Dissertação de Mestrado

Inatel
Instituto Nacional de Telecomunicações

CONTRIBUIÇÕES NO DESENVOLVIMENTO DO SERVIÇO NOVAGENESIS EMBARCADO PARA INTERNET DAS COISAS

Vâner José Magalhães

Novembro de 2017

Contribuições no Desenvolvimento do Serviço NovaGenesis Embarcado para Internet das Coisas

VâNER JOSÉ MAGALHÃES

Dissertação apresentada ao Instituto Nacional de Telecomunicações, como parte dos requisitos para obtenção do Título de Mestre em Telecomunicações.

ORIENTADOR: Prof. Dr. Antônio Marcos Alberti.

Santa Rita do Sapucaí
2017

Magalhães, Vâner José
M188c
Contribuições no Desenvolvimento do Serviço NovaGenesis Embarcado para Internet das Coisas. / Vâner José Magalhães. – Santa Rita do Sapucaí, 2017.
74 p.

Orientador: Prof. Dr. Antônio Marcos Alberti.
Dissertação de Mestrado em Telecomunicações – Instituto Nacional de Telecomunicações – INATEL.
Inclui bibliografia.

1. NovaGenesis 2. Internet do futuro 3. Internet das coisas 4. Rede colaborativa. 5. Serviço de proxy/gateway embarcado I. Alberti, Antônio Marcos. II. Instituto Nacional de Telecomunicações – INATEL. III. Título.

CDU 621.39

Ficha Elaborada pelo Bibliotecário da Instituição CRB6-2718

Inatel
Mestrado em Telecomunicações

FOLHA DE APROVAÇÃO

Dissertação defendida e aprovada em _____ /_____ /_____,
pela comissão julgadora:

Prof. Dr. Antônio Marcos Alberti
INATEL

Prof. Dr. Juliano Araújo Wickboldt
UFRGS

Prof. Dr. Guilherme Augusto Barucke Marcondes
INATEL

Prof. Dr. Antônio Marcos Alberti
INATEL

Coordenador do Curso de Mestrado
Prof. Dr. José Marcos Câmara Brito

"A vitória mais bela que se pode alcançar é vencer a si mesmo"

Santo Inácio de Loyola

À minha família

Agradecimentos

Agradeço primeiramente a Deus por todos os dons e bênçãos que me deu.

Aos meus pais Francisco Magalhães e Inhulra Aparecida Rosa Magalhães que dedicaram suas vidas por mim.

À minha esposa Ana Luiza de Miranda Magalhães que sempre me apoiou e esteve do meu lado durante esta jornada.

Às minhas filhas Maria Júlia de Miranda Magalhães e Maria Eduarda de Miranda Magalhães, que são minha motivação.

Ao meu irmão João Gabriel Magalhães pelo incentivo e amizade.

Ao Prof. Dr. Antônio Marcos Alberti pela dedicada orientação e pelos valorosos ensinamentos.

Ao Inatel, ICT-Lab, ICC e Ericsson por viabilizarem este sonho.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Quadros

Lista de Siglas

6LowPAN	*IPv6 over Low Power Wireless Personal Area Network*
ADI	Habitante Digital Ativo - *Active Digital Inhabitant*
BLE	*Bluetooth Low Energy*
CL	*Connection Less*
CoAP	*Constrained Application Protocol*
DHT	Tabela *Hash* Distribuida - *Distributed Hash Table*
EPGS	Serviço de *Proxy/Gateway* Embarcado - *Embedded Proxy-Gateway Service*
GIRS	Serviço de Resolução Genérica de Indireção - *Generic Indirection Resolution Service*
GPS	Sistema de Posicionamento Global - *Global Positioning System*
GW	*Gateway*
HT	Tabela *Hash* - *Hash Table*
HTS	Serviço de Tabela *Hash* - *Hash Table Service*
IHC	*Bluetooth Low Energy*
FI	Internet do Futuro - *Future Internet*
FIoT	Internet das Coisas do Futuro - *Future Internet of Things*
ICN	Rede Centrada na Informação - *Information-centric Networking*
Id	Identificador
IHC	*Interhost Communcation*
IoT	Internet das Coisas - *Internet of Things*
IP	*Internet Protocol*
Loc	Localizador

MAC	*Media Access Control*
M-EPGS	EPGS principal - *Main EPGS*
MC	Computador Principal - *Main Computer*
MTU	Unidade Máxima de Transmissão - *Maximum Transmission Unit*
NB	Ligação de Nomes - *Name Binding*
NG	NovaGenesis
NLN	Nome em Linguagem Natural
NRS	Serviço de Resolução de Nomes - Name Resolution Service
OS	Sistema Operacional - Operating System
P-EPGS	EPGS parceiro - *Peer EPGS*
PDI	Habitante Digital Passivo - *Passive Digital Inhabitant*
PDU	Unidade de Dados de Protocolo - *Protocol Data Unit*
PG	*Proxy-Gateway*
PGCS	*Proxy-Gateway-Controller Service*
PSS	*Publish/Subscribe Service*
RPL	*Routing Protocol for Low Power and Lossy Networks*
SDA	Arquitetura Definida por Serviço - *Service-Defined Architecture*
SDN	Rede Definida por *Software* - *Software-Defined Network*
SDU	Unidade de Dados de Serviço - *Service Data Unit*
SLA	Acordo de Nível de Serviço - *Service Level Agreement*
SOA	Arquitetura Orientada a Serviço - *Service-Oriented Architecture*
SOC	Computação Orientada a Serviço - *Service-Oriented Computing*
SVN	Nome Autoverificável - *Self-Verified Name*

Publicações

ALBERTI, A.M.; SCARPIONI, G.D.; MAGALHÃES, V.J.; CERQUEIRA, S.A.; RODRIGUES, J.J.P.C.; RIGHI, R.R.. Advancing NovaGenesis Architecture Towards Future Internet of Things. In: **Special Issue on 5G and Beyond - Mobile Technologies and Applications for IoT**, 2017. IEEE Internet of Things Journal, 2017. p. 394-399. Rio de Janeiro, Brasil.

ALBERTI, A.M.; SCARPIONI, G.D.; MAGALHÃES, V.J.. Rede IoT Colaborativa NovaGenesis. In: **XXXV Simpósio Brasileiro de Telecomunicações e Processamento de Sinais (SBrT 2017)**. SBrT 2017, 3-6 DE SETEMBRO DE 2017, SÃO PEDRO, SP.

Resumo

Magalhães, V.J. Contribuições no Desenvolvimento do Serviço NovaGenesis Embarcado para Internet das Coisas [dissertação de mestrado]. Santa Rita do Sapucaí: Instituto Nacional de Telecomunicações; 2017.

A Internet das Coisas (IoT) está se transformando em uma aplicação chave para as arquiteturas de Internet do Futuro (FI). Muitos acreditam que a maioria dos dispositivos de Internet das Coisas serão sensores e atuadores equipando "coisas" ordinárias, como carros, postes de iluminação, estacionamentos, etc. Nesse contexto, faz muito sentido pensar em Internet das Coisas como uma dos mais importantes requisitos para a Internet do Futuro. Nesta dissertação, é proposto uma extensão da arquitetura NovaGenesis (NG) de Internet do Futuro para o mundo IoT. A NG é apresentada em sua forma original e em seguida é proposto um novo serviço que pode ser embarcado em dispositivos com capacidades restritas. Este novo serviço foi chamado de "Serviço de *Proxy/Gateway* Embarcado" (*Embedded Proxy/Gateway Service* - EPGS). Com a utilização da NG via EPGS, uma Internet das Coisas do Futuro é criada, introduzindo ingredientes como: rede centrada em informação para troca de dados IoT, armazenamento e processamento; arquitetura orientada a serviços para IoT; composição dinâmica de serviços para IoT; e controle por *software* para gerenciamento e operação de dispositivos. Nomeação ilimitada e resolução de nomes da NovaGenesis são aplicadas para encaminhar e rotear dados de IoT com proveniência e integridade. Serviços de IoT são compostos dinamicamente, criando um fluxo de dados entre os serviços e as aplicações interessadas. São avaliados alguns cenários experimentais para demostrar a arquitetura NovaGenesis para IoT utilizando o novo serviço proposto. Por fim, são listadas limitações conhecidas e apresentadas algumas sugestões de melhorias e de trabalhos futuros.

Palavras-chave: Internet das Coisas, Internet do Futuro, NovaGenesis, NovaGenesis embarcada, Internet das Coisas do Futuro.

Abstract

Magalhães, V.J. Contribuições no Desenvolvimento do Serviço NovaGenesis Embarcado para Internet das Coisas [dissertação de mestrado]. Santa Rita do Sapucaí: Instituto Nacional de Telecomunicações; 2017.

The Internet of things is becoming one of the key applications for future Internet architectures. Since many believe that the majority of Internet devices will become sensors and actuators equipped over ordinary "things", like cars, lampposts, parking lots, city squares, etc., it makes a lot of sense to think on the Internet of things as one of the most important requirements for the future Internet (FI). In this dissertation, an extension of the NovaGenesis (NG) future Internet is presented in order to adapt it to the IoT world. The NG is presented in its original form and then a new service is proposed to run embedded in devices with restricted capabilities. This new service has been called "Embedded Proxy/Gateway Service" (EPGS). With the use of NG via EPGS, a Future Internet of Things (FIoT) is created, introducing ingredients such as: information-centric networking for IoT data exchanging, storage, and processing; service-oriented architecture of IoT dynamic IoT services composition; and software-control for IoT devices management and operation. NovaGenesis unlimited naming and name resolution is applied to forward and route IoT raw data with provenance and integrity. IoT services are dynamically composed, creating a data chain from device representative/gateway services to interested applications. Some experimental scenarios were evaluated in order to demonstrate the NovaGenesis architecture for IoT using the proposed new service. Finally, some limitations are listed and some suggestions for improvements and future work are presented.

Keywords: Internet of things, future Internet, NovaGenesis, Embedded NovaGenesis.

Capítulo 1

Introdução

A Internet não é mais uma rede fixa de computadores entre instituições governamentais. Ela invadiu muitos aspectos de nossa vida e sociedade, mudando estilos de vida, prática nos negócios, relações de trabalho entre empregados e empregadores, canais de comunicações e até mesmo em interações sociais. Como a Internet foi aberta para o público geral e começou a ser usada por uma crescente diversidade de aplicações, seu papel e tamanho mudaram consideravelmente do seu propósito original [1][2]. Muitas soluções paliativas foram aplicadas para estender seu escopo, por exemplo, IPSec, *mobile* IP (MIP), IPv6, incluindo os mais recentes focados em conectar dispositivos restritos à Internet, como o *Constrained Application Protocol* (CoAP), *IPv6 over Low Power Wireless Personal Area Network* (6LowPAN) e *Routing Protocol for Low Power and Lossy Networks* (RPL). Estas extensões evolucionárias tentam preencher a visão de (Internet das Coisas - *Internet of Things* (IoT)), que pode ser definida como conectar bilhões de coisas ordinárias à Internet. Preocupados de que as tecnologias atuais de Internet não se adequem para suportar inteiramente este crescimento exponencial do número de dispositivos, mobilidade, interatividade, conteúdo, tráfego, segurança, e problemas de privacidade, muitas iniciativas surgiram pelo mundo para redesenhar a Internet sob o nome de Internet do Futuro - *Future Internet* (FI). [3][4][5].

O termo Internet do Futuro foi adotado nas primeiras iniciativas preocupadas em repensar a Internet, incluindo o projeto 4D [6], a iniciativa *Future Internet Design* (FIND) [7], o *Global Environment for Network Innovations* (GENI) [8], e a iniciativa *European Future Internet Assembly* (FIA) [9]. Por FI, queremos dizer qualquer rede parecida com a Internet que surja no futuro. Isto inclui iniciativas evolucionárias, em que os protocolos fundamentais da Internet atual são mantidos e novas ideias são introduzidas aos poucos de forma incremental, ou iniciativas revolucionárias, em que a arquitetura é totalmente redesenhada. Exemplos de arquiteturas de FI evolucionárias que englobam IoT são: FIWARE [10], DIAT [11] e OpenIoT [12]. O FIWARE provê uma plataforma para integrar serviços via interfaces de serviço de próxima geração. A *Distributed Internet-Like Architecture for Things* (DIAT) é uma arquitetura que tenta resolver o problema da heterogeneidade dos dispositivos IoT, bem com a escalabilidade, segurança e interoperabilidade sem a necessidade da interversão humana. Por fim, a OpenIoT visa a interoperabilidade dos diversos cenários possíveis em IoT integrado a computação em nuvem. Ele é compatível com o padrão da W3C para redes de sensores semânticos para representar sensores físicos e virtuais. Sua contribuição é na solução do problema de combinação dos dados providos por diferentes cenários IoT.

Tipicamente, as arquiteturas de Internet do Futuro com foco em IoT endereçam algum dos seguintes paradigmas: rede centrada na informação (information-centric networking - ICN) [1, 13, 14], arquitetura orientada à serviços (*service-oriented architecture* - SOA) [13, 15], rede definida por software (*software-defined networking* - SDN) [16], rede centrada em serviço (*service-centric networking* - SCN) [17], virtualização de função de rede (*network function virtualization* - NFV) [18, 19], nomes autoverificáveis (*self-verifying naming* - SVN) [20, 13, 21], separação de identificador/localizador [22], e resolução distribuida de nomes.

Surpreendentemente, pouquíssimas iniciativas revolucionárias (também chamadas propostas *clean slate*) estão cobrindo IoT – apesar do fato de provavelmente a grande maioria dos dispositivos na Internet do futuro serem "coisas".

O *Information and Communications Technologies* (ICT) *Laboratory* do Inatel está desenvolvendo uma arquitetura chamada NovaGenesis (NG) [23] que tem por objetivo recriar a Internet utilizando as tecnologias mais adequadas e modernas. Ela engloba vários ingredientes chave para a construção de uma nova Internet, que dificilmente são encontrados juntos em outras abordagens de FI. Para a NovaGenesis (NG), IoT é um ponto chave e algumas iniciativas estão sendo tomadas para melhor atender esta utilização. Por estes motivos, esta dissertação escolheu a NovaGenesis como arquitetura de Internet do Futuro para os cenários IoT propostos.

A NG foi desenvolvida em linguagem de programação C++, utilizando estruturas de dados e códigos desenhados para serem executados em computadores com grande capacidade de memória e processamento, com poucas restrições de recursos. O propósito desta dissertação é estender o escopo da NovaGenesis a fim de adaptá-la para a utilização em dispositivos com capacidades restritas (processamento, memória e energia), permitindo a utilização do conceito NG em IoT. Para isso foi criado um serviço chamado EPGS, que nada mais é do que uma versão compacta e otimizada das funcionalidades principais da NG para ser embarcado em dispositivos tipicamente utilizados em IoT.

O EPGS deve conhecer todo o protocolo de comunicação NG, gerar nomes, descobrir parceiros, fazer contratos e prover dados úteis. Tudo isso utilizando de forma otimizada os recursos providos pelo *hardware*, seja na utilização de memória, processamento e consumo de energia.

O Capítulo 2 explica o objetivo da NG, apresenta seus conceitos e por fim, faz um resumo da implementação atual da arquitetura. No Capítulo 3 é apresentado o novo serviço NG proposto por esta dissertação, visando estender a NG ao mundo IoT. O EPGS é explicado teoricamente, desde a criação de uma linha de comando até a publicação dos dados sensoreados. O Capítulo 4 detalha tecnicamente como o EPGS foi desenvolvido, quais tecnologias foram utilizadas e a estrutura do projeto. O Capítulo 5 apresenta dois cenários experimentais utilizados para validar o ciclo de vida do EPGS, coletar métricas e estender seu escopo até criar uma rede colaborativa para dispositivos IoT. Por fim, o Capítulo 6 conclui a dissertação, apresenta um lista com sugestões de melhorias e deixa sugestões para trabalhos futuros a partir do EPGS.

Capítulo 2

NovaGenesis

O projeto NovaGenesis (NG) começou a ser desenvolvido em 2008 com o objetivo de redesenhar a Internet utilizando as melhores tecnologias atuais. A NG estende o escopo de uma rede pois além da troca de dados, ela inclui funcionalidades de processamento e armazenamento (*cache* de rede) dos dados.

Dentre um conjunto de ingredientes *estado-da-arte* em se tratando de FI, a NG utilizou a nomeação, resolução de nomes, ciclo de vida de conteúdos e serviços, e controladores/representadores em *software* das "coisas". O objetivo é criar um *framework* genérico que pudesse lidar como nomeação, ciclo de vida, identificação persistente, independência de localização, diversidade de conteúdo, serviços, e coordenação de "coisas" (orquestração).

2.1 Conceitos Fundamentais

2.1.1 Nomeação, Nomes Autoverificáveis, e Resolução de Nomes

Nomes são uma forma de denotar existências. Eles podem ou não carregar significado. Chamamos de Nome em Linguagem Natural (NLN) um nome que possui significado, como por exemplo "Gato". Os nomes que não possuem significado para as pessoas são chamados Nomes Planos - *Flat Names*. Eles podem ser gerados a partir da passagem de um NLN por uma função *Hash*. Isso gera um Nome Autoverificável - *Self-Verified Name* (SVN), uma vez que o NLN é embaralhado. O nome é dito ser autoverificável, porque em qualquer tempo, as palavras binárias da existência podem ser embaralhadas (*hashed*) novamente e obter exatamente o mesmo nome. A NG utiliza o algoritmo de *Hash* chamado MurMurHash3 [24]. Uma vez que a NG engloba os dois tipos de nomes, é capaz de fazer ligações que representam relações entre entidades nomeadas [25][26][27][28][21]. Uma Ligação de Nomes - *Name Binding* (NB) é um mapeamento entre dois ou mais nomes na forma: < chave; valor(es) >. A Figura 2.1 ilustra um nó de IoT sendo que "NXP" é o nome do *hardware*, "*Event* OS" é o nome do Sistema Operacional (OS - *Operating System*) e "Processo 1", "Processo 2", "Processo 3" e "Processo 4" são nomes de processos que estão rodando neste OS. Assim sendo, existe uma relação entre estas entidades nomeadas, que pode ser representada pelos seguintes NBs:

- < **NXP, *Event* OS** > - O *Hardware* "NXP"possui o sistema operacional "*Event* OS".
- < **Event OS, Processo 1** > - O "*Event* OS"possui um processo chamado "Processo 1".
- < **Event OS, Processo 4** > - O OS "*Event* OS"possui um processo chamado "Processo 4".

Nestes exemplos, a ligação é feita entre NLN, mas poderia ser feita utilizando SVN:

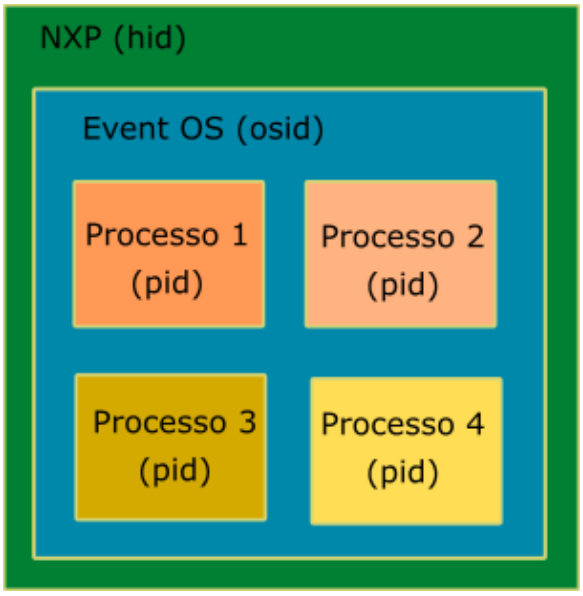

Figura 2.1: Exemplo de nomeação de um dispositivo IoT. O nome "NXP" denota o identificador do hardware (hid); o nome "Event OS" é o identificador do sistema operacional (osid); o nome "Processo 1" é o identificador de um processo (pid), assim como os nomes "Processo 2", "Processo 3" e "Processo 4".

- < **7E3FFFA6, 027552A1** > - 7E3FFFA6 é o SVN do Identificador do *Hardware* "NXP" e 027552A1 é o SVN do "*Event* OS".
- < **027552A1, 7CE42568** > - O OS "*Event* OS" possui um processo chamado "Processo 1".

2.1.2 Identificação, Localização e Separação Id/Loc

A identificação e localização são uma necessidade para todos os sistemas de comunicação. Um Identificador (Id) é um nome denota a existência de alguma entidade em um certo escopo e um Localizador (Loc) é um nome que aponta para uma posição onde um entidade pode habitar ou estar ligada e provê uma noção de distância entre posições dentro de um espaço [29]. A NG utiliza SVNes como identificadores, assim como outras arquiteturas de FI, especialmente a NetInf [14] e o XIA [13].

Alguns sistemas de comunicação como o IPv4 e o IPv6, não separam identificação de localização, causando assim um grande problema quando se trata de mobilidade. Com o identificador igual o localizador, uma entidade pode não ser encontrada quando é movida. Para evitar este problema, a NG utiliza um nome para o identificar e outro nome para localizar, fazendo a separação Id/Loc. Na NG, uma Ligação de Nomes - *Name Binding* (NB) é feita entre o identificador e o localizador a fim de conectar a identificação e a localização de uma entidade. Por exemplo, um dispositivo IoT pode usar o seu próprio nome (número de série, código de barras, etc) como identificador (Id), neste exemplo o nome identificador é "NXP". Esse dispositivo está fisicamente localizado pelas coordenadas Sistema de Posicionamento Global - *Global Positioning System* (GPS) "-22.251929,-45.7092637". Essas coordenadas podem ser utilizadas como localizador (Loc). Então, a NG faz uma NB entre o Id e o Loc: < ‘‘`-22.251929,-45.7092637`’’, ‘‘`NXP`’’ >.

A mobilidade pode ser provida alterando este NB. Se por algum motivo o "NXP" se movimentar, basta modificar esta única ligação de nome, por exemplo para: < ‘‘`-22.239177, -45.705877`’’, ‘‘`NXP`’’ >, sem necessidade de atualizar outros nomes da rede. Note que o identificador permaneceu o mesmo: "NXP". Desta forma, para encontrar este dispositivo basta

buscar pelo seu nome, já que o identificador não é alterado. Assim, tem-se a independência entre identificadores (Id) e localizadores (Loc), o que é muito importante quando se trata de IoT e dispositivos móveis [22].

Supondo que exista um outro dispositivo localizado pelo nome "-22.256589,-45.6963542". Baseado neste localizador (neste exemplo, as coordenadas GPS) é possível prover um noção de distância entre os dois dispositivos.

2.1.3 Serviços e Contratos

Um programa de computador, um processo ou qualquer outra entidade virtual que tenha como objetivo processar, trocar ou armazenar informação pode ser considerado um serviço [30].

Protocolos são regras que as partes envolvidas devem saber previamente e obedecer a fim de comunicar. Na NG os protocolos de comunicação são implementados como um serviço, pois eles processam, trocam e armazenam informações para construir redes.

A NG faz uso de microserviços para compor a arquitetura de rede. O ciclo de vida dos serviços é baseado em contratos, isto é, após a descoberta dos serviços candidatos a trabalharem juntos (pares), um contrato é formulado e negociado. Assim, um contrato pode ser definido como uma peça de informação que representa os limites, responsabilidades, cláusulas a serem respeitadas, bem como os critérios para completar e punir serviços mal executados. Um sinônimo típico para contrato é um Acordo de Nível de Serviço - *Service Level Agreement* (SLA). A negociação do SLA é um passo muito importante no ciclo de vida dos serviços, que inclui muitos outros passos, como instanciação, exposição de funcionalidades/capacidades, descoberta de pares candidatos, contratação, monitoramento, *log*, finalização e estimação de reputação. Este modelo de serviço baseado em contratos é fundamental para IoT, pois privacidade de dados sensíveis deve ser tratada por serviços confiáveis.

2.1.4 *Proxies*, *Gateways*, e Controladores

Um *proxy* é um serviço de procurador, que representa outros componentes na rede. Ele expõe as capacidades e os recursos para outros serviços interessados. Um *gateway* é um portal, tradutor, um ponto de entrada/saída entre diferentes tecnologias. O *gateway* traduz ou encapsula mensagens de certo protocolo de entrada para o protocolo de saída desejado. Por exemplo, um *gateway*, entre as tecnologias Wi-Fi e *Bluetooth* pode extrair a carga útil do quadro Wi-Fi e encapsula-la dentro de quadros *Bluetooth*. O modelo NovaGenesis também define a noção de controlador por *software*. Um controlador é um serviço que controla e/ou toma decisões sobre as configurações de recursos físicos ou outros serviços. O modelo NovaGenesis aplica as ideias de Rede Definida por *Software* - *Software-Defined Network* (SDN) de forma mais geral no contexto IoT, por exemplo, empregando controladores por *software* para configurar outras funcionalidades além do encaminhamento de quadros.

2.2 Implementação Atual

A NG está em constante desenvolvimento e pode ser divida em três frentes: (i) serviço de resolução de nomes e armazenamento das informações de rede; (ii) *Proxy-Gateway-Controller Service* (PGCS) que representa e encapsula as mensagens no formato exigido pela camada de enlace; (iii) e os aplicativos de usuários, que são clientes ou fornecedores de serviços.

2.2.1 Resolução de Nomes e Armazenamento de Informações de Rede

As Ligação de Nomes - *Name Binding* (NB) são armazenadas em Tabelas *Hash* Distribuidas - *Distributed Hash Tables* (DHTes). As DHTes armazenam as ligações de nomes publicados e assinados. Durante a publicação de uma ligação de nomes para a DHT, um serviço define quais

outros serviços estão autorizados a assinar esta NB. Portanto, NBs são acessíveis somente para serviços autorizados. Quando um serviço inicializa, ele publica ligações entre muitos NLNs e SVNes (um sub-grafo de nomes), expondo seus nomes para outros serviços. Serviços que representam "coisas" (*proxies*) podem expor suas capacidades e estados publicando NBs para outros serviços. Esta abordagem cria uma grande *web* de ligações de nomes que é acessada pelo modelo de publicação/assinatura e armazenada de forma distribuída em tabelas *hash*. Isto cria um Serviço de Resolução de Nomes - Name Resolution Service (NRS).

Serviço de Publicação/Assinatura – *Publish/Subscribe Service* (PSS)

Ligações entre nomes e eventuais conteúdos associados são publicadas por qualquer serviço através do envio de mensagem para uma instância do *Publish/Subscribe Service* (PSS). Este serviço permite publicação e assinatura de conteúdos associados a ligações de nomes, estendendo o NRS como um *cache* de rede. A razão para isto é que tipicamente conteúdos são armazenados nos sistemas operacionais (OS) como arquivos, que tem um nome. Portanto, uma ligação de nomes pode relacionar o conteúdo do nome com seu nome de arquivo no sistema de arquivos do OS. O PSS armazena quais serviços estão pré-autorizados a acessar os NB e/ou dados publicados. O PSS tem seis primitivas: (i) publicar NB; (ii) publicar NB/dados e notificar outros serviços sobre a publicação; (iii) assinar um NB/dados; (iv) assinar um NB/dados e notificar outros serviços sobre a assinatura; (v) entregar uma assinatura de NB/dados; e (vi) revogar um NB/publicação de dados. O PSS pode ser visto como uma API distribuída, acessível via identificadores PSS.

Serviço de Tabela *Hash* - *Hash Table Service* (HTS)

Responsável por armazenar os NBs e seus conteúdos associados, se houver algum. Considere por exemplo o Processo 3 no NXP da Figura 2.1. Imagine que este processo está gerando conteúdo hipotético, que são dados de medidas de temperatura e salvando no arquivo "*Sample* 3". Este conteúdo é armazenado como um arquivo em uma pasta de uma instância Serviço de Tabela *Hash* - *Hash Table Service* (HTS). Além disso, o conteúdo hipotético do arquivo, quando submetido à função *hash*, gera um SVN que por exemplo, poderia ser "291074AA". A ligação de nomes < 291074AA, Sample 3 > pode relacionar o SVN do conteúdo com seu NLN.

Para uma maior organização dos dados e permitir escalabilidade, o HTS divide os NBs em categorias. Além disso, os nomes podem ser de um entidade Habitante Digital Passivo - *Passive Digital Inhabitant* (PDI) ou um Habitante Digital Ativo - *Active Digital Inhabitant* (ADI). Um PDI é algum estimulo de entrada, intermediário, ou de saída que será ou foi tratado por um ADI. Exemplos de PDI são dados sensoreados, códigos fonte, textos, imagens, vídeos, nomes de arquivos, descritores. Já os ADI são entidades que realizam operações em resposta a um estímulo ou comando, como por exemplo, processos ou aplicativos.

Para cada categoria, o HTS possui uma tabela *hash* ou *Hash Table* (HT) correspondente.

O Quadro 2.1 exemplifica algumas das principais categorias já criadas.

Além das categorias citadas no Quadro 2.1, existem outras predefinidas que não serão abordadas nesta dissertação.

Serviço de Resolução Genérica de Indireção - *Generic Indirection Resolution Service* (GIRS)

O PSS não armazena os NB/dados publicados. Ele encaminha para uma instância do Serviço de Resolução Genérica de Indireção - *Generic Indirection Resolution Service* (GIRS), que seleciona um HTS para armazená-los. Assim, NB/dados publicados são encaminhados pelo GIRS para uma instancia HTS, onde eles são armazenados por uma estrutura de tabela *hash*. Portanto, o GIRS é um serviço intermediário entre instâncias do PSS e HTS. Na implementação atual, o GIRS calcula o resto da divisão da chave NB pelo número de instâncias

Quadro 2.1: Algumas das principais categorias.

Código	**Descrição**
Categoria 1	Ligação entre um PDI e seu nome autoverificável (SVN). Por exemplo, a ligação entre o NLN "PGCS" com o seu SVN correspondente, "BEE09CC1".
Categoria 2	Ligação entre o SVN de um PDI com o SVN de um ou mais ADIs. Por exemplo, a ligação entre o SVN do exemplo acima, "BEE09CC1" com o SVN do identificador do processo do "PGCS", que pode ser "7E764DC1".
Categoria 6	Ligação entre o SVN do identificador de uma entidade física (um hardware) com o SVN de um ADIs. Por exemplo, a ligação entre o SVN do identificador de um dispositivo de IoT com o SVN do identificador de um aplicativo que está rodando neste dispositivo.
Categoria 9	Ligação entre o SVN do identificador de uma entidade física (um hardware) com um PDIs. Por exemplo, a ligação entre o SVN do identificador de um dispositivo de IoT com um NLN que descreve este dispositivo.
Categoria 17	Ligação entre os NLN de identificadores que não fazem parte da NovaGenesis. Por exemplo, a ligação entre o nome de um sensor com o nome de suas características.

HTS, selecionando uma instância HTS para armazenar o NB de acordo com o resultado deste cálculo.

2.2.2 Serviço de *Proxy/Gateway* e Controlador - *Proxy-Gateway-Controller Service* (PGCS)

Adicionalmente aos sistemas de resolução de nomes e *cache* de rede, um quarto serviço *Proxy-Gateway-Controller Service* (PGCS) é implementado. O PGCS prove: (i) serviço de *gateway* para encapsulamento (e extração) de mensagem sobre as tecnologias de rede já estabelecidas, como Ethernet ou Wi-Fi; (ii) um serviço de *proxy* para representar outros serviços NovaGenesis dentro de um Sistema Operacional - Operating System (OS); (iii) serviço de controlador com funções para inicializar o domínio. O PGCS permite que o PSS, GIRS e HTS se descubram durante a inicialização. Visto que um endereço é um nome que denota a posição onde uma existência pode habitar ou ser acoplada, o PGCS envia para o HTS, ligações de nomes entre formatos de endereço já estabelecidos (exemplo, um endereço no mundo real ou um MAC de *Ethernet* emulado) e/ou endereços NG. O PGCSs também tem uma tabela *Hash* interna onde ele copia/armazena localmente NBs descobertos. Independentemente do formato de endereço usado para conectar PGCSs, dentro da NG toda comunicação é orientada a nomes. Adicionalmente às funcionalidades do *gateway*, o PGCS publica NBs sobre serviços NG dentro de um OS para outros PGCSs no mesmo domínio. Finalmente, serviços de usuários (ou aplicativos) podem expor seus NBs/dados para outros serviços via API PSS. Assim, outros serviços podem assinar NBs de serviços ou conteúdo. O PSS faz o encontro entre publicadores e assinantes, permitindo que eles descubram como nomes publicados são relacionados entre si de uma forma segura. Todo ciclo de vida dos serviços é baseado na API PSS. Eventualmente, serviços pode assinar NBs sucessivamente para identificar e localizar outras existências, armazenando estes NBs nas suas estruturas internas, encaminhando e roteando informações baseada neles. Os nomes NovaGenesis são relacionados ao endereço MAC habilitando encaminhamento e roteamento sobre Wi-Fi e Ethernet.

2.2.3 Aplicativos de Usuários

Os aplicativos de usuários são quaisquer *softwares* que implementem os conceitos do modelo de Internet do Futuro NG. Eles podem prover ou consumir dados mediante assinatura de contrato. Existe infinidade de possibilidades, como por exemplo, aplicativos consumidores de métricas (temperatura, umidade, luminosidade), aplicativos para publicação de conteúdo (fotos, vídeos, mensagens), aplicativos para configurar dispositivos IoT, etc.

Capítulo 3

Desenvolvimento do Serviço NovaGenesis para IoT

Durante o desenvolvimento do projeto NovaGenesis, viu-se a necessidade de expandir seu horizonte para além das redes compostas por dispositivos com alta capacidade de processamento e grande quantidade de memória. A NG devia ser capaz de ser aplicada em dispositivos com capacidades computacionais restritas, visando sua utilização embarcada em dispositivos de IoT ou *smartphones*.

Dada essa necessidade, foi criado o Serviço NovaGenesis Embarcado que foi chamado de Serviço de *Proxy/Gateway* Embarcado - *Embedded Proxy-Gateway Service* (EPGS). Ele é uma extensão do projeto NG visando integrar Internet do Futuro com IoT.

3.1 O Serviço de *Proxy/Gataway* Embarcado - *Embedded Proxy Gateway Service* - EPGS

Ele foi desenvolvido como uma implementação reduzida (*lightweight*) do PGCS da NG para rodar embarcada, mas mantendo todas as características de segurança, confiabilidade, baseada em contratos e com a arquitetura controlada por *software*. A arquitetura do EPGS permite que ele gere Nome Autoverificável - *Self-Verified Name* (SVN) para qualquer entidade, característica ou informação relevante para o sistema e armazene em tabelas categorizadas. Ele é capaz de criar e analisar linhas de comando NG, implementar o tratamento de mensagens NG e seu *payload* (caso exista). Adapta as mensagens para serem enviadas por diferentes arquiteturas de camada de enlace, tratando a fragmentação e a remontagem de dados recebidos.

O EPGS tem como premissa permitir seu embarque e integração com diferentes dispositivos e sistemas operacionais. Desta forma, métodos com forte dependência do OS foram criados de forma que possam ser facilmente adaptados. Ele também provê métodos para receber informações sobre o dispositivo, características dos sensores e dados coletados.

O EPGS otimiza os algorítimos para utilização mínima de memória ROM e processamento (para economia de energia) e a utilização de memória RAM é feita de forma dinâmica e pontual. Em outras palavras, dados são alocados somente quando necessários e removidos assim que não forem mais necessários.

Desta forma, o EPGS foi capaz de implementar as funcionalidades do PGCS em uma versão capaz de ser embarcada na grande parte dos dispositivos IoT do mercado.

A Figura 3.1 ilustra a construção em camadas do EPGS, sendo que no topo da pilha estão as tarefas (ações) que serão executadas.

Na transmissão, as tarefas são codificadas em diversas Linhas de Comando NG. Os comandos são encapsulados em uma Mensagem NG. A mensagem é adaptada pelo *Proxy/Gateway* para

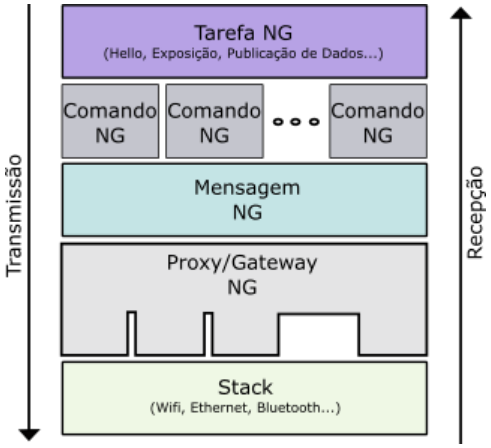

Figura 3.1: Modelo em Camadas da arquitetura do EPGS.

uma das arquiteturas de rede (*Stack*) e por fim enviadas ao destinatário. O *Proxy/Gateway* é uma versão reduzida do PGCS.

Na recepção, os dados recebidos da arquitetura de rede (*Stack*) são processados pelo *Proxy/-Gateway* e então convertidos em Mensagens NG. Da mensagem são retirados os Comandos NG que formarão a tarefa a ser executada.

3.2 Geração de Nomes Autoverificáveis

O EPGS deve ser capaz de gerar Nome Autoverificável - *Self-Verified Name* (SVN) de 4 *bytes* para todas as características e funcionalidades do *hardware* onde será embarcado. Para atingir este requisito, foi utilizado o algoritmo **MurMurHash3** para geração de *hash codes*. Assim, para uma entrada de tamanho arbitrário, é gerado um nome de 4 *bytes* (valor utilizado na atual versão). Notou-se porém, que para entradas com pequeno tamanho havia uma maior chance de acontecer colisão de *hash*, ou seja, SVNes iguais para entradas diferentes. Para evitar este comportamento, um algoritmo pré *hash* foi incluído. Ele consiste em multiplicar cada *byte* de entrada por um vetor com os 16 primeiros números primos (do 2 até o 53) antes de enviá-los para o algorítimo de *Hash*. Desta forma, o tamanho da entrada é aumentado em 16 vezes e a utilização de números primos prove uma melhor distribuição do resultado. O aumento da entrada e o espalhamento dos dados ajuda a reduzir as colisões. A Figura 3.2 exemplifica este algoritmo.

3.3 A Linha de Comando NovaGenesis

Todas as tarefas executadas pelo EPGS são compostas por um conjunto de **Linhas de Comando NovaGenesis**. Uma Linha de Comando NG é um texto, feito em uma única linha, e cada parâmetro é separado por um espaço. Ela pode ser dividido em cabeçalho e argumentos como mostrado na Figura 3.3.

No cabeçalho temos o identificador NovaGenesis, o nome, as alternativas e a versão do comando. Todo comando começa com os caracteres “ng” minúsculos, que identificam o comando NovaGenesis. O próximo parâmetro do cabeçalho é o nome do comando, precedido por um sinal de menos. Em seguida, vem o parâmetro das alternativas do comando, precedido por dois sinais de menos. E por fim, temos a versão. A segunda parte do comando é composta pelos argumentos. Essa seção é encapsulada por colchetes “[...]”. Pode existir um ou mais argumentos.

A Figura 3.4 representa os argumentos genéricos de uma linha de comando.

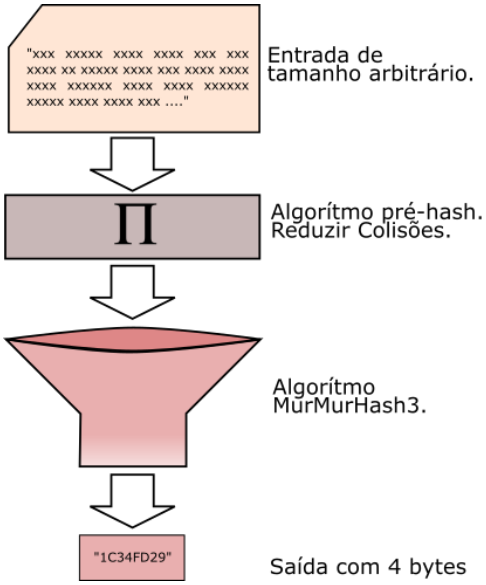

Figura 3.2: Ilustração do algoritmo de geração de Nomes Autoverificáveis.

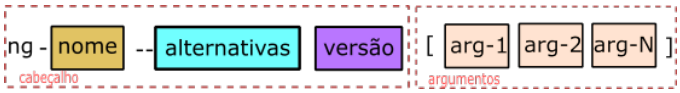

Figura 3.3: Formação básica de uma linha de comando NG.

Cada argumento é encapsulado por sinais de menor e maior "< . . . >". O primeiro parâmetro do argumento indica a quantidade elementos que ele contém. Já o segundo parâmetro indica qual é o tipo dos elementos. E por fim vêm os elementos.

Este é um exemplo real de comando:

```
ng -hello --ihc 0.1 < 5 s NULL NULL Ethernet eth0 ac:22:0b:13:01:34 > ]
```

É um comando NovaGenesis (ng) com nome *hello* com alternativa ihc (*interhost communication*) e versão 0.1. Além disso ele possui somente um argumento. Esse argumento é composto por 5 elementos do tipo *String* (s). Os elementos são as palavras: NULL, NULL, Ethernet, eth0 e ac:22:0b:13:01:34.

3.4 A Mensagem NovaGenesis

A Mensagem NovaGenesis engloba todos os comandos e dados necessários para a comunicação e processamento. Ela é composta por várias Linhas de Comando e opcionalmente pode conter uma sessão de dados (*Payload*).

Cada Linha de Comando ocupa uma linha da mensagem e utiliza o terminador padrão Linux (*carriage return*), ou seja, o caractere ASCII '\n' ou 0x0A.

A segunda parte da mensagem é composta pelos dados (*payload*) a serem transmitidos. Essa parte é opcional. Caso exista, ela é separada da seção de comandos por um caractere de nova linha, como descrito acima.

Um exemplo de dado a ser transmitido é um arquivo contendo informações obtidas por um sensor. O formato do arquivo é de livre escolha, mas neste exemplo escolhemos o formato JSON, que é um padrão de formatação de dados que facilita a leitura/escrita por humanos e

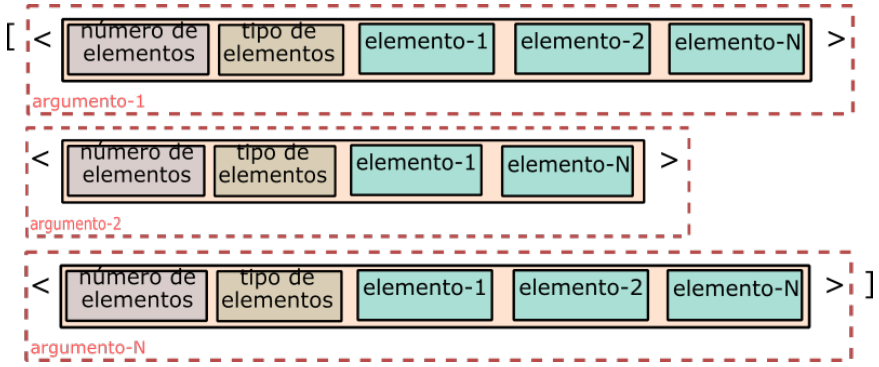

Figura 3.4: Estrutura genérica dos argumentos de uma linha de comando NG.

a interpretação/geração pelas máquinas, além de necessitar de menos memória para guardar a informação útil.

A Figura 3.5 é um exemplo de mensagem NG.

```
ng -m --cl 0.1 [ < 1 s 15B239D1 > < 4 s 7E3FFFA6 027552A1 7CE42568 NULL > < 4 s 0BD95286 ED12F3ED DECC2634 82C5E549 > ]
ng -p --notify 0.1 [ < 1 s 18 > < 1 s 6153124D > < 1 s Temperature.json > < 5 s pub 0BD95286 ED12F3ED A1C31B34 BEAC1234 > ]
ng -info --payload 0.1 [ < 1 s Temperature.json > ]
ng -message --type 0.1 [ < 1 s 1 > ]
ng -message --seq 0.1 [ < 1 s 19 > ]
ng -scn --seq 0.1 [ < 1 s 89D6557E > ]

{ Temperature:32 }
```

Figura 3.5: Exemplo de uma Mensagem NovaGenesis.

São seis comandos, cada um em uma linha terminada por '\n'. Separando a seção de Linhas de Comandos da seção de dados tem-se outro terminador de linha '\n'. A seção de dados é composta por um JSON com uma medida de temperatura.

A mensagem é considerada válida e portanto pode ser transmitida somente se conter mais de duas linhas de comando. Mensagens com duas ou menos linhas de comando são consideradas inválidas e portando descartadas.

3.5 O *Proxy/Gateway* (PG)

A mensagem preparada deve ser enviada ao destinatário. Essa função é atribuída ao módulo *Proxy/Gateway* do EPGS. Ele é uma versão leve do PGCS.

A função do *Proxy* representar o serviço disponibilizado pelo dispositivo onde o EPGS esta embarcado para toda a rede NG.

O *Gateway* é responsável por adaptar a Mensagem NovaGenesis para ser tratada por uma dessas arquiteturas de camada de enlace. Ele é capaz de adicionar campos de cabeçalho, informações de fragmentação e tamanho da Mensagem NG, como ilustrado na Figura 3.6. Ele é bidirecional, portanto é utilizado tanto na transmissão quanto na recepção. A Figura 3.7 exemplifica uma mensagem pronta para ser enviada via Wi-Fi.

O EPGS NovaGenesis deve ser capaz de utilizar diferentes arquiteturas de camada de enlace (*stacks*), como *Ethernet*, Wi-Fi, *Bluetooth*, LoRa, IEEE 802.15.4, etc, para sua comunicação. Como cada arquitetura tem uma particularidade, com por exemplo a necessidade de pareamento, campos de cabeçalho ou o tamanho da Unidade Máxima de Transmissão - *Maximum Transmission Unit* (MTU), um parte do *gateway* pode ser customizada para se adequar as necessidades da camada de enlace. Para a comunicação entre os nós na rede NG é exigido somente que sejam enviados os cabeçalhos de Fragmentação, Tamanho de Mensagem e a Mensagem em si. Em outras palavras, o EPGS é compatível com qualquer camada de enlace que permita o envio de um *payload* independente da MTU.

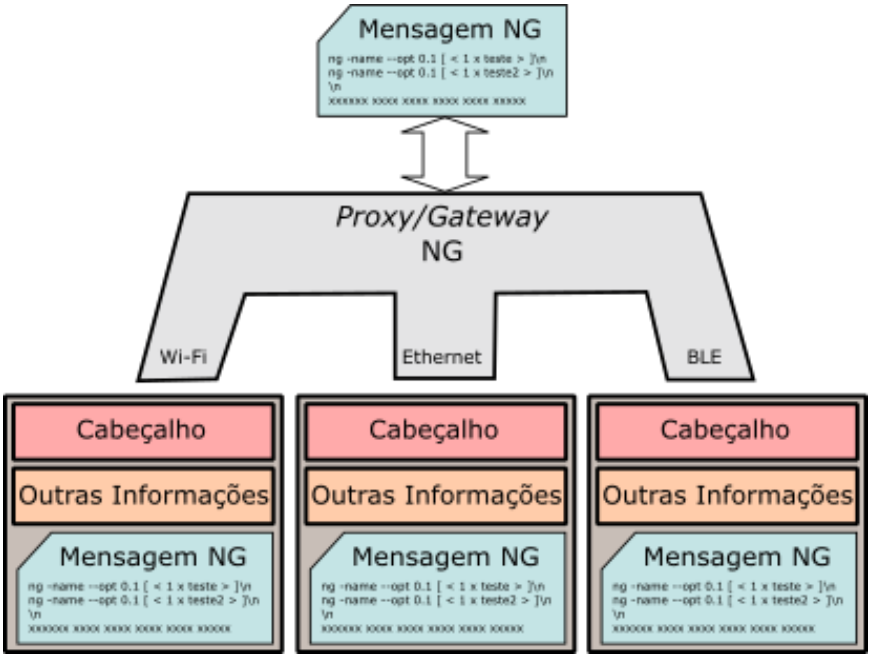

Figura 3.6: Ilustração do funcionamento do *Proxy/Gateway* NovaGenesis.

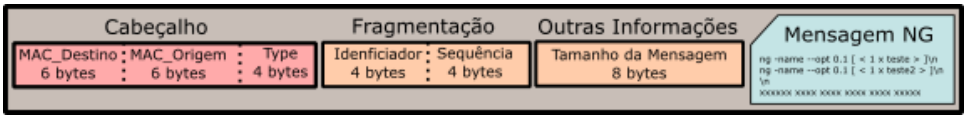

Figura 3.7: Mensagem NG e cabeçalhos preparados pelo PG e prontos para envio.

3.6 As Tarefas

Uma tarefa NovaGenesis é a solicitação de alguma ação ou a confirmação de execução de alguma ação. Ela é uma mensagem composta por diversas linhas de comando a fim de produzir uma ação ou confirmação desejada. Como o EPGS é uma versão reduzida da NovaGenesis, somente as seguintes tarefas são implementadas no ciclo de vida do EPGS: Anúncio (*Hello*), Exposição de Palavras Chave (*Exposition*), Publicação de Oferta de Serviço (*Service Offering*), Negociação de Contrato (*Publication/Subscription*) e Publicação de Dados (*Publication*).

Uma tarefa a ser executada é representada por uma Mensagem NG. A Mensagem é composta por várias Linhas de Comando NG, sendo que primeira e a última são mandatórias. A primeira Linha de Comando de uma Mensagem é sempre a Linha de Comando de Roteamento. Em seguida, são as Linhas de Comando particulares de cada tarefa seguidas por duas linhas de Comando de controle, a Linha de Comando de Tipo de Mensagem e a Linha de Comando de Sequência de Mensagem. Por fim, vem a Linha de Comando de Verificação. A tarefa de *Hello* é uma exceção, pois não contém as Linhas de Comando de Tipo e de Sequência de Mensagem. Uma tarefa é considerada válida somente se a mensagem conter as duas linha mandatórias (Roteamento e Verificação) e pelo menos mais uma linha particular da tarefa. Mensagens com menos de 3 linhas de comando são descartadas.

A **linha de comando de roteamento**, como demonstrada na Figura 3.8 é responsável por expor os seus SVNes aos seus pares e também determinar o destino da mensagem.

O nome do comando é "m" e possui o argumento "cl" (*connection less*) e sua versão é a "0.1". O primeiro argumento possui um único elemento do tipo *String*. Este elemento representa o nome autoverificável do domínio. No caso do EPGS o nome do domínio é "*Intra_Domain*", o que gera o *hash* **15B239D1**. Essa é uma palavra padrão para representar elementos internos a

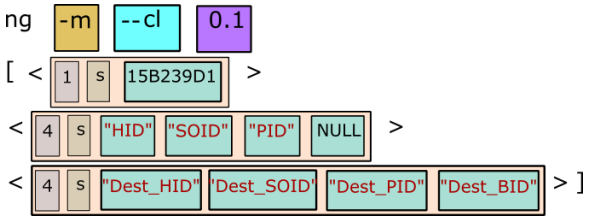

Figura 3.8: Linha de Comando NG de Roteamento.

qualquer domínio. O segundo argumento é composto por quatro elementos do tipo *String*. Os elementos representam o nome autoverificável das seguintes informações do EPGS: identificador do *Hardware*, identificador do Sistema Operacional, identificador do Processo e identificador do Bloco. Na NG, um bloco é definido como uma instância de um objeto que habita um processo. No EPGS não existe bloco, portanto ele é sempre NULL. O último argumento é semelhante ao segundo argumento, mas as informações são referentes ao destino da mensagem. São quatro elementos do tipo *string* representando o nome autoverificável das seguintes informações do Destino: identificador do *Hardware*, identificador do Sistema Operacional, identificador do Processo e identificador do Bloco. Em casos onde o destino ainda não é conhecido (*broadcast*), estas informações serão a *string* "*empty*".

A **linha de comando de tipo de mensagem** possui um nome autoexplicativo e é ilustrada na Figura 3.9.

Figura 3.9: Linha de Comando NG de Tipo de Mensagem.

O nome do comando é "*message*" com alternativa "*type*" e versão "0.1". Possuiu um único argumento composto por um único elemento do tipo *string*. Este elemento especifica o tipo da mensagem.

Este é um exemplo de linha de comando de tipo de mensagem:

```
ng -message --type 0.1 [ < 1 s 1 > ]
```

No exemplo, a mensagem é do tipo "1", que significa Mensagem do Usuário.

A **linha de comando de sequência de mensagem** indica o número que identifica a mensagem. É sempre iniciado em 0 e incrementado a cada mensagem enviada. A Figura 3.10 ilustra esta linha de comando.

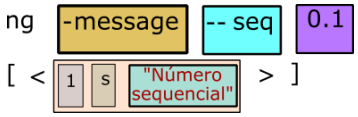

Figura 3.10: Linha de Comando NG de Sequência de Mensagem.

O nome do comando é "*message*" com alternativa "seq" e versão "0.1". Possuiu um único argumento composto por um único elemento do tipo *string*. Este elemento indica o número sequencial da mensagem. Segue um exemplo de linha de comando de sequência:

```
ng -message --seq 0.1 [ < 1 s 17 > ]
```

No exemplo, o identificador desta mensagem é 17. A próxima mensagem a ser enviada, deverá possuir o identificador 18.

A **linha de comando de verificação** é responsável por assinar a mensagem. Ela pode ser utilizada na recepção para garantir a integridade dos dados. A Figura 3.11 representa este tipo de linha de comando.

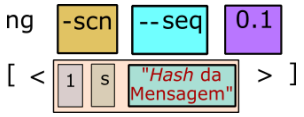

Figura 3.11: Linha de Comando NG de Verificação de Mensagem.

O nome do comando é "scn" com o alternativa é "seq" e versão "0.1". Tem um único argumento, que contém um único elemento do tipo *String*. Ele elemento representa o nome autoverificável (SVN) da mensagem, excluindo a linha de comando de verificação. A carga útil (*payload*) de uma mensagem, caso exista, não é utilizada no cálculo do SVN. Considera-se apenas seu tamanho, substituindo todos os seus dados pela seguinte frase:

```
"There is a payload of <Tamanho dos Dados> bytes"
```

O <Tamanho dos Dados> é substituído pelo número de *bytes* dos dados.

A seguir, um exemplo de linha de comando de verificação:

```
ng -scn --seq 0.1 [ < 1 s 3E9740CA > ]
```

3.6.1 Descoberta - *Hello*

A ação de *Hello* é responsável pela apresentação de um nó a outros pares na rede NovaGenesis. Ele é uma mensagem NovaGenesis composta por linhas de comando que descrevem o nó, expondo qual é sua arquitetura de rede (*stack*), interface e identificador. O *Hello* deve ser enviado para toda a rede (*broadcast*).

A mensagem de *Hello* é construída com a Linha de Comando de Roteamento, seguida pela Linha de Comando de *Hello* e por fim, a Linha de Comando de Verificação. A Figura 3.12 ilustra o formato dessa mensagem.

Figura 3.12: Formato de uma Mensagem NG de *Hello*.

As linhas de comando de Roteamento e Verificação já foram descritas anteriormente. A linha de comando de *Hello* tem a forma mostrada na Figura 3.13.

Seu nome é "*hello*" com alternativa "ihc" (Interhost Communication) e versão "0.2". Possui um único argumento com nove elementos do tipo *string*: os quatro primeiros são os SVNes dos seus identificadores (*hardware*, sistema operacional, processo e bloco), os dois próximos elementos são sempre NULL e os três últimos são o nome da arquitetura de rede, da interface e do identificador. A Figura 3.14 é um exemplo de mensagem de *Hello* completa:

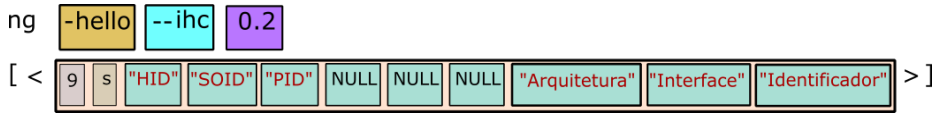

Figura 3.13: Linha de Comando de *Hello*.

```
ng -m --cl 0.1 [ < 1 s 15B239D1 > < 4 s 7E3FFFA6 027552A1 7CE42568 NULL > < 4 s empty empty empty empty > ]
ng -hello --ihc 0.2 [ < 9 s 7E3FFFA6 027552A1 7CE42568 NULL NULL NULL Wifi eth0 00:12:34:AB:CD:EF > ]
ng -scn --seq 0.1 [ < 1 s D3EE629C > ]
```

Figura 3.14: Exemplo de uma Mensagem NG de *Hello*.

Durante o *Hello*, o *Publish/Subscribe Service* (PSS) do núcleo da rede NG não é conhecido, portanto os elementos que representam o PSS na linha de comando de roteamento, serão preenchidos pela *string* "*empty*". Neste exemplo a arquitetura de rede é "*Wi-Fi*", a interface é a "eth0" e o identificador é o MAC "00:12:34:AB:CD:EF".

3.6.2 Exposição de Palavras Chave - *Exposition*

A exposição de palavras-chave é a tarefa responsável por enviar para o PSS NovaGenesis as características que descrevem o dispositivo onde o EPGS está instalado. A exposição para o PSS é feita fazendo a ligação (*binding*) entre a palavra-chave na forma de linguagem natural e sua representação em forma de nome autoverificável. Um exemplo é a exposição de "Termômetro" em linguagem natural e "79EEBDD6" em forma de nome autoverificável. A outra ligação é feita entre o nome autoverificável da palavra-chave e o nome autoverificável do identificador do processo que está expondo a palavra-chave. A Figura 3.15 ilustra este processo.

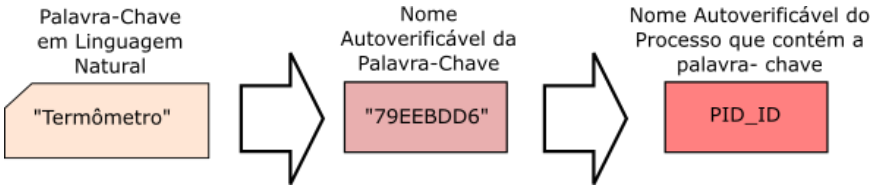

Figura 3.15: Exemplo da Ligação de Nomes.

Desta forma, um cliente que necessite dos serviços do medição de temperatura pode fazer uma busca por "Termômetro" e receberá do PSS os nomes do EPGS que expôs esta funcionalidade. Isto permite que as partes se descubram e estabeleçam um contrato para prestação de serviços.

A mensagem de Exposição (*exposition*) é iniciada pela linha de comando de roteamento, seguida por diversas linhas de comando de publicação de ligação e finalizada pelas linhas de tipo de mensagem, sequência de mensagem e verificação. A Figura 3.16 representa o formato dessa mensagem.

A **linha de publicação de ligação (*publication/bind*)** é ilustrada na Figura 3.17.

O nome do comando é "p" (*publish*) com alternativa "b" (*binding*) e versão "0.1". O primeiro argumento indica a categoria da ligação e é formado somente por um elemento do tipo *string*. O segundo e terceiro argumentos representam os elementos que devem ser ligados. Eles são compostos por um elemento do tipo *string*, que pode ser um nome autoverificável, um nome em linguagem natural ou o identificador de um dos elementos do EPGS. A Figura 3.18 é um exemplo de mensagem de exposição de palavras chave:

Neste exemplo são feitas várias ligações entre os nomes em linguagem natural, os nomes autoverificáveis e os nomes autoverificáveis do identificador do processo: As duas primeiras linhas de publicação de ligação indicam que o nome **1A53F830** representa o texto "EPGS"

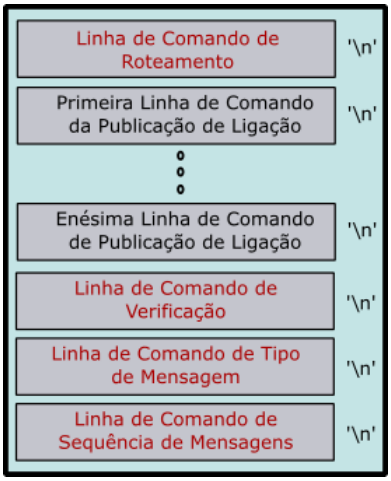

Figura 3.16: Formato de uma Mensagem NG de Exposição de Palavras Chave.

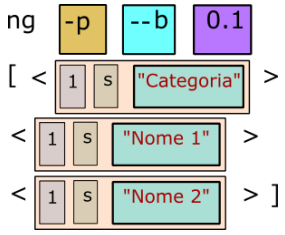

Figura 3.17: Linha de Comando de *Exposition*.

e que está disponível no processo **7CE42568**. As outras duas linhas da sequência fazem a ligação entre o nome **EC417E01** com a palavra "Termômetro" e o identificador do processo **7CE42568**. Note que todas essas palavras-chave estão relacionadas ao processo **7CE42568**, que é o processo do EPGS.

3.6.3 Oferta de Serviço - *Service Offering*

A Oferta de Serviço é o primeiro passo para o estabelecimento do contrato. O EPGS publica a oferta para o *Proxy/Gateway/Controller Service* (PGCS) do núcleo da rede NG, que por sua vez, encaminha a oferta para o outros nós do sistema NG. A mensagem de Oferta de Serviço é composta pelas Linhas de Comandos de Roteamento, de Verificação, de Tipo e de Sequência, pelas das Linhas de Comandos de Publicação da Oferta e de Informação do *Payload* e, finalmente, pelo *payload*. A Figura 3.19 representa o formato dessa mensagem.

A **Linha de Comando de Publicação da Oferta** envia informações relativas ao arquivo que contém os serviços a serem ofertados como demonstrado na Figura 3.17.

O seu nome é "p" com o alternativa "*notify*" e tem versão "0.1". O primeiro argumento possui um único elemento do tipo *string* e representa a categoria da ligação (ver Tabela **??**). O segundo argumento também possui somente um elemento do tipo *string* e representa o nome autoverificável do conteúdo do arquivo que contém os serviços ofertados. O terceiro argumento possui um elemento do tipo *string*, representando o nome do arquivo. E o quarto e último argumento é composto por cinco elementos do tipo *string*. O primeiro elemento é o texto "pub" e informa que é uma publicação. Já os outros quatro argumentos representam o nome autoverificável dos identificadores do *Hardware*, do Sistema Operacional, do Processo e do Bloco do PSS.

A **Linha de Comando de Informação do Arquivo** informa somente o nome do arquivo que contém as informações do serviços ofertados como mostrado na Figura 3.21.

```
ng -m --cl 0.1 [ < 1 s 15B239D1 > < 4 s 7E3FFFA6 027552A1 7CE42568 NULL > < 4 s 0BD95286 ED12F3ED DECC2634 82C5E549 > ]
ng -p --b 0.1 [ < 1 s 2 > < 1 s 1A53F830 > < 1 s 7CE42568 > ]
ng -p --b 0.1 [ < 1 s 1 > < 1 s 1A53F830 > < 1 s EPGS > ]
ng -p --b 0.1 [ < 1 s 2 > < 1 s 0D80E2D6 > < 1 s 7CE42568 > ]
ng -p --b 0.1 [ < 1 s 1 > < 1 s 0D80E2D6 > < 1 s Embedded_Proxy_Gateway_Service > ]
ng -p --b 0.1 [ < 1 s 2 > < 1 s EC417E01 > < 1 s 7CE42568 > ]
ng -p --b 0.1 [ < 1 s 1 > < 1 s EC417E01 > < 1 s Termômetro > ]
ng -p --b 0.1 [ < 1 s 5 > < 1 s 027552A1 > < 1 s 7CE42568 > ]
ng -p --b 0.1 [ < 1 s 6 > < 1 s 7E3FFFA6 > < 1 s 027552A1 > ]
ng -p --b 0.1 [ < 1 s 9 > < 1 s Host > < 1 s 7E3FFFA6 > ]
ng -p --b 0.1 [ < 1 s 2 > < 1 s OS > < 1 s 027552A1 > ]
ng -message --type 0.1 [ < 1 s 1 > ]
ng -message --seq 0.1 [ < 1 s 0 > ]
ng -scn --seq 0.1 [ < 1 s 6B796331 > ]
```

Figura 3.18: Exemplo de uma Mensagem NG de exposição de palavras chave.

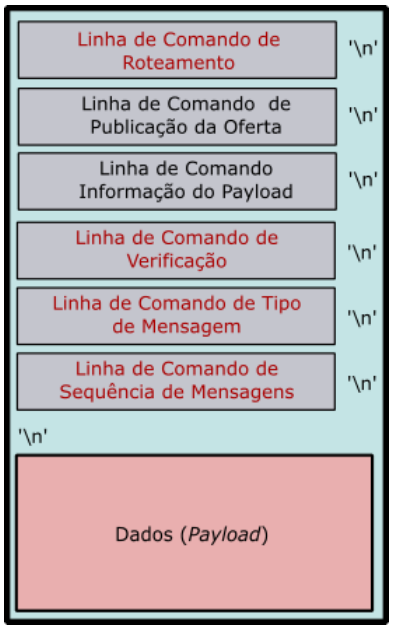

Figura 3.19: Formato de uma Mensagem NG de Oferta de Serviço.

O seu nome é "*info*" com a alternativa "*payload*" e versão "0.1". O seu único argumento é composto por um único elemento do tipo *string* que representa o nome do arquivo.

O *payload* da mensagem é formado pelo conteúdo do arquivo que contém as informações da oferta de serviço. O conteúdo é um texto representando uma mensagem NovaGenesis com linhas de comando que fazem a ligação entre o nome de uma característica do serviço e seu valor.

A Figura 3.22 é um exemplo de mensagem de oferta de serviço:

Neste exemplo, o arquivo "*Service_Offer.txt*" é publicado para o PSS e o PGCS é notificado. O conteúdo do arquivo possui a assinatura **EA0B2B4E** e informa que é ofertado um sensor chamado "EPGS_Sensor" e suas características, como tipo, faixa de operação e resolução.

3.6.4 Negociação de Contrato - *Service Acceptance*

Caso algum cliente se interesse pela oferta de serviço publicada, a negociação de contrato é iniciada. O EPGS é notificado sobre a negociação e recebe a chave do contrato. Para confirmar, o EPGS faz a aceitação do serviço assinando a chave de contrato recebida. A mensagem de **Aceitação de Serviço** é composta pelas Linhas de Comandos de Roteamento, de Verificação, de Tipo e de Sequência, e também pela Linha de Comando de Assinatura da Chave do Contrato. A Figura 3.23 representa o formato desta mensagem.

A **Linha de Comando de Assinatura da Chave** faz a assinatura da ligação do nome

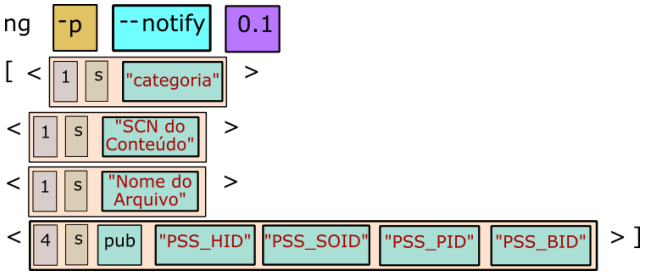

Figura 3.20: Linha de Comando de Oferta de Serviço.

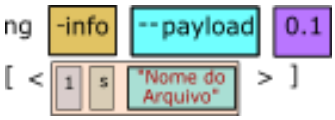

Figura 3.21: Linha de Comando de Informação do Arquivo.

autoverificável da chave de contrato com a categoria como demonstrado na Figura 3.24.

Seu nome é "s" (*subscribe*) e com alternativa "b" (*bind*) e versão "0.1". É composta por um dois argumentos. O primeiro argumento possui um único elemento do tipo *string* que representa a categoria da ligação (Tabela **??**). O segundo argumento também possui um único elemento do tipo *string* que representa o nome autoverificável da chave do contrato.

3.6.5 Publicação de Dados - *Data Publishing*

Após a realização da negociação do contrato, o EPGS está pronto para disponibilizar para seu cliente os dados requisitados. A mensagem de publicação de dados *(data publishing)* é composta pelas Linhas de Comandos de Roteamento, de Verificação, de Tipo e de Sequência, pelas das Linhas de Comandos de Publicação dos Dados e de Informação do *Payload* e, finalmente, pelo *payload*. A Figura 3.25 representa o formato desta mensagem.

A **Linha de Comando de Publicação de Dados** envia informações relativas ao arquivo que contém os dados requisitados. Seu formato é ilustrado na Figura 3.26.

O seu nome é "p" com o alternativa "*notify*" e tem versão "0.1". O primeiro argumento possui um único elemento do tipo *string* e representa a categoria. O segundo argumento também possui somente um elemento do tipo *string* e representa o nome autoverificável do conteúdo do arquivo que contém os dados requisitados. O terceiro argumento possui um elemento do tipo *string*, representando o nome do arquivo. E o quarto e último argumento é composto por cinco elementos do tipo *string*. O primeiro elemento é o texto "pub" e informa que é uma publicação. Já os outros quatro argumentos representam o nome autoverificável dos identificadores do *Hardware*, do Sistema Operacional, do Processo e do Bloco do Cliente dos Dados (assinante).

A **Linha de Comando de Informação do Arquivo** informa somente o nome do arquivo que contém as informações do serviços ofertados. Esta linha já foi detalhada na Seção 3.6.2.

O *payload* da mensagem é formado pelo conteúdo do arquivo que contém os dados requisitados a serem publicados.

A Figura 3.5 é um exemplo de mensagem de publicação de dados. Neste exemplo, o arquivo "*Temperature*.json" é publicado e o aplicativo cliente é notificado. O conteúdo do arquivo possui a assinatura "6153124D" e é formado por um texto representando em JSON a medida de temperatura coletada.

```
ng -m --cl 0.1 [ < 1 s 15B239D1 > < 4 s 7E3FFFA6 027552A1 7CE42568 NULL > < 4 s 0BD95286 ED12F3ED DECC2634 82C5E549 > ]
ng -p --notify 0.1 [ < 1 s 18 > < 1 s EA0B2B4E > < 1 s Service_Offer.txt > < 5 s pub 0BD95286 ED12F3ED CE362E2E E86E7B38 > ]
ng -info --payload 0.1 [ < 1 s Service_Offer.txt > ]
ng -message --type 0.1 [ < 1 s 1 > ]
ng -message --seq 0.1 [ < 1 s 1 > ]
ng -scn --seq 0.1 [ < 1 s 6748556D > ]

ng -sr --b 0.1 [ < 1 s 17 > < 1 s EPGS_Sensor > < 5 s sensorType sensorRangeMin sensorRangeMax sensorResolution > ]
ng -sr --b 0.1 [ < 1 s 17 > < 1 s sensorType > < 1 s Temperature > ]
ng -sr --b 0.1 [ < 1 s 17 > < 1 s sensorRangeMin > < 1 s -20 > ]
ng -sr --b 0.1 [ < 1 s 17 > < 1 s sensorRangeMax > < 1 s 100 > ]
ng -sr --b 0.1 [ < 1 s 17 > < 1 s sensorResolution > < 1 s 0.1 > ]
```

Figura 3.22: Exemplo de uma Mensagem NG de Oferta de Serviço.

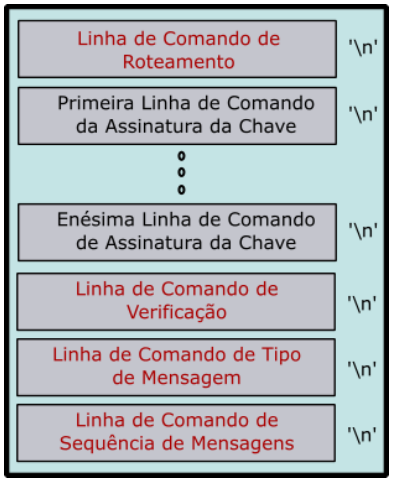

Figura 3.23: Formato de uma Mensagem NG de Aceitação de Oferta de Serviço.

3.7 Ciclo de Vida do EPGS

O ciclo de vida do EPGS é composto pelos seguintes estados: Inicializar, Descobrir, Expor Recursos, Oferecer Serviço, Aceitação de Serviço, Publicar Dados e Finalizar.

O estado Inicializar começa quando uma instância do EPGS é criada. Neste estado, a estrutura de dados do EPGS é alocada e preenchida com as informações de *hardware* e características dos sensores. Também são gerados os nomes autoverificáveis. O próximo estado é o de Descobrir parceiros na rede. Este estado é dividido em duas partes: (i) enviar *Hello* para a rede (*broadcast*) e (ii) receber *Hello* de parceiros. No envio de *Hello*, uma mensagem NovaGenesis como descrita na a Seção 3.6.1 é enviada, informando sua presença aos pares na rede, provendo seus nomes autoverificáveis e informações da camada de enlace. O envio de *Hello* pode ser interrompido a qualquer momento para poupar energia. O recebimento de *Hello* pode ocorrer a qualquer instante do ciclo de vida. Ao receber um *Hello*, o EPGS armazena as informações do parceiro e pode utilizá-las para comunicações futuras. Após o processo de *Hello*, os dados para comunicação com o PGCS já devem ser conhecidos. Com isso, o EPGS pode fazer a exposição dos seus recursos. Este estado consiste no envio de uma mensagem NovaGenesis diretamente para o PGCS contendo ligações entre nomes. As ligações podem ser entre nomes NLN e SVN ou entre SVNes. A mensagem foi descrita na Seção 3.6.2. O próximo estado é oferecer o serviço representado pelo EPGS para possíveis clientes. Para isso, o EPGS faz uma oferta de serviço para o PGCS. O PGCS então pode "vender" este serviço para aplicações clientes interessadas. Este estado consiste no envio de uma mensagem NovaGenesis de publicação de oferta de serviço, como descrita na Seção 3.6.3. Caso o PGCS encontre uma aplicação cliente interessada no serviço, o EPGS receberá o contrato contendo os nomes autoverificáveis da aplicação, assinará e enviará a aceitação. Isso é feito enviando uma mensagem NovaGenesis de aceitação de serviço, como descrita na Seção 3.6.4. Assim termina a fase de negociação de contrato. Com o contrato estabelecido, o EPGS pode começar a publicar os dados providos pelos sensores para a aplicação cliente. Este é o estado de Publicação de Dados e a mensagem é mostrada

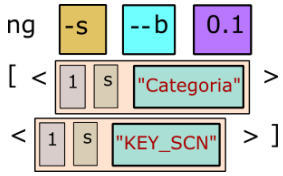

Figura 3.24: Linha de Comando de Aceitação de Oferta de Serviço.

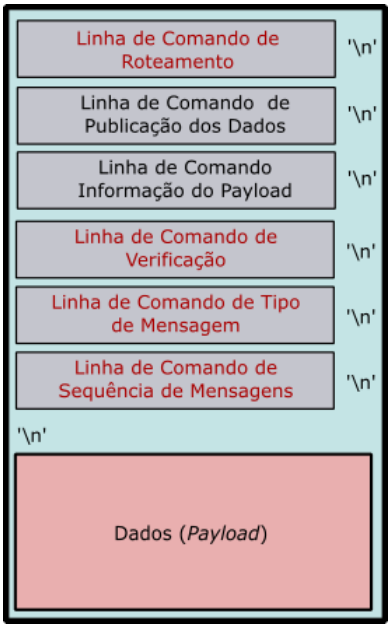

Figura 3.25: Formato da Mensagem NG de Publicação de Dados.

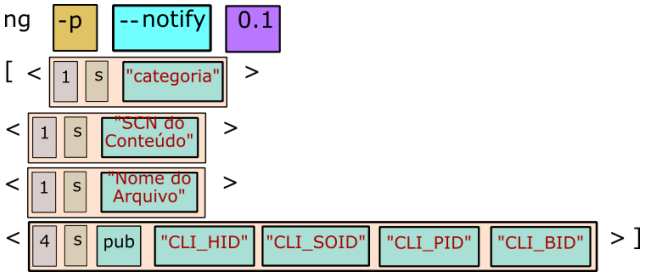

Figura 3.26: Linha de Comando de Publicação de Dados.

na Seção 3.6.5. O último estado é o de Finalização, em que todas as estruturas alocadas para o funcionamento do EPGS são liberadas. Neste ponto, todos os contratos são descartados e o *hardware* contendo o EPGS pode ser desligado com segurança. A Figura 3.27 representa o diagrama de sequência do funcionamento básico do EPGS.

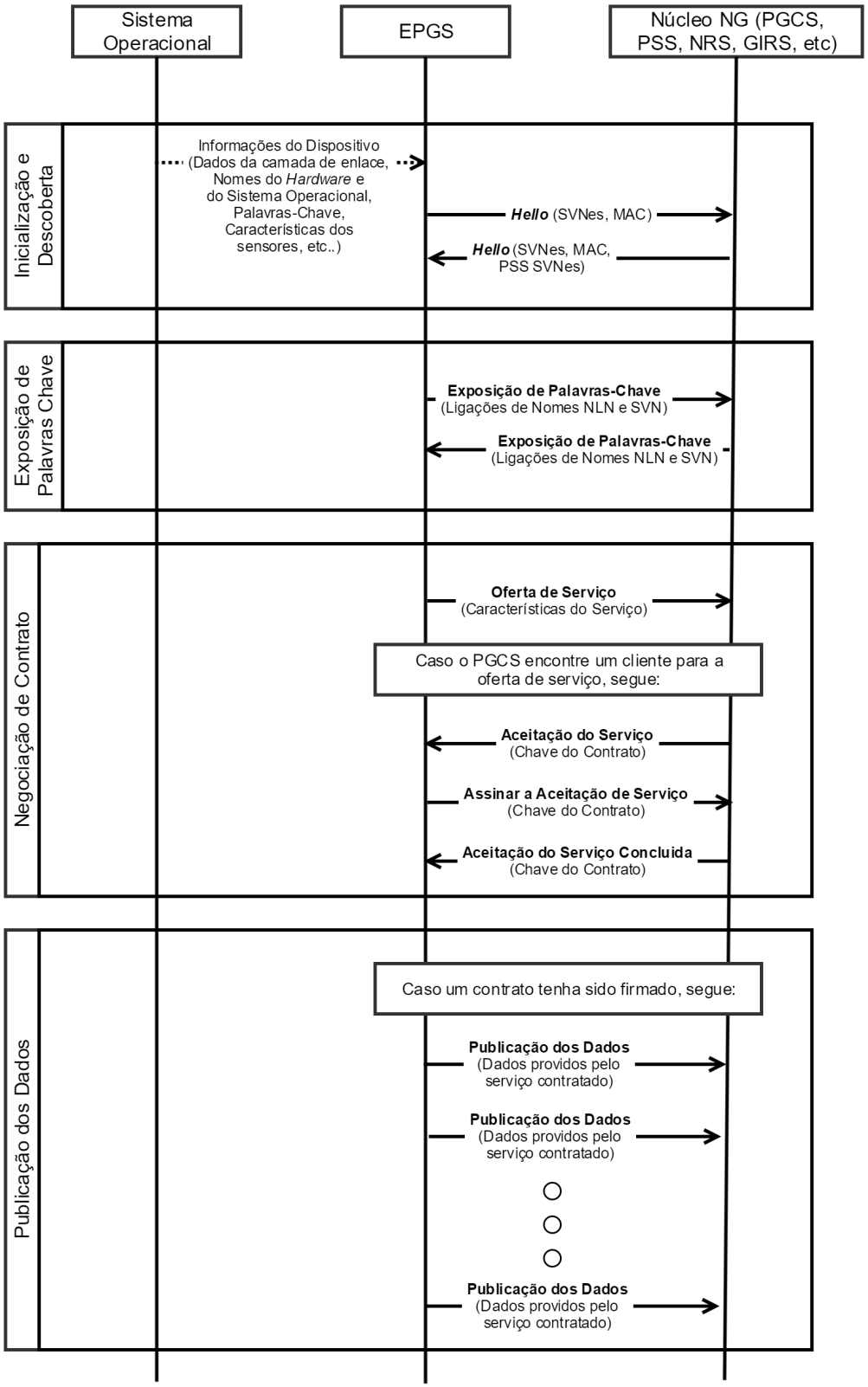

Figura 3.27: Diagrama de Sequência básico do EPGS.

Capítulo 4

Implementação do Serviço NovaGenesis para IoT

O Serviço NovaGenesis para IoT (EPGS) foi desenvolvido utilizando a linguagem de programação ANSI C. Ela foi escolhida por ser a linguagem tradicionalmente mais utilizada e compatível com os *kits* de IoT comercialmente usados.

4.1 Estrutura do Projeto

O projeto EPGS foi organizado por pastas, sendo que cada pasta é um módulo contendo arquivos que tem funções semelhantes. Os arquivos que não pertencem a nenhuma pasta, são os arquivos que podem ser personalizados de acordo com cada *hardware* de destino e também os arquivos que precisam ser incluídos no projeto que utilizará o EPGS. A Figura 4.1 mostra como é a árvore de diretórios do projeto.

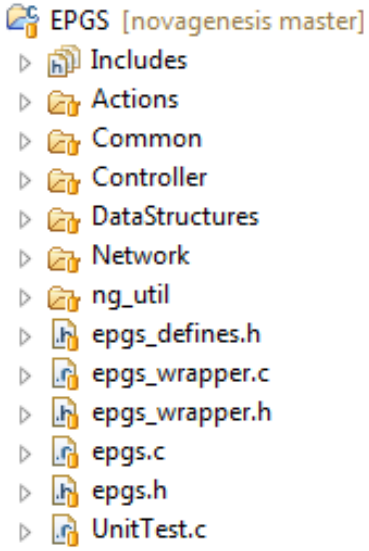

Figura 4.1: Árvore de diretórios da estrutura do projeto EPGS.

4.1.1 Arquivo epgs_defines.h

Responsável por definir algumas constantes utilizadas pelo EPGS, como o nome do domínio (*Intra_Domain*), o nome das arquiteturas de rede (*Ethernet*, Wi-Fi, *Bluetooth*), os dois *bytes* que representam o tipo NovaGenesis no cabeçalho *ethernet* (1234) e códigos de erros. Este arquivo pode ser alterado caso algum desses dados precisem ser atualizados.

4.1.2 Arquivos epgs_wrapper.h e epgs_wrapper.c

O EPGS precisa acessar algumas funcionalidades específicas da arquitetura onde será embarcado, como por exemplo alocar memória. Muitas arquiteturas utilizam o comando "malloc" para este fim. Outras arquiteturas podem alocar memória de outra forma. Para garantir compatibilidade e evitar o trabalho de substituição das chamadas em todo código do EPGS, foi criado um módulo encapsulador, onde os métodos devem ser escritos conforme a arquitetura utilizada. O Quadro 4.1 descreve esses métodos.

Quadro 4.1: Métodos do módulo *Wrapper* que devem ser personalizados.

Método	**Descrição**
ng_rand	Geração de número aleatório.
ng_atoi	Conversão de caracteres que representam números em seu valor inteiro.
ng_calloc, ng_malloc e ng_realloc	Alocação de memória.
ng_free	Liberação de memória.
ng_memcpy	Cópia de dados de uma posição de memória para outra.
ng_strcmp, ng_strcpy, ng_strncpy, ng_strlen, ng_strcat, ng_strspn, ng_strcspn	Manipulação de *strings*.
ng_printf, ng_sprintf	Formatação e impressão de textos.
ng_GetTime	Hora em milissegundos.
ng_EthernetSendData e ng_BLESendData	Envio de dados para as interfaces.

Desta forma, basta que o encapsulador seja modificado, não necessitando substituir em todos os pontos do EPGS.

4.1.3 Arquivos epgs.h e epgs.c

Estes arquivos são responsáveis por implementar a máquina de estados e as funções de configuração do EPGS. O aplicativo que deseja utilizar o EPGS deve incluir este arquivo, executar o método de inicialização e utilizar os métodos para configurar o EPGS. Esses métodos são descritos no Quadro 4.2.

4.1.4 Arquivos UnitTest.c

Este arquivo implementa um aplicativo que simula uma troca de mensagens. Utilizado somente para testes do EPGS. Deve ser removido quando o EPGS for incluído em um projeto real.

4.1.5 Módulo ng_util

Este módulo contém um arquivo com métodos utilizados por todo EPGS. Estão presentes neste módulo uma implementação simples e reduzida do protocolo JSON e também de um método de particionamento de *string*, semelhante ao strtok do ANSI C.

4.1.6 Módulo *Common*

Neste módulo são implementados os métodos responsáveis pela criação e manipulação das Linhas de Comando e Mensagens NovaGenesis e também pela geração de nomes autoverificáveis

Quadro 4.2: Métodos para configuração do EPGS.

Método	Descrição
initEPGS	Inicializa o EPGS.
destroyEPGS	Libera todos os recursos alocados pelo EPGS.
setHwConfigurations	Configuração dos identificadores do *hardware*, sistema operacional e processo do aplicativo que utiliza o EPGS. Além disso, recebe as configurações de rede: arquitetura, interface e identificador.
addKeyWords	Adiciona palavras-chave que representam o aplicativo que utiliza o EPGS.
addHwSensorFeature	Adiciona as especificações e características do sensor disponível.
processLoop	Método que deve ser chamado dentro do *loop* principal do aplicativo. Este método deve ser executado frequentemente.
newEthernetReceivedMessage	Método a ser chamado quando dados forem recebidos via Ethernet.
newBLEReceivedMessage	Método a ser chamado quando dados forem recebidos via *Bluetooth*.
setDataToPub	Especifica os dados a serem publicados.
enablePeriodicHello	Habilita ou desabilita o envio periódico de mensagens de descoberta (*Hello*).

MurMurHash.

4.1.7 Módulo *DataStructures*

O EPGS utiliza várias estruturas (*structs*) para armazenar os dados necessários para a execução das tarefas. O módulo *DataStructures* é responsável pela definição, criação e manipulação dessas estruturas.

4.1.8 Módulo *Actions*

As ações ou tarefas que o EPGS executa são definidas neste módulo e são descritas no Quadro 4.3.

Quadro 4.3: Ações executadas pelo EPGS.

Método	Descrição
epgs_exposition	Ação de criar a mensagem para exposição dos recursos.
epgs_hello	Ação de criar a mensagem de descoberta novos parceiros.
epgs_pubNotify	Ação de criar a mensagem para publicação dados.
epgs_subServiceAcceptance	Ação de criar a mensagem para assinar uma oferta de serviço.

4.1.9 Módulo *Network*

O módulo *network* é responsável por preparar, adaptar e enviar os dados de acordo com a arquitetura de rede. Ele implementa o *Proxy* e *Gateway*.

4.1.10 Módulo *Controller*

O módulo *Controller* é responsável por unir todos os módulos e controlar qual ação será executada. Também é responsável por entender as mensagens recebidas dos parceiros e tomar uma ação, caso necessário.

4.2 Geração de Nomes Autoverificáveis (SVN)

O EPGS utiliza a especificação do MurMurHash3 implementada por Austin Appleby [24] para geração dos nomes autoverificáveis. Esta implementação está no módulo *Common* nos arquivos MurmurHash3.c e MurmurHash3.h.

Para evitar as colisões (mesmo *hash* para entradas diferentes) em dados pequenos, foi implementado um mecanismo de preparação dos dados antes da criação do *hash*. Ele é implementado nos arquivos ng_epgs_hash.c e ng_epgs_hash.h.

Para a geração dos nomes autoverificáveis com redução de colisões, foi criado o seguinte método: GenerateSCNFromCharArrayBinaryPatterns4Bytes. Ele é utilizado em vários pontos do EPGS.

4.3 Linha de Comando NovaGenesis

A criação e manipulação das Linhas de Comando NovaGenesis são feitas no módulo Common, pelos arquivos ng_command.c e ng_command.h. Ela é criada em forma de estrutura de dados (*struct*), e cada membro da estrutura implementa um campo da linha de comando:

```
1  struct _ng_arguments {
2          char** Elements;
3          unsigned int NoE;
4  };
5
6  struct _ng_command {
7          // Command Name
8          char*    Name;  // ng -msg
9
10         // Command Alternative
11         char*    Alternative; // --cl
12
13         // Command Version
14         char     Version[5]; // 0.1
15
16         // Arguments and elements container
17         struct _ng_arguments **Arguments;
18
19         // Number of arguments
20         unsigned int     NoA;
21 };
22 typedef struct _ng_command NgCommand;
23 typedef struct _ng_arguments NgArguments;
```

A manipulação da estrutura é feita através de métodos, que são descritos no Quadro 4.4.

4.4 Mensagem NovaGenesis

Assim como a Linha de Comando, a Mensagem NovaGenesis é implementada no módulo *Common* pelos arquivos ng_message.c e ng_message.h. Uma estrutura de dados (*struct*) implementa a mensagem e cada membro representa uma especificação.

```
1  struct _ng_message {
2          // Type
3          short    Type;
4
5          // CommandLine container
6          NgCommand        **CommandLines;
```

Quadro 4.4: Métodos para manipulação da estrutura de dados da Linha de Comando NG.

Método	Descrição
ng_create_command	Dados o nome, alternativa e versão, cria uma nova linha de comando.
NewArgument	Adiciona um novo argumento a linha de comando.
GetArgument	Retorna o argumento da linha de comando localizado em um dado índice.
GetNumberofArguments	Informa o número total de argumentos.
SetArgumentElement	Adiciona um elemento a um dado argumento da linha de comando.
GetArgumentElement	Obtém o elemento apontado por um dado índice de um dado argumento da linha de comando.
GetNumberofArgumentElements	Informa o número total de elementos de um dado argumento.
ConvertNgCommandToString	Transforma a estrutura de dados em um texto formatado de acordo com as especificações de uma linha de comando NovaGenesis.
ConvertStringToNgCommand	Transforma um texto formatado de acordo com as especificações de linha de comando NovaGenesis em estrutura de dados.

```
7
8           // Number of command lines
9           unsigned int    NoCL;
10
11          // Has payload control flag. True means that the message has a
                payload.
12          bool    HasPayloadFlag;
13
14          // Payload char array. Can be used to carry a payload in memory
                instead of saving it to a file
15          char    *Payload;
16
17          // Message char array. Can be used to carry the message to
                memory instead of using a file
18          char    *Msg;
19
20          // The size of the payload in bytes
21          int     PayloadSize;
22
23          // The size of the message in bytes
24          int     MessageSize;
25
26          // Time for priority queue comparison
27          double  Time;
28  };
29
30  typedef struct _ng_message NgMessage;
```

A manipulação dos dados é feita através dos métodos descritos no Quadro 4.5.

Quadro 4.5: Métodos para manipulação da estrutura de dados da Mensagem NG.

Método	Descrição
ng_create_message	Cria uma nova Mensagem NovaGenesis.
ng_destroy_message	Libera todos os recursos alocados e destrói a Mensagem.
NewCommandLine	Adiciona uma nova Linha de Comando à Mensagem.
GetCommandLine	Obtém a Linha de Comando apontada por um dado índice.
SetPayloadFromCharArray	Adiciona um texto como carga útil da Mensagem.
MessageToString	Transforma a estrutura de dados em um texto formatado de acordo com as especificações de Mensagem NovaGenesis.
MessageFromString	Converte um texto formatado de acordo com as especificações de Mensagem NovaGenesis em uma estrutura de dados.

4.5 Estrutura de Dados NovaGenesis

Após a inicialização do EPGS, uma estrutura de dados é criada para armazenar todas as informações necessárias para a sua execução. Esses dados devem persistir enquanto o EPGS existir.

```
struct _ng_epgs {
        // Descriptors and identifiers
        struct _ng_net_info *MyNetInfo;
        struct _ng_net_info *PGCSInfo;
        struct _ng_scn_id_info *PSSScnIDInfo;
        struct _ng_scn_id_info *APPScnIDInfo;
        struct _ng_hw_descriptor *HwDescriptor;

        // Message Relay - Peer's information
        int PeersNetInfoCount;
        struct _ng_net_info **PeersNetInfo;

        // Information about received messages
        struct _ng_received_msg *ReceivedMsg;

        // State of the EPGS
        States ngState;

        // Enable periodic Hello
        bool enableHello;

        // Counter of messages
        int MessageCounter;

        // Key of the contract
        char* key;

        // Information about the data to publish
        struct _ng_data_to_pub *DataToPub;

        char* pubDataFileName;
        char* pubData;
        int pubDataSize;
};
```

Todos os campos dessa estrutura representam uma configuração ou informação necessária ao funcionamento do EPGS, como por exemplo, o seu estado atual.

O Quadro 4.6 descreve cada um dos campos dessa estrutura.

Foi criado um método para fazer a liberação dos recursos de forma segura quando o EPGS não for mais utilizado. Este método é chamado de destroy_NgEPGS.

4.6 O *Proxy/Gateway*

O módulo *Proxy/Gateway* do EPGS é composto por funções de conversão de endereço, cálculo de fragmentos (dado o tamanho total a ser enviado), tratamento de cabeçalhos, envio, encaminhamento e recebimento de mensagens. A diferença entre o envio e o encaminhamento é a não necessidade do cálculo de fragmentos quando encaminhando. É neste módulo que o tamanho máximo suportado para cada envio, a MTU é configurada.

Para o procedimento de envio ou de encaminhamento de uma mensagem NovaGenesis, o módulo é responsável por converter o endereço MAC de destino recebido em formato texto para *bytes*. A mensagem NovaGenesis é copiada para o *buffer* de Unidade de Dados de Serviço -

Quadro 4.6: Métodos para manipulação da estrutura de dados NovaGenesis.

Método	**Descrição**
MyNetInfo	Informa qual é a arquitetura de rede, o identificador e a interface utilizada pelo EPGS.
PGCSInfo	Informa qual é a arquitetura de rede, o identificador e a interface do PGCS.
PSSScnIDInfo	Nomes autoverificáveis dos identificadores do PSS (Hid, Soid, Pid e Bid).
APPScnIdInfo	Nomes autoverificáveis dos identificadores do aplicativo cliente (Hid, Soid, Pid e Bid).
HwDescriptor	Conjunto de chaves e valores referentes as características do *hardware* e sensores.
PeersNetInfoCount	Número de pares conectados.
PeersNetInfo	Informa qual é a arquitetura de rede, o identificador e a interface de cada um dos pares conectados.
ReceivedMsg	Informa o tamanho e o conteúdo de uma mensagem recebida.
ngState	Armazena o estado atual do EPGS.
enableHello	Habilita ou desabilita o envio periódico de mensagens de descoberta de pares.
MessageCounter	Contador de mensagens enviadas pelo EPGS. Inicializado em zero.
Key	Chave do contrato.
DataToPub	Dados que deverão ser publicados e seu tamanho.

Service Data Unit (SDU) juntamente com o cabeçalho NovaGenesis de 8 *bytes* que representa o tamanho da mensagem. Em seguida é calculado em quantos fragmentos a SDU deve ser particionada para não desrespeitar a MTU. Definidos os números de fragmentos, um *buffer* de Unidade de Dados de Protocolo - *Protocol Data Unit* (PDU) para cada fragmento é criado. No PDU são adicionados os endereços MAC de destino (8 *bytes*), de origem (8 *bytes*) e o tipo (2 *bytes*). No caso de uma mensagem NovaGenesis, definiu-se o tipo com o valor de 4660 ou 0x1234 em hexadecimal. Depois, é adicionado o cabeçalho de fragmentação da NovaGenesis. Ele é composto por 4 *bytes* que especificam o número do fragmento (iniciando em 0) e um número aleatório de 4 *bytes* que identifica a mensagem. Por fim, é adicionado a SDU. Estando com a PDU pronta, ela é enviada e o processo é repetido para cada fragmento. A Figura 4.2a) ilustra o passo a passo do encapsulamento da mensagem NG a ser enviada.

O processo de recebimento consiste primeiramente em remover o cabeçalho *Ethernet* (8 *bytes* do MAC de Destino, 8 *bytes* do MAC de Origem e 2 *bytes* do Tipo). Então é obtido o número do fragmento e o identificador da mensagem. Isto termina a remoção da PDU, restando a SDU para ser analisada. Da SDU é obtido o tamanho da mensagem NovaGenesis a ser recebida. Dado a MTU e o tamanho da mensagem é calculado o número de fragmentos que serão recebidos. Caso tenha somente um fragmento, os dados referentes a mensagem já estão totalmente recebidos e podem ser processados. No caso mais de um fragmento, os dados parciais da mensagem são armazenados em um *buffer* de recebimento e o procedimento de recebimento é feito novamente, removendo o cabeçalho *Ethernet*, obtendo o número do fragmento (uma unidade maior que o anterior) e o identificador da mensagem (igual ao recebido anteriormente). Neste caso a SDU conterá somente os dados da mensagem (sem informação de tamanho). Estes dados são adicionados ao *buffer* de recebimento. Quando todos os fragmentos terminarem de ser recebidos, a mensagem pode ser processada. A Figura 4.2b) ilustra o passo a passo da extração da mensagem NG recebida.

As funções que de fato fazem o envio e recebimento dos dados devem sem implementadas pela aplicação que utilizará o EPGS, expondo estas funções através do módulo *Wrapper*.

4.7 O Tratamento das Mensagens Recebidas

Toda vez que o dispositivo receber um pacote de dados, ele deve notificar o EPGS e encaminhar o pacote atraves da função **newEthernetReceivedMessage**, descrita no arquivo **epgs.h**. Este pacote de dados é analisado e decomposto pelo *Proxy/Gateway*, como mostrado na Figura 4.2b). Com a mensagem NG extraída, o módulo *controller* do EPGS analisa a pri-

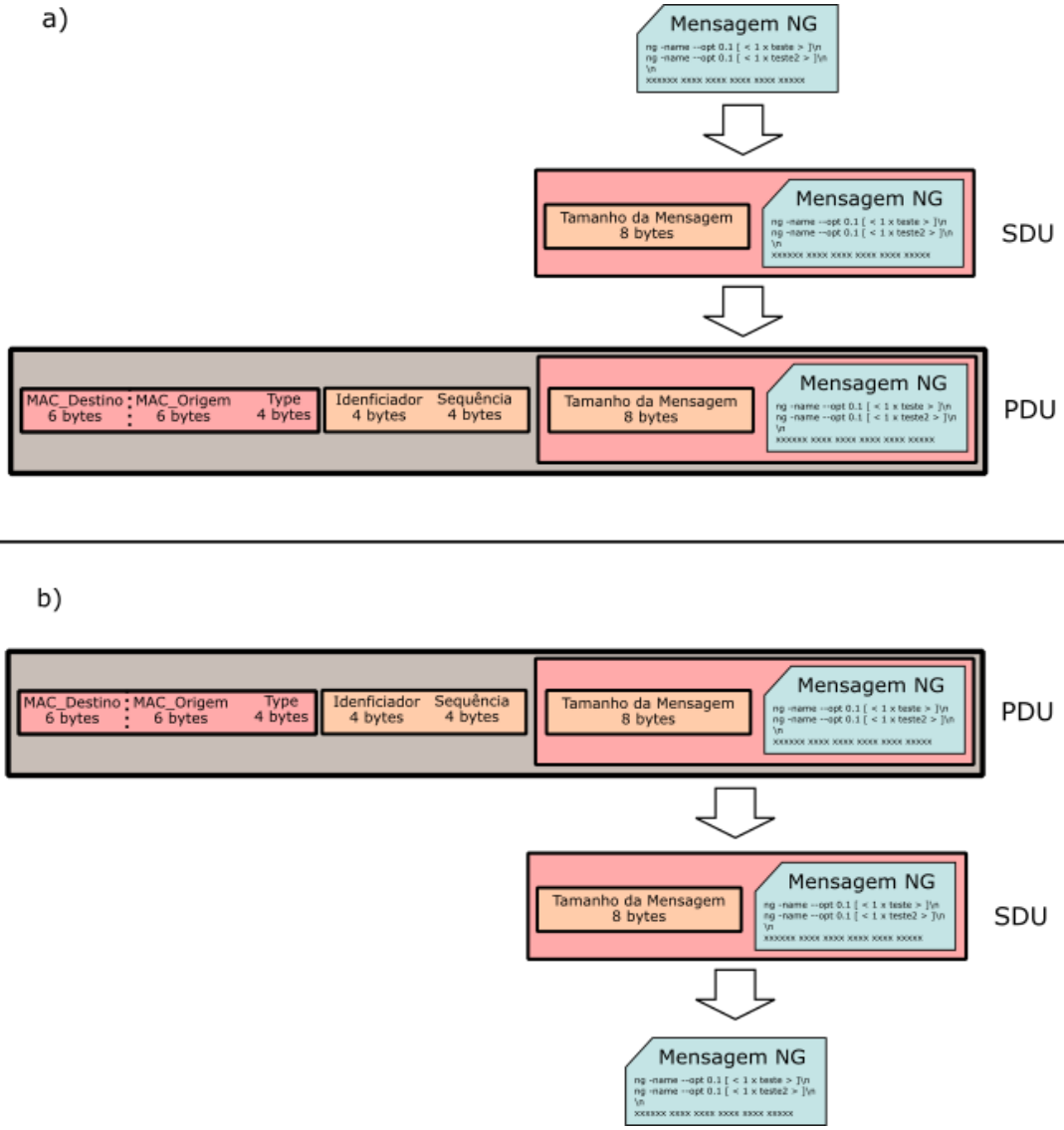

Figura 4.2: Em a) é ilustrado o processo para a preparação da Mensagem NG a ser enviada. Em b) é ilustrado o processo de recebimento e extração da Mensagem NG.

meira linha de comando da mensagem, que é sempre a linha de comando de roteamento. Se a mensagem não for destinada a ele próprio, a mensagem é descartada ou encaminhada (este comportamento é detalhado no Capítulo 5.2). Mas se a mensagem for destinada a ele, o *controller* vai analisar o restante das linhas de comando a fim de tomar a decisão correta.

Capítulo 5

Experimentos

Foram propostos dois cenários experimentais. O primeiro embarca o EPGS em um *hardware* NXP-LPC1769^{TM} conectado com um computador PC via Wi-Fi. Dados de temperatura coletados pelo NXP são enviado utilizando a arquitetura NG para um aplicativo cliente localizado no PC. Este experimento foi publicado em [31] e visa validar a troca de mensagens e coletar métricas de consumo de memória e processamento de forma real. O segundo cenário é a simulação em software de uma rede colaborativa NG, onde um EPGS sem visada para o núcleo da rede NG consegue estabelecer comunicação utilizando EPGSes intermediários como "encaminhadores" de mensagem, formando assim uma rede em malha. Este segundo cenário gerou o artigo publicado em [32].

5.1 Cenário Experimental 1 - Sensor de Temperatura

O objetivo deste cenário é verificar a troca de mensagens NG entre os nós da rede a fim de fazer a descoberta, estabelecimento de contrato e publicação de dados sensoreados. Além disso, também foram coletadas métricas de utilização do dispositivo.

Foram utilizados dois dispositivos NXP-LPC1769^{TM} embarcados com o EPGS sobre o sistema operacional "*Event OS*" [33]. Eles foram conectados ao computador PC com o núcleo da rede NG utilizando um ponto de acesso Wi-Fi. A Figura 5.1 ilustra o cenário experimental.

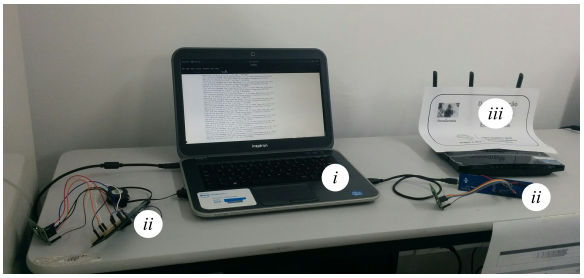

Figura 5.1: Cenário Experimental 1: (*i*) Núcleo NovaGenesis e aplicação cliente IoT (IoTTestApp) rodando no computador; (*ii*) dois EPGSes NovaGenesis rodando em *hardwares* NXP-LPC1769 na direita e na esquerda (cabo USB somente para energia); (*iii*) Ponto de acesso Wi-Fi.

O NXP-LPC1769^{TM} possui um processador ARM Cortex-M3 com 32 kbits de memória SRAM e seu *kit* de testes provê um rádio Wi-Fi e um sensor de temperatura.

No computador PC estão instalados todos os nós necessários para o funcionamento da rede NG, como o PGCS, PSS, HTS e a aplicação cliente IoT (IoTTestApp).

5.1.1 As Trocas de Mensagens

As trocas de mensagens NG demonstram o funcionamento do sistema. Nas Seções seguintes são exibidas algumas mensagens criadas e recebidas pelo EPGS.

Inicialização

Na fase de Inicialização, o EPGS envia uma mensagem de *Hello* para toda a rede (*broadcast*), informando os nomes autoverificáveis (SVNs) dos seus identificadores (*Hardware*, Sistema Operacional, Processo e Bloco) e também os dados da sua camada de enlace (Endereço MAC, Interface e Identificador). Esta mensagem é exibida na Figura 5.2.

```
ng -m --cl 0.1 [ < 1 s 28FD4420 > < 4 s 4C7CF9B2 5F472DA7 1A53F830 NULL > < 4 s empty empty empty empty > ]
ng -hello --ihc 0.2 [ < 9 s 4C7CF9B2 5F472DA7 1A53F830 NULL NULL NULL Wifi wlan0 ac:22:0b:13:01:34 > ]
ng -scn --seq 0.1 [ < 1 s 604007EC > ]
```

Figura 5.2: Mensagem de *Hello* enviada pelo EPGS para toda a rede.

De forma análoga, o PGCS envia uma mensagem de *Hello* para o EPGS informando seus SVNes e dados da camada de enlace. O *Hello* do PGCS ainda contém os SVNes dos identificadores do PSS. Os SVNes do PSS são necessários nas fases seguintes, para publicação de NBs. A mensagem é exibida na Figura 5.3.

```
ng -m --cl 0.1 [ < 1 s 28FD4420 > < 4 s 0BD95286 ED12F3ED 7E764DC1 4D623F20 > < 4 s empty empty empty empty > ]
ng -hello --ihc 0.2 [ < 6 s A4324A2D AB9B70B4 57ECEB4F Wi-Fi wlan0 ac:22:0b:c9:df:3b >
< 4 s 0BD95286 ED12F3ED 8E8B52EC 7EA46815 > ]
ng -scn --seq 0.1 [ < 1 s 1A81A5E3 > ]
```

Figura 5.3: Mensagem de "*hello*" enviada pelo PGCS para toda a rede.

Esta é a fase de Inicialização. Assim o EPGS e o PGCS sabem da existência um do outro e podem começar uma comunicação direcionada.

Exposição de Palavras Chave

O EPGS envia para o PSS suas palavras chave. Este passo é muito importante, pois assim o PGCS é capaz de saber quais as funcionalidades disponibilizadas pelo dispositivo onde o EPGS esta embarcado.

```
ng -m --cl 0.1 [ < 1 s 28FD4420 > < 4 s 4C7CF9B2 5F472DA7 1A53F830 NULL > < 4 s 0BD95286 ED12F3ED 8E8B52EC 7EA46815 > ]
ng -p --b 0.1 [ < 1 s 2 > < 1 s 1A53F830 > < 1 s 7CE42568 > ]
ng -p --b 0.1 [ < 1 s 1 > < 1 s 1A53F830 > < 1 s EPGS > ]
ng -p --b 0.1 [ < 1 s 2 > < 1 s 0D80E2D6 > < 1 s 7CE42568 > ]
ng -p --b 0.1 [ < 1 s 1 > < 1 s 0D80E2D6 > < 1 s Embedded_Proxy_Gateway_Service > ]
ng -p --b 0.1 [ < 1 s 2 > < 1 s EC417E01 > < 1 s 7CE42568 > ]
ng -p --b 0.1 [ < 1 s 1 > < 1 s EC417E01 > < 1 s Termômetro > ]
ng -p --b 0.1 [ < 1 s 5 > < 1 s 027552A1 > < 1 s 7CE42568 > ]
ng -p --b 0.1 [ < 1 s 6 > < 1 s 7E3FFFA6 > < 1 s 027552A1 > ]
ng -p --b 0.1 [ < 1 s 9 > < 1 s Host > < 1 s 7E3FFFA6 > ]
ng -p --b 0.1 [ < 1 s 2 > < 1 s OS > < 1 s 027552A1 > ]
ng -message --type 0.1 [ < 1 s 1 > ]
ng -message --seq 0.1 [ < 1 s 0 > ]
ng -scn --seq 0.1 [ < 1 s 6B796331 > ]
```

Figura 5.4: Mensagem de Exposição de Palavras Chave enviada pelo EPGS para o PSS.

A Figura 5.4 ilustra a mensagem de exposição de recursos do EPGS. As palavras chaves em Linguagem Natural expostas foram "EPGS", "*Embedded_Proxy_Gateway_Service*" e "Termômetro". Desta forma, possíveis clientes descobrir que este dispositivo é um EPGS ou Termômetro.

Negociação de Contrato

Após expor suas palavras chaves (que representam suas funcionalidades e características), o EPGS pode ofertar seus serviços à possíveis clientes interessados. Isto é feito publicando uma mensagem de oferta de serviço para o PSS, com notificação para o PGCS. A notificação é necessária, pois é o PGCS que é responsável por "vender" o serviço. Esta mensagem é mostrada pela Figura 5.5.

```
ng -m --cl 0.1 [ < 1 s 28FD4420 > < 4 s 4C7CF9B2 5F472DA7 1A53F830 NULL > < 4 s 0BD95286 ED12F3ED 8E8B52EC 7EA46815 > ]
ng -p --notify 0.1 [ < 1 s 18 > < 1 s EA0B2B4E > < 1 s Service_Offer.txt > < 5 s pub 0BD95286 ED12F3ED 7E764DC1 4D623F20 > ]
ng -info --payload 0.1 [ < 1 s Service_Offer.txt > ]
ng -message --type 0.1 [ < 1 s 1 > ]
ng -message --seq 0.1 [ < 1 s 1 > ]
ng -scn --seq 0.1 [ < 1 s 6748556D > ]

ng -sr --b 0.1 [ < 1 s 17 > < 1 s EPGS_Sensor > < 5 s sensorType sensorRangeMin sensorRangeMax sensorAccuracy > ]
ng -sr --b 0.1 [ < 1 s 17 > < 1 s sensorType > < 1 s Temperature > ]
ng -sr --b 0.1 [ < 1 s 17 > < 1 s sensorRangeMin > < 1 s -20 > ]
ng -sr --b 0.1 [ < 1 s 17 > < 1 s sensorRangeMax > < 1 s 100 > ]
ng -sr --b 0.1 [ < 1 s 17 > < 1 s sensorAccuracy > < 1 s 0.2 > ]
```

Figura 5.5: Mensagem de Oferta de Serviço enviada pelo EPGS para o PSS, notificando o PGCS.

Ao receber a oferta, o PGCS publica uma mensagem de aceitação de serviço para o EPGS. O EPGS recebe esta mensagem e confirma o recebimento assinando a aceitação de serviço. Por fim, o PGCS entrega para o EPGS os SVNes da aplicação cliente interessada no serviço, enviando uma mensagem de assinatura de serviço, como mostrada na Figura 5.6.

```
ng -m --cl 0.1 [ < 1 s 15B239D1 > < 4 s 0BD95286 ED12F3ED CE362E2E 4479BCFE > < 4 s 7E3FFFA6 027552A1 7CE42568 NULL > ]
ng -d --b 0.1 [ < 1 s 18 > < 1 s 12BC34CD > < 1 s Service_Acceptance.txt > ]
ng -info --payload 0.1 [ < 1 s Service_Acceptance.txt > ]
ng -d --b 0.1 [ < 1 s 2 > < 1 s 12BC34CD > < 3 s ED12F3ED A1C31B34 BEAC1234 > ]
ng -d --b 0.1 [ < 1 s 9 > < 1 s 12BC34CD > < 1 s 0BD95286 > ]
ng -message --type 0.1 [ < 1 s 1 > ]
ng -scn --ack 0.1 [ < 2 s EADFC501 1D293F1B > ]
ng -scn --ack 0.1 [ < 2 s D2F5B7E1 7865E5D4 > ]

0BD95286 ED12F3ED A1C31B34 BEAC1234
```

Figura 5.6: Mensagem de assinatura da aceitação de serviço enviada pelo PGCS para o EPGS.

Note que o *payload* desta mensagem é composto pelos nome autoverificáveis da aplicação cliente interessada.

Publicação dos Dados Sensoreados

Depois do contrato firmado, o EPGS começa a publicar os dados referentes as medidas de temperatura. Uma mensagem é enviada para o PSS com notificação para a aplicação cliente. Esta mensagem publica um arquivo chamado "Measures.json", sendo que o conteúdo deste arquivo é enviado no *payload* e é um texto em formato JSON com a informação da medida de temperatura coletada. A mensagem é mostrada na Figura 5.7 e na Figura 5.8 é apresentado um gráfico com as medidas feitas pelo experimento durante aproximadamente 189,6 horas. Foram coletadas 1.296.000 amostras.

```
ng -m --cl 0.1 [ < 1 s 28FD4420 > < 4 s 4C7CF9B2 5F472DA7 1A53F830 NULL > < 4 s 0BD95286 ED12F3ED 8E8B52EC 7EA46815 > ]
ng -p --notify 0.1 [ < 1 s 18 > < 1 s 6153124D > < 1 s Temperature.json > < 5 s pub 0BD95286 ED12F3ED A1C31B34 BEAC1234 > ]
ng -info --payload 0.1 [ < 1 s Measures.json > ]
ng -message --type 0.1 [ < 1 s 1 > ]
ng -message --seq 0.1 [ < 1 s 2 > ]
ng -scn --seq 0.1 [ < 1 s D109E40E > ]

{ Temperature:32 }
```

Figura 5.7: Mensagem de publicação dos dados sensoreado enviado pelo EPGS para o PSS.

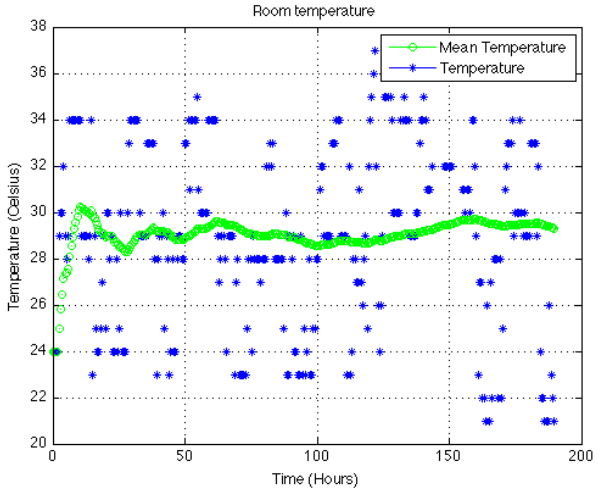

Figura 5.8: Medidas de temperatura feitas por aproximadamente 189,6 horas.

5.1.2 Métricas de Consumo de Memória e Processamento

Neste experimento foram coletadas algumas métricas de consumo de memória e também de uso do processador quando o EPGS está rodando sobre o sistema operacional "EventOS" no dispositivo "NXP LPC1769". O EPGS ocupou 42.440 *bytes* de memória ROM e para implementar a pilha NG completa, o EPGS consumiu 8.720 *bytes* de memória RAM. Comparado com o os dados obtidos por Baccelli et al. [34], a implementação da NG para IoT (ou seja, o EPGS) requer menor uso de memória do que algumas outras arquiteturas. A Tabela 5.1 mostra todas as métricas de consumo de memória que foram coletadas.

Tabela 5.1: Comparação da utilização de memória entre a NG e o CCN-Lite.

	Memory	
NovaGenesis	Complete ROM (bytes)	42,440
	Standalone ROM (bytes)	25,328
	Complete RAM (bytes)	8,720
	Standalone RAM (bytes)	2,108
CCN-Lite	ROM (bytes)	16,628
	RAM (bytes)	5,112

O uso do processador é medido indiretamente através da contagem de ciclo de máquina necessário para cada ação. Por exemplo, a publicação dos dados sensoreados utilizou 2.491.954 ciclos de máquina. Não é possível garantir que a implementação para IoT da NG é mais rápida pois a comparação com outras arquiteturas é inviável, já que cada uma tem uma característica e função diferente. A Tabela 5.2 mostra as métricas de utilização da capacidade computacional de alguns métodos das duas arquiteturas em comparação.

5.1.3 Resultados Práticos

Para verificar os pacotes de dados transmitidos entre o EPGS e o PGCS, foi utilizada a ferramenta chamada WireShark™. Na Figura 5.9 um pacote de publicação de dados sensoreados é enviado pelo EPGS para o PGCS. Note que foi utilizado um filtro pelo tipo **0x1234**, que é o tipo definido para pacotes NG.

Tabela 5.2: Comparação da utilização de CPU entre a NG e o CCN-Lite.

	CPU cycles	
NovaGenesis	RunHello	1,898,091
	RunExposition	3,833,896
	RunServiceOffer	3,375,442
	RunServiceAcceptance	2,348,335
	PubData	2,491,954
CCN-Lite	memcmp_ssse3	14,002,814
	ccnl_nonce_find_or_append	7,525,050
	ccnl_i_prefixof_c	4,062,659
	dehead	1,462,304
	ccnlcoreRXiorc	956,238
	ccnl_extract_prefix_nonce_ppkd	895,590
	memcpy_ssse3	845,042

5.2 Cenário Experimental 2 - Simulação de Rede Colaborativa

Um outro cenário testado foi a implementação de uma rede colaborativa para dispositivos IoT. O objetivo é utilizar a capacidade de roteamento baseado em nomes da NG para que um dispositivo receba mensagens e a encaminhe para o próximo dispositivo, caso esta mensagem não for destinada a ele. Com isso é possível expandir o alcance da rede, sem aumentar a potência de transmissão dos dispositivos e consequentemente consumindo menos energia.

5.2.1 Roteamento Baseado em Nomes Autoverificáveis

O roteamento de mensagens da NG é baseado nos nomes autoverificáveis do destinatário. A tupla formada pelos SVNes dos identificadores do *hardware*, sistema operacional, processo e bloco é utilizada para unicamente identificar o destinatário. Quando um nó é introduzido na rede NG, ele obrigatoriamente deve enviar uma mensagem de *Hello*, como exemplificada abaixo:

```
ng -hello --ihc 0.2 [ < 9 s 4C7CF9B2 5F472DA7 1A53F830 NULL NULL NULL Wifi wlan0 ac:22:0b:13:01:34 > ]
```

Os nós que receberem esta mensagem de *Hello* armazenam nas suas Tabelas *Hash* - *Hash Tables* (HTs) a tupla de identificadores (4C7CF9B2 5F472DA7 1A53F830 NULL) e fazem a ligação com os dados da camada de enlace (Wifi wlan0 ac:22:0b:13:01:34) deste parceiro. Desta forma é possível determinar que a entidade definida pela tupla de SVNes "4C7CF9B2 5F472DA7 1A53F830 NULL" está localizada no endereço MAC ac:22:0b:13:01:34, do identificador wlan0 de uma interface Wi-Fi.

Conhecendo os parceiros, um EPGS é capaz de determinar o destinatário da mensagem analisando a linha de comando de roteamento (detalhada no Capítulo 3.6). Abaixo tem-se um exemplo de linha de comando de roteamento:

```
ng -m --cl 0.1 [ < 1 s 28FD4420 > < 4 s 4C7CF9B2 5F472DA7 1A53F830 NULL > < 4 s 0BD95286 ED12F3ED 8E8B52EC 7EA46815 > ]
```

A tupla do último argumento (0BD95286 ED12F3ED 8E8B52EC 7EA46815) determina o destinatário da mensagem. O EPGS analisa esta tupla e toma as seguintes decisões:

1. **Consome a Mensagem**: Se a tupla for a do próprio EPGS, a mensagem é consumida e as ações necessárias são tomadas.
2. **Encaminha a Mensagem**: Se a tupla não for a do próprio EPGS, ele procura na sua HT. Se encontrar, a mensagem é encaminhada para o endereço da camada de enlace do destinatário correto.

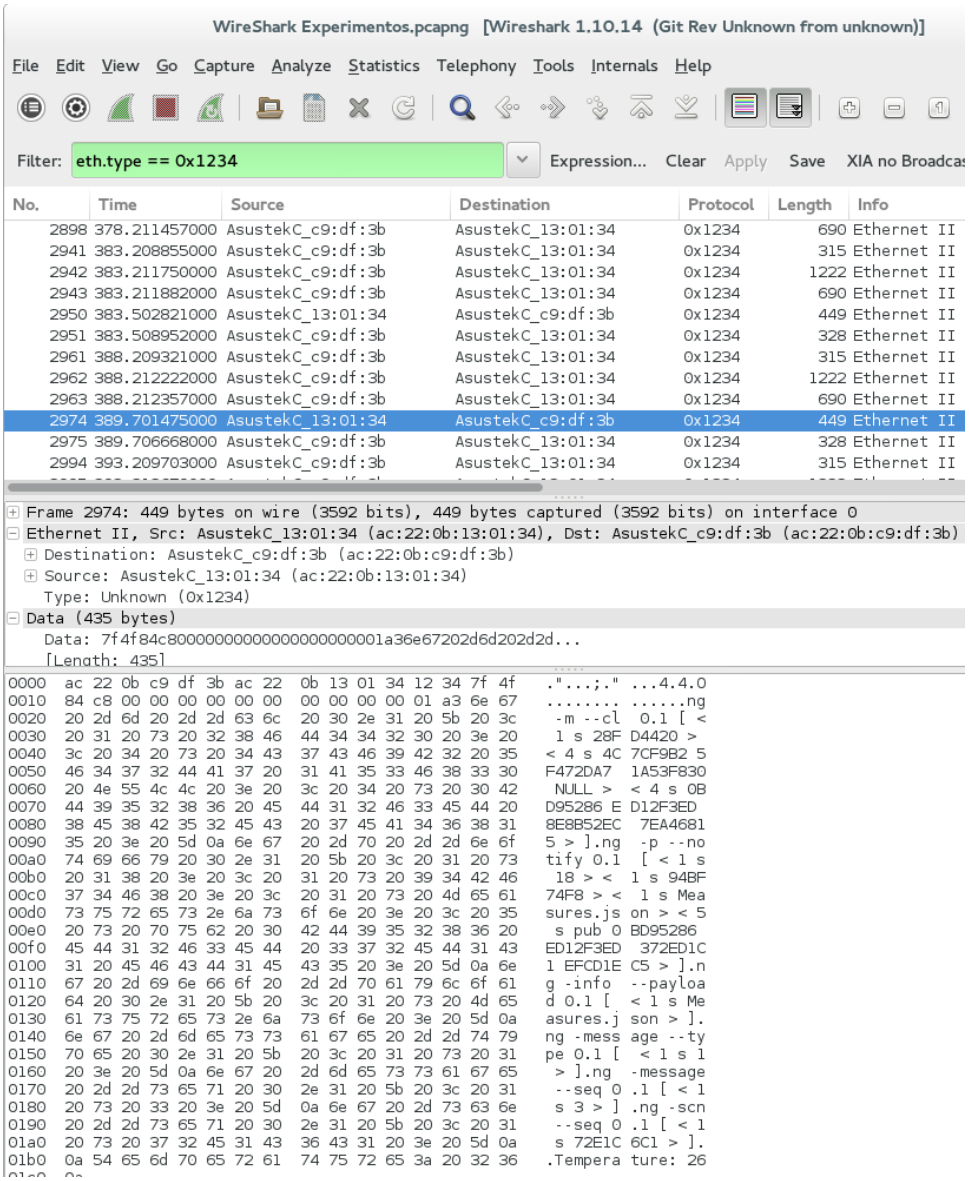

Figura 5.9: Dados reais coletados com a ferramente WireShark™.

3. **Descarta a Mensagem**: Se a tupla não for a do próprio EPGS e também não for encontrada na HT, o EPGS descarta a mensagem.

5.2.2 O Experimento

O ambiente experimental foi todo simulado através de um aplicativo desenvolvido em linguagem C que cria duas instâncias de EPGSes e também simula o PGCS do núcleo da rede. O cenário simulado é exemplificado pela Figura 5.10.

- **Computador Principal - *Main Computer* (MC)** - Contém o núcleo NG (PGCS e PSS) e também a aplicação cliente das medidas, chamada de IoTTestApp. Seu endereço MAC é AC:22:0B:C9:DF:3B e a tupla que identifica o PSS é "0BD95286 ED12F3ED DECC2634 82C5E549".
- **EPGS principal - *Main EPGS* (M-EPGS)** - Dispositivo IoT com sensor de temperatura, com EPGS e comunicação Wi-Fi. O seu endereço MAC é 00:12:34:AB:CD:EF.

Figura 5.10: Cenário do Experimento simulado.

- **EPGS parceiro - *Peer EPGS* (P-EPGS)** - Dispositivo IoT com controlador de acesso (número e o sexo das pessoas que acessam uma sala) e EPGS para NG sobre Wi-Fi. Seu endereço MAC é A1:B2:C3:D4:E5:F6.

Para simular a cooperação, o P-EPGS foi posicionado sem visada para o MC e, portanto, sem comunicação direta. O M-EPGS foi posicionado para conseguir estabelecer comunicação tanto com o MC quanto com o P-EPGS.

O teste é iniciado com base no cenário mostrado no Capítulo 5.1. O M-EPGS e o MC fazem as trocas de mensagens de todas as fases, até a publicação da temperatura medida pelo M-EPGS. A novidade é a introdução do P-EPGS. Quando ele entra na rede, envia sua mensagem de *Hello* para toda rede. Como o M-EPGS está no alcance, ele recebe o *Hello*, armazena as informações do P-EPGS e envia sua própria mensagem de *Hello*, contendo os dados do PSS no último argumento, pois o M-EPGS já o conhece. O *Hello* é mostrado na Figura 5.11a, destacando o endereço MAC de destino como FF:FF:FF:FF:FF:FF (endereço de *broadcast*). Para expor suas palavras-chave para o PSS (que está no MC), o P-EPGS precisa enviar sua mensagem, com destino ao PSS através do endereço MAC do M-EPGS (00:12:34:AB:CD:EF). A Figura 5.11b mostra esta mensagem, destacando o endereço MAC de destino (que é do M-EPGS) e a tupla do PSS. Ao receber esta mensagem, o M-EPGS verifica que ela deve ser encaminhada para o MC. A Figura 5.11c mostra o encaminhamento da mensagem, alterando somente o endereço MAC de destino, agora apontando para o MAC do MC.

a)
```
FF FF FF FF FF FF A1 B2 C3 D4 E5 F6 12 34 15 25 47 F2 00 00 00 00 00 00 00 00 00 00 DE
ng -m --cl 0.1 [ < 1 s 15B239D1 > < 4 s ABCD5286 ED12F3ED CE362E2E NULL > < 4 s empty empty empty empty > ]
ng -hello --ihc 0.1 [ < 6 s NULL NULL NULL Wifi eth0 a1:b2:c3:d4:e5:f6 > ]
ng -scn --seq 0.1 [ < 1 s 4B00DED6 > ]
```

b)
```
00 12 34 AB CD EF A1 B2 C3 D4 E5 F6 12 34 14 36 44 B2 00 00 00 00 00 00 00 00 00 01 C0
ng -m --cl 0.1 [ < 1 s 15B239D1 > < 4 s 7E3FFFA6 027552A1 7CE42568 NULL > < 4 s 0BD95286 ED12F3ED DECC2634 82C5E549 > ]
ng -p --notify 0.1 [ < 1 s 18 > < 1 s 876BC23A > < 1 s Room_1_Access.json > < 5 s pub APP_HID APP_OSID APP_PID APP_BID > ]
ng -info --payload 0.1 [ < 1 s Room_1_Access.json > ]
ng -message --type 0.1 [ < 1 s 1 > ]
ng -message --seq 0.1 [ < 1 s 15 > ]
ng -scn --seq 0.1 [ < 1 s 8B616B6D > ]

{ Male:1220, Female:456, Total:1676 }
```

c)
```
AC 22 0B C9 DF 3B 00 12 34 AB CD EF 12 34 51 3D 51 3D 00 00 00 00 00 00 00 00 00 01 C0
ng -m --cl 0.1 [ < 1 s 15B239D1 > < 4 s 7E3FFFA6 027552A1 7CE42568 NULL > < 4 s 0BD95286 ED12F3ED DECC2634 82C5E549 > ]
ng -p --notify 0.1 [ < 1 s 18 > < 1 s 876BC23A > < 1 s Room_1_Access.json > < 5 s pub APP_HID APP_OSID APP_PID APP_BID > ]
ng -info --payload 0.1 [ < 1 s Room_1_Access.json > ]
ng -message --type 0.1 [ < 1 s 1 > ]
ng -message --seq 0.1 [ < 1 s 15 > ]
ng -scn --seq 0.1 [ < 1 s 8B616B6D > ]

{ Male:1220, Female:456, Total:1676 }
```

MAC de destino SVNes de destino

Figura 5.11: a) Mensagem de *Hello* enviada pelo P-EPGS para toda rede (*broadcast*); b) Mensagem de publicação de dados enviada pelo P-EPGS para o M-EPGS, mas com o PSS como destino; c) Mensagem de publicação de dados do P-EPGS encaminhada pelo M-EPGS para o PSS.

O diagrama de sequência simplificado deste experimento é mostrado na Figura 5.12. Durante a fase de Identificação e Descoberta, os elementos da rede enviam suas mensagens de *Hello* e armazenam as informações dos pares em suas HTs. Quando o P-EPGS necessita enviar uma mensagem para o MC, ele consulta em sua HT e obtém os dados da camada de enlace do M-

EPGS. Assim ele envia a mensagem para o M-EPGS, mas com a tupla de SVNes do PSS. Ao receber, o M-EPGS identifica que a mensagem tem como destinatário o PSS. Então ele buscar em sua HT os dados da camada de enlace do PSS e encaminha a mensagem.

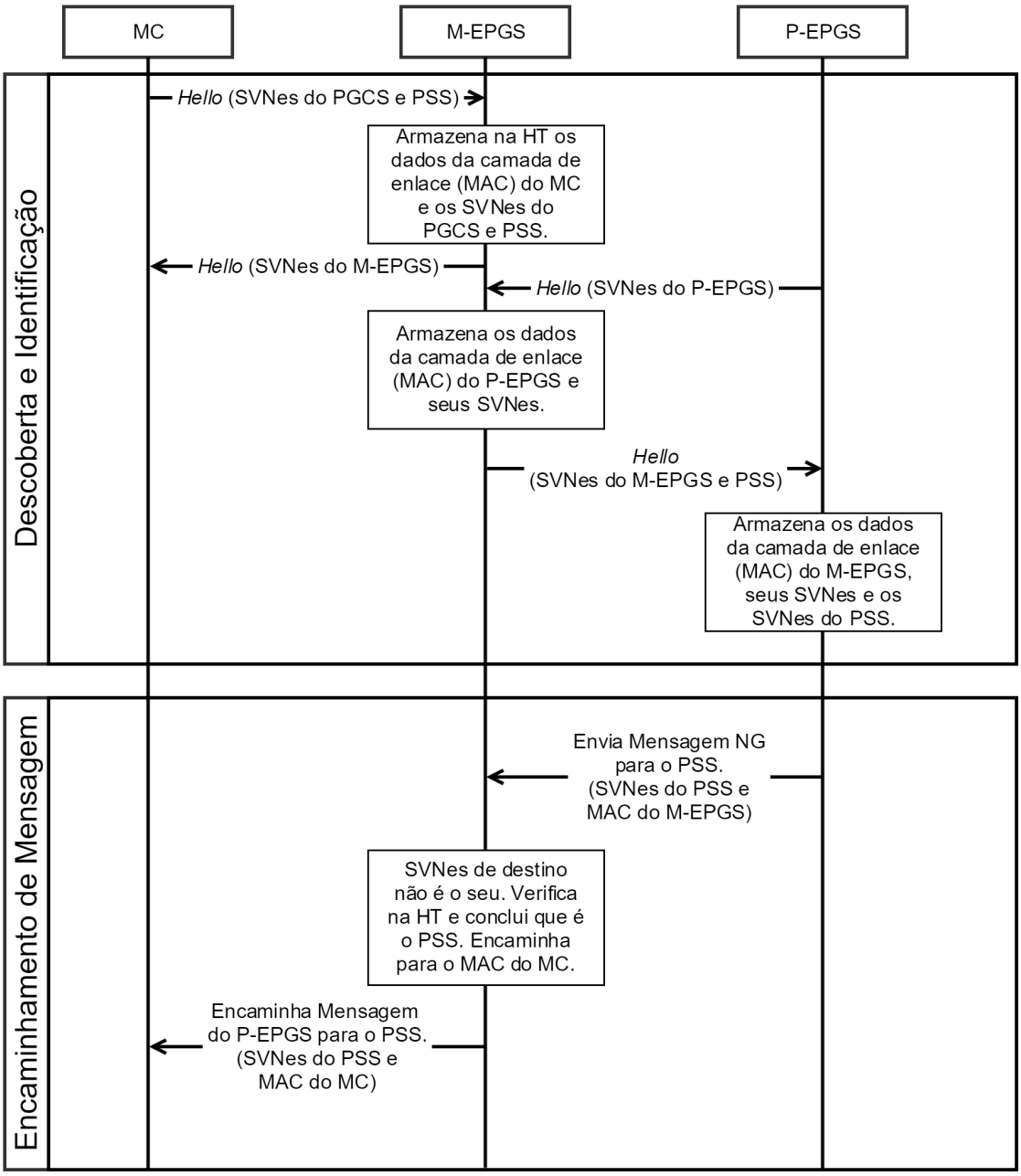

Figura 5.12: Diagrama de sequência das principais mensagens NG envolvidas na Rede Colaborativa.

Com este experimento demonstramos que a utilização da arquitetura de FI NovaGenesis pode ser aplicada para construção de uma rede de dispositivos IoT, onde eles podem colaborar uns com os outros para aumentar a cobertura da rede sem aumentar a potência de transmissão e consequentemente economizando bateria.

Capítulo 6

Conclusão

Esta dissertação teve como objetivo aplicar os conceitos da arquitetura de Internet do Futuro NovaGenesis no mundo IoT. Este conceito foi chamado de Internet das Coisas do Futuro - *Future Internet of Things* (FIoT) [31] e foi alcançado através do desenvolvimento de um novo serviço chamado EPGS que é capaz de ser embarcado em dispositivos com capacidade restrita de memória, processamento e energia. O EPGS foi desenvolvido em linguagem de programação C por ser muito utilizada para *software* embarcado. Métodos que têm dependência do sistema operacional foram encapsulados permitindo uma adaptação simples e rápida. Os algorítimos foram modificados para reduzir significativamente a quantidade de memória ROM utilizada mas manter todas as características e funcionalidades que o PGCS possui. A utilização de memória RAM também foi otimizada, para manter alocado somente o que realmente está sendo utilizado no momento.

Com a criação do EPGS, foram aplicadas as principais tecnologias de rede de computadores atuais em sistemas IoT. Dados são processados e armazenados segundo uma Rede Centrada na Informação - *Information-centric Networking* (ICN), o que provê escalabilidade e eficiência na distribuição das informações para os serviços. Com a integração de ICN e Arquitetura Orientada a Serviço - *Service-Oriented Architecture* (SOA), a descoberta e a contratação dos serviços é feita através de roteamento baseado em nomes e identificação perene das entidades. A proveniência das fontes e integridade dos dados é garantida pela utilização de SVN. Todas as entidades da arquitetura podem ser nomeadas em linguagem natural e SVN e armazenadas em um espaço para nomes ilimitado. A independência de Identificador e Localizador permite que uma entidade possa ser encontrada mesmo em cenários com mobilidade dos dispositivos. As outras arquiteturas de FI com foco em IoT normalmente não integram todos esses paradigmas.

Como contribuição para o estado da arte, os dispositivos IoT utilizando o EPGS podem ser configurados de acordo com um contrato. Um serviço pode expor as capacidades da "coisa" para outros serviços permitindo assim, a configuração dos parâmetros e funcionalidades definida por serviço. Este novo paradigma foi chamado de Arquitetura Definida por Serviço - *Service-Defined Architecture* (SDA).

Dois cenários foram testados. No primeiro foi demonstrada a troca de mensagens entre o dispositivo IoT e o núcleo da rede NG, e a exposição, negociação e contratação do serviço de medida de temperatura e a publicação dos dados sensoreados para a aplicação cliente. A integridade dos dados é garantida pela Linha de Comando de Verificação. Nesse primeiro teste foram coletadas métricas de utilização de memória e processamento e os resultados. O comparativo da utilização da NG embarcada com outras arquiteturas tipicamente utilizadas em IoT, provou que ela é superior quando se trata do consumo de memória e tem um bom desempenho em termos de necessidade processamento. No segundo cenário, foram feitos testes para validar a capacidade de roteamento baseado em nomes. Foi provado que a utilização de EPGSes intermediários possibilita o aumento do alcance de uma rede de dispositivos IoT sem a necessidade de aumentar a potência de transmissão e consequentemente economizando energia.

Os testes confirmaram a aplicabilidade da NG como uma arquitetura viável de Internet das

Coisas do Futuro (FIoT) utilizando o serviço EPGS embarcado nos dispositivos.

6.1 Limitações Conhecidas e Melhorias Necessárias

O EPGS foi testado em ambiente controlado e não prevê algumas situações que devem ser tratadas em ambiente real ou de produção. Ele possui algumas limitações conhecidas e alguns pontos necessários ao bom funcionamento estão pendentes de implementação.

6.1.1 Limitações Conhecidas

Estas são algumas limitações encontradas:

- Tratamento de Desconexão - O EPGS não possui nenhum mecanismo que permite perceber desconexões de parceiros.
- Recisão de Contrato - Atualmente o EPGS permite a criação de um único contrato. O contrato só é quebrado quando o sistema é desligado.
- Reenvio de Mensagens - O EPGS não consegue reenviar mensagens previamente enviadas.
- Tratamento de Fragmentos recebidos fora de ordem.
- Tratamento de Fragmentos de mensagens recebidas simultaneamente.

Estas limitações não impediram a execução dos experimentos realizados e por isso não foram corrigidas. Elas podem ser corrigidas em trabalhos futuros.

6.1.2 Melhorias e Novas Funcionalidades Sugeridas

Algumas melhorias e funcionalidades são sugeridas para a utilização confiável do EPGS e também para a expansão do seu escopo:

- Implementação de outras arquiteturas de camada de enlace no *Proxy/Gateway*, como por exemplo: *Bluetooth Low Energy* (BLE), LORA, 802.15.4.
- Recebimento de configurações: Tratar Mensagens NG que solicitem alteração de configuração do *Hardware*.
- Verificar a integridade da mensagem analisando a linha de comando de verificação (*hash* da mensagem).

6.2 Sugestões de Aplicação Reais e para Trabalhos Futuros

O EPGS pode ser utilizado para uma infinidade de aplicações, embarcado em *hardwares* dedicados ou até mesmo em *smartphones*. Com isso, o EPGS pode ser utilizado em uma rede de sensores diversos, provendo dados a clientes interessados e também pode ser embarcado em *Smartphones* e contratar serviços, por exemplo de armazenamento de fotos.

6.2.1 Sugestões para Aplicações Reais

Aplicativo de Fotos Android

Com o EPGS embarcado em um *smartphone Android*, criar uma aplicação que tire fotos e solicite via NG um serviço para seu armazenamento.

Iluminação de Área

Embarcar o EPGS em um dispositivo com um atuador capaz de acender a luz de um poste. Uma aplicação *desktop* pode contratar os serviços destes atuadores e enviar comandos para acender ou apagar os postes remotamente. Além disso, utilizando a capacidade de criação de rede colaborativa (ver Capítulo 5.2), um dispositivo pode encaminhar as mensagens de outros, sem necessidade visada direta e economizando energia.

6.2.2 Sugestões para Testes

Todos os testes do EPGS aconteceram em ambiente muito bem controlado e com isso muitas situações reais não são validadas. Por isso, sugiro fazer testes em larga escala e em campo.

Laboratório de Processamento de Dados

Apesar de não ser em campo e nem em dispositivos restritos, um bom teste seria embarcar EPGSes em todos os computadores com o sistema operacional Linux disponíveis em um laboratório e comunicar via Ethernet. Para isso, seria necessário criar um *raw socket* que seria usado pelo EPGS para transmissão das mensagens e um gerador de métricas (temperatura, umidade, luminosidade, etc) simuladas. Este teste validaria o funcionamento do EPGS em larga escala.

Inatel *Smart Campus*

Após validar o teste em escala, um outro testes seria embarcar o EPGS nos dispositivos do *Smart Campus* do Inatel. Desta forma, a NG para IoT seria testada em ambiente real de produção. Seria possível coletar métricas disponibilizadas pelos sensores dos dispositivos e também atuar ativando as luzes dos postes.

Referências Bibliográficas

[1] B. Ahlgren, P. Aranda, P. Chemouil, S. Oueslati, L. Correia, H. Karl, M. Söllner, and A. Welin, "Content, connectivity, and cloud: ingredients for the network of the future," *Communications Magazine, IEEE*, vol. 49, no. 7, pp. 62–70, July 2011.

[2] T. Leighton, "Improving performance on the internet," *Commun. ACM*, vol. 52, no. 2, pp. 44–51, Feb. 2009.

[3] A. M. Alberti, "A conceptual-driven survey on future internet requirements, technologies, and challenges," *Journal of the Brazilian Computer Society*, vol. 19, no. 3, pp. 291–311, 2013.

[4] A. M. Alberti, R. C. Brandão, A. Vaz, and B. M. Martins, "Internet of information and services (iois)," in *Convergence and Hybrid Information Technology*, ser. Communications in Computer and Information Science. Springer Berlin Heidelberg, 2012, vol. 310, pp. 53–60.

[5] C. Partridge, "Helping a future internet architecture mature," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 1, pp. 50–52, Dec. 2013.

[6] H. Iqbal and T. Znati, "Overcoming failures: Fault-tolerance and logical centralization in clean-slate network management," in *INFOCOM, 2010 Proceedings IEEE*, march 2010, pp. 1 –5.

[7] S. Paul, J. Pan, and R. Jain, "Architectures for the future networks and the next generation internet: A survey," *Comput. Commun.*, vol. 34, pp. 2–42, January 2011.

[8] G. group, "Geni design principles," *Computer*, vol. 39, no. 9, pp. 102 –105, sept. 2006.

[9] P. Stuckmann and R. Zimmermann, "European research on future internet design," *Wireless Communications, IEEE*, vol. 16, no. 5, pp. 14 –22, october 2009.

[10] F. Ramparany, F. G. Marquez, J. Soriano, and T. Elsaleh, "Handling smart environment devices, data and services at the semantic level with the fi-ware core platform," in *2014 IEEE International Conference on Big Data (Big Data)*, Oct 2014, pp. 14–20.

[11] C. Sarkar, A. U. Akshay, R. V. Prasad, A. Rahim, R. Neisse, and G. Baldini, "DIAT: A scalable distributed architecture for IoT," in *IEEE Internet of Things Journal*, vol. 2, no. 3, jun 2015, pp. 230–239.

[12] J. Soldatos, N. Kefalakis, M. Hauswirth, M. Serrano, J. P. Calbimonte, M. Riahi, K. Aberer, P. P. Jayaraman, A. Zaslavsky, I. P. Žarko, L. Skorin-Kapov, and R. Herzog, *OpenIoT: Open source internet-of-things in the cloud*. Cham: Springer International Publishing, 2015, vol. 9001, pp. 13–25.

[13] D. Han, A. Anand, F. Dogar, B. Li, H. Lim, M. Machado, A. Mukundan, W. Wu, A. Akella, D. G. Andersen, J. W. Byers, S. Seshan, and P. Steenkiste, "Xia: Efficient support for evolvable internetworking," in *Proc. of the 9th USENIX Conf. on Networked Systems Design and Implementation*, ser. NSDI'12. USENIX, 2012, pp. 23–23.

[14] C. Dannewitz, D. Kutscher, B. Ohlman, S. Farrell, B. Ahlgren, and H. Karl, "Network of information (netinf) - an information-centric networking architecture," *Comput. Commun.*, vol. 36, no. 7, pp. 721–735, Apr. 2013.

[15] L. Sun, Y. Li, and R. A. Memon, "An open iot framework based on microservices architecture," *China Communications*, vol. 14, no. 2, pp. 154–162, February 2017.

[16] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[17] E. Nordström, D. Shue, P. Gopalan, R. Kiefer, M. Arye, S. Ko, J. Rexford, and M. J. Freedman, "Serval: An end-host stack for service-centric networking," in *NSDI*, S. D. Gribble and D. Katabi, Eds. USENIX Association, 2012, pp. 85–98.

[18] S. Vrijders, D. Staessens, D. Colle, F. Salvestrini, E. Grasa, M. Tarzan, and L. Bergesio, "Prototyping the recursive internet architecture: the irati project approach," *IEEE Net.*, vol. 28, no. 2, pp. 20–25, March 2014.

[19] S. Sun, M. Kadoch, L. Gong, and B. Rong, "Integrating network function virtualization with sdr and sdn for 4g/5g networks," *IEEE Network*, vol. 29, no. 3, pp. 54–59, May 2015.

[20] A. M. Alberti, M. A. F. Casaroli, D. Singh, and R. da Rosa Righi, "Naming and name resolution in the future internet: Introducing the novagenesis approach," *Future Generation Computer Systems*, vol. 67, pp. 163 – 179, 2017.

[21] A. Ghodsi, T. Koponen, J. Rajahalme, P. Sarolahti, and S. Shenker, "Naming in content-oriented architectures," in *Proceedings of the ACM SIGCOMM Workshop on Information-centric Networking*, ser. ICN '11. New York, NY, USA: ACM, 2011, pp. 1–6.

[22] W. Ramirez, X. Masip-Bruin, M. Yannuzzi, R. Serral-Gracia, A. Martinez, and M. Siddiqui, "A survey and taxonomy of id/locator split architectures," *Computer Networks*, vol. 60, pp. 13 – 33, 2014.

[23] A. Alberti, V. de O Fernandes, M. Casaroli, L. de Oliveira, F. Pedroso, and D. Singh, "A novagenesis proxy/gateway/controller for openflow software defined networks," in *Network and Service Management (CNSM), 2014 10th International Conference on*, Nov 2014, pp. 394–399.

[24] A. Appleby. (2016) Murmurhash3. [Online]. Available: https://github.com/aappleby/smhasher/wiki/MurmurHash3

[25] J. Saltzer, *Name Binding in Computer Systems*, 1977.

[26] J. F. Shoch, "Inter-Network Naming, Addressing and Routing," in *17th IEEE Conference on Computer Communication Networks*, 1978.

[27] J. H. Saltzer, "Naming and binding of objects." in *Advanced Course: Operating Systems*, ser. Lecture Notes in Computer Science, M. J. Flynn, J. Gray, A. K. Jones, K. Lagally, H. Opderbeck, G. J. Popek, B. Randell, J. H. Saltzer, and H.-R. Wiehle, Eds., vol. 60. Springer, 1978, pp. 99–208.

[28] B. M. Hauzeur, "A Model for Naming, Addressing and Routing," *ACM Trans. Inf. Syst.*, vol. 4, no. 4, pp. 293–311, 1986.

[29] W. Chun, T.-H. Lee, and T. Choi, "Yanail: yet another definition on names, addresses, identifiers, and locators," in *Proceedings of the 6th International Conference on Future Internet Technologies*, ser. CFI '11. New York, NY, USA: ACM, 2011, pp. 8–12.

[30] A. M. Alberti, D. Mazzer, M. M. Bontempo, L. H. de Oliveira, R. da Rosa Righi, and A. C. Sodré Jr., "Cognitive radio in the context of internet of things using a novel future internet architecture called novagenesis," *Computers & Electrical Engineering*, pp. –, 2016.

[31] A. M. Alberti, G. D. Scarpioni, V. J. Magalhaes, S. A. Cerqueira, J. J. P. C. Rodrigues, and R. da R. Righi, "Advancing novagenesis architecture towards future internet of things," *IEEE Internet of Things Journal*, vol. PP, no. 99, pp. 1–1, 2017.

[32] V. Magalhães, G. Scarpioni, and A. M. Alberti, "Rede IoT colaborativa NovaGenesis," in *XXXV Simpósio Brasileiro de Telecomunicações e Processamento de Sinais (SBrT 2017) (SBrT 2017)*, São Pedro, Brazil, Sep. 2017.

[33] E. Frigieri, G. Scarpioni, and M. Mokarzel, "Eventos." [Online]. Available: https://github.com/edielsonpf/EventOS

[34] E. Baccelli, C. Mehlis, O. Hahm, T. C. Schmidt, and M. Wählisch, "Information centric networking in the iot: Experiments with ndn in the wild," in *Proc. of the 1st ACM Conf. on Information-Centric Networking*, ser. ACM-ICN '14. New York, NY, USA: ACM, 2014, pp. 77–86.