UNIVERSIDADE FEDERAL DO ABC

Curso de Pós-Graduação em Engenharia da Informação

Dissertação de Mestrado

Daniele Freitas de Jesus

Estudo de Arquiteturas de Segurança para Internet das Coisas Utilizando o Protocolo DTLS

Santo André

2015

Curso de Pós-Graduação em Engenharia da Informação

Dissertação de Mestrado

Daniele Freitas de Jesus

Estudo de Arquiteturas de Segurança para Internet das Coisas Utilizando o Protocolo DTLS

Trabalho apresentado como requisito parcial para obtenção do título de Mestre em Engenharia da Informação, sob orientação do Professor Doutor João Henrique Kleinschmidt.

Santo André
2015

Este exemplar foi revisado e alterado em relação à versão original, de acordo com as observações levantadas pela banca no dia da defesa, sob responsabilidade única do autor e com a anuência de seu orientador.

Santo André, ____de _____ de 2015.

Assinatura do autor: _____

Assinatura do orientador: _____

RESUMO

O termo Internet das Coisas se refere à interconexão de redes formadas por objetos inteligentes e as tecnologias disponíveis na Internet tradicional, assim como o conjunto das tecnologias que compõem esses objetos e de suas aplicações e serviços. A Internet das Coisas possui grandes limitações em recursos computacionais (como memória, processamento e taxa de transmissão da rede) e ainda são grandes os desafios para possibilitar segurança nas informações transmitidas. Toda implementação de segurança deve levar essas limitações em consideração, o que motivou essa pesquisa. Portanto este trabalho analisa diferentes arquiteturas de segurança para a Internet das Coisas, utilizando o protocolo DTLS e os protocolos que já estão sendo utilizados na Internet das Coisas. Nessa pesquisa foram estudados os principais conceitos de Internet das Coisas, suas camadas e protocolos, segurança, criptografia, gerenciamento de chaves, dando destaque ao protocolo DTLS. Foi utilizado o ambiente de simulação Contiki, para a realização de testes e verificação do consumo de recursos do sistema, como energia e tempo de uso do processador, obedecendo à pilha de protocolos das camadas disponíveis estudadas. Os resultados apresentaram a redução de consumo de energia e tempo em relação ao formato do DTLS, modelo de *handshake* e arquitetura de segurança.

Palavras-Chave - Internet das Coisas; Criptografia; Gerenciamento de Chaves; Custo Energético; DTLS; *Contiki.*

Dedico este trabalho a toda minha família e amigos, que me ampararam e sempre compreenderam quando precisei estar ausente para me dedicar a minha pesquisa e estudo. Em especial a minha mãe, companheira sempre presente nessa jornada de conhecimento.

Daniele Freitas de Jesus

Agradecimentos

Agradeço primeiramente a Deus que sempre me deu forças para continuar estudando e me dedicando. Ao meu orientador João Henrique Kleinschmidt, que não só me introduziu ao mundo da pesquisa científica, como também teve sempre ativa participação ao longo do projeto, permitindo o aprimoramento do trabalho e contribuindo sempre para a minha aprendizagem e conhecimento nesta área tão envolvente da pesquisa.

Agradeço também a todos os professores que me dedicaram sua atenção, compartilharam suas experiências e saberes ao longo do curso, em especial a professora Maria das Graças B. Marietto, que sempre me incentivou a me engajar em novos horizontes.

Daniele Freitas de Jesus

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

6LoWPAN - *IPv6 Over Low Power Wireless Personal Area Network*

AC - *Access Control*

AES - *Advanced Encryption Standard*

CoAP - *Constrained Application Protocol*

CRC - *Cyclic Redundancy Check*

DES - *Data Encryption Standard*

DODAG - *Destination-Oriented Directed Acyclic Graph*

DoS - *Denial of Service*

DTLS - *Datagram Transport Layer Security*

ECC - *Elliptic Curve Cryptography*

EPC - *Eletronic Product Code*

ETSI - *European Telecommunications Standards Institute*

FFD - *Full Function Device*

H2H - *Human to Human*

H2T - *Human to Things*

HTTP - *Hyper Text Transfer Protocol*

I - *Intensity*

IEEE - *Institute of Electrical and Electronics Engineers*

IETF - *Internet Engeneering Taks Force*

IoT - *Internet of Things*

IP - *Internet Protocol*

IPv6 - *Internet Protocol version 6*

ISO - *International Organization for Standardization*

LLN - *Low-Power and Lossy Networks*

M2M - *Machine to Machine*

MAC - *Medium Access Control*

MEMS - *Micro-Electro-Mechanical Systems*

NFC - *Near Field Communications*

OSCAR - *Object Security Architecture*

PC - *Personal Computer*

PHY - *Physics*

PKI - *Public Key Infrastructure*

PSK - *Pre-Shared Key*

RAM - *Ramdom Access Memory*

RFD - *Reduced Function Device*

RFID - *Radio Frequency Identification*

ROLL - *Routing Over Low Power and Lossy*

ROM - *Read Only Memory*

RPL - *Routing Protocol for Low-power*

RSA - *Rivest-Shamir-Adlleman*

RSSF - *Redes de Sensores Sem Fio*

RTT - *Round Trip Time*

SOA - *Service Oriented Architecture*

T2T - *Things to Things*

TCP - *Transmission Control Protocol*

TLS - *Transport Layer Security*

TLV - *Type-Length-Value*

UDGM - *Unit Disk Graph Model*

UDP - *User Datagram Protocol*

V - *Voltage*

XML - *Extensible Markup Language*

I - INTRODUÇÃO

O termo Internet das Coisas (IoT - *Internet of Things*) vem sendo amplamente utilizado para se referir à interconexão da rede formada por objetos inteligentes e as tecnologias globais na Internet, além do conjunto das tecnologias que compõem esses objetos e o conjunto de suas aplicações e serviços. Algumas características chaves identificam esse paradigma: dispositivos heterogêneos, a necessidade de escalabilidade, a troca de dados onipresente por meio de tecnologias sem fio e escassez de recursos para economizar eventuais custos na construção da arquitetura [MIORANDI, 2012].

As aplicações da Internet das Coisas vão desde o sensoriamento e monitoramento de florestas e mares, transporte de cargas, controle de identificação inteligente na saúde, ambientes corporativos e residenciais, até o uso pessoal e social, tornando objetos ou coisas mais inteligentes e independentes, além de ampliar o leque de comunicação das tecnologias disponíveis no mundo atual.

As principais tecnologias que tornam a IoT possível são as Redes de Sensores Sem Fio (RSSF) e Identificação por Rádio Frequência (RFID - *Radio Frequency Identification*) [MIORANDI, 2012], [ATZORI, 2010], [KOPETZ, 2011].

As redes de sensores sem fio podem ser formadas por dezenas ou até centenas de nós sensores, dispositivos autônomos de baixo custo e tamanho reduzido, com capacidade para realizar sensoriamento, processamento e transmissão de informação. Uma característica vital nestas redes, como em qualquer outro dispositivo na IoT, é seu baixo consumo energético, que deve ser considerado nas implementações das aplicações e serviços [LOUREIRO, 2003], [ATZORI, 2010], [MIORANDI, 2012].

A tecnologia de RFID se tornou uma ótima solução por conta de seu baixo custo, além de ser considerada como uma das soluções para identificação de objetos inteligentes na Internet das Coisas, já que possui a premissa de identificação única, rápida e fácil dos objetos [ATZORI, 2010], [MIORANDI, 2012], [KOPETZ, 2011], [MEDAGLIA, 2010], [MIRI, 2012].

Com um contínuo aumento da integração e comunicação de dispositivos da IoT com a Internet tradicional, torna-se necessário o desenvolvimento de mecanismos para segurança das informações trafegadas. Devido o canal de comunicação da IoT ser sem fio, as informações trafegadas são muito vulneráveis a ataques, sendo que os dispositivos possuem recursos escassos em processamento, armazenamento e energia [CHEN, 2012].

A segurança da informação na IoT pode englobar vários conceitos diferentes, mas de forma geral, refere-se ao fornecimento básico de serviços de segurança, que incluem a confidencialidade, autenticação, integridade, autorização, não-repúdio e disponibilidade dos dados. Estes serviços de segurança podem ser

implementados com uma combinação de mecanismos de criptografia e mecanismos não-criptográficos, que implementam os aspectos das políticas de segurança [GARCIA-MORCHON, 2012].

Os recursos escassos na Internet das Coisas ainda são os grandes limitantes para possibilitar segurança das informações transmitidas e, diferente das redes tradicionais, não podem necessariamente utilizar os mesmos mecanismos. Portanto todo progresso nesse segmento é importante para a comunidade acadêmica e de pesquisa dessa área.

A falta de padronização para a Internet das Coisas é um problema, pois os protocolos existentes para a Internet não são padronizados para dispositivos com poucos recursos. Entretanto, vários novos protocolos, como CoAP (*Constrained Access Protocol*) e 6LoWPAN (*IPv6 over Low Power Wireless Personal Area Network*) já estão sendo utilizados para substituir protocolos como HTTP (*Hyper Text Transfer Protocol*) e IP (*Internet Protocol*) na IoT. Vários autores como [RAZA; ET AL. 2012], [KOTHMAYR; ET AL. 2013], [BERGMANN; ET AL, 2012], [VUCINIC; ET AL. 2015], [HUMMEN; RAZA; ET AL, 2014], [ROMAN, ET AL., 2011], [HARTK; BERGMANN, 2012], têm proposto o uso do protocolo *Datagram Transport Layer Security* (DTLS), uma alternativa estudada por estas pesquisas como protocolo de segurança para IoT. Esses trabalhos propõem o uso desse protocolo com os protocolos CoAP e 6LoWPAN, mas não analisam esta pilha de protocolos nos principais cenários da Internet das Coisas.

Dentro deste contexto, esta pesquisa procura fazer uma contribuição na área de segurança da Internet das Coisas, utilizando os protocolos propostos para a IoT e o protocolo DTLS na arquitetura de segurança da rede. Os resultados parciais desta pesquisa foram apresentados em [JESUS; KLEINSCHMIDT, 2014] e [JESUS; KLEINSCHMIDT, 2015].

1.1 Problema de Pesquisa

A Internet das Coisas precisa estabelecer uma comunicação segura entre os nós da rede e, por isso, é importante que todos os nós usem os mesmos protocolos para a comunicação segura. Entretanto existe uma falta de padronização na segurança da IoT, sendo o protocolo DTLS o mais promissor a ser utilizado e que tem recebido recente interesse da comunidade acadêmica e de órgãos como o IETF (*Internet Engineering Taks Force*). Esta dissertação estuda arquiteturas de segurança, que utilizando o protocolo de DTLS e os protocolos que são propostos para a IoT, consigam garantir suas funções sem consumir muito a energia da rede e tenha métricas de eficiência, flexibilidade e segurança garantidas.

1.1.2 Delimitação de Escopo

Para realização dos testes não foram utilizados sensores e objetos reais da IoT, mas sim os simulados disponíveis no ambiente *Contiki* [DUNKELS, 2004]. Entretanto, devido às conformidades do

software com as placas existentes no mercado (os modelos disponíveis no Contiki possuem implementações precisas com as placas reais) e por ser utilizado em várias pesquisas da área, utilizar o Contiki garante uma boa aproximação com dados obtidos em ambientes reais. [DUNKELS, 2004], [TOHEED; RAZI, 2010].

1.1.3 Justificativa

Os protocolos da Internet existentes não estão otimizados para a comunicação em dispositivos que necessitam baixo consumo de energia, por isso a necessidade de adaptação destes protocolos para a IoT. Portanto, a implementação de segurança deve levar em consideração essas limitações.

1.2 Objetivos

1.2.1 Objetivos Gerais

O objetivo geral do trabalho é estudar diferentes arquiteturas de segurança para Internet das Coisas utilizando o protocolo DTLS como camada de segurança.

1.2.2 Objetivos Específicos

- Estudar os principais conceitos da IoT, suas camadas e protocolos, os principais conceitos de segurança, criptografia e gerenciamento de chaves.
- Analisar diferentes arquiteturas de segurança utilizando o protocolo DTLS com os protocolos padronizados da IoT.
- Analisar o processo de *handshake* do DTLS.
- Comparar o DTLS padrão com o DTLS comprimido (*Lithe*).
- Analisar o desempenho e o consumo de energia da rede do DTLS em cenários da IoT.
- Implementar diferentes arquiteturas de segurança e cenários no ambiente de simulação *Contiki*.

1.3 Estrutura da Dissertação

No capítulo I foi feita uma introdução da dissertação e uma breve apresentação da pesquisa. No capítulo II deste trabalho, serão apresentados os principais conceitos da Internet das Coisas e suas aplicações, as camadas existentes na IoT e seus protocolos. No capítulo III serão apresentados os conceitos gerais sobre segurança, criptografia e gerenciamento de chaves, alguns problemas na implementação do gerenciamento de chaves na Internet das Coisas, o protocolo DTLS e suas principais características e funções, e alguns trabalhos correlatos importantes para esta pesquisa. No capítulo IV, Metodologia, será apresentado um resumo do Contiki, um breve resumo sobre os arquivos alterados do protocolo DTLS, os parâmetros utilizados e a configuração dos cenários de testes. O capítulo V apresenta

os resultados e discussões desta pesquisa. Por fim, as Conclusões Finais utilizadas ao longo deste trabalho, apresentados no capítulo VI.

II - INTERNET DAS COISAS

A Internet das Coisas permite que objetos inteligentes[1] possam se comunicar em "qualquer momento", em "qualquer lugar" e com "qualquer um", graças ao seu conjunto de tecnologias (como RFID, dispositivos sensores/atuadores, entre outros) e aplicações disponíveis a esses objetos no mercado [ATZORI, 2010] , [MIORANDI, 2012].

Na Figura 1, [ATZORI, 2010] apresenta a Internet das Coisas por meio de três principais visões: perspectivas "Orientada pelas Coisas" (nessa visão, são considerados simplesmente os dispositivos componentes da rede), "Orientada pela Semântica" (considera o papel que a rede irá desempenhar e os dispositivos necessários para cada um), e "Orientada pela Internet" (essa visão considera a conexão da Internet tradicional com as Coisas e vice-versa).

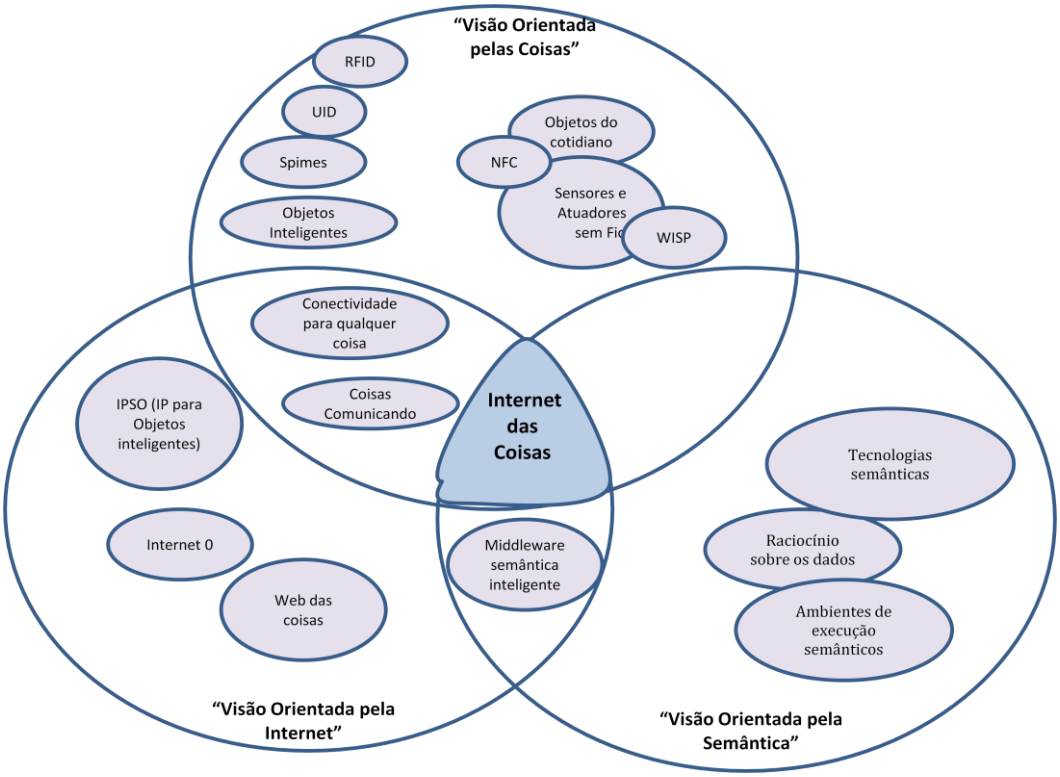

Figura 1 - Paradigma Internet das Coisas
Fonte: Adaptado de ATZORI, 2010.

A Internet das Coisas possui algumas características-chaves que precisam ser consideradas para seu funcionamento [MIORANDI, 2012]: dispositivos heterogêneos; escalabilidade; troca de dados onipresente por meio de proximidade nas tecnologias sem fio; soluções otimizadas de energia; localização

[1] **Objetos Inteligentes**: De acordo com [MIORANDI, 2012], são entidades físicas com conjunto de características associadas a um conjunto mínimo de comunicação, tais como a capacidade de ser descoberto, aceitar e responder mensagens de outros objetos, possuir identificador único.

e rastreamento de recursos; capacidade de auto-organização; interoperabilidade semântica, gerenciamento de dados, mecanismos de segurança e de preservação da privacidade incorporado.

A Internet das Coisas possui grande número de aplicações possíveis, mas de forma simplificada, podem ser subdivididas em grandes áreas como: Domínio de Transporte e logística; Domínio da saúde; Domínio de Ambiente Inteligente (casa, escritório, fábrica); Domínio pessoal e social [ATZORI, 2010].

Cada domínio pode ser subdividido em áreas [ATZORI, 2010]:

- Domínio de Transporte e logística: Logística (tecnologia de processamento de informações em tempo real em quase todos os elos da cadeia de suprimentos, que vão desde a concepção de *commodities*, compra de matéria-prima, produção, armazenamento e transporte, distribuição e venda de produtos semi-acabados e produtos acabados, processamento e serviço de "retornos" em pós-venda); *Mobile Ticketing* (cartazes ou painéis que fornecem informações sobre os serviços de transporte podem ser equipados com uma *tag*, um marcador visual, e um número identificador); Monitoramento de Parâmetros Ambientais (monitoramento de bens perecíveis durante o transporte, verificando temperatura, umidade, entre outros); Mapas Ampliados (mapas turísticos podem ser equipados com *tags* que permitem que telefones devidamente equipados para navegar no mapa e automaticamente chamar serviços web que fornecem informações sobre hotéis, restaurantes, monumentos e eventos relacionados com a área de interesse do usuário);
- Domínio da saúde: Rastreamento (visa a identificação de uma pessoa ou objeto em movimento); Identificação e Autenticação (inclui a identificação do paciente para reduzir incidentes prejudicial, prontuário eletrônico abrangente e atual manutenção, e identificação em hospitais); Coleta de Dados (coleta e transferência automática de dados são principalmente destinadas à redução do tempo de processamento do formulário médico, automação de processos, automatizando auditoria de procedimento e gestão de inventário médico); Sensoriamento (dispositivos sensores colocados em pacientes, apóiam no diagnóstico, fornecendo informações em tempo real sobre os indicadores de saúde do mesmo);
- Domínio de Ambiente Inteligente: Casas e Escritórios Confortáveis (sensores e atuadores distribuídos em casas e escritórios podem tornar a vida mais confortável, além de evitar alguns incidentes domésticos); Instalações Industriais (ambientes inteligentes também ajudam a melhoria da automação de plantas industriais); Museu e Ginásio Inteligentes (ajudar na exploração das instalações do museu; no ginásio, o treinador pode carregar o perfil do exercício para a formação de cada aluno);

- Domínio pessoal e social: Redes Sociais (relacionado à atualização automática de informações para redes sociais); Consultas Históricas (consultas históricas sobre objetos e dados de eventos, para o estudo das tendências de algumas atividades ao longo do tempo); Perdas (utilização de RFID em objetos, para que possam ser rastreados quando perdidos); Furtos (similar a perdas, permite ao usuário saber se objeto foi movido de uma área restrita, indicando provável furto).

Em relação às arquiteturas físicas, os dispositivos de comunicação de curto alcance - *Near Field Communications* (NFC) - as Redes de Sensores e Atuadores, juntamente com os dispositivos de RFID são reconhecidos como os componentes atômicos da Internet das Coisas, que ligam o mundo real com o mundo digital [ATZORI, 2010]. Outros autores resumem os principais dispositivos da IoT como sendo formada por Redes de Sensores Sem Fio (RSSF) e RFID [MIORANDI, 2012], [KOPETZ, 2011].

O padrão NFC que é uma extensão do padrão ISO / IEC 14443 de proximidade de cartões, é uma tecnologia de comunicação sem fio de curto alcance de alta frequência que permite a troca de dados entre dispositivos em distâncias menores que 20 cm. Esta tecnologia é compatível com os *smartcards* e leitores existentes, bem como com outros dispositivos NFC, já em uso para o transporte público e pagamento. Esse padrão é destinado principalmente para uso em telefones celulares [KOPETZ, 2011].

Com os progressos recentes no campo de sistemas micro - eletro - mecânicos (MEMS - *Micro-Electro-Mechanical Systems*), foi possível construir pequenos objetos integrados inteligentes, chamados de nós sensores, que contêm um sensor, um microcontrolador e um controlador de comunicação sem fio. Um nó sensor pode adquirir uma variedade de parâmetros químicos, biológicos ou sinais físicos para medir as propriedades de seu ambiente. Eles possuem recursos limitados e são alimentados tanto por uma pequena bateria computacional ou energia colhida a partir de seu ambiente, o que têm limitado sua energia e tempo de vida, além de ter uma memória pequena e capacidades de comunicação restrita. As Redes de Sensores Sem Fio (RSSF) são formadas por dezenas, ou até centenas de nós sensores, para aplicações de controle, tráfego, militar, segurança, ambiente e medicina. [LOUREIRO, 2003], [ATZORI, 2010], [MIORANDI, 2012], [KOPETZ, 2011].

A maioria das RSSF está sendo baseadas no padrão IEEE 802.15.4 [ATZORI, 2010], que define as camadas física e MAC (*Medium Access Control*) com baixo consumo energético e taxa de transferência de comunicação em redes de área pessoal, que será retomado no capítulo 2.1.

Para automatizar o processo de identificação do objeto, eliminando a ligação humana, etiquetas eletrônicas, as chamada etiquetas RFID, foram desenvolvidas. Estas podem ser lidas a partir de uma pequena distância por um leitor de RFID. A *tag* (etiqueta) RFID armazena um único *Eletronic Product*

Code (EPC) do objeto anexado, que se torna sua identificação. O leitor RFID pode atuar como um *gateway* para a Internet e transmitir a identidade do objeto, juntamente com o tempo de leitura e localização do objeto (isto é, a localização do leitor) a um sistema de computador remoto que gerencia um banco de dados maior, tornando possível rastrear objetos em tempo real. São usados em vários cenários, abrangendo de logística à saúde e segurança. Os maiores responsáveis por esse padrão são EPCGlobal, ETSI (*European Telecommunications Standards Institute*) e ISO (International Organization for Standardization) [ATZORI, 2010], [MIORANDI, 2012], [KOPETZ, 2011], [MEDAGLIA, 2010], [MIRI, 2012].

Os sistemas RFID são classificados por seus objetivos fins, que são, normalmente, autorização e monitoramento. Sistemas de autorização substituem as abordagens mais tradicionais de concessão de acesso de uma entidade para uma determinada área, enquanto que os sistemas de monitoramento estabelecem a localização de uma entidade nessa área. Embora seus objetivos sejam diferentes, o *hardware* subjacente é idêntico nos dois tipos de sistemas [MIROWSKI; ET AL, 2009].

Com o aumento contínuo da integração e comunicação entre *PCs (Personal Computer)*, H2H (*Human to Human*), H2T (*Human to Things*) e T2T (*Things to Things*), a quantidade e a qualidade de informações trafegadas na IoT causa preocupação aos pesquisadores da área, já que é extremamente vulnerável a ataques, devido ao canal de comunicação ser sem fio e sua estrutura física possuir poucos recursos de armazenamento, energia e capacidade de processamento, inviabilizando esquemas mais complexos para suporte à segurança. A Figura 2 ilustra a comunicação na IoT [CHEN, 2012].

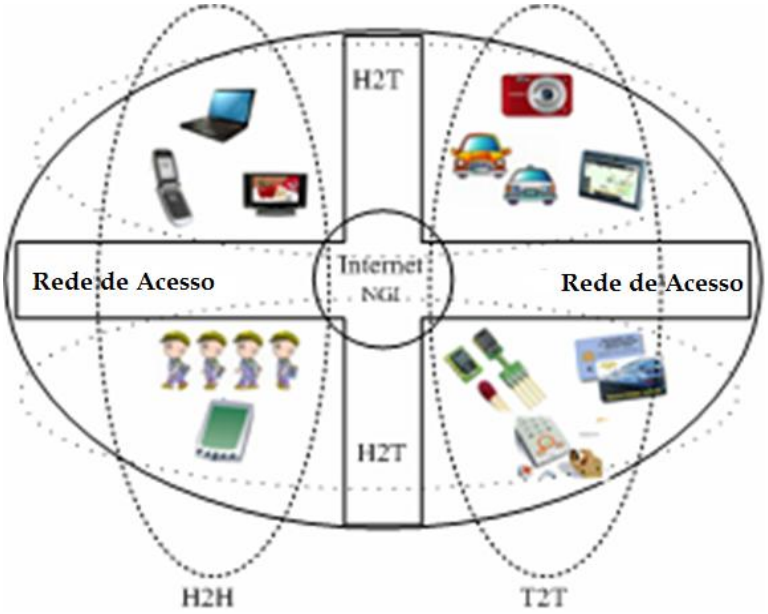

Figura 2 - Comunicação IoT
Fonte: Adaptado de CHEN, 2012

Estruturalmente, a Internet das Coisas requer arquiteturas de software que sejam capazes de lidar com grandes quantidades de informações, consultas e computação, fazendo uso de novos paradigmas de processamento de dados, processamento de fluxo, filtragem, agregação e mineração de dados, todos sustentados por padrões de comunicação, tais como HTTP e IP. Entretanto, a energia é desperdiçada por transmissão de dados desnecessários, por conta de sobrecarga de protocolo e padrões de comunicação não-otimizados. Protocolos de Internet existentes como o HTTP e o TCP não são otimizados para a comunicação em estruturas que exijam baixo consumo de energia, o que dificulta a adaptação dos protocolos existentes. Por isso, a necessidade em se padronizar protocolos para a IoT [PALATTELLA, 2013].

Na Figura 3 são apresentados os principais protocolos e aplicativos para a Internet das Coisas, distribuídos em camadas que serão citados a seguir.

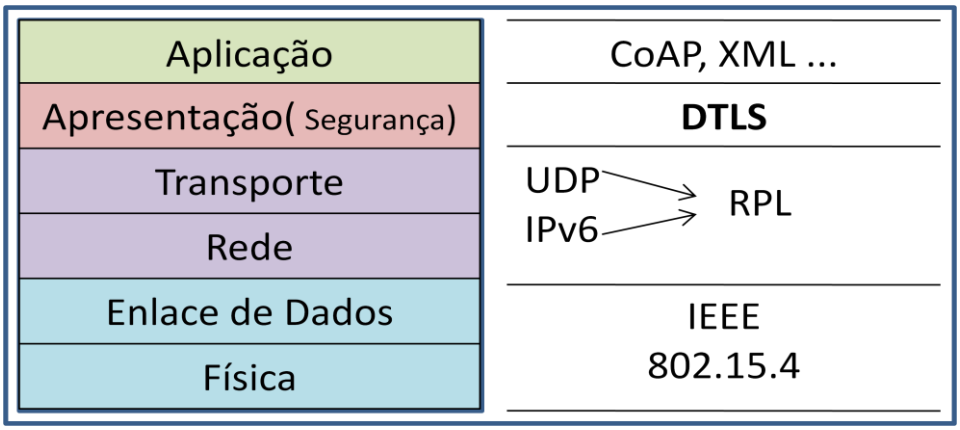

Figura 3 - Pilha de Protocolos por Camada
Fonte: Adaptado de KOTHMAYR; ET AL. 2013

Os protocolos disponíveis e atualmente usados em cada camada, de acordo com [KOTHMAYR; ET AL. 2013] são: na camada de Aplicação, *Constrained Application Protocol* (CoAP), *Extensible Markup Language* (XML) entre outros; em Segurança, o *Datagram Transport Layer Security* (DTLS); na camada de Transporte e Rede, o *User Datagram Protocol* (UDP), IPv6 e *Routing Protocol for Low-power and Lossy Networks* (RPL); na camada de Acesso e Física, o IEEE 802.15.4. Os principais protocolos da IoT serão apresentados nas próximas seções.

2.1 IEEE 802.15.4

IEEE 802.15.4 é um padrão da camada de enlace/física específico para resolver o problema de baixo consumo de energia e baixa taxa de transferência nas redes pessoais sem fio. Pode ser definido em dois grupos de dispositivos: dispositivos de função plena (FFD - *Full Function Device*) e dispositivos de

função reduzida (RFD - *Reduced Function Device*). O padrão IEEE802.15.4 define a camada física (PHY - *Physics*) e a camada de acesso ao meio (MAC) da rede [OLIVEIRA; ET AL. 2012].

Os dispositivos do tipo FFD suportam todas as funcionalidades de rede e podem suportar topologias ponto-a-ponto, por causa de sua capacidade de roteamento de múltiplos saltos. Já os dispositivos RFD suportam apenas um conjunto limitado de funcionalidades e são utilizados principalmente para operações de detecção e/ou atuação, além de somente ser usado em topologia estrela[OLIVEIRA; ET AL. 2012].

A camada física do IEEE 802.15.4 possui um equilíbrio entre a eficiência energética, o alcance do sinal e a taxa de transmissão dados. Define 16 canais de frequência, localizados a cada 5 MHz entre 2,405 GHz e 2,480 GHz, e os canais possuem apenas 2 MHz de largura. O rádio pode arbitrariamente enviar e receber em qualquer um desses canais e cada rádio compatível é capaz de mudar de canal em não mais do que 192 ms [PALATTELLA, 2013].

O IEEE 802.15.4 também define um protocolo MAC (uma subcamada da camada de enlace, em que os protocolos usados determinam quem será o próximo em um canal de multiacesso [TANENBAUM, 2003]), isto é, a camada interagindo diretamente com o rádio. Ele define o formato do cabeçalho MAC (com campos de endereço da origem e do destino) e como os dispositivos podem se comunicar uns com os outros. Esta camada MAC é voltada para redes do tipo estrela, em que todos os dispositivos se comunicam diretamente com um nó central [PALATTELLA, 2013].

O grande impulso no uso do IEEE802.15.4 foi devido à sua integração com padrões industriais populares, como por exemplo, a pilha de protocolo *ZigBee*. Por padrão a comunicação 802.15.4 não oferece qualquer segurança, entretanto, especifica o uso do padrão AES (*Advanced Encryption Standard*) para criptografia, autenticação única ou criptografia e autenticação [VLADISLAV, 2012].

2.2 IPV6 over Low power Wireless Personal Area Network (6LoWPAN)

IPv6 over Low Power Wireless Personal Area Network (6LoWPAN) é uma tecnologia que o Grupo de Trabalho 6LoWPAN da *Internet Engineering Task Force* (IETF) desenvolveu para integrar completamente a Internet das Coisas na Internet tradicional, bem como para permitir a maior parte dos recursos de IPv6 em nós com recursos limitados, e também para a transmissão de pacotes IPv6 nos Padrões IEEE 802.15.4 [HONG; ET AL. 2012].

Devido às limitações da Internet das Coisas, o grupo de trabalho 6LoWPAN dedicou-se em definir um protocolo, que inclui auto-configuração de endereços IPv6, apoio à transmissão na camada de enlace,

redução de roteamento, gerenciamento de sobrecarga, adoção de protocolos de aplicativos leves e o apoio para os mecanismos de segurança [PALATTELLA, 2013].

Para realizar a entrega do quadro usando uma comunicação *unicast*, um cabeçalho de endereçamento é utilizado antes de quaisquer outros cabeçalhos de encapsulamento 6LoWPAN. Já quando a comunicação é *multicast/broadcast* na camada de enlace, será necessário, para mecanismos de inundação controlada ou para a descoberta de topologia, um cabeçalho de *Broadcast*, que segue imediatamente um cabeçalho *Mesh Addressing* [PALATTELLA, 2013].

Os endereços atribuídos para interfaces 6LoWPAN são formados com um identificador de interface derivado diretamente dos endereços MAC. Fornece também uma técnica de compactar cabeçalhos IPv6, que é a codificação LoWPAN NHC. Ele comprime cabeçalhos próximos com formatos diferentes, identificando-os com um padrão de comprimento variável que segue imediatamente o cabeçalho LoWPAN IPHC comprimido, enquanto a RFC 6282 permite um formato de compressão para cabeçalhos UDP usando LoWPAN NHC [PALATTELLA, 2013].

2.3 IETF ROLL - RPL

O *Routing Protocol for Low-power* (RPL) é um protocolo de roteamento de vetor de distância inteiramente definido para LLNs (*Low-Power and Lossy Networks*) baseadas em IPv6. Ele está atualmente sob especificação pelo Grupo de Trabalho do IETF ROLL (*Routing Over Low power and Lossy*). É projetado para ser um protocolo mais flexível no sentido de proporcionar um padrão para a IoT [BAUER, 2010].

O RPL constrói suas rotas em intervalos periódicos, sendo assim considerado um protocolo de roteamento pró-ativo. Em uma rede física, é possível executar várias instâncias de RPL. Os nós na rede estão autorizados a fazer parte de mais de uma instância. Começando a partir de um ou mais nós raízes , em que cada instância constrói uma estrutura de roteamento de árvore na rede, o que resulta em um *Destination-Oriented Directed Acyclic Graph* (DODAG) [BAUER, 2010], [PALATTELLA, 2013].

2.4 Constrained Application Protocol (CoAP)

O IETF *Constrained Application Protocol* é um protocolo web que roda sobre o protocolo não confiável UDP e é projetado principalmente para a Internet das Coisas. O CoAP é uma variante do protocolo síncrono mais utilizado na web, HTTP, e é feito sob medida para dispositivos M2M (*Machine to Machine*) de comunicação restrita [RAZA; ET AL, 2013].

Os mecanismos centrais do CoAP são [KOVATSCH, 2011]:

- Os aplicativos podem enviar mensagens CoAP confiáveis ou não confiáveis;
- Se destina a fornecer comunicação em grupo via IP *multicast*;
- Tem notificações *push* nativas por meio de um mecanismo de publicação/assinatura chamado de "observação de recursos".

O CoAP adota padrões de HTTP, como abstração de recursos, *URIs*, interação *RESTful*, e as opções de cabeçalho extensíveis, mas usa representações binárias compactas que são projetados para serem fáceis de analisar [KOVATSCH, 2011]. A Figura 4 ilustra as camadas e a arquitetura do CoAP.

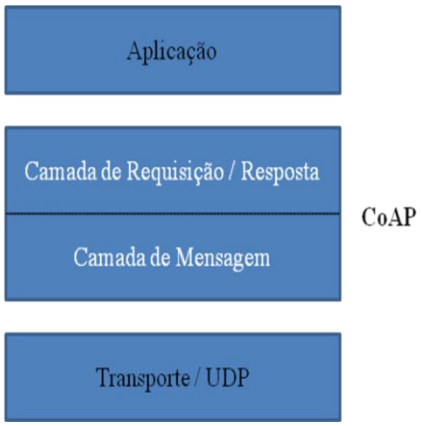

Figura 4 - Arquitetura CoAP
Fonte: Adaptação de PALATTELLA, 2013

A camada de transporte é responsável por fornecer confiabilidade fim a fim por meio de redes baseadas em IP. Como descrito anteriormente, para as LLNS é viável utilizar o UDP. O UDP é um protocolo orientado a *datagrama*, que fornece um procedimento para enviar mensagens a outras aplicações com um mecanismo de protocolo com o mínimo de sobrecarga. Como TCP, o UDP fornece aplicação de multiplexação por meio do conceito de porta.

O desempenho do TCP [RAZA; ET AL. 2013] em redes sem fio é ineficiente por conta de seus algoritmos de controle de congestionamento, retransmissões e etc. Por isso, o uso do UDP na Internet das Coisas. Entretanto, HTTP foi designado para rodar em cima do TCP, e para garantir seu funcionamento em UDP a IETF propõe padronizar o uso com CoAP.

A camada de aplicação nas LLNS deve se preocupar com os vários dispositivos de capacidades diferentes envolvidos na rede, tanto os que não necessitam restringir o uso de protocolos, como os que precisam, e para isso se faz necessária uma adaptação. Um requisito para a camada de aplicação é limitar a extensão dos pacotes.

A arquitetura do CoAP é dividida em duas camadas, uma de mensagem (*Message Layer*) e uma de requisição e resposta (*Request/Response Layer*) [PALATTELLA, 2013].

A função da camada de mensagem é controlar a troca de mensagens sobre UDP entre dois pontos fins. As mensagens utilizam um identificador para se detectar duplicatas e ter confiabilidade. Existem quatro tipos de mensagens que são especificados no cabeçalho [PALATTELLA, 2013]:

- confirmáveis: mensagens que requerem uma resposta à requisição; as que não podem ser processadas são respondidas por mensagens *reset*.
- não confirmáveis: não precisam de repostas.
- de reconhecimento: mensagens que confirmam a recepção de uma mensagem confirmável.
- *reset*: no caso de uma mensagem confirmável não poder ser processada.

Já a camada de requisição/resposta inclui um código de método ou código de resposta, respectivamente. Como CoAP é implementado ao longo de transporte não-confiável, mensagens CoAP podem chegar fora de ordem, aparecerem duplicadas ou serem perdidas sem aviso prévio, por isso a necessidade da camada de requisição/resposta. A requisição é a solicitação da aplicação, composta do método, identificador e carga, enquanto a resposta indica o resultado de entender e satisfazer a requisição[PALATTELLA, 2013].

Os quadros CoAP são codificados em um simples formato binário. Uma mensagem consiste num cabeçalho CoAP de tamanho fixo, seguido por opções em *Type-Length-Value* (TLV) e uma carga útil. O número de opções é determinado pelo cabeçalho, conforme ilustrado na Figura 5. A carga é constituída pelos bytes após as opções, se houver; e o seu comprimento é calculado a partir do comprimento de datagramas [PALATTELLA, 2013].

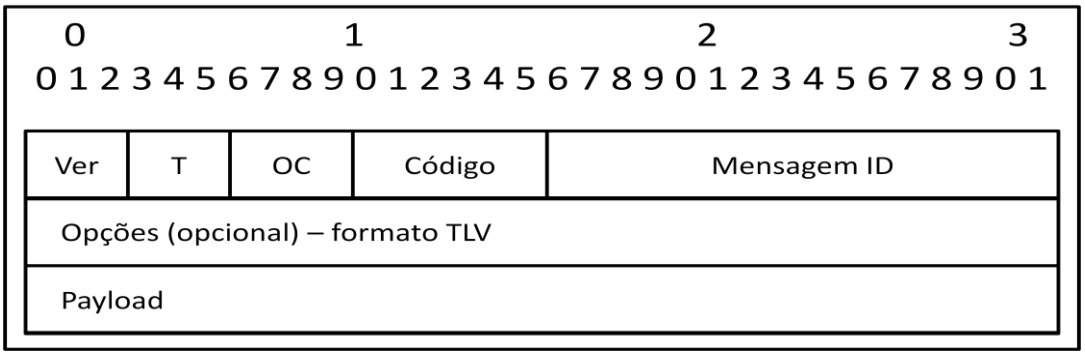

Figura 5 - Quadro CoAP
Fonte: Adaptado de PALATTELLA, 2013

Os métodos básicos do CoAP são [PALATTELLA, 2013]:

- *GET*: operação segura que recupera uma representação da informação correspondente ao recurso identificado pelo URI de requisição.
- *POST*: Solicita o processamento no recurso identificado pela requisição URI. Normalmente isso resulta em um novo recurso ou o recurso alvo que está sendo atualizado.
- *PUT*: Solicita que o recurso identificado pela requisição URI seja atualizado ou criado com a representação em anexo.
- *DELETE*: Os pedidos de métodos que o recurso identifica pela requisição URI são excluídos.

No próximo capítulo serão apresentados os conceitos gerais de segurança, criptografia e gerenciamento de chaves, bem como o protocolo da camada de segurança, DTLS.

III - SEGURANÇA

Os serviços básicos de segurança incluem: confidencialidade, autenticação, integridade, autorização, não-repúdio e disponibilidade. Estes podem ser implementados com uma combinação de mecanismos de criptografia, tais como cifras de bloco, de funções *hash* ou algoritmos de assinatura, e mecanismos não-criptográficos, que implementam os aspectos das políticas de segurança [GARCIA-MORCHON, 2012].

Segundo [VADSLAV] os principais serviços de segurança são resumidos:

- Confidencialidade: todas as informações importantes que estão sendo transmitidas entre as partes que estão se comunicando, permanecem desconhecidas para todos os outros. Isto normalmente é alcançado criptografando os dados de tal forma que, mesmo que um terceiro escute todos os pacotes transmitidos não seria capaz de acessar os dados.
- Autenticação: permite a identificação entre as partes durante uma comunicação. Desta forma, um atacante não pode se passar por alguém para obter acesso a dados importantes ou para espalhar informações inválidas. Autenticação é geralmente obtida por meio do envio de uma mensagem com código de autenticação (MAC - *Medium Access Control*), juntamente com a mensagem.
- Integridade: garante que a mensagem foi recebida exatamente da mesma forma como foi enviada, e não sofreu qualquer forma de alteração no trajeto. A simples verificação de redundância cíclica (CRC - *Cyclic Redundancy Check*), que é usada para detectar erros aleatórios durante a transmissão de pacotes ou um MAC pode fornecer integridade.
- Autorização: garante que somente nós autorizados possam acessar os serviços e recursos na rede.
- Disponibilidade: indica que os nós da rede prestarão serviços sempre que necessário, apesar de possíveis ataques de negação de serviços (DoS - *Denial of Service*).

Já o não-repúdio ou irretratabilidade, impede que o emissor de uma mensagem negue a sua autoria [FERNANDES, 2006].

A criptografia é fundamental para prover segurança, pois por meio dela é possível atender a todos os requisitos clássicos: a confidencialidade e integridade dos dados trafegados, e a autenticação e a irretratabilidade do emissor [FERNANDES, 2006], [DING, 2010]. No próximo sub capítulo serão apresentados os principais conceitos de criptografia.

3.1 Criptografia

A criptografia é a pedra fundamental para a proteção da infraestrutura de rede. Embora padrões, como AES, possam ser utilizados para alguns dispositivos da Internet das Coisas, outros, tais como etiquetas RFID passivas, são extremamente limitadas em recursos e, por isso, não podem utilizar mecanismos de segurança muito complexos. Esses mecanismos poderiam incluir algoritmos simétricos, funções *hash*, e geradores de números aleatórios [ROMAN, 2011].

A criptografia é tradicionalmente separada em: simétrica (uma única chave é usada para criptografar e decriptografar as mensagens) ou assimétrica (chaves diferentes: chave privada, que deve ser mantida em segredo, e chave pública, distribuída entre os membros da rede) [BALA, 2012], [FERNANDES, 2006].

As cifras simétricas podem ser classificadas em cifras de bloco e cifras de fluxo. As cifras de bloco operam num conjunto de dados de tamanho fixo, enquanto as de fluxo produzem uma sequência pseudo-aleatória, que é combinada com a mensagem a ser cifrada, independente do tamanho. As principais operações realizadas pelos algoritmos simétricos são o ou-exclusivo, a troca de colunas, a troca de linhas, a permutação, a rotação e a expansão, que são operações de baixo custo computacional. Os algoritmos mais conhecidos são DES (*Data Encryption Standard*) e o AES [FERNANDES, 2006], [MARGI, 2009].

Na criptografia assimétrica é exigido um maior custo computacional que a simétrica, por fazer uso de operações como o logaritmo discreto, curva elíptica e fatoração de inteiros, para que o objetivo principal seja alcançado, que é a partir de uma das chaves, não ser possível encontrar a outra. Nessa categoria também são possíveis a distribuição de chaves de forma segura e a assinatura de mensagens. Os algoritmos mais conhecidos são o RSA (*Rivest-Shamir-Adleman*), o *Diffie-Hellman*, e a ECC (*Elliptic Curve Cryptography*) [FERNANDES, 2006].

A principal função da criptografia de chave pública é resolver o problema de distribuição de chaves, e para isso algumas técnicas de distribuição de chaves públicas são utilizadas, sendo que as principais técnicas podem ser agrupadas, de forma geral, em [STALLINGS, 2008]:

- Anúncio Público: utiliza algum algoritmo de chave pública muito aceito, permitindo a qualquer participante enviar sua chave pública por *broadcast*, a outros participantes da rede, mas em contrapartida, qualquer um pode falsificar o anúncio;

- Diretório disponível publicamente: é um diretório dinâmico, disponível publicamente com chaves públicas, mantido e de responsabilidade de alguma entidade confiável, mas se for invadido ou ter sua chave privada descoberta, pode colocar toda a rede em perigo;

- Autoridade de chave Pública: oferece um maior controle na distribuição de chaves públicas para o diretório disponível publicamente, pois cada participante conhece, com segurança, uma chave pública para a autoridade, e a autoridade é a única a conhecer a chave privada correspondente;

- Certificados de Chave Pública: para corrigir o gargalo da Autoridade de chave pública, é possível fazer uso de um certificado para a negociação de chaves entre participantes, de forma segura e sem precisar contatar uma autoridade.

Na criptografia é necessário realizar o gerenciamento de chaves, que é um conjunto de processos e mecanismos que estabelecem relações entre as partes na rede utilizando chaves, além de fazer a manutenção dessas relações, gerando, armazenando, transferindo, carregando e destruindo essas chaves de acordo com uma política de segurança [HE, 2012], [BALA, 2012], [CAMPISTA; DUARTE, 2002].

3.2 Gerenciamento de Chaves

As quatro possibilidades de fazer o gerenciamento de chaves são [FERNANDES, 2006]: a pré-distribuição (é a distribuição das chaves pelos nós interessados antes do início da comunicação), o transporte (as entidades trocam chaves para se comunicar), a arbitração (utiliza um arbitrador central para criar e distribuir chaves entre os participantes) e o acordo de chaves (troca de chaves posterior ao início da rede).

O gerenciamento de chaves pode ser classificado em [BALA, 2012]: soluções dinâmicas ou estáticas; sistema homogêneo (modelo de rede fixa) ou heterogêneo (redes planas e clusters); redes hierárquicas ou distribuídas.

No gerenciamento estático é adotado o princípio de pré-distribuição de chaves, e estas serão fixas durante todo o tempo de vida da rede. Já no gerenciamento dinâmico, as chaves são atualizadas por meio de processos periódicos ou sob demanda, conforme a aplicação da rede. Quando existe algum nó comprometido, esses têm suas chaves revogadas [HE, 2012].

Os Sistemas Dinâmicos de Gerenciamento de Chaves podem ser classificados em dois grupos: Sistema Distribuído de Gerenciamento de Chaves Dinâmicas e Sistema Centralizado de Gerenciamento de Chaves Dinâmicas [HE, 2012].

Sistema Distribuído de Gerenciamento de Chaves Dinâmicas é um conjunto de processos, em que não existe nenhum controlador de chave central, tais como uma estação base (um dispositivo mais poderoso, que geralmente se comporta como uma interface entre os serviços prestados pelos nós e os usuários da rede) ou terceiro. Já o Sistema Centralizado de Gerenciamento de Chaves Dinâmicas é um conjunto de mecanismos que usa um único controlador de chave central (estação base ou terceiro), para gerenciar e substituir chaves de criptografia na rede, ficando globalmente responsável pelo gerenciamento de chaves [HE, 2012].

Em [HE, 2012] são listadas as quatro formas básicas para garantir o processo de segurança no Gerenciamento Dinâmico:

- Revogação de nó: revogando a chave secreta, quando detectar que um nó sensor foi comprometido, gerar uma nova chave e redistribuir aos nós sensores associados, exceto ao que foi comprometido;

- Sigilo para frente e para trás: encaminhar segredo é usado para prevenir um nó de usar uma chave antiga para criptografar novas mensagens, e o sigilo para trás é o oposto, é utilizado para evitar que um nó com a nova chave consiga decifrar as mensagens recebidas anteriormente, criptografada com chaves anteriores [MISHRA, 2002];

- Resistência ao conluio: um adversário pode atacar a rede para comprometer um número de nós da rede, fazendo esses nós conspirar e colaborar a revelar todas as chaves do sistema, prejudicando toda a rede;

- Resiliência: é a mensuração do impacto na rede, quando um nó é capturado e o adversário tenta recuperar informações secretas de sua memória.

As três métricas de eficiência no Gerenciamento Dinâmico visam verificar [HE, 2012]:

- Memória: a quantidade de memória necessária para armazenar as credenciais de segurança, tais como chaves e certificados de confiança.
- *Overhead* de transmissão: o número e tamanho das mensagens trocadas para o processo de geração de chave, e reposição do nó e despejo de nó.
- Energia: o consumo de energia envolvido no processo de acordo de chave, a transmissão e recepção de dados, assim como o processo de cálculo para a geração e distribuição de novas chaves.

Também são listadas três métricas de flexibilidade no Gerenciamento Dinâmico [HE, 2012]:

- Mobilidade: mobilidade de estações base, de nós ou de ambos é necessário em certas aplicações.
- Escalabilidade: as soluções de gerenciamento de chaves dinâmicas devem ser escaláveis para diferentes tamanhos de rede.
- Conectividade de Chave: é definido como a probabilidade em que dois nós (ou mais) são capazes de estabelecer chaves após recodificação. Conectividade local considera a conectividade entre qualquer par de nós vizinhos.

Um ponto importante para o gerenciamento de chaves é a ampla quantidade de cenários abrangidos pela Internet das Coisas, devendo ser considerada a flexibilidade dessas técnicas para que funcionem direito.

Os esquemas distribuídos de gerenciamento de chaves dinâmicas evitam um ponto único de falha e permitem uma melhor escalabilidade da rede. Entretanto, eles são propensos a erros de projeto, porque os nós comprometidos podem participar do processo de despejo de nó [HE, 2012].

No sistema centralizado [HE, 2012] é impossível para os nós comprometidos sabotar o processo de despejo de nó, mas devido às mensagens do processo criptográfico serem encaminhadas por multi-saltos, do controlador aos outros nós, a latência na revogação se torna muito maior que na distribuída.

3.3 Segurança na Internet das Coisas

Com todos os principais conceitos de segurança, criptografia e gerenciamento de chaves contextualizados nos sub capítulos anteriores, é possível visualizar o uso destes na Internet das Coisas. Segundo [HENNEBERT; SANTOS, 2014], a segurança na IoT pode ser realizada em diferentes camadas:

- Camada de Enlace: o padrão IEEE 802.15.4 implementa recursos de segurança para criptografia de dados e autenticação, provida por meio de sete modos de segurança: sendo os principais o AES-CBC-MAC, conjunto de codificação que garante a autenticação a 32, 64 ou 128 bits; AES-CTR, permite a criptografia com cifra de bloco de 128 bytes de comprimento para garantir a confidencialidade; AES-CCM*, combina autenticação com AES-CBC MAC seguido de criptografia AES-CTR; ASH, inclui também um campo "Controle de Segurança", que inclui parâmetros de segurança e um campo "chave identificadora";

- Camada de Rede: *IPsec* (um conjunto de protocolos para autenticar e criptografar cada pacote IP de uma sessão de comunicação; inclui a negociação de chaves criptográficas usadas para a

criptografia.); *Compressed IPsec* para 6LoWPAN (os recursos de segurança podem ser adicionados ao IP usando compressão de cabeçalho e IPHC NHC para a compressão de cabeçalho seguinte) e *IPsec SA* (IKEv2 comprimido leve para IPsec, que é um protocolo para o estabelecimento de uma chave de sessão entre dois pares, baseado em criptografia de curva elíptica. O protocolo de Diffie-Hellman é usado em ambos os casos para troca de chave);

- Camada de Transporte/Aplicação: *Datagram Transport Layer Security*, um protocolo usado para proteger o tráfego da rede, que será apresentado no próximo sub capítulo.

3.4 Datagram Transport Layer Security (DTLS)

O *Datagram Transport Layer Security* é um protocolo padronizado de segurança que automatiza o gerenciamento e distribuição de chaves, a autenticação e a encriptação dos dados. Enquanto o *Transport Layer Security* (TLS) fornece segurança para TCP e exige um transporte confiável, usado em aplicações como o HTTP, o DTLS protege e utiliza protocolos orientados a datagrama, como UDP, utilizado na Internet das Coisas.

Ambos os protocolos são intencionalmente similares e compartilham a mesma configuração e conjuntos de codificação. O DTLS utiliza todos os recursos do TLS, mas introduz importantes modificações necessárias para superar a insegurança enfrentada ao se utilizar o UDP como protocolo de transporte [HEER; ET AL, 2011], [VLADISLAV, 2012].

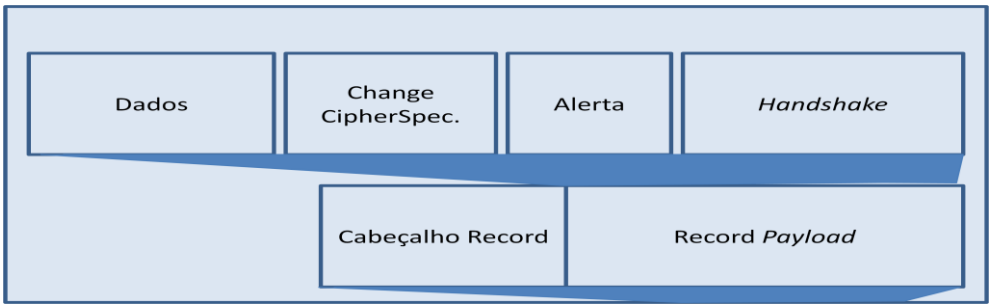

Figura 6 - Pacote com o DTLSe
Fonte: Adaptado de RAZA; ET AL. 2013

O DTLS é usado para proteger o tráfego de *datagramas* para aplicações cliente/servidor. É composto de um protocolo *Record* que carrega outros protocolos como o *Alert*, *ChangeCipheSpec*, *Handshake* e *Data*, apresentados na Figura 6.

O *Handshake* é um protocolo organizado em *flights*, usado para negociar chaves de segurança, conjuntos de codificação e métodos de compressão. O *handshake* inicial autentica o servidor e,

opcionalmente, o cliente pode usar uma infraestrutura de chave pública (PKI - *Public Key Infrastructure*) [RAZA; ET AL. 2013], [HENNEBERT; SANTOS, 2014]. O *handshake* completo é apresentado na Figura 7.

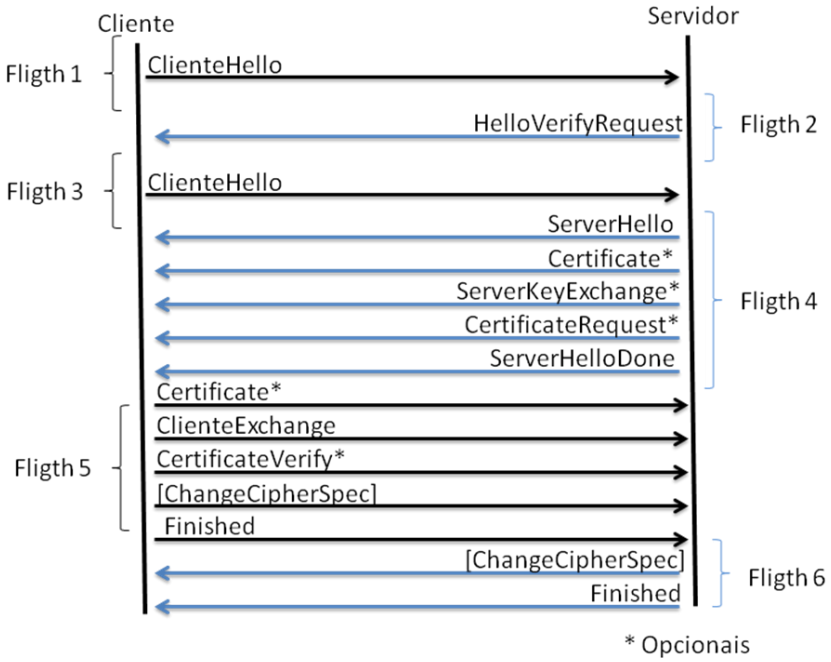

Figura 7 - As seis mensagens *flights* de um *full Handshake* no DTLS
Fonte: Adaptado de HARTKE, 2012 e HUMMEN; RAZA; ET AL, 2014

Para que o *Handshake* possua confiabilidade, o DTLS utiliza algumas abordagens que garantem isso no UDP [RESCORLA; MODADUGU, 2006]:

- Perdas de Pacotes: na primeira fase do *handshake*, assim que o cliente tenha transmitido a mensagem *ClientHello*, ele espera ver um *HelloVerifyRequest* a partir do servidor. No entanto, se a mensagem de servidor é perdida, o cliente sabe que, ou o *ClientHello*, ou o *HelloVerifyRequest* foi perdida e a retransmite. Quando o servidor recebe a retransmissão, ele sabe quem a retransmitiu. O servidor também mantém um timer de retransmissão e retransmite quando o timer expirar.
- Reordenamento: em DTLS, cada mensagem *handshake* tem atribuído um número a uma sequência específica dentro desse *handshake*. Quando um nó recebe uma mensagem *handshake*, ele pode rapidamente determinar se a mensagem é a próxima mensagem de espera ou não. Se for, processa, se não, enfileira-o para um futuro tratamento.
- Tamanho da mensagem: mensagens de *handshake* do TLS e DTLS podem ser muito grandes. No entanto, os *datagramas* do UDP são muitas vezes limitados a fragmentação. Para compensar esta limitação, cada mensagem *handshake* no DTLS pode ser fragmentada

> ao longo de vários registros DTLS. Cada mensagem *handshake* contém um fragmento comprimido e um com o comprimento da mensagem original. Assim, um beneficiário na posse de todos os bytes de uma mensagem *handshake* pode remontar a mensagem original desfragmentada.

O protocolo *Alert* pode ser usado durante o *Handshake* para reportar erros ou avisos [HENNEBERT; SANTOS, 2014]. As mensagens de alerta (*Alert*) podem ser geradas caso o registro seja ilegalmente recebido de novo, além de serem usadas para detectar quando um ponto é persistente em enviar mensagens ruins, e encerrar o estado de conexão local [RESCORLA; MODADUGU, 2006].

O protocolo *ChangeCipherSpec* habilita a alteração do pacote corrente de cifras no DTLS [HENNEBERT; SANTOS, 2014]. Similar ao TLS, a mensagem *ChangeCipherSpec*, não é tecnicamente uma mensagem *Handshake*. Entretanto, é associada como parte do mesmo *flight*[2] da mensagem do estado *Finished*, para efeitos de tempo de espera e retransmissão [RESCORLA; MODADUGU, 2006].

No campo *Data* os dados da aplicação podem ser fragmentados, comprimidos, e encriptados com o modo ou configuração de segurança em progresso [HENNEBERT; SANTOS, 2014].

Todas as mensagens enviadas via DTLS são precedidos com cabeçalho Record de 13 bytes. Este cabeçalho especifica o conteúdo da mensagem (por exemplo, dados de aplicativo ou *handshake* de dados), a versão do protocolo utilizado, bem como um número de sequência de 64 bits e o comprimento do registro [KOTHMAYR; ET AL. 2013].

O DTLS utiliza três estados básicos de tempo: *PREPARING* (a implementação faz todos os cálculos necessários para preparar o próximo lance de mensagens), *SENDING* (depois de ter o *buffer* de mensagens limpos pela ação anterior, ele transmite os *flights* de mensagens) e *FINISHED* (quando todas as mensagens foram enviadas, sendo o último processo do *handshake*) [RESCORLA; MODADUGU, 2012].

Existem quatro configurações possíveis de segurança no uso do DTLS [KEOH; ET AL, 2014]:

- *NoSec*: nessa configuração não há segurança e o protocolo DTLS está desativado.
- *PreSharedKey*: o DTLS está habilitado e a autenticação *PreShared Key* (PSK) será usada; o dispositivo nessa configuração será fornecido com uma lista de chaves e, a cada chave, é incluído uma lista de nós para que esta chave possa ser usada.

[2] ***Flights***: ou vôos são mensagens ou subdivisões integrantes do *handshake*.

- *RawPublicKey*: o DTLS está habilitado e os dispositivos usam um par de chaves assimétricas, mas essas chaves não estão dentro de um certificado X.509. O dispositivo também tem uma lista de nós com quem pode se comunicar.

- *Certificate*: DTLS está habilitado com pares de chaves assimétricas e o certificado X.509 é usado para realizar o papel de uma autoridade certificadora. O dispositivo também tem uma lista de âncoras de confiança para que os certificados de validação de caminho recebidos de outras entidades possam ser realizados.

De acordo com [KEOH; ET AL, 2014], quando o DTLS é usado para Internet das Coisas, pode-se atingir as seguintes proteções de segurança:

- Acesso a Rede: o 6LoWPAN *Border Router* (6LBR), um dispositivo semelhante a um servidor de acesso à rede em implantação na WiFi regular, é idealmente responsável pela autenticação e autorização de dispositivos antes de autorizá-los a aderir à rede. O *handshake* do DTLS bem sucedido cria um canal seguro entre o novo dispositivo e a entidade gestora (por exemplo, o 6LBR).

- Canal de Comunicação Seguro: uma sessão DTLS ponto- a -ponto pode ser estabelecida entre dois dispositivos de comunicação, um dentro do 6LoWPAN e a outra fora, para transportar com segurança os dados de aplicativos (mensagens CoAP). Os dados do aplicativo estão protegidos pela camada de registro DTLS, ou seja, autenticada e criptografada com uma chave de sessão recente e única.

- Gerenciamento de Chaves: como o DTLS tem a capacidade de renovar as chaves de sessão, este mecanismo pode ser utilizado para apoiar o gerenciamento de chaves em uma rede 6LoWPAN. Durante a fase de acesso à rede, a 6LBR distribui uma chave de L2 (camada 2, ou camada de enlace) como parte do procedimento de autenticação de acesso à rede, possibilitando a reutilização desta no mesmo canal para facilitar o gerenciamento de chaves, permitindo que a chave de enlace (L2) seja atualizada pela 6LBR quando necessário.

3.5 Trabalhos Relacionados

Em [RAZA; ET AL. 2012], os autores propuseram a compressão de cabeçalho para 6LoWPAN e DTLS. Eles ligaram o modelo compactado de DTLS deles com o 6LoWPAN usando mecanismos padronizados. Concluem que a compressão de DTLS proposta reduz significativamente o número de bits

de segurança adicionais, cerca de 62%. Propuseram o uso do 6LoWPAN-GHC para comprimir os cabeçalhos *Record* e *Handshake.*

Já em [RAZA; ET AL. 2013], os autores apresentam uma adaptação e integração de DTLS e CoAP para a Internet das Coisas. Com *Lithe*, os autores também propõem uma nova compressão do cabeçalho DTLS, que visa reduzir significativamente o consumo de energia, utilizando mecanismos de compressão 6LoWPAN-NHC (os bits de identificação são usados para diferenciar do 6LoWPAN-GHC), que pode reduzir significativamente o comprimento de cabeçalhos DTLS. Com essa compressão, é possível reduzir 62% no cabeçalho *Record* e 75% no *handshake*. A compressão de cabeçalho no DTLS, como IPHC, só é aplicada dentro das redes 6LoWPAN, isto é, entre nós sensores e o 6BR. Isto é porque os cabeçalhos DTLS fazem parte da carga útil UDP e todas as informações necessárias para o encaminhamento já são extraídas na camada IP.

Segundo [RAZA; ET AL. 2013], o mais importante, é que essa proposta de compressão de cabeçalho DTLS não compromete as propriedades de segurança fornecidas pelo DTLS a uma rede ponto - a- ponto. Simultaneamente, ainda reduz o número de bytes de transmissão, mantendo o DTLS em conformidade com o padrão. Eles ainda avaliaram essa abordagem baseada em uma implementação DTLS para o Sistema Operacional Contiki. E segundo a pesquisa, os resultados da avaliação mostraram ganhos significativos em termos de tamanho de pacote, consumo de energia, tempo de processamento, e tempos de resposta em toda a rede quando o DTLS comprimido é habilitado.

Em [KOTHMAYR; ET AL. 2013] os autores apresentam o primeiro esquema de segurança de autenticação de duas vias (*two-way*) totalmente implementado para a Internet das Coisas, com base em padrões da Internet existentes, especificamente o protocolo DTLS. Esse esquema de segurança proposto é baseado em RSA. Ele é projetado para trabalhar sobre pilhas de comunicação padrão que oferecem UDP/IPv6, o 6LoWPANs. A implementação de DTLS foi apresentado no contexto de uma arquitetura de sistema e de viabilidade do esquema, demonstrado por meio da avaliação em uma plataforma de hardware adequada para a Internet das Coisas.

Na arquitetura de segurança de [KOTHMAYR; ET AL. 2013], é introduzido um servidor de controle de acesso (AC - *Access Control*) - que é uma entidade confiável - e um servidor mais robusto em recursos, em que os direitos de acesso para os editores (nó elo) da rede são armazenados de forma segura. A identidade de um assinante padrão geralmente é pré-configurada em um editor antes que seja implantado. O AC verifica que o assinante tem o direito de acessar à informação disponível do editor. O editor, em seguida, só tem que avaliar a identidade do assinante e verificar o bilhete que recebeu a partir

da AC. Na Internet, as identidades são usualmente estabelecidas por meio de uma infraestrutura de chaves públicas e os identificadores fornecidos por meio de certificados X.509. O Certificado X.509 contém, entre outras informações, a chave pública de uma entidade e seu nome comum. O certificado é assinado por um terceiro, chamado de Autoridade Certificadora, que serve, em primeiro lugar, para que a assinatura permita ao receptor detectar modificações no certificado, e em segundo lugar, também indica que a AC verificou a identidade da entidade que solicitou o certificado. A Figura 8 apresenta esta arquitetura, que exemplifica o acesso de informações de um assinante a uma editora.

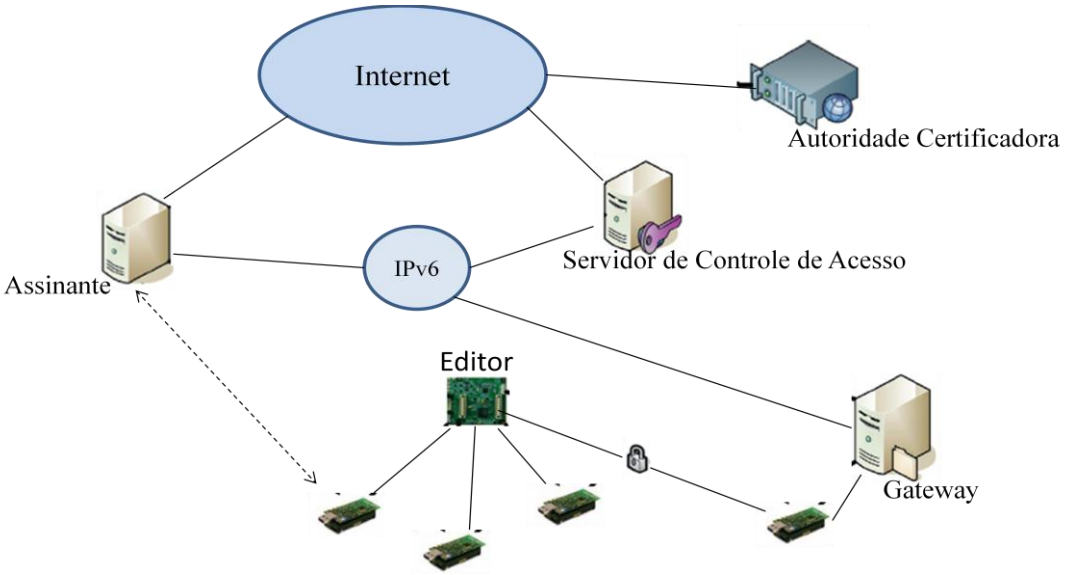

Figura 8 - Arquitetura de Segurança de KOTHMAYR; ET AL
Fonte: Adaptado de KOTHMAYR; ET AL, 2013

Em [BERGMANN; ET AL, 2012] apresentam um processo de três passos (descoberta, impressão e configuração) para inicialização de um nó com três protocolos - CoAP, IPv6 e DTLS - usados em dispositivos com recursos restritos. As fases desses passos incluem descoberta de serviços, distribuição de credenciais de segurança e configuração do aplicativo num nó específico, usando interfaces de usuário barato, tais como botões *push* e diodos emissores de luz.

Os autores dividiram o processo de inicialização em três fases, cada um dos quais é concebido para reduzir, até certo ponto, a vulnerabilidade global do sistema. A primeira fase (descoberta) compreende a detecção de um nó na rede que pode ajudar um novo nó em *bootstrapping*[3]. Na segunda fase (impressão), o novo nó é fornecido com o material de chave para estabelecer um canal seguro entre uma configuração servidor e o novo nó. A configuração real do novo nó é realizada durante a terceira fase, usando o canal seguro que foi configurado antes (configuração) [BERGMANN; ET AL, 2012].

[3]***Bootstrapping***: no trabalho de [BERGMANN; ET AL, 2012] é a configuração de um nó para permitir a sua participação na rede de operação normal.

Em [VUCINIC; ET AL. 2015] é sugerido um modelo de arquitetura de segurança que remove a noção de estado entre servidor e cliente. Os autores utilizam o modelo produtor / consumidor em sua proposta de arquitetura chamada OSCAR (*Object security architecture*). Nesta arquitetura os principais componentes são os Produtores (nós utilizando CoAP que providenciam os dados para serem criptografados), Consumidores (nós clientes que também utilizam CoAP, que requisitam os dados dos Produtores), Servidores de Autorização (entidades confiáveis que armazenam os certificados dos Produtores) e Servidores *Proxy* (providenciam a troca entre os Produtores e Consumidores).

Segundo [VUCINIC; ET AL. 2015], é possível abstrair IoT, seus sensores e atuadores, como uma interface para o mundo físico. Tomadores de Decisão (usuários humanos, centros de inteligência, ou os próprios dispositivos de atuação limitados) baseiam o seu raciocínio em dados de entrada provenientes dos fenômenos físicos detectados. Os autores acreditam que o modelo produtor-consumidor representa bem esse problema também em termos de segurança, ilustrado na Figura 9.

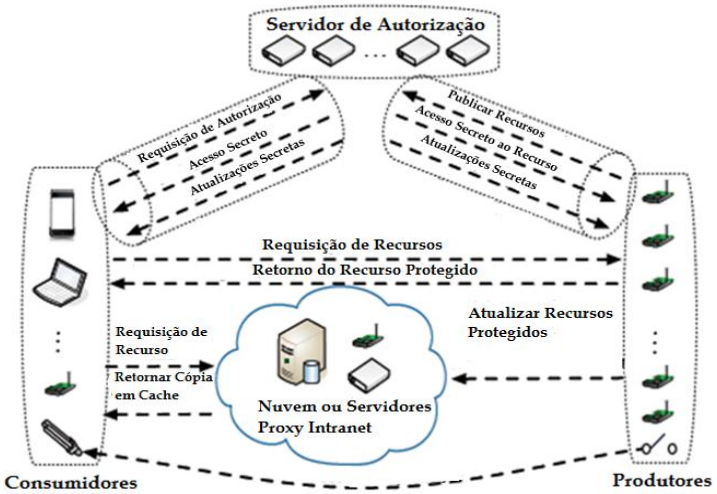

Figura 9 - OSCAR baseada na arquitetura Produtor/Consumidor
Fonte: Adaptação de VUCINIC; ET AL, 2015

Produtores (medidores inteligentes, sensores tradicionais, detectores de movimento, etc.) alimentam aos consumidores as informações necessárias. Já os Consumidores (dispositivos de comando, centros de recolha, os usuários humanos) recolhem essas informações e podem gerar mais ações. A inspiração para a utilização desse modelo vem de *Cloud Computing* e um recente trabalho de [JUNG; ET AL., 2013] no controle de acesso de dados. Entretanto, uma diferença importante é que os produtores, no caso da IoT não são responsáveis pelas decisões de controle de acesso para o conteúdo que eles geram, e sim uma política do operador de rede. [VUCINIC; ET AL. 2015].

Além do modelo produtor-consumidor, alguns autores trabalham com o modelo de delegação, ambas arquiteturas de segurança que têm sido propostas para a Internet das Coisas.

Outra contribuição relevante para essa pesquisa está em [HUMMEN; RAZA; ET AL. 2014], em que detalha o impacto da chave-pública utilizada na manutenção do protocolo DTLS na memória, requerida em dispositivos com limitações de recursos. Propuseram uma arquitetura de segurança por delegação, que habilita esses dispositivos a se comunicarem com segurança em domínios independentes, em que um servidor estabelece a conexão com o DTLS, ou seja, delegando os processos de segurança a esse servidor.

Para [HUMMEN; RAZA; ET AL, 2014] o protocolo DTLS já fornece uma alternativa eficiente para criptografia de chave pública, com o suporte para o *handshake* à base de chave simétrica. No entanto, o *handshake* à base de chave simétrica requer chaves secretas para serem pré-compartilhadas e prontamente disponíveis em ambos os pontos finais de comunicação, ao estabelecer uma conexão segura fim- a- fim. Em outras palavras, esse *handshake* requer um mecanismo de provisionamento de chave que implanta de forma segura informações secretas aos pontos finais antes de uma conexão DTLS segura poder ser estabelecida. Para solucionar esse dilema, especialmente quando os pontos finais pertencem a domínios de redes independentes e se comunicam por meio de infraestrutura não confiável, os autores propuseram um projeto de uma arquitetura de delegação como um mecanismo simples de provisionamento de chaves.

O principal bloco de construção dessa arquitetura de delegação proposta é a introdução de um servidor de delegação, que permite separar o estabelecimento da conexão DTLS inicial da proteção posterior de dados da aplicação, conforme apresentado na Figura 10. A principal tarefa do servidor de delegação é fornecer um dispositivo com limitações de recursos com o contexto de segurança necessário para se comunicar de forma segura com um terminal remoto. Esse servidor de delegação tem um papel semelhante ao de um centro de distribuição de chaves em protocolos de segurança de rede local, como o *Kerberos* [HUMMEN; RAZA; ET AL, 2014].

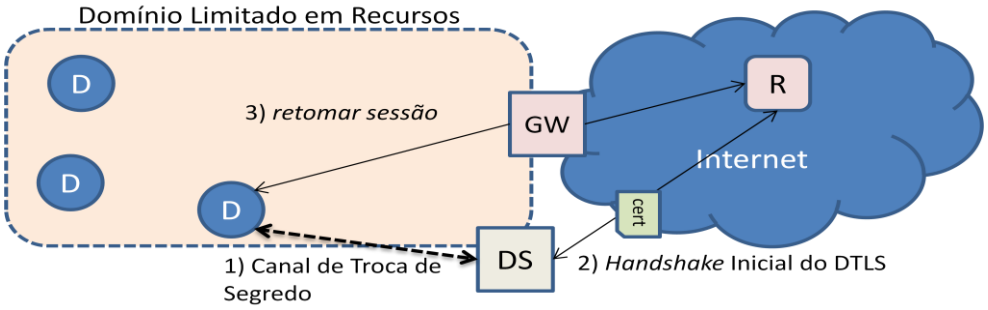

Figura 10 - Arquitetura por Delegação
Fonte: Adaptação de HUMMEN; RAZA, 2014

No entanto, em contraste com estas abordagens, esse servidor de delegação proposta pelos autores, não mantém chaves secretas pré-compartilhadas para todos os parceiros de comunicação possíveis, mas em vez disso, estabelece um contexto *on-demand* de segurança com um ponto final remoto para alcançar uma solução escalável, com relação ao grande número de serviços de Internet que um dispositivo restrito pode se comunicar com potencial. Para estabelecer este contexto de segurança *on-demand*, o servidor de delegação age em nome de um dispositivo com limitações de recursos durante um *handshake* com base em certificado. O servidor de delegação, em seguida, entrega o contexto de segurança estabelecido para o dispositivo restrito [HUMMEN; RAZA; ET AL, 2014].

Em [ROMAN, ET AL., 2011], os autores estudaram as possíveis soluções para o problema de estabelecer uma chave de sessão entre um cliente e um servidor no contexto da Internet das Coisas, onde um ou mais pares são nós sensores de uma rede de sensores sem fio. Este problema não é trivial, pois as limitações inerentes de muitos nós sensores podem dificultar a aplicação de mecanismos de gerenciamento de chaves existentes. Nessa análise, eles tentaram aplicar os principais sistemas existentes de gestão especializados no estabelecimento de chaves na camada de enlace entre os nós vizinhos, como a estratégia de chave pré-compartilhada.

Na estratégia de chave pré-compartilhada, os clientes e os servidores devem partilhar algum material pré-estabelecido de chaveamento, que pode ser usado para derivar uma chave compartilhada entre os pares. Na verdade, em um cenário de Redes de Sensores, existem várias abordagens para implementar esta estratégia particular estudadas por [ROMAN, ET AL., 2011], e ilustradas na Figura 11:

- 1-1: Nesta abordagem, um grupo de clientes (Gc) compartilham a mesma chave pré-compartilhada (kGcGs) com um grupo de servidores (Gs).

- 1-s: Nesta abordagem, cada servidor (s) tem sua própria chave pré-compartilhada (Ks). Consequentemente, os clientes devem guardar uma chave por servidor que eles se conectarem.

- c-1: Nesta abordagem, cada cliente (c) tem a sua própria chave pré-compartilhada (Kc). O resultado disso é que os servidores devem saber com antecedência quais são os clientes que tentam se conectar a eles, e por isso armazenam uma chave por cliente.

- c-s: Nesta abordagem, cada cliente individual (c) e servidor (s) compartilham de uma chave pré-compartilhada comum (Kcs). Esta abordagem combina as desvantagens de ambos 1-s e c-1.

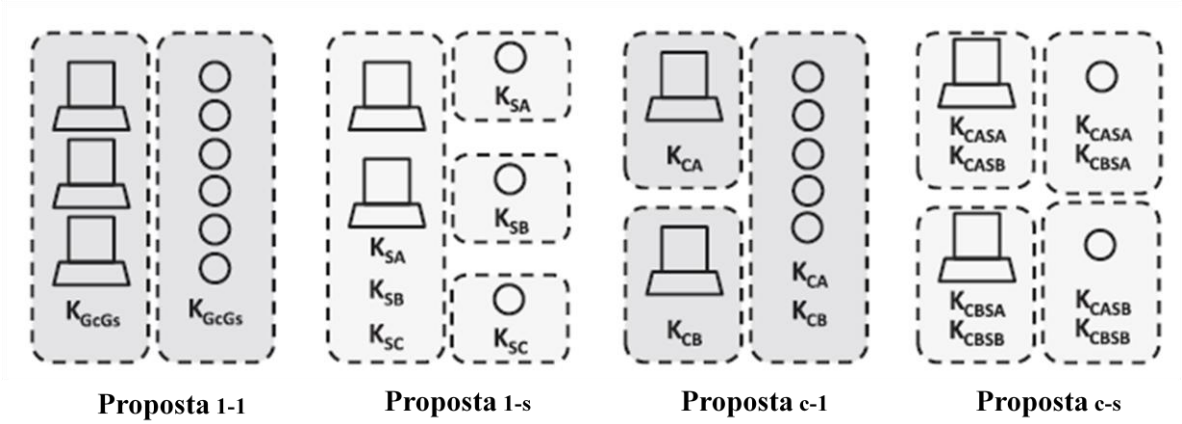

Figura 11 - Abordagem de Compartilhamento de Chaves Pré-Compartilhadas
Fonte: Adaptado de ROMAN, ET Al., 2011

Estas abordagens podem ser aplicadas ao DTLS, já que este utiliza o modo de segurança PSK, com chaves pré-compartilhadas. Mas existem algumas desvantagens no uso destas abordagens, e de acordo com [ROMAN, ET AL., 2011], uma das principais está relacionada com a propriedade de distribuição. Todas as informações secretas devem ser pré-carregadas antes de qualquer troca de informações, portanto, só é possível conectar com segurança clientes e servidores que já se conhecem mutuamente. Outra desvantagem é a resiliência global das abordagens, pois um elemento (clientes em 1-s, os servidores em c- 1, todos em 1-1) irá armazenar todas as chaves pré-compartilhadas, como esse elemento é o elo mais fraco da cadeia de segurança, qualquer adversário que ganhar o controle dele assumirá o controle de toda a rede. Além disso, é necessário considerar que a chave pré-compartilhada será reutilizada a cada vez. Não só é aconselhável a utilização de um mecanismo que derive uma chave de sessão a partir da tal chave pré-compartilhada, como também deve ser atualizado ao longo do tempo.

Entretanto, os benefícios de usar qualquer uma das abordagens de chave pré-compartilhada em um contexto de IoT, ainda são significativas. A sobrecarga computacional é insignificante, pois a chave pré-compartilhada já é pré-carregada e não há necessidade de se engajar em negociações de alto custo. A escalabilidade de todas as abordagens é alta. Na propriedade de autenticação, é possível proporcionar a autenticação do cliente (C-1) ou autenticação do servidor (1-s), embora todas as abordagens fornecem autenticação do grupo. E finalmente, a extensibilidade de todas as abordagens é alta [ROMAN, ET AL., 2011].

Um trabalho interessante utilizado nesta pesquisa foi a [HARTK; BERGMANN, 2012], que consideraram alguns obstáculos na implementação da camada do DTLS em ambientes com restrições, e apresentou algumas idéias para uma versão restrita do DTLS que é amigável para redes com poucos recursos computacionais.

As idéias estão enquadradas nas seguintes categorias [HARTK; BERGMANN, 2012]:

- Diretrizes para implementação: uso de técnicas de implementação para atingir DTLS mais leves, sem afetar a conformidade com as especificações ou interoperabilidade relevantes com outras implementações. Incluindo técnicas para reduzir a complexidade, o consumo de memória, ou o uso de energia.
- Compressão de cabeçalho apátrida: os registros do DTLS ficam comprimidos explicitamente sem a construção de qualquer contexto de compressão. Isso ocorre utilizando formas mais curtas para representar os mesmos bits de informação ou depender de informação que já é compartilhada por cliente e servidor.
- Perfil Protocolo: O uso do DTLS de um modo particular, por exemplo, as implementações existentes do DTLS deveriam continuar a serem utilizados sem alterações importantes, pois eles podem ser configurados conforme a necessidade da rede.

Uma das idéias propostas pelos autores [HARTK; BERGMANN, 2012] são *handshakes* reduzidos: considerando que um *handshake* tem dois *flights* adicionais quando comparado com TLS, resultante da adição de uma troca de *cookies* sem estado. Esta troca é projetada para evitar certos ataques de negação de serviço DoS (*Denial of Services*). Se o conjunto de codificação permite que o servidor permaneça apátrida após o envio do *ServerHello..ServerHelloDone* (presentes nos *flights* 1 e 2), então estes serviços se encaixam em um datagrama, permitindo que um *handshake* completo talvez pudesse ser encurtado para quatro *fligths* (ou seja, retirar os *fligths* 2 e 3, incluindo os serviços destes no *flight* 1), conforme apresentado na Figura 12.

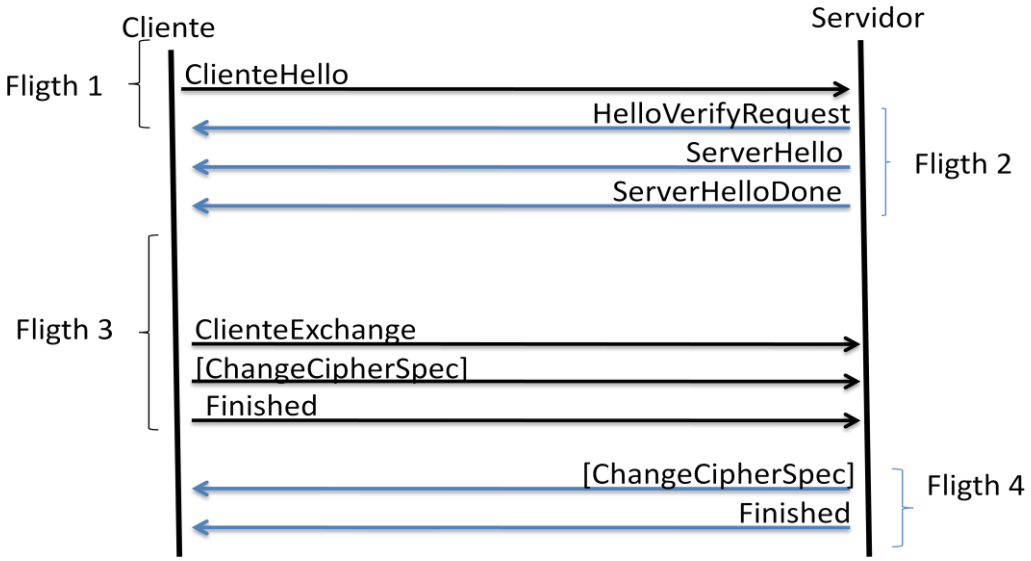

Figura 12 - *Handshake* reduzido
Fonte: Adaptado de HARTK; BERGMANN, 2012

Entretanto nenhuma das pesquisas correlatas estudaram a arquitetura da rede com todos os protocolos da pilha nos principais cenários da Internet da Coisas, que será focado e apresentado nessa pesquisa, utilizando o *Lithe* para a realização dos testes destes cenários, considerando as abordagens estudadas pelas pesquisas anteriores.

IV - METODOLOGIA

Neste capítulo são apresentados o ambiente de simulação Contiki, as ferramentas e parâmetros utilizados nesta pesquisa, além dos cenários definidos para os testes realizados.

5.1 Ambiente de Simulação Contiki

O Sistema Operacional *Contiki*, é um open source pioneiro para a Internet das Coisas, orientado a eventos. Com custos mínimos em uso de memória, processamento e armazenamento, o sistema vem acompanhado de bibliotecas e aplicações, escritas em linguagem C, para uso nas placas sensores. Possui a capacidade de carregar e descarregar aplicações no equipamento em execução ou ser emulado no simulador *Cooja*, antes de ser gravado [DUNKELS, 2004], [TOHEED; RAZI, 2010].

Contiki possui grande portabilidade, podendo ser utilizado em diversos dispositivos sem fio de baixa potência, dos quais muitos podem ser encontrados facilmente pela internet [CONTIKI-OS.ORG]. Ele comporta os padrões IPv6 e IPv4, juntamente com os últimos padrões sem fio de baixo consumo: 6LoWPAN, RPL, CoAP. O sistema *Contiki*, desenvolvido no Instituto Sueco de Ciência da Computação, é dividido em duas partes [WAGENKNECHT, 2013]: o núcleo e os programas carregáveis, como mostra a Figura 13.

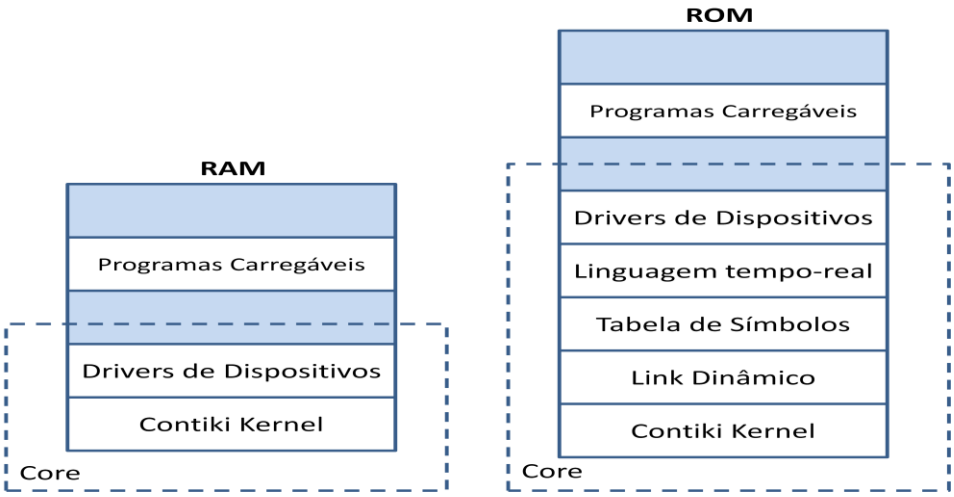

Figura 13 - Partes do Contiki
Fonte: Adaptado de WAGENKNECHT, 2013

O núcleo contém o *kernel*, *drivers* de dispositivo, um conjunto de aplicativos padrão, partes de bibliotecas em linguagem C e uma tabela de símbolos. Já os programas carregáveis são os demais programas que não modificam o núcleo. Ele não fornece uma camada de abstração de hardware, mas os *drivers* de dispositivo e aplicativos podem se comunicar diretamente com o *hardware*. Essa comunicação

entre os processos, sempre passa pelo *kernel* [WAGENKNECHT, 2013]. A pilha µIP[4] é responsável pelas definições de TCP/IP, e a Rime[5] controla as transmissões de rádio nos sensores [DUNKELS; OSTERLIND, 2008].

Outra característica importante é o *Power Trace*, que usa um rastreamento de estado de energia para estimar o consumo de energia do sistema. O *Power Trace* é um sistema para criação de perfis de energia em nível da rede nas redes sem fio de baixa potência, permitindo não somente a estimativa de energia em tempo de execução, mas também de perfis que mostram o custo de energia por atividade. Isso garante aos desenvolvedores de sistemas os meios para otimizar seus sistemas em relação do consumo de energia, já que com o Power Trace eles são capazes de identificar o pior dos consumidores de energia e avaliar todas as economias que suas otimizações podem fazer ao sistema [DUNKELS; ERIKSSON, 2011].

Power Trace utiliza um modelo linear de potência, em que a potência instantânea é estimada como a soma de todos os estados de energia ativos. Esse sistema de energia é derivado a partir do momento em que o sistema gasta seu primeiro estado de energia [DUNKELS; ERIKSSON, 2011].

A energia para as atividades individuais, como transmissões de pacotes e recepções, ou a leitura de um bloco de dados de um arquivo em memória flash externa, são capturadas em cápsulas de energia, como apresentado na Figura 14. As cápsulas de energia contém uma representação da energia de uma atividade, e podem ser atribuídas a aplicações, protocolos ou outras atividades, como controle de tráfego [DUNKELS; ERIKSSON, 2011]. Nele são utilizados o modelo de *broadcast* para envio das mensagens e, durante a transmissão, são efetuados os cálculos de consumo energético para cada modelo de dispositivo.

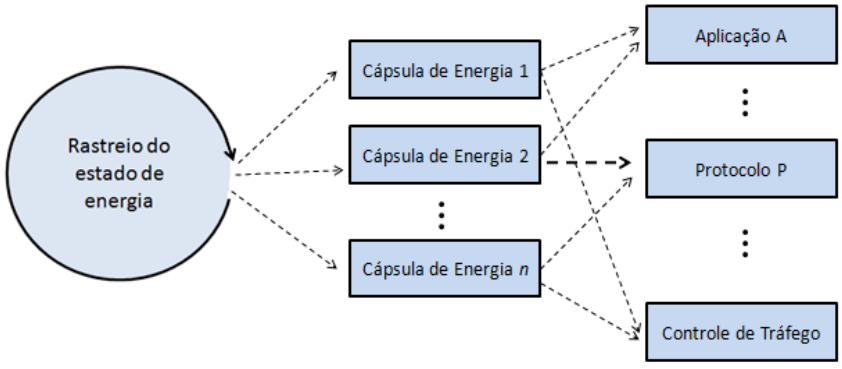

Figura 14 - Processo do *Power Trace*
Fonte: Adaptado de DUNKELS; ERIKSSON, 2011.

[4] **µIP**: Protocolo utilizado para sinalizar a chegada de novos pacotes, implementado em cima do Contiki [WAGENKNECHT, 2013].
[5] **Rime**: Pilha de rede para rádio de baixa potência. A pilha de protocolos Rime fornece um conjunto de primitivas de comunicação, que vão desde transmissão *unicast* a um vizinho local, até um ponto- a- ponto confiável [SAADALLAH, 2012].

O *Power Trace* grava o consumo de energia da atividade por meio da abertura de uma cápsula de energia assim que a atividade se inicia e a fecha quando essa atividade termina. Essa cápsula de energia pode ser reaberta várias vezes para que sejam gravados estados de operações de nível inferior separados no tempo naquela atividade, garantindo a verificação precisa de consumo de cada atividade e sub-atividade na rede [DUNKELS; ERIKSSON, 2011].

Na ferramenta podem ser encontrados vários tipos de *mote* (nós sensores) de diferentes fabricantes*: java mote, cooja mote, MicaZ mote, EXP430F5438 mote, wismote mote, Z1 mote, Sky mote e ESB mote,* entre outros. Esses modelos são baseados em placas reais, de acordo com os padrões do fabricante, ou permitem implementar modelos que ainda não estão disponíveis na plataforma [DUNKELS, 2004].

5.2 Ferramentas

O equipamento utilizado para realizar as simulações e testes foi um Notebook Samsung *Active Book* 6, com Intel Core i5-3230M CPU @ 2.60GHz, 7,9GB RAM, Intel HD *Graphics* 4000, Microsoft Windows 8 *Single Language* 64-bit.

Para a utilização do Contiki, a ferramenta de virtualização de máquina VMware Player, *versão* 6.0.2 build-1744117, *da VMware, Inc.,* foi necessária. A versão do InstantContiki (pacote que contém o Sistema Operacional Ubuntu 12, ambiente de simulação Cooja, ferramentas do Contiki) utilizadas nos testes foi a 2.7, e os pacotes de aplicações no Cooja foram: PowerTrace, ER-CoAP, IPv6 e UDP-IPv6. Para que a rede simulada trabalhasse com segurança, o pacote de implementação DTLS do *Lithe* [RAZA; ET AL. 2013], que não está disponível no Contiki, foi instalado e utilizado nos testes.

5.3 Principais arquivos do protocolo DTLS

Os principais arquivos do protocolo DTLS modificados nessa pesquisa, têm suas funcionalidades resumidas a seguir, para melhor compreensão dos mesmos:

- *config.h*: neste arquivo são definidos o número máximo de pontos com DTLS; o número máximo de *cipher context* que podem ser usadas paralelamente; o número máximo de funções *hash* que podem ser usados em paralelo; o tamanho máximo de *buffer* para as mensagens DTLS, entre outros parâmetros pertinentes.

- *Makefile*: este arquivo trabalha em conjunto com o arquivo project-config.h e define as regras utilizadas no DTLS, como a habilitação ou não de outros protocolos (CoAP, IPv6, RPL, etc); os

endereços dos clientes na rede, além de chamar outros pacotes e bibliotecas necessárias para seu funcionamento.

- *project-config.h*: são definidos os padrões do DTLS, se será comprimido ou não, a codificação de rotas, cabeçalhos, definições de transações CoAP e outras habilitações inerentes do cabeçalho do DTLS.

- *dtls-client.c e dtls-server*: implementam o DTLS nos sensores no modo cliente/servidor, respectivamente; funcionam em combinação com os arquivos *coap-client* e *coap-server*.

- *coap-client, coap-server*: são implementados os processos CoAP para a comunicação da rede.

5.4 Parâmetros

O modelo do canal de rádio UDGM (*Unit Disk Graph Model*) foi usado nas simulações (que define o alcance de transmissão de cada sensor) e a pilha de protocolo utilizada foi composta dos protocolos: CoAP; DTLS; UDP; 6LoWPAN e IEEE 802.15.4, conforme a Figura 15.

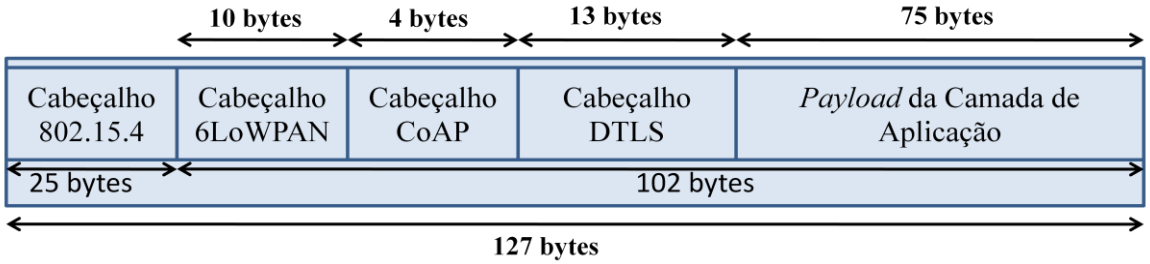

Figura 15 - Pacote de Protocolos
Fonte: Adaptado de GRANJAL, ET AL., 2015

Depois de realizar os primeiros testes com o *Power Trace* e entender melhor sua dinâmica, foram extraídas as linhas de código responsáveis pelos cálculos de consumo, e inseridas nos arquivos do DTLS, disponibilizada por [RAZA; ET AL. 2013], selecionada por utilizar os protocolos da pilha estudada. Devido à necessidade de memória, o nó em que os testes são realizados é o *Wismote* (16 MHz, MSP430, microcontrolador 16-bit RISC, 128/16 kB de ROM/RAM e transceptor IEEE 802.15.4 [CC2520]), por ser mais robusto em memória e processamento, necessários para a utilização do DTLS.

Em todos os testes, o algoritmo de criptografia utilizado é o AES e o tamanho de chave é de 128 bits. Outro elemento mantido como padrão em todos os testes foi a não utilização de certificado no protocolo de segurança e no *handshake*.

Os arquivos *dtls-client.c*, *dtls-server.c* receberam em suas linhas de código os comandos do *PowerTrace* do Contiki, além de ter seus IPs alterados para não haver conflitos entre eles durante as

simulações com mais de um cliente. Em testes preliminares foi verificado que o servidor só conseguia receber transmissões do primeiro cliente que lhe solicitasse, fazendo o *handshake* somente com este e ignorando os demais. Por conta dessa alteração de IP, foram criadas cópias extras dos arquivos *coap-client*, já que cada arquivo coap é considerado um ente da simulação, ligado ao seu respectivo arquivo *dtls-client*.

Para que fosse possível a extração dos dados de consumo por *fligths* do *handshake*, a linha de código do *Power Trace*, incluída nos arquivos do servidor e do cliente, teve duas configurações possíveis, sendo uma utilizando 1 (um) segundo e outra com 0.5 (meio) segundo, definindo o *delay* para calcular as cápsulas de energia. Conforme a necessidade de avaliação de determinado *fligth*, a posição da linha de código do *Power Trace* foi deslocada nos arquivos.

Para calcular o consumo de energia em *mJ* [RAZA; ET AL, 2013], foi usada a Equação 1, em que I é a corrente (medida em mA) e *ticks* são os ciclos do relógio do processador:

$$\text{Energia } [mJ] = \frac{Ticks \text{ x I}[mA] \text{ x Tens ão}[V]}{Ticks \text{ por Segundos}} \text{ (Equação 1)}$$

Devido à utilização do *wismote*, os valores padrões do MSP430 que foram considerados na fórmula para calcular o consumo de energia foram: para corrente o valor máximo de 312 *m*A e para tensão foi de 3,6 V, conforme especificações do fabricante em [MSP430, 2014]. Como as saídas do *Cooja* apresentam o percentual destes valores e não o valor numérico, foi necessário a conversão destes percentuais com seus respectivos valores máximos, para serem utilizados na fórmula apresentada na Equação 1.

As métricas de desempenhos avaliadas nos testes foram o consumo de energia despendida e o tempo de realização para envio de pacote de dados, consolidação do *handshake* e o consumo geral por arquitetura de segurança estudada. Para isso foram criados três grupos para estudo: pacote de dados, *handshake* e arquitetura de segurança. Cada grupo avaliou separadamente o servidor e o cliente, as versões de DTLS comprimido (*Lithe*) e padrão e, no grupo pacote de dados, também foi avaliada a versão sem DTLS (CoAP), para possibilitar o comparativo entre eles.

5.5 Cenários de Testes

O primeiro cenário utilizado nos testes foi composto de um servidor de DTLS, que fornece os serviços, e um cliente, que é o consumidor dos serviços na rede, ambos com as mesmas características, já que utilizam a mesma configuração de *Wismote*, conforme a Figura 16.

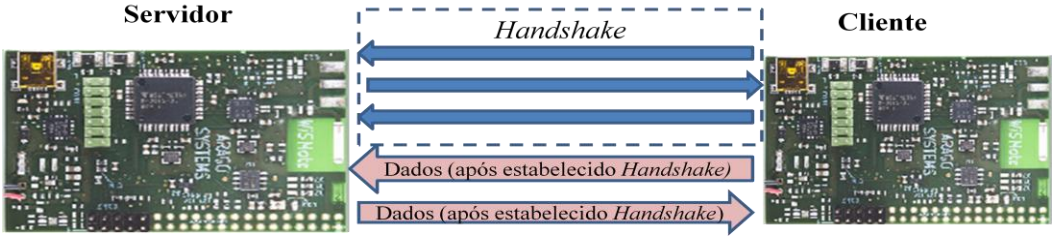

Figura 16 - Arquitetura de rede cliente/servidor

A mensuração do consumo entre cliente e servidor, durante o estabelecimento do *handshake*, o envio de dados, e o processo de envio de dados sem o *handshake*, foram realizados nesse cenário apresentado na Figura 16, e as informações dessas simulações serviram de parâmetros para o estudo dessa pesquisa. Neste cenário de simulação, a disposição dos clientes em relação ao servidor, foram dispostos com a mesma distância entre o nó cliente e o nó servidor.

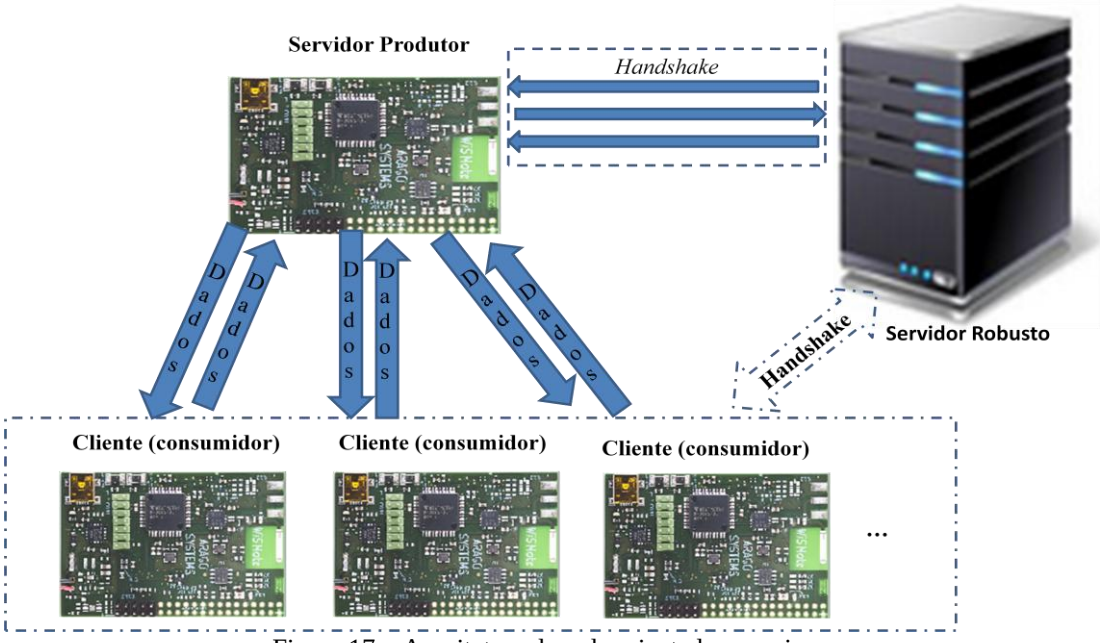

Figura 17 – Arquitetura de rede orientada a serviços

Também foram realizados testes com a arquitetura apresentada na Figura 17, uma Arquitetura Orientada a Serviços (SOA - *Service Oriented Architecture*). Nesta arquitetura tem-se 3 tipos de nós: servidor DTLS (produtor), cliente DTLS (consumidor) e um servidor robusto. Um servidor DTLS é um produtor de serviços e fornece serviços a rede para os consumidores (clientes DTLS) que o solicitarem. Os consumidores utilizam as informações recebidas dos produtores para alguma função específica de sua aplicação. O servidor robusto representa um servidor de autorização/delegação, com uma relação confiável com os produtores, armazenando as chaves destes e garantindo aos consumidores recursos seguros para que esses possam se comunicar com os produtores. Este servidor robusto não possui as limitações de energia dos servidores e clientes DTLS. O *handshake* é realizado entre o produtor e o

servidor robusto, que armazena as informações obtidas com o *handshake* para fazer este processo com os consumidores. Os consumidores então não fazem o *handshake* com o produtor, mas sim com o servidor robusto. Só a troca de dados (consumo do serviço) é feita diretamente com o produtor, possibilitando a redução de processamento e consumo de energia no produtor.

V - RESULTADOS E DISCUSSÕES

Com as simulações no Contiki, algumas discussões puderam ser levantadas conforme cada resultado avaliado nesta pesquisa. As próximas subseções apresentam os resultados gráficos e suas discussões pertinentes. As barras de erros tiveram como base o intervalo de confiança de 99%. E para cada gráfico de teste foi realizada a média de quinze simulações.

6.1 Análise de desempenho do handshake

Para limitar esse grupo de testes foram verificados o consumo de energia e tempo por *flight* do DTLS no servidor e no cliente, permitindo visualizar o impacto de cada processo do DTLS na rede avaliada, baseada na arquitetura da Figura 16. Esse grupo foi subdividido em rede com DTLS comprimido e DTLS padrão.

O gráfico da Figura 18 apresenta as comparações de tempo total de *handshake* do servidor e do cliente nos dois cenários possíveis de DTLS, comprimido e padrão. O tempo total do cliente ficou mais alto em ambos os grupos, se comparado ao servidor, isso porque o tempo do servidor é contado logo em seguida ao recebimento da solicitação (1ª mensagem) do cliente. As diferenças no tempo entre padrão e comprimido são imperceptíveis no cliente, já que seu comportamento se mantém estável em ambas configurações. No servidor com o DTLS comprimido o tempo total foi inferior ao do DTLS padrão, apesar de ser mínima essa diferença.

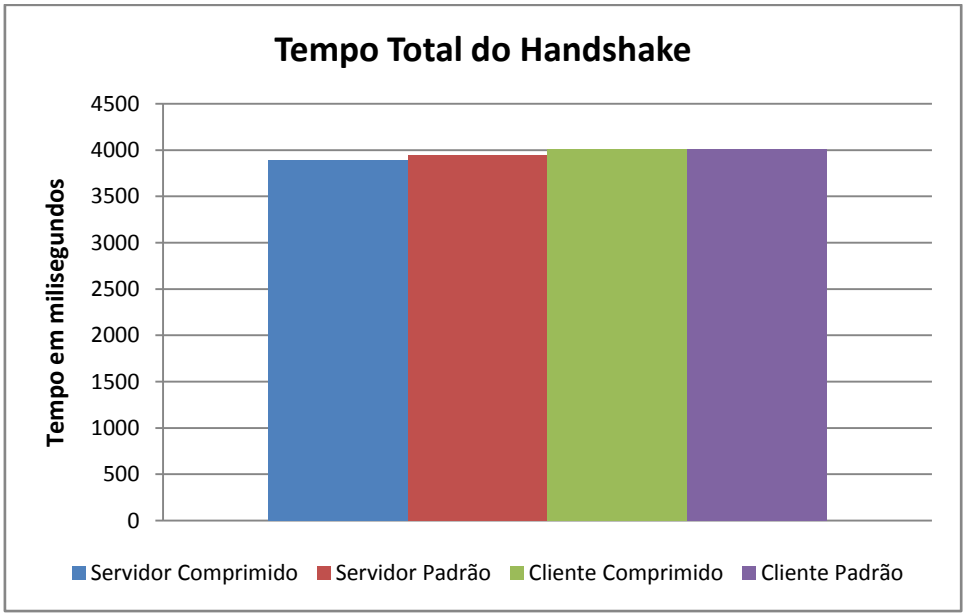

Figura 18 - Tempo Total do *handshake* (Cliente e Servidor)

Comparando servidor e cliente nos dois grupos avaliados, a Figura 19 apresenta o gráfico de média de tempo separadas por *fligths*. Por causa das definições e troca de chaves ocorridas no *handshake*,

ou seja, no estabelecimento da comunicação segura, é possível verificar que, tanto no cliente, quanto no servidor, os *fligths* 4, 5 e 6 são os que consomem maior tempo da rede.

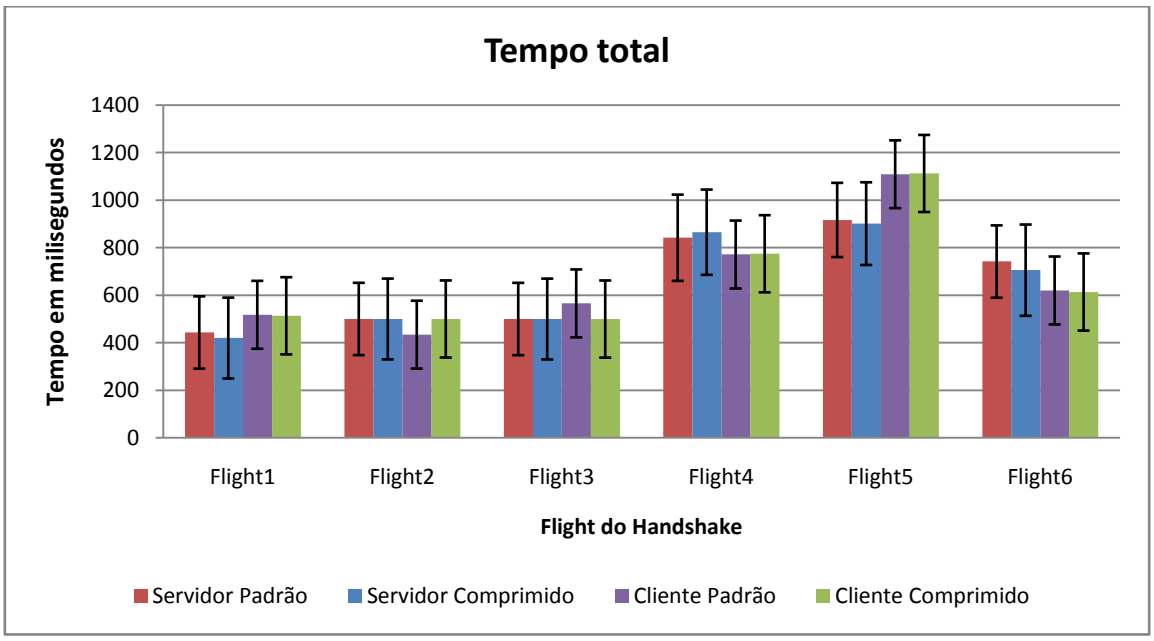

Figura 19 - Tempo Total do *handshake*

A Figura 20 apresenta o consumo total energético de uma rede cliente/servidor, confirmando a vantagem do uso do DTLS comprimido, já que seu consumo foi menor em relação ao uso do DTLS padrão, tanto no cliente quanto no servidor. Vale ressaltar que a compressão do cabeçalho no DTLS comprimido reduz o tamanho da mensagem e, por consequência, a complexidade do processamento diminui, o que leva a redução de consumo de energia.

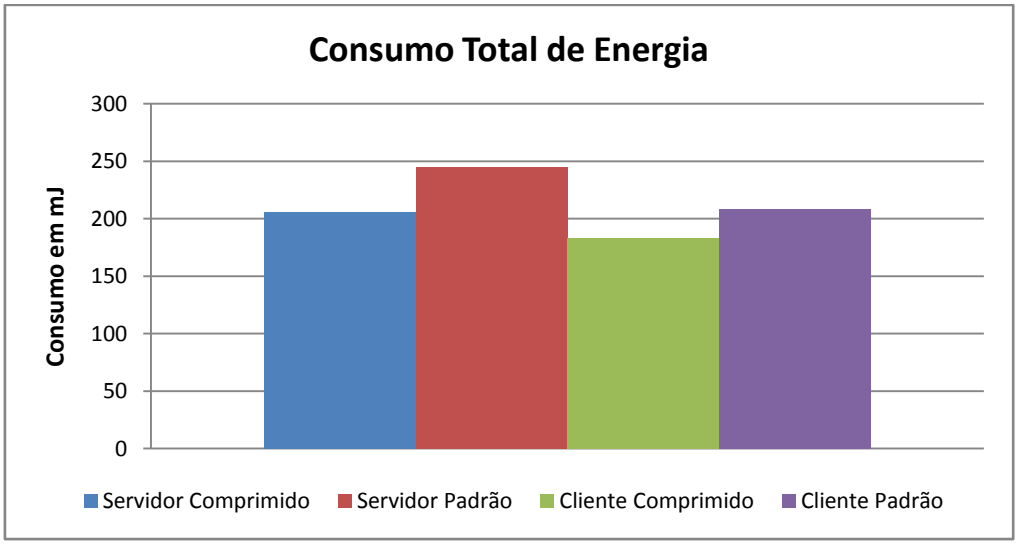

Figura 20 - Consumo Total de Energia do *handshake*

Na Figura 21, são comparados cliente e servidor nas redes estudadas por *flight* do *handshake*. No cliente com DTLS padrão, o pico de consumo ocorre no *fligth* 4, fase em que ele recebe os dados que

serão utilizados na definição das chaves, solicitação, ou não, de certificados. Já no servidor padrão, esse pico ocorre no *flight* 5, fase em que ele define a chave a ser utilizada na comunicação segura e recebe a confirmação do cliente nesse processo, para solicitar o fechamento do *handshake*, que ocorre no *flight* 6. No servidor com DTLS padrão o consumo de energia se mantém maior que o DTLS comprimido do início ao fim do *handshake.*

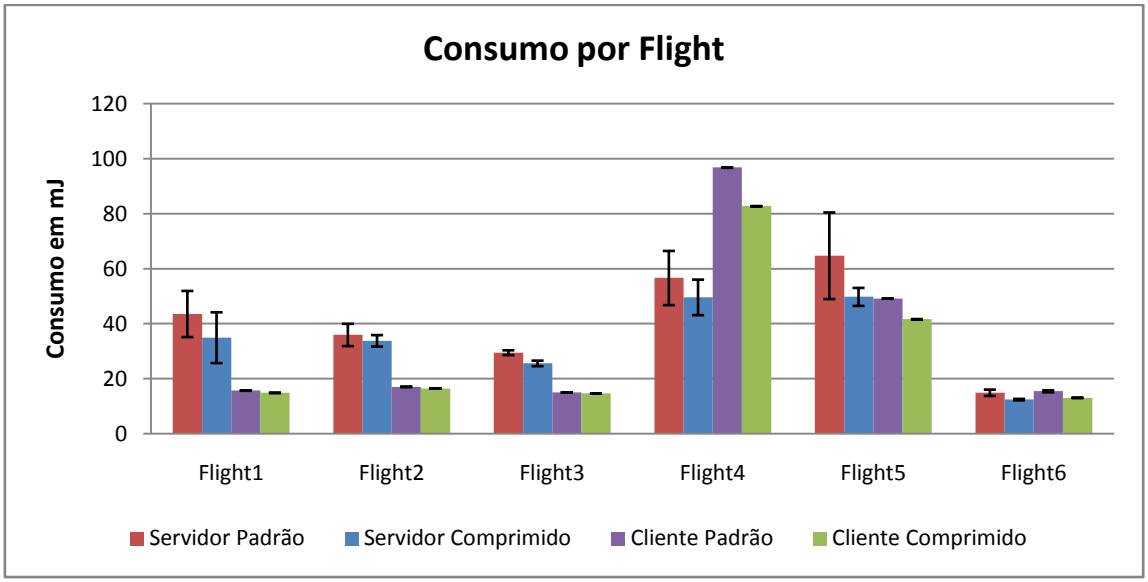

Figura 21 - Consumo de Energia por *flight*

Confirmando o trabalho de [RAZA; ET AL. 2013], o uso do DTLS comprimido tem maior vantagem que a versão sem compressão em relação ao consumo de energia da rede, mas não tanta diferença em relação ao tempo total, já que a quantidade de datagramas continua a mesma.

Pouco menos que a metade do tempo total são consumidos nos *fligths* 1, 2 e 3, um ponto interessante, já que esses *fligths* apenas iniciam o processo do *handshake* por meio de mensagens entre cliente e servidor; já o restante do tempo estão nos *flights* 4, 5 e 6, responsáveis pelas definições de chaves, compressão e certificados (que nessa pesquisa não foram utilizados) da comunicação da rede segura. Quanto ao consumo, os *flights* 4, 5 e 6 consomem o dobro dos *fligths* 1, 2 e 3, por causa de seus processos.

Esses resultados do *handshake* levam a questionar se a proposta de [HARTK; BERGMANN, 2012] na utilização do *handshake* reduzido seria realmente interessante para a economia de energia da rede, já que são retirados os *flights* 2 e 3, realocando os serviços destes no *flight* 1. Para verificar os resultados desta configuração os gráficos das Figuras 22 e 23 a seguir apresentam os resultados desta simulação.

Com a Figura 22 pode-se visualizar o resultado positivo ao se utilizar o *handshake reduzido*, garantindo uma redução aproximada de consumo de energia: no servidor com DTLS padrão a redução é

de 27% e no com comprimido é de 29%; no cliente com DTLS padrão é de 15% e no comprimido é de 17% de redução. Como esperado, ao se reduzir a quantidade de datagramas no *handshake*, o consumo de energia total fica menor.

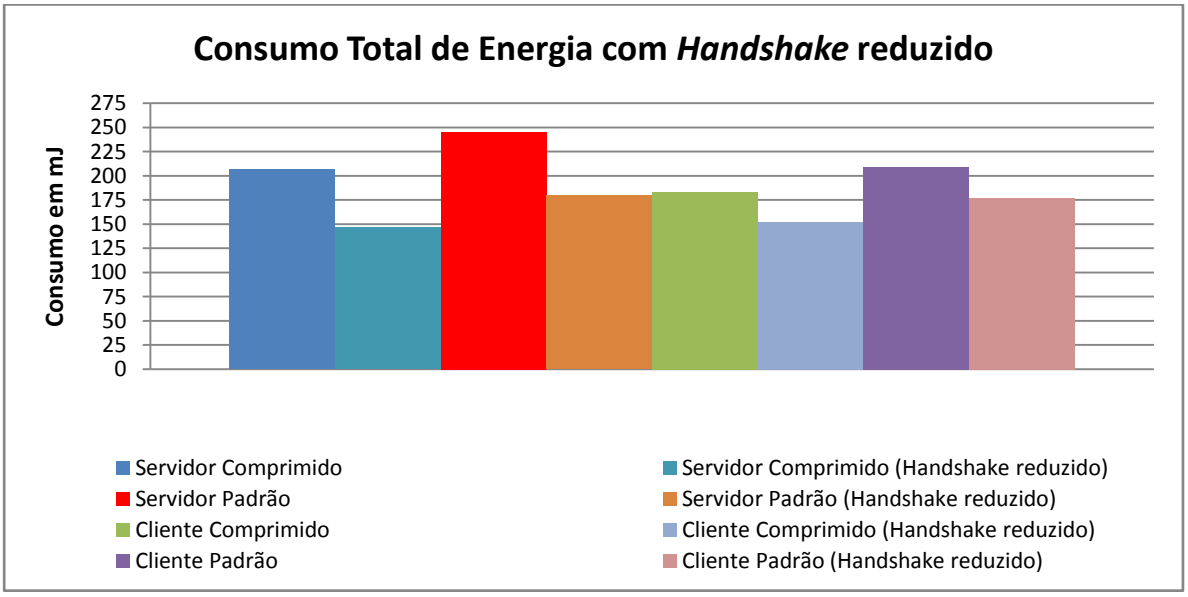

Figura 22 - Consumo Total de Energia *handshake* reduzido

Entretanto, o tempo total também foi reduzido aproximadamente 25% no servidor e 24% no cliente, conforme apresenta a Figura 23, deixando clara a vantagem de se utilizar o *handshake* reduzido, sem perder a segurança proposta pelo protocolo DTLS.

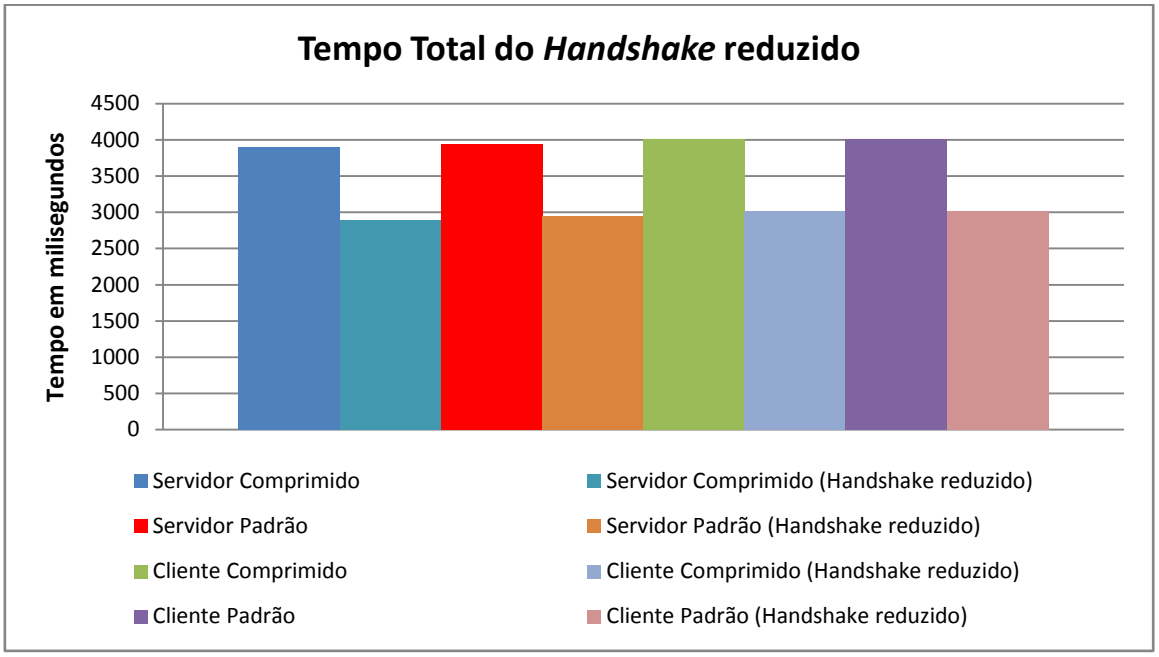

Figura 23 - Tempo Total *handshake* reduzido

6.2 Análise de desempenho de um pacote de dados

Nesse grupo de testes, baseada na arquitetura cliente/servidor da Figura 16, os gráficos apresentam o consumo de energia na rede com DTLS padrão, DTLS comprimido e sem DTLS (utilizando somente o

protocolo CoAP para a comunicação), para uma requisição do cliente ao servidor e envio de um pacote de dados pelo servidor.

O cálculo de consumo nas redes que utilizaram o protocolo DTLS (padrão ou comprimido), foi iniciado a partir de estabelecido o *handshake*, portanto, o consumo gerado para o estabelecimento deste processo não é considerado.

Na Figura 24 pode-se verificar o consumo total de energia em *mJ* de cada rede avaliada (DTLS padrão, DTLS comprimido e sem DTLS). Interessante notar que a não utilização de segurança consome menos da metade do que quando se utiliza o protocolo DTLS.

Vale ressaltar que a necessidade de proteger os dados na transmissão torna permissível esse consumo mais alto, dependendo da aplicação da rede. Caso a aplicação não dependa da segurança, a economia de energia e tempo é interessante a se considerar.

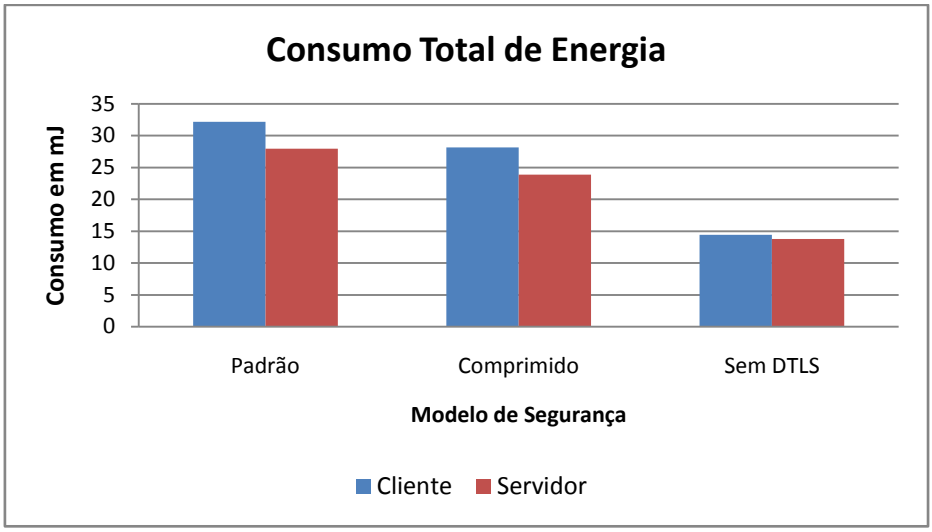

Figura 24 - Consumo Total no envio de um pacote de dados

A Figura 25 apresenta o total de tempo de processamento utilizado no servidor para o recebimento e realização da requisição do cliente de um pacote de dados. O tempo total também é muito diferente ao se utilizar o DTLS, que foi de aproximadamente 51% a mais na rede que usa DTLS padrão e 52% na com DTLS comprimido, isso em relação a rede sem DTLS. Vale lembrar que essa diferença alta entre o uso de segurança ocorre por conta dos processos necessários para criptografar e descriptografar a mensagem enviada pelo cliente, antes de responder sua requisição, e já na rede sem segurança o texto é puro, bastando ao servidor responder a requisição do cliente. Assim como nos gráficos anteriores de tempo total, a rede com DTLS comprimida e padrão tem uma pequena diferença de tempo entre si.

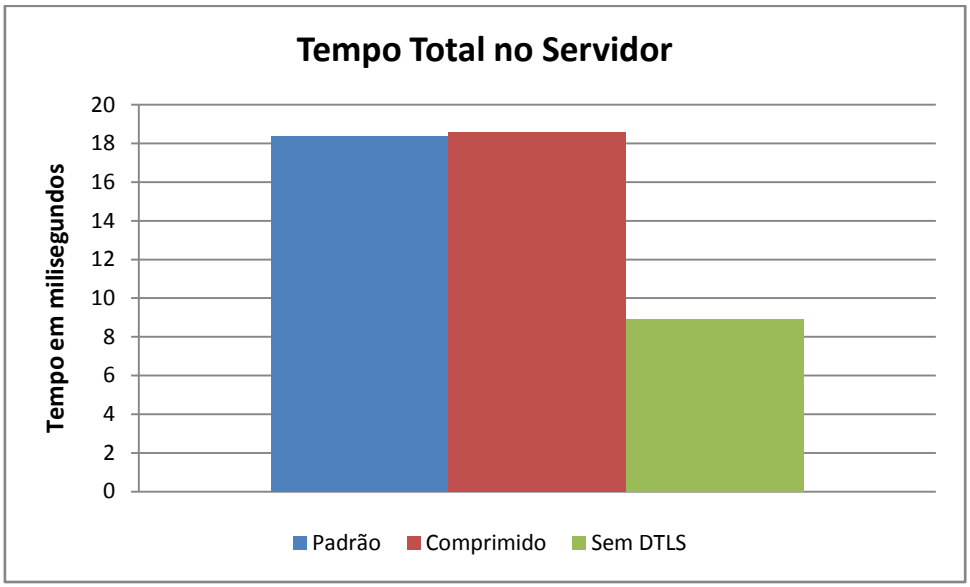

Figura 25 - Tempo total no servidor

O tempo para envio e recebimento de um pacote de dados (requisição do cliente para o servidor) é o RTT (*Round Trip Time*). O tempo RTT para um cliente sem DTLS teve uma grande diferença em relação a um cliente com DTLS. O RTT aumentou aproximadamente 73% no uso de DTLS padrão e DTLS comprimido, como apresenta a Figura 26, conforme já explicado, motivado pela descompressão da mensagem de requisição e compressão da mensagem resposta do servidor ao cliente.

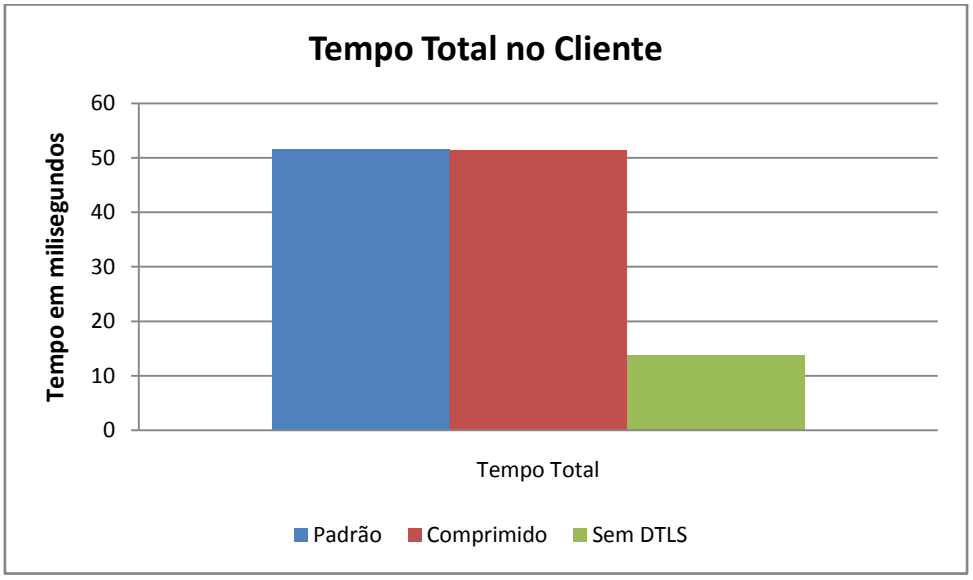

Figura 26 - Tempo total (RTT) no cliente

6.3 Análise de desempenho por Arquitetura de Segurança

Tendo como base os resultados das simulações das subseções anteriores, foi possível elaborar os gráficos por arquitetura de segurança.

Esse grupo de estudo foi subdividido em arquitetura cliente/servidor com DTLS padrão, arquitetura cliente/servidor com DTLS comprimido, SOA padrão (arquitetura orientada a serviços com DTLS padrão) e SOA comprimido (arquitetura orientada a serviços com DTLS comprimido). Nas arquiteturas SOA, o cenário de simulação utilizado é o apresentado na Figura 17, considerando que os clientes já realizaram o *handshake* com o servidor robusto e já estando na rede, realiza a troca de dados com o servidor DTLS. Já no grupo de arquitetura cliente/servidor, cada cliente realiza o *handshake* com o servidor DTLS, baseado na Figura 16, com vários clientes na rede, como serão apresentados nos gráficos a seguir.

A Figura 27 apresenta o tempo total de uma requisição do cliente ao servidor por arquitetura em uma rede com até 10 clientes. Este tempo inclui o RTT (tempo de envio e recebimento de um pacote de dados), somado ao tempo necessário para estabelecimento do *handshake*. Conforme aumenta a quantidade de clientes participantes nas redes cliente/servidor com DTLS padrão e cliente/servidor com DTLS comprimido, o tempo médio necessário chega aumentar mais de 81% em relação às redes SOA, deixando clara a vantagem dessas arquiteturas em relação ao tempo.

Isso acontece porque nas redes SOA, o *handshake* no servidor produtor só é realizado uma vez (com o servidor robusto) e na rede cliente/servidor, é necessário realizar o *handshake* com cada um dos clientes da rede. Importante lembrar que o tempo do *handshake* é maior que o tempo de envio de um pacote de dados. Não há diferenças significativas no DTLS padrão e comprimido em relação ao tempo.

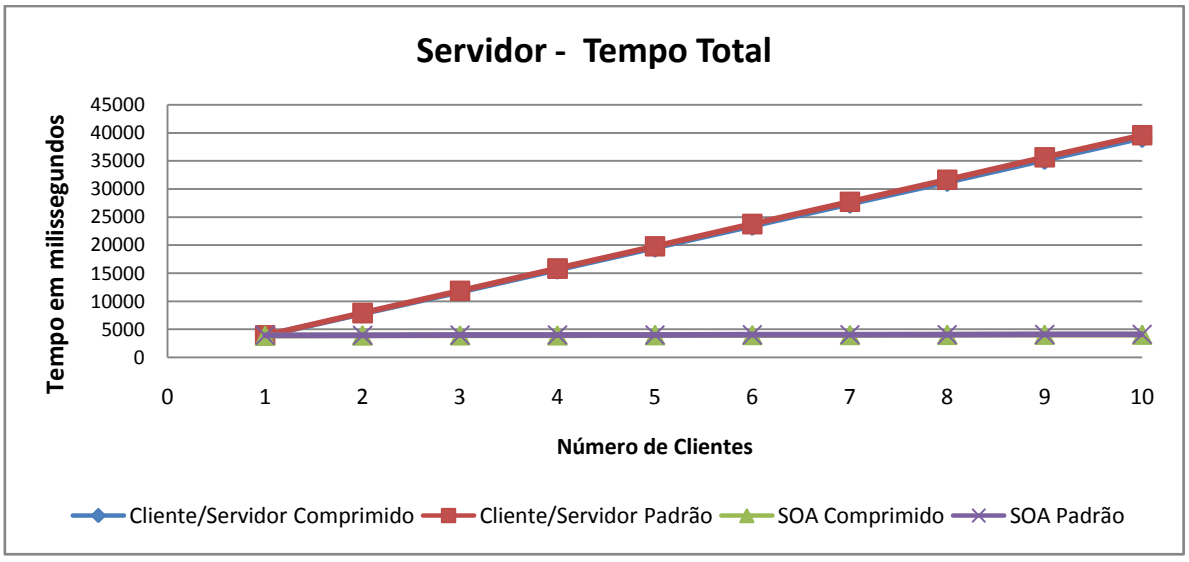

Figura 27 - Tempo no servidor

A Figura 28 apresenta o consumo de energia do servidor por arquitetura de segurança. Na rede cliente/servidor padrão é de aproximadamente 78% maior que a SOA padrão e a cliente/servidor comprimido é 68% maior que SOA comprimido. Isto acontece porque na arquitetura SOA o processo de

handshake é feito apenas uma vez pelo servidor (com o servidor robusto), deixando para o servidor robusto a tarefa de fazer o *handshake* com os demais clientes.

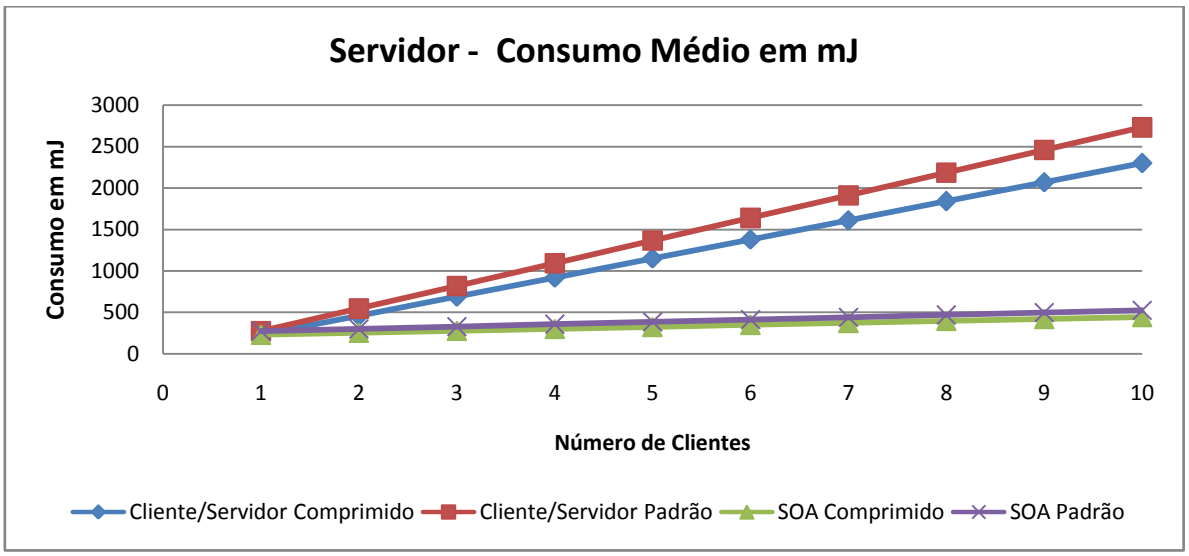

Figura 28 - Consumo em mJ no servidor

Um cliente DTLS não altera seu comportamento seja na arquitetura cliente/servidor ou SOA, já que ele continua realizando o *handshake* normalmente, seja direto com o servidor (no modelo cliente/servidor), seja no servidor robusto (SOA). A Figura 29 apresenta o tempo consumido por cada requisição do cliente ao servidor (cada requisição equivalendo um pacote). Com o aumento do número de requisições, o tempo também aumenta, sem que haja diferenças significativas entre as arquiteturas. A principal vantagem da arquitetura SOA é desonerar o produtor de serviços, e não o consumidor, que continua com o mesmo consumo de energia e tempo, independente da arquitetura utilizada.

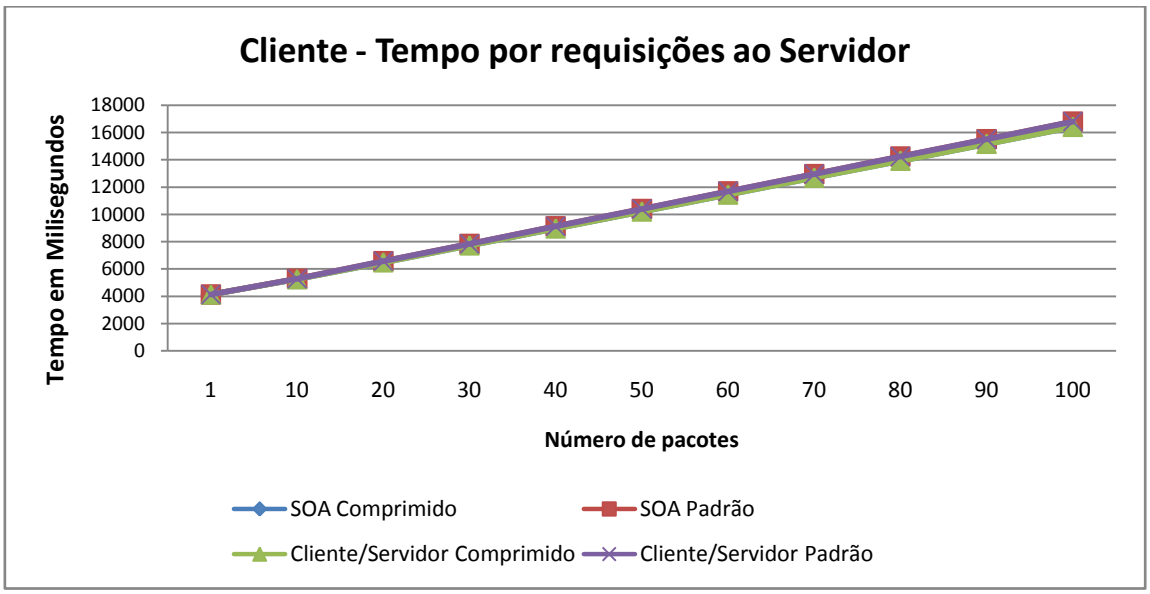

Figura 29 - Tempo por requisição do cliente ao servidor

O comportamento é similar em redes com mais clientes, como por exemplo até 100 clientes, tanto em consumo, quanto no tempo. As redes SOA obtiveram vantagem com relação a essas análises em relação as outras. Devido repetição no comportamento, não foi necessária a colocação de novos gráficos para apresentar essas informações.

Considerando o *handshake* reduzido, as Figuras 30 e 31 mostram quanto seria economizado em energia nas redes SOA, já que tiveram melhores resultados, se comparadas às redes cliente/servidor comprimida ou padrão, por isso essas redes não serão consideradas nesses gráficos.

A Figura 30 apresenta uma economia interessante que ocorre no servidor ao se utilizar o *handshake* reduzido. Ao se utilizar o *handshake* reduzido o consumo de energia é aproximadamente 25% menor nas redes SOA, tanto no comprimido, como no padrão.

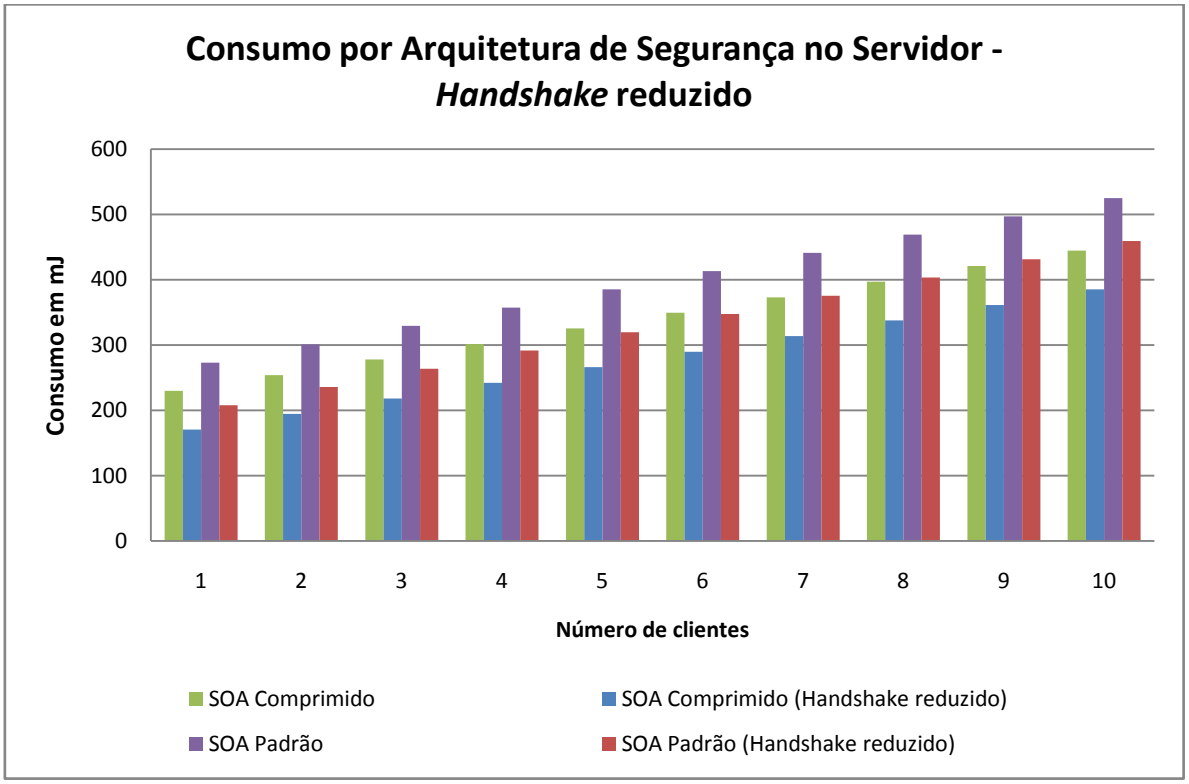

Figura 30 - Consumo por arquitetura com *handshake* reduzido

O tempo total também sofre uma considerável redução ao se utilizar o *handshake* reduzido, como a Figura 31 mostra. Como os *flights* 1 e 2 tiveram suas funções realocadas nos *flights* 3 e 4, o tempo necessário para realização do *handshake* é menor no modelo reduzido, por isso que na SOA comprimido a redução foi de aproximadamente 17% e na SOA padrão, 16% de redução.

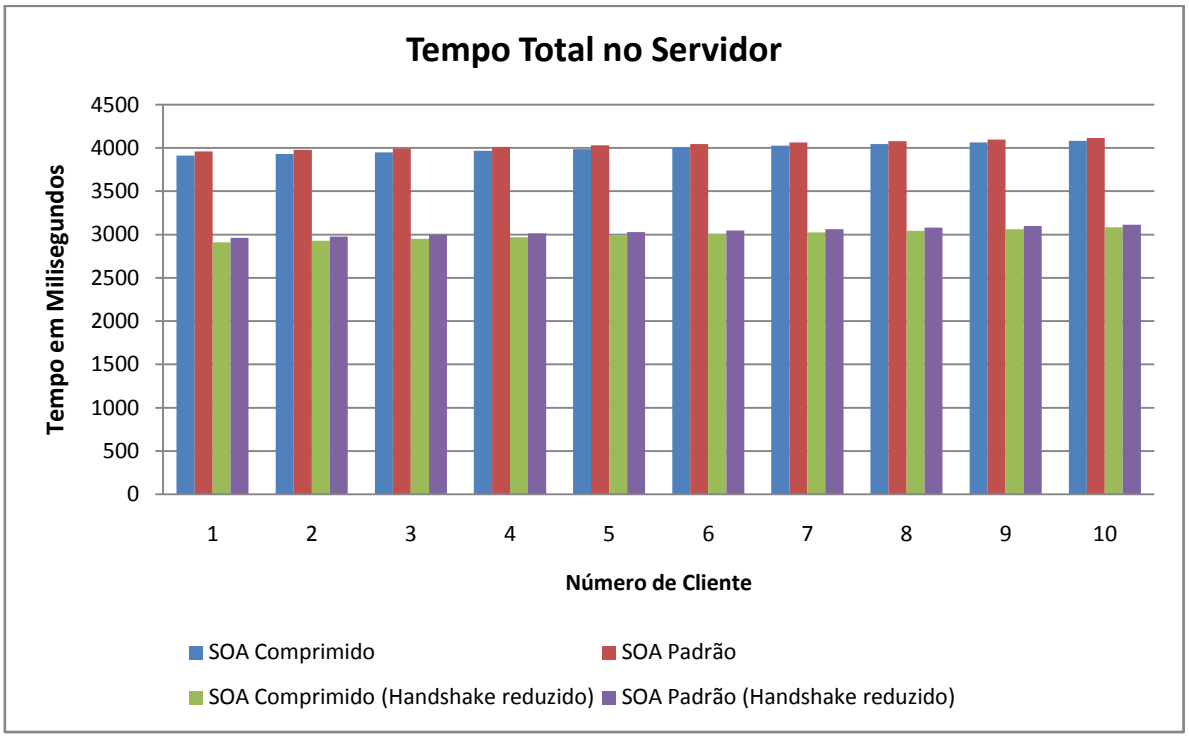

Figura 31 - Tempo no servidor com *handshake* reduzido por clientes

Com base nos resultados dos últimos gráficos apresentados, é realmente vantajoso, tanto em relação ao tempo, quanto em relação ao consumo de energia, a utilização do *handshake* reduzido, permitindo um melhor desempenho de consumo, sem perder as vantagens de segurança do protocolo DTLS.

6.4 Análise das Arquiteturas de Segurança

O DTLS na arquitetura cliente/servidor pode ser utilizado nas abordagens 1-1, 1-s, c-1 e c-s (que tem todas as desvantagens de 1-s e c-1) [ROMAN; ET AL, 2011]. Para comparar essas arquiteturas de rede cliente/servidor e a arquitetura orientada a serviços, foram utilizadas também outras métricas estudadas por [HE, 2012], descritas no capítulo 3. Ao se analisar essas redes pelas métricas de segurança, temos:

- Na revogação de nós: apesar do DTLS não ter um mecanismo de revogação de nós, na arquitetura SOA seria mais fácil a revogação de produtores DTLS, por exemplo, por ter o servidor centralizado.
- Quanto ao sigilo para frente e para trás: na rede SOA, os clientes recebem uma chave de acesso por servidor produtor e, por isso, deve-se haver um plano para alterar frequentemente essas chaves, para garantir a segurança na rede; já na rede cliente/servidor, na abordagem c-1, o servidor têm as chaves de cada cliente, todas distintas entre si. Em princípio, esta métrica independe da arquitetura utilizada e depende mais de uma política de atualização de chaves.

- Resistência ao conluio: caso um cliente (na rede SOA) tenha sua chave capturada (chave de acesso ao servidor), o servidor robusto terá que revogar aquele nó, mas também terá de refazer a chave de acesso ao servidor (*handshake* com o servidor) e reenviar essa chave a todos os clientes da rede (*handshake* com os clientes); no caso de uma captura do cliente (na rede cliente/servidor), o servidor só precisará revogar aquele nó, se for 1-1 ou c-1. No caso da captura do servidor produtor (na rede SOA), toda a rede estará comprometida e o servidor robusto deverá revogar todas as chaves. No servidor da rede cliente/servidor, uma captura prejudica toda a rede, novamente dependendo se for 1-1, c-1 e 1-s. Ao utilizar a abordagem 1-1 na arquitetura cliente/servidor, uma chave é distribuída para um grupo de clientes e, caso algum deles seja capturado, o servidor teria um grande consumo de energia para refazer e repassar uma nova chave ao restante da rede. Na abordagem 1-s, cada servidor possui uma chave própria, e caso o cliente precise acessar, terá de armazenar a chave de cada servidor que necessite; se esse cliente for capturado, todos os servidores ao qual ele teve acesso precisarão ter suas chaves revogadas. Na abordagem c-1, o servidor tem uma chave diferente para todos os clientes que devem acessá-lo, e sua chave seria distribuída para esses clientes. Caso um desses clientes fosse capturado, o servidor teria de revogar o cliente capturado, sem maiores prejuízos na rede.
- Resiliência: as redes SOA terão melhor garantia de se manter operantes em segurança, caso o servidor produtor seja capturado, que as redes cliente/servidor, já que esta é fatalmente prejudicada se o servidor for capturado, dependendo se for 1-1, c-1 e 1-s. Entretanto, se o servidor robusto for capturado, toda a rede estará prejudicada.

Também pode-se analisar essas redes pelas métricas de flexibilidade:

- Mobilidade: as redes cliente/servidor têm maior mobilidade que as redes SOA, pois não estão limitados fisicamente ao servidor robusto (que é fixo), já que pelo menos uma vez tanto o cliente, quanto o servidor produtor, devem realizar o *handshake* com ele e, dependendo da aplicação fim isso pode ser uma limitação importante. A mobilidade pode gerar desconexões e necessidade de realização do *handshake* novamente, algo muito prejudicial na arquitetura cliente/servidor.
- Escalabilidade: com o uso do protocolo DTLS, essa escalabilidade é possível em ambas configurações de arquitetura de rede, mas a rede SOA possui maior vantagem em redes com um número grande de clientes, pois consegue gerenciar suas chaves melhor do que as redes cliente/servidor, devido a limitação da configuração física do servidor. O servidor robusto não tem restrições de número de clientes (consumidores) e servidores (produtores), outra vantagem da arquitetura SOA sob a arquitetura cliente/servidor.

- Conectividade de chave: na rede SOA a probabilidade de estabelecimento de chaves após a recodificação entre nós, é maior por conta de ser o servidor robusto a gerenciar essa funcionalidade importante, o que é bem mais frágil na rede cliente/servidor, por conta da restrição da arquitetura física do servidor (em relação a memória e processamento).

As métricas de eficiência também podem ser utilizadas para avaliar as redes estudadas:

- Memória: Na abordagem 1-1 a exigência de memória é menor, pois as chaves são compartilhadas para grupos (servidores ou produtores e clientes ou consumidores), muito positivo em redes com grande quantidade de nós. Na abordagem 1-s, a maior exigência de memória recai sobre o cliente. Na abordagem c-1, na arquitetura cliente/servidor a maior exigência de memória recai sobre o servidor. Já na arquitetura SOA, esses problemas são minimizados, pois o servidor robusto possui maior recursos em armazenamento, comportando as chaves do servidor produtor e dos clientes, e conforme necessário o acréscimo de muito mais clientes, o aumento de memória é muito menos custoso que no servidor da rede cliente/servidor.
- *Overhead* de transmissão: como nas redes SOA o servidor robusto é o responsável pelas autorizações/delegações, é possível comportar um número muito maior de mensagens para a realização dos *handshake* do servidor produtor e dos clientes, reposição de nós e despejo de nós, que o servidor da rede cliente/servidor. Em comparação a arquitetura SOA a abordagem 1-1 na cliente/servidor, ocorre menos trocas de mensagens para o processo de geração de chave, pois esta é compartilhada por grupo, o que já ocorre na arquitetura SOA por ser centralizada, administrando melhor estas chaves. Na abordagem 1-s, o cliente tem maior *overhead* e na abordagem c-1 o servidor tem maior *overhead*.
- Energia: Na arquitetura SOA, o consumo de energia foi bem menor no servidor produtor que na arquitetura cliente/servidor, entretanto a rede como um todo possui um consumo alto, mas a grande vantagem está no servidor robusto que não tem limitação de energia. Outra vantagem é na utilização desta arquitetura quando a aplicação fim utilize-se de muitos clientes e/ou muitos produtores.

Apesar das vantagens da arquitetura SOA sob a arquitetura cliente/servidor, deve ser considerado o custo adicional do servidor robusto à rede; entretanto, conforme o aumento do número de clientes e servidores produtores na rede, esse custo adicional torna-se permissível em relação as vantagens que são proporcionadas nesse modelo de arquitetura.

VI - CONSIDERAÇÕES FINAIS

A Internet das Coisas possibilita a interconexão de objetos inteligentes com a rede tradicional. Suas aplicações podem ser desde sensoriamento e monitoramento, como na logística, saúde, controle de identificação, militar e entre outras mais diversas áreas. Ao se implantar segurança na Internet das Coisas, deve-se considerar suas limitações de recursos em memória, processamento e consumo energético, levando a comunidade científica a adaptar e desenvolver protocolos para a IoT: CoAP, DTLS, 6LoWPAN e IEEE 802.15.4, que formam a pilha de protocolos utilizada neste estudo.

O *Datagram Transport Layer Security* (DTLS) é um protocolo de segurança que automatiza o gerenciamento e a distribuição de chaves, que não consome recursos da rede de IoT como seus similares presentes na rede tradicional, além de automatizar a autenticação e a encriptação dos dados, utilizado para proteção das mensagens trafegadas via UDP para aplicações cliente/servidor. O *handshake* é um dos protocolos do DTLS, organizado em *flights*, usado para negociar chaves de segurança, conjuntos de codificação e métodos de compressão. Alguns autores [RAZA; ET AL. 2013] apresentaram uma adaptação e integração de DTLS e CoAP para a Internet das Coisas, chamado *Lithe*, que é uma compressão do cabeçalho DTLS, e nesse trabalho recebeu a notação de DTLS comprimido, que reduz significativamente o consumo de energia.

Por meio de simulações no Sistema Operacional Contiki, foi avaliado o consumo de energia ao se utilizar o protocolo DTLS numa arquitetura cliente/servidor e numa arquitetura orientada a serviços (SOA), tanto com a versão padrão do protocolo, quanto na versão comprimido. Na arquitetura cliente/servidor, a mensuração do consumo de energia e tempo ocorre entre cliente e servidor, durante o estabelecimento do *handshake* e o envio de pacote de dados, já na arquitetura SOA o *handshake* é realizado entre o servidor DTLS (que nesse contexto é o fornecedor) e um servidor robusto (que representa um servidor de autorização ou delegação), e vários clientes (que são os consumidores), que realizam o *handshake* com o servidor robusto e se comunicam de forma segura com o servidor fornecedor.

A diferença do tempo total do *handshake* no servidor nas duas formas de DTLS é muito pequena, em média 1% menor no formato comprimido em relação ao formato padrão. Ao se utilizar o *handshake* reduzido a redução é importante nas duas formas do DTLS, sendo aproximadamente 25% menor que o *handshake* completo, isso por que foi retirado dois *fligths* (ou seja, menos *datagramas*) do processo, sem perder os recursos de segurança disponíveis no protocolo. No cliente essa diferença do tempo total são pequenas em relação ao padrão e comprimido.

Em relação ao consumo total de energia, com o *handshake* completo no servidor, o formato comprimido é em média 16% menor em relação ao formato padrão, e no cliente o formato comprimido é 12% menor que o formato padrão. Já em comparação ao *handshake* reduzido, no servidor a redução é de 26% em relação ao DTLS padrão e de 28% no comprimido; já no cliente a redução é de 15% em relação ao DTLS padrão e de 16% no comprimido.

Graças a compressão do cabeçalho no DTLS comprimido, o tamanho da mensagem fica reduzido e, por isso, a exigência de processamento diminui, levando a redução de consumo de energia na rede, o que torna esse formato recomendado para uso nas redes IoT. E com as informações do tempo e do consumo de energia obtidas, mostra que o *handshake* reduzido obteve excelentes resultados, o que leva a ser um bom modelo para ser utilizado na implantação da rede.

Apesar dos testes realizados por um pacote de dados mostrar que incluir segurança na transmissão aumenta consideravelmente o tempo e o consumo de energia na rede, a segurança não é um fator que se pode ser desconsiderada. Entretanto, com base nesses resultados, o administrador da rede pode avaliar melhor o impacto do uso desse protocolo de segurança na rede.

A arquitetura SOA tem vantagens no consumo de energia e tempo em relação a arquitetura cliente/servidor, pois o *handshake* dos consumidores são realizados no servidor robusto. Assim, garante ao servidor produtor um maior tempo de vida na rede, já que só realiza um *handshake* com o servidor robusto, podendo responder as requisições dos consumidores num canal seguro sem consumir muita energia. O consumo de energia na rede cliente/servidor utilizando DTLS padrão é de aproximadamente 77% maior que a SOA padrão, e a cliente/servidor com o DTLS comprimido é 68% maior que SOA comprimido. O tempo total chega aumentar em média mais de 81% em relação as redes SOA. Já no cliente não ocorre essas alterações de consumo, pois o comportamento na arquitetura cliente/servidor e na arquitetura SOA, é muito similar.

Esses resultados colocaram a arquitetura SOA em maior vantagem de ser utilizada numa rede IoT, entretanto, também é possível considerar um aperfeiçoamento nessa rede ao se utilizar o *handshake* reduzido. Utilizando o *handshake* reduzido o consumo de energia é aproximadamente 25% menor na arquitetura SOA. Já em relação ao tempo total, as redes que utilizaram o *handshake* reduzido tiveram aproximadamente 17% de redução do tempo, confirmando as duas configurações da arquitetura SOA, com o *handshake* reduzido, como vantajoso para implantação numa rede IoT. A desvantagem de se usar SOA é o custo do servidor robusto e a sua centralização. Ataques direcionados ao servidor robusto podem trazer grandes danos à rede, como comprometimento das chaves de criptografia.

Algumas propostas para trabalhos futuros interessantes incluem verificar o impacto do uso de certificados com o DTLS na arquitetura de segurança e a implementação da arquitetura SOA, com DTLS com o *handshake* reduzido, em motes reais para uma comparação com os resultados da simulação.

REFERÊNCIAS

[ATZORI, 2010] ATZORI, Luigi; IERA, Antonio; MORABITO, Giacomo. The Internet of Things: A survey. Elsevier: Computer Networks. Volume 54, Issue 15, 28 October 2010, Pages 2787–2805.

[BALA, 2012] BALA, Suman; SHARMA, Gaurav; VERMA, Anil K. A Survey and Taxonomy of Symmetric Key Management Schemes for Wireless Sensor Networks. CUBE '12 Proceedings of the CUBE International Information Technology Conference. Pages 585-592. India: 2012.

[BAUER, 2010] BAUER, Veronika Maria. Routing in Wireless Sensor Networks: An Experimental Evaluation of RPL. Master's Thesis. École Polytechnique. 2010.

[BERGMANN; ET AL, 2012] BERGMANN, Olaf; GERDES, Stefanie; SCHAFER, Silke; JUNGE, Florian; BORMANN, Carsten. Secure Bootstrapping of Nodes in a CoAP Network. WCNC 2012 Workshop on Internet of Things Enabling Technologies, Embracing Machine-to-Machine Communications and Beyond. 2012.

[CAMPISTA; DUARTE, 2002] CAMPISTA, Miguel Elias Mitre; DUARTE, Otto Carlos Muniz Bandeira. Segurança em Redes de Sensores. Rio de Janeiro. 2002.

[CHEN, 2012] CHEN, Dong; CHANG, Guiran; Sun, Dawei; JIA, Jie; WANG, Xingwei. Lightweight key management scheme to enhance the security of internet of things. Int. J. Wireless and Mobile Computing, Vol. 5, No. 2, 2012.

[DING, 2010] DING, Yan; ET AL. A Security Differential Game Model for Sensor Networks in Internet of Things – New security and privacy challenges. Nova York: 2010.

[DUNKELS, 2004] DUNKELS, Adam; GRONVALL, Bjorn; VOIGT, Thiemo. Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. Swedish Institute of Computer Science: 2004.

[DUNKELS, ERIKSSON, 2011] DUNKELS, Adam; ERIKSSON, Joakim; FINNE, Niclas; TSIFTES, Nicolas. Powertrace: Network-level Power Profiling for Low-power Wireless Networks. SICS Report. 2011.

[DUNKELS; OSTERLIND, 2008] DUNKELS, Adam; OSTERLIND, Fredrik. Contiki Programming Course: Hands-On Session Notes. 2008.

[FERNANDES, 2006] FERNANDES, Natalia C.; MOREIRA, Marcelo D. D.; VELLOSO, Pedro B.; COSTA, Luís Henrique M. K.; DUARTE, Otto Carlos M. B. Ataques e Mecanismos de Segurança em Redes Ad Hoc. Universidade Federal do Rio de Janeiro:2006.

[GARCIA-MORCHON, 2012] GARCIA-MORCHON, O.; Keoh, S.; KUMAR, S.; HUMMEN, R.; AACHEN, RWTH; STRUIK, R. Security Considerations in the IP-based Internet of Things. Draft-garcia-core-security-04. September 27, 2012.

[GRANJAL, ET AL., 2015] GRANJAL, Jorge; MONTEIRO, Edmundo; SILVA, Jorge Sá. Security in the integration of low-power Wireless Sensor Networks with the Internet: A Survey. Ad Hoc Networks 24 (2015) 264-287. Elsevier: 2015.

[HARTKE, BERGMANN, 2012] HARTKE, K; BERGMANN O. Datagram Transport Layer Security in Constrained Environments. Draft-hartke-core-codtls-01. Internet Engineering Task Force (IETF). 2012.

[HE, 2012] HE, Xiaobing; NIEDERMEIER, Michael; MEER, Hermann de. Dynamic key management in wireless sensor networks: A survey. Journal of Network and Computer Applications 36 (2013) 611–622. Alemanha: 2012.

[HEER; ET AL, 2011] HEER, Tobias; GARCIA-MORCHON, Oscar; HUMMEN, René; KEOH, Sye Loong; KUMAR, Sandeep S.; WEHRLE, Klaus. Security Challenges in the IP-based Internet of Things. Springer Science+Business Media, LLC. 2011. Wireless Pers Commun (2011) 61:527–542.

[HENNEBERT; SANTOS, 2014] HENNEBERT, Christine; SANTOS, Jessye dos. Security Protocols and Privacy Issues into 6LoWpAN Stack: A syntesis. IEEE Internet of Things Journal, VOL. 1, NO. 5, October 2014.

[HONG; ET AL. 2012] HONG, Yu; JINGSHA, He; TING, Zhang; PENG, Xiao; YUQIANG, Zhang. Enabling end-to-end secure communication between wireless sensor networks and the Internet. World Wide Web. Springer Science+Business Media New York 2012.

[HUMMEN; RAZA; ET AL, 2014] HUMMEN, René; RAZA, Shahid; VOIGT, Thiemo; WERLE, Klaus. Delegation-based Authentication and Authorization for the IP-based Internet of Things. IEEE 2014.

[JESUS; KLEINSCHMIDT, 2014] JESUS, Daniele Freitas de; KLEINSCHMIDT, João Henrique. Estudo de arquiteturas de segurança para internet das coisas. II Simpósio da Pós - graduação da UFABC. Santo André: 25 a 27 de Novembro de 2014. Pôster de resumo estendido.

[JESUS; KLEINSCHMIDT, 2015] JESUS, Daniele Freitas de; KLEINSCHMIDT, João Henrique. Estudo do Consumo de Energia do Protocolo DTLS para Internet das Coisas. VI Computer on the Beach 2015. Florianópolis - Santa Catarina: 22 de março de 2015. Pôster de resumo estendido.

[JUNG; ET AL., 2013] JUNG, T. X.-Y. Li, and Z. Wan, "Privacy Preserving Cloud Data Access With Multi-Authorities." Infocom. IEEE, 2013.

[KEOH; ET AL, 2014] KEOH, Sye Loong; KUMAR, Sandeep S.; TSCHOFENIG, Hannes. Securing the Internet of Things: A Standardization Perspective. IEEE Internet of Things Journal, Vol. 1, NO. 3, June 2014.

[KOPETZ, 2011] KOPETZ , H. Internet of things. Real-Time Systems: Design Principles for Distributed Embedded Applications, Real-Time Systems Series, Springer Science+Business Media, LLC 2011. Wien: 2011.

[KOTHMAYR; ET AL, 2013] KOTHMAYR, Thomas; SCHMITT, Corinna; Hub, Wen; BRÜNIG, Michael; CARLE, Georg. DTLS based security and two-way authentication for the Internet of Things. Ad Hoc Networks. Elsevier. 2013.

[KOVATSCH, 2011] KOVATSCH, Matthias; DUQUENNOY, Simon; DUNKELS, Adam. A Low-Power CoAP for Contiki. Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on.

[LOUREIRO, 2003] LOUREIRO, Antonio A.F.; ET Al. Redes de Sensores Sem Fio. Belo Horizonte: 2003.

[MARGI, 2009] MARGI, Cíntia Borges; SIMPLÍCIO JR, Marcos; BARRETO, Paulo S.M.L.; CARVALHO, Tereza C.M.B. Segurança em Redes de Sensores Sem Fio. IX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais . 2009.

[MEDAGLIA, 2010] MEDAGLIA, Carlo Maria; SERBANATI, Alexandru. An overview of privacy and security issues in the internet of things. D. Giusto et al. (eds.), The Internet of Things: 20th Tyrrhenian Workshop on Digital Communications. Roma: 2010.

[MIORANDI, 2012] MIORANDI, Daniele; et al. Internet of things: Vision, applications and research challenges. Ad Hoc Networks 10 (2012) 1497–1516. 2012.

[MIRI, 2012] MIRI, J. A. Lima; NEVINS, M. Analysis of Relay Attacks on RFiD Systems. IEEE Latin America Transactions, VOL. 10, NO. 1, JAN. 2012.

[MIROWSKI; ET AL, 2009] MIROWSKI, Luke; HARTNETT, Jacqueline; WILLIAMS, Raymond. An RFID Attacker Behavior Taxonomy. IEEE CS. PERVASIVE computing. 2009.

[MISHRA, 2002] MISHRA S. Key management in large group multicast. Technical report, Department of Computer Science, University of Colorado; 2002.

[MSP430, 2014] Disponível em: www.ti.com/product/msp430f5437 Acesso em: 23/03/2014 ás 14h22min.

[OLIVEIRA; ET AL. 2012] OLIVEIRA, Luís M. L.; RODRIGUES, Joel J. P. C.; SOUSA, Amaro F. de; LLORET, Jaime. Denial of service mitigation approach for IPv6-enabled smart object networks. Concurrency And Computation: Practice And Experience. Concurrency Computat.: Pract. Exper. 2013; 25:129–142. Published online 12 June 2012 in Wiley Online Library (wileyonlinelibrary.com).

[PALATTELLA, 2013] PALATTELLA , Maria Rita; ACCETTURA , Nicola; VILAJOSANA , Xavier; WATTEYNE , Thomas; GRIECO, Luigi Alfredo; Boggia, Gennaro; Dohler, Mischa. Standardized Protocol Stack for the Internet of (Important) Things. IEEE COMMUNICATIONS SURVEYS & TUTORIALS, VOL. 15, NO. 3, THIRD QUARTER 2013.

[RAZA; ET AL. 2012] RAZA, Shahid; TRABALZA, Daniele; VOIGT, Thiemo. 6LoWPAN Compressed DTLS for CoAP. 8th IEEE International Conference on Distributed Computing in Sensor Systems. 2012.

[RAZA; ET AL. 2013] RAZA, Shahid; SHAFAGH, Hossein; HEWAGE, Kasun; HUMMEN, René; VOIGT, Thiemo. Lithe: Lightweight Secure CoAP for the Internet of Things. IEEE Sensors Journal, VOL. 13, NO. 10, OCTOBER 2013

[RESCORLA; MODADUGU, 2006] RESCORLA, E.; MODADUGU, N. Datagram Transport Layer Security. 2006. RFC 4347: Datagram Transport Layer Security. Internet Engineering Task Force (IETF). 2006.

[RESCORLA; MODADUGU, 2012] RESCORLA, E.; MODADUGU, N. Datagram Transport Layer Security Version 1.2. Internet Engineering Task Force (IETF). RFC Standard 6347,

[ROMAN, 2011] ROMAN, Rodrigo; NAJERA, Pablo; LOPEZ, Javier. Securing the Internet of Things. IEEE Computer Society. Volume:44 , Issue: 9. Pages 51 - 58. SEPTEMBER 2011.

[ROMAN; ET AL, 2011] ROMAN, Rodrigo; ALCARAZ, Cristina; LOPEZ, Javier; SKLAVOS, Nicolas. Key management systems for sensor networks in the context of the Internet of Things. Computers and Electrical Engineering, vol. 37, no 2, 147-159, Março 2011.

[SAADALLAH, 2012] SAADALLAH, Bilel; LAHMADI, Abdelkader; FESTOR, Olivier. CCNx for Contiki: implementation details. TECHNICAL REPORT N° 432, November 2012, Project-Team MADYNES.

[STALLINGS, 2008] STALLINGS, William. Criptografia e Segurança de Redes.. Princípios e Práticas. 4º ed. Pearson: 2008.

[TANENBAUM, 2003] TANENBAUM, Andrew S. Redes de Computadores. 4º ed. Campus: 2003. Pg. 263.

[TOHEED, RAZI, 2010] TOHEED, Qamar; RAZI, Hassan. Asymmetric-Key Cryptography for Contiki.. Master of Science Thesis in the Programme Networks and Distributed Systems.2010.

[VLADISLAV, 2012] VLADISLAV, Perelman. Security in IPv6-enabled Wireless Sensor Networks: An Implementation of TLS/DTLS for the Contiki Operating System. Jacobs University | School of Engineering and Science. 2012.

[VUCINIC; ET AL, 2015] VUCINIC, Malisa; TOURANCHEAU, Bernard; ROUSSEAU, Franck; DUDA, Andrzej; DAMON, Laurent; GUIZZETTI, Roberto. OSCAR: Object security architecture for the Internet of Things. Ad Hoc Networks. 2015.

[WAGENKNECHT, 2013] WAGENKNECHT, Gerald. Energy-Efficient Management of Heterogeneous Wireless Sensor Networks. 2013.