nos sistemas de automação cada serviço opera de forma autônoma desempenhando um papel diferente, como por exemplo produção, transporte, monitoração, etc. Os serviços são fornecidos em ambos os sentidos, no qual os dispositivos requisitam e disponibilizam serviços. Em alguns casos é possível que os componentes do sistema disputem entre si por determinados recursos. Neste caso, os dispositivos controladores destes recursos devem ser capazes de priorizar as requisições para se alcançar os objetivos definidos.

Os serviços têm desempenhando um papel importante em cenários centrados em dispositivos, pois muitos padrões da indústria para redes cabeadas e sem fio para comunicações máquina-à-máquina não oferecerem a flexibilidade necessária aos novos padrões de automação. Em muitos casos, os protocolos de comunicação e os conceitos utilizados são dependentes de um tipo específico de equipamento ou de tecnologias proprietárias. Isto dificulta a migração da solução utilizada entre tipos de hardware diferentes e a reprogramação de todos o sistema quando é necessária a substituição de um determinado componente. Com o uso dos serviços, isto não seria necessário, visto que para substituir um determinado componente do sistema basta que o substituto responda à mesma interface ou que cada serviço que o controla tenha a interface de comunicação readequada. Além disso, a flexibilidade dos serviços permite que eles sejam utilizados para a adaptação de sistemas legados em novos sistemas de automação. À medida que os sistemas legados forem substituídos por novos sistemas que atendam aos novos paradigmas, apenas estes devem ser adequados às interfaces existentes e não os seus clientes, reduzindo o impacto desta substituição no restante do sistema.

3 TRABALHOS RELACIONADOS

Desde a primeira versão da especificação do DPWS, um número significativo de trabalhos foram elaborados para estudar o uso dos Web Services em sistemas embarcados. O projeto SIRENA (BOHN; BOBEK; GOLATOWSKI, 2006) teve objetivo de alavancar o uso das arquiteturas orientadas a serviços em dispositivos embarcados sem a necessidade de *gateways* ou outros intermediários em quatro domínios distintos de automação: industrial, automotivo, residencial e de telecomunicações. Ele descreve o funcionamento do DPWS e de um *framework* desenvolvido que estava fortemente ligado ao desenvolvimento do DPWS especificado pela Microsoft, o qual possui suporte nos seus sistemas operacionais desde o Windows Vista. Este foi o primeiro *framework* encontrado para o DPWS que não o suporte dado pela Microsoft no .NET Micro Framework 2.5. Este trabalho descreve os motivos que levaram à escolha do DPWS como base para o uso dos Web Services, comparando com outras tecnologias de controle de dispositivos que apesar de similares possuem algumas desvantagens em relação à escolha feita. O *framework* desenvolvido pelo projeto SIRENA, como grande parte dos *frameworks* DPWS desenvolvidos em linguagem C, utiliza o pacote gSOAP que executa todo o tratamento do suporte à tecnologia dos Web Services, que é a base do DPWS. Sobre a implementação de cliente e serviço gerada pelo gSOAP atendendo às restrições do DPWS é acrescentado o código que implementa as especificações WS-* que não são parte da implementação padrão dos Web Services. Embora o *framework* proveniente do projeto SIRENA não seja diretamente utilizado atualmente, serviu de base para as implementações mais conhecidas como o DPWS Core do SOA4D e o WS4D-gSOAP. Como sequência do projeto SIRENA foram realizados os projetos SOCRADES e SODA. Ambos dedicados a utilização do DPWS em ambiente industrial.

O SOCRADES (Service-Orientated Cross-layer InfRAstructure for Distributed Smart Embedded Devices) (DE SOUZA et al., 2008) foi um projeto desenvolvido entre 2006 e 2009 que teve como objetivo principal desenvolver uma plataforma de projeto, execução e gerenciamento de uma nova geração de sistemas de automação, explorando as arquiteturas orientadas a serviços nos níveis de aplicação e de dispositivo. Foi liderado pela Schneider Electric e teve a participação de 15 grandes instituições de 6 países europeus, entre elas grandes corporações Europeias ligadas à área de automação industrial. Este projeto buscou integrar as necessidades da indústria com a tecnologia dos Web Services. Este estudo foi importante pois realizou um levantamento dos requisitos necessários para a aplicação dos serviços na área de automação, separando-os em requisitos funcionais e não funcionais, como a integração de dispositivos legados, gerenciamento e históricos integrados, segurança e suporte a composição.

O projeto SODA (Service Oriented Device and Delivery Architecture) foi um projeto desenvolvido entre 2006 e 2008, cujo objetivo foi a criação de um ecossistema orien-

Feature	Axis2	gSOAP	J2ME	uDPWS
Versão DPWS	V1	V1 & V1.1	V1 & V1.1	V1.1
IPv6	não	parcial	sim	sim
Plataforma	servidor	disposit.	disposit.	disposit. red.
DPWS security		canal	canal	
Suporte a discovery proxy			sim	
MTOM	sim	sim	sim	
Eventos		sim	sim	
SOAP-over-UDP binding	sim	sim	sim	

Tabela 3.1: Funcionalidades nos diferentes toolkits desenvolvidos pelos WS4D.

tado a serviços sobre o framework desenvolvido no SIRENA para comunicações de mais alto nível. Com foco em dispositivos embarcados de tempo real para diversos domínios, tais como, controle automotivo, automação industrial, predial e residencial, telecomunicações, telemetria e instrumentação médica. Um dos objetivos do projeto era econômico buscando a utilização dos Web Services em dispositivos ao custo de 5 euros, reduzindo os recursos necessários para a implementação da arquitetura orientada a serviços. Colaborou com o projeto SOCRADES no desenvolvimento de frameworks. Desta colaboração resultou o SOA4D. O SOA4D possui implementações em C (DPWS Core) e em Java (DPWS4J Core), implementações do WS-Managment e drivers para o OSGi.

O WS4D (ZEEB et al., 2010b) é um projeto em constante atividade com o objetivo de levar a tecnologia dos Web Services e Arquiteturas Orientadas a Serviços para os domínios de automação, entretenimento, telecomunicações e sistemas automotivos. Este projeto tem gerado uma grande quantidade de trabalhos relacionados ao uso de Web Services no dispositivos. Além disso, deste projeto provém os frameworks, WS4D-gSOAP (C/C++), WS4D-uDPWS (C), WS4D-JMEDS (Java) e WS4D-Axis2 (Java). Sendo o segundo o mais importante para este trabalho, pois ele é a base da implementação utilizada neste trabalho para o estudo realizado. O uDPWS foi desenvolvido especificamente para dispositivos com recursos computacionais restritos. Ele será descrito com maiores detalhes no capítulo a respeito das tecnologias utilizadas.

Diversos trabalhos tem utilizado o DPWS, sendo alguns exemplos fornecidos a seguir.

Milagaia (2009) apresentou uma forma de integrar dispositivos de produção reais e simulados. Isto permite que os sistemas reais possam substituir gradativamente os dispositivos simulados tendo-se sempre a visão de todo o sistema em funcionamento através de uma interface genérica para dispositivos baseados no DPWS. Esta interface suporta reconfiguração e reprogramação, obtendo aumento da escalabilidade e redução dos requisitos de memória. Foi mostrada a integração do DPWS com sistemas multiagentes distribuídos, elaborando-se uma camada intermediária (*middleware*) que permite que os agentes interajam com os dispositivos DPWS. A prova de conceito desenvolvida é totalmente simulada a partir de modelos tridimensionais.

Zeeb (2007) forneceu uma descrição resumida do perfil de uso DPWS no *framework* desenvolvido pelo WS4D, servindo como fonte introdutória para o entendimento do funcionamento do DPWS. O mesmo grupo de pesquisa investigou a utilização da arquitetura baseada em componentes (ZEEB et al., 2010a), conhecida em aplicações corporativas, no ambiente de Web Services embarcados utilizando o DPWS. Neste trabalho são investigadas formas de composição de serviços distribuídos aplicadas a dispositivos. Zeeb(2007b)

ainda descreveu alguns percalços encontrados na implementação dos frameworks e listadas algumas lições aprendidas.

Moritz (2010) investigou uma possível substituição do DPWS, baseado em especificações WS-*, por uma abordagem REST (Representational State Transfer) que exige menos recursos computacionais do que o DPWS. Foi feita uma comparação entre as duas tecnologias verificando que há espaço para o crescimentos de ambas em diferentes cenários, nos casos em que é necessário desempenho sem salvamento do estado da comunicação no protocolo ou quando é necessária independência da camada de transporte.

Eichhorn (2010) realizou uma análise da aplicabilidade dos Web Services e do DPWS na comunicação com dispositivos embarcados em veículos automotores e uma comparação entre as tecnologias existentes para a utilização em sistemas embarcados que resultou em uma tabela que serve como consulta para outros trabalhos nesta área.

Sousa (2009) foi desenvolvida uma abordagem que permite a carga dinâmica de serviços simples nos dispositivos para atender a necessidades específicas que surgem em tempo de execução, aumentando a agilidade da adaptação dos serviços às alterações nos processos produtivos. A vantagem desta abordagem é que o serviço a ser executado passa a ser um parâmetro do processo e não precisa ser conhecido antecipadamente, sendo atualizado em tempo de execução.

Jammes (2009) realizou um estudo para a integração da especificação Web Services para gerenciamento, o WS-Management, com o DPWS. O WS-Management teve a sua especificação publicada pelo DMTF. Ele tem o objetivo de prover gerenciamento, utilizando Web Services para PCs, servidores, dispositivos e outras entidades gerenciáveis, adicionando as vantagens do DPWS.

Bobek (2008) analisou formas de especificar os serviços e dispositivos através de modelos (*templates*), buscando fornecer a base teórica ao desenvolvimento dos dispositivos, a qual não havia sido suficientemente abordada até então. Esta especificação dos *templates* é particularmente interessante quando é necessário o desenvolvimento de serviços em dispositivos semelhantes que necessitam de uma interface padronizada para facilitar o uso por terceiros.

Outro trabalho interessante que pode ser citado é a integração de redes de sensores utilizando *gateways* DPWS (SAMARAS; GIALELIS; HASSAPIS, 2009), permitindo que dispositivos, que não possuem recursos computacionais suficientes para a execução do DPWS possam ser integrados nas redes de monitoramento. Moritz (2010) propôs abordagens de codificação e compressão para otimizar o consumo de rede de dispositivos com DPWS.

Trabalhos mais recentes buscaram explorar novas possibilidades na aplicação do DPWS em diversas áreas, como a composição de serviços em dispositivos na Internet das Coisas (CUBO; BROGI; PIMENTEL, ????) e a integração de nodos sensores e atuadores sem fio através do uso de composição de Web Services com monitoramento via Internet (KYUSAKOV et al., 2013). Outros trabalhos investigaram a aplicação do DPWS na área de energia elétrica para a medição inteligente de energia (ALTMANN et al., ????) e para o mapeamento do protocolo IEC 61850 para Web Services, disponibilizando a automação de subestações através das redes de dados na forma de serviços.

4 ALTERNATIVAS AO USO DO DPWS

Nesta seção serão descritos resumidamente as alternativas ao uso do DPWS. Desde a popularização dos dispositivos foram diversas as iniciativas para a monitoração e controle de dispositivos através de redes de dados. Cada solução foi voltada a um domínio de aplicação específico podendo ser utilizada de forma genérica para outros. Não é a intenção deste trabalho a afirmação do DPWS como solução única para todos os casos, mas demonstrar de que modo a flexibilidade fornecida pelas tecnologias nas quais é baseado pode ser aplicada em ambientes de automação, focando principalmente em monitoração e controle.

4.1 SNMP

O SNMP (Simple Network Management Protocol) (CASE et al., 1990) é sem dúvida o protocolo mais conhecido e amplamente utilizado para a monitoração dispositivos de rede. A sua simplicidade faz com que ele possa ser facilmente implementado em dispositivos com capacidade computacional extremamente reduzida. Enquanto na monitoração de dispositivos o SNMP é o mais popular, no controle a situação é diferente, pois a falta de uma camada de segurança amplamente aceita e a falta de expressividade em alguns casos motivou a investigação de novas formas de gerenciamento.

4.2 Jini

O Jini (Sun, 2001) é uma solução baseada em Java com o objetivo de fornecer mecanismos para a descoberta, distribuição e migração de executáveis entre computadores. O centro de um sistema usando Jini é o *lookupservice*, que é um repositório que armazena informações a respeito dos serviços com suas descrições e funcionalidades. A comunicação é baseada no Java RMI (Remote Method Invocation). Sendo baseado em Java é dedutível que os componentes deste sistema deverão executar sobre uma máquina virtual Java.

4.3 OSGi

O OSGI (The OSGi Alliance, 2007) é voltado para redes residenciais de dispositivos. Tem um arquitetura baseada em Java. Possui um dispositivo central chamado de *gateway* que provê a plataforma de comunicação necessária para acessar os *bundles*, que são unidades coesas e autossuficientes normalmente implementadas como classes Java associadas a determinados recursos, as quais proveem uma interface por meio da qual ou-

tros processos acessam as suas funcionalidades. O sistema é projetado para ser dinâmico podendo descobrir e instalar novos *bundles* em tempo de execução. No OSGi os dispositivos são descritos por meio de interfaces Java. Nesta arquitetura também é possível a alteração do executável do dispositivo durante a execução. O acesso remoto aos *bundles* não está definido no OSGi, sendo necessária a sua associação com outras tecnologias de comunicação de dispositivos como por exemplo o UPnP ou até mesmo o DPWS. Neste caso, seria necessário considerar o peso de três tecnologias sobre o sistema embarcado: a máquina virtual, o OSGi e o perfil de comunicação sobre ele.

4.4 UPnP

O UPnP (Universal Plug and Play) (UPNP, 2006) usa protocolos padronizados e abertos baseados em XML, assim como o DPWS, para descrever e controlar os dispositivos. A base da comunicação é a pilha TCP/IP utilizando em algumas implementações o SOAP para a descrição. São definidos mecanismos de endereçamento, descoberta, descrição, controle, eventos e apresentação dos dispositivos. A comunicação é ponto-a-ponto nos dispositivos aumentando a confiabilidade do sistema. Em sua primeira versão (1.0) o UPnP não definia estratégias de segurança para a proteção dos dados dos dispositivos e não fornecia suporte nativo ao IPv6 na sua especificação.

4.5 Corba

Por muito tempo o Corba (OMG, 2006) foi considerado um *middleware* promissor no acesso a métodos remotos, fornecendo interoperabilidade e independência de plataforma, hardware e linguagem de programação. Entretanto a sua descrição de alto nível dos dispositivos se opunha aos anseios do mercado que necessitava de tecnologias intuitivas que facilitassem o aprendizado e se acelerassem o tempo de desenvolvimento.

4.6 REST

Uma tecnologia muito popular atualmente é a dos *Web Services* REST (FIELDING, 2000) que descrevem serviços usando XML sobre HTTP, sem a sobrecarga do SOAP usado no DPWS buscando melhor desempenho. Ele utiliza seis operações (GET, POST, PUT, DELETE, HEAD, OPTIONS) que são definidas na especificação do protocolo HTTP. Não há armazenamento de estado entre as mensagens, o que em alguns casos pode ser necessário e exigir que os estados seja adicionados a todas as mensagens.

4.7 Netconf Light

O Network Configuration Protocol Light (NETCONF Light) (SCHONWALDER, 2012) é um *draft* recente que trata do uso do protocolo de gerenciamento de redes Netconf de forma modular para permitir a sua aplicação em dispositivos com recursos extremamente reduzidos. Ele parte do princípio que tanto a quantidade de dados configuráveis quanto a quantidade de sistemas de gerência para os quais os dispositivos deste tipo devem responder é pequena. A comunicação segura ainda é um ponto em aberto ao qual o *draft* sugere a utilização de TLS ou DTLS, mas sem fornecer maiores detalhes de como a sua aplicação deve ser realizada.

5 ANÁLISE DO DPWS

O DPWS é um perfil de uso dos Web Services que define um conjunto de restrições de implementação para o desenvolvimento de Web Services para o seu uso em dispositivos com recursos reduzidos. As especificações utilizadas pelo DPWS definem os formatos utilizados para o envio de mensagens de descoberta, descrição, notificação de eventos, dentre outras. Ele é totalmente compatível com a tecnologia dos Web Services e inclui pontos de extensão que permitem a integração dos dispositivos nos sistemas de gerenciamento existentes nas corporações. Além das especificações WS-*, o DPWS é baseado em protocolos consolidados para a transmissão dos dados, como a pilha TCP/IP e o HTTP. Mesmo sendo projetados para ser uma forma reduzida dos Web Services SOAP, os dispositivos DPWS tem como vantagem a possibilidade de ser acessados diretamente por qualquer cliente Web Service SOAP disponível nas empresas.

Apesar de ser definido sobre XML, a codificação UTF-8 não é obrigatória no DPWS, ela apenas é a mais utilizada devido a questões de interoperabilidade entre sistemas. É permitido que outras formas de codificação sejam utilizadas para os dados, como por exemplo o SOAP Message Transmission Optimization Mechanism (MTOM) (MARTIN GUDGIN NOAH MENDELSOHN, 2005) e o Efficient XML Interchange (EXI) (BOURNEZ, 2009).

O DPWS foi publicado em 2004, sendo tornado padrão pela OASIS (Organization for the Advancement of Structured Information Standards) em 2009 (NIXON et al., 2009), em conjunto com WS-Discovery 1.1 e SOAP-over-UDP 1.1, ambas relativas ao processo de descoberta dos dispositivos DPWS na rede. Estas especificações adicionais foram necessárias para simplificar o processo de descoberta dos dispositivos. Apesar da interoperabilidade entre equipamentos que implementem uma determinada versão do DPWS, as versões 1.0 e 1.1 do perfil não são compatíveis entre si.

Diversas implementações do DPWS podem ser encontradas, sendo que as mais conhecidas são o WS4D.org e o SOA4D.org. Os sistemas operacionais Microsoft Windows Vista e 7 possuem suporte ao DPWS. Além destas versões o .NET Micro Framework também possui o perfil integrado. Em alguns casos, onde o DPWS possui tamanho maior do que o dispositivo pode suportar, outras tecnologias de conexão podem ser utilizadas até um determinado ponto da rede que servirá de *gateway* convertendo as mensagens desta outra tecnologia para o DPWS.

As arquiteturas orientadas a serviços possuem dois componentes principais, os serviços e os clientes. Adicionam-se a eles, os serviços de diretórios para a localização dos serviços. No DPWS a função dos serviços de diretórios foi substituída pelo uso da especificação WS-Discovery. No DPWS os serviços são implementados nos dispositivos e os clientes podem ser executados em qualquer equipamento com suporte à comunicação utilizando XML, podendo estar inclusive em um outro dispositivo.

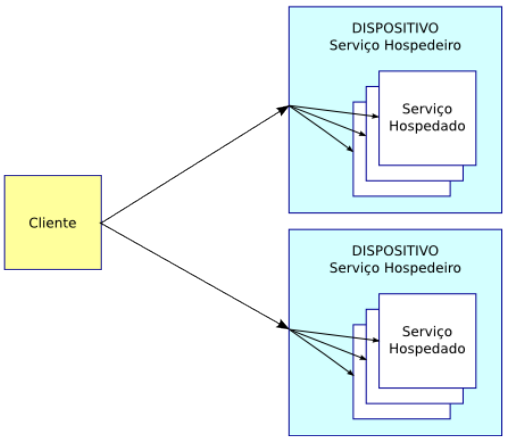

Figura 5.1: Tipos de serviços do DPWS.

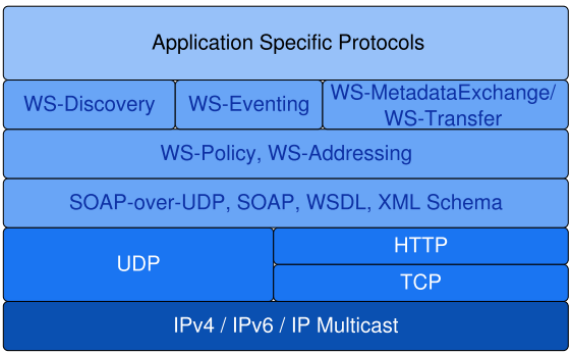

Figura 5.2: Pilha de especificações utilizadas no DPWS.

São definidos dois tipos de serviços sendo executados nos dispositivos como mostrado na figura 5.1, o serviço hospedeiro e os serviços hospedados. O primeiro é diretamente associado ao dispositivo, atua no processo de descoberta e fornece o acesso aos serviços hospedados. Os serviços hospedados são ligados fortemente às funções desempenhadas pelo dispositivo. Os serviços hospedeiros implementa alguns serviços adicionais:

- Descoberta: usado para a publicação dos serviço disponibilizados pelo dispositivo.
- Troca de metadados: provê acesso dinâmico aos serviços hospedados e seus metadados
- Publicação e inscrição de serviços de eventos: possibilita que outros dispositivos se inscrevam para o recebimento de eventos assíncronos

O DPWS é baseado na pilha de protocolos que dá suporte ao uso do HTTP e aos datagramas UDP e em especificações básicas ao uso de Web Services como WSDL 1.1 (CHRISTENSEN et al., 2001), XML Schema (W3C, 2004b), SOAP 1.2 (W3C, 2007), WS-Addressing (BOX et al., 2004), WS-MetadataExchange (MALHOTRA et al., 2009), WS-Transfer (CHOU et al., 2009), WS-Policy (BAJAJ et al., 2006), WS-Security (NADALIN et al., 2006), WS-Discovery (NIXON; REGNIER, 2009) e WS-Eventing (MALHOTRA et al., 2009). A pilha de especificações utilizadas no DPWS pode ser vista na Figura5.2 (ZEEB et al., 2010b). Os protocolos que dão suporte ao DPWS como SOAP, HTTP e a especificações WSDL e XML-Schema não serão explicadas neste trabalho por não fazerem parte do escopo principal.

Os dispositivos são descritos através de metadados, os quais carregam consigo um

número de versão que permite aos clientes acompanhar a evolução das funcionalidades disponíveis. As descrições dos serviços são baseados nas especificações WS-Metadata Exchange e WS-Transfer, e a descoberta no WS-Discovery. O envio de notificações de modificações no estado do dispositivo é feita usando o padrão *Publisher/Subscriber* através da especificação WS-Eventing. O DPWS suporta o endereçamento utilizando IPv4 e IPv6 nativamente, podendo o endereço utilizado ser obtido por DHCP, opcionalmente. O DPWS prevê o uso de segurança utilizando a especificação WS-Security. Na prática, porém, a utilização do WS-Security não foi amplamente aceita devido à exigências de recursos desta especificação ocasionada pelo uso de criptografia assimétrica.

Os serviços são disponibilizados como *Endpoints* identificados em seções diferentes de metadados nos arquivos XML de descrição. Os serviços hospedados nos dispositivos são *Web Services* regulares, contendo as operações que podem ser chamadas pelos clientes.

Apesar das vantagens de interoperabilidade e modularidade do DPWS, o perfil possui uma curva de aprendizado considerável e em alguns casos gera alguns equívocos. A forma como as especificações WS-* são escritas não auxiliam no aprendizado, pois referenciam muito as outras especificações WS-*. Também faltam em alguns casos trabalhos que apliquem efetivamente o DPWS. Esta situação tem melhorado com o desenvolvimento de *toolkits* de desenvolvimento como os do WS4D, SOA4D e do suporte no .Net Microframework.

O DPWS apenas define a estrutura básica para a comunicação e disponibilização dos serviços. Isto repassa ao projetista da aplicação a responsabilidade de especificar como os serviços serão adaptados às suas necessidades. Entretanto, em alguns casos não é fácil de identificar quais pontos da especificação DPWS são mandatórios e não são passíveis de modificação.

Em algumas aplicações, somente um dispositivo isolado não atende às necessidades do sistema. Por ser baseado nos Web Services, uma forma de fazer os dispositivos trabalharem em conjunto é utilizar composição ou agregação definidos nas especificações Web Services. Nestas mensagens são enviadas informações a respeito do tipo e escopo do dispositivo.

5.1 Especificações WS-* usadas no DPWS

Embora tenham sido brevemente comentadas no início deste capítulo serão detalhadas nesta seção as especificações WS-* utilizadas no DPWS Estas especificações definem as *tags* e atributos necessários para transportar as informações e o comportamento esperado de cada ente nesta comunicação.

O DPWS define o perfil de uso das especificações restringindo a forma como elas devem ser utilizadas, de modo a reduzir o tamanho das mensagens enviadas. Ainda há na especificação algumas partes que não são totalmente claras em relação a essas restrições no que diz respeito ao formato das mensagens, o que pode gerar problemas de compatibilidade em alguns casos.

O dispositivo deve no mínimo atender o envio de envelopes SOAP 1.2 sobre HTTP. Deve suportar padrões de mensagens de requisição/resposta e responder mensagens de sentido único.

5.1.1 WS-Discovery

Responsável pelo anúncio e busca de dispositivos na rede. É implementada utilizando as definições do SOAP-over-UDP. Qualquer dispositivo na rede pode enviar mensagens *Probe* para a descoberta de novos dispositivos.

Em ambientes com uma grande quantidade de dispositivos ou nos casos em que é necessário o acesso administrativamente separado, foi definida a figura do *discovery proxy*, que evita a grande quantidade de mensagens de *multicast* e permite a tradução de endereços quando necessária.

Para reduzir o consumo de banda, apenas o serviço que descreve o dispositivo deve ser mencionado nas mensagens de divulgação. Os serviços hospedados não participam do processo de divulgação na rede.

Para suportar o WS-Discovery as mensagens devem ser enviadas usando tanto *multicast* quanto *unicast*. No caso de dispositivos com recursos extremamente restritos, o WS-Discovery pode ser suprimido da implementação.

No caso genérico, o cliente descobre o dispositivo usando o WS-Discovery e obtém descrição do dispositivo, a partir da qual são obtidos os endereços dos serviços na seção Relationship MetadataSection.

Informações a repeito do dispositivo são disponibilizadas nas seções de metadados ThisModel e ThisDevice.

Dentro da especificação WS-Discovery são definidos mecanismos para que o dispositivo anuncie a sua entrada e saída de uma determinada rede. Essas mensagens (*Hello e Bye*) permitem que os clientes fiquem sabendo da existência de novos dispositivos na rede ou através de mensagens *Probe* possam consultar os dispositivos presentes na rede. Uma das finalidades deste processo de descoberta é substituir serviços centralizados de registro de serviços, como os UDDIs, definidos nas especificações Web Services. A descoberta de serviços utiliza mensagens *multicast* sobre protocolo UDP.

Um problema do processo de descoberta é a quantidade de mensagens *multicast* e *unicast* que trafegam na rede apenas para este processo.

Em cenários menos dinâmicos onde o os componentes do sistema são conhecidos previamente, é possível otimizar o processo de identificação dos serviços analisando previamente os dados de descrição do serviço.

5.1.2 WS-Addressing

Provê um identificador único UUID para cada entidade DPWS (serviço ou dispositivo). Estas informações são transportadas no cabeçalho SOAP. Permitindo utilizar qualquer protocolo de transporte. Estes identificadores únicos são atribuídos aos chamados *endpoints*. Com essa especificação pode-se mudar o protocolo de transporte sem impacto na implementação Web Service, podendo ser qualquer tipo de dados XML.

5.1.3 WS-MetadataExchange / WS-Transfer

Estas especificações definem como as informações a respeito dos serviços são repassadas aos clientes. No WS-MetadataExchange são criadas MetadataSections que podem ser especificadas pelo projetista do sistema. O WS-Transfer é usado para recuperar os metadados dos serviços. Em muitos casos, a simplicidade do WS-Transfer é suficiente para a transferência das informações a respeito dos serviços disponibilizados pelos *endpoints*.

As descrições dos serviços hospedados e hospedeiros podem ser consultadas utilizando operações Get do WS-Transfer.

Após o processo inicial de descoberta no qual são lidas as informações sobre o dispositivo hospedeiro, mensagens para obter detalhes sobre os serviços hospedados podem ser enviadas. Mensagens *Get Device Metadata* podem ser usadas para obter informações de atributos como FriendlyName, SerialNumber, FirmwareVersion, Manufacturer, ManufacturerURL, ModelName, ModelNumber, ModelURL e PresentationURL. Mensagens *GetServiceMetadata* são enviadas para obter informações a respeito de um certo serviço, os atributos retornados dependem da implementação deste serviço, a qual não é controlada pelo DPWS.

5.1.4 WS-Eventing

Provê um mecanismo do tipo *publisher/subscriber* que permite que cada cliente solicite a assinatura de notificação apenas dos eventos necessários através de um filtro. São definidos três tipos de *endpoints* o *subscriber*, o *event source* e o *subscriptions manager*. O Subscription Manager armazena as assinaturas de eventos, o *subscriber* é o receptor das notificações e o *event source* é o dispositivo.

Quando o estado do dispositivo se altera, ele envia notificações deste evento para todos os assinantes cujos filtros combinem com o evento ocorrido. Os eventos devem ser classificados previamente para permitir que o cliente assine tipos de de eventos e não apenas eventos isoladamente.

Um serviço capaz de enviar os eventos deve ter o atributo EventSource=true no WSDL e de ser capaz de processar as requisições de assinatura de eventos. O DPWS define um modo de entrega dos eventos e um mecanismo de filtragem que devem ter o comportamento definido no perfil para limitar os recursos exigidos do dispositivo. O modo de entrega *push* é definido na própria WS-Eventing. O filtro definido é o *action filter* que permite ao cliente definir quais os eventos deseja receber. Os dispositivos devem suportar obrigatoriamente apenas xs:duration, não precisando responder ao xs:date já que os recursos necessários para a sincronização de relógio não são adequados aos sistemas embarcados.

Para que um determinado cliente passe a receber as notificações de eventos de um determinado dispositivo ele deve enviar uma mensagem de assinatura destes eventos. Fica a cargo do projetista do sistema a definição de que meio de transporte será usado para entregar as mensagens aos assinantes, podendo ser, por exemplo, usado o TCP com uma conexão por assinante aumentando a confiabilidade ou usando UDP para otimizar o consumo de banda e recursos.

As assinaturas dos eventos são feitas através de mensagens *Subscribe*, esta assinatura tem um tempo de expiração definido. Antes que este tempo expire o cliente deve enviar uma mensagem *Renew Subscription* para renovar a assinatura dos eventos. O cliente pode consultar o estado atual da assinatura através de uma mensagem *Get Subscription Status*. Para deixar de receber as notificações de eventos utiliza-se a mensagem *Cancel Subscription*.

5.1.5 WS-Policy

Esta especificação permite aos serviços que divulguem as suas políticas de uso, as quais devem ser aceitas pelos seus usuários. São comumente definidas características de qualidade de serviço (QoS) e segurança.

5.1.6 WS-Security

Provê um conjunto opcional de mecanismos de segurança fim-a-fim, garantindo confidencialidade, integridade e autenticação. Cada parte do envelope SOAP pode ser criptografada separadamente ou não. Neste trabalho é utilizada uma abordagem diferente para prover segurança à comunicação. Será analisada a comunicação sobre SSL. As vantagens e desvantagens sobre o WS-Security serão tratadas no capítulo 7, quando será apresentada a sua comparação com o SSL no gerenciamento de dispositivos.

5.2 Principais limitações ao uso DPWS

Apesar dos ganhos em modularidade, distributividade, dinamicidade, escalabilidade e reconfigurabilidade, o consumo de recursos ainda é uma limitação para a ampla adoção do DPWS. O estudo de novas formas de representação dos dados sem a perda da expressividade é necessário para reduzir quantidade de recursos necessários para facilitar a aplicação do DPWS em dispositivo extremamente reduzidos.

Sistemas industriais são complexos e heterogêneos nos quais os dispositivos devem se comunicar entre si. Isto faz com que todos os dispositivos devem poder gerenciar os WSDLs de todos os dispositivos. Isto pode causar alguns problemas:

- Explosão de código: para poder tratar todas as operações e mensagens existentes nos WSDLs dos serviços os tratadores destas operações devem ser adicionados ao código.
- Tratadores definidos previamente : a interação entre os serviços se torna limitada à medida que os tratadores das operações devem ser definidos em tempo de desenvolvimento.
- Alto esforço de reprogramação : uma vez que os tratadores são definidos previamente, a inserção de um novo dispositivo necessita de reprogramação de todos os outros.
- Impossibilidade de modificações em tempo de execução : as limitações dos dispositivos impossibilitam que código para o tratamento de reprogramação em tempo de execução sejam adicionados.

6 INTEGRAÇÃO DO DPWS EM UM SISTEMA DE COMUNICAÇÃO SEGURA

O ponto de partida deste trabalho foi a necessidade de substituir o protocolo SNMP na monitoração segura de dispositivos em algumas aplicações. Áreas que necessitavam de monitoração de valores de forma segura ou maior expressividade no controle e configuração dos dispositivos sofriam com a simplicidade do SNMP. Esta simplicidade fez do SNMP o padrão de fato para a monitoração de equipamentos de rede, mas acabou limitando sua representatividade. Apesar da versão 3 do SNMP prover o suporte à comunicação segura, ela nunca teve a mesma aceitação que as versões anteriores.

O baixíssimo consumo de recursos fornecido pelo SNMP não é mais um requisito obrigatório em todas as aplicações de monitoração. A redução de custo dos equipamentos de rede e o aumento da capacidade de processamento dos dispositivos permitem que existam hoje redes dedicadas ao controle e monitoração em diversos ambientes, aumentando a banda disponível para o gerenciamento dos equipamentos.

Foram pesquisadas alternativas ao uso do SNMP, dentre as quais destacavam-se os Web Services e o CORBA. Nas duas abordagens de objetos distribuídos do CORBA e de serviços dos Web Services, os Web Services tiveram um sucesso maior no mercado. Acredita-se que isto se deva a falta de padronização dentre as diferentes implementações do CORBA. Segundo Gisolfi (2001), apesar da interoperabilidade entre implementações CORBA de diferentes fabricantes, níveis mais altos como segurança e transações ainda possuíam pontos não padronizados que impediam a integração das diferentes soluções. Outro fator contra o Corba é a necessidade dos equipamentos controlados operarem em ambientes sob domínios administrativos semelhantes.

Os Web Services tem vantagens na questão da interoperabilidade por serem baseados em padrões abertos da Internet, permitindo a integração entre diferentes fabricantes de forma segura utilizando protocolos bem definidos e acessíveis a todos. Por utilizarem o HTTP como base, os Web Services atravessam facilmente *firewalls* sem necessidade de permissões especiais, sendo necessários apenas os cuidados tomados para o uso de qualquer servidor Web comumente encontrado nas empresas. Além disso, o uso do XML, amplamente difundido como formato de transmissão e armazenamento de documentos, permite a serialização dos dados a serem transmitidos, com a vantagem sobre outros formatos de representação de ter fácil entendimento por carregar consigo a semântica dos dados. O XML é um formato flexível e de fácil extensão, permitindo alcançar a interoperabilidade necessária para a comunicação entre domínios administrativos diferentes. A identificação deste movimento do mercado para a utilização dos Web Services motivou a escolha desta tecnologia como base para este trabalho.

Inicialmente, foram identificadas duas tecnologias de gerenciamento de recursos utili-

zando Web Services, o WS-Management e o WSDM-MUWS (Web Services Distributed Management - Management Using Web Services) definidos pela OASIS e pela DMTF respectivamente. O uso destas tecnologias foi objeto de diversos trabalhos ligados ao gerenciamento (MOURA et al., 2007) (ROHR, 2009), os quais demonstraram que apesar da sobrecarga em relação ao uso do SNMP para o gerenciamento, o desempenho dos Web Services não era proibitivo. Por volta de 2007 ocorreram iniciativas para a integração do WSDM-MUWS ao WS-Management, mas até a elaboração deste trabalho não haviam sido concluídas.

Com o andamento das pesquisas, foi identificado que o DPWS, cuja especificação foi publicada em 2009, é um perfil de uso dos Web Services mais abrangente que o WS-Management, que era focado no gerenciamento de equipamentos de TI. O DPWS tem uma aceitação maior no ambiente industrial e de automação, visto a existência dos projetos já citados anteriormente, SODA, SIRENA e SOCRADES e a não observância de trabalhos relacionando o WS-Management e a automação. Existiram algumas iniciativas de utilização do WS-Management sobre o DPWS, na sua versão inicial antes da padronização por parte da OASIS, como é o caso do WS-Management para DPWS (SOA4D, 2011), mas que está inativo desde 2008.

A partir da escolha do DPWS, foi observado que dentre as duas implementações mais conhecidas, WS4D e SOA4D, já mencionadas anteriormente, o WS4D possuía maior atividade. A decisão em relação às duas implementações foi determinada principalmente pelo desenvolvimento do uDPWS que removia a necessidade de utilização do gSOAP, o qual era usada como base das implementações de ambas as iniciativas. O uDPWS resolve a questão de implementação no dispositivo, mas não teve como o objetivo a segurança dos dados.

Este trabalho visa complementar o que foi feito até o momento e tem como foco principal o estudo dos efeitos do uso da camada de segurança na monitoração de dispositivos utilizando Web Services. Para a sua implementação foram definidas 3 etapas principais:

- Pesquisa a respeito das tecnologias disponíveis, para a comunicação usando Web Services sobre uma camada de segurança, cuja escolha será discutida na seção a seguir.
- Implementação da solução em ambientes simulado e embarcado.
- Execução dos testes e análise dos resultados usando parâmetros que podem facilmente se adaptar em diversos domínios de aplicação.

6.1 Projeto do sistema de comunicação segura

Como mencionado anteriormente, foi decidido fazer a análise da comunicação DPWS segura com dispositivos através da comparação da implementação do uDPWS sem a segurança fornecida no site do uDPWS com a implementação da segurança desenvolvida neste trabalho. Esta forma de análise por comparação facilita a extensão dos resultados deste trabalho para outras plataformas e sistemas operacionais. Dentro dos resultados apresentados no capítulo 9 serão mostradas diferenças entre os resultados em plataformas ARM9 e x86 e entre os sistemas operacionais Linux e Contiki.

A arquitetura definida para o dispositivo uDPWS com comunicação segura pode ser vista na Figura 6.1.

Com o intuito de mensurar a influência do sistema operacional escolhido no desempenho da implementação desenvolvida, foi utilizado o sistema operacional Contiki (DUN-

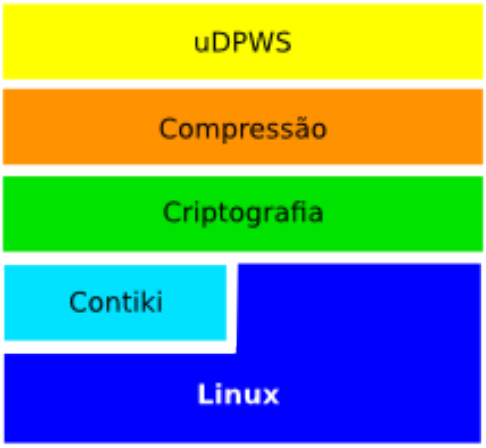

Figura 6.1: Camadas da implementação do dispositivo DPWS

KELS; GRONVALL; VOIGT, 2004), que é focado em dispositivos com recursos extremamente reduzidos, como alternativa ao Linux. O Contiki é melhor descrito no capítulo 7. Foi definido que os testes seriam realizados usando o Contiki sobre Linux e o Linux exclusivamente. Para isto, foi estendida a implementação de exemplo de um serviço de medição de temperatura desenvolvida no uDPWS para utilizar as camadas de segurança e compressão. Após a extensão do serviço foi realizada a conversão das chamadas de sistema para acesso da pilha TCP/IP com Contiki em chamadas equivalentes do Linux, usando sockets POSIX. Isto foi feito para mensurar qual o impacto do Contiki no dispositivo de teste, visto que não fazia parte do escopo deste trabalho o desenvolvimento dos drivers para que o Contiki operasse diretamente sobre o processador ARM9 utilizado. Maiores detalhes sobre a forma como a integração entre o uDPWS e as novas camadas adicionadas serão fornecidos nas seções que seguem.

A implementação descrita neste capítulo tem como possíveis aplicações:

- **Gerenciamento de dispositivos em smart-grids** para medição de geração e consumo de energia e controle de cargas sensíveis.
- **Controle seguro de processos industriais**, como por exemplo, para monitoração de sistemas de de I/O remotos.
- **Aplicações em redes de sensores sem fio**, para monitoração ambiental, principalmente nos casos em que há armazenamento local dos dados.
- **Automação residencial** através da Internet no controle de alarmes, automação de portões e iluminação e no entretenimento.
- Gerenciamento através de **redes administrativamente diferentes**, ultrapassando firewalls inclusive.

Apesar da adição da camada de segurança ser motivada principalmente pelo tráfego de controle e configuração, é esperado que o maior impacto do tamanho das mensagens seja decorrente das mensagens de aquisição de dados de monitoração. Neste trabalho, assume-se que os dados de monitoração devem ser criptografadas pois em ambientes sem fio nos quais os seus dados podem trafegar existe a possibilidade de interferência externa.

A monitoração de dados em processos industriais se dá muitas vezes através da leitura

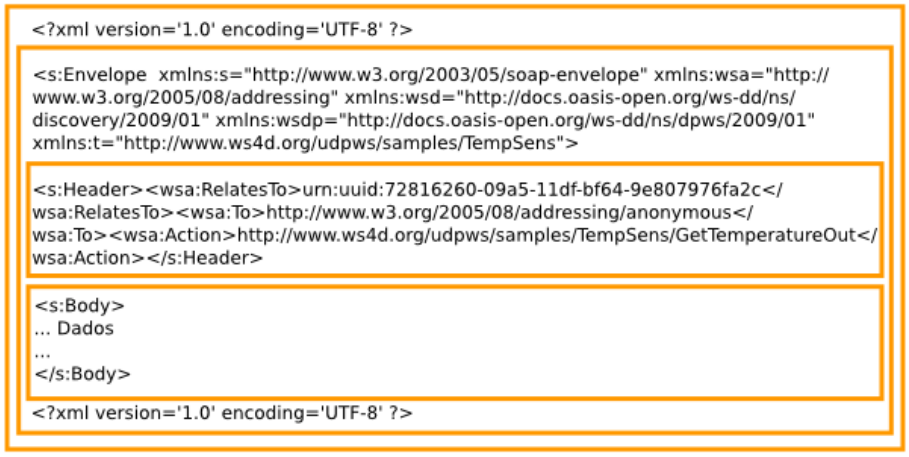

Figura 6.2: Formato básico do arquivo utilizado nos testes.

de valores numéricos, os quais foram utilizados como base dos dados sintéticos utilizados para a avaliação. A partir dessa definição foi montado o formato do arquivo que serviu como base dos testes. O formato do arquivo utilizado é o envelope SOAP definido como formato de mensagens definido pelo DPWS. Sendo assim, o serviço de medição de temperatura proposto no uDPWS foi alterado para fornecer múltiplos valores de temperatura aleatórios. O arquivo ficou com o formato mostrado na figura 6.2.

Os dados foram deixados como uma lacuna na representação da figura 6.2 por necessitarem de uma análise melhor, visto que a forma como eles são representados se reflete diretamente no tamanho das mensagens enviadas.

Os dados são representados entre *tags* XML com a utilização de *namespaces*. Ficando como mostrado abaixo:

<NAMESPACE:NOME_DA_TAG>000000000000</NAMESPACE:NOME_DA_TAG>

A representação de um número é feita utilizando-se algarismos de 0 a 9 em ASCII e caracteres '.' ou ',' e um sinal '-' quando necessário. Apesar de parecer inicialmente óbvio, mencionar esta característica da representação dos dados é importante por ser um dos fatores que ampliam o espaço utilizado pela representação dos dados, uma vez que um inteiro em uma arquitetura de 32 bits consome 4 bytes para ser representado. A escolha do formato de dados equivalente a 32 bits foi motivada pelo fato de que ele tem sido largamente utilizado para representar dados inteiros e em ponto flutuante, podendo ter diversas aplicações como, por exemplo, medição de temperatura, geração e consumo de energia elétrica, vazão de líquidos e fluxo de dados. O mesmo formato pode ser utilizado na medição de diversas outras grandezas para as quais a leitura de valores escalares fornece a informação necessária para o controle de um determinado processo. No caso do XML, um inteiro equivalente pode utilizar de 1 a 11 bytes (ASCII) para representar os números 0 e -2147483647, por exemplo.

Sendo assim, são necessários no mínimo 21 bytes para representar um número inteiro equivalente a 32 bits, como mostrado abaixo:

<t:T>-2147483647</t:T>

ou seja, são necessários 22 bytes para a representação de um inteiro. Os nomes do *namespace* e as *TAGs* foram reduzidos um caractere cada. Isso auxilia na redução da representação mas também elimina uma grande vantagem do XML que é carregar a semântica junto com os dados. O *namespace* pode ser renomeado no cabeçalho, mas o mesmo não pode ser feito com as tags dos dados. O impacto do tamanho das *tags* no

tamanho dos arquivos será discutido um pouco mais na seção que trata das vantagens da compressão.

Devido aos dados fixos no cabeçalho do envelope do SOAP, há uma grande quantidade de dados que precisam ser transmitidos independente da informação útil que será transmitida. Este tamanho varia de acordo com os nomes escolhidos para as funções e espaços de nomes utilizados. No caso do serviço de medição de temperaturas há um custo fixo de aproximadamente 600 bytes para o cabeçalho do envelope DPWS. Isso torna pouco interessante que sejam transmitidos dados isoladamente, sendo assim quanto maior a quantidade de dados transmitidos em cada envelope melhor será o desempenho quando considerado o custo por dado útil.

Apesar dos dados serem representados apenas como inteiros correspondentes a valores de mesmo nome da grandeza, será visto com mais detalhes no capítulo de resultados que esta representação é valida para a grande parte dos casos de uso na monitoração.

6.2 Implementação do dispositivo DPWS

Nesta seção será descrita a implementação do dispositivo de comunicação utilizando o DPWS com segurança, a qual seguiu as etapas abaixo:

- Descrição dos serviços no arquivo WSDL
- Geração do código para atender as requisições no dispositivo
- Integração da camada de segurança
- Adição de compressão para a redução da banda ocupada

6.2.1 Camada de Segurança

Quando se trata de segurança de Web Services, existem duas possibilidades principais: segurança entre a camada de transporte e a de aplicação usando SSL/TLS, e segurança diretamente nas mensagens transmitidas usando WS-Security. As vantagens e desvantagens destas tecnologias tornam cada uma mais adequada a certos cenários, dentre os quais a adição de segurança em dispositivos embarcados.

A principal diferença é que o SSL/TLS criptografa o *payload* e o WS-Security pode ser usado para criptografar ou assinar apenas os atributos necessário dentro da mensagem. A seguir são descritas mais algumas vantagens e desvantagens destas tecnologias:

Vantagens do SSL:

- Amplamente conhecido e utilizado em diversas aplicações e domínios.
- Criptografa toda a conexão facilitando a sua aplicação.
- Com autenticação cruzada tanto o cliente pode autenticar o servidor quanto o inverso durante o *handshake*.

Desvantagens do SSL:

- A conexão deve estar sempre ativa para transportar os dados e tanto o dispositivo quanto o cliente devem ter uma conexão direta como pontos finais da comunicação.
- Como a autenticação é feita apenas no *handshake*, acredita-se que os dados subsequentes são confiáveis, podendo ser desenvolvido algum tipo de ataque que se aproveite disso.
- Caso a criptografia esteja separada dos dados e estes sejam armazenados em filas no servidor, os dados poderão ser lidos por alguns ataques.

Vantagens do WS-Security:

- Facilmente aplicável em arquiteturas orientadas a serviços.
- Criptografia por mensagem e não por conexão.
- As autenticações entre servidor e cliente são sempre feitas através do reconhecimento das chaves usadas para criptografar as mensagens.
- As mensagens ficam o mínimo de tempo descriptografadas, apenas durante o uso da informação, se esta for necessária.

Desvantagens do WS-Security:

- A popularidade do WS-Security não é tão grande quando a do SSL, necessitando de tempo adicional de treinamento.
- A criptografia assimétrica em todas as mensagens prejudica o desempenho.

Analisando as diferenças entre eles, conclui-se que o WS-Security fornece um nível de segurança maior do que o SSL, por ter criptografia assimétrica por mensagem, permitindo que os pacotes sejam roteados e encaminhados, mesmo através de *proxies*. Nas vantagens do WS-Security residem as suas principais limitações. O uso da criptografia assimétrica reduz o desempenho do sistema consideravelmente, conforme será demonstrado a seguir.

A diferença de desempenho do SSL é mais significativa quando a quantidade de requisições aumenta, pois o *handshake* ocorre apenas no início da conexão, reduzindo gradativamente o seu impacto à medida que uma quantidade maior de dados é enviada. No SSL, apenas o *handshake* é feito utilizando criptografia assimétrica no tráfego de dados. Já o WS-Security utiliza criptografia assimétrica para cada mensagem com uma chave para a criptografia e uma para a assinatura. A criptografia assimétrica é sabidamente mais custosa do que a criptografia simétrica. Não há uma forma de evitar a criptografia assimétrica no WS-Security. A vantagem do WS-Security é que a criptografia por mensagem permite evitar ataques conhecidos do SSL, como *Channel Hijacking* e *HTTP Splitting*.

Para que o SSL possa ser utilizado sem comprometer a segurança do sistema é necessário que não haja intermediários entre o dispositivo e o cliente ou que pelo menos os dados sejam encaminhados por um canal seguro, sem o armazenamento dos dados descriptografados em nenhum tipo de *log* ou *buffer* intermediário.

Como é possível selecionar o que é criptografado no WS-Security, é possível permitir que determinadas partes da mensagem sejam criptografados de forma que serão acessíveis para determinadas pessoas enquanto outras serão acessíveis apenas a um grupo mais restrito. Isto exige a utilização de diferentes chaves simétricas.

Como pode ser visto na figura 6.3 (SOSNOSKI, 2010), o tempo de resposta do SSL não passa do dobro do tempo usado para tráfego sem criptografia, para poucas respostas e pouco mais do que uma vez para um número maior de respostas, enquanto o tempo com WS-Security com assinatura e criptografia é mais do que 13 vezes superior para muitas respostas e maior do que 22 vezes maior para poucas respostas.

Quando se trata de sistemas embarcados para aplicações específicas de monitoração e controle, os quais necessitam de otimização do seu desempenho aliada a uma expressividade considerável, podem ser incluídas algumas limitações no domínio da aplicação para que seja possível o uso do SSL/TLS sem comprometer a segurança. A criptografia de todos os dados da conexão não é um problema, pois os dados são dimensionados para o mínimo necessário e provavelmente todos necessitem o mesmo nível proteção. As restrições de conexão ponto a ponto representam um problema menor quando se pode garantir que os dispositivos não necessitem de acesso por meio de proxies, o que não é o ideal mas

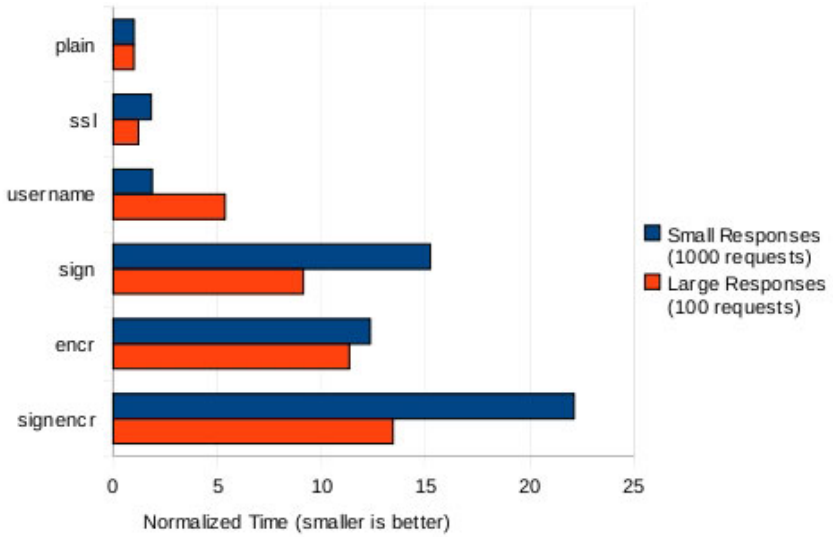

Figura 6.3: Comparativo de desempenho do WS-Security.

atende às necessidades neste caso.

Após a escolha do SSL como tecnologia de criptografia, foi necessário proceder a escolha da implementação a ser utilizada como base do trabalho. Foram selecionadas cinco bibliotecas para análise, cujas características exigidas foram o código aberto e tamanho adequado a sistemas embarcados. São elas: OpenSSL (The OpenSSL Project, 2003), cyaSSL (yaSSL Company, 2012), axTLS (RICH, 2012), polarSSL (B.V., 2012) e matrixSSL (Peersec Networks, 2012).

O OpenSSL é a biblioteca gratuita mais aceita e amplamente utilizada sendo uma referência tanto na criptografia dos dados quanto na geração dos certificados, foi preterida por ser a implementação com o maior consumo de recursos segundo a sua documentação (180kB aproximadamente).

As outras bibliotecas possuem necessidades computacionais semelhantes. Analisando a forma de utilização do PolarSSL e do CyaSSL foi verificado que ambas substituíam as funções *write* e *read* dos sockets POSIX por funções próprias que internamente chamavam as originais. Isto não é um problema na implementação Linux utilizada, mas passa a ser com o TCP/IP do Contiki, que não é compatível com a implementação POSIX tornando-se mais difícil a utilização destas bibliotecas.

Dentre as duas bibliotecas restantes foi escolhida a MatrixSSL por questões de documentação e clareza nos exemplos fornecidos, os quais possuíam uma implementação de cliente/servidor genérico não voltado diretamente ao uso da criptografia para servidor Web HTTP como no axTLS. Sendo assim, por questões de facilidade de implementação escolheu-se a biblioteca matrixSSL, ficando a comparação desta com as demais como sugestão para trabalhos futuros.

6.2.2 Consumo de banda e compressão

Apesar do XML ser aceito como padrão de fato para a representação de dados e intercâmbio de informações através da Internet, a auto descrição, que garante ao XML a flexibilidade, gera o principal problema do seu uso em comunicação de dados que é o tamanho dos dados gerados. Por este motivo muitas estratégias de compressão e representação binária dos dados foram desenvolvidas nos últimos anos.

Como o DPWS utiliza o XML como formato de dados, ele é afetado pela verbosi-

dade do mesmo. A necessidade de redução de tamanho de representação é diretamente associada à compressão. Apesar do uso da compressão ser intuitivo antes de aplicá-lo é necessário avaliar qual o seu impacto no desempenho do dispositivo.

Nesta seção será descrita a avaliação feita para a escolha do algoritmo de compressão a ser utilizado para reduzir a largura de banda necessária para a transmissão dos dados do DPWS exigindo o mínimo de recursos adicionais. O impacto da compressão no desempenho do dispositivo DPWS será mostrado mais adiante nesse documento. Ele traz a análise dos resultados da implementação realizada. A seleção da técnica de compressão a ser testada foi baseada em alguns fatores comuns ao uso da compressão em comunicação de dados e a premissas inerentes aos dispositivos. A partir disto, foram avaliados resultados de estudos realizados com diferentes técnicas de compressão.

A decisão de qual estratégia utilizar depende de alguns fatores:

- Taxa de compressão desejada: dependente da largura de banda disponível.
- Tempo disponível para compressão: afeta diretamente o tempo de resposta do sistema, sendo crucial em sistemas de tempo real e de *streaming* de dados.
- Quantidade de memória disponível: em muitos algoritmos a aceleração do processo de compressão é realizada através da criação de dicionários em memória os quais são consultados no processo de compressão.
- Tempo disponível para a descompressão: em técnicas de compressão assimétricas, nas quais o tempo de compressão e descompressão são diferentes pode ser interessante escolher a técnica que exige menos recursos do dispositivo de menor capacidade computacional.
- Adequação do padrão de compressão ao tipo de dados transmitidos: alguns algoritmos são mais adequados a dados aleatórios enquanto outros a dados que seguem um determinado padrão.

Este trabalho utilizou como premissas para a escolha da técnica a utilizar a redução do processamento, o menor tempo de compressão e o mínimo de memória.

Sakr (2009) avaliou nove diferentes compressores de XML, incluindo compressores de propósito geral e compressores desenvolvidos especificamente para XML, dentre os quais alguns que permitiam a consulta de partes do XML sem a necessidade de descompressão total do arquivo.

Neste estudo, foi demonstrado que o gzip possui a pior taxa de compressão dentre todos os compressores testados, apresentado taxas até 40% maiores que compressores específicos de XML (taxa de 33% para gzip contra 17% para XWRT). Isso seria inicialmente um motivo para descartá-lo, mas como mencionado anteriormente, a taxa de compressão pode ser sacrificada em parte para se obter o balanceamento adequado da razão compressão/consumo de recursos. Neste ponto o cenário se inverte, pois o gzip possui o menor tempo de compressão e descompressão em torno de um quarto do tempo utilizado pelo XWRT que é o melhor algoritmo dentre os compressores específicos.

Augeri (2007) também concluiu que outros algoritmos como XMill e WBXML podem ser escolhas bastante úteis em alguns casos, mas na grande maioria a melhor escolha são os compressores de propósito geral.

6.2.2.1 Escolha do nível de compressão

Foi definido que o mecanismo de compressão utilizado seria o gzip (RFC 1952) por ser de propósito geral e suportado pela RFC 2616 que define a versão 1.1 do protocolo HTTP. Para a realização dos testes foi utilizada a versão 1.2.5 da biblioteca Zlib.

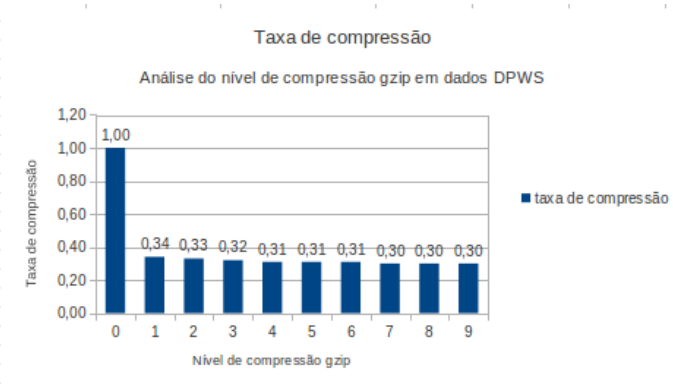

Figura 6.4: Taxas de compressão obtidas com gzip.

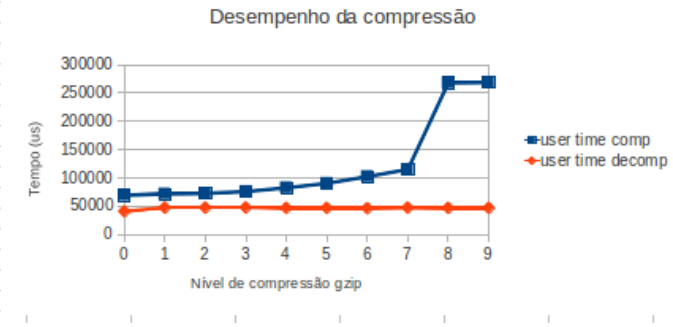

Figura 6.5: Tempo de compressão ARM9.

Os testes realizados para a definição do nível de compressão foram realizados em duas etapas, a primeira avaliou a taxa de compressão obtida com cada nível utilizado. Foi utilizado para a avaliação um arquivo transmitido pelo serviço DPWS composto pelo cabeçalho de envelope gerado pelo SOAP e o corpo formado por 1000 inteiros de 10 caracteres cada, limitados por *tags* e *namespace* com 1 caractere comprimento cada.

As taxas de compressão obtidas são mostradas na figura 6.4. Pode-se notar que para os dados XML utilizados pelo DPWS que seguem um determinado padrão de comportamento usando na prática um conjunto restrito de caracteres a taxa de compressão obtida passa a não variar significativamente com níveis de compressão acima de 4.

A segunda etapa avaliou o custo computacional da compressão, tanto em plataforma ARM9 quanto em x86. Os resultados obtidos são mostrados nas figuras 6.5 e 6.6.

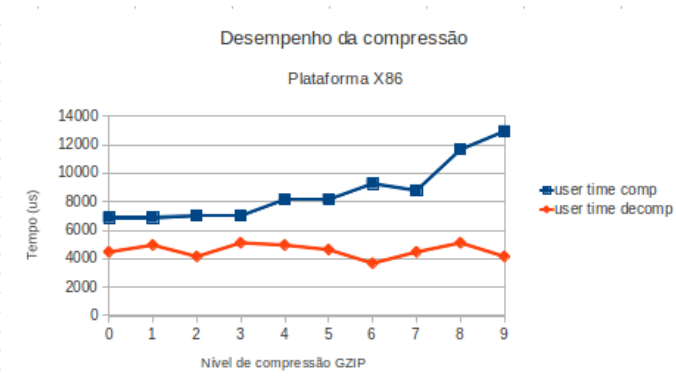

Figura 6.6: Tempo de compressão x86.

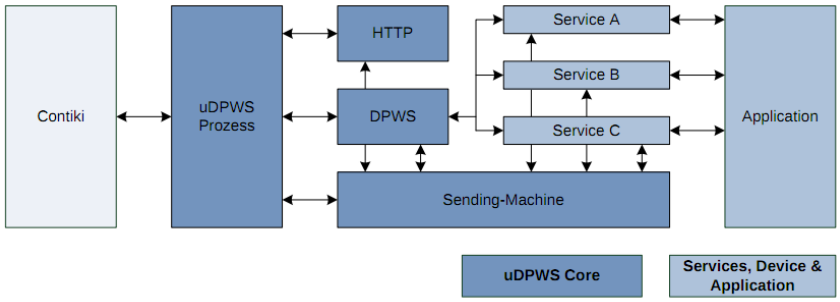

Figura 6.7: Estrutura do uDPWS.

A descompressão não é afetada pelo nível de compressão solicitado ao algoritmo pois apenas utiliza as informações geradas pelo processo de compressão previamente. No caso da compressão, o tempo varia pouco entre os níveis de 0 a 3, com um crescimento aproximadamente linear entre os níveis 4 a 7 e um salto no consumo de tempo nos dois níveis mais altos de compressão. Como o nível de compressão do algoritmo acima de 4 não traz ganho significativo para a redução do consumo de banda foi decidido utilizar este nível nos testes.

Para minimizar o consumo de banda foi utilizada a compressão de todo o envelope SOAP, ao invés da compressão por mensagem. Isto aumenta a taxa de compressão mas retarda o início da transmissão da resposta e exige *buffers* maiores no dispositivo o que em sistemas com recursos extremamente reduzidos pode impedir a sua utilização ou forçar a redução da quantidade máxima de valores transferidos. Foi necessário alterar também o uDPWS que montava as mensagens sob demanda de acordo com o tamanho dos *buffers* disponíveis para ele passar a montar a mensagem completa.

6.2.3 Implementação

Após a definição das tecnologias e bibliotecas a serem utilizadas foi iniciada a implementação em plataforma Contiki operando sobre Linux, resumidos na lista abaixo:

- Implementação do protocolo DPWS: uDPWS.
- Camada de segurança SSL/TLS : MatrixSSL.
- Compressão : Zlib (gzip)

Como mencionado anteriormente, estas tecnologias foram escolhidas visando o melhor desempenho e a facilidade de uso neste trabalho, mas isto não faz delas as únicas opções de implementação.

A base da implementação foi a realizada no projeto do uDPWS. Esta implementação tem a estrutura mostrada na figura 6.7 (LERCHE, 2010).

O uDPWS possui suporte a três plataformas distintas : TelosB, AVRRaven e a que foi utilizada neste trabalho que é a implementação na qual a estrutura mostrada na figura 6.7 é executada sobre o sistema operacional Linux. Neste ambiente com o Linux como hospedeiro, o Contiki não utiliza a pilha TCP/IP do Linux mas sim um *driver* Ethernet que acessa diretamente uma interface TAP, que é uma interface virtual que apesar de se comportar como uma interface de rede local do sistema operacional não tem hardware associado. Uma interface deste tipo pode ter um endereço IP e ser roteada internamente pelo sistema operacional, mas não envia os dados para a rede externa diretamente, necessitando da utilização de algumas ferramentas adicionais, como é mostrado na figura 6.8.

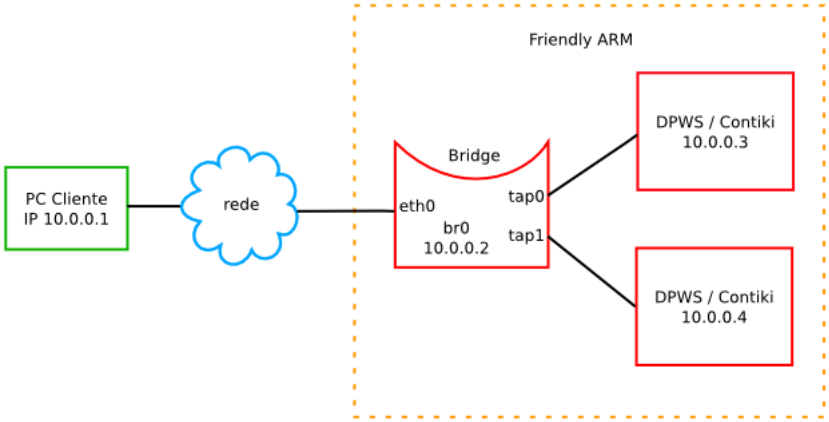

Figura 6.8: Utilização do uDPWS alterado para uso através rede

Esta abordagem permite de forma fácil criar uma ligação entre o cliente em execução no sistema hospedeiro e o dispositivo em execução no Contiki. Isto é adequado a este tipo de comunicação, mas necessita de uma adaptação para o funcionamento quando o cliente não é o sistema hospedeiro, mas sim um cliente em execução em outro hardware conectado através de uma interface de rede física. Para não fugir do objetivo principal do trabalho e execução optou-se por utilizar uma solução que não exigiria a alteração do *driver* de rede usado no Contiki para o acesso à rede. A solução escolhida foi a criação de uma textit-bridge virtual internamente no sistema hospedeiro para redirecionar os dados presentes na interface TAP para a rede física. Em ambiente Linux, o comando *brctl* permite criar uma *bridge* virtual que permite interligar as diferentes interfaces de rede, virtuais ou físicas, do processador na mesma rede Ethernet. Ele depende de configurações específicas do *kernel*, CONFIG_BRIDGE e CONFIG_TUN, sendo necessário recompilar o *kernel* nos casos em que estas opções de compilação não estiverem ativas.

A estrutura lógica do dispositivo é mostrada na figura 6.8. A interface TAP passa a ser vista como uma interface física em uma rede virtual, a qual possui seu próprio endereço IP e é utilizada exclusivamente pelo Contiki como sua interface de rede se comunicando diretamente com o cliente.

Para a adição da camada de segurança foi decidido aplicar a implementação da camada de segurança inicialmente no Linux. Para a implementação neste sistema operacional as chamadas do Contiki foram substituídas por chamadas de acesso a sockets POSIX, como *bind*, *accept*, *connect*, *send* e *recv*. Esta alteração permitiu integrar de forma simplificada o MatrixSSL e a camada de compressão ao uDPWS.

Após a integração e os testes iniciais em plataforma Linux foi realizada a integração do código com a versão original no Contiki. Esta integração precisou de adaptações no código do MatrixSSL pois apesar deste código ser implementado de forma não bloqueante, não há uma chamada de sistema equivalente ao *select*, que permite que a *thread* em execução fique aguardando o escalonamento do sistema. No Contiki, o escalonamento é colaborativo, não existindo portanto chamadas de sockets que preemptem o processo para que o tratador da pilha TCP/IP e o *driver* que acessa a interface de rede TAP possam ser invocados e executar as suas tarefas pendentes. Esta liberação de recursos deve ser feita pela *thread* em execução que libera a CPU através de uma chamada de macro específica. Tanto o envio de dados quanto a sua recepção depende da suspensão da *thread* por certos instantes para que os dados sejam recebidos ou enviados.

7 DESCRIÇÃO DAS TECNOLOGIAS UTILIZADAS NOS EXPERIMENTOS

A partir dos testes preliminares e de questões conceituais das camadas de implementação do dispositivo definidas no capítulo anterior, é possível aprofundar o estudo sobre as implementações de terceiros utilizadas como base deste trabalho. Com base nas particularidades das ferramentas escolhidas é possível melhorar o processo de otimização de tempo de resposta e consumo de banda, cujos resultados são mostrados no capítulo 9.

7.1 WS4D

Dentre os três principais projetos que desenvolveram *toolkits*, WS4D, SO4D e .Net Microframework, foi escolhido o projeto WS4D principalmente por conter a implementação do uDPWS, que é focado no desenvolvimento de dispositivos DPWS em ambientes com recursos muito reduzidos, como sensores sem fio utilizando protocolo 802.15.4. Outros motivos foram a disponibilização dos códigos-fonte *open-source* de toda a implementação e o nível de atividade do projeto, que têm recebido atualizações e novos *toolkits* constantemente.

O WS4D (WS4D, 2011) é uma iniciativa que envolve parceiros acadêmicos e industriais estabelecido por volta de 2006. Esta iniciativa partiu dos resultados do projeto SIRENA, visando integrar os dispositivos nas redes de comunicação e possibilitar a criação de uma comunidade em torno do DPWS para o desenvolvimento de *toolkits*, aumentando a sua disponibilidade e utilização na forma de código aberto, desenvolvendo suítes de teste e promovendo a padronização para garantir a interoperabilidade dentre as diversas implementações existentes.

Ao longo do tempo, o WS4D tem desenvolvido diversos toolkits, dentre os quais:

- WS4D-Axis2 : baseado no Axis2 e voltado à utilização em servidores, este toolkit suporta diversos protocolos de transporte e aplicação como TCP, HTTP, JMS e SMTP. A implementação DPWS foi feita adicionando módulos ao Axis2. Ele serve como ponto comum entre o mundo DPWS e as aplicações Web Services convencionais em servidores. O Axis2 suporta a especificação WS-Addressing nativamente, ficando a cargo do módulo DPWS a adição das outras especificações necessárias. Como o WS4D-Axis2 foi desenvolvido sobre J2SE ele não é adequado a sistemas embarcados, mas pode ser utilizado para o gerenciamento dos dispositivos e outras implementações mais complexas, como os discovery proxies.
- WS4D-JMEDS : o *WS4D Java Multi Edition DPWS Stack* é adequado a diversos tamanhos de implementações desde sistemas embarcados até sistemas onde o consumo de recursos não é uma limitação. Ele define o ambiente *Connected Device*

Configuration para dispositivos com maior capacidade e o *Connected Limited Device Configuration* que é o menor subconjunto do JavaME e pode ser usado nos dispositivos com recursos reduzidos.

- WS4D-gSOAP : implementado como uma extensão do gSOAP, que é um *toolkit* bastante conhecido para o desenvolvimento de aplicações Web Services embarcadas. Ele foi desenvolvido para ter um pequeno *footprint* e Web Services com alta taxa de dados. A disponibilização de um gerador automático de código em C também é uma vantagem do gSOAP. A implementação atual suporta IPv4, tendo o IPv6 como opcional mas ainda em fase experimental.

Além dos toolkits citados acima, foi desenvolvido o uDPWS (LERCHE, 2010). Este toolkit é a base da implementação analisada neste trabalho e por isso será detalhado na seção a seguir.

7.1.1 uDPWS

O uDPWS foi desenvolvido como parte da dissertação de mestrado de Christian Lerche na Universidade de Rostock, Alemanha. Ele faz parte do projeto WS4D e foca na economia de recursos para a aplicação do DPWS em sistemas embarcados com recursos reduzidos. Dentro desta classe de dispositivos se destacam os sensores sem fio que dispõem de algumas centenas kB de memória de processamento e armazenamento disponíveis.

O uDPWS associado ao Contiki (DUNKELS; GRONVALL; VOIGT, 2004) possui a implementação dos protocolos IP, TCP, UDP, HTTP e SOAP e as rotinas de suporte à integração dos sensores diretamente nas redes de dados existentes. Nos testes realizados na implementação original o DPWS foi embarcado em rádios Crossbow TelosB e Atmel AVR Raven além de sistemas operacionais Linux. O uDPWS foi implementado utilizando IPv6 na sua versão para sistemas embarcados 6LowPAN.

Nesta primeira versão em fase de protótipo, foram disponibilizados o básico do DPWS para atender as requisições dos serviços e o WS-Discovery para a descoberta de dispositivos na rede. A implementação do WS-Eventing foi sugerida como trabalho futuro.

Como possuem recursos limitados, os nós sensores não provém serviços e métodos complexos. Para aumentar a representatividade das informações coletadas é necessária a criação de serviços que utilizem as informações coletadas dos serviços dos sensores. Dentre as limitações definidas no uDPWS estão o uso de um subconjunto ainda mais restrito do WS-Addressing (To, Action e MessageID). Nos dispositivos mais restritos, informações a respeito dos serviços como o WSDL são conhecidos estaticamente e não são disponibilizados para consulta em tempo de execução.

Os campos das mensagens são definidos estaticamente e comparados para definir se as requisições devem ser atendidas pelo serviço hospedeiro ou por um dos hospedados.

Por ser compatível com o DPWS, os clientes podem ser implementados com qualquer um dos *toolkits* existentes, ou até mesmo com clientes Web Services convencionais, desde que os serviços hospedados sejam acessados diretamente.

Este trabalho usa como base a implementação para Linux do uDPWS, que consiste no DPWS rodando no sistema operacional Contiki que por sua vez é executado como um *daemon* na plataforma Linux.

7.2 Contiki

O Contiki (DUNKELS; GRONVALL; VOIGT, 2004) é um sistema operacional *open-source* desenvolvido para sistemas baseados no uso de microcontroladores com pouquíssimos recursos computacionais, focando na chamada Internet das Coisas (*Internet of Things*). Em média o Contiki consome por volta de 40kB de ROM e 2kB de RAM. Ele inclui a pilha TCP/IP uIP, que possui um conjunto de primitivas de roteamento para sensores sem fio chamada Rime. Possui suporte ao IPv4 e ao IPv6 com 6LowPAN. Possui alta configurabilidade e uma extensa biblioteca de funções para a comunicação em rede.

O suporte à multitarefa é fornecido pelas chamadas Proto Threads que são implementadas de forma a não necessitar de código adicional em execução para cada thread, apenas dois bytes para a sua identificação. Toda a implementação do Contiki é desenvolvida em ANSI C facilitando a sua compilação para qualquer plataforma que possua um compilador compatível. O controle das *threads* é feito através de MACROS em C que facilitam a programação e otimizam o desempenho do sistema. Para reduzir o custo do escalonamento, apenas a estratégia cooperativa é implementada, exigindo que uma proto thread libere o processador para que as outras possam utilizá-lo.

O Contiki é desenvolvido por um grupo de acadêmicos e empresas liderados por Adam Dunkels do Swedish Institute of Computer Science. Algumas vantagens no uso do Contiki são listadas abaixo:

- IPv6 de baixo consumo: a pilha IPv6 otimizada, combinada a mecanismos de controle de rádio como o ContikiMAC permite o uso de dispositivos alimentados por baterias em redes IPv6 com baixíssimo consumo.
- Simuladores: o desenvolvimento do Contiki é facilitado pelo uso de simuladores como o MSPsim, que permite o completo teste do sistema antes de sua aplicação no hardware.
- Facilidade de interação: por ser compatível com a pilha TCP/IP pelo uso do uIP, o Contiki pode ser facilmente controlado através da ativação de um servidor Web embarcado ou do uso da interface por linha de comando.
- Capacidade de armazenamento: o Contiki possui suporte ao uso de um sistema de arquivos em memória flash, permitindo que sejam armazenados dados no dispositivo em múltiplos arquivos.

7.2.1 uIP

A comunicação TCP/IP do Contiki utiliza a pilha uIP, desenvolvida para o uso em microcontroladores de 8 bits. Suporta tanto IPv4 quanto IPv6. No trabalho de desenvolvimento do uDPWS foi utilizada a comunicação IPv6, neste trabalho será utilizado IPv4 por questões de disponibilidade do protocolo nas plataformas de teste.

A biblioteca uIP é baseada em eventos, podendo utilizar Proto Sockets, que não é o caso da implementação do uDPWS por questões de desempenho.

Não há filas de recepção ou de envio de mensagens, há apenas um *buffer* de bytes para armazenar uma única mensagem. Sendo assim, as mensagens recebidas devem ser tratadas tão logo sejam recebidas. Caso chegue uma segunda mensagem antes da anterior ser tratada, esta segunda mensagem é descartada. Para otimizar o tratamento das mensagens é disparado um evento de recepção de mensagens tão logo a rotina de amostragem do tamanho do buffer de entrada detecte que ele possua dados.

Por questões de otimização, parte da implementação do uIP é baseada em macros que

se assemelham a funções na sua utilização e que podem ser aplicadas tanto para UDP quanto TCP. Algumas das quais são mostradas abaixo:

- uip_udpconnection() : identifica que a conexão utiliza o protocolo TCP.
- uip_newdata () : retorna verdadeiro se novos dados foram recebidos no buffer uip_appdata. A variável uip_conn armazena informações a respeito da conexão atual.
- uip_acked () : identifica que o outro lado da comunicação confirmou o recebimento dos dados.
- uip_rexmit () : identifica que a última mensagem precisa ser retransmitida usando a função uip_send().
- uip_connected() : identifica que uma nova conexão TCP foi estabelecida.
- uip_closed() : retorna verdadeiro se a conexão foi fechada com sucesso, através de uma chamada da função uip_close().
- uip_aborted() e uip_timedout() : identifica que a conexão foi terminada de modo inesperado.
- uip_send(): função que efetivamente envia os dados para a rede.
- uip_datalen(): função que retorna a quantidade de dados disponíveis no buffer de recepção.
- uip_appdata: buffer de recepção.

Estas funções foram apresentadas aqui devido a seu uso pelo uDPWS e pela implementação utilizada neste trabalho.

7.3 MatrixSSL

A biblioteca MatrixSSL (Peersec Networks, 2012) implementa os protocolos TLS/SSL para o uso em sistemas embarcados. Ela contem um módulo completo de criptografia e inclui criptografia simétrica e assimétrica, com diversas suítes de cifragem disponíveis. O MatrixSSL é distribuído em duas versões, open-source e comercial, a versão comercial possui uma quantidade maior de suítes de cifragem (com mais opções de uso de TLS e uso de Diffie-Hellman para a troca de chaves assimétricas).

São suportadas as versões SSL 3.0, TLS 1.0 e 1.1. O MatrixSSL é independente de protocolo de transporte, mas a maioria das aplicações se concentram no uso de HTTP sobre TCP.

A adição da biblioteca a um aplicativo demanda cerca de 50kB, incluindo algoritmos de criptografia RSA, ECC, 3DES, AES, ARC4, SHA1, MD5, RC2, os quais compõem suítes de cifragem RC4-MD5, RC4-SHA, DES-CBC3-SHA, AES128-SHA e AES256-SHA, dentre os quais alguns apenas disponíveis na versão comercial. Suporte a cache de conexões, renegociação de chaves e outras estratégias para a melhoria do desempenho estão disponíveis. Na versão open-source, os certificados X.509 devem ser gerados por outra ferramenta pois o MatrixSSL não disponibiliza a geração de chaves. A melhor forma de compreender como o MatrixSSL é integrado à aplicação é através da documentação do próprio código e dos exemplos fornecidos.

Existem diversas implementações do MatrixSSL para sistemas operacionais como VxWorks, uClinux, eCos, FreeRTOS, ThreadX, WindowsCE, PocketPC, Palm, pSOS, SMX, BREW, MacOS X, Linux e Windows, em diversas plataformas como ARM, MIPS32, PowerPC, H-8, SH3, i386 e x86-64.

Por ser desenvolvida em Ansi C, o Matrix SSL pode ser integrado ao código do uDPWS sem adaptações na estrutura interna do código, necessitando da adaptação dos códigos de exemplo de comunicação cliente/servidor fornecidos e nas rotinas de acesso às funções de sistema operacional para a criação de sockets, controle de concorrência e alocação de memória. A integração do MatrixSSL ao código do uDPWS será descrita na seção que descreve o trabalho desenvolvido.

O MatrixSSL disponibiliza de forma gratuita a maior parte dos algoritmos de troca de chaves, criptografia e autenticação de mensagens definidos na RFC5246 que especifica a versão 1.2 do protocolo TLS.

- **Key Exchange** : RSA, DH_DSS, DH_RSA, DHE_DSS, DHE_RSA, DH_anon
- **Cipher** : RC4_128, 3DES_EDE_CBC, AES_128_CBC, AES_256_CBC
- **Message Authentication code** : MD5, SHA, SHA256

As exceções ao suporte do Matrix SSL são os algoritmos de trocas de chaves baseados em Diffie-Hellman (iniciados em DH) e os algoritmos de cifragem e autenticação de mensagens de 256 bits. Estes algoritmos são suportados apenas na versão comercial.

7.4 Plataforma alvo: Friendly ARM

Para a validação da solução proposta foi escolhida a plataforma FriendlyARM Mini2440 baseada no processador Samsung S3C2440A ARM920T de 400MHz. O kit de desenvolvimento do Mini2440 possui as seguintes características adicionais:

- 64 MB de memória RAM 32 bits.
- 256 MB de memória Flash
- conexões serial, USB, Ethernet
- interface para cartão SD
- conexão de periféricos para áudio, LCD *touchscreen*, além de LEDs e botões para facilitar o desenvolvimento.
- suporte aos sistemas operacionais Windows CE, Linux e Android
- suporte a depuração usando JTAG

Com estas características o kit escolhido supera em muitas vezes a capacidade computacional dos rádios utilizados nos testes durante o desenvolvimento do uDPWS: o TelosB e o AVR Raven (LERCHE, 2010). Por este motivo, este trabalho foi baseado na comparação entre os resultados com segurança habilitada e sem segurança habilitada. Esta comparação permite traçar um paralelo com outras plataformas.

Foi definido como sistema operacional a ser utilizado o Linux devido ao seu amplo uso na área de sistemas embarcados atualmente e pelo suporte dado a ele no FriendlyARM.

8 DEFINIÇÃO DE MÉTRICAS E PLANEJAMENTO DOS EXPERIMENTOS

Este capítulo tem como objetivo descrever a metodologia utilizada nos experimentos e os seus objetivos. Os resultados destes experimentos são mostrados no capítulo seguinte.

Para a execução dos testes foi utilizada uma combinação de *scripts shell e expect*, os quais foram executadas a partir do computador cliente para acessar via *telnet* o dispositivo, disparar o serviço hospedeiro com os parâmetros correspondentes à execução desejada e localmente disparar as consultas no cliente. Nos casos em que a execução utilizava o Contiki foi necessário configurar a *bridge* interna como discutido anteriormente e mostrado na figura 6.8 utilizando o comando **brctl**, que cria uma interface TAP conectada a uma bridge virtual, antes de realizar as consultas a partir do cliente.

Na tabela 8.1 é mostrada a lista dos experimentos a serem executados, descrevendo os seus principais objetivos. Ela foi baseada nas necessidades existentes para o projeto de um sistema embarcado com comunicação em rede, privilegiando os parâmetros que influenciam na especificação de requisitos dos dispositivos e no seu impacto na rede. Apenas o experimento que mede o tempo de execução das etapas e a análise estatística serão melhor descritos nas seções a seguir, visto que o *footprint* é uma medição bastante utilizada e os demais medem o tempo de resposta do dispositivo a partir da variação de parâmetros.

8.1 Medição dos tempos das etapas da comunicação

Para definir a influência dos parâmetros adicionados ao uDPWS neste trabalho foram identificadas as etapas envolvidas na comunicação segura com compressão, as quais são mostradas na figura 8.1.

A identificação destas etapas permite definir pontos de medição intermediários melhorando a avaliação das otimizações que podem ser feitas afim de reduzir o tempo de resposta de uma consulta completa ao dispositivo. A montagem da requisição pode ser substituída por uma *string* fixa na qual são alterados apenas os dados de endereçamento dos serviços e o tempo de utilização da informação depende fortemente do uso que o cliente fará dela. Isto torna a análise destas etapas pouco relevante do ponto de vista genérico, sendo desconsideradas nesse trabalho.

As etapas de criptografia, envio e recepção foram medidas em conjunto, pois a medição é feita nos casos com e sem criptografia habilitada e com isso os tempos de cada etapa podem ser deduzidos a partir destas duas medições. Outro fato que motivou esta escolha é que o atraso causado pela comunicação na rede é muito menor do que o tempo consumido pela criptografia dos dados.

Experimento	Objetivo
Footprint	Medir o espaço ocupado pelo programa na memória de armazenamento. Importante para o dimensionamento de recursos na etapa de projeto do dispositivo
Consumo de Banda	Mensurar o impacto na rede de dados e a forma como ele é afetado pela variação dos parâmetros do processo, como segurança e compressão. Como a quantidade de dados transmitidos e o *overhead* dos cabeçalhos dos protocolos utilizados são constantes, o consumo de banda pode ser calculado se forem fornecidos os valores dos parâmetros utilizados.
Tempo de Execução por Etapas	Medir o tempo de execução de cada etapa para facilitar a otimização no processo através da identificação dos pontos de maior consumo de recursos, identificando aquelas que possuem maior consumo.
Tempo de Resposta por Sistema Operacional	Demonstrar a diferença de desempenho entre um sistema operacional com suporte a interrupções (Linux) e escalonamento preemptivo e outro com escalonamento cooperativo e com tratamento de pacotes por tempo (Contiki).
Tempo de Resposta por Suíte de Criptografia	Analisar o impacto no tempo de resposta que tem a alteração da suíte de criptografia com o tráfego DPWS. Permite alternar entre algoritmos de cifragem de bloco, como o AES e o 3DES, e de fluxo, como o RC4.
Tempo Consumido por Dado Útil	Fornecer outra visão sobre o consumo de recursos, demonstrando as vantagens de transmitir uma quantidade maior de dados em um mesmo pacote para minimizar o *overhead* gerado pelo envelope SOAP.
Tempo de Resposta por Janela de Dados	Mostrar o impacto da redução no tamanho dos buffers internos do dispositivo e no aumento da quantidade de mensagens transmitidas.
Análise Estatística	Inferir quais os parâmetros do processo tem significância estatística, realmente afetando o tempo de resposta. A suposição de que para determinadas quantidades de dados úteis a compressão não prejudica o desempenho no tráfego seguro é comprovada com esse estudo. Esta análise estatística foi baseada no Planejamento de Experimentos, que é uma metodologia consolidada para a otimização de processos em diversas áreas e que será descrita na seção 8.2.

Tabela 8.1: Experimentos realizados.

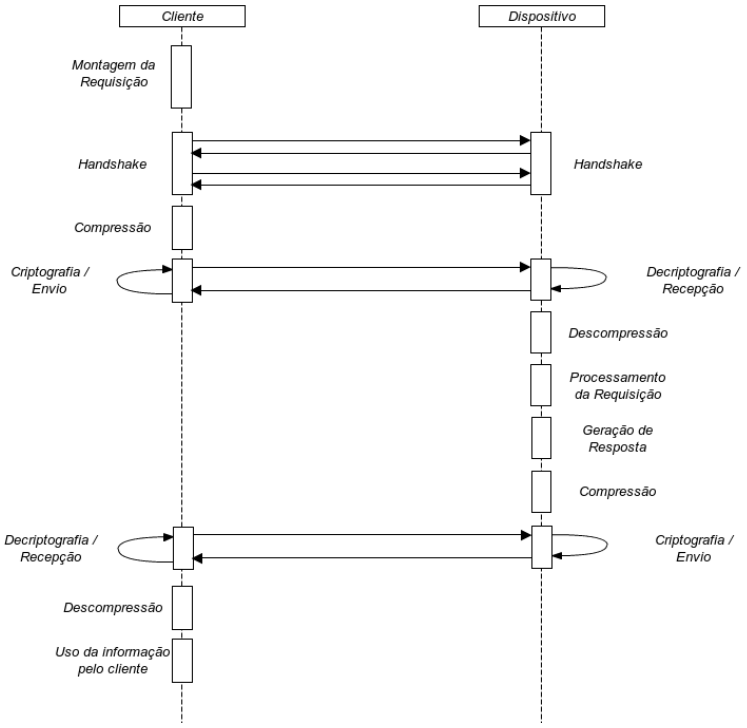

Figura 8.1: Etapas da comunicação entre cliente e dispositivo.

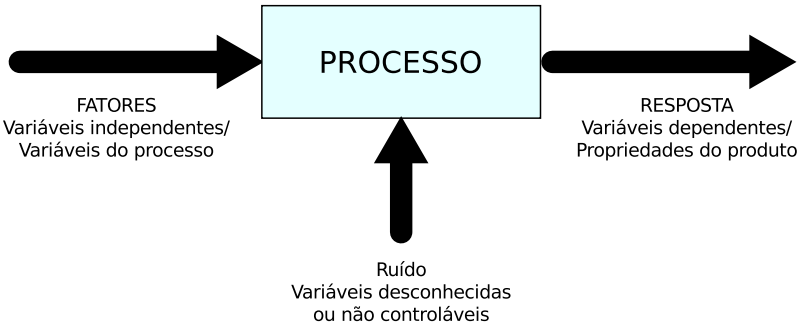

Figura 8.2: Variáveis de um experimento.

8.2 Análise estatística

Por ser um trabalho focado em mensurar o desempenho da camada de segurança em um dispositivo utilizando DPWS, tornou-se importante a utilização de um método científico de avaliação dos resultados obtidos, que validasse hipóteses formuladas a respeito do comportamento do DPWS. Dentre as técnicas existentes, escolheu-se o planejamento de experimentos por se adaptar bem ao tipo de avaliação necessário.

O planejamento de experimentos (MONTGOMERY, 2008) é uma metodologia antiga, inicialmente aplicada à agricultura, posteriormente difundida a diversas áreas como biologia, engenharia e mais recentemente à computação. É fortemente baseado em conceitos estatísticos, focado na otimização do planejamento, execução e análise de experimentos. O seu uso permite que se estruture uma sequência de ensaios para traduzir em forma de equação o impacto das variáveis (fatores) no resultado. O resultado dos experimentos e a sua análise permitem se obter a confirmação estatística se um conjunto de dados é ou não afetado de forma significativa por alguma das variáveis de um determinado

processo.

No planejamento de experimentos 2^k supõe-se que os fatores tem níveis fixos com erro completamente aleatorizado e as hipóteses de normalidade podem ser satisfeitas. O seu uso tem a vantagem de simplificar a análise de um determinado problema de otimização, sendo útil nos estágios iniciais da pesquisa principalmente quando há muitos fatores a serem investigados, pois permite identificar os mais significativos.

Para aplicar a metodologia do planejamento de experimentos, é necessário estar familiarizado com a sua terminologia, a qual difere um pouco da utilizada na computação. O experimento é realizado em um sistema, que pode ser qualquer produto, processo ou serviço, o qual é avaliado por indicadores de desempenho, que representa características de qualidade do sistema resultantes da operação do mesmo.

Um experimento está planejado quando estão definidos:

- Unidade experimental : ... Ex.: um programa ou processo.
- Variável em análise : ... Variáveis de entrada das quais se deseja estudar os efeitos. Ex: quantidade de dados utilizados.
- Tratamentos em comparação : define-se qual será o parâmetro de saída do sistema e a forma como ele será medido.
- A forma como os tratamentos serão designados às unidades experimentais.
- Definir quantas repetições (execuções) serão realizadas sobre o mesmo cenário.
- Variabilidade: variação existente nos resultados das repetições de um experimento mantendo-se fixos os parâmetros de entrada.
- Características de qualidade : características importantes para o usuário.
- Variáveis de resposta : aspectos do produto que podem ser medidos permitindo quantificar as caraterísticas de qualidade.
- Parâmetros de processo : todas as variáveis que podem ser alteradas tendo efeitos sobre a variável de resposta.
- Fatores controláveis : parâmetros do processo que podem ser alterados de forma controlada durante o experimento.
- Fatores constantes : são parâmetros de processo que podem ser controlados mas que não são alterados durante o experimento.
- Fatores não controláveis (Ruído) : são variáveis que alteram o resultados mas que não podem ser controlados. Dependendo da intensidade, quantidade e variabilidade, os fatores de ruído podem inviabilizar o experimento.

Na figura 8.2 são mostrados os valores de entrada e saída de um experimento a ser avaliado com o planejamento de experimentos. Na figura, o processo pode ser um programa, uma tarefa do dispositivo ou mesmo um processo complexo formado pela composição de diversos serviços. As variáveis do processo (entrada) são aquelas sobre as quais o realizador do experimento tem controle direto, podendo atribuir os valores desejados, as variáveis de resposta são os valores medidos para avaliar os resultados do processo. O ruído ou variáveis não controláveis são aquelas sobre as quais não se tem controle direto e que podem afetar diretamente a confiabilidade do resultado. O ruído deve ser tão pequeno quanto possível, mas em muitos casos ele pode ser apenas estimado após a execução dos testes.

A execução dos testes foi feita através de scripts variando os seguintes parâmetros de entrada do dispositivo e do cliente:

Fator	Valores possíveis
Dados úteis transmitidos	de 1 a 1000 valores inteiros com incremento de 50 valores
Janela de transmissão	250, 500, 1000 e 1400 bytes
Tráfego seguro	habilitado ou desabilitado
Compressão	habilitada ou desabilitada
Plataforma	ARM9 ou x86
Sistema operacional	Linux ou Contiki

Tabela 8.2: Valores dos fatores definidos para o experimento.

- **Janela de dados transmitidos** : define o tamanho do buffer utilizado para a transmissão dos valores. Na grande maioria dos casos é necessário mais do que um quadro Ethernet para a transmissão de todos os dados. Entretanto, em alguns dispositivos embarcados não é possível ter *buffers* de tamanho muito grande. Nos testes foram utilizados os valores de 250, 500, 1000 e 1400 bytes para o tamanho do buffer.
- **Quantidade de valores transmitidos** : este trabalho tem como foco principal a utilização do DPWS para a monitoração de grandezas em equipamentos industriais para o controle de processos. Como alternativa para a transmissão dos valores destacam-se duas abordagens: consultar os valores instantâneos obtendo uma atualização imediata dos dados, que pode ser interessante para valores críticos, ou armazenar temporariamente os valores menos críticos ou que tenham finalidade histórica para consulta posterior. Dados deste tipo podem ser transmitidos em blocos maiores com frequência menor. Nos testes foi realizada a variação dos dados de 1 a 1000 inteiros de 10 dígitos com sinal. O tamanho dos dados é medido em dígitos devido à representação do XML ser textual e não ter uma limitação devido ao formato representado. Em teoria o XML pode representar qualquer número real, mas neste trabalho foram limitados a 10 dígitos por ser uma representação equivalente a inteiros de 32 bits encontrados comumente nas arquiteturas atuais.
- **Tráfego seguro** : para ser possível mensurar o impacto da camada de segurança sobre tráfego do DPWS os testes foram realizados em duas situações, com o tráfego seguro habilitado e desabilitado.
- **Compressão** : com intuito semelhante à camada de segurança, os testes foram executados com a camada que incluía a compressão habilitada e desabilitada.
- **Arquitetura** : como mencionado anteriormente os testes foram executados em plataformas x86 e ARM. Os resultados apresentados foram gerados para ambas arquiteturas sempre que foi identificado que a diferença de plataforma poderia interferir nos resultados.

Os fatores dos experimentos a serem realizados e os valores possíveis que eles podem assumir são resumidos na tabela 8.2

A partir da definição dos fatores dos experimentos a serem realizados, percebeu-se que os mesmos precisariam ser analisados de forma particionada, pois a quantidade de combinações diferentes dificultaria o entendimento dos resultados obtidos. A quantidade de combinações é definida pela pelo produto da quantidade de níveis possíveis para cada fator (equação 8.1).

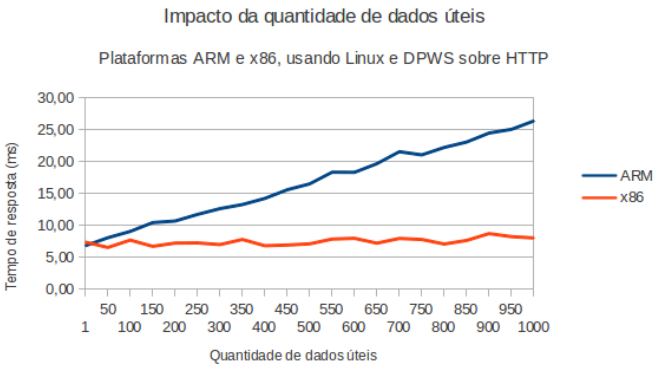

Figura 8.3: Análise preliminar do tempo de resposta a partir dos dados úteis.

$$\begin{aligned} QtdeTestes &= DU * TJ * Seg * Comp * Plat * SisOp \\ QtdeTestes &= 21 * 4 * 2 * 2 * 2 * 2 = 1344 \end{aligned} \tag{8.1}$$

Onde *DU* é a quantidade de dados úteis, *TJ* os tamanhos da janela de transmissão utilizados, *Seg* o uso ou não de segurança, *Comp* o uso ou não de compressão, *Plat* as plataformas utilizadas e *SisOp* os sistemas operacionais utilizados. A quantidade de dados úteis corresponde à quantidade de valores monitorados que são respondidos a cada requisição e que são vistos pelo usuário como informação útil, como por exemplo um valor de consumo de energia em um determinados instante ou a vazão de um líquido em um processo.

A quantidade de combinações de níveis e fatores possíveis dificultava a análise estatística. Devido a isso, foi necessária uma análise prévia para reduzir esta quantidade e permitir a aplicação de experimentos 2^k. A ideia destes experimentos é ter fatores com dois níveis. A quantidade de dados úteis e o tamanho de janela possuíam quantidade maior do que dois níveis. No caso do tamanho de janela, foi decido remover os valores intermediários deixando no experimento, os níveis de 250 e 1400 bytes DPWS por pacote.

Para a quantidade de dados úteis foram realizados testes preliminares que demonstraram que a variação do tempo de resposta poderia ser considerada aproximadamente linear com relação a variação apenas deste parâmetro, como pode ser visto na figura 8.3, que mostra os resultados dos testes para tamanho de janela 1400 bytes, usando DPWS sobre HTTP.

A variação da quantidade de dados úteis não deve ser encarada como um parâmetro binário de mínimo ou máximo pois ela depende fortemente das necessidades da aplicação. Foi decidido utilizar a quantidade de inteiros como um fator fixo nas análises, realizando os estudos para diferentes níveis. Foram selecionados os níveis de 1, 50, 100 e 1000 valores úteis para o estudo.

Dado que o foco deste trabalho é o uso do DPWS em sistemas embarcados, foi definido que os estudos estatísticos seriam realizados para a plataforma ARM, que é menos afetada por outros processos em execução, cuja utilização de recursos pode reduzir a confiabilidade dos resultados, conforme foi observado em testes preliminares.

Após estas definições, a quantidade de testes foi reduzida significativamente sendo definida pela equação 8.2.

$$\begin{aligned} QtdeTestes &= TJ * Seg * Comp * SisOp \\ QtdeTestes &= 2 * 2 * 2 * 2 = 16 \end{aligned} \tag{8.2}$$

O resultado calculado é a quantidade de testes para cada repetição do experimento. Foi definido que seriam utilizadas 10 repetições do experimento para aumentar a sua representatividade estatística. Os dados úteis foram omitidos nesta fórmula porque a quantidade de dados úteis é em parte definida pela aplicação na qual o dispositivo será utilizado. Esta aplicação define a taxa de atualização dos dados e qual a quantidade de dados que estará disponível nessa taxa de atualização. Sendo assim, é interessante saber que tipo de impacto tem uma determinada quantidade de dados úteis, mas não é mandatório para poder realizar o estudo. Neste trabalho foram realizados experimentos para 4 quantidades diferentes de dados úteis, totalizando 640 execuções.

9 ANÁLISE E RESULTADOS

Este capítulo tem como objetivo apresentar os resultados dos experimentos definidos no capítulos anterior e discutir os principais aspectos relacionados aos valores encontrados. Nos testes foi desconsiderado o tempo de geração dos certificados pois estes podem ter compilação prévia e serem armazenados estaticamente, principalmente se tratando de sistemas que devam envolver baixo consumo. Testes preliminares demonstraram que o tempo de handshake é constante para cada sistema operacional, sendo independente dos demais parâmetros.

9.1 Footprint

Na avaliação do footprint necessário para a aplicação foi utilizada a compilação do Contiki para Linux em plataforma ARM9. Os tamanhos dos executáveis gerados a partir de variações nos parâmetros de compilação são mostradas na figura 9.1. Os gráficos mostram também o tamanho de cada bloco funcional em cada caso.

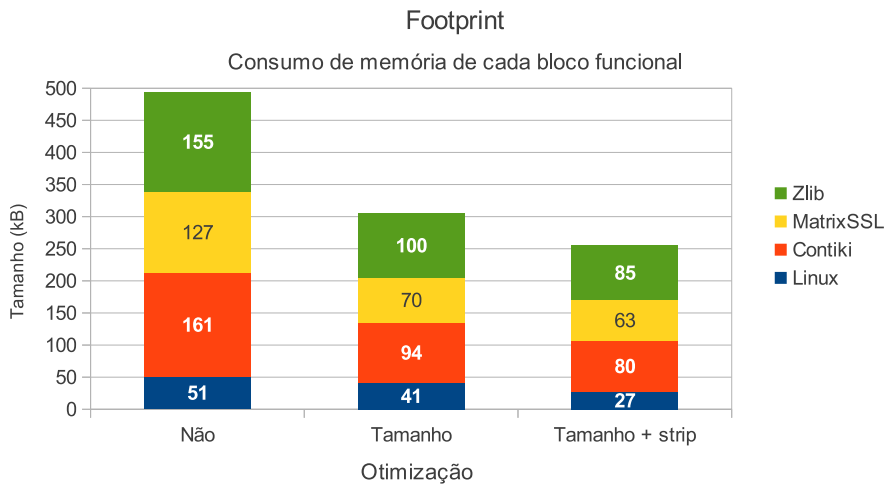

Figura 9.1: Tamanho do aplicativo de acordo com os componentes habilitados.

Os resultados foram gerados com e sem a otimização voltada à redução de tamanho gerada pelo *gcc*. Outra medição realizada foi o tamanho arquivo binário após a execução do comando *strip*. Este comando fornecido pelo GNU permite remover os símbolos e trechos de código não utilizados reduzindo ainda mais o tamanho do programa gerado pelo compilador.

Os valores obtidos na prática foram superiores aos fornecidos na documentação das

implementações utilizadas. Isto se deve ao fato das documentações apresentarem como valores mínimos obtidos usando o mínimo possível de recursos. Nesta aplicação entretanto foi utilizada a configuração padrão fornecida com as bibliotecas que é a mais usada no caso geral.

Como os experimentos realizados tem foco no desempenho e economia de recursos foi utilizada a versão de menor tamanho na geração dos resultados apresentados no decorrer deste capítulo.

9.2 Consumo de banda

O consumo de banda pode ser calculado previamente, pois varia apenas em função dos fatores que são conhecidos previamente. A equação de cálculo do consumo de banda aproximado, permite avaliar antecipadamente qual o impacto que certas técnicas tem sobre a banda utilizada sem a necessidade de simulações que podem consumir tempo que muitas vezes não está disponível. As equações 9.1 a 9.4 demonstram o cálculo da banda utilizada pela requisição e as equações 9.5 a 9.8 demonstram o cálculo referente à resposta. Estas equações levam em consideração os pacotes SYN, FIN e ACK necessários para a conexão TCP e para a confirmação de recebimento dos pacotes, os quais tem maior impacto a medida que a quantidade de pacotes aumenta.

$$TamDadosReq = TamReq * TaxaComprReq \tag{9.1}$$

$$QtdePacReq = TamDadosReq/TamJanela \tag{9.2}$$

$$TotalSemSegReq = TamDadosReq + QtdePacReq * (2 * TamCabec) \tag{9.3}$$

$$TotalReqSeg = TamDadosReq + QtdePacReq * (MAC + 2 * TamCabec) \tag{9.4}$$

$$TamDadosResp = (TamResp + QtdeVal * TamValor) * TaxaComprResp \tag{9.5}$$

$$QtdePacResp = TamDadosResp/TamJanela \tag{9.6}$$

$$TotalSemSegResp = TamDadosResp + QtdePacResp * (2 * TamCabec) \tag{9.7}$$

$$TotalRespSeg = TamDadosResp + QtdePacResp * (MAC + 2 * TamCabec) \tag{9.8}$$

onde:

TamReq, é o tamanho da requisição DPWS com os bytes do envelope SOAP.
TaxaComprReq, é a taxa de compressão média da requisição.
TamDadosReq, é o tamanho dos dados utilizados pela requisição DPWS.
TamJanela, é o tamanho de janela de dados utilizada na requisição DPWS.
QtdePacReq, é a quantidade de pacotes da requisição DPWS.
TamCabec, é o tamanho do cabeçalho TCP/IP, 66 bytes em média.
TotalSemSegReq, é o tamanho total em bytes quando a segurança não é utilizada.
MAC, é o tamanho do *Message Authentication Code* das mensagens SSL/TLS.
TotalReqSeg, é o tamanho total em bytes com comunicação segura habilitada.
TamResp, é o tamanho do cabeçalho do do envelope SOAP + DPWS.
QtdeVal, é a quantidade de valores numéricos transmitidos.
TamValor, é o tamanho de um valor numérico no formato XML, foi usaso 21 bytes.
TaxaComprResp, é o tamanho dos dados da resposta após a compressão.
QtdePacResp, é a quantidade de pacotes da resposta.

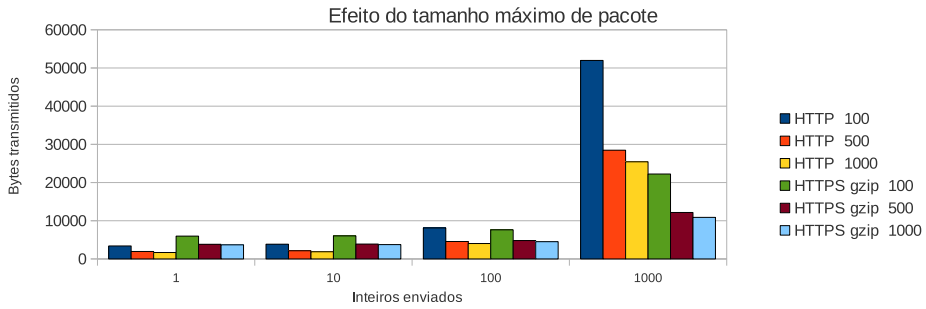

Figura 9.2: Efeito do tamanho do pacote sobre a quantidade de dados transmitidos.

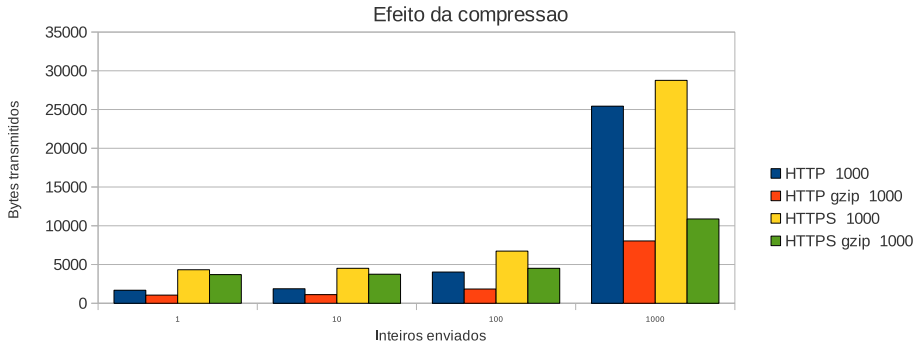

Figura 9.3: Efeito do tamanho do pacote sobre a quantidade de dados transmitidos.

TamDadosResp, é o tamanho dos dados da resposta DPWS.

TotalSemSegResp, é o total em bytes com segurança desabilitada.

TotalRespSeg, é o total em bytes com comunicação segura habilitada

A esses dados somam-se os 6 pacotes de SYN, FIN e ACK, com 66 bytes em média cada.

A figura 9.2 mostra o efeito do tamanho da janela sobre a quantidade total de bytes transmitidos. São mostradas as quantidade de bytes necessários para 3 tamanhos de janelas, 100, 500 e 1000 valores numéricos por pacote. Foi utilizado DPWS sobre HTTP sem compressão e sobre HTTPS com compressão. Nota-se que para uma quantidade de valores menor do que 100 mesmo com a compressão os dados HTTPS necessitam de banda maior ou igual aos dados HTTP. Quando a quantidade de valores ultrapassa 100 valores por consulta o tamanho dos dados passa ser menor para os dados seguros comprimidos. Se fossem comparados os dados entre HTTP e HTTPS, ambos comprimidos, o HTTP ocuparia menos banda. Entretanto, como será visto mais adiante nesse documento, o uso de compressão reduz o desempenho significativamente no tráfego HTTP, mas pouco altera o tráfego HTTPS percentualmente, já que o incremento de segurança impacta em redução considerável de desempenho.

Na figura 9.3 é mostrado o efeito da compressão nos quatro tipos de tráfego testados neste trabalho, HTTP e HTTPS com e sem compressão. Neste caso é mostrado o comportamento esperado no qual a taxa de compressão média é aplicada a cada um dos tráfegos. Estes dados foram testados com tamanho de janela igual a 1000.

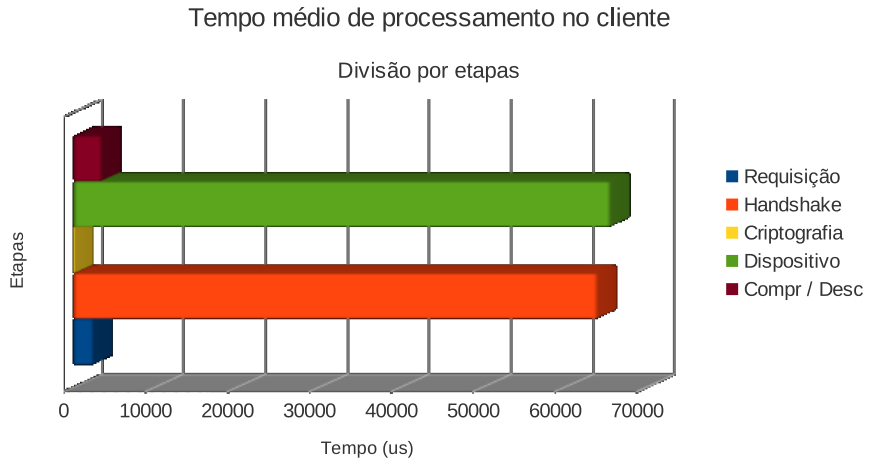

Figura 9.4: Tempo de execução no cliente por etapas ($\sigma = 3600\mu s$).

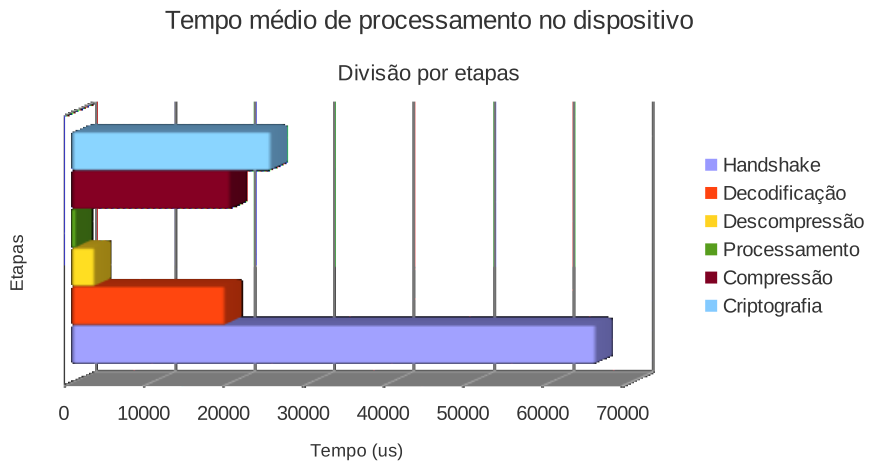

Figura 9.5: Tempo de execução no dispositivo por etapas ($\sigma = 4093\mu s$).

9.3 Análise do tempo médio de execução por etapas

Como planejado na seção 8.1, foram medidos os tempos médios de execução por etapas no cliente e no dispositivo, cujos resultados são exibidos nas figuras 9.4 e 9.5

Os tempos mostrados nestes dois gráficos está diretamente relacionado às etapas mostradas na figura 8.1, mas com algumas simplificações na quantidade de etapas. No cliente o tempo de compressão e descompressão foram somados para representar o seu impacto total. O tempo do dispositivo representa o tempo desde o envio da requisição para o dispositivo até a resposta ser recebida pelo cliente, representando aproximadamente a soma dos tempos de todas as etapas do dispositivo excetuando-se o *handshake*. No dispositivo, as etapas de processamento da requisição e geração da resposta foram representadas como uma única etapa chamada de processamento.

A etapa mais significativa em termos de tempo de execução é o *handshake*. Este comportamento era esperado, uma vez que no *handshake* ocorre a negociação das chaves de criptografia simétricas que serão usadas durante a comunicação dos dados entre cliente e dispositivo. Esta negociação é feita utilizando-se certificados e criptografia assimétrica

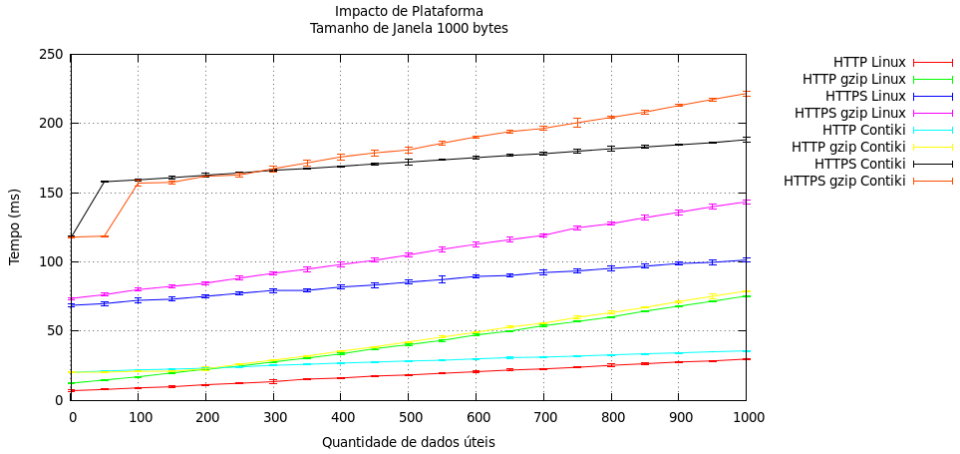

Figura 9.6: Impacto da plataforma usando janela de 1000 bytes.

utilizando algoritmos como o RSA e o Diffie-Hellman Efêmero. Como mencionado anteriormente a criptografia assimétrica é algumas vezes mais lenta do que a simétrica.

O *handshake* só precisa ser realizado obrigatoriamente no estabelecimento da conexão, podendo ser desconsiderado o seu impacto na comunicação, pois o seu tempo de execução se torna menos significativo à medida que a quantidade de mensagens que trafegam por esta conexão segura aumenta.

9.4 Impacto do sistema operacional

Nas execuções no Contiki o desempenho foi afetado pelo fato do tratamento dos pacotes da pilha TCP/IP ser feito por amostragem no buffer controlada por tempo. Testes práticos mostraram que a redução deste tempo a valores que otimizariam o tratamento dos pacotes não pode ser utilizado pois impede que as demais tarefas do sistema operacional sejam executadas. Na figura 9.6 são mostrados os gráficos de desempenho de acordo com cada combinação do sistema operacional com a presença ou não de criptografia e compressão. Neste mesmo gráfico é mostrado em linhas verticais o intervalo de confiança dos pontos geradores das linhas de dados. Como pode ser observado, os mínimos e máximos estimados não afetam a confiabilidade dos dados apresentados uma vez que o erro é pequeno.

Em todos os casos o desempenho utilizando o sistema operacional Contiki se mostrou inferior ao utilizando Linux, isso ocorreu porque o sistema operacional Contiki foi executado sobre a mesma versão de Linux que é utilizada separadamente nos testes, sendo portanto esse o comportamento esperado.

No caso do sistema operacional, a análise mais importante é a que calcula a razão entre Linux e Contiki pois mostra mais claramente a diferença de comportamento do Contiki em relação ao Linux. Foram testados tráfego HTTP e HTTPS com e sem compressão, como mostrado na figura 9.7.

Nesse gráfico, nota-se que o sistema operacional tem impacto mais significativo no caso do HTTP, pois o uso do processamento é menor em relação ao uso da rede. Nos casos que tem maior exigência de processamento, como o uso da criptografia ou da compressão, a diferença de sistema operacional não é tão relevante pois os testes foram executados sobre a mesma plataforma ARM9.

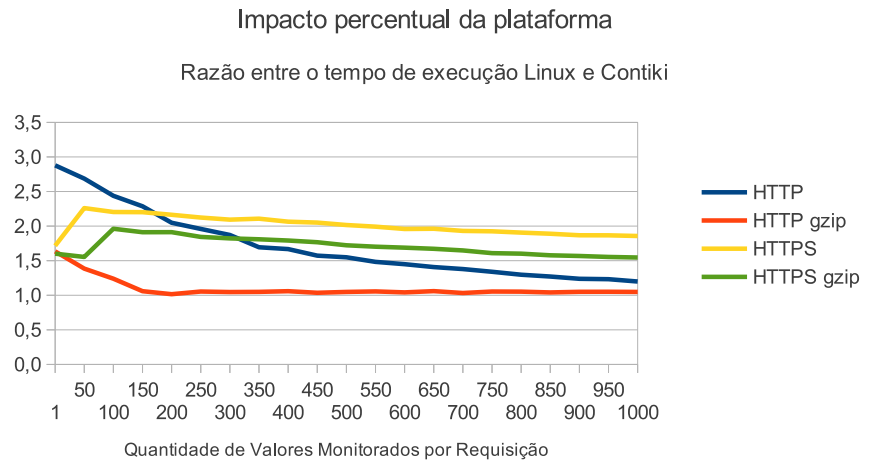

Figura 9.7: Impacto percentual da plataforma usando janela de 1000 bytes.

9.5 Impacto da suíte de criptografia

Após resultados preliminares que permitiram restringir o espaço de testes, foi analisado o impacto dos algoritmos escolhidos para a suíte de segurança. Os resultados podem ser vistos nas figuras 9.8 e 9.9. Estes gráficos são comparativos em relação ao desempenho com a comunicação HTTP sem segurança (TCP/IP).

Por exemplo, a suíte de criptografia SSL_3DES_SHA tem tempo de execução de 6,13 vezes maior do que o HTTP. Estes gráficos levam em conta somente a etapa de envio e criptografia dos dados, desconsiderando o *handshake* e a etapa de processamento das requisições.

Estes resultados se assemelham aos obtidos por Gisolfi (2001), embora os valores testados neste trabalho se baseiem no tempo de resposta do sistema e não apenas nos resultados da suíte de cifragem.

A suíte de criptografia que utiliza o algoritmo 3DES para a cifragem dos dados tem desempenho conhecidamente pior do que o RC4 e o AES e por isso os resultados comprovam que não é aconselhável utilizá-lo em aplicações embarcadas, já que o AES fornece nível de segurança melhor e é mais rápido que o 3DES.

Por ser um algoritmo de *stream* e não de bloco como o AES e o 3DES, o RC4 obteve um desempenho considerável, mas possui problemas comprovados de segurança. Entretanto, em ambientes sob o mesmo domínio administrativo utilizando mensagens pequenas, fornece um grau de segurança razoável.

O *Message Authentication Code*, que é o código usado para garantir a integridade das mensagens, pode ser calculado usando MD5 ou SHA-1. Embora ambos possuam possibilidade de colisões nos *hashes* gerados (WANG; YIN; YU, 2005)(WANG; YU, 2005), o SHA-1 ainda é mais utilizado que o MD5 em aplicações que necessitem de assinatura, ficando o segundo sendo suportado apenas por compatibilidade com aplicações legadas, visto que o algoritmo de assinatura utilizado é definido no *handshake*.

O SHA-1 mostrou-se um pouco mais lento do que o MD5, mas devido a recomendação de não utilização do MD5 pelo US-Cert (US-CERT, 2008) e dado que a diferença de desempenho entre ambos é pequena, recomenda-se o uso do SHA-1 mesmo em ambientes embarcados.

Nesta seção foram resumidos os principais resultados da aplicação dos algoritmos

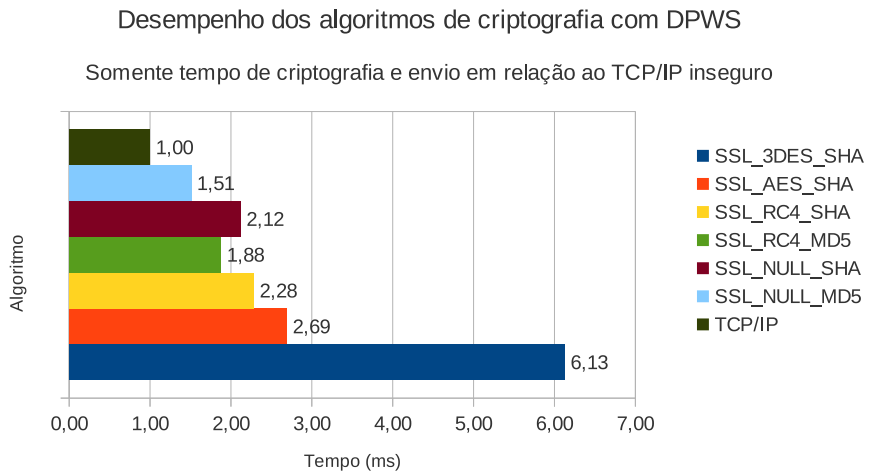

Figura 9.8: Impacto da suíte de segurança em plataforma x86

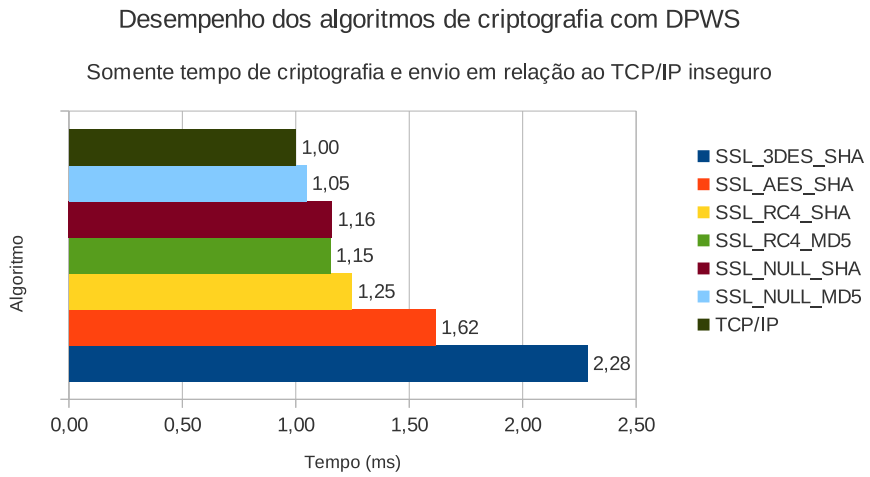

Figura 9.9: Impacto da suíte de segurança em plataforma ARM

de criptografia na comunicação utilizando o DPWS. Existem outros trabalhos na área que aprofundam a análise das suítes de criptografia (ZHAO et al., 2005), (AGRAWAL; SHARMA, 2010) e (ALSHAMSI; SAITO, 2005) que podem ser consultados para obtenção de informações mais detalhadas.

9.6 Comunicação de dados úteis

Até o momento foram apresentados os dados brutos levando em consideração o tempo para a transmissão das mensagens. Esta informação é relevante do ponto de vista de consumo de recursos de rede e processamento, mas para ser aplicada na tomada de decisões de projeto precisa ser complementada com o custo de transmissão de dados úteis. São considerados dados úteis aqueles que carregam a informação de monitoração ou controle. No caso deste trabalho, os dados úteis são os valores numéricos das grandezas medidas.

Na figura 9.10 é mostrado que o aumento da quantidade de dados transmitidos por requisição é bastante significativo quando é levado em conta o tempo utilizado para enviar

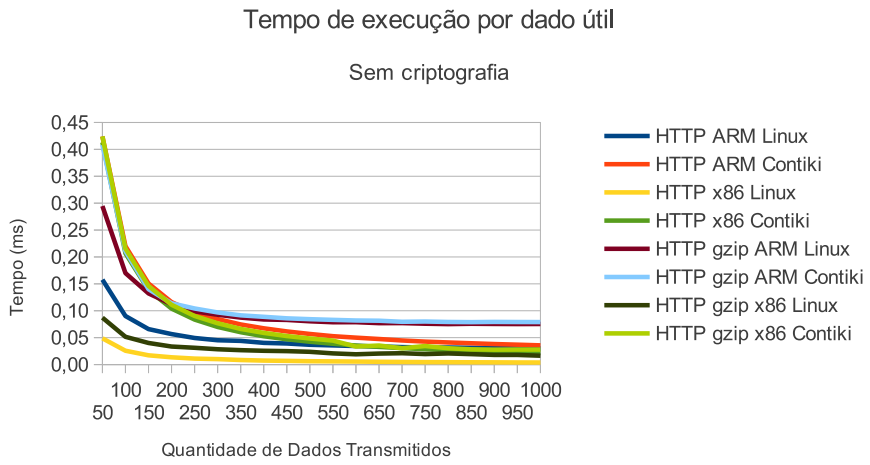

Figura 9.10: Desempenho a partir dos dados úteis transmitidos sem criptografia.

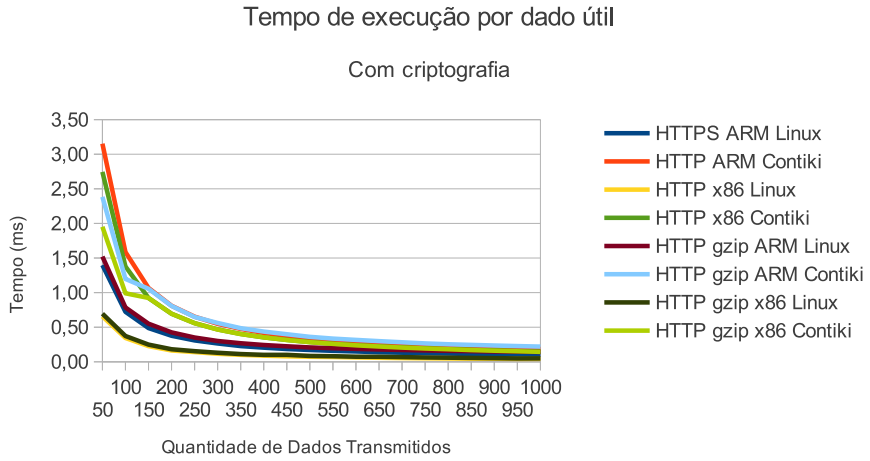

Figura 9.11: Desempenho a partir dos dados úteis transmitidos com criptografia.

cada dado. O que limita esta abordagem é que o retardo na obtenção das informações pelo cliente pode ser significativo principalmente quando é necessária a monitoração de valores críticos de processo. Entretanto, se são necessárias informações como medição de consumo de energia elétrica, estes dados podem ser armazenados nos equipamentos e consultados com uma frequência menor.

9.7 Análise da janela de dados

Foram realizados testes com 4 tamanhos de janelas de dados: 250, 500, 1000 e 1400 bytes de dados DPWS por mensagem. A janela limita a quantidade de dados a enviar e aumenta a quantidade de dados de cabeçalho transmitidos na rede. Apesar deste custo, um tamanho de janela de dados pequeno pode ser usado quando há dados que precisam de atualização com maior frequência, pois são necessárias poucas mensagens para o envio dos dados. Há também a situação em que o uso de janelas pequenas se deve aos tamanhos de *buffers* das interfaces de rede em alguns sistemas embarcados.

As figuras 9.12 a 9.15 mostram a variação do tempo de resposta de acordo com a quan-

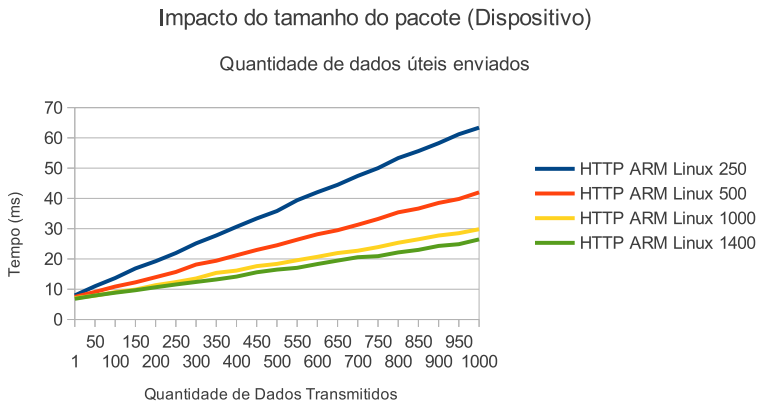

Figura 9.12: Desempenho em função do tamanho de janela usando HTTP.

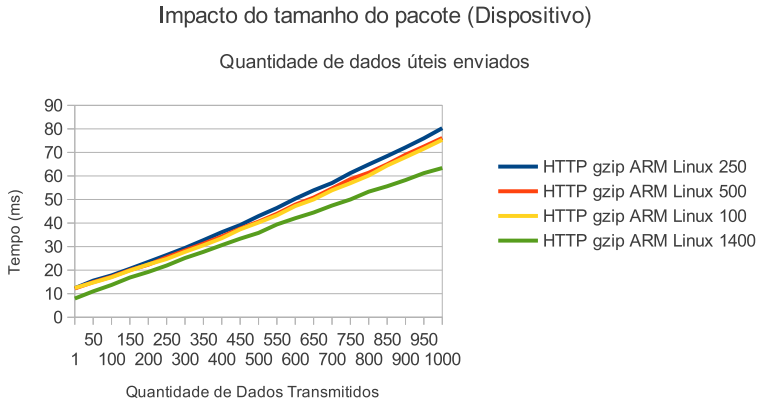

Figura 9.13: Desempenho em função do tamanho de janela usando HTTP e gzip.

tidade de dados úteis trafegados e separados por cada tamanho de janela utilizado. Através de uma análise visual dos gráficos percebe-se que o tamanho da janela tem influência quando se trata de comunicação sem compressão, mas acaba tendo uma influência bem menor quando é adicionada a compressão dos dados.

A análise dos gráficos permite identificar que o uso da compressão reduz o efeito do aumento do tamanho de janela como forma de reduzir o tempo de execução. Ao comparar os dados com e sem compressão é visto que o desempenho com compressão piora para janelas maiores, aproximando todos a reta que define o desempenho para a janela de tamanho 250 bytes.

9.8 Análise Estatística dos Resultados

Como mencionado no capítulo anterior, a análise estatística foi baseada na metodologia do planejamento de experimentos utilizando os fatores 2^k e níveis mostrados na figura na tabela 9.1.

Como mencionado anteriormente o planejamento de experimentos 2^k permite a visualização gráfica da influência de cada fator no resultado. Devido à segurança ser um requisito do tipo de tráfego tratado neste trabalho, a análise inicial foi reduzida para 3 fatores, variando-se apenas os fatores A, B e C da tabela e mantendo-se a utilização da

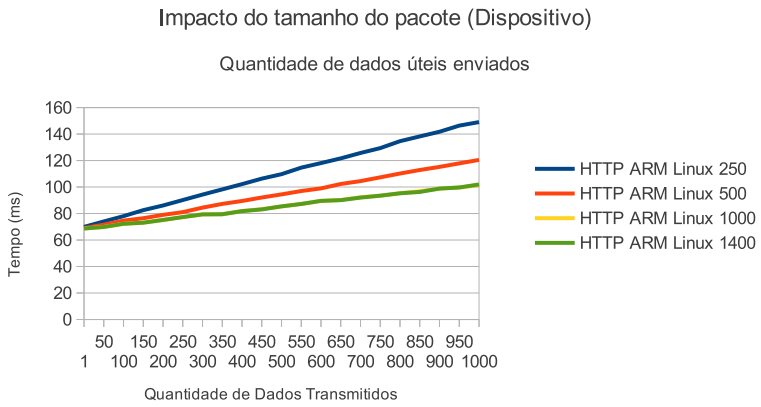

Figura 9.14: Desempenho em função do tamanho de janela usando HTTPS.

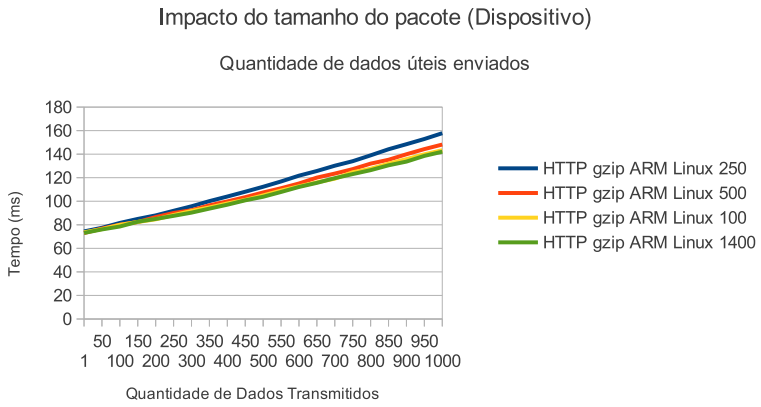

Figura 9.15: Desempenho em função do tamanho de janela usando HTTPS gzip.

segurança como um fator fixo. Além disso, foram realizadas 4 análises diferentes, utilizando a quantidade de valores transmitidos por requisição como fator fixo em cada uma delas, a saber 1, 50, 100 e 1000 valores.

A análise realizada para 1 valor será utilizada como exemplo da metodologia de cálculo e foi baseada no cálculo dos efeitos de cada fator e da suas interações. Foram realizados 10 repetições dos experimentos correspondentes a cada uma das combinações dos valores dos fatores mostrados na tabela 9.1. Os experimentos foram realizados com os procedimentos descritos no capitulo 6, no qual o cliente executa uma consulta ao dispositivo, medindo-se o tempo total entre o início da recepção da requisição até o fim do envio da resposta para o cliente.

	Baixo	Alto
Fator A: Sistema Operacional	Linux	Contiki
Fator B: Compressão	não	sim
Fator C: Tamanho de janela	250	1400
Fator D: Segurança	não	sim

Tabela 9.1: Níveis dos fatores do experimento 2^k.

Tratamento	Média	A	B	AB	C	AC	BC	ABC	Res(ms)
(1)	+	-	-	+	-	+	+	-	109,08
a	+	+	-	-	-	-	+	+	601,03
b	+	-	+	-	-	+	-	+	152,20
ab	+	+	+	+	-	-	-	-	597,77
c	+	-	-	+	+	-	-	+	94,91
ac	+	+	-	-	+	+	-	-	206,24
bc	+	-	+	-	+	-	+	-	147,74
abc	+	+	+	+	+	+	+	+	207,46
Contraste	264,55	1108,56	93,93	-97,98	-803,72	-766,48	14,19	-5,23	
Efeito		27,71	2,35	-2,45	-20,09	-19,16	0,35	-0,13	

Tabela 9.2: Montagem do experimento 2^k.

Para facilitar os cálculos dos efeitos foi montada a tabela 9.2, a qual possui a coluna de tratamento que serve como índice da combinação de cada experimento, sendo a primeira linha (1) correspondente a todos os fatores utilizados em nível baixo (Linux sem compressão e com tamanho de janela 250) e a linha abc correspondente a todos os fatores em nível alto (Linux com compressão e com tamanho de janela 1400).

A coluna de resultado mostra a soma dos tempos de resposta das 10 repetições para cada combinação de níveis alto e baixo para cada fator. As colunas de A a ABC tem o objetivo de facilitar a montagem das equações de cálculo dos contrastes de cada combinação. Os sinais indicam se os fatores foram utilizados em nível alto (+) ou baixo (-) na combinação correspondente. A coluna da média possui todos os sinais positivos, mas com uma finalidade diferente, pois indica que todos os resultados devem ser somados e divididos pela quantidade de combinações, calculando a média dos resultados.

O contraste do efeito de cada fator ou da interação entre eles é calculado somando-se os valores do resultado aplicado o sinal da coluna, como mostrado na equação 9.9. O efeito é obtido dividindo-se pela quantidade de combinações diferentes, como mostrado na equação e 9.10. Na equação 9.10 o valor *n* é a quantidade de repetições efetuadas e o valor *k*, a quantidade de fatores utilizada.

$$\begin{aligned} &C_A = -(1) + a - b + ab - c + ac - bc + abc \\ &C_A = -109,08 + 601,03 - 152,20 + 597,77 \\ &-94,91 + 206,24 - 147,74 + 207,46 = 1108,56 \end{aligned} \tag{9.9}$$

$$Efeito_A = \frac{(Contraste)}{2^{k-1} * n} = \frac{(1108,56)}{2^{3-1} * 10} = 27,71 \tag{9.10}$$

Para que possa ser utilizada a análise de variância para definir quais os fatores são significativos é necessário realizar o cálculo das somas quadradas, como mostrado na equação 9.11.

$$SQ_A = \frac{(Contraste)^2}{2^k * n} = \frac{(1108,56)^2}{2^3 * 10} = 15361,37 \tag{9.11}$$

Para completar a análise de variância é necessário calcular a soma de quadrados total com a equação 9.12 e a soma de quadrados dos resíduos por subtração como na equação 9.13. Na equação 9.12, é calculado o somatório dos quadrados de todos os tempos de resposta medidos, em cada uma das 10 repetições em cada combinação de níveis, do qual

é subtraído o quadrado da soma total dos tempos de resposta medidos (T) dividida pela quantidade total de execuções.

$$\begin{aligned} SQ_T &= \left(\sum_{ijk} x_{ijk}^2\right) - \frac{T^2}{N} \\ SQ_T &= 11,26^2 + 10,74^2 + ... + 20,87^2 - \frac{(11,26+10,74+...+20,87)^2}{2^3 * 10} \end{aligned} \tag{9.12}$$

A soma quadrada dos resíduos, é calculada pela subtração de todas as somas quadráticas calculadas da soma quadrática total.

$$\begin{aligned} SQ_R &= SQ_T - (SQ_A + SQ_B + SQ_{AB} + ... + SQ_{ABC}) \\ SQ_R &= 31065,19 - (15361,37 + 110,28 + ... + 0,34) = 52,46 \end{aligned} \tag{9.13}$$

A definição se cada um dos fatores e suas interações são estatisticamente significativas é feita através da metodologia da análise de variância.

Há suposições básicas para validar a análise de variância:

- Distribuição normal dos dados.
- Homogeneidade das variâncias.
- Aleatoriedade dos erros.
- Aditividade dos efeitos.
- Independência estatística dos valores observados (sem correlação).

Se as suposições de normalidade e homogeneidade não forem satisfeitas, o resultado da análise de variância deixa de ser exato, e passa a ser aproximado. Em raras situações a suposição de aditividade dos efeitos não é satisfeita. Nesse caso, uma transformação dos dados pode recuperar a aditividade e permitir uma análise mais precisa.

A partir dos dados coletados e calculados é montada a tabela 9.3 que mostra a análise de variância (ANOVA). A coluna GDL corresponde aos graus de liberdade utilizados, sendo 1 para os tratamentos e $2^k * (n-1)$ para o Erro. A média dos quadrados (MQ) é calculada dividindo-se a soma quadrada pelos graus de liberdade correspondentes. A partir destes valores calcula-se o valor para o teste F com a equação 9.14, onde o MQ_G é a média quadrada do tratamento que está sendo testado e o MQ_R a média quadrada do erro.

$$F_{calc} = \frac{MQ_G}{MQ_R} \tag{9.14}$$

O teste F indicará que o fator ou interação é significativo se $F_{calc} > F_{tab}$. Na tabela, os fatores significantes foram identificados como muito significantes ou pouco significantes, indicando que devem ser priorizado o trabalho de otimização nos fatores em que F_{calc} é muito maior o que F_{tab}. O F_{tab} utilizado foi obtido da tabela da distribuição F de Snedecor com $\alpha = 0,05$. Cada valor de contraste é testado com a seguinte suposição: H0: $C_i = 0$, ou seja, o coeficiente não difere significativamente de zero; H1: $C_i \neq 0$, ou seja, o coeficiente difere significativamente de zero.

Após os cálculos para 1 valor por requisição com segurança habilitada, foi realizada a mesma análise para 50, 100 e 1000 valores com segurança para determinar quais fatores são mais significativos em cada caso. Os valores calculados de F e os resultados da comparação com os valores tabelados são mostrados nas tabelas 9.4. Em cada par de

Fonte	SQ	GDL	MQ	Fcalc	Ftab	Signif.
Sistema Operacional	15361,37	1	15361,37	4684,79	4,49	muito
Compressão	110,28	1	110,28	33,63	4,49	pouco
Sist. Oper. * Compressão	120,01	1	120,01	36,60	4,49	pouco
Janela	8074,64	1	8074,64	2462,54	4,49	muito
Sist. Oper. * Janela	7343,58	1	7343,58	2239,59	4,49	muito
Compressão * Janela	2,52	1	2,52	0,77	4,49	não
Sist. Oper. * Compr. * Janela	0,34	1	0,34	0,10	4,49	não
Erro	52,46	16	3,28			

Tabela 9.3: Tabela ANOVA.

Fonte	Ftab	Fcalc	Sign	Fcalc	Sign	Fcalc	Sign
Valores por requisição		50		100		1000	
Segurança habilitada		sim		sim		sim	
Sistema Operacional	4,49	5665,92	++	5817,96	++	85,29	++
Compressão	4,49	257,69	+	350,88	+	37,60	+
Sist. Oper. * Compressão	4,49	693,46	+	873,81	+	0,28	nao
Janela	4,49	604,69	+	847,20	+	34,30	+
Sist. Oper. * Janela	4,49	357,61	+	382,16	+	0,36	nao
Compressão * Janela	4,49	316,87	+	307,15	+	7,11	+
Sist. Oper. * Compr. * Janela	4,49	435,70	+	448,00	+	0,01	nao

Tabela 9.4: Teste F em diferentes cenários.

colunas correspondentes a um destes cenários é usada a notação ++ para indicar que o fator ou interação é muito significativo, + se ele é pouco significativo e *não* se ele não é significativo.

Alguns fatores merecem destaque nessa análise, os casos de 50 e 100 valores com segurança a compressão tem significância maior e como será visto na análise dos gráficos dos efeitos principais nestes casos, a presença de compressão gera redução no tempo de resposta ao contrário do que acontece nos outros casos.

Os efeitos calculados para os casos escolhidos com segurança habilitada são mostrados graficamente na figura 9.16. Nota-se que os gráficos de efeito do sistema operacional tem inclinação maior do os outros fatores. Como esperado, os gráficos de efeito do tamanho de janela mostram que a redução no tamanho da janela realmente gera perda de desempenho devido aos bytes consumidos nos cabeçalhos Ethernet, IP e TCP como esperado. O objetivo dos gráficos mostrados é permitir a avaliação visual da inclinação da reta formada entre a média de todos os tempos de resposta medidos nas execuções dos experimentos com o fator em análise no nível baixo até a média dos valores encontrados com o fator em nível alto. A média em cada um dos casos é calculada usando todas as execuções em que há a variação dos outros fatores. Por se tratar de uma simplificação para facilitar a visualização, os pontos intermediários da reta não devem ser utilizados para estimar valores visto que o planejamento 2^k utiliza dois níveis discretos para a avaliação de cada fator.

Após a análise com 3 fatores, a utilização de tráfego seguro foi adicionada como

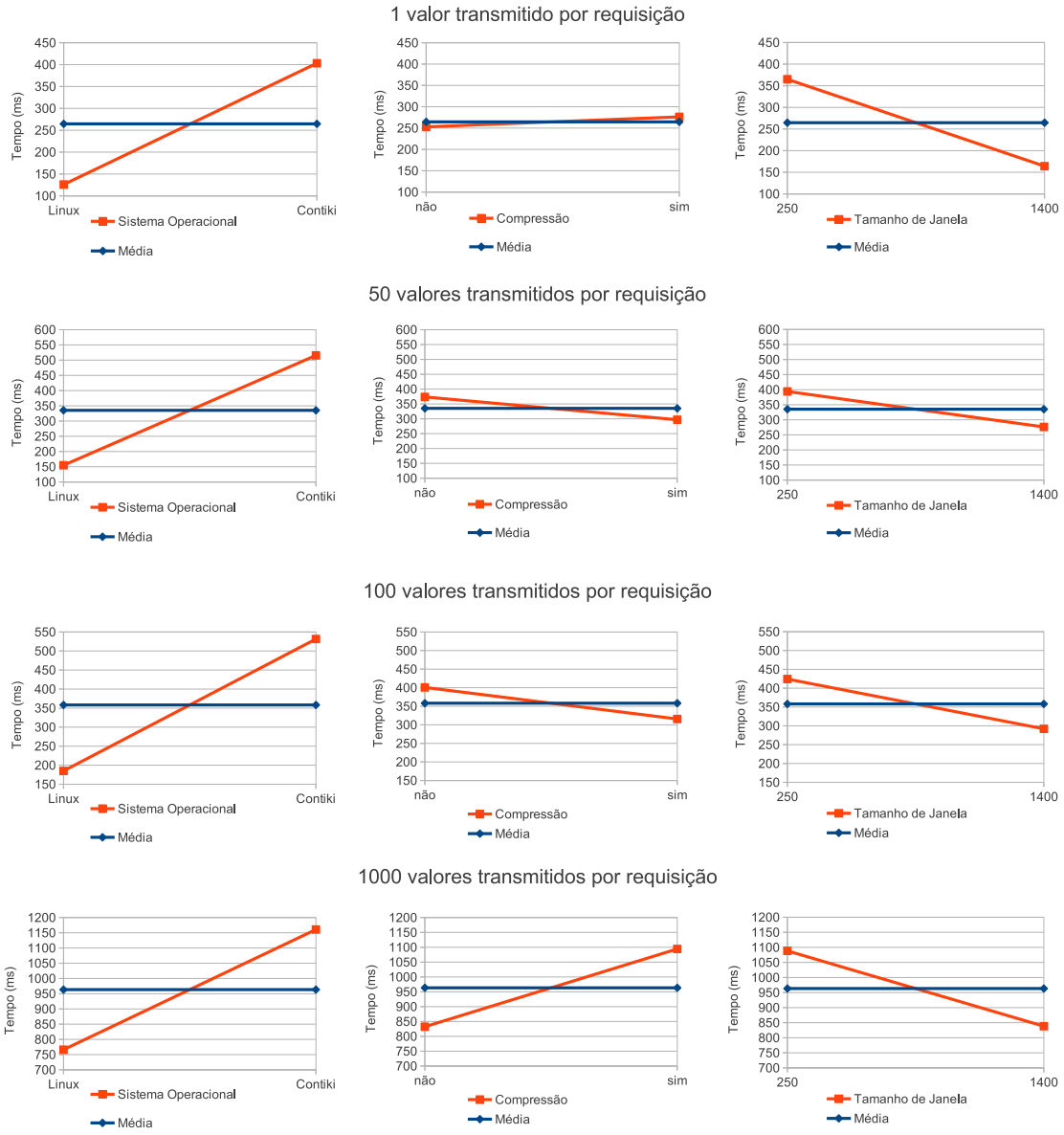

Figura 9.16: Gráficos de efeitos principais para diferentes cenários.

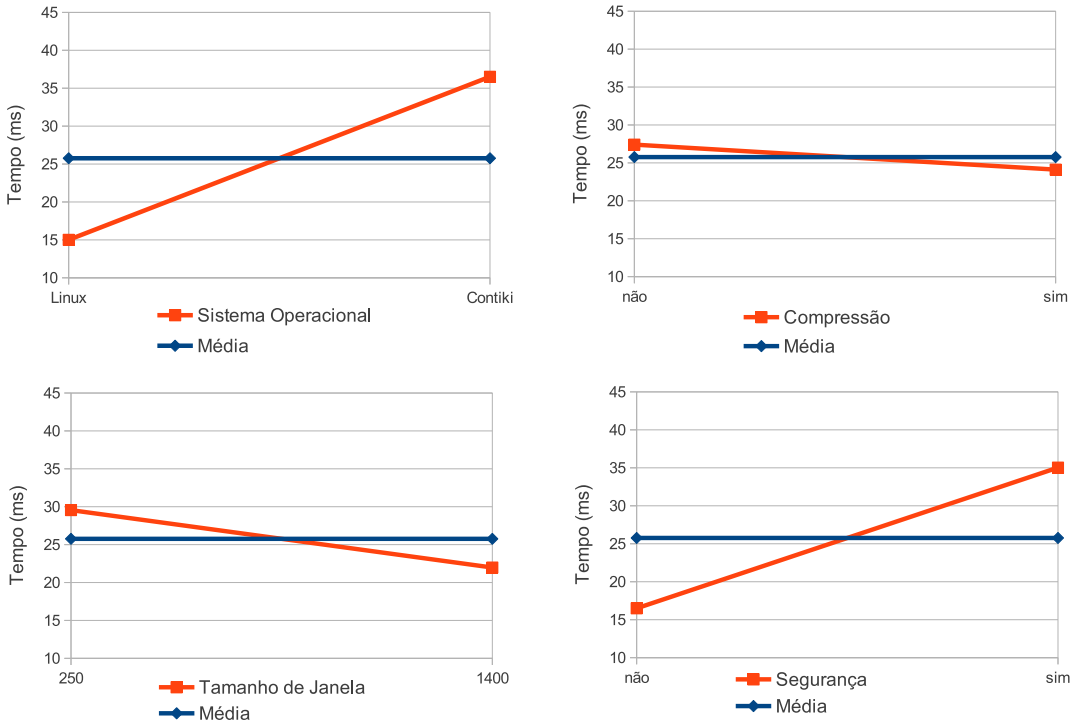

Figura 9.17: Gráfico dos efeitos principais com segurança e enviando 100 valores.

fator, para mensurar que impacto tem a segurança no tempo de resposta do dispositivo. O cálculo seguiu os mesmos passos mostrados para 3 fatores, aumentando-se a quantidade de linhas e colunas na tabela de cálculo dos efeitos. Na figura 9.17 são mostrados os efeitos principais de cada fator. Como era esperado devido a consumo de processamento ocasionado pelos algoritmos de criptografia, a adição de segurança é na média um dos fatores mais significativos no que se refere ao impacto sobre o tempo de resposta. Como pode ser observado na figura, com segurança habilitada, o tempo de resposta médio é de 35ms, enquanto sem a segurança esse tempo cai para 16ms.

A distribuição percentual dos efeitos principais dos fatores e suas interações pode ser vista na figura 9.18. O uso de tráfego seguro e a troca de sistema operacional afetam o desempenho de forma semelhante. A adição de segurança responde sozinha por 16,65% da variação do tempo de de resposta quando aplicada, além de suas interações com os outros fatores.

Analisando os motivos pelos quais a compressão gera perda de desempenho com quantidades baixas de valores e nas grandes quantidades de valores, sendo melhor para o caso médio, chegou-se a conclusão de que isto se deve ao fato de utilizar o canal de comunicação de forma otimizada, ou seja, transmitindo o máximo de informação no mesmo o *frame* Ethernet. Isto pode ser observado no gráfico percentual da variação de desempenho proporcionada pela compressão conforme mostra a figura 9.19. Apesar da perda de desempenho com a compressão em alguns casos, deve ser levado em conta o ganho de desempenho da rede com a redução proporcionada por ela que pode reduzir a um terço o consumo de banda.

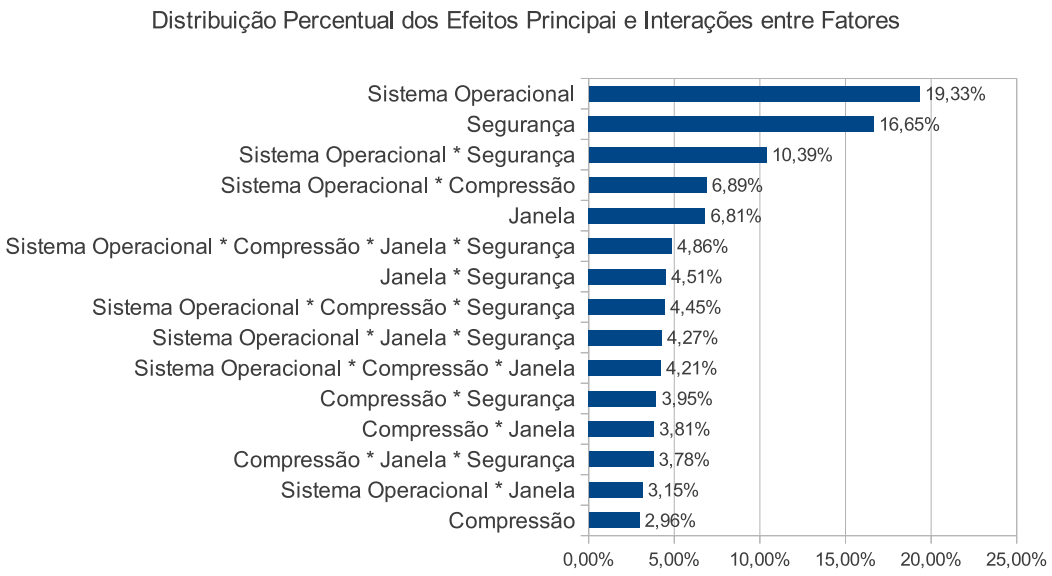

Figura 9.18: Distribuição percentual dos efeitos e seus fatores.

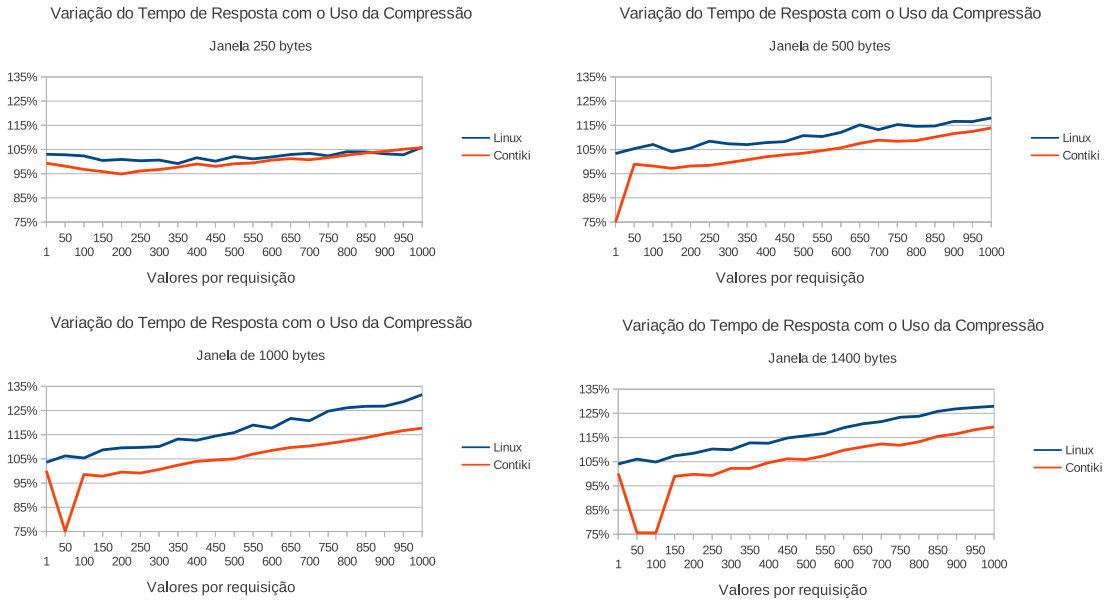

Figura 9.19: Variação do desempenho com o uso da compressão.

10 CONCLUSÃO

A busca por um novo paradigma de gerenciamento de dispositivos foi em grande parte bem sucedida com a especificação do DPWS. O perfil atende os principais requisitos existentes para a aplicação em automação, como a independência de plataforma, que é inerente à sua especificação baseada em protocolos e especificações abertos, e a capacidade de crescer de acordo com o tamanho dos equipamentos onde operam. O DPWS foi baseado em muitas tecnologias existentes como o as especificações WS-*, o HTML e o formato XML. Os pontos não atendidos completamente foram cobertos com novas especificações como no processo de descoberta dos dispositivos por meio do WS-Discovery. A escalabilidade é obtida por meio da fácil adaptação do DPWS aos recursos disponíveis no dispositivo controlado, que pode ser desde um sensor de uma rede de sensores sem fio, até um potente servidor no qual são executados os sistemas de gerenciamento corporativos.

Com a disseminação da utilização dos Web Services e as possibilidades abertas pela especificação do DPWS foram identificados pontos que não foram cobertos pelos estudos realizados até o momento. A avaliação de estratégias que combinassem a economia de recursos com o tráfego seguro dos dados através de redes inseguras ou sob o controle de terceiros. Como exemplos destas aplicações, é possível citar sistemas de segurança, redes de sensores e sistemas de medição de energia.

A fim de contribuir na solução de alguns dos problemas nesta área, este trabalho focou a adição e análise da camada de segurança do DPWS, com o objetivo de mensurar os requisitos necessários para a sua implantação e na identificação de possibilidades de redução do seu impacto, tanto no desempenho dos dispositivos quanto no tráfego de rede. Foi realizada a integração do uDPWS com as bibliotecas MatrixSSL e Zlib para a integração da segurança e compressão necessárias ao estudo.

O estudo realizado após a implementação mostrou o impacto considerável que a adição da camada de segurança tem sobre a comunicação usando o DPWS. A adição de segurança não aumenta significativamente o consumo de banda, pois o único acréscimo é o referente ao *Message Authentication Code*, que usa algoritmos de *hash* como MD5 e SHAn. A compressão se mostrou uma alternativa para minimizar o consumo de banda do protocolo sem afetar o desempenho, visto que o consumo de CPU gerado pela compressão reduz a quantidade de dados que devem ser criptografados, fazendo com que o consumo total de recursos não se altere significativamente. Nos resultados apresentados, pode-se comprovar que a compressão é percentualmente mais vantajosa quando se consegue reduzir através da sua aplicação os dados que necessitariam de diversos quadros Ethernet para o tamanho menor ou igual a um único quadro. Isto ocorreu nos casos de 100 ou 50 valores monitorados por requisição. Apesar da impressão deixada pelos resultados da avaliação dos efeitos da compressão, não se pode assumir que ela não deva ser usada para

quantidades de dados maiores, como no caso dos 1000 valores por requisição mostrados no trabalho. É importante sempre avaliar se as necessidades de economia de banda podem ser suficientes para justificar uma certa perda de desempenho no dispositivo, mas com um consumo de banda da ordem de 3 ou 4 vezes menor por requisição.

Apesar destas vantagens, o consumo de recursos do DPWS ainda se mostra muito alto para sistemas embarcados extremamente reduzidos, normalmente encontrados na chamada *Internet of Things*. Já existem alguns trabalhos avaliando o uso do REST como alternativa nestes casos, visto que a sobrecarga gerada pelo envelope SOAP é eliminada. Em (MORITZ et al., 2010) são discutidos os cenários em que Web Services DPWS e REST se adaptam melhor do que o outro. Por se tratar de um assunto novo, o uso dos Web Services no gerenciamento de dispositivos ainda é uma área em aberto. São necessários estudos mais aprofundados a respeito do uso do DPWS definindo na indústria o caminho a ser tomado para a adoção do uso dos Web Services no gerenciamento de equipamentos.

10.1 Trabalhos futuros

Como mencionado acima, há uma série de pontos onde os trabalhos realizados até o momento podem ser expandidos. A avaliação de outros formatos mais eficientes de representação que reduzam a quantidade de dados trafegados e o tempo de processamento das requisições no lado dos dispositivos é importante para aumentar os domínios de aplicação onde os Web Services podem ser utilizados. Uma opção viável é o uso do EXI formato binário para a representação de dados XML.

A realização de testes com diferentes larguras de banda é importante para verificar qual o comportamento dos Web Services em redes de sensores que possuem limitações nesse sentido. É provável que a compressão tenha um impacto maior com a restrição de banda, mas é necessário confirmar esta suposição e verificar como o tempo de transmissão afetará o processamento dos dados, pois nos casos em que a velocidade de transmissão é limitada pelo hardware, o tempo de representação de um bit é aumentado, fazendo com que a implementação e os testes precisem garantir que o processo em execução no dispositivo não seja bloqueado por esta transmissão, pois isso geraria perdas de desempenho. Outro fator importante a considerar é o algoritmo de compressão utilizado. Uma série de algoritmos de compressão podem ser avaliados para o uso com o formato XML utilizado. Dentre eles sugere-se uma modificação no algoritmo gzip que utilize um dicionário fixo gerado a partir de um conjunto de dados médio do DPWS.

O desenvolvimento de um gerente específico para o DPWS, focando na descoberta e na escalabilidade da solução também é de grande importância, pois os trabalhos até hoje tem focado no desenvolvimento do dispositivo, deixando para trabalhos futuros a análise do impacto da interação com os dispositivos. É necessário comprovar que as redes atuais podem absorver o impacto do crescimento da quantidade de dispositivos para redes de milhares de sensores ou dispositivos, quantidade facilmente encontradas em aplicações de controle de processos ou em aplicações de medição de energia.

REFERÊNCIAS

AGRAWAL, H.; SHARMA, M. Implementation and analysis of various symmetric cryptosystems. **Indian Journal of Science and Technolog**, [S.l.], v.3, n.12, p.1173–1176, Dec. 2010.

ALSHAMSI, A.; SAITO, T. A Technical Comparison of IPSec and SSL. **Advanced Information Networking and Applications, International Conference on**, Los Alamitos, CA, USA, v.2, p.395–398, 2005.

ALTMANN, V. et al. Investigation of the use of embedded Web Services in smart metering applications. In: ANNUAL CONFERENCE ON IEEE INDUSTRIAL ELECTRONICS SOCIETY, 38., IECON, 2012, Montreal, Canada. **...** [S.l.: s.n.].

BAJAJ, S. et al. **Web Services Policy 1.2 - Framework (WS-Policy)**. [S.l.]: W3C, 2006. <http://specs.xmlsoap.org/ws/2004/09/policy/ws-policy.pdf>. Acesso em: 15 dec. 2012.

BOHN, H.; BOBEK, A.; GOLATOWSKI, F. SIRENA - Service Infrastructure for Real-time Embedded Networked Devices: a service oriented framework for different domains. **Mobile Communications and Learning Technologies, Conference on Networking, Conference on Systems, International Conference on**, Los Alamitos, CA, USA, v.0, p.43, Apr. 2006.

BOOTH, D. et al. **Web Services Architecture**. [S.l.]: World Wide Web Consortium, 2004. <http://www.w3.org/TR/ws-arch/>. Acesso em: 15 dec. 2012.

BOURNEZ, C. **Efficient XML Interchange Evaluation**. [S.l.]: W3C, 2009. W3C Working Draft, <http://www.w3.org/TR/2009/WD-exi-evaluation-20090407>. Acesso em: 15 dec. 2012.

BOX, D. et al. **Web Services Addressing (WS-Addressing)**. [S.l.]: W3C, 2004. <http://www.w3.org/Submission/ws-addressing/>. Acesso em: 15 dec. 2012.

B.V., O. **PolarSSL library**. <http://www.polarssl.org>. Acesso em: 15 dec. 2012.

CASE, J. D. et al. **Simple Network Management Protocol (SNMP)**. <http://tools.ietf.org/html/rfc1157>. Acesso em: 15 dec. 2012.

CERAMI, E. **Web services essentials**. [S.l.]: O'Reilly, 2002. (Cookbooks Series).

CHINTHAKA, E. **Enable REST with web services, part 1**: rest and web services in wsdl 2.0. <http://www.ibm.com/developerworks/ webservices/library/ws-rest1/>. Acesso em: 15 dec. 2012., online.

CHOU, W. et al. **Web Services Transfer (WS-Transfer)**. [S.l.]: W3C, 2009. W3C Working Draft, <http://www.w3.org/TR/2009/WD-ws-transfer-20091217>. Acesso em: 15 dec. 2012.

CHRISTENSEN, E. et al. **Web Services Description Language (WSDL) 1.1**. 2001.

CUBO, J.; BROGI, A.; PIMENTEL, E. Towards behaviour-aware compositions of things in the future internet. In: INTERNATIONAL WORKSHOP ON ADAPTIVE SERVICES FOR THE FUTURE INTERNET, 2., WAS4FI, 2012. **...** [S.l.: s.n.].

DE SOUZA, L. M. S. et al. Socrades: a web service based shop floor integration infrastructure. **Networks**, [S.l.], v.4952, p.50–67, 2008.

DUNKELS, A.; GRONVALL, B.; VOIGT, T. Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. In: ANNUAL IEEE INTERNATIONAL CONFERENCE ON LOCAL COMPUTER NETWORKS, 29., LCN, 2004, Tampa, USA. **...** IEEE Computer Society, 2004. p.455–462. (LCN '04).

FIELDING, R. T. **REST:** architectural styles and the design of network-based software architectures. 2000. Doctoral dissertation — University of California, Irvine.

HURWITZ, J. et al. **Service Oriented Architecture For Dummies**. 2nd.ed. [S.l.]: For Dummies, 2009.

KYUSAKOV, R. et al. Integration of Wireless Sensor and Actuator Nodes With IT Infrastructure Using Service-Oriented Architecture. **IEEE Trans. Industrial Informatics**, [S.l.], v.9, n.1, p.43–51, 2013.

LERCHE, C. **Implementierung Web Services basierter Kommunikation fur Gerate mit starken Ressourcenbeschrankungen**. 2010. Dissertação (Mestrado em Ciência da Computação) — Universitat Rostock.

MALHOTRA, A. et al. **Web Services Metadata Exchange (WS-MetadataExchange)**. [S.l.]: W3C, 2009. W3C Working Draft, <http://www.w3.org/TR/2009/WD-ws-metadata-exchange-20091217>. Acesso em: 15 dec. 2012.

MALHOTRA, A. et al. **Web Services Eventing (WS-Eventing)**. [S.l.]: W3C, 2009. W3C Working Draft, <http://www.w3.org/TR/2009/WD-ws-eventing-20091217>. Acesso em: 15 dec. 2012.

MARTIN GUDGIN NOAH MENDELSOHN, M. N. H. R. **SOAP Message Transmission Optimization Mechanism**. [S.l.]: W3C - World Wide Web Consortium, 2005. <http://www.w3.org/TR/soap12-mtom/>. Acesso em: 15 dec. 2012.

MONTENEGRO, G. et al. **Transmission of IPv6 Packets over IEEE 802.15.4 Networks**. <http://www.ietf.org/rfc/rfc4944.txt>. Acesso em: 15 dec. 2012., RFC 4944 (Proposed Standard).

MONTGOMERY, D. **Design and Analysis of Experiments**. [S.l.]: John Wiley & Sons, 2008. (Student solutions manual).

MORITZ, G. et al. Devices Profile for Web Services and the REST. **Industrial Informatics (INDIN)**, [S.l.], v.8th, p.584–591, 2010.

MOURA, G. C. M. et al. On the Performance of Web Services Management Standards - An Evaluation of MUWS and WS-Management for Network Management. In: IFIP/IEEE INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT, 10., IM, 2007, Neubiberg, Germany. **...** [S.l.: s.n.], 2007. p.459–468.

NADALIN, A. et al. Web Services Security: soap message security 1.1 (ws-security 2004). **OASIS Standard**, [S.l.], v.200401, Feb. 2006.

NEWCOMER, E. **Understanding Web Services: xml, wsdl, soap, and uddi**. [S.l.]: Addison-Wesley Professional, 2002.

NIXON, T. et al. Devices Profile for Web Services Version 1.1. **OASIS**, [S.l.], n.July, p.1–43, 2009.

NIXON, T.; REGNIER, A. **Web Services Dynamic Discovery (WS-Discovery) Version 1.1**. [S.l.]: OASIS, 2009. OASIS Standard, <http://docs.oasis-open.org/ws-dd/discovery/1.1/wsdd-discovery-1.1-spec.html>. Acesso em: 15 dec. 2012.

OMG, O. M. G. **CORBA Component Model 4.0 Specification**. [S.l.]: Object Management Group, 2006. Specification. (Version 4.0).

PAPAZOGLOU, M. P.; HEUVEL, W.-J. Service oriented architectures: approaches, technologies and research issues. **The VLDB Journal**, Secaucus, NJ, USA, v.16, n.3, p.389–415, July 2007.

Peersec Networks. **Matrix - Open Source Embedded SSL**. <http://matrixssl.org>. Acesso em: 15 dec. 2012.

PELTZ, C. Web Services Orchestration and Choreography. **Computer**, Los Alamitos, CA, USA, v.36, n.10, p.46–52, 2003.

RICH, C. **axTLS Embedded SSL**. <http://axtls.sourceforge.net>. Acesso em: 15 dec. 2012.

RICHARDSON, L.; RUBY, S. **RESTful Web Services**. 1.ed. [S.l.]: O'Reilly Media, Inc., 2007.

RICHARDSON, L.; RUBY, S. **RESTful web services - web services for the real world.** [S.l.]: O'Reilly, 2007. I-XXIV, 1-419p.

ROHR, E. M. Z. L. M. R. T. **Segurança em gerenciamento de redes baseado em web services**. 2009. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal do Rio Grande do Sul. Instituto de Informática. Programa de Pós-Graduação em Computação.

SAMARAS, I. K.; GIALELIS, J. V.; HASSAPIS, G. D. Integrating Wireless Sensor Networks into Enterprise Information Systems by Using Web Services. **2009 Third International Conference on Sensor Technologies and Applications**, [S.l.], v.-, n.Xml, p.580–587, 2009.

SCHONWALDER, J. **Network Configuration Protocol Light (NETCONF Light)**. [S.l.]: IETF, 2012. n.01. (Internet Draft).

SILVA, B. L. P. da. REST vs WS-*: uma visão pragmática. **Java Magazine**, [S.l.], v.54, p.38–47, 2008.

SOA4D. **DPWS Core**. <https://forge.soa4d.org/projects/dpwscore/>. Acesso em: 15 dec. 2012.

SOSNOSKI, D. **Java web services**: the high cost of (ws-)security. <http://www.ibm.com/developerworks/java/library/j-jws6/index.html>. Acesso em: 15 dec. 2012., on-line.

SUDA, B. **SOAP Web Services**. 2003. Dissertação (Mestrado em Ciência da Computação) — University of Edinburgh.

Sun, M. I. **Jini Networking Technology**: an executive overview. [S.l.]: Sun Microsystems Inc., 2001. <http://wwws.sun.com/software/jini/whitepapers/jini-execoverview.pdf>. Acesso em: 15 dec. 2012.

The OpenSSL Project. **OpenSSL**: the open source toolkit for SSL/TLS. <http://www.openssl.org>. Acesso em: 15 dec. 2012.

The OSGi Alliance. **OSGi Service Platform Core Specification, Release 4.1**. <http://www.osgi.org/Specifications>. Acesso em: 15 dec. 2012.

THURAISINGHAM, B. **Secure Semantic Service-Oriented Systems**. Hoboken: CRC Press, 2010.

UPNP, F. UPnP Device Architecture Introduction. **Forum American Bar Association**, [S.l.], p.1–43, 2006. <http://www.upnp.org/>. Acesso em: 15 dec. 2012.

US-CERT. **MD5 vulnerable to collision attacks**. <http://www.kb.cert.org /vuls/id/836068>. Acesso em: 15 dec. 2012., On-line.

VALIPOUR, M. H. et al. A brief survey of software architecture concepts and service oriented architecture. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE AND INFORMATION TECHNOLOGY, 2., ICCSIT, 2009, Kiev, Ukraine. **...** [S.l.: s.n.], Aug. 2009. p.34–38.

W3C. **XML Schema Part 0**: primer second edition ; w3c recommendation 28 october 2004. <http://www.w3.org/TR/xmlschema-0/>. Acesso em: 15 dec. 2012.

W3C. **SOAP Version 1.2 Part 0**: primer (second edition). <http://www.w3.org/ TR/soap12-part0/>. Acesso em: 15 dec. 2012., online.

W3C, W. G. **Web Services Glossary**. <http://www.w3.org/TR/ws-gloss/>. Acesso em: 15 dec. 2012.

WANG, X.; YIN, Y. L.; YU, H. Finding Collisions in the Full SHA-1. In: INTERNATIONAL CRYPTOLOGY CONFERENCE, 25., CRYPTO, 2005, Santa Barbara, USA. **...** Springer, 2005. p.17–36.

WANG, X.; YU, H. How to Break MD5 and Other Hash Functions. In: ANNUAL EUROCRYPT CONFERENCE, 24. EUROCRYPT, 2005, Aarhus, Denmark. **...** [S.l.: s.n.], 2005. p.19–35.

WIDJAJA, T.; BUXMANN, P. Service-oriented architectures: modeling the selection of services and platforms. In: EUROPEAN CONFERENCE ON INFORMATION SYSTEMS, 17., ECIS, 2009, Verona, Italy. **...** [S.l.: s.n.], 2009. p.2420–2431.

WS4D. **Web Services for Devices**. <http://www.ws4d.org>. Acesso em: 15 dec. 2012.

yaSSL Company. **CyaSSL Embedded SSL Library**. <http://www.yassl.com/yaSSL/Products-cyassl.html>. Acesso em: 15 dec. 2012.

ZEEB, E. et al. Towards component orientation in embedded Web Service environments. In: IEEE INTERNATIONAL CONFERENCE ON EMERGING TECHONOLOGIES AND FACTORY AUTOMATION, 15., ETFA, 2010, Bilbao, Spain. **...** IEEE, 2010. p.1–8.

ZEEB, E. et al. WS4D: toolkits for networked embedded systems based on the devices profile for web services. In: INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING WORKSHOPS, 39., ICIP, 2010, San Diego, USA. **...** IEEE Computer Society, 2010. p.1–8.

ZHAO, L. et al. Anatomy and Performance of SSL Processing. In: INTERNATIONAL SYMPOSIUM ON PERFORMANCE ANALYSIS OF SYSTEMS AND SOFTWARE, ISPASS, 2005, Austin, USA. **...** [S.l.: s.n.], 2005. p.197–206.