Dissertação de Mestrado

Inatel
Instituto Nacional de Telecomunicações

Avaliação de Desempenho de Mecanismos de Nomeação e de Distribuição de Conteúdos na Arquitetura NovaGenesis

ÉLCIO CARLOS DO ROSÁRIO

ABRIL / 2021

AVALIAÇÃO DE DESEMPENHO DE MECANISMOS DE NOMEAÇÃO E DE DISTRIBUIÇÃO DE CONTEÚDOS NA ARQUITETURA NOVAGENESIS

ÉLCIO CARLOS DO ROSÁRIO

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
2021

Rosário, Élcio Carlos do
R822a
Avaliação de Desempenho de Mecanismos de Nomeação e de Distribuição de Conteúdos na Arquitetura NovaGenesis. Élcio Carlos do Rosário. – Santa Rita do Sapucaí, 2021.
201p.

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

1. Contêiner 2. Docker 3. NovaGenesis 4. Resolução de Nomes 5. Ambiente de Experimentação 6. Mestrado em Telecomunicações. I. Alberti, Antônio Marcos. II. Instituto Nacional de Telecomunicações – INATEL. III. Título.

CDU 621.39

Ficha Catalográfica elaborada pelo Bibliotecário da Instituição
CRB6-2718 – Paulo Eduardo de Almeida

FOLHA DE APROVAÇÃO

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

Prof. Dr. Antônio Marcos Alberti
INATEL

Prof. Dr. Breno Gontijo Tavares
INATEL

Prof. Dr. José Augusto Suruagy Monteiro
UFPE

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

“A mente que se abre a uma nova ideia jamais voltará ao seu tamanho original”

Albert Einstein

Aos meus pais,
meus primeiros grandes professores.

Agradecimentos

Agradeço primeiramente a Deus pelo dom da vida e bênçãos que me deu todos os dias.

Agradeço aos meus pais José Américo do Rosário e Maria Aparecida de Oliveira que dedicaram suas vidas por mim.

Aos meus irmãos e irmãs que mesmo muitas vezes distantes me apoiaram.

À minha querida esposa Victória Carvalhaes Pinto Rosário que sempre me apoiou, esteve ao meu lado nos bons e maus momentos e cuidou de nosso filho com maestria.

Ao meu querido filho Mateus Carvalhaes Pinto de Oliveira Rosário, que é meu presente de Deus.

Aos familiares de minha esposa que ajudaram a cuidar do meu filho durante todo o curso de mestrado.

Agradeço ao Professor Dr. Antônio Marcos Alberti pelos ensinamentos, pela dedicada orientação, confiança e paciência frente aos meus desafios.

À secretaria de pós-graduação e em especial à Gisele Moreira dos Santos, pelo carinho e dedicação do seu tempo em todos os momentos que precisei.

Aos meus padrinhos Ademir Oliveira da Costa e Mary Therezinha Gonçalves da Costa pelo acolhimento, hospedagem na sua casa, pelo apoio e carinho em todos os dias que estava em Santa Rita do Sapucaí. Agradeço também à Sabrina Costa, filha dos meus padrinhos.

À minha amiga e querida Pedagoga Cristina Nascimento de Oliveira por sempre apoiar minhas decisões no SENAC (Serviço Nacional de Aprendizagem Comercial) e na vida.

Aos meus amigos (as) do ICT-LAB, que considero como irmãos de muito aprendizado, trabalhos em equipe e momentos de descontração. Em especial Jorge Roberto Carneiro, Victor Hugo Domingues D'avila, José Rodrigo dos Santos, Tibério Tavares Rezende, Everton Morais, Epper Bonomo, Thiago Bueno, Luiz Felipe Fernandes de Almeida e Monalisa Conceição Silva.

Aos amigos dos laboratórios parceiros do ICT-LAB. Em especial Elvira Salvador Diogo, Rita de Cassia Duarte Leles, Mauro Alexandre Amaro da Cruz, Vitor Alexandre Campos Figueiredo, Diego Jeldú Cuba Zúñiga, Flávia Larisse da Silva Fernandes e Eligário Milton da Costa Semedo.

Por fim, agradeço ao SENAC, INATEL e ICT-Lab por tornar possível a realização deste sonho.

Élcio Carlos do Rosário

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AAAIF	Ambiente para Avaliação de Arquiteturas de Internet do Futuro
AAIF	Avaliação de Arquiteturas de Internet do Futuro
AD	*Autonomous Domain*
API	*Aplication Programming Interface*
BSD	*Berkeley Software Distribution*
CaR	*Content-Aware Router*
CC	*Content Consumer*
CCN	*Content Centric Network*
CID	*Content Identifier*
CloNe	*Cloud Networking*
CP	*Content Provider*
CPU	*Central Processing Unit*
CR	*Contet Router*
CRC	*Content Resolution Controller*
CS	*Content Store*
CURLING	Content-Ubiquitous Resolution and Delivery Infraestructure for Next-Generation Services
DAG	*Directed Acyclic Graph*
DHT	*Distributed Hash Table*
DIF	*Distributed IPC Facility*
DNS	*Domain Name System*
DONA	*Data-Oriented Network Architecture*
DTN	*Delay Tolerant Network*
FIA	*Future Internet Architecture*
FIB	*Forwarding Information Table*
FITS	*Future Internet Testbed with Security*
FNS	*Flash Network Slides*
FQDN	*Fully Qualified Domain Name*
GENI	*Global Environment for Networking Innovation*
GIRS	*Generic Indirection Resolution Service*
GPS	*Global Positioning System*
GRNS	*Global Resolution Name Service*
GSTAR	*Generalized Storage-aware Routing*
GUID	*Globally Unique Identifier*
GW	*Gateway*
HID	*Host Identifier*

HT	*Hash Table*
HTSe	*Hash Table Service*
ICN	*Information Centric Networking*
ID	Identificador
IDD	*Inter-DIF Directory*
ihc	*interhost communication*
INATEL	Instituto Nacional de Telecomunicações
IoT	*Internet of Things*
IP	*Internet Protocol*
IPC	*Inter-Process Communication*
IPCP	*IPC Process*
IR	*Indirection Resolution*
KVM	*Kernel-based Virtual Machine*
LOC	Localizador
LXC	*LinuX Containers*
MAC	*Media Address Code*
MD5	*Message Digest Algorithm 5*
MF	*Mobility First*
MQTT	*Message Queuing Telemetry Transport*
NA	*Network Address*
NB	*Name Binding*
NCS	*Name Certification Service*
NDO	*Name Data Object*
NetInf	*Network Information*
NG	NovaGenesis
NLNe	*Natural Language Name*
NRNCS	*Name Resolution and Network Caching Service*
NRS	*Name Resolution Service*
NSF	*National Science Foundation*
OCI	*Open Container Initiative*
OConS	*Open Connectivity Service*
OpenVZ	*Open Virtuozzo*
ORBIT	*Open-access Research Testbed for Next-Generation Wireless Networks*
OSID	*Operating System Identifier*
PaaS	*Plataform-as-a-Service*
PG	*Proxy Gateway*
PGCS	*Proxy/Gateway/Controller Service*
PID	*Process Identifier*
PIT	*Pending Interest Table*
PS	*Publish/Subscribe*
PSIRP	*Publish/Subscribe Internet Routing Paradigm*
PSS	*Publish/Subscribe Service*
PURSUIT	*Publish/Subscribe Internet Tecnology*
RAM	*Random-Access Memory*
RIB	*Rina Information-Base*
RINA	*Recursive InterNetwork Architeture*

RTT	*Round Trip Time*
SAIL	*Scalable and Adaptive Internet Solutions*
SDN	*Software-Defined Network*
SID	*Service Identifier*
SIP	*Session Initiation Protocol*
SLA	*Service Level Agreement*
SO	Sistema Operacional
SVNe	*Self-verifying Name*
TCP	*Transmission Control Protocol*
TI	Tecnologia da Informação
UnionFS	*Union File System*
URI	*Uniform Resourse Identifier*
URL	*Uniform Resource Locator*
VM	*Virtual Machine*
XIA	*eXpressive Internet Architeture*
XID	*eXpressive Identifier*
XIP	*eXpressive Internet Protocol*

Lista de Símbolos

μ	*- Representa o tempo médio*
x	*- Representa o tempo instantâneo*
N	*- Representa o número de eventos*
σ	*- Representa o desvio padrão*
σ_μ	*- Representa o erro padrão a partir da média* μ
z	*- Distância padrão da média* μ
$\bar{x}$	*- Intervalo de confiança ou margem de erro*
μ_{low}	*- Representa a média menos a margem de erro*
IC	*- Representa a margem de erro*

Resumo

Sabe-se que devido ao uso massivo de informações e busca de conteúdos na Internet, vários desafios têm surgido. Disponibilidade de endereços para dispositivos fixos ou móveis, segurança, suporte à mobilidade sem perda de comunicação, suporte à conectividade múltipla, acesso a conteúdos pelo nome, são alguns dos desafios da Internet atual. Com o intuito de resolver esses e outros desafios, várias propostas de arquiteturas de Internet do Futuro foram criadas. As novas arquiteturas trabalham em conjunto com a arquitetura atual, ou poderão até mesmo substitui-las algum dia. Nesse contexto, o presente trabalho se propõe a realizar uma avaliação de desempenho da arquitetura NovaGenesis (NG), uma arquitetura de Internet do Futuro criada do zero, utilizando serviços de distribuição de conteúdos (mensagens e fotos) com uso de contêineres *Docker*, estudar o escalonamento da arquitetura NG em atender os seus serviços. Uma prova de conceito da NG em diferentes ambientes é avaliada e os resultados analisados comparativamente. Além da transferência de conteúdos com armazenamento temporário, o presente trabalho se propõe a realizar avaliações de desempenho de resolução de nomes em um domínio local. O desempenho da NG é avaliado em ambiente físico, máquina virtual e contêiner. Concluímos que o *Docker* é uma ferramenta importante para avaliar novas arquiteturas, como a NovaGenesis, mas que pode ser aplicada a outras arquiteturas. Também concluímos que a resolução de nomes e a troca de conteúdos nomeados em um domínio tem um desempenho satisfatório, embora melhorias no protótipo tenham o potencial de melhorar consideravelmente os resultados. O uso de contêineres como alternativa às máquinas virtuais é crescente em ambientes de provedores de serviço de Nuvem e essa tendência também é observada em experimentos científicos conduzidos pela comunidade de pesquisa, pois apresenta maior versatilidade, otimização de tempo e menor consumo computacional quando comparado a outras técnicas de virtualização. Os desenvolvedores estão adotando a prática de uso de contêineres para a programação de microsserviços. Dentre os vários benefícios, trazem redução de sobrecarga de *hardware*, velocidade na inicialização de aplicações, facilidade no escalonamento e balanceamento de carga entre os contêineres. O crescimento dessa tecnologia é creditado às facilidades oferecidas pela plataforma *Docker*, que facilitou a criação e execução de contêineres. Apesar das inerentes vantagens, o uso de contêineres em pesquisa científica na área de redes de comunicações, mais especificamente na avaliação de desempenho de novas arquiteturas, permanece uma importante lacuna.

***Palavras-Chave*:** Contêiner, Docker, NovaGenesis, Resolução de nomes, Armazenamento temporário, Troca de conteúdos nomeados, Avaliação, Ambiente de Experimentação, Arquitetura de Internet do futuro

Abstract

We know that due to the massive use of information and the search for content on the Internet, several challenges have arisen. Address availability for fixed or mobile devices, security, and mobility are examples of some challenges of the current Internet. In order to solve these and other challenges, several proposals for Future Internet architectures were created to meet the needs in the virtual world. New architectures were created in order to work together with the current Internet architecture or even replace them someday. In this context, the present work proposes a performance evaluation of the NovaGenesis (NG) architecture, a Future Internet architecture created from scratch, using content distribution services (messages and photos) with Docker containers, evaluating the feasibility of adopting this platform for scaling NG services and/or other Future Internet architectures. A proof of concept of NG in different environments is evaluated and the results analyzed comparatively. This work proposes to accomplish NG performance evaluations inside containers for name resolution. Experimental scenarios are also evaluated using the transfer of photographs, comparing the performance of NG in physical environment, virtual machine and container. We conclude that Docker is an important tool for evaluating new architectures, such as NovaGenesis, but it can be applied to other ones. The use of containers as an alternative to virtual machines is increasing in environments of Cloud service providers and this trend is also observed in scientific experiments oriented by the research community, due to its greater versatility, time optimization and low computational consumption. Developers are adopting the practice of using containers for programming microservices. Among the various benefits, they reduce the overhead of *hardware*, speed in application startup, ease of scaling and load balancing among containers. The growth of this technology is related to the facilities offered by the Docker platform, which facilitated the creation and execution of containers. Despite the inherent advantages, there is an important gap to be filled in the use of containers in scientific research in the area of communications networks, more specifically in the performance evaluation of new architectures.

***Keyords*:** Container, Docker, NovaGenesis, Name Bindings, Evaluation, Experimentation Environment, Future Internet Architecture

Capítulo 1

Introdução

Este Capítulo relata os desafios da Internet quanto a nomeação, resolução de nomes e armazenamento de conteúdo, a justificativa de virtualização de Sistema Operacional, principais motivações para a realização deste trabalho, lista os objetivos de pesquisa almejados e, finalmente, mostra como está estruturado o restante da presente dissertação.

1.1 Internet e Seus Desafios

A Internet é uma das maiores e mais importantes criações já inventadas pela humanidade. Essa infraestrutura transformou a maneira como a sociedade se comunica, trabalha e se diverte desde que se tornou o ambiente global de troca de informações. Em outras palavras, é um artefato muito importante para a economia, negócios, governos e instituições contemporâneas. No entanto, alcançou escalas muito além daquelas imaginadas por seus projetistas em seu conceito original. Hoje, a arquitetura é composta por centenas de protocolos que interagem entre si e foi incrementalmente adicionada para atender às exigências de um mundo tecnológico em constante mudança [1]. Os protocolos *Transmission Control Protocol* (TCP) / *Internet Protocol* (IP) foram projetados em um momento em que os recursos de computação, armazenamento e comunicação eram escassos, o que tornava o projeto limitado e, como resultado, uma arquitetura focada na transferência robusta de pacotes entre *hosts* foi projetada. Apesar de sua importância ou da evolução dessa infraestrutura, os principais protocolos da arquitetura da Internet permanecem praticamente inalterados desde a década de 1970, apresentando vários aspectos que precisam ser aprimorados para atender às demandas atuais e futuras de aplicativos. Como a sociedade é atraída para novas aplicações, as tecnologias emergentes devem lidar com as imperfeições e limitações dos protocolos adotados na infraestrutura atual [2].

Um dos exemplos notáveis da atual limitação da arquitetura está relacionado à mobilidade dos dispositivos móveis. Quando a Internet foi projetada, as máquinas eram fixas e usavam endereços estáticos. Isso contrasta com a realidade atual, na qual existem dispositivos móveis com endereços dinâmicos conectados à rede. Devido a essa limitação, os endereços podem ser constantemente mudados e a maneira como estes endereços são atribuídos aos dispositivos, pode ser facilmente desconectados de uma rede de telecomunicações para outra. Aliado a esse problema de mobilidade, não há

registro perene ou temporário entre o nome dos usuários e os endereços confiados às suas máquinas. Consequentemente, o rastreamento da origem de um acesso é dificultado devido a essa alternância de identidade ao longo do tempo e do local. Ainda, essa alteração de endereço pode causar a perda de conexão entre os programas executados em um computador, fazendo com que sejam interrompidas na maioria das situações de comutação de conexão [3].

Para ilustrar algumas limitações da Internet, exemplos incluem a complexidade da arquitetura, dificuldade de introdução de novos modelos de serviços, novas funções [4–7]; falta de suporte para adicionar novos espaços de nomes aos serviços e conteúdos [5–8]; limitação de endereços IPv4, dificuldades de mudanças para endereços IPv6 [7]; distribuição de conteúdo, armazenamento em cache, proveniência e integridade [4, 9]; e os recursos limitados na resolução de nomes do *Domain Name System* (DNS) [6, 10]. Estes estão intrinsecamente relacionados em como as entidades na rede são nomeadas e também em como os nomes são resolvidos entre domínios. Nomes de domínio, soquetes, endereços *Media Address Code* (MAC), endereços IP, todos estes são considerados nomes na infraestrutura atual. Na Internet atual, não é feita a distinção de nomes independentes de localização de endereços dependentes de localização. Para exemplificar, observa-se o IP. São atribuídas às interfaces dos dispositivos uma identificação individual, a fim de permitir a localização desses dispositivos utilizando o endereçamento hierárquico da Internet. Logo, ao se mover uma entidade, seu IP (localizador) é obrigatoriamente alterado, afetando assim as aplicações acima, pois o endereço IP também faz parte do soquete, identificando o serviço de transporte para as aplicações. Soma-se ainda o efeito no recebimento de pacotes, que impacta diretamente na experiência dos usuários [11].

Preocupados com os problemas da Internet atual, vários estudos apontam para a necessidade de arquiteturas alternativas, projetadas a partir do zero e compostas por novos paradigmas que atendam às aplicações atuais e futuras, em especial que forneçam novas abordagens para nomeação e resolução de nomes. Essas alternativas estão sendo desenvolvidas para sanar os problemas mencionados, trabalhando em conjunto ou substituindo a Internet existente, e coletivamente são conhecidas como *Future Internet Architecture*s (FIAs) [12], podendo ter como direção o paradigma de *Information Centric Networking* (ICN). A principal característica da ICN é centralizar o projeto da rede no conteúdo disponível e na resolução de nomes de conteúdo. A busca do conteúdo desejado se dá pelo seu nome, muitas vezes usando o modelo de comunicação *publish/subscribe*. Isso ocorre independentemente de onde esse conteúdo esteja localizado na rede [13], com armazenamento temporário (*caching*) dos conteúdos mais populares nos elementos da rede. Uma vez que os conteúdos passam a ser recuperados a partir de armazenamento distribuído na rede, os mecanismos de segurança precisam garantir que conteúdos publicados com restrição de acesso sejam consumidos apenas por usuários devidamente autorizados. Nesse contexto, muitos tem investido em arquiteturas de rede voltadas para a distribuição eficiente de conteúdos na Internet. Nesse trabalho, focamos na arquitetura NovaGenesis (NG), em desenvolvimento no Instituto Nacional de Telecomunicações desde 2008. NG implementa uma ICN sobre novos protocolos de comunicação auto-organizáveis. A auto-organização de protocolos, serviços, é realizada de baixo para cima para criar um ecossistema digital, com base em nomes, associações de nomes e contratos para atender a metas e

políticas da arquitetura.

NG é uma FIA *clean-slate* (Folha em Branco). Foi iniciada com a ideia de partir do zero como se não houvesse a Internet convencional, e tem como princípios, a distribuição rápida e eficiente de conteúdos [14]. A arquitetura foi desenvolvida com o intuito de resolver alguns problemas inerentes da troca de conteúdos na Internet, tais como: (i) serviços produtores e consumidores de conteúdos auto-organizáveis; (ii) composição dinâmica de serviços baseados em contratos; (iii) nomeação e resolução de nomes generalizada para qualquer entidade da rede; (iv) acesso ininterrupto e assíncrono a conteúdos; (v) armazenamento de conteúdos em cache; (vi) proveniência, rastreabilidade e integridade dos conteúdos; (vii) ciclo de vida de conteúdos e serviços. Analisar o desempenho em escala e com experimentos do protótipo da NG tem sido uma questão em aberto desde 2012.

Para avaliar esta arquitetura, nesse trabalho será utilizada uma metodologia de experimentação e um novo ambiente de análise da NG. Essa metodologia possibilita testar a aplicabilidade da NG em máquinas físicas, máquinas virtuais, e ainda contêineres *Docker*. No entanto, serão utilizados aplicativos NG para serviços de resolução de nomes, publicação de conteúdo e armazenamento em *cache*. Estes serão utilizados nos experimentos, a fim de comprovar a eficácia na resolução dos problemas mencionados anteriormente.

1.2 Virtualização

A virtualização desempenha um papel importante no fornecimento do ambiente em que os aplicativos são executados. Além disso, com a tendência de *softwarization* das arquiteturas, cada vez mais funções de rede são transformadas em programas de computador que executam em *data centers*. Para reduzir custos e aumentar a eficiência do *data center* de uma empresa, a área responsável pela infraestrutura de Tecnologia da Informação (TI) investiu pesadamente na consolidação de servidores nas últimas décadas. A consolidação de servidores permite que vários servidores virtuais sejam executados em um mesmo *host*, o que leva a uma melhor utilização da capacidade do *hardware* reduzindo o espaço e o consumo de energia nos *datacenters* [15, 16].

Vários aplicativos (ou funções virtuais de rede), que tradicionalmente são executados em servidores dedicados, foram mesclados em um *cluster* de servidores compartilhados. Tanto que, embora a consolidação de servidores tenha um enorme potencial para melhorar a utilização de recursos e o desempenho de aplicativos, também pode introduzir nova complexidade no gerenciamento de servidores consolidados. Isso traz novos desafios, incluindo a escolha da tecnologia de virtualização correta e da configuração de consolidação para um conjunto específico de aplicativos/funções [16, 17]. Há muitas soluções de virtualização, tecnologias baseadas em *hypervisor* (conhecido como gerenciador de máquina virtual), cujos representantes conhecidos são VMware [18], Microsoft Hyper-V [19], XEN [20] e *Kernel-based Virtual Machine* (KVM) [21]. Em cada instância de uma máquina virtual, um Sistema Operacional (SO) completo (como o GNU/Linux) geralmente é executado sobre esse *hardware* virtualizado.

Atualmente, os provedores de computação em nuvem estão adotando cada vez mais a virtualização baseada em contêineres. Os contêineres podem ser vistos como uma alternativa leve à virtualização baseada em *hypervisor*. Enquanto que o *hypervisor* abstrai o *hardware*, o que resulta em sobrecarga em termos de virtualização de *hardware* e *drivers* de dispositivos virtuais [22].

O contêiner implementa o isolamento do processo no nível do SO para evitar essas sobrecargas. Esses contêineres são executados no mesmo *kernel* do SO compartilhado que o *host* subjacente e um ou mais processos podem ser executados dentro de cada contêiner [23].

O avanço dessa tecnologia é atribuída ao *Docker* [24], que rapidamente se tornou uma ferramenta de gerenciamento de contêineres e atrativa a diferentes ambientes e desenvolvedores devido aos seus recursos e facilidade de uso. O uso de contêineres é devido a fatores comprovados, por exemplo, [25]:

- **Eficiência** - Uso eficiente de *Central Processing Unit* (CPU), uma melhor utilização de *Random-Access Memory* (RAM) e armazenamento. O contêiner compartilha esses recursos com o SO do *host* (hospedeiro), permitindo uma maior velocidade na inicialização e na interrupção das aplicações, com pouca ou nenhuma sobrecarga em comparação com aplicativos executados nativamente no SO hospedeiro;
- **Portabilidade** - Aplicativos distribuídos podem ser criados, movidos e executados usando contêineres. Os desenvolvedores e administradores de aplicativos podem executar o mesmo aplicativo em *laptops*, *Virtual Machine* (VM) ou na nuvem, automatizando a implantação em contêineres. Os contêineres podem ser facilmente compactados em uma imagem e movidos para outros ambientes [19];
- **Produtividade** - Os desenvolvedores não precisam se preocupar em configurar o ambiente ao executar o teste ao criar o aplicativo. Durante o teste, eles podem simular facilmente o sistema distribuído no ambiente de produção; e
- **Serviços em Nuvem** - A computação em nuvem tem atraído cada vez mais a migração de serviços que exige uma infraestrutura física. Com o aumento da demanda de serviços em nuvem, os provedores de serviços estão intensificando o uso da tecnologia de contêineres em suas soluções de *Plataform-as-a-Service* (PaaS). E ainda, em muitos casos, as imagens de contêiner *Docker* requerem menos espaço em disco do que imagens com máquinas virtuais equivalentes, possibilitando que as imagens possam ser implantadas rapidamente na nuvem.

1.3 Motivação

Arquiteturas folha em branco (*clean slate*) não se baseiam nos protocolos TCP/IP, o que torna incomum a avaliação prática em escala, como é o caso da NG. Apesar disso, resultados práticos dessa solução são necessários para a aceitação e para a evolução da arquitetura. Diante da necessidade de resolução das deficiências da Internet atual, esse trabalho busca formas de avaliar a arquitetura NG apresentando as novidades quanto à maneira de nomear entidades, serviço de resolução de nomes, localizar, identificar e armazenar conteúdos.

O uso da tecnologia de contêineres *Docker* traz benefícios para as tecnologias de Avaliação de Arquiteturas de Internet do Futuro (AAIF), e em especial para a arquitetura NG. Podemos definir AAIF como sendo o conjunto de tecnologias, recursos, plataformas e ferramentas necessárias para experimentação e avaliação de novas arquiteturas de Internet [26–29]. Dentre as vantagens, destaca-se a rápida inicialização de múltiplos contêineres e a redução de recursos computacionais necessários quando comparados com ambiente físico e/ou máquinas virtuais. Com isso é possível aumentar a quantidade de serviços NG distribuídos por vários contêineres. O uso de contêineres nesse trabalho é uma estratégia de experimentação.

A adoção de uma plataforma de contêineres para o provisionamento e experimentação com serviços desenvolvidos na NG, sem que haja o uso de TCP/IP nos experimentos, bem como a utilização dos serviços NG, é uma das novidades desse trabalho. Para tanto, foi necessário avaliar a compatibilidade da rede *Docker* e se os recursos oferecidos são compatíveis com os requisitos de avaliação da NG.

1.4 Objetivos

O trabalho tem como objetivo principal avaliar as soluções oferecidas pela arquitetura NG para o problema de distribuição de conteúdos (objetos de informação) com armazenamento temporário de conteúdos populares em *cache* de rede, usando sistema de resolução de nomes distribuído, nomeação auto-verificável, contratação dinâmica via auto-organização semântica. Para tanto, o trabalho primeiro aborda a resolução de ligações entre nomes, e posteriormente, como esse recurso suporta a auto-organização para distribuição de conteúdos. Para atingir o objetivo principal, foram definidos os seguintes subobjetivos específicos:

1. Realizar estudos da arquitetura NG, enfatizando como o modelo *publish/subscribe* é executado;
2. Realizar um estudo sobre o *Docker* e formas de utilização de contêineres no GNU/Linux;
3. Comparar o uso de contêineres com VMs e ambiente de rede física, avaliando as vantagens e desvantagens de seu uso;
4. Investigar a compatibilidade de rede *Docker* para os experimentos com a NG;
5. Revisão bibliográfica dos trabalhos relacionados ao tema da dissertação;
6. Propor uma metodologia de teste para ambientes físicos e virtuais que permita avaliar a solução NG quanto a resolução de nomes de conteúdo distribuído;
7. Desenvolver *scripts* usando linguagem Python para automatizar as rotinas de execução de criar, inicializar, acessar e excluir contêineres e executar os serviços NG;
8. Proposta de um novo Ambiente para Avaliação de Arquiteturas de Internet do Futuro (AAAIF) visando avaliar a NG usando contêineres *Docker*.
9. Criar uma imagem *Docker* com as bibliotecas essenciais para iniciar os serviços NG e *scripts* Python do AAAIF;
10. Realizar avaliação de desempenho da troca de conteúdos nomeados com a NG

em Contêineres *Docker* usando o AAAIF desenvolvido. Com essa avaliação, pretende-se provar que a NG oferece solução alternativa a Internet atual e outras propostas de Internet do futuro, no que tange a distribuição de conteúdos usando armazenamento temporário em *cache* de rede e mecanismo de encontro entre interessados e publicadores em um domínio de rede.

11. Comparar resultados em ambiente físico, máquina virtual e contêiner.

1.5 Publicações

Os seguintes artigos foram produzidos como resultado das pesquisas relacionadas a este trabalho e parcerias com outros trabalhos:

❶ José Rodrigo dos Santos, Tibério Tavares Rezende, Thiago Bueno da Silva, **Elcio C. do Rosário** e Antônio M. Alberti. **Proposta de Arquitetura para Distribuição de Conteúdos Nomeados em NovaGenesis com P4**. In Workshop de Pesquisa Experimental da Internet do Futuro (WPEIF). Rio de Janeiro, Brasil, May 11-13, 2020. Sociedade Brasileira de Computação.

❷ **Elcio C. do Rosário**, Tibério Tavares Rezende, Victor Hugo D. D'Ávila e Antônio M. Alberti. **Distribuição de Conteúdos com NovaGenesis em Containers Docker**. In Workshop de Pesquisa Experimental da Internet do Futuro (WPEIF). Rio de Janeiro, Brasil, May 11-13, 2020. Sociedade Brasileira de Computação.

❸ **Elcio C. do Rosário**, Victor Hugo D. Davila, Thiago Bueno da Silva and Antônio Marcos Alberti. **A Docker-Based Platform for Future Internet Experimentation: Testing NovaGenesis Name Resolution.** "Communications (LATINCOM), 2019 IEEE Latin-America Conference on."(2019): 1-6.

❹ D'Ávila, V. H. D., **Rosário,E. C.**, Santos, J. R., and Alberti, A. M. (2019). **NovaGenesis NameBindings: Avaliação de Virtualização em Escala**. In Workshop de Pesquisa Experimental da Internet do Futuro (WPEIF) (pp.7–12). Gramado, Brasil, May 6-10, 2019. Sociedade Brasileira de Computação.

❺ Gabriel Henrique Coutinho Cândido, Antonio Marcos Alberti, **Elcio Carlos do Rosário**, Vaner José Magalhães, Gabriel dias Scarpioni, Everton Morais. **Teste em escala da arquitetura NovaGenesis** In: INCITEL 2018, Santa Rita do Sapucaí. Anais do INCITEL 2018.

❻ José D. Adriano, **Elcio Carlos do Rosário**, Joel J. P. C. Rodrigues. **Wireless Sensor Networks in Industry 4.0: WirelessHART and ISA100.11a**. 13th IEEE International Conference on Industry Applications (INDUSCON). (2018): 1-6

❼ Antonio M. Alberti, Everton S. de Morais, **Elcio C. Rosário**, Gabriel D. Scarpioni and Vaner J. Magalhães. **Future Internet of Things: Experimenting With NovaGenesis and Virtual Sensors [Invite]**. 18º SBMO – SIMPÓSIO BRASILEIRO DE MICRO-ONDAS E OPTOELETRÔNICA. Anais MOMAG 2018, Santa Rita do Sapucaí.

❽ Alberti, A. M., **Rosário, E. C.**, Cassiano, G., Santos, J. R., D'Ávila, V. H. D., and Carneiro, J. R.. **Performance Evaluation of NovaGenesis Information-**

Centric Network. In 2nd International Multidisciplinary Conference on Computer and Energy Science (pp. 1–6). Split, Croatia, July 12-14, 2017. 2nd International Multidisciplinary Conference on Computer and Energy. Science (SpliTech).

❾ Ferreira, F. A., Miranda, F. S., **Rosário, E. C.** do, D'Ávila, V. H. D., and Alberti, A. M.. **NovaGenesis no Ambiente FIBRE: Primeiras Impressões.** In Workshop de Pesquisa Experimental da Internet do Futuro (WPEIF) (pp. 6–9). Belém, Brasil, May 15-19, 2017. Sociedade Brasileira de Computação.

1.6 Organização da dissertação

A dissertação foi organizada da seguinte forma. Após a introdução apresentada no Capítulo 1, os conceitos de fundamentação para compreender o problema sendo atacado, ingredientes utilizados no ferramental de avaliação de desempenho proposto, incluindo apresentação detalhada da proposta NG são abordados no Capítulo 2. No Capítulo 3, encontram-se os trabalhos relacionados ao tema desta pesquisa, bem como um comparativo de ICNs, a co-relação entre elas, uma avaliação das lacunas de pesquisa existentes e como são avaliadas experimentalmente. A proposta de um novo AAAIF visando avaliar a distribuição de conteúdos com a NovaGenesis usando contêineres *Docker* é feita no Capítulo 4. A metodologia utilizada para avaliação dos experimentos é compreendida no Capítulo 5 e os resultados obtidos dos experimentos utilizando o AAAIF *versus* VMs e ambiente físico são apresentados no Capítulo 6. Para finalizar, no Capítulo 7 estão as considerações finais e lições apreendidas nesse trabalho.

Capítulo 2

Fundamentação

Este Capítulo tem como objetivo apresentar uma fundamentação teórica dos principais conceitos relevantes ao tema desse trabalho: **avaliação das soluções de resolução de ligações entre nomes e de distribuição de conteúdos usando armazenamento temporário proposta pela NovaGenesis**. Iniciamos apresentando as limitações da Internet na Seção 2.1. Para o entendimento dos principais conceitos da arquitetura NG, foi preparada a Seção 2.2. Na Seção 2.4 é apresentada a evolução história dos contêineres, ferramenta chave para o AAAIF proposto nessa dissertação. Na Seção 2.5 estão as definições de contêiner. Na Seção 2.6 será feita uma comparação entre virtualização via contêineres *Docker versus* VMs. É descrito sobre o *Docker* na Seção 2.7 e como ele explora os principais componentes do *kernel* do GNU/Linux na Seção 2.8. Enfim, na Seção 2.9 foram realizadas considerações parciais sobre os assuntos descritos neste Capítulo, sempre tendo como perspectiva a avaliação de desempenho da solução NG para resolução de nomes e distribuição de conteúdo utilizando *cache* temporário de rede.

2.1 Limitações de Nomeação, Resolução de Nomes e Distribuição de Conteúdo na Internet

Várias limitações no *design* da Internet atual estão se tornando cada vez mais aparentes. Alguns exemplos incluem, nomeação, resolução de nomes, distribuição de conteúdo e armazenamento [4, 9, 30].

2.1.1 Nomeação

Um sistema de nome é uma forma de organizar entidades seguindo a uma sintaxe lógica e semântica. Um grupo de nomes considerados válidos pode ser reconhecido como espaço de nomes ou de nomeação. Considere-se que, uma grande quantidade de dispositivos, conteúdos e serviços estão sendo integrados ao atual sistema de nomeação da Internet. A nomeação é realizada utilizando endereços IP, no qual os elementos de rede recebem números que identificam dispositivos. Os valores são limitados, sendo representados com 32 bits (IPv4) e/ou desafiador quanto à implementação da versão 128 bits (IPv6) em toda infraestrutura da Internet [31]. Um exemplo de nomeação na

arquitetura atual seria, https://www.inatel.br/ictlab/images/equipe/elcio.jpg. Uma estrutura hierárquica baseada no endereço do servidor e seus subdiretórios. Observa-se que, o nome do arquivo é elcio.jpg, ele está endereçado dentro de vários diretórios no domínio inatel.br (inatel.br./131.221.240.56). Caso o endereço IP seja alterado ou o arquivo movido para outro domínio, o mesmo (arquivo) não será localizado, impactando na experiência do usuário ao tentar acessar.

Neste cenário, um problema aparente permanece em discussão: será que o atual sistema de nomeação atende aos novos serviços que serão desenvolvidos? Pesquisadores alegam que o DNS não permite a personalização dos nomes e a sua adaptabilidade no contexto ao qual ele esta inserido. Na Internet, existem somente dois espaços de nomes globais, o endereçamento IP (identificador e localizador na Internet atual) e o DNS, e ambos são acoplados na infraestrutura (domínio e topologias/segmentos de rede) [31–34]. Estritamente falando, o DNS é estruturado em diretórios distribuídos por várias localidades no mundo e o espaço de nomes seria o espaço dos nomes de domínios.

2.1.2 Resolução de Nomes

O atual sistema de resolução de nomes hierárquicos (DNS) da Internet permite a resolução de *Natural Language Name*s (NLNes) de domínios (e outros serviços de rede) para endereços IP [35]. Através deste sistema, pode-se resolver um endereço de *Uniform Resource Locator* (URL) para um endereço IP, permitindo o acesso a sites, conteúdos e aplicações de maneira amigável às pessoas. Isso, diferentemente do endereço (facilmente interpretado por máquinas) do diretório em que estão localizados os conteúdos. Por exemplo, uma pessoa digita em seu navegador o URL do domínio **https://www.example.com/** para usar o serviço da *Web* "exemplo" de uma maneira conveniente. Por trás disso, o DNS converte os NLNes de domínio hierárquicos em endereços IP, por exemplo (203.0.113.1). Em geral, pode-se dizer que a maioria dos URLs possui apenas um único endereço e vice-versa.

Com propósito genérico, a arquitetura TCP/IP utiliza serviços de resolução de nomes apenas para camadas superiores, em específico, a camada de aplicação. O funcionamento do DNS, por exemplo, identifica o endereço IP de um *host* (podendo utilizar IPs diferentes em várias interfaces de rede) e varre todas as camadas do TCP/IP, a fim de alcançar no servidor remoto a aplicação do DNS. Isso mostra que, mesmo flexível, alcançar as informações (endereço IP) na camada de rede depende de uma varredura completa nas camadas superiores. Outras limitações vêm na mesma direção ocasionando problemas estruturais, como a rigidez e a necessidade de expandir o espaço de nomes hierárquicos, juntamente com o seu endereçamento. Conforme as referências empregadas vão se encaixando tanto como características classificatórias, como também classificações particulares à posição do nó que as hospedas, as dificuldades de mobilidades de dispositivos móveis são claramente notórias [36, 37].

2.1.3 Identificação e Localização

A identificação e localização são essenciais para todos os sistemas de telecomunicações. Um Identificador (ID) é um nome usado para identificar alguma existência

individual de objeto em algum escopo (espaço de nome), enquanto um Localizador (LOC) indica a posição atual em que um objeto individual está em algum espaço [38]. Do ponto de vista do escopo (sala, prédio, domínio) temos os identificadores. Do ponto de vista dos espaços (coordenadas e distâncias entre os objetos) temos os localizadores. Um escopo define um grupo de identificadores, os espaços definem como os objetos do grupo se relacionam [39].

Os endereços IPv4/v6 são utilizados para identificar redes e *hosts*, bem como localizá-los na estrutura de endereçamento da Internet. Como o endereço IP é utilizado para identificar e localizar um objeto, o mesmo pode não ser encontrado quando é movido de uma rede para outra, causando impacto quanto à mobilidade. Seria como se uma pessoa tivesse seu CPF alterado ao mudar de endereço, o que não faz nenhum sentido. O endereço IP define na rede local onde o *host* está localizado, enquanto na camada de transporte (TCP) identifica o ponto exato de conexão. Por exemplo, um computador no laboratório ICT Lab. (INATEL) recebe o IP 10.10.0.19, este número identifica um dispositivo entre vários. Entretanto, o mesmo endereço é utilizado pelo TCP para manter seus fluxos de dados. Dessa forma, o mesmo número será utilizado para dois objetivos: (i) identificação do dispositivos entre milhares; (ii) localização para entrega ou envio de dados na rede. O problema se agrava ainda mais quando aplicativos da camada acima (camada de aplicação) reutilizam o IP como identificador, em vez de usar um *Fully Qualified Domain Name* (FQDN) da sua própria camada. Outra limitação está relacionada ao esgotamento de endereço IPv4 devido ao crescimento exponencial de aparelhos criados e inseridos na rede. Para tanto, novas arquiteturas propõem o desacoplamento do ID(usado para identificar o dispositivo durante a comunicação)/LOC(usado para entregar o datagrama), permitindo inserir novos espaços de nomes [39]. Na Seção 2.2.3 será discutido e apresentado exemplo de divisão do ID/LOC.

2.1.4 Distribuição de Conteúdos

Com o avanço de aplicações e dispositivos de criação e modificação de conteúdos, nos dias atuais, as redes de telecomunicações passam por grandes desafios, em que parte dos usuários deixaram de ser somente consumidores para também serem criadores de conteúdos. A maior parte dos conteúdos, como vídeos, fotos, e mensagens são enviados a todo momento nas redes sociais. Em outras palavras, os usuários estão focados no conteúdo que a Internet possui, enquanto a infraestrutura atual foca na conectividade de terminais, de redes e em onde os conteúdos serão armazenados [40].

O problema desse modelo é que a arquitetura atual está preocupada no caminho percorrido para alcançar e/ou armazenar o conteúdo produzido pelos bilhões de consumidores e produtores. Toda transferência de informação foi planejada em pacotes com controles associados ao remetente e ao destinatário [40]. De encontro com esse problema, várias propostas vêm sendo implementadas com o viés na distribuição, armazenamento e recuperação do conteúdo por meio de *cache* de rede. Em outras palavras, cópias são armazenadas nos roteadores e/ou servidores próximos aos consumidores. Assim, estes conteúdos podem ser consumidos de forma rápida independente de sua origem [41–43].

2.2 NovaGenesis

O projeto NovaGenesis [44] teve início no ano 2008, no Instituto Nacional de Telecomunicações (INATEL). O objetivo inicial do projeto foi repensar a arquitetura de troca de informações tradicional, cobrindo vários aspectos, tais como: *things*, redes, *cloud computing*, serviços, inteligência artificial, *blockchain*, identidades, credenciais, *softwarization*, virtualização e biometria. A NG pode ser vista como uma meta arquitetura capaz de suportar inúmeras tecnologias atuais e as que estão por vir [44].

A NG é uma arquitetura de Internet do Futuro baseada em quatro pilares: (i) nomeação, resolução de nomes e *cache* de armazenamento temporário de conteúdo; (ii) ciclo de vida das entidades; (iii) representante do mundo físico; (iv) estrutura flexível de camadas. As condições de mapeamento e armazenamento de conteúdos na rede foram inseridas no primeiro pilar. Para o segundo pilar, os serviços e todas as funcionalidades de rede e a forma de comunicação entre eles é realizada por meio de *Aplication Programming Interface*s (APIs). No terceiro pilar, permite-se transcender elementos e dispositivos do mundo físico para o virtual como serviços. Por fim, no último pilar permite-se implementar novos protocolos como serviços. A ideia é criar algo como uma loja de aplicativos, que permite pesquisar implementações de *software* de protocolos e instalar estes novos protocolos na camada de *software* da rede de telecomunicações.

2.2.1 Nomeação, Nomes Auto-verificáveis, e Resolução de Nomes Hierárquico

A nomeação na NG é ilimitada, podendo nomear qualquer entidade em NLNes, um nome que denota significado, como, por exemplo, "Maçã". É possível também utilizar os *Self-verifying Name*s (SVNes), nomes não compreensíveis aos seres humanos, criados a partir da codificação de um NLN ou de algum objeto de informação, utilizando um algoritmo de função *hash*. O uso de SVN como uma alternativa ao NLN [45] possibilita verificar a relação entre uma entidade nomeada e seu nome a qualquer momento. De acordo com Ghodsi et at. [10], os SVNes têm melhor segurança, escalabilidade e flexibilidade que os NLNes. Uma maneira de calcular SVNes é empregar funções *hash* criptográficas. Como exemplo, um SVN pode ser calculado a partir de um nome de um contêiner com o *Message Digest Algorithm 5* (MD5), onde SVN = MD5(conteiner-01) = 876837d1e16460f884e777476e2e2fdc. O MD5 é uma função *hash* e "conteiner-01" um exemplo de nome de contêiner [46]. O *hash* resultante é considerado um nome auto-verificável, uma vez que o NLN é embaralhado. O nome é considerado auto-verificável, porque as palavras binárias podem ser embaralhadas, aumentando sua integridade e identificando o nome do objeto (dispositivo, arquivo, etc.) como único na rede. Na NG, foi implementado o algoritmo de função *hash* MurMurHash3 [47]. Com os dois tipos de nomes (NLN, SVN) empregados, a NG é capaz de realizar ligações que representam relações entre entidades nomeadas.

Com a capacidade de uso de dois tipos de nomes (NLN, SVN), a NG oportuniza a ligação de nomes entre as entidades envolvidas. Cada *Name Binding* (NB) é um conjunto entre dois ou mais nomes no formato: $<$ chave; valor(es) $>$ [44]. Na NG, podemos nomear o *Host Identifier* (HID), *Operating System Identifier* (OSID),

Process Identifier (PID) e IDs de componentes internos. O desenvolvedor não fica limitado a um conjunto fixo de nomes e espaços de nomeação permitidos.

Essa solução permite representar a relação entre o objeto e a quem (dispositivo) esse objeto está contido. A Figura 2.1 mostra um "SO Ubuntu", "conteiner-01"é o nome do contêiner e "Processo 1" e "Processo 2" são os nomes dos processos em execução no contêiner. Logo, observamos a relação entre as entidades nomeadas, representadas com os seguintes NBs:

- < **SO Ubuntu, conteiner-01** >: O conteiner-01 está contido no SO Ubuntu;
- < **Processo 1, conteiner-01** >: O Processo 1 está contido no conteiner-01.

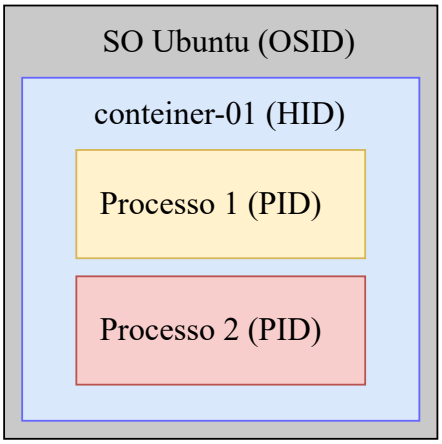


Figura 2.1: *Exemplo de nomeação de um SO com um contêiner. O "Processo 1" e "Processo 2" estão contido no "conteiner-01". PID é o identificador do processo e OSID é o identificador do OS. O "conteiner-01" (HID) está contido no "SO Ubuntu"*

O exemplo mostrou a ligação com o uso de NLNes. Outra forma seria com a utilização de SVN, como a seguir:

- < **2DF2CB16 , E5FFE52D** >: 2DF2CB16 é o SVN do Identificador do "SO Ubuntu" e E5FFE52D é o SVN do "conteiner-01";
- < **E5FFE52D, E75B71AD** >: O E75B71AD é o Identificador do Processo 1. Ele está contido no "conteiner-01", ou seja, no Identificador E5FFE52D.

2.2.2 Resolução de Nomes e Cache de Rede

Um dos princípios fundamentais de ICNs é o *cache* de rede ou armazenamento temporário em rede. Com ele, os usuários podem acessar os conteúdos armazenados entre os nós, otimizando a distribuição global de informações. Na NG, o conteúdo é armazenado por um sistema de serviço de *cache* de rede [26, 44].

Na atual implementação da arquitetura NovaGenesis, a mesclagem de resolução de nomes e conteúdos armazenados no *cache* da rede em um domínio formam o chamado *Name Resolution and Network Caching Service* (NRNCS). Um sistema de serviços de resolução de nome e *cache* para um domínio NG. Nas operações do NRNCS, os serviços publicam e assinam os NBs e/ou conteúdos associados, implementando o modelo publica/assina de acesso. O modelo publica/assina na NG assemelha com o

protocolo *Message Queuing Telemetry Transport* (MQTT), porém de forma invertida. Na NG, primeiro se publica, depois se assina, enquanto que no MQTT, primeiro se assina, depois se publica [44, 48].

Com o NRNCS, os desenvolvedores implementam novos espaços de nomes e existentes, empregando NLNes e/ou SVNes. Isso resolve as limitações de nomes do IP e também a utilização somente de NLN no DNS para resolução de nomes. Pois, as possibilidades de nomear objetos são ilimitadas. Logo, o NRNCS é uma solução na distribuição de conteúdos. Os conteúdos são localizados pelo seu nome, independente de onde ele esteja. Diferente do modelo atual da Internet, que necessita da especificação de todo o caminho para alcançar, por exemplo, uma foto.

A implementação atual do NRNCS na arquitetura NG mantém três serviços: (i) *Publish/Subscribe Service* (PSS), serviço responsável pela leitura/escrita de processos por meio do modelo publica/assina. Este faz o encontro entre o interessado e o publicador do conteúdo/NBs e recebe o *binding* que deve ser armazenado, passando-o para o *Generic Indirection Resolution Service* (GIRS); (ii) GIRS, serviço intermediário que escolhe quais instâncias (uma ou mais) de *Hash Table Service*s (HTSes) serão utilizados para armazenar uma ligação de nome/conteúdo. A decisão é realizada com base no padrão binário do nome de origem e uma ligação de nome; (iii) e o HTS, serviço de *hash table* do domínio responsável por armazenar o *binding* e o conteúdo associado, se existir. Assim, o NRNCS realiza balanceamento de carga, gerencia o grafo de serviços de resolução de nomes e armazenamento em um domínio [44].

Toda troca de conteúdos por padrão vai parar no PSS, que carrega os NBs e conteúdo para o *cache* local. Ele possui uma API com 5 primitivas: (i) o processo publica um NB (e conteúdo se houver) para o PSS; (ii) outro processo assina NB/conteúdo via PSS; (iii) PSS publica e notifica o parceiro sobre o serviço; (iv) PSS assina e notifica o parceiro sobre o serviço; (v) PSS revoga uma publicação, ou seja, retira do *cache* o conteúdo. Portanto, o PSS pode ser interpretado como uma API distribuída, podendo ser descoberta e acessada por seus nomes (SVNes) [44].

2.2.3 Identificador e Localizador na NG

Na NG, todos objetos são nomeados e recebem um identificador único. À solução cria somente um grafo para tudo. Quando precisa identificar e localizar algum objeto, uma pesquisa recursiva é realizada nesse grafo. Para tanto, envia perguntas (pistas) para a rede e obtém respostas até atingir o alvo desejado. A Figura 2.2 mostra um grafo de NBs que inclui domínio, *host*, SO, processo e conteúdo (foto). O *host* 1 está contido no domínio 1, onde o *host* 1 contém um SO e que o processo 1 reconhecido como "2743" no SO contém uma foto com o SVN 01011339.

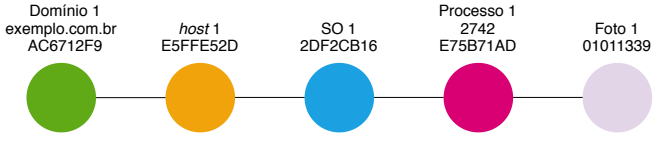


Figura 2.2: *Grafo que exemplifica os NBs de objetos dentro de um domínio. A relação dos objetos pode ser entendida como está contido ou contém, dependendo do modo que estiver analisando. Adaptado de [14]*

No contexto de localizadores, tomamos como exemplo um *Laptop* que recebe o nome de identificação “ICTLAB01”. Este *Laptop* pode ser localizado pelas coordenadas *Global Positioning System* (GPS) “-22.256897,-45.696977”. Com as coordenadas e também com o identificador a NG faz uma NB entre ID e o LOC: <“-22.256897,-45.696977”, “ICTLAB01”>.

Quando o dispositivo é movido de localidade, basta modificar o NB, por exemplo para: <“-22.251928,-45.7092636”, “ICTLAB01”>. Observe que o identificador permanece o mesmo “ICTLAB01”. Para localizar novamente o *Laptop* basta pesquisar pelo seu nome. Dessa forma, identificadores e localizadores são independentes, essa divisão é muito importante quando utilizamos dispositivos móveis [49].

2.2.4 Serviços e Contratos

Um processo no SO ou qualquer programa que objetiva enviar ou armazenar informação pode ser interpretado como um serviço [38]. A comunicação nas redes atuais requer a utilização de protocolos entre as partes envolvidas no processo. A NG utiliza uma abordagem diferenciada, os protocolos de comunicação são implementados como um serviço, onde processam, trocam e armazenam dados construindo assim a rede.

Os serviços são organizados com base em nomes, NBs e contratos que atendem os objetivos e políticas da arquitetura. O contrato é a formalização da prestação de serviços. Ele define as regras a serem respeitadas, responsabilidades, bem como os critérios de confirmação aos serviços que seguiram as leis ou punição na prestação dos serviços que infringem as regras. Tudo na NG é visto como um serviço, de tal modo que qualquer coisa que se faça, ou seja, armazenamento, processamento, transferência de informação e *cache* são considerados serviços [44].

O *Service Level Agreement* (SLA) é um sinônimo de contrato. Os SLAs podem indicar exatamente quais são os nomes acessíveis a outros domínios ou outros mecanismos de autorização que podem ser empregados na arquitetura. Portanto, para o ciclo de vida dos serviços, que inclui exposição de capacidades, descoberta de parceiros, negociação, contrato, operação e finalização. A negociação do SLA é um processo de extrema importância para o completo ciclo de vida de serviços. Esse modelo de contrato de serviço pode ser aplicado aos requisitos de *Internet of Things* (IoT) [50, 51] e também nas cidades inteligentes [52]. A NG utiliza em sua implementação uma abordagem baseada em contratos, em que os domínios estabelecem SLAs antes de assinar cópias de NBs de outros domínios. Uma NB local sempre será armazenada dentro de um domínio de maneira confiável.

2.2.5 Ciclo de Vida de Serviços e Conteúdos

A NG é a única proposta de Internet do futuro que inclui o ciclo de vida completo de serviços no *core*. Segundo o Prof. Dr. Antônio Marcos Alberti [53], criador da proposta NG, isso é feito para suportar a certificação dos serviços de rede. A arquitetura permite fazer a exposição de serviços (incluindo implementações de protocolos de comunicação), depois permite a descoberta de possíveis serviços parceiros, permite que estes negociem entre si usando o NRNCS, formalizando um contrato, iniciando as

operações e finalizando os mesmos quando não mais necessário. Nenhuma outra arquitetura desfruta do uso de contratos para orquestração da implementação de protocolos, serviços e aplicações.

2.2.6 Exposição e Descoberta

Os serviços expõem o que eles fazem, ou seja, expressam o que fazem por meio de nomes, grafo de nomes e através do sistema de resolução de nomes/*cache* de rede, se descobrindo por meio de NLNs e/ou SVNes. A ideia da exposição é ter vários serviços que geram conteúdos (mensagens, fotos, vídeos) de repositórios confiáveis de maneira que não haja duplicidade entre eles [44].

Para promover a descoberta de serviços na NG, entidades de redes assinam um contrato com NB, onde são utilizadas palavras-chave expostas. As etapas de exposição e descoberta são conduzidas pelo serviço *Proxy/Gateway/Controller Service* (PGCS). O PGCS foi criado para representar dispositivos e coisas em um domínio NG, podendo atuar tanto como *Proxy* quanto *Gateway* (GW). Como *Proxy*, o PGCS é capaz de representar recursos e serviços de *hardware* físico ou virtual em sua arquitetura interna, através do envio de NBs para outros PGCSes. Como *Gateway*, ele encapsula mensagens sobre soquetes do SO ou simulados, para o caso de uma implementação de simulador de rede [44].

2.2.7 Encapsulamento de Mensagens

As mensagens internas aos serviços no domínio NG são objetos na memória, as quais são transformadas de objeto para mensagens serializadas quando descem para camada de enlace utilizando tecnologias padrão, tais como *Ethernet*, Wi-Fi, LoRa e ZigBee. Não limitando as implementações que podem ser usadas. Ao chegarem na interface são serializadas, ou seja, as mensagens são transformadas em *strings*, caracteres ASCII [44]. No protótipo atual, não é feita a codificação de fonte.

O GW é o responsável por encapsular as mensagens entre os serviços NG. As mensagens são recebidas ou enviadas em um SO via memória compartilhada. Cada dispositivo possui um representante virtual, que é o PGCS. O PGCS representa o dispositivo onde ele se conecta [44] para fins de exposição de recursos físicos para o ecossistema de serviços. A representação do físico é hoje chamada de Gêmeo Digital, mas a NG implementou essa ideia em 2012.

2.2.8 Modelo em Camadas

O desenho da NG foi criado seguindo um modelo recursivo orientando a objetos. Onde os componentes dos serviços são conhecidos como blocos (*Block*). Um bloco de código NG é uma instância de objeto que habilita um processo NG. Estes são classificados como blocos especializados e blocos comuns. O bloco especializado executa tarefas específicas no serviço onde ele se encontra. De outra forma, os blocos comuns estão presentes em todos os serviços (modelo de *software* auto-similar). Cada serviço (PGCS, PSS, GIRS e HTS) possui seu bloco especializado. Os blocos GW e *Hash Table* (HT) são essenciais a todos os serviços [44].

O GW é um *kernel* de eventos discretos. Os protocolos são implementados em cima desse *kernel*. Este é responsável por manter uma fila de prioridade de entrega para as mensagens que aguardam processamento em um serviço. As mensagens são retiradas dessa fila e encaminhadas para componentes internos visando processamento ou para uma fila de saída relacionada com a comunicação entre processos em um mesmo SO [44].

O bloco HT é uma estrutura de dados com diversas tabelas *hash* que armazena pares <chave; valor(es)>. Nessas tabelas são armazenadas as NBs. Uma mensagem NG pode conter 1000 NBs, 1000 pares de chaves e valores.

Para o envio de mensagens/conteúdo entre serviços em SO diferentes é utilizado o *Proxy Gateway* (PG). O bloco PG é exclusivo do PGCS, implementa a transmissão e recepção por *socket* no SO Linux. O PG, GW e HT estão implementados no PGCS. O PGCS é um *software router* que encapsula mensagens NG na Camada 2 para SO interno ou externo. Uma das funções do PGCS é ofertar um Gêmeo Digital para coisas conectadas, ou seja, um serviço que representa alguma entidade do mundo físico no mundo virtual [44].

A responsabilidade do PGCS está no controle das mensagens que entra e sai do SO. Logo, o PGCS inicializa o serviço, *bootstrapping* com vizinhos, faz comunicação via *sockets*. Além dessas funções, realiza também a representação e controle de dispositivos físicos e tradução de protocolos. Além disso, fornece inicialização, exposição de nomes, descoberta e conectividade para todos os processos [44].

No PSS, o bloco *Publish/Subscribe* (PS) é encarregado da publicação/assinatura de NBs e conteúdo, se existir. O GIRS contém um bloco especializado, nomeado como *Indirection Resolution* (IR). O IR recebe a função de localizar e escolher o melhor HTS, ou seja, baseado na divisão do espaço binário das chaves nos NBs. Por fim, o serviço HTS possui um bloco especializado *Distributed Hash Table* (DHT), que fornece uma estrutura de tabela *hash* distribuída.

2.2.9 Estrutura de Linha de Comando NovaGenesis

O PGCS (principal serviço NG), que representa o dispositivo onde se conecta, executa suas tarefas utilizando um conjunto de *Linhas de Comando NG*. Cada linha de Comando NG é um texto único em uma linha. Os parâmetros desta linha são separados por espaço. Uma linha contém: (i) um comando; (ii) métodos alternativos para executar a ação desejada; (iii) versões de implementação disponíveis; (iv) e um conjunto de argumentos vetoriais. Logo abaixo, mostramos a estrutura de uma *Linha de Comando NG*.

ng -command - -alternative version [vectorial arguments]

onde:

ng: É o identificador NovaGenesis.
-command: É a ação que será executada.
- -alternative: Seleciona entre várias alternativas qual ação deve realizar.
version: Seleciona a versão desejada da implementação.
[vectorial arguments]: São os argumentos do comando.

No início, os caracteres (*ng*) escritos em minúsculo identificam o comando NG. O próximo parâmetro (*command*) com um sinal de menos é o nome do comando. Na sequência vem o parâmetro *alternative* com dois sinais de menos. Por fim, o parâmetro *version* que indica a versão.

Cada comando pode ter um ou mais argumentos vetoriais, estruturados da seguinte maneira entre colchetes *[...]*:

[n type E1 E2 E3 ... En]

onde:

n: Número de elementos no vetor de argumento.
type: Tipo dos elementos no vetor de argumento.
E1 E2 E3 ... En: São os elementos do vetor de argumento.

Um exemplo real de comando NovaGenesis é mostrado na Figura 2.3. O comando *hello* com alternativa *interhost communication* (ihc) e versão 0.1 possui um argumento. O argumento do tipo *String* (s) possui cinco elementos, que são as palavras: NULL, NULL, Ethernet, eth0 e 02:42:ac:11:00:03.

```
ng -hello --ihc 0.1 [ < 5 s NULL NULL Ethernet eth0 02:42:ac:11:00:03 > ]
```

Figura 2.3: *Comando NG (ng) com o nome hello.*

2.2.10 Mensagem NovaGenesis

A NG troca arquivos de mensagens (ASCII) compostos por várias linhas de comandos e os dados (carga útil) necessários ao processamento e comunicação. As linhas são separadas utilizando um terminador padrão do SO GNU/Linux, ou seja, o caractere ASCII /n ou 0x0A. Uma mensagem completa consiste em uma sequência de comandos linha por linha mais a carga útil no final do arquivo. Linhas em branco separam a carga útil do comando. Essa estrutura é semelhante à adotada nas requisições *Session Initiation Protocol* (SIP) [54].

Entre os vários tipos de formatos de codificação, a NG adotou o formato .txt, que é um formato simples e leve indicando uma oferta de transmissão de conteúdo. Simples pelo fácil entendimento por humanos e leve por usar pouca memória para armazenar informações. Na Figura 2.4 mostramos um exemplo de mensagem NG.

Onde:

$-m$: Indica um comando de encaminhamento/roteamento.
$--cl$: Indica encaminhamento/roteamento não orientado a conexão.
0.1: Versão 0.1.
15B239D1: É o SVN gerado por função *hash* quando usamos o PGCS com o nome do domínio "Intra_Domain".
E5FFE52D: É o ID do *host* de destino.
2DF2CB16: É o ID do OS.
E75B71AD: É o ID do Processo.
58654983: É o ID de um Bloco interno ao processo.

```
ng -m --cl 0.1 [ < 1 s 15B239D1 >  < 4 s E5FFE52D 2DF2CB16 E75B71AD 58654983 >
< 4 s E5FFE52D 2DF2CB16 512AA9F5 6AA04B23 > ]
ng -p --notify 0.1 [ < 1 s 18 > < 1 s Service_Acceptep_1015132772.txt >
< 5 s pub 474E0F5F 865A3E7 F3F1E08A 8BCD0F7E >]
ng -info --payload 0.1 [ < 1 s Service_Accpeted_1051132772.txt]
ng -p --b 0.1 [ < 1 s 2 > < 1 s F9916CDD > < 1 s 58654983]
ng -p --b 0.1 [ < 1 s 2 > < 1 s F9916CDD > < 1 s E75B71AD]
ng -p --b 0.1 [ < 1 s 2 > < 1 s F9916CDD > < 1 s 2DF2CB16]
ng -p --b 0.1 [ < 1 s 2 > < 1 s F9916CDD > < 1 s E5FFE52D]
ng -message --type 0.1 [ < 1 s 1 > ]
ng -message --seq 0.1 [ < 1 s 1 > ]
ng -scn --seq 0.1 [ < 1 s 80E97282 > ]
```

Figura 2.4: *Exemplo de mensagens NovaGenesis.*

-p –notify: Indica publicação com notificação de que um conteúdo foi publicado.
-info –payload: Indica informação da carga útil.
-p –b: Publica os código de bloco.
-message –type: Comando que possui um único argumento e um elemento do tipo *String*. Este elemento indica o tipo de mensagem.
-message –seq: Comando que possui um único argumento e um elemento do tipo *String*. Este elemento indica o número de sequência da mensagem.
-scn –seq: Indica a sequência de elementos SVN.

2.3 Comunicação entre os Serviços NG

Quando um processo inicia, primeiro ele executa o seu *Gateway*, onde há um método (uma *Action* de inicialização) que comunica com o *Gateway* do PGCS. O PGCS utiliza uma memória compartilhada do SO via uma chave do Linux. Desta forma, toda vez que se inicializa uma memória compartilhada no SO é preciso informar a semente que o SO irá usar para gerar essa chave. Essa semente, por sua vez, foi adotada de forma fixa para o protótipo NG com o número 11. Então, quando qualquer processo se inicia, ele sabe que vai existir um PGCS com este número. Assim, ele "diz" ao PGCS: estou aqui, tenho esse nome. O processo, então, expõe o que ele tem.

Todos os processos tem SVNes, além dos blocos internos, SO e *hosts*. Os serviços se descobrem, seja por memória compartilhada ou *socket*. Esse processo de descoberta é dividido em duas etapas: *hello* e *exposition*. O *hello* é uma mensagem simples e pequena com o objetivo de expor o básico de um PGCS (e outros serviços que rodam no mesmo OS, VM ou contêiner), enquanto o *exposition* é uma mensagem consideravelmente grande capaz de realizar a exposição de recursos mais completa, entregando vários NBs. Com isso, os processos se descobrem no mesmo SO ou entre SOs distintos. Todos os processos do *core* se descobrem sozinhos por meio de mensagens. Isso é o que se chama de *BootStrapping* Auto-Organizado (inicialização de um domínio local NG).

A sequência lógica de inicialização do *Core* NG em um *host* inicia com o PGCS, seguidos dos serviços PSS, GIRS e HTS. Assim que estes serviços se descobrem, o próximo passo é executar os PGCSes distribuídos por vários *hosts*. Para isso, o PGCS do *Core* NG deve conter os endereços MAC de todos PGCSes que estão nos diferentes *hosts*, enquanto os PGCSes de outros *hosts* precisam conhecer o endereço MAC

do PGCS do *Core* NG. O endereço MAC é inserido manualmente nos parâmetros de inicialização do PGCS. A descoberta por *broadcast* MAC também é possível.

A Figura 2.5 e a Figura 2.6 ilustram um domínio de comunicação entre *hosts* que utiliza a arquitetura NovaGenesis. Os blocos GWs e HTes habita todos os serviços (comuns). É possível observar o relacionamento e a comunicação entre os blocos internos e serviços, tendo a linha pontilhada como uma ligação virtual. Ela conecta os *hosts* através do serviço PGCS e é refletida sobre uma conexão física *Ethernet* entre os mesmos.

A comunicação entre blocos dentro de um mesmo serviço, bem como a comunicação entre serviços dentro de um mesmo SO, fica a cargo do GW que utiliza recursos de memória compartilhada e sistema de eventos discretos. A comunicação entre serviços em diferentes espaços (SO, por exemplo) é realizada por um bloco especializado, denominado PG.

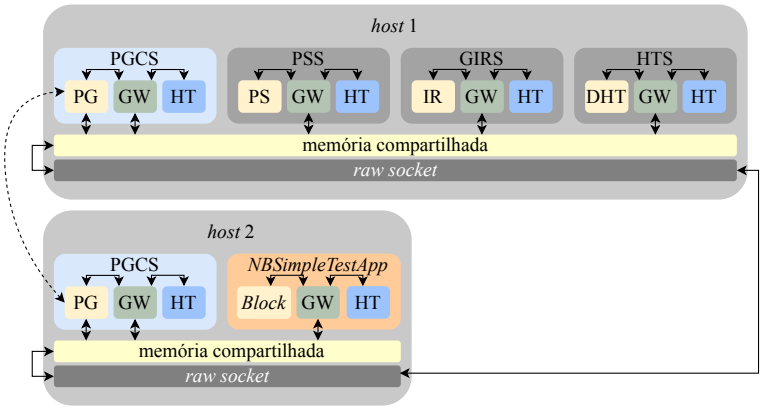


Figura 2.5: *Publicação de NBs e armazenamento no cache do domínio NG.*

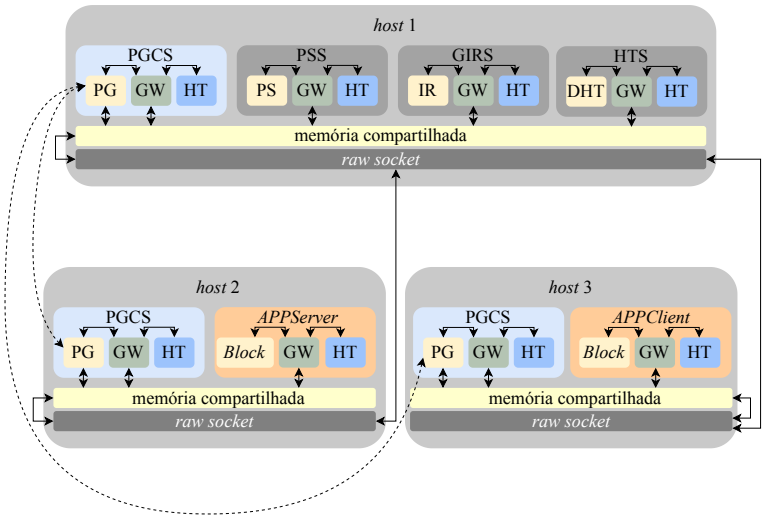


Figura 2.6: *Publicação de conteúdo e armazenamento no Servidor do domínio NG.*

Dentre os serviços NG, podem ser destacados: *NBSimpleTestApp*, que é uma aplicação para teste de publicação e assinatura de NBs; e *APPServer* e *APPClient*, direcionados à distribuição e armazenamento de conteúdo. Esses serviços podem ser considerados aplicativos da rede, que são programas desenvolvidos com linguagem C/C++

utilizando os conceitos do modelo FI NovaGenesis, os quais podem ser produtores ou consumidores de conteúdo. Estes aplicativos junto com todo o *core* NG, por sua vez, são soluções para os problemas relacionados às limitações da Internet atual discutidos anteriormente, e seus respectivos processos de comunicação serão detalhados a seguir.

2.3.1 Aplicativo de Publicações NBs

Preocupado com a limitação relacionada à resolução de nomes da Internet atual, o aplicativo de publicação de NBs denominado *NBSimpleTestApp* foi desenvolvido pelo Prof. Dr. Antônio Marcos Alberti e pode ser usado para testar uma possível solução a esse problema, devido às características de uso de nomes de linguagem natural combinados com nomes auto-verificáveis da NG, além da proximidade da informação através do uso de *caches* de redes. Ou seja, *NBSimpleTestApp* quando integrado ao *core* NG permite testar a solução proposta pela NG para resolução de nomes e distribuição de conteúdos. Na NG, as associações de nomes entre duas entidades como computadores, serviços, pessoas, nomes ou objetos que se desejam mapear são definidas como NBs, tal qual exemplificado na Subseção 2.2.1. Primeiro o aplicativo publica grandes quantidades de NBs para o NRNCS. Depois ele assina um NB aleatório para verificar o desempenho da entrega. Logo abaixo, listamos a métrica (Valor Instantâneo) de desempenho coletada pelo *NBSimpleTestApp* em execução e as estatísticas de *Round Trip Time* (RTT), a fim de mostrar resultados encorajadores relacionados à resolução de nomes, tornando a NG uma forte candidata à evolução da infraestrutura da Internet atual.

- **Instante de Tempo**: Tempo em que o evento aconteceu;
- **Valor Instantâneo**: Calcula o RTT das publicações e assinaturas;
- **Média**: Média aritmética dos tempos instantâneos

$$\mu = \frac{\sum x}{N} \tag{2.1}$$

  onde:
  - μ: Representa o tempo médio.
  - x: Representa o tempo instantâneo.
  - N: Representa o número de eventos.
- **Desvio Padrão**: Calcula a variação entre os valores instantâneos

$$\sigma = \sqrt{\frac{\sum (x-\mu)^2}{N}} \tag{2.2}$$

  onde:
  - σ: Representa o desvio padrão.
  - x: Representa o tempo instantâneo.
  - μ: Representa o tempo médio.
  - N: Representa o número de eventos.

- **Erro Padrão**: Efetua o cálculo de erro dos valores instantâneos

$$\sigma_\mu = \frac{\sigma}{\sqrt{N}} \tag{2.3}$$

onde:

  - σ_μ: Representa o erro padrão a partir da média μ.
  - N: Representa o número de eventos.
  - σ: Representa o desvio padrão.

- **Margem de Erro**: A margem de erro mede a probabilidade na qual se espera que o resultado da amostra seja próximo de 1

$$IC = \bar{x} \pm z\frac{\sigma}{\sqrt{N}} \tag{2.4}$$

onde:

  - z: Distância padrão da média μ.
  - $\bar{x}$: Intervalo de confiança ou margem de erro.
  - σ: Representa o desvio padrão.
  - N: Representa o número de eventos.

- **Média Menos a Margem de Erro**: Efetua o cálculo da margem de erro para cima, ou seja, margem superior

$$\mu_{low} = \mu - IC \tag{2.5}$$

onde:

  - μ_{low}: Representa a média menos a margem de erro.
  - μ: Representa o tempo médio.
  - IC: Representa a margem de erro.

- **Média Mais a Margem de Erro**: Efetua o cálculo da margem de erro para baixo, ou seja, margem inferior

$$\mu_{up} = \mu + IC \tag{2.6}$$

onde:

  - μ_{low}: Representa a média mais a margem de erro.
  - μ: Representa o tempo médio.
  - IC: Representa a margem de erro.

Um arquivo App.ini define os parâmetros que guiam a execução da aplicação. São definidos oito parâmetros, tal qual mostrado na Figura 2.7: (1) Define o atraso antes de iniciar o processo de descoberta de parceiros; (2) Define o intervalo entre execuções periódicas de tarefas na aplicação; (3) Quantidade de publicações que serão realizadas ao todo; (4) É o número total de assinaturas que aleatórias serão realizadas, com taxa padrão de 1 assinatura por segundo; (5) Define a quantidade de mensagens que serão enviadas em cada *burst*; (6) Define a quantidade de NBs em cada mensagem do

burst; (7) Define a quantidade de HTSes que serão utilizados no experimento. O valor adicionado é somente para orientar o usuário; e (8) Utilizado para nomear os arquivos com os dados (NBs) estatísticos de um experimento em execução. Na execução do *NBSimpleTestApp* são gerados dois arquivos: App_Core_pubrtt_2e+06_trial1.dat; e App_Core_subrtt_2e+06_trial1.dat. Os valores de atraso são definidos em segundos. A Figura 2.7 mostra que um atraso de 3 segundos foi definido para descoberta de possíveis parceiros (1), um total de 1000 mensagens serão enviadas (6), cada uma com 1000 NBs (configurado no código fonte do aplicativo), onde serão utilizados 50 *bursts* ao todo (5). Cada *burst* será uniformemente distribuído em 60 segundos (2). A quantidade total de assinaturas foi configurada para 360 (4). Por fim, o *NBSimpleTestApp* vai publicar 1 milhão (3) de NBs ao todo.

```
1 DelayBeforeDiscovery 3
2 DelayBeforeRunPeriodic 60
3 NumberOfPublications 1000000
4 NumberOfSubscriptions 360
5 NumberOfMessagesPerBurst 50
6 NumberOfPubsPerMessage 1000
7 NumberOfHTs 1hts
8 Trial trial1
```

Figura 2.7: *Parâmetros do App.ini que guiarão a execução do NBSimpleTestApp.*

O aplicativo *NBSimpleTestApp* é responsável por publicar milhões de NBs no *cache* de rede e também assinar os NBs em um domínio da NG. Com as publicações em milhões forma uma enorme rede de NBs que é acessada pelo modelo de publica/assina, gerando resultados estatísticos que comprovam as novidades e vantagens da NG. O diagrama de sequência da Figura 2.8 mostra a comunicação do aplicativo com outros serviços da arquitetura. Os serviços inicializam, fazem a exposição dos NLNes e SVNes, pesquisa e descobrem parceiros.

Como pode ser visto na Figura 2.8, no *host* 1 (Passo 1a), o aplicativo envia os NBs com destino ao PSS, que escolhe o GIRS (Passo 1b) e, em seguida (Passo 1c) o GIRS seleciona uma instância HTS. Para finalizar o processo de publicações, o HTS envia confirmações (Passo 1d) de armazenamento realizado com sucesso.

Um ou vários HTS podem armazenar os NBs. Quando existem vários HTS na rede o GIRS se encarrega pela distribuição das mensagens para cada HTS, sem que tenha duplicidade da mesma mensagem.

Quando o aplicativo conclui todas as publicações de NBs, automaticamente o modo de publicação é alterado para assinatura, consultando de volta NBs aleatórios do conjunto publicado e armazenado no HTS do domínio NG. Os passos de assinaturas também são mostrados no diagrama da Figura 2.8. O NB é assinado, o aplicativo envia a assinatura (Passo 2a) para o PSS, em seguida o GIRS entrega (Passo 2b) para HTS.

É exatamente assim que o RTT de publicações e assinatura de NBs são medidos, ou seja, quanto tempo os NBs precisam para percorrer o caminho até alcançar a instância HTS e o *NBSimpleTesApp* receber as confirmações.

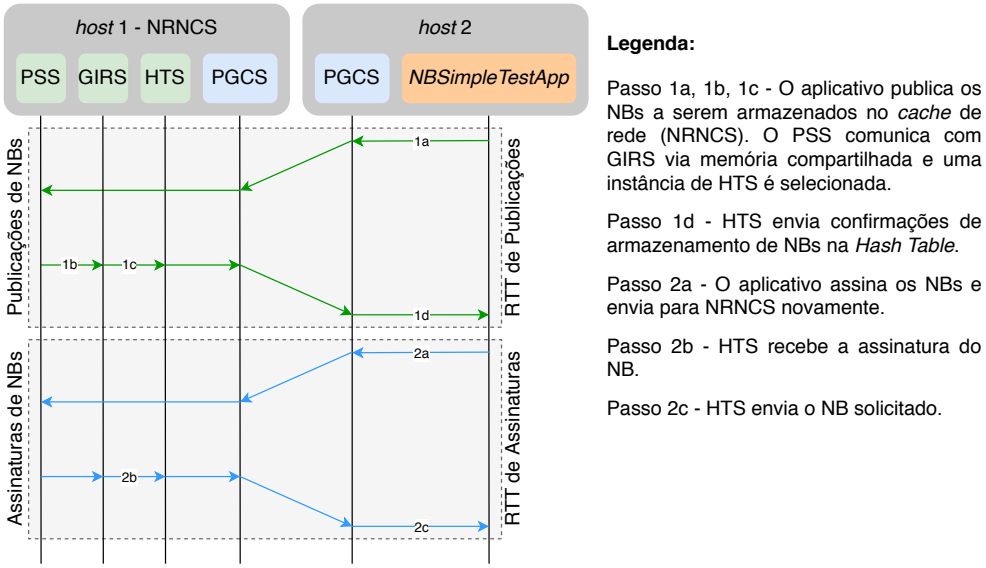


Figura 2.8: *Diagrama de Sequência de NBs.*

2.3.2 Aplicativo de Distribuição e Repositório de Conteúdo

A distribuição de conteúdos eficiente, confiável e baseada em *cache* é um dos principais desafios da Internet atual [3]. Pensando nisso, o Prof. Dr. Antônio Marcos Alberti desenvolveu um par de aplicativos que utiliza o *cache* de rede e entrega de conteúdo nomeados fornecidos pela NG [53] para provar o conceito de distribuição de conteúdos da NG. Desta forma, a NG aborda o desafio de projetar um aplicativo distribuído cliente/servidor múltiplo para distribuição de conteúdo. Além dos clientes (fontes), os servidores (repositórios) também podem publicar conteúdos nomeados de acordo com as políticas definidas pelo usuário. Os servidores mantêm um repositório coerente de todo o conteúdo que uma pessoa possui usando IDs de usuário exclusivos (UIDs).

O *APPClient* e o *APPServer* consistem de serviços de gerenciamento de conteúdo distribuído que descobrem, negociam e se oferecem para publicar/assinar, encaminhar e armazenar em *cache* o conteúdo nomeado. Assim, as aplicações usam o NRNCS e o PGCS para associar nomes que sejam relacionados ao conteúdo, por exemplo, mensagens, arquivos com fotos, video e áudio aos seus IDs de processos (PIDs). As estatísticas de desempenho utilizadas por essas aplicações na publicação e assinatura de conteúdos nomeados são: (i) Instante de Tempo; (ii) Valor Instantâneo; (iii) Média; (iv) Desvio Padrão; (v) Erro Padrão; (vi) Margem de Erro; (vii) Média Menos a Margem de Erro; e (viii) Média Mais a Margem de Erro.

Para guiar a execução do aplicativo cliente, outro arquivo App.ini é utilizado e configurado adequadamente. A Figura 2.9 mostra os parâmetros definidos nesse App.ini, que são: (1) Tempo que o aplicativo espera para publicar uma oferta de serviço para outros aplicativos do cenário de troca de conteúdo; (2) Tempo de espera antes do aplicativo buscar parceiros; (3) Atraso entre a realização de tarefas periódicas, como por exemplo validação de um parceiro descoberto; (4) Atraso antes de buscar novamente por possíveis parceiros; (5) Atraso antes de executar novo *burst* de publicação de conteúdos; (6) Quantidade de conteúdos publicados por vez em cada *burst*; e (7)

Atraso entre a publicação de cada conteúdo. Os tempos são definidos em segundos.

```
1 DelayBeforePublishingServiceOffer 1
2 DelayBeforeDiscovery 3
3 DelayBeforeRunPeriodic 40
4 DelayBeforeANewPeerEvaluation 5
5 DelayBeforeANewPhotoPublish 8
6 PhotoBurstSize 25
7 DelayBeforeAPhotoPub 0.1
```

Figura 2.9: *Parâmetros do App.ini que guiarão a execução do APPClient.*

Na Figura 2.10, observa-se o diagrama de sequência utilizado pelo *APPClient* e *APPServer* quando em funcionamento conjunto em um cenário de distribuição de conteúdos de um domínio. O *APPClient* localiza o conteúdo que será processado, publicando-o e notificado-o para o *APPServer* via PSS. O *APPServer* assina, recebe, analisa e armazena esse conteúdo.

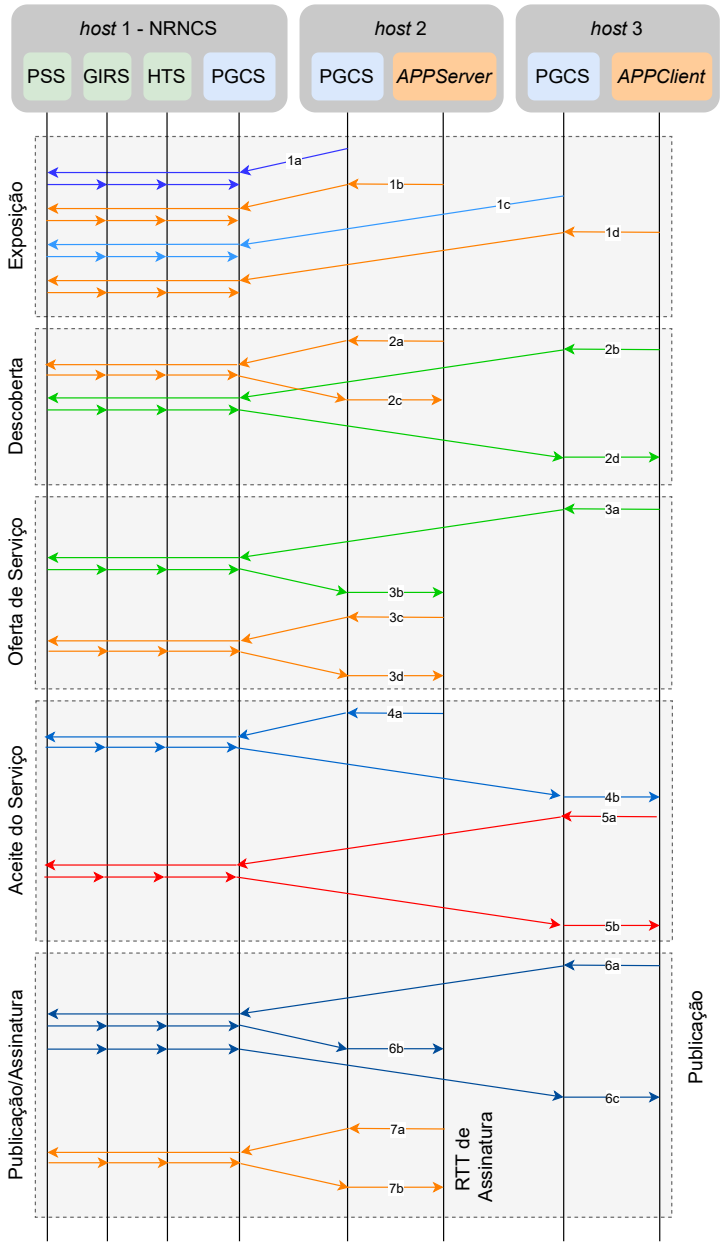
Na fase inicial de execução, os PGCSes expõem para rede seus NBs, informando os NLNes e SVNes. A exposição também inclui os NBs do PSS, GIRS e HTS que estão no mesmo SO (Passo 1a, 1c). Em seguida, o *APPServer* e *APPClient* expõem seus NBs ao *cache* de rede, com seus respectivos NLNes e SVNes (Passo 1b, 1d).

A fase de descoberta de outros aplicativos do cenário que estejam rodando no domínio inicia-se após um intervalo de tempo para evitar o congestionamento de mensagens no core NG. As palavras-chave são assinadas, possibilitando a descoberta de possíveis parceiros. Esses, podem formalizar um contrato formando uma rede confiável. O *APPClient* e *APPServer* assinam palavras-chave para descobrir um ao outro (Passo 2a, 2b). O *host* que contém o NRNCS recebe as assinaturas e na sequência realiza a entrega de assinaturas (Passo 2c, 2d). Logo, que um ID do processo de um parceiro é localizado, as instâncias assinam outras ligações que estão relacionadas ao *host* ID, SO ID, processo ID do parceiro, etc. Essa fase é definida como pesquisa recursiva entre NB.

Seguindo a sequência de execução, o *APPClient* publica para seu parceiro uma oferta de serviço ou SLA (Passo 3a). A notificação de SLA é entregue para o parceiro por meio do PSS (Passo 3b), que ao receber (Passo 3c) a oferta o *APPServer* assina a parceria entregue pelo HTS (Passo 3d). Assim que o SLA for avaliado pelo *APPServer*, uma aceitação será enviada ao NRNCS (Passo 4a). Logo, o PSS envia uma notificação para o *APPClient*, informando aceitação (Passo 4b). Para concluir a oferta de serviço, o *APPClient* assina o contrato (Passo 5a) de parceria, que será enviado pelo HTS em seguida (Passo 5b).

Depois da fase de aceitação de serviço o *APPClient* pode iniciar a publicação de foto para o *APPServer* (Passo 6a). Nesse momento o *APPServer* recebe uma notificação do PSS, informando que um novo conteúdo foi publicação (Passo 6b) e o *APPClient* recebe do HTS uma confirmação do conteúdo publicado (Passo 6c). O HTS armazena o conteúdo publicado por um ou vários clientes. Também é possível configurar topologias com vários aplicativos servidores.

Quando o *APPServer* recebe a publicação de novos conteúdos (Passo 6b), estes são analisados por meio de seu SVN, se o conteúdo é realmente novo sua assinatura será confirmada, caso contrário não deve ser efetuada (Passo 7a). A entrega do conteúdo



Legenda:

Passo 1a, 1c - Os PGCSes expõem seus NLNes, SVNes e aplicativos existentes onde ele se conecta.

Passo 1b, 1d - Aplicativo cliente e servidor expõem seus NLNes ao *cache* de rede (NRNCS).

Passo 2a, 2b - Aplicativo servidor e cliente fazem pesquisa recursiva de palavras-chave no NRNCS.

Passo 2c, 2d - Com os resultados da pesquisa recursiva os aplicativos são capazes de se encontrarem.

Passo 3a - Cliente publica oferta de serviço ao servidor.

Passo 3b - NRNCS envia notificação ao aplicativo servidor.

Passo 3c - Aplicativo servidor assina a oferta de serviço.

Passo 3d - NRNCS realiza a entrega de oferta do serviço.

Passo 4a - Aplicativo servidor envia aceitação de serviço ao cliente.

Passo 4b - Aplicativo cliente recebe notificação de aceitação de serviço.

Passo 5a - A aceitação é assinada pelo aplicativo cliente.

Passo 5b - NRNCS realiza entrega de aceitação do serviço.

Passo 6a - O aplicativo cliente inicia as publicações de fotos ao repositório somente quando o contrato for estabelecido.

Passo 6b - O NRNCS envia notificação ao aplicativo servidor sobre um novo conteúdo publicado.

Passo 6c - Aplicativo cliente recebe notificação de entrega bem sucedida.

Passo 7a - Aplicativo servidor assina somente fotos diferentes.

Passo 7b - Aplicativo servidor recebe as fotos de NRNCS.

Figura 2.10: *Diagrama de Sequência de Aplicativo de armazenamento e Repositório de conteúdo.*

publicado pelo *APPClient* ao *APPServer* é efetuada pelo NRNCS do *host* 1 (Passo 7b).

A Tabela 2.1 mostra um resumo das terminologias aplicadas a NG [44].

2.4 Evolução Histórica da Tecnologia de Contêineres

O uso de contêineres deve-se ao *chroot*. Adicionado no SO UNIX versão 7, em 1979, o *chroot* alterou o diretório raiz de um processo e seus filhos para um local no sistema de arquivo. Com essa mudança iniciou o isolamento de processo no sistema de arquivo que é apenas visível para aquela hierarquia de processos. Em 1982, o *chroot*

Tabela 2.1: *Terminologia NovaGenesis.*

Termo	**Significado**
Name	Um conjunto de símbolos (em linguagem natural ou calculada) usado para denotar entidades.
Identifier	Um nome que é exclusivo em algum escopo.
Locator	Um nome que oferece o conceito de distância no espaço.
Name Binding (NB)	É uma ligação entre nomes.
Process	Um programa de computador com vários *blocos internos* e suas *Ações*.
Block	A menor estrutura nomeada dentro de um processo. Abrange várias *ações*.
Mensage	A unidade de dados do protocolo da NovaGenesis.
CommandLine	Um *script* contendo comandos e parâmetros a serem executados no receptor.
Serviço	Sinônimo de um *Processo* destinado a troca, processamento e/ou armazenamento de informações em qualquer substrato computacional.
Hash Table (HT)	O *Bloco* que implementa uma tabela hash para armazenar ligações de nomes.
Gateway(GW)	Um *Bloco* que implementa a comunicação entre blocos e entre processos.
Proxy/Gateway (PG)	Um *Bloco* dentro do *Processo Proxy/Gateway/Controller Service* (PGCS) Serviço principal que representa demais serviços e interliga com outros PGCS.
Distributed Hash Table (DHT)	Um *Bloco* pertencente ao HTS que realiza ações específicas.
Publish/Subscribe (PS)	Um *Bloco* pertencente ao PSS que realiza ações específicas.
Indirection Resolution (IR)	Um *Bloco* pertencente ao GIRS que realiza ações específicas.
Hash Table Service (HTS)	Um *Processo* que armazena e recupera os vínculos de nomes no nível do domínio.
Generic Indirection Resolution Service (GIRS)	Um serviço que seleciona um HTS para armazenar uma ligação de nome ou conteúdo.
Publish/Subscribe Service (PSS)	Um *Processo* que faz o encontro entre associações de nome ou editores/assinantes de conteúdo.
Proxy/Gateway/Controller Service (PGCS) (PGCS)	Um *Processo* que representa alguns serviços principais em um determinado computador. Além disso, é um *gateway* para outros PGCSes.

foi adicionado ao SO *Berkeley Software Distribution* (BSD) baseado no UNIX [55].

Semelhante ao *chroot*, o FreeBSD, em 1998, começou a disponibilizar o utilitário Jail implementado por Derrick T. Woolworth na R & D Associates. O Jail incluiu melhorias no isolamento de processos adicionais para o sistema de arquivos. Ele permitiu fornecer um endereço IP para cada Jail e também instalações e configurações personalizadas de programas [56].

Como o FreeBSD Jails, o Linux VServer, lançado em 2001, é um mecanismo capaz de particionar sistemas de arquivos, adicionar endereços de rede, controle de memória e CPU no sistema. Esse controle é realizado através de níveis de isolamento no *kernel*. Cada parte isolada, ou seja, partição, é chamada de contexto de segurança (*Security Contexts*) e o sistema virtualizado dentro desse contexto é conhecido como servidor privado virtual (*virtual private server*) [55, 56].

O Solaris contêineres teve sua versão beta lançada em 2004, combinando controle de recursos do SO e separação de limite por zonas. As zonas comportam como um servidor virtual isolado em uma instância no SO. Essas zonas são dividas em dois tipos, zonas globais (*global zones*) e não-globais (*no-global zones*). A zona global é um ambiente padrão, local onde o Solaris está instalado e na qual são executadas as principais operações como instalações, inicializações e desligamentos. As zonas

chamadas não-globais são controladas e tem seus recursos limitados pela zona global [57].

A empresa Parallels em 2005 decidiu liberar o projeto de contêineres *Open Virtuozzo* (OpenVZ) para a comunidade Linux [58]. O *OpenVZ* faz isolamento, gerenciamento de recursos e fornece virtualização ao nível de *kernel*. Cada contêiner *OpenVZ* possui seus arquivos, usuários, grupo de usuários, rede, árvore de processos, separados individualmente. Devido à falta de correções e novas implementações, a evolução do projeto foi interrompida [59].

Em 2006, foi iniciado o desenvolvimento do recurso *Process Container* dentro do Google por engenheiros (Paul Menage e Rohit Seth) [60]. Um ano depois, esse desenvolvimento passou a ser conhecido por *Control Groups* ou *cgroups*. Entre as funcionalidades do *cgroups* estão, limitações e isolamentos de uso (memória, E/S de disco, rede, CPU) de grupos de processos. O *cgroups* foi adicionado no *kernel* Linux na versão 2.6.24 lançada no mês de janeiro de 2008 [60, 61].

O *LinuX Containers* (LXC) foi criado em 2008. Foi a primeira e mais completa ferramenta de gerenciamento de contêineres Linux. Permite executar várias unidades virtuais simultaneamente, semelhante a *chroots*, que são isoladas para garantir a segurança necessária, utilizando recursos disponíveis de forma eficiente. É um projeto *Open Source* patrocinado pela Canonical Ltd., empresa também responsável por distribuir o Linux Ubuntu [62]. O LXC foi inserido no *kernel* Linux desde a versão 2.6.27. Entre as ferramentas disponíveis, o LXC oferece suporte avançado de rede e armazenamento, gerenciador de contêineres e também a biblioteca *liblxc*. A *liblxc* é uma API (*Application Programming Interface*) disponível em Python 2, Python 3, Go, Lua, Haskell e Ruby, que interage com as funcionalidades do *kernel* [62, 63].

Em 2011, a CloudFoundry criou o *software* Warden usando o código fonte do LXC nos estágios iniciais. Mais tarde o código foi substituído por sua própria implementação. O Warden traz a novidade de não somente fornecer isolamento usando o Linux. Essa versão permite também isolar ambientes em qualquer sistema operacional. A execução é feita como um serviço (*daemon*) e oferece uma API de gerenciamento de contêineres, gerencia *cgroups* e o ciclo de vida dos processos [64].

O *Docker* nasceu no ano de 2013 [65] e passou a ser disponibilizado em janeiro de 2016. Seu projeto foi desenvolvido por uma empresa de PaaS, focada na infraestrutura de computação em nuvem, chamada *dotCloud*, que logo em seguida foi renomeada para *Docker Inc*. O projeto inicial usou como base o código fonte do LXC, semelhante ao que foi feito no projeto Warden e depois substituiu a biblioteca *liblxc* por uma própria, a *libcontainer*. O *Docker* disponibiliza um ecossistema completo para o gerenciamento de contêineres [65].

A CoreOS em 2014 também iniciou um projeto semelhante ao *Docker*, chamado de *Rocket* ou *rkt*. O aplicativo *Rocket* tem como objetivo principal fornecer maior segurança em comparação ao projeto *Docker*. A interface principal do *Rocket* inclui um aplicativo executável em vez de um *daemon* em execução em segundo plano, como no *Docker* [62].

Um dos gigantes de tecnologia no mundo decidiu incluir o suporte a contêineres ao seu sistema operacional. A Microsoft em 2015 incluiu na versão Microsoft *Win-*

dows 2016 as APIs e o cliente *Docker*. São dois tipos distintos de uso de contêineres no *Windows*: contêineres *Hyper-V* e *Windows Server Containers*. No tipo *Hyper-V*, o contêiner não usa o *kernel* do *host* hospedeiro. Nele, é criada uma máquina virtual otimizada para cada contêiner. Já com *Windows Server Containers*, todos os contêineres são executados fazendo uso do *kernel* do sistema operacional do hospedeiro. O isolamento de aplicações é realizado através da tecnologia de isolamento de processos e *namespaces* [19].

Como forma de padronização aos formatos e execução de contêineres, no ano de 2015 a CoreOS e o *Docker* criaram a *Open Container Initiative* (OCI) [66]. Logo, outras empresas (Google, Amazon, RetHat, Oracle, e outras) aderiram à iniciativa. O OCI fornece duas especificações: a de imagem (*image-spec*) e a de execução (*runtime-spec*). Na execução, estão os procedimentos para realizar o *download* de uma imagem OCI e como desempacotar um sistema de arquivos. A especificação de imagem descreve os passos para criar uma imagem, comandos, argumentos, variáveis de ambiente para iniciar a ferramenta *Docker* ou outras no *host* desejado [66]. Uma ferramenta interessante de gerenciamento de contêineres é o *Kubernetes*, também conhecido como K8s, é um sistema de código aberto para automatizar a implantação, escalonamento e gerenciamento de aplicativos em contêineres [67].

2.5 Definição de Contêiner

Um contêiner é uma instância no sistema operacional hospedeiro com bibliotecas e configurações necessárias para fazer uma aplicação funcionar. Assim, não necessitando um sistema operacional completo.

O uso de contêineres tem como objetivo, gerar unidades computacionais gerenciáveis e auto-suficientes em uma infraestrutura de computação para executar aplicações [68]. Os contêineres isolam suas aplicações e dependências para não causar conflitos com aplicativos de outros contêineres [25]. A execução de cada contêiner tem seu próprio espaço no núcleo do *kernel* do sistema operacional [69].

Pahl explica que um contêiner é criado a partir de uma imagem base [68]. À medida que se acrescenta novas aplicações, são criadas camadas gerando assim uma nova imagem, que pode ser transportada facilmente.

Trabalhar com contêineres possibilita utilizar mecanismo para criar e executar imagens, importar ou exportar imagens de repositório público e/ou privado. Nos repositórios encontram-se milhares de imagens padrão e também imagens com aplicativos que já foram instalados, configurados, e disponibilizados para comunidades de desenvolvedores, como por exemplo, imagens que contém as aplicações MySQL, MongoDB, Zabbix, e Node.js já estão prontas para uso. Com a API de contêineres, os desenvolvedores gerenciam a criação, definição, distribuição de imagens e também o controle de execução, reinicialização, parada, finalização e exclusão dos contêineres [63, 68].

As aplicações no interior do contêiner dependem da necessidade e o planejamento da infraestrutura. Uma boa prática é utilizar uma aplicação por contêiner, pois isso permite implementar novas pilhas de serviço referente a essa aplicação, aumentando

a segurança devido a possíveis ataques, dado que cada contêiner possui sua própria chave de acesso. E ainda, se houver a necessidade de parar ou reiniciar o contêiner, essas operações não impactam diretamente outras aplicações que estão em diferentes contêineres [65].

A Figura 2.11 mostra o *kernel* do sistema operacional hospedeiro e as camadas da imagem de um contêiner com algumas aplicações instaladas. Assim o contêiner que inicialmente tinha uma imagem base, como por exemplo da distro Linux Ubuntu, adicionou uma camada de um servidor de e-mail, outra para um servidor *Web* e por último uma camada gravável aguardando novas instalações. Os itens *Storage Drivers, Namespaces, Networking* e *cgroups* são detalhados nas Subseções 2.8.1, 2.8.2, 2.8.3 e 2.8.4, respectivamente.

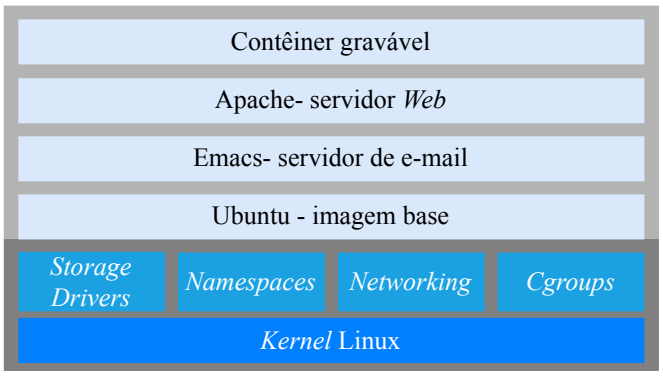


Figura 2.11: *Camadas de uma imagem de contêiner.*

2.6 Comparação entre Contêineres e Máquinas Virtuais

Basicamente, um sistema operacional é responsável por disponibilizar a interface dos programas facilitando assim a comunicação dos recursos físicos (*hardware*) do *host* que os executam. O avanço exponencial dos recursos tecnológicos e o poder computacional nos dias atuais, possibilitaram o uso de técnicas de virtualização. Dessa forma, os recursos computacionais (CPU, memória, largura de banda de rede, armazenamento) excedentes são melhor aproveitados por fatias virtuais da máquina [70].

Na definição de Morais [70], a virtualização [71] é a capacidade de abstrair os recursos de *hardware* utilizando o mesmo equipamento para gerar vários sistemas operacionais, cada um com seus próprios recursos. Os componentes básicos para virtualização são: a máquina virtual e o seu gerenciador (*hypervisor*). Com o *hypervisor* são executadas as máquinas virtuais e realizada a gerência de camadas entre elas e os recursos físicos. A máquina virtual tem suas próprias características, trabalhando como um computador individual (virtual) [70].

Segundo Mouat [25], os contêineres são virtualizações sem a necessidade de um sistema operacional completo, tornando assim sua execução mais rápida. Apesar de a primeiro momento os contêineres parecerem iguais à virtualização de VMs, as virtualizações diferem. O uso de contêineres dispensa o *hypervisor*, imagens diferen-

tes e *kernel* independente para cada contêiner. A VM emula uma máquina totalmente exclusiva com seus recursos computacionais, diferente de um contêiner que facilita a transferência de aplicações em contêiner de uma infraestrutura para outra, de forma ágil.

Bernstein [57] explica que, tanto máquinas virtuais e contêineres, utilizam da interface de um SO quando executam as aplicações. Na máquina virtual é necessária a instalação completa de uma versão do sistema operacional desejado. Já os contêineres usam pedaços isolados, ou seja, fatias do sistema em execução. Com isso as implementações de serviços nos contêineres resultam em imagens extremamente pequenas quando comparadas às imagens de VMs [65].

A instalação de sistema operacional com o uso de VM gera uma imagem grande, dificultando o envio da VM para outros ambientes, sem contar com as possíveis incompatibilidades dos *hypervisores* [68]. Em comparação, os contêineres mostram maior flexibilidade, eficiência dos recursos e o gerenciamento mais prático do ambiente virtual [65, 72].

A Figura 2.12 mostra uma comparação em camadas entre máquina virtual com *hypervisor* e virtualização com contêiner. O uso de máquinas virtuais é interessante para aplicações a serem instaladas que exigem distribuições e versões diferentes do SO. Em contraste, os contêineres utilizam o mesmo *kernel* do SO na instalação dos aplicativos, reduzindo consideravelmente o *overhead* gerado em comparação às VMs.

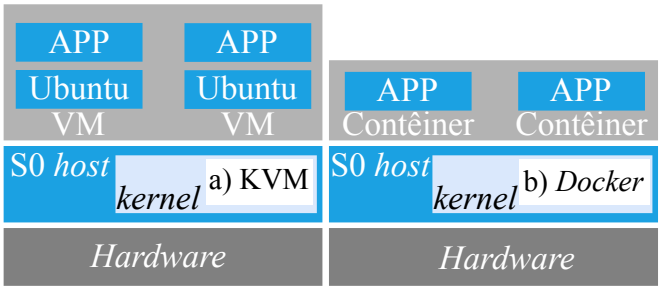


Figura 2.12: *Comparativo entre: a) KVM Hypervisor e b) Docker Engine.*

2.7 Docker

O *Docker* é baseado em LXC. Uma plataforma aberta para comunidade de desenvolvedores e administradores de rede, criado e mantido pela empresa *Docker Inc*, consiste de um ecossistema que permite construir, compartilhar e executar aplicações distribuídas em contêineres. O *Docker Hub* é um repositório de registros de armazenamento e distribuição de imagens de máquinas computacionais. No ecossistema *Docker* existem várias ferramentas, dentre elas a gerência e a orquestração de *clusters* de contêineres, recurso chamado de *Swarm* [65].

O crescimento da utilização de contêineres no mundo deve-se em maior parte ao *Docker*, que por meio de uma API amigável e imagens de fácil mobilidade, simplificou o uso da tecnologia. A curva de aprendizado do LXC requer muito estudo e trabalho manual, ou seja, requer conhecimento avançado em Linux e também nos comandos utilizados para manipular contêineres LXC. Todo o processo de instalação e configuração

tem que ser minunciosamente feito pelo usuário. No entanto, a instalação do *Docker* é bem simples, com apenas alguns comandos um usuário pode começar a trabalhar com contêineres *Docker*. Além da facilidade de uso, ele conta com a contribuição de múltiplos colaboradores e um repositório repletos de imagens para *download* [65].

Quando se cria um ambiente isolado, esse ambiente é um contêiner, o qual é guardado dentro de uma imagem e que por sua vez pode ser guardada em um repositório público ou privado. Uma vez isolado, um ambiente torna-se uma outra máquina, podendo se tornar um servidor de e-mail, servidor *Web*, etc. A ideia de deixar o ambiente em uma imagem é a possibilidade de importar ou exportar essa imagem para qualquer infraestrutura [65].

2.8 Arquitetura Linux Explorada com o Docker

Os recursos do *kernel* Linux que o *Docker* faz uso no provisionamento da tecnologia de contêineres são apresentados na Figura 2.13. Esses recursos são divididos em grupos de blocos funcionais. Grupo *Storage*, *Namespaces*, *Networking* e *cgroups* [65, 73, 74].

Para criar contêineres em *Docker*, vários recursos do *kernel* do SO Linux são utilizados. Esses recursos estão descritos nas subseções a seguir.

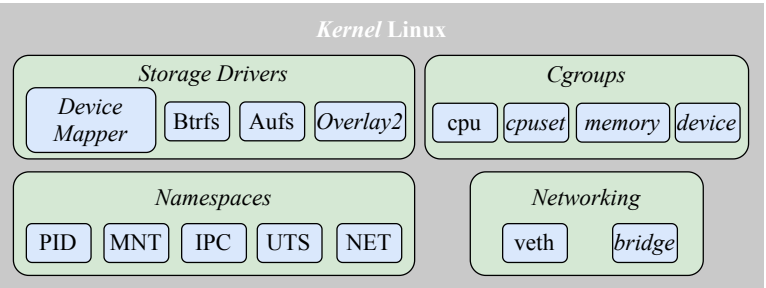


Figura 2.13: *Componentes do kernel Linux utilizados pelo Docker.*

2.8.1 *Storage Drivers*

O *Docker* armazena os dados usando o mecanismo *Union File System* (UnionFS) do SO, guardando-os em uma imagem. O *UnionFS* é um sistema de arquivos em camadas sobreposta permitindo que os desenvolvedores implementem um conjunto de camadas que são apresentadas como um único diretório virtual aos usuários. As imagens são arquivos que recebem parâmetros de configuração, binários de aplicações e capturam de forma instantânea as instalações e configurações do contêiner. Devemos salvar as alterações do contêiner em uma imagem para uso futuro. Assim, mesmo reiniciando ou desligando o contêiner não perdemos as instalações e/ou configurações [75]. O *Docker* pode usar diversas variantes do sistema de arquivos, incluindo:

- **AUFS**: Foi o primeiro controlador de armazenamento em uso com *Docker*. Como resultado, tem uma história longa e estreita com *Docker*, sendo considerado pelos usuários como muito estável. Tem muitas implementações do mundo físico e forte apoio da comunidade de desenvolvedores. O AUFS tem várias carac-

terísticas que o tornam uma boa escolha para *Docker*. Esses recursos permitem tempos de inicialização de contêiner menores, uso eficiente de armazenamento em disco e em memória;

- **Btrfs**: Tem recursos avançados como *snapshotting* na camada de sistema de arquivos. É muito rápido quando comparado a outros sistemas de arquivos usados pelo *Docker*. A desvantagem do *Btrfs* é que ele não permite compartilhamento de *cache* de página e não é suportado pelo SELinux;
- **DeviceMapper**: É um *framework* baseado no *kernel* que está subjacente a muitas tecnologias avançadas de gerenciamento de volume no Linux; e
- **Overlay2**: Até o momento do desenvolvimento deste trabalho o sistema de arquivos mais moderno é o *overlay2*. O *Docker* por padrão utiliza este sistema. Para o uso de outros anteriores (AUFS, Btrfs, DeviceMapper), pacotes adicionais devem ser instalados nas versões posteriores do Ubuntu 14.4 no *kernel* 3.13. O *Overly2* é semelhante ao AUFS, mas com implementação mais simples e ainda mais rápido.

2.8.2 *Namespaces*

As tecnologias de contêineres fazem uso de *Namespaces* no isolamento de processos com os recursos do *kernel* do Linux. Assegurando que cada contêiner enxergue apenas o seu próprio ambiente e não afete ou tenha acesso a processos em execução dentro de outros contêineres. Além disso, os *Namespaces* fornecem acesso restrito aos sistemas de arquivos como, por exemplo, o *chroot*, por ter uma estrutura de diretório para cada contêiner [55, 56]. Alguns dos *Namespaces* que o *Docker* utiliza são:

- **PID**: Isolamento de processo para cada contêiner;
- **MNT**: Gerenciamento de pontos de montagem para unidades de armazenamento;
- **IPC**: Gerencia o acesso a recursos entre processos;
- **UTS**: Isola espaço no *kernel* Linux e versão de identificadores; e
- **NET**: Gerencia interfaces de rede.

2.8.3 *Networking*

O *Docker* utiliza recursos de redes para fornecer isolamento completo para os contêineres. Os três tipos de redes são: rede *default*, redes definidas pelo usuário e redes de sobreposição. (i) na rede *default* todos os contêineres são executados seguindo o padrão da rede *bridge* (172.17.0.0/16); (ii) com as redes definidas pelo usuário, através de comandos determina-se a classe (A, B ou C) de IPs, o nome da rede e o *range* de IPs; (iii) em redes de sobreposição, quando é criado um serviço que usa uma rede sobreposta, o nó do gerenciador estende automaticamente a rede de sobreposição para os nós que executam tarefas de serviços por diferentes redes [76, 77].

Os contêineres utilizam redes do *Docker*. Duas interfaces são criadas em dois *Namespaces* diferentes. Uma interface reside no interior do próprio contêiner e uma outra interface no *host* hospedeiro. A interface interna do contêiner é chamada de **eth0** e no sistema hospedeiro é dado um nome aleatório, tal como **vethxxx**. As interfa-

ces são ligadas através de uma *bridge* (Docker0) no *host* hospedeiro para permitir a comunicação entre contêineres e rotear pacotes [78, 79]. A Figura 2.14 ilustra a rede *Docker*.

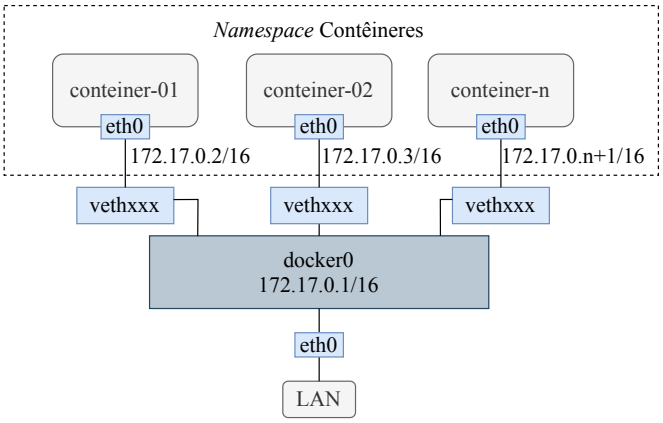


Figura 2.14: *Rede Docker em um computador físico.*

2.8.4 Cgroups

O *cgroups* (grupos de controle) [73] é um sistema de arquivos empilhável que fornece mecanismo do *kernel* para isolar recursos que incluem memória, CPU e E/S de disco. A estrutura do *cgroups* faz parte do *kernel* do Linux desde a versão 2.6.24. e permite criar camadas de forma dinâmica com diferentes sistema de arquivos ou diretórios usando o mesmo SO [73, 74].

O *Docker Engine* em *Linux* também faz uso da tecnologia chamada *cgroups*. Com o uso do *cgroups*, o *Docker* consegue isolar as aplicações e gerenciar apenas os recursos desejados. Permite compartilhar recursos de *hardware* disponíveis para contêineres e, se necessário, configurar limites e restrições. Alguns dos *cgroups* que o *Docker* utiliza são: CPU, *CPUSET*, *MEMORY* e *DEVICE*. O grupo CPU permite configurar de forma proporcional aos contêineres o uso de CPU da máquina física. Com o *CPUSET* cria-se máscaras de CPU executando *threads* dentro de um contêiner. Usando o *MEMORY* define-se limites de memória *RAM*. Por fim, o *DEVICE* configura quais dispositivos podem ser usados dentro do contêiner [65, 73, 74].

2.9 Considerações Parciais

Neste Capítulo, foram apresentados os conceitos fundamentais para o entendimento da proposta do trabalho. Assim sendo, foram discutidas as limitações da Internet atual voltadas à resolução de nomes e distribuição de conteúdos. A fim de solucionar esses problemas, foi apresentada a arquitetura de Internet do Futuro NG, com ênfase nos aplicativos *NBSimpleTestApp*, *APPClient* e *APPServer*, responsáveis por desenvolver novas propostas capazes de atender a essas limitações mencionadas anteriormente. Além disso, para a avaliação de desempenho da proposta NG e seus respectivos serviços, foram apresentados os conceitos fundamentais para o entendimento da tecnologia de contêiner *Docker*.

Associado ao objetivo principal neste trabalho, avaliar a arquitetura NG como proposta de solução às limitações da Internet. Utilizando aplicativo de resolução de nomes e aplicativo de distribuição de conteúdos em *cache* temporário. Será proposto o ambiente AAAIF, a fim de mudar a forma de experimentação de arquiteturas ICNs, especialmente NG. Ele será planejado para executar os serviços NG dentro de contêineres *Docker*, máquinas físicas e VMs. Como principais benefícios, o AAAIF abre possibilidades de experimentos em escala (milhares de contêineres com serviços NG) e rápida migração nos ambientes físicos (de laboratório) utilizados. Assim, como nuvens e plataformas de *testBed*. O AAAIF permitirá ao usuário definir: (i) o grafo de conectividade na rede; (ii) a topologia utilizada, sem nenhuma restrição entre os *hosts*; (iii) instalação de pacotes e configurações; (iv) ordem de execução dos aplicativos; (v) onde e quando os aplicativos distribuídos devem ser executados; (vi) e coleta dos resultados.

Capítulo 3

Trabalhos Relacionados

Esse Capítulo tem como propósito apresentar os trabalhos focados em novas propostas para as limitações de nomeação, resolução de nomes, distribuição de conteúdo com *cache* de rede auto-organização de serviços de mídia. Inclui também uma discussão sobre como essas novas propostas foram avaliadas experimentalmente, através de uma pesquisa detalhada relacionando os ambientes de experimentação das arquiteturas de Internet do Futuro. Na pesquisa bibliográfica foram incluídos trabalhos de arquiteturas de ICN, que utilizam o modelo publica/assina de comunicação entre processos e como eles são tipicamente experimentados/avaliados em outros trabalhos. Na Seção 3.1 estão os critérios utilizados na pesquisa. Os trabalhos relacionados são apresentados nas Seções 3.2, 3.3, 3.4, 3.5, 3.6 e 3.7. Por fim, na Seção 3.8 estão presentes as oportunidades de pesquisa e considerações parciais.

3.1 Metodologia de Pesquisa e Escolha dos Trabalhos Relacionados

Para a pesquisa dos trabalhos relacionados foi utilizado o *WebSite Google Scholar* e o *WebSite* de Periódicos CAPES/MEC. Estes são responsáveis por indexarem grande quantidade de periódicos de bases como *IEEE-Xplorer, ACM - Digital Library* e *Springer*. As palavras chaves utilizadas para pesquisa foram: *"Information Centric Networking"*, *"Future Internet Architecture"*, *"Publish/Subscribe"* e *"Evaluation / Experimentation ICNs"*. Os resultados das pesquisas foram analisados com o objetivo de encontrar trabalhos que explorassem FIAs com o paradigma ICN. Como elas implementam nomeação, resolução de nomes, distribuição e recuperação de conteúdo e também quais tecnologias e plataformas são utilizadas na avaliação dessas propostas.

Ao todo foram analisadas 80 referências como: artigos científicos, sites oficiais de projetos, livros acadêmicos e dissertações relacionadas a este trabalho. Entre os temas mais relevantes, foram selecionadas 33 referências com maior profundidade e relevância com às palavras chaves utilizadas. Foram analisados os critérios de relevância através da leitura do resumo, introdução e conclusão dos artigos, classificando-os para posterior leitura completa dos mesmos. Atualmente são poucos projetos de FIAs que estão em constante evolução e que, publicam artigos com relevância para comunidade

científica. Neste trabalho foram elencadas 6 FIAs.

3.2 CCN

A *Content Centric Network* (CCN) é uma alternativa ao uso tradicional de redes IP, que possui o paradigma de conexão baseado em *host-centrism*, ou seja, a conexão entre *hosts* é base de projeto da arquitetura. Já a CCN, apresenta uma comunicação orientada aos nomes dos conteúdos de interesse, buscando desta maneira aprimorar o suporte a mobilidade e ao *multicast*. Ela cobre o conceito apresentado por Van Jacobson de transição de uma arquitetura de rede centrada na comunicação entre máquinas, para uma arquitetura centrada na troca de conteúdos nomeados. Dessa forma, a arquitetura passa a nomear o conteúdo. Assim, o conteúdo como por exemplo um vídeo pode ser alcançado pelo nome, evitando a busca por nomes de servidores que os hospedam. Um vídeo pode receber um nome no endereço "/inatel.br/video/elcioNG_p1.mp4", onde o nome seria elcioNG, o p1 primeiro seguimento. Algum aplicativo poderia expressar o interesse neste vídeo e gerenciar a segmentação [40].

Dentro deste contexto, a CCN recomenda uma mudança na semântica dos serviços de rede atual. Em sua concepção, defende que a rede abandone o conceito de envio de pacotes de informação a um determinado endereço (IP) e passe a encaminhar os pacotes de informação através da identificação fornecida por um nome (nomeação) de um conteúdo desejado. Com isso, os autores da ideia defendem que as redes CCN permitem resolver uma gama de problemas, aos quais incluem comunicação fim-a-fim, distribuição de conteúdos e resolução de nomes. A arquitetura utiliza o nome de conteúdo como entrada em uma tabela de interesse (no roteador) para solicitações pendentes, enquanto o DNS segue um modelo hierárquico de nome de servidores distribuídos no mundo, buscando o conteúdo desejado [40, 80, 81].

Na rede CCN, a comunicação apresenta troca de dois tipos de pacotes: pacotes de interesse (*Interest*) e pacotes de dados (*Data*). Ambos os pacotes são providos de um nome que identifica uma porção de informação que pode ser transmitida como um pacote pela rede. Durante a comunicação em uma rede CCN, um usuário, que queira acessar determinada informação, realiza a requisição por meio do envio de um pacote de interesse à rede. Este pacote carrega o nome da informação que o usuário requisitante deseja acessar [80].

O pacote de interesse é encaminhado pelos roteadores no núcleo da rede até atingir a entidade que possua armazenada a informação requisitada. Quando o pacote de interesse atinge a entidade que contém armazenada a informação requisitada pelo usuário, a entidade responde enviando pacote de dados. O pacote de dados enviado possui o nome, o conteúdo e a chave de assinatura do produtor.

Os roteadores de rede CCN possuem três componentes de dados para realizar o envio dos pacotes de interesse e informação pela rede. São eles: *Pending Interest Table* (PIT), *Forwarding Information Table* (FIB) e *Content Store* (CS) [26]. A Figura 3.1 mostra o processo de pacote de interesse e de dados em um nó CCN.

- **PIT**: Dispõe armazenados todos os pacotes de interesse repassados pelo roteador a outras entidades de rede e que ainda não foram atendidos. Cada entrada da

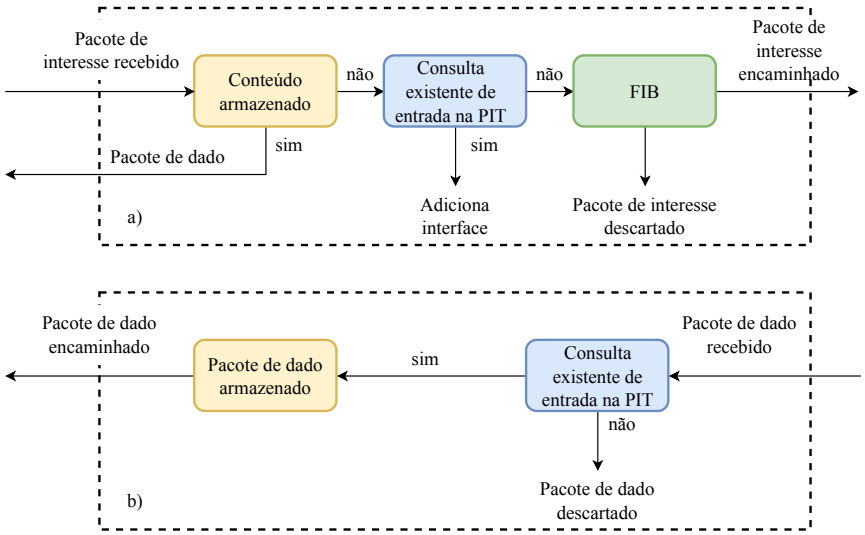


Figura 3.1: *Processo de encaminhamento em um nó CCN (a) encaminhamento de pacotes de interesse e (b) encaminhamento de pacotes de dados. Adaptado de [40].*

tabela PIT vincula o nome da informação ou dados com as interfaces de entrada e saída;

- **FIB**: Possui informações úteis no processo de encaminhamento de pacotes de interesse que não podem ser atendidos pelo roteador que o recebeu;
- **CS**: Armazena temporariamente todos os pacotes de dados recebidos pelo roteador. O armazenamento tem como objetivo atender os pacotes de interesse futuros que requerem a mesma informação. Com isso, o roteador é capaz de atender a um pacote de interesse recebido sem ter que buscar a informação na fonte primária.

Quando um roteador CCN recebe um novo pacote de interesse, primeiramente consulta a CS para verificar se a informação requisitada se encontra armazenada por ele. Se ela estiver armazenada no roteador, então encaminha o pacote de dados pela interface ao qual recebeu o pacote de interesse. Caso contrário, o roteador realiza uma consulta à tabela PIT para verificar a existência de uma entrada para o pacote de interesse em questão. Se a entrada já existe, o roteador armazena a interface ao qual recebeu o pacote de interesse. No caso de inexistência de tal entrada, o roteador registra uma nova entrada na tabela PIT e encaminha o pacote de interesse ao produtor do conteúdo [26].

No artigo [80] apresenta-se uma plataforma de *testbed* com objetivo de avaliação FIAs, chamada de *Future Internet Testbed with Security* (FITS). Ela se aplica a arquitetura CCN. O projeto CCN foi avaliado através da plataforma de experimentação FITS criando redes virtuais isoladas [80]. Foi utilizado para tal, o *hypervisor* XEN para criar roteadores virtuais em VMs na rede FITS e o protocolo *OpenFlow* foi utilizado para o gerenciamento de fluxo de dados.

3.3 XIA

A *eXpressive Internet Architeture* (XIA) é um projeto de FIAs financiado pela agência *National Science Foundation* (NSF). É considerada expressiva devido aos pacotes condizerem com a intenção da aplicação. Uma arquitetura flexível, baseada em nome, onde as existências relevantes são classificadas como *Principals Types* [7].

A arquitetura XIA tem como objetivo melhorar a evolução e a confiabilidade da rede. A fim de atingir seu objetivo, a XIA define uma rede única que oferece suporte intrínseco para a comunicação entre múltiplos *Principals Types* de comunicação, nomeadamente: *hosts*, conteúdos e serviços, sendo capaz de evoluir as suas funcionalidades para acomodar novos *Principals* que surgirem no futuro. O seu projeto é baseado em um gargalo, como a atual arquitetura de Internet, mas esse gargalo é flexível e pode evoluir para acomodar novos modelos tecnológicos, armazenamento e computação à medida que surgem [7].

O projeto XIA é desenvolvido em volta de três princípios centrais nomeados como: (i) segurança intrínseca, a fim de garantir a integridade e autenticidade das comunicações; (ii) *Principal Types* que suportam vários protocolos que poderão ser usados para expressar diretamente a intenção de acessar funcionalidades específicas (roteamento); e (iii) endereçamento flexível, baseado na solução *Scion* [82]. Este protocolo de seleção de caminho suporta benefícios de segurança significativos em relação às abordagens tradicionais de encaminhamento via endereço de destino [83]. A seguir descrevemos sobre alguns *Principals Types*.

- ***Autonomous Domain* (AD)**: Um representante do domínio. Identifica sistemas autônomos da rede. Fornece hierarquia para outros *Principals*, ou seja, fornece controle sobre roteamento;
- **HID**: Identificador único de algum dispositivo ou *host* na rede, ou seja, é imutável independentemente da interface usada ou rede à qual um dispositivo está conectado;
- ***Service Identifier* (SID)**: Permite verificar a identidade do serviço e obter o serviço de onde o conteúdo foi gerado. Representa um serviço de aplicativo executando em um ou mais *hosts* na rede; e
- ***Content Identifier* (CID)**: Permite identificar o tipo de conteúdo na rede, por exemplo, uma foto, vídeo, música, ou outro conteúdo existente independente da sua localização.

O *eXpressive Internet Protocol* (XIP) é independente do *Principal Type*, que permite a comunicação base da rede XIA. Define um formato de endereço, cabeçalho do pacote e a lógica de processamento dos pacotes associados. Uma função especial do XIP é fornecer o formato flexível para utilizar caminhos alternativos de um destino. No entanto, a intenção de um endereço XIP pode não ser suportada por todos os roteadores. Assim, a XIA expressa suas intenções de roteamento hierarquicamente usando três mecanismos básicos de construção: intenção, retorno e escopo. A intenção define o propósito do pacote, o retorno ou noção de *fallback* é utilizado quando algum roteador não saiba encaminhar pacote ou não entenda o *Principal Type*. Por exemplo, sendo um SID com impossibilidade de prosseguir, a XIA possibilita a tentativa em seu

escopo usando AD, HID, CID ou outro *Principal Type* [7].

A XIA representa seus endereços utilizando *Directed Acyclic Graph*s (DAGs) para facilitar a implementação de roteamento hierárquico de mecanismos de intenção, retorno e escopo. O DAG é alocado no cabeçalho do pacote, que armazena o endereço de destino. A Figura 3.2 mostra os elementos XIA, onde podemos identificar a forma de comunicação entre eles. Quando os dados chegam na entrada, o roteador executa um processo específico da fonte baseando no *eXpressive Identifier* (XID) do DAG desta fonte. Para encaminhar ao próximo destino, o roteador verifica o XID do próximo salto. Caso este for suportado, seleciona um *Principal Type* e faz o encaminhamento da informação. Caso contrário, utiliza um encaminhamento com rota alternativa, replicação ou desvio. Por fim, o pacote pode ser descartado ou o nó pode ser considerado inacessível gerando um erro [7].

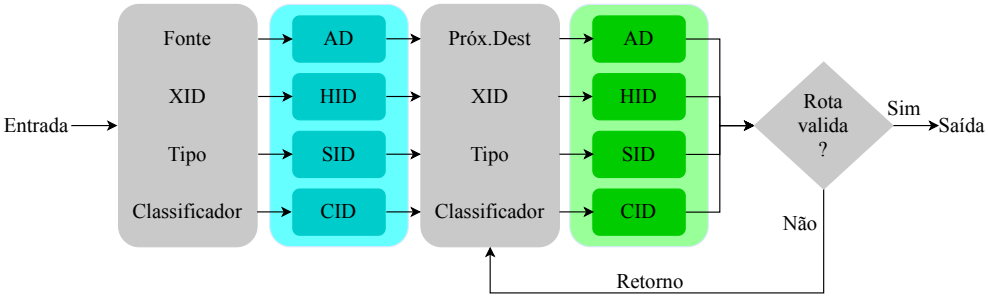


Figura 3.2: *Diagrama simples com os elementos de processamento do pacote XIP. Adaptado de [7].*

O processamento paralelo dos pacotes acelera o encaminhamento XIP. A estrutura dos ADs e HIDs são auto-certificáveis: eles são gerados por *hash* da chave pública de um domínio autônomo ou de um *host*, vinculando de forma imperceptível a chave ao endereço. Fazendo com que os pacotes encaminhados para elas possam ser processados em paralelo. Enquanto, o SID e o CID são estruturas que dependem de poucas informações e sendo processadas de forma oportunísticas. Por fim, a XIA utiliza um serviço de resolução de nomes para traduzir os endereços XIP em nomes legíveis, associando também nomes de serviços legíveis a identificadores de serviços [7, 83].

Nos experimentos do artigo Chaib et al. [84] foram utilizados contêineres, criando uma topologia em estrela. Para o cenário experimental foram desenvolvidos cinco tipos de imagens de contêiner *Docker* (cliente, *gateway*, roteador cliente, roteador *gateway* e servidor), sendo cada uma com função específica. O objetivo do trabalho foi avaliar o comportamento da arquitetura XIA interoperando com o Contiki utilizando o simulador Cooja [84, 85].

3.4 RINA

A *Recursive InterNetwork Architeture* (RINA) [86] é uma proposta de arquitetura de Internet do Futuro *clean slate* que parte do princípio de que rede é apenas comunicação entre processos *Inter-Process Communication* (IPC). A rede deve fornecer essencialmente um bom serviço IPC às aplicações. A rede fornece um recurso pelo

qual os processos de aplicativos em vários sistemas podem se comunicar, popularizando o modelo de IPC local. Comparada com o modelo atual da Internet, a arquitetura RINA é baseada em um único tipo de camada, que é repetida recursivamente várias vezes, de acordo com os requisitos do *designer* de rede normalmente em escopos distintos. Essa camada é nomeada como recurso de IPC distribuído *Distributed IPC Facility* (DIF). É um aplicativo distribuído que fornece serviços IPC para os aplicativos distribuídos acima dentro de um determinado escopo. Esses aplicativos podem ser outros DIFs ou aplicativos regulares. A rede RINA nomeia aplicativos dentro do contexto do *Inter-DIF Directory* (IDD). Por exemplo, um navegador WEB se comunica com outros aplicativos por meio de IDDes vizinhos [87].

Um DIF é formado por diversos *IPC Process*s (IPCPs). O processo convencional do IPC consiste em elementos dedicados à transmissão de dados, controle de transmissão de dados ou gerenciamento de camadas. Dentro de um DIF, nomeação, roteamento e outras funções de rede são personalizáveis, adaptando suas funções para um conjunto de aplicações ou operar de maneira otimizada em um ambiente, portanto desacoplando políticas de funções. Todos os DIFs fornecem a mesma interface, independentemente de sua classificação em relação a outros DIFs. Essa interface permite que os aplicativos solicitem e gerenciem IPCP. Cada IPCP pode associar com um novo DIF e desvincular do anterior [8, 86].

Na RINA os eventos de gerência são assinados com base no paradigma do modelo publica/assina. As assinaturas na FIA RINA, são gerenciadas por meio do *daemon Rina Information-Base* (RIB). O RIB fornece todas as funções de gerenciamento de camadas (registro, gerenciamento de *namespace*, alocação de fluxo, alocação de recursos, coordenação de segurança, etc.) com os meios para interagir com os RIBs dos IPCPs de mesmo nível [8].

A Figura 3.3 mostra a comunicação da rede RINA utilizando dois *hosts* e alguns roteadores. O processo de interação entre roteadores com um conjunto de IPCPs nos DIFs é expandido à medida que novos nós são adicionados. Demostramos de forma resumida três funções do *daemon* RIB, não limitando a somente estes recursos dentro do IPCP. Em [41] outros recursos podem ser explorados.

Existem quatro implementações do RINA: Alba, TRIA *network systems LLC*, *Boston university* e a RINA investigou uma alternativa ao TCP/IP (IRATI) [8], que foi financiado pelo FP7 na Europa.

Com relação à resolução de nomes, RINA utiliza os IDDs para aplicativos que não estejam registrados na base local da DIF. De forma semelhante ao DNS, realiza busca em um *namespace* hierárquico. Um ou vários DIFs são selecionados para suprir as necessidades de comunicação. Se o DIF não atende às demandas, um processo se unir a um DIF existente para alcançar um certo serviço/processo desejado. Em [87] os autores apresentaram um exemplo de resolução de nomes na rede RINA. Então, a arquitetura resolve o nome do aplicativo, e ele se encarrega de disponibilizar o conteúdo.

No trabalho de [89] foram apresentados experimentos utilizando a implementação da RINA IRATI. O cenário com três VMs criadas com o *hypervisor* XEN hospedadas na mesma máquina física teve como objetivo avaliar a degradação do RTT e o *goodput* ao usar políticas criptográficas. No entanto, não avaliaram escalabilidade.

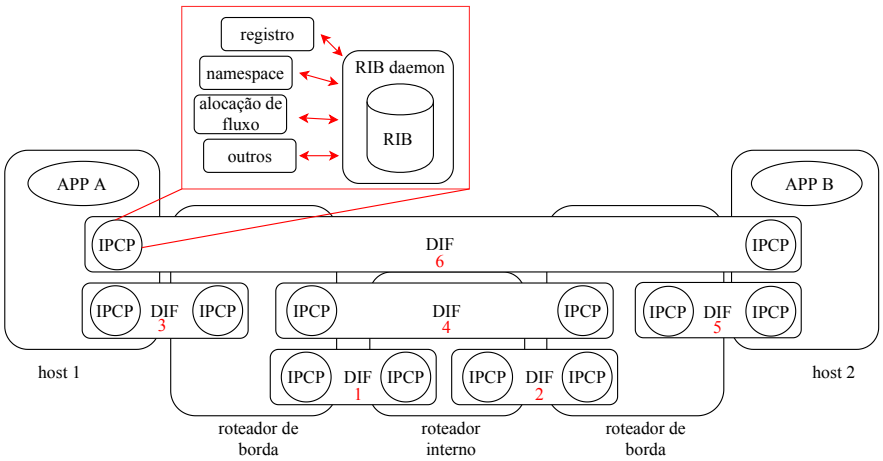


Figura 3.3: *Um exemplo simples de camadas da RINA. O escopo das camadas 1, 2, 3 e 5 é local para um link físico. O escopo para as camadas 4 e 6 é maior e abrange vários nós. Adaptado de [88].*

3.5 SAIL

A *Scalable and Adaptive Internet Solutions* (SAIL) [90], foi financiada pelo FP7 na Europa. A arquitetura pode recuperar o conteúdo de duas maneiras: a primeira é através da resolução de nomes e a segunda é através do roteamento baseado em nomes, que permite a adaptação em diferentes ambientes de rede. De acordo com o modelo usado, a fonte publica o *Name Data Object* (NDO) registrando um link de nome de um *Name Resolution Service* (NRS) ou anunciando informações de roteamento em um protocolo de roteamento. Na SAIL, um nó que contém uma cópia do NDO (incluindo cache local) pode optar por registrar sua cópia no NRS para adicionar um novo link de nome. Como alternativa, o destinatário pode enviar diretamente uma solicitação *GET* com o nome NDO, que será encaminhada para uma cópia disponível do NDO usando o roteamento baseado em nome [90].

A SAIL engloba três componentes independentes, a saber: *Network Information* (NetInf), *Cloud Networking* (CloNe) e *Open Connectivity Service* (OConS). O NetInf é uma área do projeto SAIL dedicada ao estudo de ICN, no qual os roteadores fornecem encaminhamento, publicação, armazenamento local, pesquisa e recuperação de NDOs. OConS expõe recursos de rede heterogêneos como serviços de conectividade aberta para CloNe, NetInf ou outras estruturas. O CloNe integra computação em nuvem e rede virtual para lidar com o ciclo de vida complexo de máquinas virtuais e seus requisitos de rede. No entanto, não há nenhuma menção explícita no suporte CloNe para comunicação de serviço baseada em nome. Os serviços podem ser nomeados no NetInf, mas o suporte conjunto do Clone e NetInf para o SVN não está claro na documentação. O CloNe cria *Flash Network Slides* (FNS) para fornecer rede virtual customizada sob demanda entre diferentes nuvens de provedores usando o suporte da OConS. O NetInf é visto como um aplicativo para o CloNe e o OConS. O NetInf contribui para a expressividade da funcionalidade de rede, o CloNe para agilidade de implantação e o OConS para elasticidade [90, 91].

O NetInf combina elementos presentes nas arquiteturas CCN [40], na arquitetura

Publish/Subscribe Internet Tecnology (PURSUIT) [92] e pode até trabalhar em modo híbrido. Além disso, pode ser utilizado por diferentes tecnologias de roteamento e encaminhamento. A seguir detalhamos como o NetInf utiliza nomeação, resolução de nomes e armazenamento de rede.

- **Nomeação ou Identificação**: A nomeação no NetInf utiliza um padrão do tipo *Uniform Resourse Identifier* (URI) ni://A/L, em que A refere-se ao nome da autoridade e L a localização da autoridade. Logo, a autoridade ou sua localização pode ser um valor *hash* ou uma sequência de caracteres [93];
- **Resolução de Nomes e Roteamento**: A resolução de nomes e o roteamento pode ser dividido em dois tipos, acoplados e desacoplados. No desacoplado, o NRS em seu mapeamento usa os nomes dos objetos para um identificador da localização onde o conteúdo está armazenando, exemplo como no endereço IP. O NRS é considerado uma DHT. Como mostrado na Figura 3.4, para tornar um conteúdo disponível, o editor (*publisher*) envia uma publicação com seu localizador no NRS local que armazena a L (localização) do nome. Assim o NRS local mescla todos os nomes com endereços L diferentes em um único endereço A e o envia para o NRS global (passos 1 e 2). Para encontrar um conteúdo, um consumidor (*subscriber*) envia mensagem GET para o seu NRS local que por sua vez, realiza uma consulta no NRS global e retorna com a localização do conteúdo desejado (passos 3 a 6). Por fim, o conteúdo é alcançado pelo consumidor através do NRS local e NRS global e consome o conteúdo desejado (passos 7 a 12) [93].

  Para o tipo acoplado, apresentado na Figura 3.5, o protocolo de roteamento é utilizado para publicar os nomes dos conteúdos e preencher as tabelas de roteamento dos *Contet Router*s (CRs), como na arquitetura CCN. Um consumidor envia mensagem requerendo um conteúdo para o CR local que é propagada *host-by-host* até o editor ou até um *cache* (passos A a C). Quando o conteúdo é localizado, este é encaminhado utilizando o caminho reverso até o consumidor (passos D a F) [93].
- **Armazenamento na Rede**: No NetInf, além de armazenar no meio do caminho (*inpath*) através dos CRs, também provê implementação de armazenamento do conteúdo em larga escala e recursos de replicação em cooperação com o NRS. Essas *caches* são conhecidas como publicadores de conteúdo [93].

No trabalho de Kaida [94] foi utilizado vários (total de 28) nós sensores em um ambiente simulado de uma *Smart City*. Cada nó sensor possui quatro sensores: um sensor de movimento, um sensor de energia elétrica, um sensor de luz e um sensor de gás. Estes nós armazena os dados e atuam como nós editores e consumidores de conteúdos utilizando implementações da arquitetura NetInf. Nos experimentos foi avaliado a taxa de perda de pacotes e redução de tráfego na rede com a distribuição de conteúdo em *cache*.

3.6 MobilityFirst

O projeto *Mobility First* (MF) propõe um modelo de Internet com suporte adequado aos casos de uso atuais. Para tanto, uma nova pilha de protocolos foi desenvolvida,

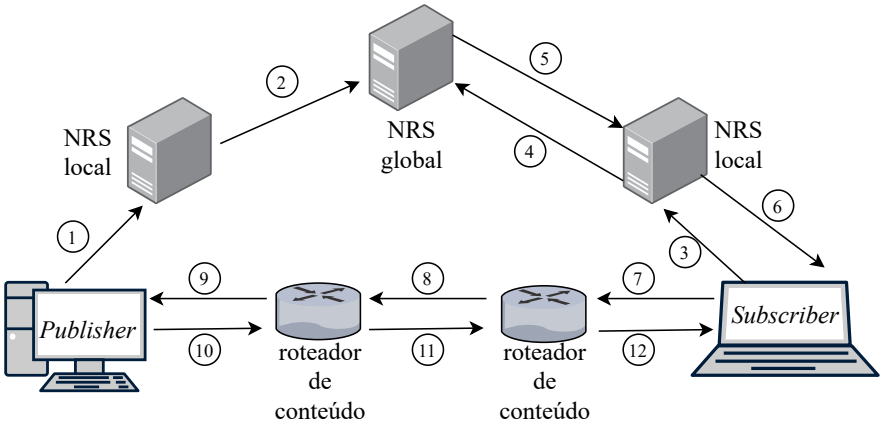


Figura 3.4: *Exemplo de um cenário utilizando a arquitetura NetInf com o tipo de roteamento desacoplado. Adaptado de [93].*

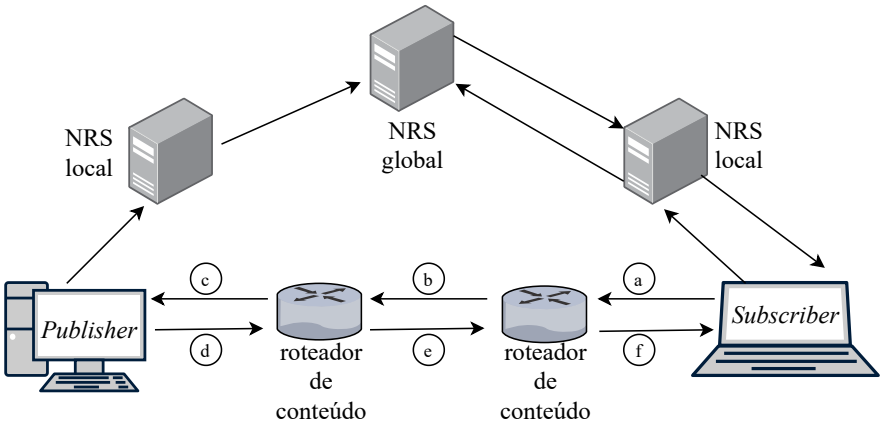


Figura 3.5: *Exemplo de um cenário utilizando a arquitetura NetInf com o tipo de roteamento desacoplado. Adaptado de [93].*

buscando unificar as redes tradicionais com as redes celulares, trazendo um foco em mobilidade de conteúdo e serviços [43].

A arquitetura MF traz em sua implementação conceitos de cache de rede, identificador e localizador, *Software-Defined Network* (SDN), conteúdo auto-certificado, modelo publica/assina e serviços de resolução de nomes [43].

MF implementa um *Globally Unique Identifier* (GUID) que é mapeado em um *Network Address* (NA); e, um roteamento com capacidade de armazenamento de conteúdo nos nós da rede seguindo o paradigma de ICN e que ajuda durante os processos de mobilidade através do *cache* de conteúdo pelo tempo necessário, para consumo posterior [95, 96].

O *Generalized Storage-aware Routing* (GSTAR) é outro componente essencial da arquitetura MF. Ele consiste de um recurso que suporta oscilações de qualidade de serviço da rede de acesso, tal como a sua interrupção, garantindo a entrega do conteúdo nas mais diversas situações [97].

Por exemplo, se um usuário estiver usando acesso de alta qualidade em um determinado momento, os dados serão entregues dentro do tempo de comutação do GSTAR.

Outro exemplo, na presença de redes de acesso de baixa qualidade ou mesmo desconexões, durante a troca entre redes de acesso, o mecanismo GSTAR pode se adaptar a essa situação e começar a trabalhar no modo *store-and-forward*. Nesse caso, mesmo que o usuário use um novo NA e identifique o usuário com o mesmo GUID, os dados serão armazenados e entregue posteriormente. Por fim, caso nenhuma das opções acima consiga entregar o conteúdo, o GSTAR poderia utilizar um tempo maior de desconexão, armazenando o conteúdo e entregando em algum momento, dessa forma a rede atuaria como um *Delay Tolerant Network* (DTN) [97, 98].

O MF também utiliza o recurso chamado *Global Resolution Name Service* (GRNS), alocado no plano de controle e cuja responsabilidade é mapear os GUIDs para os endereços de rede (NA). No caso de um dispositivo trocar de rede, o NA é alterado, mas o GUID permanece inalterado para que novas rotas possam ser calculadas. As primitivas usadas na arquitetura MF incluem GUID e NA em seus cabeçalhos, que são usados no primeiro momento para roteamento de pacotes. Se o roteamento de NA não for possível, o GNRS executará a análise para calcular um novo NA associado ao mesmo GUID, dessa forma o conteúdo será entregue aos usuários móveis mesmo com a mudança do NA [97, 98].

Em relação à nomeação/localização, o nome legível por humanos (NLN) pode ser gerenciado e atribuído a um GUID exclusivo por *Name Certification Service* (NCS). O GUID atribuído por um NCS é derivado de uma chave pública, permitindo assim, serviços de autenticação e segurança na rede. A derivação do GUID como um *hash* criptográfico de uma chave pública também permite que eles sejam auto-certificados (SVN), ou seja, autenticar um nó não requer uma autoridade externa. Tanto os nomes (identificadores) quanto os endereços (localizações) são definidos como GUIDs. Um exemplo de GUID de conteúdo (vídeo) seria, GUID=13247...99 [43, 97, 98].

A comunicação baseada em GUID é um ponto fundamental na arquitetura, torna flexível trabalhar com uma diversidade de *endponts*, como exemplo dispositivos, usuários, serviços e conteúdos. No entanto, independente da localização, suporta a comunicação de *device-to-device*, *device-to-service*, *centric-content*, *anycast*, *multicast* e outros [97, 98]. Na Figura 3.6 mostramos os componentes de rede MF.

No trabalho de Bronzino et al. [43] são apresentadas as plataformas de *testbed* *Open-access Research Testbed for Next-Generation Wireless Networks* (ORBIT), a rede *Global Environment for Networking Innovation* (GENI), e a arquitetura de FI, MF. Os autores de [43] utilizaram a plataforma ORBIT que emula redes físicas e *wireless* para teste em campo, sendo avaliadas métricas como, custo computacional, *throughput* e latência. Já utilizando a plataforma GENI (que oferece infraestrutura com *backbones* de *layer* 2) foram alocadas máquinas virtuais XEN, e roteadores que implementam o roteamento e encaminhamento do protocolo *MobilityFirst* para avaliação em um cenário real de *streaming* de vídeo. No terceiro cenário foi utilizado um terminal celular nos experimentos, porém avaliando somente a experiência do usuário, sem levar em conta métricas de avaliação de rede [43].

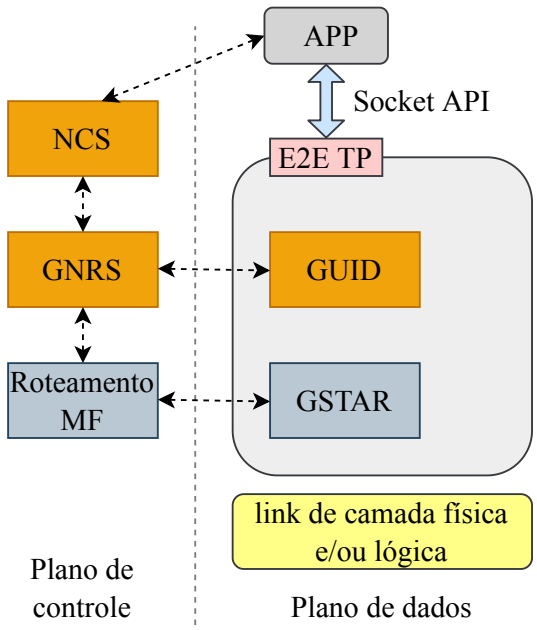


Figura 3.6: *Principais componentes de rede MobilityFirst. Adaptado de [96].*

3.7 CURLING

No trabalho de [84] é apresentado o *Content-Ubiquitous Resolution and Delivery Infraestructure for Next-Generation Services (CURLING)*, uma proposta de ICN que estabelece uma Internet centrada em conteúdos, abordando desde publicação, resolução de nomes (com roteamento *unicast* e *multicast* baseado em conteúdo) e finalmente entrega de conteúdos. CURLING utiliza os conceitos de SDN (*Software Define Network*), desacoplando o plano de dados do plano de controle. Com essa implementação possibilita o gerenciamento via *software* no controle de transmissão de conteúdo entre as redes locais ou em domínios diferentes.

A Figura 3.7 ilustra a rede do CURLING, abordando a forma como a arquitetura emprega a resolução de nomes hierárquicos *hop-by-hop*. A arquitetura conta com duas entidades principais:

- ***Content Resolution Controller* (CRC)**: Encarregado em descobrir a origem do conteúdo, oferecer suporte à entrega e facilita a publicação do conteúdo. No mínimo um CRC é implementado no domínio local para a resolução de publicação e consumo de conteúdo;
- ***Content-Aware Router* (CaR)**: Trabalha em conjunto com CRC local, adicionando caminhos de entrega de conteúdo acionados por destinatários.

Os componentes, *Content Provider*s (CPs) e *Content Consumer*s (CCs) são entidades internas que aferem recursos as publicações e consumo dos conteúdos. O conteúdo é localizado e recuperado pelo seu ID. Um CID (identificador do conteúdo) pode ser estabelecido por meio de *hashing* criptográfico do conteúdo [84, 99].

Como outras propostas de ICN, CURLING utiliza de armazenamento de conteúdo em rede, mas somente nos roteadores de borda, ou seja, aparelhos com recursos de armazenamento em *cache*. Seu *design* contém compatibilidade com a rede base-

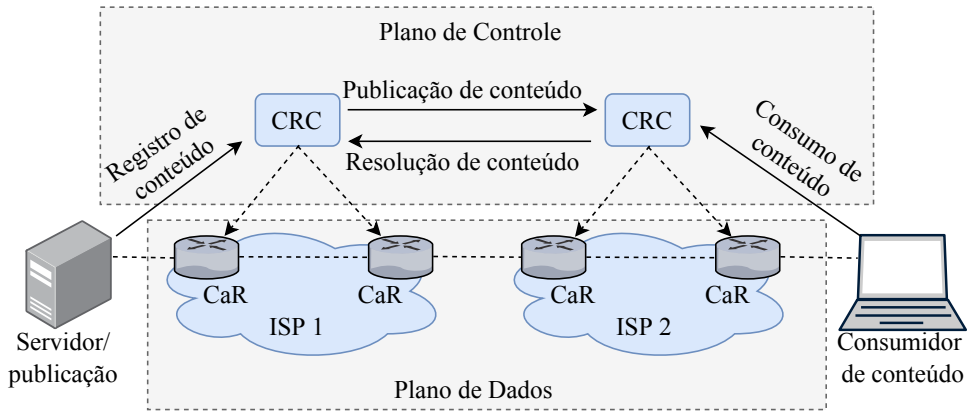


Figura 3.7: *Abordagem de alto nível de resolução de conteúdo hop-by-hop da arquitetura do CURLING. Adaptado de [84].*

ada em IP. Diferente de algumas abordagens radicais como CCN/NDN [42], a arquitetura *Data-Oriented Network Architecture* (DONA) [100] e também a arquitetura *Publish/Subscribe Internet Routing Paradigm* (PSIRP) [101] projetadas desde o inicio para substituir o protocolo IP.

Para avaliar a arquitetura do CURLING, no trabalho de [84] foi desenvolvido um modelo analítico e simulações híbridas de sub-topologias presentes na Internet atual para obter métricas de latência na resolução de conteúdo, ganho de mecanismo de otimização de caminho, número de saltos, *multihoming* e *peering*, mostrando a relevância do projeto.

3.8 Análise e Oportunidades de Pesquisa

Os trabalhos discutidos foram avaliados de acordo com quatro quesitos. Os quesitos analisados estão abaixo, e apresentados na Tabela 3.1:

- **Nomeação**: Refere-se à forma como a arquitetura utiliza nomeação;
- ***Caching***: Define como o conteúdo é armazenado na rede;
- **Resolução de nomes**: Refere-se ao método e/ou função utilizada na resolução de nomes (ou ligações entre nomes);
- **Roteamento**: Define o mecanismo de roteamento de conteúdo na rede;
- **VM e/ou contêiner**: Refere-se ao ambiente de experimentação da arquitetura.

Analisando os trabalhos, podem ser destacadas as seguintes lacunas:

- Necessidade de alteração no código fonte ou desenvolvimento de códigos adicionais para utilizar NLNes e SVNes (CCN [40,80]; RINA [86,89]; SAIL [90,94]; CURLING [84]);
- Utiliza contêineres *Docker*, mas exige muito trabalho manual e propenso a erros (XIA [7, 102]);
- Não efetua contrato de prestação de serviços de distribuição de conteúdos (CCN [40, 80]; XIA [7, 102]; RINA [86, 89]; SAIL [90, 94]; MF [43, 95]; CURLING [84]);

- Não utilizou contêineres *Docker* na avaliação da arquitetura (CCN [40, 80]; RINA [86, 89]; SAIL [90, 94]; MF [43, 95]; CURLING [84]);

Tabela 3.1: *Comparativo das tecnologias utilizadas nos trabalhos relacionados.*

Trabalho	**1. Nomeação, 2. Caching, 3. Resolução de nomes, 4. Roteamento**	**VM e/ou Contêiner**
CCN [40, 80, 103]	1. Utiliza nomeação com base em seus NLNes estruturados. Pode usar SVNes em parte da estrutura de nomes. 2. Também utiliza seus NLNes estruturados no armazenamento de conteúdo. 3. Existe o Sistema de Mapeamento de Serviço de Nomes, é um sistema que fornece resolução de nomes para CCN. 4. Roteia e distribui conteúdo com seus NLNes.	Plataforma de *testbed* FITS com VMs.
XIA [7, 102]	1. A arquitetura nomeia domínio, *host*, conteúdo, serviço como *Principals Types*. Estes são suportados por XIDs. 2. A arquitetura faz uso dos NLNes e SVNes no armazenamento em *cache*. 3. Oferece um serviço de resolução de NLNes para XID chamado NameSrv. 4. O roteamento nos roteadores utiliza encaminhamento interativo. A noção de *Fallback* é utilizada como caminhos alternativos para alcançar o destino.	Ambiente controlado com várias imagens *Docker*.
RINA [86, 89]	1. A nomeação fica por conta do usuário. Nas redes RINA a nomeação não é apenas contínua (pode ser feita sem afetar os fluxos existentes), não requer mecanismos especiais. 2. Cada IPC possui filas para armazenamento de pacotes após o processamento no *host* de origem e roteadores no caminho atingir o destino. 3. A resolução de nomes é por meio dos nomes dos aplicativos. 4. Os DIFs roteiam os pacotes e são personalizáveis para outras funcionalidades.	Uso de VMs.
NetInf [90, 94]	1. A nomeação com base nos SVNes ou com *hash* simples divide toda a operação em duas etapas: resolução de nomes pelo NRS e roteamento de dados pelo próprio nó. 2. Oferece armazenamento de conteúdo em cache no caminho e fora do caminho. 3. Resolução de nome por NRS local e NRS global. 4. Uma estratégia LNB (*Late Nome Binding*) dentro dos NRS atualiza dinamicamente as informações de roteamento.	Ambiente simulado com nós sensores.
MF [43, 95]	1. Usa esquema de NLNes simples e SVNes. Emprega nomes longos de 160 bits para evitar colisões e facilitar e agilizar a comparação. 2. O armazenamento de conteúdo é realizado nos roteadores no meio do caminho com o componente conhecido como GSTAR. 3. O GNRS efetua a resolução de nomes e mapeia os GUIDs para os NA. 4. A primitiva do MF utiliza GUID e NA para roteamento. Caso não seja possível rotear com NA, o GNRS calcula um novo NA associado ao mesmo GUID.	Plataforma ORBIT e GENI. Avaliação de experiência do usuário com *smartphone*.
CURLING [84]	1. A nomeação não foi especificada, aprimora o acesso fácil e a rápida distribuição de conteúdo por meio da rede. 2. Armazenamento de conteúdo nos roteadores de borda. 3. Arquitetura dissociada que separa a resolução de nomes e o encaminhamento de dados. Resolução de nome não esta clara. 4. O CaR trabalha em conjunto com CRC no roteamento e entrega de conteúdo.	Modelo analítico e simulações híbridas.

A CCN distribui e armazena conteúdo com base nos seus NLNes. Os roteadores utilizam componentes (PIT, FIB, CS) para realizar busca, envio e entrega dos dados. Pacotes de interesses percorrem a rede ao encontro de pacotes de dados. Na XIA, a arquitetura nomeia objetos por meio de IDs. IDs de conteúdos (CIDs), IDs de serviços (SIDs), IDs de *hosts* (HIDs), IDs de domínio administrativo (AD) são suportados no roteamento. Caso algum roteador não suporte algum *Principal Type* (exemplo CID), uma abordagem *fallback* será utilizada, interpreta e executa o que é possível. Em RINA, a distribuição de conteúdo nomeado pode ser implementada em algum DIF. O suporte de DIFs subjacente são essenciais nesse processo. Em SAIL, o NetInf nomeia entidades utilizando o padrão URI. Estas são instanciadas pelo CloNe, onde a comunicação entre os nós faz uso de FNS. Para o MF, o nome do conteúdo é definido por um GUID, onde a comunicação entre os nós é passeada em GUID. A identificação do conteúdo é única na rede, ou seja, independente de sua movimentação ele será localizado, somente o NA deve ser modificado. No CURLING um ID de conteúdo (CID) é estabelecido utilizando algoritmo de *hashing* criptográfico. Os roteadores armazenam os conteúdos no meio do caminho facilitando a busca por consumidores. A NG permite a nomeação de qualquer entidade em NLNes, quanto SVNes, e também relaciona seus nomes através de NBs. Com isso a arquitetura torna possível a comunicação entre milhões de entidades, como alvo.

Para a resolução de nomes, a CCN roteia pacotes de interesse na rede. Os roteadores armazenam as solicitações, quando o pacote de dados é localizado, o caminho reverso será utilizado para entrega do conteúdo. Assim, a arquitetura faz a resolução de nome de conteúdo. A rede armazena o conteúdo próximo do consumidor. Os roteadores XIA se encarregam por resolver um XID (identificador XIA) para outros. Além do roteamento de XIDs entre roteadores a XIA disponibiliza um serviço conhecido como NameSrv na resolução de NLN. A arquitetura SAIL permite dois tipos de resolução de nomes: (i) acoplados; e (ii) desacoplados. No primeiro, NRS local e global são utilizados para mapear o nome do conteúdo com um ID de localização. Para o segundo, os nomes dos conteúdos são publicados para os roteadores preencher as tabelas de roteamento. Os CRs são responsáveis pelo *caching* de rede. O GRNS mapeia os GUIDs na rede MF, facilitando a pesquisa de conteúdo armazenado no *cache* da rede. A RINA permite somente a resolução de nomes de aplicativos através de IDD. Na NG, o NRNCS é o responsável para a resolução de nomes, armazenamento de conteúdo em servidor próximo de consumidores e torna disponível qualquer nome de forma segura.

As FIAs CCN, RINA e MF foram avaliadas utilizando ambientes com VMs. Para o CURLING um modelo matemático foi implementado. A XIA foi avaliada com implementação em contêineres *Docker*, porém o ambiente dificulta a replicação dos experimentos. Muitos comandos Linux são necessários e qualquer erro de digitação ou ordem de execução impacta nos resultados finais. A arquitetura SAIL foi avaliada utilizando a implementação NetInf em ambiente simulado com nós sensores. Analisando taxa de perda de pacotes e distribuição de conteúdo em *cache*. Por fim, a NG será avaliada nesse trabalho utilizando ambiente físico, máquinas virtuais e contêineres *Docker*.

Capítulo 4

Proposta de um Ambiente de Experimentação e Avaliação de Arquiteturas de Internet do Futuro

Este Capítulo trata da contribuição original dessa dissertação, que é o desenvolvimento de um ambiente de avaliação de FIAs com o uso de contêineres *Docker*. É através desse ambiente que as melhorias da NovaGenesis no que tange a nomeação, resolução de nomes, distribuição de conteúdos com *cache* de rede e auto-organização de serviços de mídia será avaliada. O que foi desenvolvido com o objetivo de automatizar os experimentos com e sem contêineres foi detalhado na Seção 4.1. A Seção 4.2 apresenta os requisitos levados em consideração no desenho do ambiente de experimentação proposto para avaliar a NG. A preparação e estrutura do Ambiente AAIF discutimos na Seção 4.3. Por fim, as considerações parciais com relação à solução proposta são feitas na Seção 4.4.

4.1 AAAIF

O Ambiente AAIF, uma estrutura de experimentação de código aberto desenvolvida no ICT Lab. para reduzir drasticamente o tempo necessário de iniciar e conduzir experimentos em grande escala. Fornece uma implementação simples, flexível e amigável para modelar cenários de rede com NovaGenesis, bem como funções de reserva de recursos computacionais, busca e coleta de dados dos resultados e liberação dos recursos. Nossos estudos mostram que o AAAIF minimiza o tempo necessário de configuração e execução de contêineres e serviços NG em várias ordens de magnitude.

Em outras palavas, o objetivo do AAAIF é reduzir o precioso tempo gasto na preparação e configuração da rede do ambiente experimental em larga escala, além de permitir ao pesquisador resultados reproduzíveis. Deve suportar qualquer ambiente, sendo ele, laboratório com máquina física, conjunto de VMs, vários contêineres ou plataformas de *testbed*.

As funções de serviços permitem instanciar vários nós de contêineres e conectá-los para criar uma infraestrutura de experimentos. Os *scripts* oferecidos pelo AAAIF deve

atender às necessidades de qualquer plataforma.

O pesquisador deve poder selecionar facilmente o cenário, enquanto os *scripts* cuidam das configurações, acesso remoto, transferências de pastas e arquivos na rede.

Oferece uma interface simples para definir a inicialização do experimento, ou seja, (i) quais *hosts* podem executar um serviço, aplicativo cliente ou servidor; (ii) qual servidor é responsável por executar os principais serviços; (iii) quando o aplicativo cliente ou servidor devem ser executados. A ordem e o tempo para iniciar o aplicativo pode ser definida antes de iniciar o cenário; e (iv) quando finalizar o experimento, a interface disponibiliza coleta dos resultados com facilidade.

Por fim, o AAAIF deve suportar repetições de experimentos. Como o cenário do experimento é definido por *script*, a utilização do mesmo várias vezes deve gerar resultados equiparáveis. Observe que na infraestrutura, os experimentos podem selecionar diferentes *hosts*; no entanto, os resultados podem diferir a cada execução do cenário. O pesquisador deve analisar criteriosamente com base no conjunto completo de recursos computacionais, se alguma falha de *host*, serviço ou aplicativo não impactou no resultado final.

Automatizando o Uso de Contêineres

O processo de execução dos comandos para iniciar, acessar, pausar, sair e excluir um contêiner é relativamente simples aos usuários Linux, mas o gerenciamento de múltiplos contêineres manualmente demanda tempo e tarefas repetitivas. No AAAIF, automatizamos estas tarefas. Foi criado um arquivo chamado **hyperDocker.py**, que atualmente mantém 260 linhas de código. Como *script* mestre (hyperDocker), responsável por acionar todos os demais *scripts* envolvidos no Ambiente AAIF. Ao executar o comando *python hyperDocker.py* no terminal Linux temos acesso às opções de gerência dos contêineres. Cada opção é descrita em detalhes a seguir. A preparação da imagem de contêiner *Docker* está no Apêndice A. O código e o fluxograma se encontra no Apêndice C.

Opção 1 - Criar Contêineres: Cria contêineres para a FIA escolhida neste trabalho. Os contêineres são criados utilizando a imagem (ng-template) definida na Seção A.1. Cada contêiner utiliza a rede *default* apresentada na Subseção 2.8.3. O primeiro contêiner recebe o endereço IP 172.17.0.2/16 e os próximos seguem a mesma classe de IP. Os parâmetros de inicialização, definidos no código, permite o acesso remoto com usuário e senha. A opção 1, quando escolhida, solicita ao usuário a quantidade de contêineres, porcentagem de memória RAM e CPU para cada contêiner. Com a alternativa *n* (não), o uso de RAM e CPU fica a critério do *cgroups* (Subseção 2.8.4), que por sua vez efetua a distribuição proporcional entre os contêineres;

Opção 2 - Listar Contêineres: A opção 2 lista todos os contêineres criados até o momento. Estes são nomeados seguindo o padrão *ng1, ng2,...,ngN*. Para outras FIAs, o nome poderia ser modificado;

Opção 3 - Entrar em um Contêiner: A opção 3 lista os contêineres em execução. Na listagem cada contêiner recebe um número. A numeração inicia com o valor zero (0) para acesso ao primeiro contêiner, número um (1) acessa o segundo contêiner e assim sucessivamente;

Opção 4 - Estatísticas dos Contêineres: A opção 4 mostra estatísticas do uso de CPU e RAM de cada contêiner. A estatística é apresentada no formato de porcentual. Assim, podemos analisar o consumo de CPU e RAM. Quando a porcentagem aproxima do limite máximo suportado pelo *hardware*, provavelmente deve impactar nos resultados. Somente após os experimentos podemos avaliar as consequências e uso máximo dos recursos computacionais;

Opção 5 - Excluir Contêineres: Esta opção exclui todos os contêineres independente de conterem aplicativos em execução ou não. Opção extremamente importante quando precisamos finalizar o cenário de experimentos e iniciar novamente. Quando um contêiner é excluído, todos os serviços deixam de existir. Portanto, evitando sobreposição de processos antigos;

Opção 6 - Iniciar Cenário NB: A opção 6 faz o envio das pastas, subpastas, arquivos do diretório via *ssh* para o primeiro contêiner. Esta opção foi configurada para um cenário específico de resolução de nomes. Os detalhes do cenário serão discutidos na Seção 5.2;

Opção 7 - Iniciar Cenários NBs: A opção 7 também faz o envio de outro diretório com arquivos e *script* para primeiro contêiner via serviço *ssh*. Esta opção foi configurada para vários casos de cenários de resolução de nomes. Para outras FIAs, esta opção pode ser especializada de acordo ou removida;

Opção 8 - Iniciar Cenário de Conteúdo: Esta opção é responsável por copiar um diretório do diretório AAAIF para o primeiro contêiner. O código pode ser facilmente adaptado em outras FIAs. A opção 8 foi configurado com um cenário específico de distribuição de conteúdo;

Opção 9 - Baixar Relatório: A opção 9 localiza no primeiro contêiner os arquivos com os resultados dos experimentos e baixa no diretório específico do Ambiente AAIF; e

Opção 10 - Finalizar Experimentos: Por fim, a opção 10 finaliza o algoritmo, pausando e excluindo todos os contêineres. Assim, os serviços dentro de cada contêiner são removidos da lista de processos evitando sobreposição de serviços nos próximos experimentos. Caso o usuário digite uma opção inválida é solicitado uma nova opção válida.

No Apêndice B se encontra a preparação do SO nativo na máquina física.

Automatizando Experimentos com Name Bindings

Nossos estudos com a configuração e instalação manual da rede NovaGenesis de tamanho moderado apresentou que é um tarefa propensa a erros e difícil de repetir. Pensando em reduzir o trabalho manual, desenvolvemos os *scripts* para um cenário especifico de resolução de nomes da NovaGenesis. O cenário é um dos escolhidos para avaliar o nosso Ambiente de Experimentação, bem como a própria NovaGenesis, pois esta carece de análise de desempenho em maior escala. Um longo caminho foi percorrido até que os *scripts* executassem todas as etapas dos experimentos demandados pela NG tal qual apresentado no Capítulo 2.

As limitações da atual infraestrutura de Internet, discutida na Subseção 2.1.2, onde

o DNS segue uma hierarquia de resolução de nomes convertendo NLN em IPs requer melhorias ou substituição completa para uma nova solução. De encontro com as limitações, a NG traz soluções com vantagens na utilização de resolução de nomes em NLN e/ou SVN, podendo trabalhar de forma separada ou hibrida como apresentado na Seção 2.2.1. Nossa proposta de automatização de tarefas antes manuais na avaliação de NBs NG, vem com objetivo de facilitar o escalonamento dos serviços da arquitetura, destacando as novidades e vantagens. Cada opção é descrita em detalhes a seguir.

Opção 1 - Inicializar Parâmetros: Efetua o escaneamento do *hostname*, IPs, endereço *MAC* utilizando o protocolo *nmap* na rede. Essa etapa permite que o Ambiente de Experimentação desenvolvido descubra os *hosts* físicos ou virtuais que podem ser usados para executar os serviços NovaGenesis. Utiliza um pacote utilitário gratuito e de código aberto (licença) para descoberta de rede. Recebe os atributos da rede existente. Define o parâmetro do arquivo parametrosPGCS contendo o endereços MAC dos *hosts* envolvidos no experimento e imprime a interface de rede descoberta. Exibe o *range* dos IPs listados. Para finalizar, o endereço MAC do primeiro contêiner é armazenando no arquivo macServer;

Opção 2 - Enviar Arquivos NG: Uma tarefa muito comum por administradores de rede Linux, é executar comandos entre vários *hosts*. Dessa forma, com o objetivo de instalar pacotes, executar configurações e até mesmo manter um padrão de configurações. Como facilidade de acesso remoto para fazer o envio ou coleta de arquivos utilizamos um módulo no *Python* chamado *paramiko*, que foi criado justamente para fazer conexões via ssh. Com os endereços IPs e MACs filtrados e armazenados na opção anterior, na sequência utilizamos a Opção 2. A opção 2, é responsável pelo acesso remoto individual *host-by-host*. Uma pasta padrão como o nome *workspace* é criada e recebe as permissões de administrador (*root*) do SO. A pasta *workspace* recebe uma cópia da pasta novagenesis_files, contendo os arquivos binários dos serviços NG e também os *scripts*. O módulo *paramiko* em conjunto com o protocolo sftp envia os arquivos. Assim que todas as tarefas são executadas por esta opção, uma mensagem impressa no terminal nos indica arquivos transferidos com sucesso ou erros de transferências;

Opção 3 - Apagar Diretório: O acesso remoto, criação de pasta, envio de arquivo ao mesmo tempo, leva a possíveis erros e perda de pacotes na rede. Portanto, ao utilizar a opção 2 pode ocorrer alguns erros de envio, tais como, travamento do *host*, incapacidade do comutador ou *bridge* de escoar o trafego de dados na rede, tempo de execução expirado ou por outro motivo. Caso ocorrer falhas e por algum motivo um *host* não receba os arquivos do experimento, utilizamos a opção 3. Esta opção tem a função de excluir todas as pastas e arquivos presentes na pasta *workspace*. Para o reenvio de pastas e arquivos faz-se necessário o uso da opção 2 novamente. É importante salientar que o não recebimento dos arquivos impacta na execução correta dos experimentos, bem como na coleta de resultados;

Opção 4 - Executar *Core* NG: Com os parâmetros pré-configurados na execução de cada serviço (PGCS, GIRS, HTS, PSS) NG, tais como o caminho dos arquivos binários e tempo de inicialização dos serviços são registrados nos arquivos runCore e parametrosPGCS. O *script* runCore executa em sequência cada serviço, tal qual especificado no cenário de resolução de nomes ou distribuição de conteúdo em um domínio.

Cada serviço será apresentado em uma janela individual utilizando o terminal *Xterm*;

Opção 5 Executar PGCS no cliente: Cada *host* deve executar seu representante virtual (PGCS). Um serviço que representa o dispositivo onde ele se conecta como discutido na Seção 2.2.7. O acesso individual em cada *host* via ssh executa de forma paralela cada PGCS distribuído na rede, considerados clientes. A comunicação entre os PGCSes distribuídos pode ser observada através de mensagens NG;

Opção 6 - Excutar PGCSes e HTSes no cliente: Em casos específicos de cenários com NBs essa opção executa os serviços PGCSes e HTSes de forma distribuída na rede dentro de vários *hosts*;

Opção 7 - Exceutar *NBSimpleTestApp*: Realiza o acesso remoto com o módulo *paramiko* via ssh no *host* que contém o aplicativo de NBs. Essa opção executa o aplicativo *NBSimpleTestApp*. As configurações do aplicativo são estabelecidas antes de iniciar o experimento. O *NBSimpleTestApp* publica milhões de NBs;

Opção 8 - Salvar Relatório: A coleta dos resultados nos experimentos de resolução de nomes fica a cargo da opção 8. Essa opção localiza na rede os arquivos com os resultados das publicações e assinaturas de NBs. Realiza um cópia e armazena os arquivos coletados em uma pasta específica;

Opção 9 - Serviços: A liberação dos serviços na lista de processos no SO dos *hosts* considerados clientes, libera recursos computacionais e uso de memória compartilhada. Essa opção exclui os serviços NG da lista de execução dos *hosts* clientes. Os serviços liberados são PGCSes, HTSes e *NBSimpleTestApp*;

Opção 10 - Finalizar Experimentos: Por fim, a opção 10 encerra o experimento de NBs excluindo os serviços do core NG da lista de processos em execução no primeiro *host*. Assim, liberando recursos computacionais.

O código fonte dessa ferramenta do Ambiente de Experimentação desenvolvido está no Apêndice D. Neste, incluímos todos os códigos desenvolvidos para a ferramenta proposta nessa dissertação.

Automatizando Experimentos com Troca de Conteúdos

O armazenamento e busca de conteúdo requer a indicação de um ou vários servidores na Internet. Retomando a discussão na Seção 2.1.4, este é um modelo arcaico, pois a realidade atual dos usuários ao consumo de conteúdo independente de onde este esteja, não pode ser limitada no caminho específico do servidor remoto. Dessa forma, se o conteúdo, seja ele qualquer tipo de arquivo, movido de servidor ou estiver seu nome alterado, o mesmo não será localizado. Nas Seções 2.2.2 e 2.2.3 apresentamos como a NG propõe soluções de armazenamento, identificação e localização de conteúdo. O NRNCS é responsável por resolver a ligação de nomes e armazenamento de conteúdo em *cache* de rede. Mesmo que o conteúdo seja movido de local, o mesmo pode ser encontrado utilizando ID/LOC na pesquisa através de grafos.

Para um cenário específico com distribuição de conteúdo, foi necessário remodelar o fluxograma da Seção D.1 que se encontra no Apêndice D, adicionar novas funções e organizar os arquivos e pastas. O novo fluxograma se encontra no Apêndice E, Seção E.9. A seguir, descrevemos as funcionalidades das opções (6, 7, 8, 10). Portanto, as

opções não descritas aqui, foram discutidas anteriormente.

Opção 6 - Criar Fotos: Localiza a pasta que armazena o conteúdo no *host* cliente. Inicializa o *script* com a função criar fotos. O *script* gera as fotos com base no padrão RGB (*Red, Green e Blue*), altura 90 milímetros e largura 90 milímetros. Portanto, cada arquivo gerado mantém um tamanho padrão de 6 Kilobytes (6K). O usuário do Ambiente AAIF pode alterar este valor aumentando ou diminuindo as dimensões (altura, largura). Além disso, podemos criar fotos iguais, diferentes ou mistas;

Opção 7 - Executar *APPServer*: Efetua o acesso remoto no *host*. Localiza a pasta com o aplicativo *APPServer*. Executa o aplicativo. O *APPServer* realiza a comunicação com PGCS e PSS e aguarda as publicações de conteúdo. Por fim, o aplicativo recebe e armazena o conteúdo;

Opção 8 - Executar *APPClient*: Efetua o acesso remoto no *host*. Localiza a pasta com o aplicativo *APPClient*. Inicia o aplicativo *APPClient*. Portanto, o aplicativo localiza e estabelece comunicação com o Core NG seguindo as regras de contratos de serviços na arquitetura. Assim que o contrato foi estabelecido as fotos criadas com a opção 6 são enviadas;

Opção 10 - Serviços: Libera recursos computacionais e uso de memória compartilhada. Essa opção exclui os serviços NG da lista de execução dos *hosts* clientes. Os serviços liberados são PGCSes, HTSes, *APPClient* e *APPServer*;

No Apêndice E está presente o código que desenvolvemos.

4.2 Avaliação de Requisitos

O primeiro deles diz respeito à conectividade entre contêineres. Para experimentação e avaliação em Internet do futuro (NG), o ideal é que a conectividade seja no nível de Camada 2 (sem IP). O *Docker* utiliza uma *bridge* para conectar contêineres. De maneira simples, podemos definir como *bridge*, um dispositivo da Camada de Enlace, podendo ser *hardware* ou *software*, que possui a capacidade de encaminhar tráfego na rede. No universo de *Docker*, uma rede *bridge* de *software* pode ser utilizada para prover comunicação entre contêineres e garantir isolamento, conforme especificado na documentação oficial [24] do provedor do conteúdo.

No escopo do *Docker* podemos identificar dois tipos de redes, *bridge* e *overlay*. Tratando-se especificamente da rede *bridge* é utilizada para casos simples e genéricos, no qual se utiliza em ambiente local, sendo definida por padrão ao ser inicializada por um contêiner. A rede *overlay* é utilizada para um número maior de contêineres em diferentes redes (locais, nuvens), no qual exige uma configuração mais refinada, descoberta e comunicação distribuída. Neste trabalho, utilizaremos somente a rede *bridge* padrão do *Docker*. Para isso, uma única máquina física (servidor com alta capacidade de processamento) será selecionada.

A rede *bridge* pode ser subdividida em duas, a *bridge* padrão, identificada por *Docker0* ao criar um contêiner sem definir rede, a *bridge* definida pelo usuário, no qual o ecossistema fornece uma gama de opções variadas para sua configuração. As redes *bridges* são delimitadas a contêineres que estão sendo executados em um mesmo *daemon* do *Docker*, ou seja, contêineres executados em um mesmo *host*. Para comunicação

de contêineres entre diferentes *host* faz-se necessária a criação de uma *overlay* ou gerenciamento de rota ao nível de SO [65].

Para de fato aplicarmos o conceito de nomeação flexível postulado pela arquitetura NovaGenesis, investigamos o efetivo uso da rede *bridge* para garantir seu real funcionamento assimilado pela camada de enlace. Desta maneira utilizamos duas ferramentas para a averiguação, o *brctl* para administração de redes *bridges* e *sniffer* de rede tcpdump [104, 105].

O *brctl* é um pacote do *kernel* Linux utilizado para administrar *bridges ethernet*. Nele é possível adicionar, remover, configurar e inspecionar *bridges* e interfaces. Utilizando o comando <brctl showmacs docker0> no qual *brctl* refere-se ao nome do programa que será executado, *showmacs* é o argumento que indica o retorno de endereços *MACs* e *Docker0* representa a *bridge* do *Docker*. Conforme a Figura 4.1 obtida como retorno, podemos visualizar endereços *MAC* associados às portas da *bridge*, conceito compatível com a camada de enlace.

```
port     no mac addr         is local?   ageing timer
 2       02:c7:e1:f0:ba:73       yes         0.00
 2       02:c7:e1:f0:ba:73       yes         0.00
 1       4e:60:15:44:d0:98       yes         0.00
 1       4e:60:15:44:d0:98       yes         0.00
 5       9a:fb:24:50:59:d3       yes         0.00
 5       9a:fb:24:50:59:d3       yes         0.00
 3       b2:21:81:bf:3b:90       yes         0.00
 3       b2:21:81:bf:3b:90       yes         0.00
 4       de:94:e0:a5:be:56       yes         0.00
 4       de:94:e0:a5:be:56       yes         0.00
```

Figura 4.1: *Retorno do comando brctl.*

Utilizando ferramentas de *sniffer* como por exemplo *Ntop*, *Wireshark* ou *tcpdump*, podemos monitorar o fluxo de tráfego entre dispositivos [106–108]. Em um exemplo simples, conforme a Figura 4.2, executamos o comando < tcpdump -i docker0 > no qual *tcpdump* refere-se ao nome do programa que será executado para monitorar a rede, e a seguir a interface -i *Docker0* que representa a *bridge* do *Docker*. Em sua saída podemos visualizar os seguintes argumentos:

- O primeiro argumento, *timestamp*, carrega o horário em que o pacote foi capturado;
- O segundo argumento apresenta o tipo de pacote e endereço de origem. Neste caso podemos visualizar uma característica específica da NovaGenesis no qual o endereço de destino é representado pelo endereço *MAC* e a mensagem *"oui Unknow"* nos diz que o *tcpdump* não reconhece este protocolo;
- O terceiro argumento representa uma sinalização (>) que indica o sentido do pacote;
- O quarto argumento indica o tipo de pacote e endereço de destino;
- O quinto argumento, *ethertype* igual a 1234 especifica que a pilha de protocolos NovaGenesis está sendo transportada em Ethernet;
- O sexto argumento é o tamanho da mensagem; e
- Em sequência podemos ver o conteúdo da mensagem NovaGenesis. Os detalhes

dos comandos e mensagens foram apresentados na Subseção 2.2.9 e 2.2.10.

```
00:21:23.259206 02:42:ac:11:00:02 (oui Unknown) > 02:42:ac:11:00:03 (oui Unknown),
ethertype Unknown (0x1234), length 393:
        0x0000: 29f5 bcf9 0000 0000 0000 0000 0000 016b ).............k
        0x0010: 6e67 202d 6d20 2d2d 636c 2030 2e31 205b ng.-m.--cl.0.1.[
        0x0020: 203c 2031 2073 2031 3542 3233 3944 3120 .<.1.s.15B239D1.
        0x0030: 3e20 3c20 3420 7320 3630 4145 4542 3242 >.<4.s.60AEEB2B
        0x0040: 2036 3446 4542 3246 3520 3944 4231 3939 .64FEB2F5.9DB199
        0x0050: 4333 2033 3041 3844 3331 3120 3E20 3C20 B3.30A8D311.>.<.
        0x0060: 3420 7320 656D 7074 7920 656d 7074 7920 4.s.empty.empty.
        0x0070: 656d 7074 7920 656d 7074 7920 3e20 5d0a empty.empty.>.].
        0x0080: 6e67 202d 6865 6c6c 6f20 2d2d 6968 6320 ng.-hello.--ihc.
        0x0090: 302e 3120 5b20 3c20 3920 7320 3630 4145 0.1.[.<9.s.60AE
        0x00a0: 4542 3242 2036 3446 4542 3246 3520 3944 EB2B.64FEB2F5.9D
        0x00b0: 4231 3939 4233 2033 3041 3844 3331 3120 B199B3.30A8D311.
        0x00c0: 4232 4431 4245 3144 2043 3241 3546 3645 B2D1BE1D.C2A5F6E
        0x00d0: 3420 4574 6865 726e 6574 2065 7468 3020 4.Ethernet.eth0.
        0x00e0: 3032 3a34 323a 6163 3a31 313a 3030 3a30 02:42:ac:11:00:0
        0x00f0: 3220 3e20 3c20 3920 7320 3934 3144 4644 2.>.<.9.s.941DFD
        0x0100: 4132 2035 4534 4441 4134 4320 3839 3530 A2.5E4DAA4C.8950
        0x0110: 4239 4441 2043 4341 4643 4230 3720 4141 B9DA.CCAFCB07.AA
        0x0120: 3843 3934 4430 2037 4644 4244 4438 4220 8C94D0.7FDBDD8B.
        0x0130: 4574 6865 706E 6574 2065 7468 3020 3032 Ethernet.eth0.02
        0x0140: 3a34 323a 6163 3a31 313a 3030 3a30 3320 :42:ac:11:00:03.
        0x0150: 3e20 5d0a 6e67 202d 7363 6e20 2d2d 7365 >.].ng.-scn.--se
        0x0160: 7120 302e 3120 5b20 3c20 3120 7320 3942 q.0.1.[.<.1.s.9B
        0x0170: 3932 3646 4138 203e 205d 0a             926FA8.>.].
```

Figura 4.2: *Retorno do comando tcpdump.*

Outro requisito importante na definição do Ambiente de Avaliação para Arquitetura de Internet do Futuro (AAAIF) foi a taxa de transferência máxima entre contêineres utilizando a rede *bridge* virtual criada pelo *Docker*. Decidimos mensurar os impactos desta rede nos resultados que serão coletados, pois caso esses fossem inadequados, inviabilizariam a proposta. Para este teste descartamos alguns fatores que poderiam impactar negativamente no desempenho da rede, sendo eles:

- Proximidade física, uma vez que cada instância do contêiner está sendo executada em um mesmo *daemon* do *Docker*;
- Unidade de transmissão máxima (MTU), pois todos contêineres possuí sua interface padrão configuradas com o mesmo valor de MTU; e
- Tamanho das instâncias, sabendo que o desempenho de rede está também associado à configuração e características de cada máquina, utilizamos a mesma imagens do *Docker* para gerar diferentes instâncias porém similares entre si.

Para realizar o teste de *throughput* e coletar métricas de rede em tempo de execução utilizamos a ferramenta *iperf3* no Linux Ubuntu [109]. Esta ferramenta consiste em analisar o desempenho de *throughput* da rede, podendo testar TCP, UDP e SCPT utilizando paradigma cliente-servidor. Conforme a Tabela 4.1 e 4.2 podemos analisar 4 tipos de testes realizados com o protocolo TCP e UDP, transmissão única e direta do cliente para servidor, transmissão única e reversa do servidor para o cliente, transmissão com 10 conexões simultâneas e direta do cliente para servidor e transmissão com 10 conexões simultâneas e reversa do servidor para cliente [110, 111].

Analisando os resultados do ponto de vista de largura de banda vemos uma maior vazão de dados quando se é implementado o protocolo UDP, resultado já esperado pois o mesmo é utilizado quando são necessárias altas taxas de transmissão, porém

Tabela 4.1: *Teste iperf3 sobre TCP.*

Tipo	Intervalo	Transferência	Largura	Tamanho Pacote	Retransmissão
Única e direta	60 sec.	7.25 GBytes	1,04 Gbits/sec	1460 bytes	0 pacote
Única e reversa	60 sec.	7.36 GBytes	1,05 Gbits/sec	1460 bytes	1 pacote
Mult. e direta	60 sec.	9,05 GBytes	1,30 Gbits/sec	1460 bytes	21 pacotes
Mult. e reversa	60 sec.	8,02 GBytes	1,15 Gbits/sec	1460 bytes	82 pacotes

Tabela 4.2: *Teste iperf3 sobre UDP.*

Tipo	Intervalo	Transf.	Largura	Tamanho Pacote	Jitter	Perda
Única e direta	60 sec.	11.4 GBytes	1.63 Gbits/sec	1460 bytes	0.002 ms	0.02%
Única e reversa	60 sec.	11.8 GBytes	1.70 Gbits/sec	1460 bytes	0.003 ms	0.067%
Mult. e direta	60 sec.	20.0 GBytes	2.87 Gbits/sec	1460 bytes	0.001 ms	0.014%
Mult. e reversa	60 sec.	20.8 GBytes	2.98 Gbits/sec	1460 bytes	0.004 ms	0.011%

sem garantias de confiabilidade da entrega, controle de fluxo, retransmissão entre outros. O protocolo UDP se restringe a portas e *sockets*, transmitindo dados de forma não orientada a conexão, implementação que mais se aproxima da utilizada na NovaGenesis [111].

Desta maneira podemos concluir que independente do protocolo utilizado a rede *bridge* do *Docker* é totalmente capaz de escoar alta demanda de tráfego com qualidade sem impactar nos resultado dos testes que virão a ser realizados utilizando NovaGenesis, pois o mesmo neste caso não irá exigir alta demanda com relação ao tráfego. Uma pontuação pertinente é o fato do protótipo atual da NovaGenesis ser um *software* com alta demanda de desempenho que realiza roteamento via *software* também. Neste caso, as inúmeras sobreposições de camadas de *software* e a impossibilidade de se atuar no escalonamento [112] das tarefas no processador pode de fato impactar negativamente nos resultados quando se compara o cenário com contêineres a um ambiente com menos sobreposições de camadas, ou seja, sem a virtualização do *Docker*.

4.3 Desenvolvimento do Ambiente de Avaliação

Como facilitador nos experimentos, fez-se necessário a criação de *scripts* para automatizar as tarefas de execução relacionados as FIAs (nesse caso, a NovaGenesis), gerenciamento de contêineres e acesso *ssh* com uso da biblioteca *paramiko* da linguagem de programação *Python*. *Python* hoje é uma linguagem onipresente em sistemas Linux, sendo usado pela RedHat, HP, Google, Rackspace, IBM, entre outras. Também possui suas variações para trabalhar com Java (Jython), DotNet (IronPython) e outras. Uma linguagem que serve para qualquer propósito e se encaixa muito bem no contexto deste trabalho, que utilizaremos para automatizar as tarefas e criar cenários escaláveis. Muitos trabalhos usam *scripts* na orquestração dos cenários [88, 113–118].

Preparando o Ambiente de Desenvolvimento

O Linux Ubuntu 16.04 LTS quando instalado traz configurações e bibliotecas básicas. Como necessitamos de outros pacotes com ferramentas de uso do *Python*, de coleta de dados, pacotes de gerência de rede e uso de novas bibliotecas para NG. Foi

desenvolvido um *Shell Script* para automatizar estas tarefas. O *Script* é uma poderosa ferramenta de automação de comandos. Nos permite automatizar tarefas usando arquivos de texto executáveis, usuários ou sistemas podem executar uma série de operações.

No Apêndice B está presente o *Script* de instalação das bibliotecas e ferramentas. Este foi utilizado nos diferentes ambientes de experimentação/avaliação utilizados (máquina física, VM, e contêiner).

Estrutura do Diretório do Ambiente AAIF

O Ambiente AAIF foi escrito em Python e construído em um *design* modular em conjunto de algumas funções utilizando *Shell Script*. Os diretórios e subdiretórios armazenam os arquivos binários dos serviços NG e os *scripts* desenvolvidos. A Figura 4.3 mostra os diretórios.

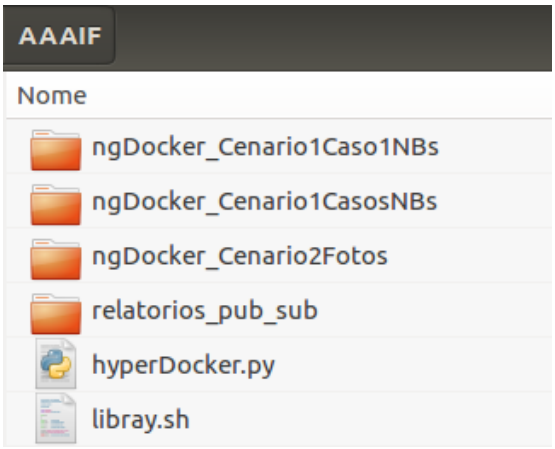


Figura 4.3: *Estrutura do diretório AAAIF.*

4.4 Considerações Parciais

Neste capítulo comprovamos com o uso de ferramentas de rede, que a bridge *Docker0* pode suportar altas taxa de tráfego de rede. Portanto, não impactando na transferência de pacotes e desempenho dos experimentos. Os testes preliminares também comprovaram, que os pacotes NG transportados pela rede de um *host* para outro, utilizaram somente a camada de enlace (sem TCP/IP).

A princípio, o objetivo no desenvolvimento do Ambiente AAIF apresentado neste capítulo, foi minimizar o tempo gasto na instalação de pacotes, tempo gasto no envio de cópias dos serviços NG, filtragem e definição dos *hosts* dos experimentos e na execução de experimentos em larga escala, além de permitir ao pesquisador obter resultados confiáveis, pois qualquer falha de comunicação entre os *hosts* ou mesmo falha de envio de arquivos (mensagens, fotos), o AAAIF se encarrega de apresentar no terminal as ocorrências.

Em segundo lugar, a solução proposta permite definir cenários de rede experimentais de qualquer tamanho, utilizando inclusive nuvem computacional pública ou pri-

vada. O usuário pode planejar como os serviços e aplicativos da arquitetura são distribuídos.

O AAAIF foi programado para suportar o maior número de *hosts* possíveis em qualquer ambiente experimental, sejam eles físicos, virtualizados ou plataforma de *testbed*. Os códigos do Apêndice D e Apêndice E foram desenvolvidos tanto para contêineres, VMs e também ambiente físico.

Outros módulos podem ser criados para suportar funcionalidades de diferentes FIAs. Com o arquivo *Dockerfile* facilmente podemos criar novas imagens com pacotes específicos de cada arquitetura apresentada no Capítulo 3.

Finalmente, o AAAIF facilita o pesquisador implementar novas funcionalidades ou exclui-las a qualquer momento. Utilizar o Ambiente AAIF com *menu* de opções. Observe que o Ambiente foi criado para experimentos de cenários específicos da NG. No entanto, pode ser facilmente modificado para suportar outras arquiteturas como CCN, XIA, RINA, SAIL, MobilityFirst, CURLING, descritas no Capítulo 3. Como o foco deste trabalho não foi comparar o desempenho entre diferentes FIAs, não implementamos funções para experimentos das FIAs citadas no nosso ambiente.

Capítulo 5

Metodologia de Avaliação para uso Conjunto com o Ambiente de Experimentação em Arquitetura de Internet do Futuro

A natureza deste trabalho de pesquisa é de caráter experimental, porque busca através da publicação e assinatura de *Name Binding* e Conteúdo armazenado em *cache*, a analise de desempenho da NG, selecionando formas para validar a arquitetura como proposta de Internet. De acordo com Wohlin et al. [119], os experimentos são ferramentas valiosas para todos os envolvidos e futuros pesquisadores na avaliação e escolha entre diferentes métodos, técnicas, linguagens e ferramentas. Essa pesquisa pode ser considerada experimental, porque objetiva validar a utilização de nomeação, resolução de nomes auto-verificáveis, armazenamento de conteúdo em *cache* de rede da NG, mediante utilização de ambiente físico e virtual controlado.

Para alcanças os objetivos desta pesquisa experimental, foram necessárias as seguintes atividades:

1. Seleção das ferramentas de *Hardware* e de *Software* necessárias para composição dos experimentos;
2. Escolha da(s) topologias que serão utilizadas;
3. Execução dos experimentos;
4. Análise e interpretação dos dados coletados.

Conforme abordado anteriormente, o atual protótipo da arquitetura NovaGenesis é composto por um total de 4 serviços principais (PGCS, GIRS, HTS e PSS) e várias aplicações, com dois cenários principais de demonstração/avaliação utilizados neste trabalho: resolução de nomes em um domínio (Cenário 1) e troca de conteúdos via *cache* de rede (Cenário 2). Outros dois cenários NG foram publicados nos periódicos [120, 121]. Considerando que o objetivo deste trabalho é a avaliação em escala da NG com foco em contêineres *Docker*, nesse Capítulo é apresentada a metodologia de avaliação da NovaGenesis utilizando o AAAIF. Para apresentar a metodologia de avaliação desenvolvida, é preciso embasá-la em configurações de equipamentos e fer-

ramentas para NovaGenesis. Nesse contexto, o SO utilizado e as configurações dos equipamentos propostos são apresentadas na Seção 5.1. Discutimos os cenários de avaliação (Cenário 1, Cenário 2) de serviços NovaGenesis na Seção 5.2. Na Seção 5.3 e Seção 5.4 abordamos como os serviços NG farão a comunicação nos experimentos a serem executados. Por fim, as métricas de avaliação definidas na metologia são apresentadas na Seção 5.5 e as considerações parciais desse Capítulo na Seção 5.6. Em resumo, esse Capítulo apresenta a metodologia, os cadernos de experimentos a serem executados em laboratório, cujos resultados são apresentados no Capítulo 6.

5.1 Equipamentos e Software

Nesta atividade, foi realizada uma pesquisa sobre qual o *Hardware* necessário para suportar os experimentos da arquitetura que permita que o pesquisador possua completo acesso ao sistema com todas permissões possíveis, que seja capaz de suportar a arquitetura NG, possua dispositivos de armazenamento físico e tenha permissão para realizar conexão com rede interna e externas. Também foi realizada uma pesquisa sobre qual o *Software* (Sistema Operacional e/ou *hypervisor* de VMs) seria necessário para executar a NG, deve ser capaz de possuir funções de rede e de escalonamento de processos, para no caso da necessidade de se ter várias instância de *Software* funcionando sob o mesmo *Hardware*. Foi selecionado o SO GNU/Linux Ubuntu 16.04 LTS pelas seguintes características: fácil instalação, configuração e utilização e compatível com as bibliotecas utilizadas na execução da NG.

A Tabela 5.1 lista os equipamentos que serão utilizados nos ambientes de avaliação dos cenários envolvendo NG. Para isso, os equipamentos foram divididos em ambientes físico (máquinas físicas disponíveis no laboratório II-1) e virtual (VMs e contêineres disponíveis no ICT Lab), além de definir os Cenários 1 e 2 em que os mesmos serão utilizados.

Tabela 5.1: *Equipamentos para avaliação de desempenho da NovaGenesis.*

Ambiente	Equipamento	Característica	Descrição	Cenário 1	Cenário 2
Virtual	Equip. 1 (1x)	Marca/Modelo	Dell PowerEdge 7640 BCC	X	X
		Processador	2x Intel R Xeon TM Silver 4114 10 Core		
		Memória	256GB (8x32GB) RDIMM DDR4 2667 MT/s		
		Placa de Rede	2x 10GbE: Intel I350; 2x 1GbE: Intel I350		
		Armazenamento	2x SSD SATA 480GB 6 Gbps; 3x HDD SATA 4TB		
Físico	Equip. 2 (1x)	Marca/Modelo	Desktop KMEX		X
		Processador	Core i5		
		Memória	16GB RAM		
		Placa de Rede	Ethernet 100 Mbits/s		
		Armazenamento	HDD SATA 500GB		
	Equip. 3 (1x)	Marca/Modelo	Laptop Dell Inspiron 14z 5423		X
		Processador	Core i5		
		Memória	4GB RAM		
		Placa de Rede	Ethernet 100 Mbits/s		
		Armazenamento	HDD SATA 500GB		
	Equip. 4 (25x)	Marca/Modelo	Desktop Dell Optiplex E7500		X
		Processador	Core 2 Duo		
		Memória	4GB RAM		
		Placa de Rede	Ethernet 100 Mbits/s		
		Armazenamento	HDD SATA 160GB		

5.2 Cenários

Nesta atividade, uma pesquisa sobre as topologias a serem utilizadas nos experimentos foi realizada. Nosso trabalho foi dividido em dois cenários de avaliação: (i) cenário de experimentos com resolução de NBs (Cenário 1); e (ii) cenário de experimentos de troca de fotos (Cenário 2). O primeiro foi desenvolvido com o objetivo de avaliar a NG, implementando soluções às limitações da Internet atual quanto à resolução de nomes (DNS). O segundo é uma proposta de distribuição de conteúdos, onde a NG assina as fotos independente de sua localização.

5.2.1 Cenário 1: Resolução de NBs

O Cenário 1 é responsável pela prova de conceito da resolução de nomes na NG. Para avaliar o impacto no desempenho da aplicação ao adicionar mais instâncias de *cache* a um domínio NG enquanto armazena a mesma quantidade de NBs, serão aplicados 4 casos utilizando diferentes topologias. Para tanto, os contêineres serão criados e receberão os *scripts* e os serviços NG através do AAAIF.

Os serviços do *core* NG (PGCS, PSS, GIRS, HTS) discutidos no Capítulo 2 (Subseção 2.2.2 e 2.2.7), bem como o Cenário 1 (Figura 2.5) com o Aplicativo *NBSimpleTestApp*, serão utilizados em cada caso. A aplicação (*NBSimpleTestApp*) é responsável por enviar milhões de publicações de NBs para o NRNCS, solicitar assinaturas de NBs aleatoriamente e então armazenar no *cache* de rede, como discutido na Subseção 2.3.1.

Os 4 casos do Cenário 1 serão descritos a seguir.

Caso 1: 1 HTS

Para um caso mais simples, representado na Figura 5.1, existe apenas um contêiner usado para o Core NG e outro para o aplicativo de ligação de nomes. Essa configuração mantém apenas uma instância de HTS, onde ela receberá os NBs publicados pelo *NBSimpleTestApp*.

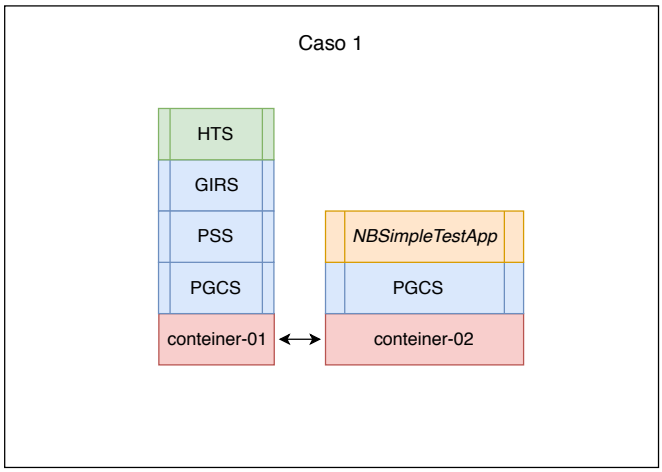


Figura 5.1: *Cenário de NBs (Cenário 1): Organização dos serviços NG para o primeiro caso contendo 1 HTS.*

Caso 2: 2 HTSes

Ampliando o primeiro caso, foi acrescentada uma nova instância de HTS na rede com objetivo de atender as demandas de armazenamento temporário de mensagens até que sejam processadas, e que apenas um HTS não atenderia a tantas requisições. Assim, com dois HTSes distribuídos na rede teremos um balanceamento de carga provavelmente mais eficiente. A Figura 5.2 mostra a distribuição dos serviços NG.

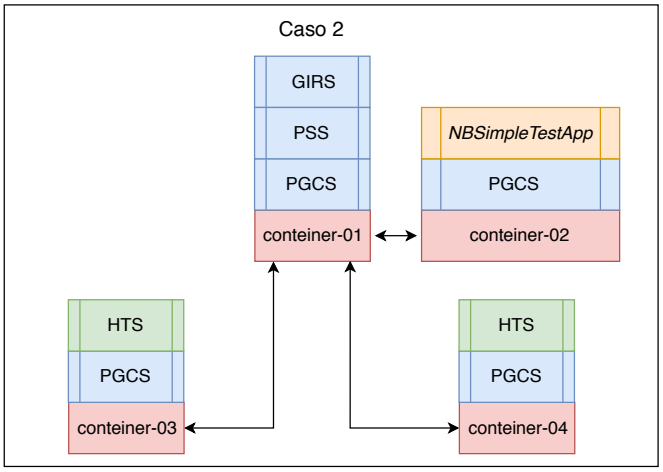


Figura 5.2: *Organização dos serviços NG para o segundo caso contendo 2 HTSes.*

Caso 3: 3 HTSes

No terceiro caso, expandimos a topologia adicionando um novo contêiner que receberá uma instância HTS. Dessa forma, a fim de distribuir o armazenamento das mensagens no *cache* de rede, foi proposto essa topologia. A Figura 5.3 mostra a distribuição dos serviços nos cinco contêineres.

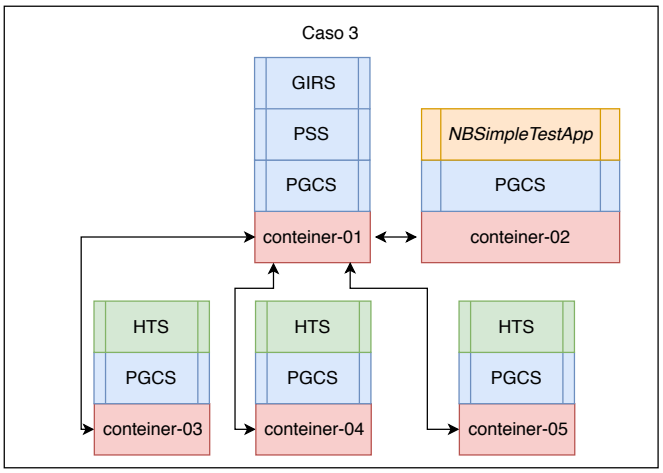


Figura 5.3: *Organização dos serviços NG para o terceiro caso contendo 3 HTSes.*

Caso 4: 4 HTSes

Por fim, no quarto caso, a topologia proposta foi planejada com seis contêineres com intuito de avaliar a escalabilidade da NG e os resultados de publicação e assinatura dos NBs. A Figura 5.4 mostra que os HTSes foram distribuídos entre quatro contêineres e um nó exclusivo executa o aplicativo *NBSimpleTestApp*.

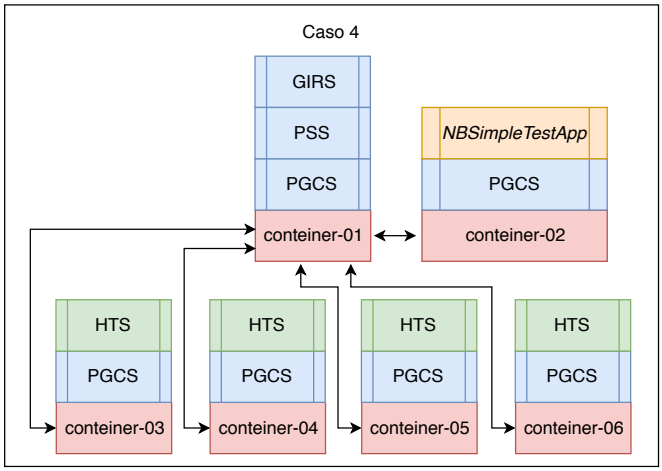


Figura 5.4: *Organização dos serviços NG para o quarto caso contendo 4 HTSes.*

A principal diferença entre os casos discutidos no Cenário 1 é a quantidade de instâncias do serviço HTS utilizada no armazenamento de NBs. Todos os contêineres executam o serviços PGCS e também a mesma versão do SO GNU/Linux Ubuntu 16.04 LTS.

5.2.2 Cenário 2

O Cenário 2 tem o intuito de avaliar a escalabilidade da NG nas assinaturas de conteúdos (fotos) via serviço de armazenamento temporário em rede (NRNCS). Desta forma, o objetivo é comprovar os benefícios da NovaGenesis quando comparada à Internet corrente e a outras FIAs. Para que essa avaliação de desempenho seja realizada em diferentes ambientes, serão considerados para o caderno de testes 3 casos diferentes, sendo estes definidos como: (i) ambiente físico com o laboratório II-1; (ii) ambiente virtual com VMs; (iii) e ambiente virtual com contêineres *Docker*.

A Figura 5.5 apresenta como foram distribuídos os serviços da NG em cada caso considerado. Na Figura 5.5 a) (ambiente físico), um servidor *desktop* executa o *Core* NG. No *laptop*, o *APPServer*, que é o lado servidor da aplicação de armazenamento decentralizado de conteúdos nomeados. Nos demais 25 computadores são executadas as aplicações clientes (Subseção 2.3.2) que enviam fotos para o servidor, as aplicações chamadas de *APPClient*. Na Figura 5.5 b) (contêiner *Docker*), no primeiro contêiner tem-se o *Core* NG. Um segundo contêiner executa o *APPServer* e os contêineres restantes executarão os *APPClient*.

Em relação ao caso de teste com VMs, a Figura 5.5 c) ilustra a configuração que inclui: uma primeira VM que recebe o *Core* NG; uma segunda VM que recebe o *APPServer*; e as demais VMs que hospedam o *APPClient*.

Todos os casos considerados contêm 27 *hosts* com o serviço PGCS executando em cada um deles e também a mesma versão do GNU/Linux. Foram utilizados 2 comutadores *Ethernet* de 100 Mbits/s com 24 portas cada, no caso que emprega o ambiente físico. Já no caso com contêineres, a transmissão de mensagens entre as instâncias serão feitas por meio da *bridge Docker*0. No caso das VMs, a troca de mensagens da NovaGenesis ocorrerá via *bridge* do KVM. O hypervisor KVM foi selecionado por fornecer virtualização ao nível de *kernel* Linux, garantindo melhor desempenho na utilização de VMs com a NG.

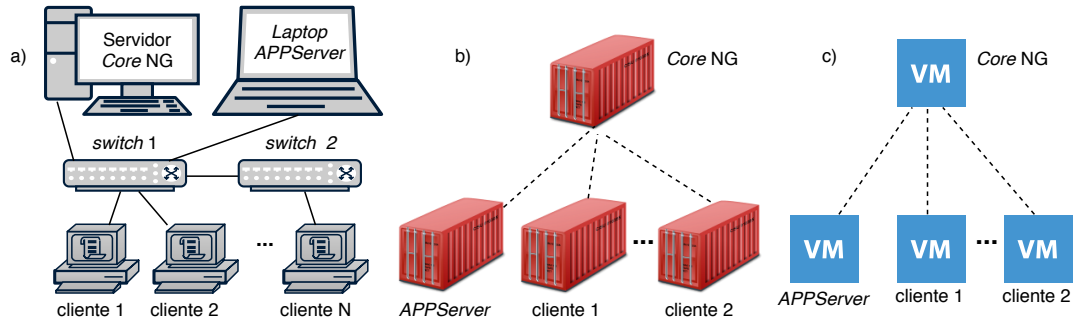


Figura 5.5: *a) Ambiente físico; b) Contêineres Docker; e c) Máquinas Virtuais em KVM.*

5.3 Caderno de Testes a Serem Realizados no Cenário de Resolução de Nomes em um Domínio

Nessa etapa da pesquisa, será definido como os experimentos serão conduzidos e quais as etapas serão necessárias para o funcionamento dos serviços da arquitetura NG. Ainda nessa etapa, serão definidos quantos ensaios de execução serão realizados para cada caso do Cenário 1. Os dados coletados serão armazenados em uma pasta confiável para posteriormente serem analisados.

Dentre os conjuntos de códigos presentes neste trabalho, será utilizado o código (Seção 4.1) responsável por gerenciar os contêineres e também o código (Seção 4.1 e Figura D.1) desenvolvido para avaliação de resolução de nomes na NovaGenesis. De forma sequencial, apresentamos os passos a serem seguidos na condução dos experimentos.

Inicialmente, serão criados dois contêineres (caso 1). Cada um deles receberá os serviços da arquitetura e *scripts* desenvolvidos. No primeiro, será executado o Core NG e no segundo aplicação *NBSimpleTestApp*.

Na fase de inicialização do Core NG, o PGCS envia uma mensagem de *Hello* para toda a rede (*broadcast*), dessa maneira informando os nomes autoverificáveis (SVNes) dos seus identificadores (apresentado na Seção 2.2.1) e também os dados da sua camada de enlace (endereço MAC, interface e identificador). A mensagem é mostrada na Figura 5.6.

Na sequência, os serviços PSS, GIRS e HTS serão executados e fazem a descoberta utilizando código *hash* como discutido na Seção 2.3. A Figura 5.7 mostra a

```
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 Ethernet eth0
02:42:ac:11:00:03 > ]
ng -scn --seq 0.1 [ < 1 s 604007EC > ]
```

Figura 5.6: *Mensagem de Hello enviada pelo PGCS para a rede.*

comunicação entre os quatro principais serviços NG.

O *Hello* do PGCS ainda contém os SVNes dos identificadores do PSS. Os SVNes do PSS são essenciais nas fases seguintes, na publicação de NBs.

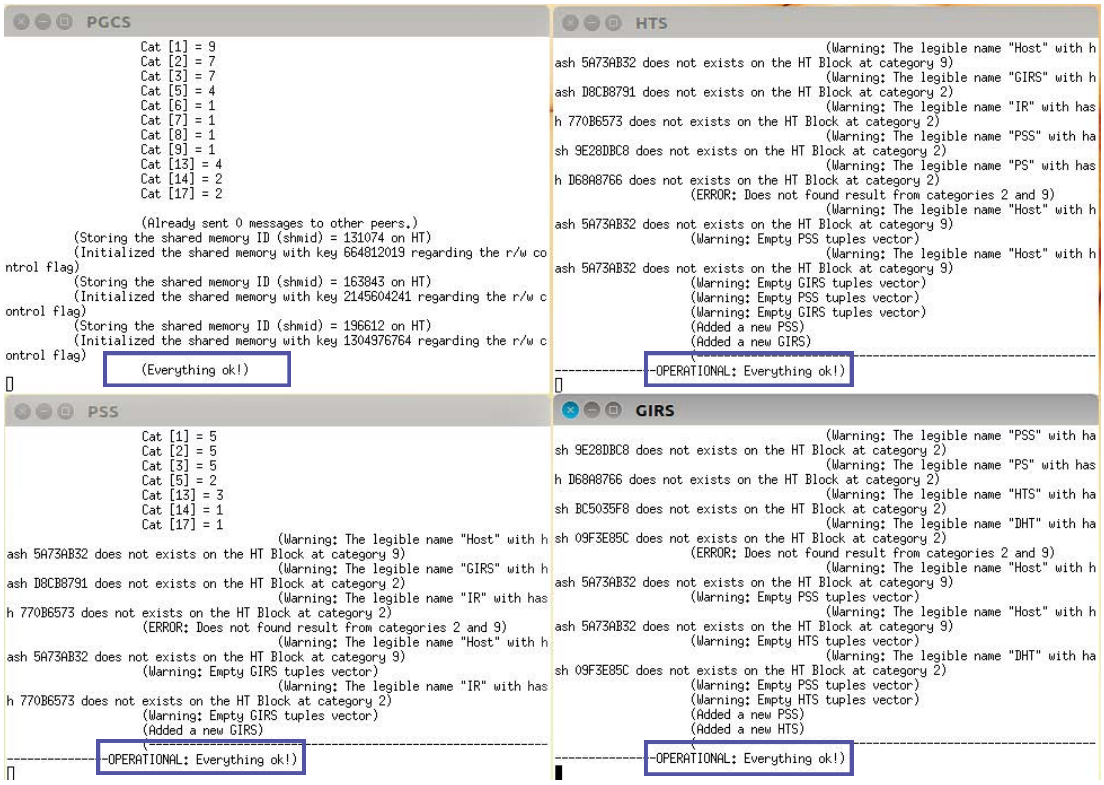


Figura 5.7: *Captura de tela, mensagens de confirmação entre os serviços (PGCS, PSS, GIRS e HTS) NG.*

O PGCS do lado do cliente, ou seja, no segundo contêiner, também envia uma mensagem de *Hello* informando seus SVNes e dados da camada de enlace. No momento que os PGCSes enviam e recebem dados, uma mensagem no PGCS do primeiro contêiner indica que a comunicação foi estabelecida. Esta mensagem pode ser vista na Figura 5.8. Onde a chave $211F772F$ é o *hash* do *HOST ID* do segundo contêiner e o endereço 02:42:ac:11:00:03 é o endereço MAC. Por fim, iniciamos as publicações de NBs executando o aplicativo *NBSimpleTestApp*.

```
(A new peer PGCS was registrered via point to point: 211F772F at the  node identified
by 02:42:ac:11:00:03)
```

Figura 5.8: *Mensagem de comunicação entre PGCSes.*

No segundo caso (Figura 5.2), utilizaremos um nó exclusivo para os serviços NG

(PGCS, PSS, GIRS) e dois nós exclusivos consistindo em HTSes e PGCSes. Onde, o PGCS também envia uma mensagem *Hello* para a rede, o PSS e GIRS descobrem o PGCS via SVNes. Os PGCSes distribuídos informam os seus SVNes. O GIRS do primeiro nó encontra os HTSes. Em todos os casos, o *NBSimpleTestApp* é implementado em um nó com um PGCS exclusivo, que atua como *gateway* entre o aplicativo e a rede.

Para os quatro casos, os NBs publicados serão variados de 1, 2, 4, 8, 16 até 32 milhões, e o número de assinaturas será fixado em 360. O número de NBs por mensagem de publicação foi definido como 1000, enquanto o número de mensagens por *burst* de publicação foi definido como 50. Cada mensagem armazena 1000 NBs e um *burst* contém 50 mensagens. O arquivo responsável por guiar o *NBSimpleTestApp* foi apresentado na Subseção 2.3.1, na Figura 2.7. Por fim, independente dos dados publicados ou número de conteúdos inscritos, cada conjunto de experimentos será de 10 ensaios para comprovar confiabilidade estatística.

5.4 Caderno de Testes para Cenário de Distribuição de Conteúdos via Serviço de Armazenamento Temporário em Rede

Os experimentos foram iniciados com base no Cenário 2 mostrado na Seção 2.3. Para cada caso (ambiente físico, VM, contêiner), iniciaremos o core NG no primeiro *host*, como mostrado na Figura 5.5. O serviço PGCS envia uma mensagem de *Hello* para toda rede. Envia os SVNes dos seus identificadores e os dados da camada de enlace. Os serviços PSS, GIRS e HTS encontram o PGCS utilizando memória compartilhada. Após o estabelecimento de comunicação entres estes serviços, o PGCS também envia para rede os SVNes dos identificadores do PSS.

Na segunda fase, uma instância do PGCS será executada em todos os *hosts*, que será responsável por encapsular mensagens NG. O PGCS do lado do core NG contém os endereços MACs dos *hosts* envolvidos no experimento. No entanto, os PGCSes dos clientes contêm o endereço MAC onde está o PGCS do core NG.

Na terceira fase, inicia-se o aplicativo (*APPServer*) responsável por armazenar o conteúdo publicado pelos clientes. O aplicativo expõe seus serviços para o *Core* NG e descobre o PSS na rede.

Visando avaliar o impacto no desempenho da NG quando aumenta-se a quantidade de fotos enviadas do cliente ao servidor, serão utilizadas 25, 50 e 100 fotografias diferentes. O valor escolhido inicialmente será 25 fotos por cliente, sendo igual ao definido no arquivo App.ini (Figura 2.9) apresentado na Subseção 2.3.2, que por sua vez é igual ao número de aplicativos clientes do laboratório II-1 (máquina física) do INATEL. Os aplicativos clientes (*APPClient*) serão inicializados com 12 segundos de intervalo, para evitar um congestionamento durante a inicialização. Além disso, eles publicam um máximo de 25 fotos a cada 1 segundo para evitar a sobrecarga de conteúdo no *cache* de rede. É importante salientar que o aplicativo servidor (*APPServer*) não realiza nenhuma publicação de foto, somente assinatura quando notificado pelo cliente pu-

blicador. Em relação às assinaturas, elas serão feitas imediatamente após o aplicativo (*APPServer*) receber uma notificação de publicação.

Para o primeiro caso, ou seja, ambiente físico utiliza-se nos experimentos do Cenário 2 o laboratório II-1 do INATEL. A Figura 5.9 mostra o laboratório dos experimentos no ambiente físico. No caso de experimentação em VMs, foram criadas 27 VMs como apresentado na Figura 5.10. Foi definida essa quantidade de VMs com o objetivo de comparar os resultados dos experimentos no ambiente virtual com o físico, utilizando o mesmo número de serviços NG. Por fim, como novidade deste trabalho mostramos na Figura 5.11 a lista de contêineres que serão utilizados nos experimentos do cenário de fotos.

Figura 5.9: *Ambiente físico de avaliação da NovaGenesis.*

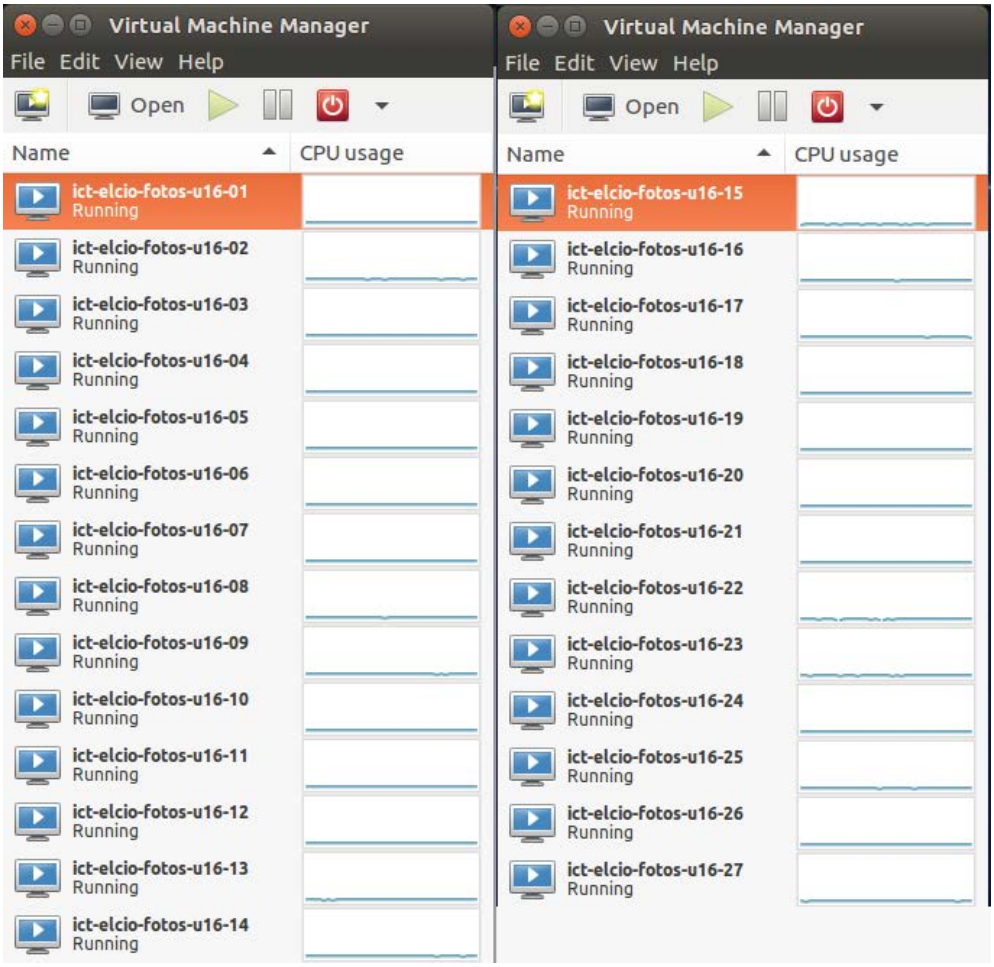


Figura 5.10: *Ambiente de VMs para experimentação da NovaGenesis.*

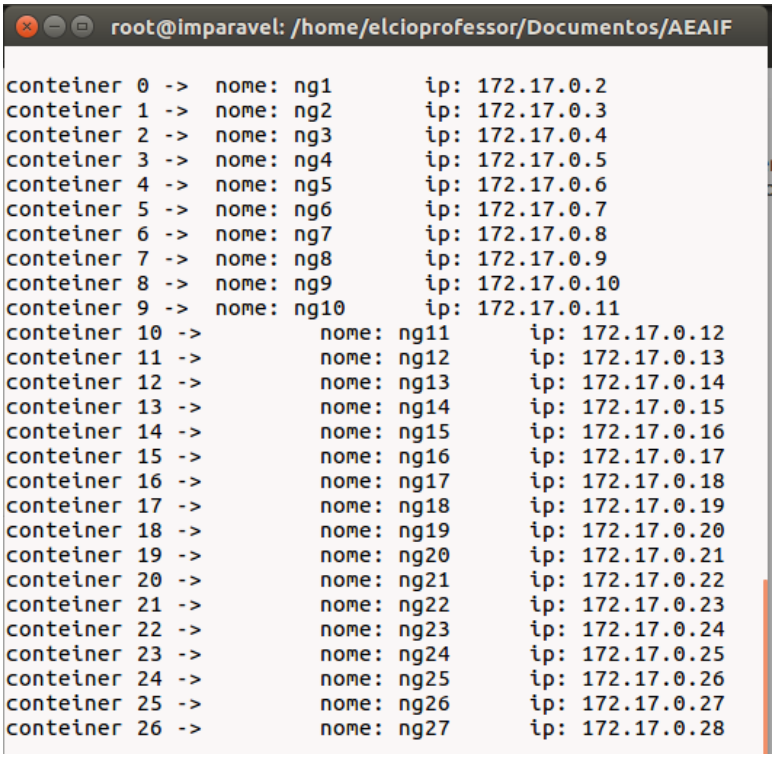


Figura 5.11: *Ambiente de contêineres para experimentação da NovaGenesis.*

Nos experimentos em contêineres *Docker* utiliza-se o algoritmo de gerência de contêineres (Figura C.1) e o algoritmo (Figura E.1) criado para avaliação de distribuição de conteúdo da NG. No entanto, os experimentos no ambiente físico e com VMs utilizam somente o algoritmo (Figura E.1) para experimentos de distribuição de conteúdo da NG, ou seja, sem o uso de contêineres. Esses algoritmos foram discutidos no Capítulo 4.

5.5 Métricas

Para avaliar os resultados obtidos nos experimentos, foram definidas métricas de avaliação de tempo de execução dos Cenários 1 e 2, bem como o tempo gasto nas publicações e assinaturas de *Name Bindings* e assinaturas de conteúdos (Fotos).

- **Tempo de Execução**: Corresponde ao tempo total, em segundo, utilizado para iniciar o experimento, execução, coleta dos resultados e finalização do cenário. Esse tempo será cronometrado para cada cenário;
- **RTT do Cenário 1**: No Cenário 1, o RTT médio de publicação/assinatura corresponde aos atrasos sofridos quando o aplicativo (*NBSimpleTestApp*) faz a oferta do serviço, o PSS aceita e assina a oferta, encaminha para o GIRS e HTS, sequencialmente. Inclui também os atrasos na transferência de NBs do *NBSimpleTestApp* para o serviço HTS;
- **RTT do Cenário 2**: O tempo médio de ida e volta da assinatura de conteúdos inclui todos os atrasos sofridos quando o aplicativo (*APPClient*) envia uma mensagem de subscrição para ao PSS, que é encaminhada para o GIRS e HTS, se-

quencialmente. Inclui também os atrasos na transferência de fotos do HTS para o destino final (*APPServer*);

5.6 Considerações Parciais

O planejamento, bem como a escolha dos equipamentos que serão utilizados para avaliar o Cenário 1 e o Cenário 2 foram discutidos nesse Capítulo.

No Cenário 1 (resolução de NBs), idealizamos quatro casos de avaliação. A principal diferença entre eles foi a quantidade de HTSes distribuídos pela rede. Uma analogia desse cenário com a infraestrutura da Internet, seria como fazer a implantação de servidor *WEB*, onde a medida que o tráfego de acesso e requisições aumentam precisamos implementar novos servidores. No nosso trabalho, foram acrescentados novos HTSes com o objetivo de otimizar o balanceamento de carga, mesmo sabendo que esse acréscimo poderia aumentar o consumo de recursos computacionais.

Para o Cenário 2, fazendo uma analogia, seria como um *drive* de armazenamento de conteúdo, onde esses conteúdos são criados e enviados para o *drive*. No entanto, embora os aplicativos cliente (*APPClient*) possam publicar e notificar as mesmas fotos para o servidor, as fotos são transferidas apenas uma vez. Os nomes auto-verificáveis (SVNes) das fotos são usados na verificação do conteúdo já armazenado, sua integridade e proveniência. Dessa forma, impedindo a duplicidade de fotos.

Tanto nos experimentos de NBs, como nos de transferência de fotos, podemos observar o passo-a-passo de execução da arquitetura NG, como por exemplo as etapas de seu ciclo de vida completo para a troca de conteúdo. Essas etapas, por sua vez, são divididas por: (i) inicialização dos principais serviços; (ii) inicialização de aplicação de distribuição de NBs e conteúdos (fotos); (iii) exposição (*exposition*) de palavras-chave (SVNes) na contratação de serviços; (iv) localização de possíveis parceiros; (v) contrato a nível de serviço; (vi) aceite do contrato entre o publicador e o assinante do conteúdo; (vii) troca do conteúdo. Os itens (iii) a (vi) somente são executados no Cenário 1.

Capítulo 6

Avaliação de Desempenho da NovaGenesis usando o Ambiente de Experimentação Desenvolvido

Este Capítulo discute os resultados obtidos em todos os experimentos, para todos os casos de todos os Cenários 1 e 2. Foram propostos dois Cenários de avaliação. O Cenário 1 publica/assina NBs utilizando quatro casos, ampliando a quantidade de instâncias de *cache* de rede e variando o número de publicações. Esta avaliação foi publicada em [122] e teve como objetivo validar a resolução de nomes NG dentro de contêineres *Docker* usando o AAAIF. O Cenário 2 utiliza o modelo publica e assina cliente/servidor publicando fotos diferentes. Foi gerada uma publicação de artigo em [123], onde o mesmo compara os resultados obtidos para distribuição de conteúdos nomeados nos três casos (ambiente físico, VMs e contêineres). Os resultados do Cenário 1 serão discutidos na Seção 6.1, enquanto os resultados do Cenário 2 na Seção 6.2. Para finalizar este Capítulo, as considerações parciais estão na Seção 6.3

6.1 Resultados de Avaliação dos Casos do Cenário 1

Nesta Seção, apresentamos os resultados do RTT médio das publicações e assinaturas nos quatro casos comentados acima. Os casos foram apresentados na Subseção 5.2.1. No diagrama de sequência da Figura 2.8, foram ilustrados os passos de comunicação do *NBSimpleTestApp* até o NRNCS.

As Figuras 6.1 e 6.2 ilustram, respectivamente, os resultados dos valores de RTT médio das publicações e assinaturas de NBs do *NBSimpleTestApp* ao NRNCS para o Caso 1, onde é utilizado 1 HTS. Na Figura 6.1, pode-se observar que, à medida que aumentamos o número de NBs publicados, há um acréscimo no RTT médio das publicações de NBs devido ao maior consumo de processamento computacional exigido. No entanto, na Figura 6.2, uma pequena inclinação positiva no gráfico pode ser observada quando o número de publicações de NBs passa de 2 para 4 milhões, aumentando consequentemente o RTT médio das assinaturas. Além disso, entre 4 e 32 milhões de publicações de NBs, o gráfico se apresenta de forma praticamente constante e linear, ou seja, para essa faixa de valores, o RTT médio de assinatura de NB

sofre poucas alterações, demostrando a escalabilidade da solução.

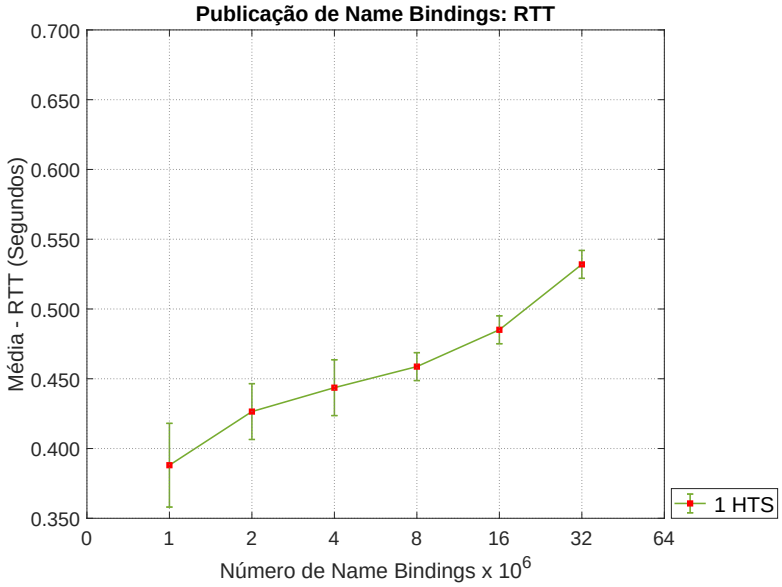


Figura 6.1: *Caso 1: RTT médio de publicações de 1000 NBs do NBSimpleTestApp até NRNCS.*

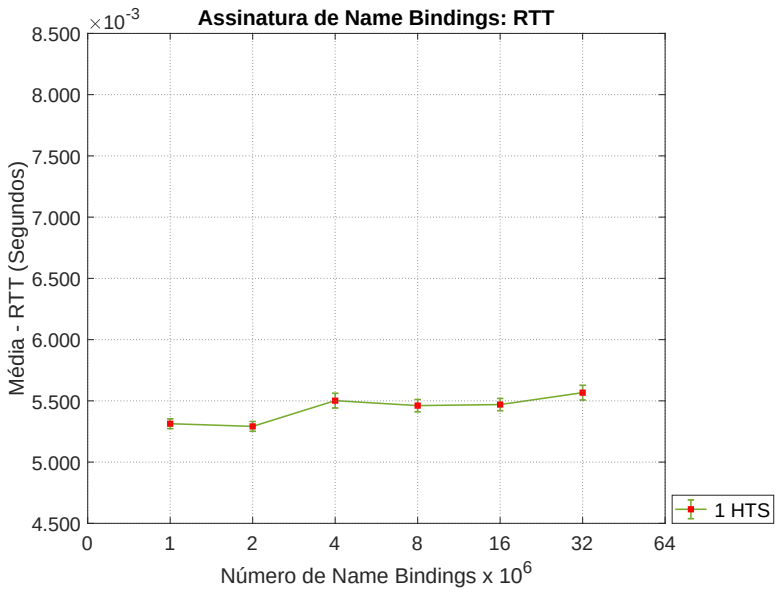


Figura 6.2: *Caso 1: RTT médio de assinatura de 1 único NB do NBSimpleTestApp até o NRNCS.*

Para o Caso 2, utilizando 2 HTSes, as Figuras 6.3 e 6.4 podem ser analisadas para avaliar, respectivamente, o RTT médio das publicações e assinaturas de NBs. Na Figura 6.3, à medida que dobramos a quantidade total de NBs publicados, o tempo médio de respostas de publicações tende a aumentar. Em contrapartida, como observado pela Figura 6.4, pode-se notar um aumento no RTT médio de assinaturas de 1 NB para a faixa de 4 a 8 milhões de publicações, e um decréscimo para as demais quantidades.

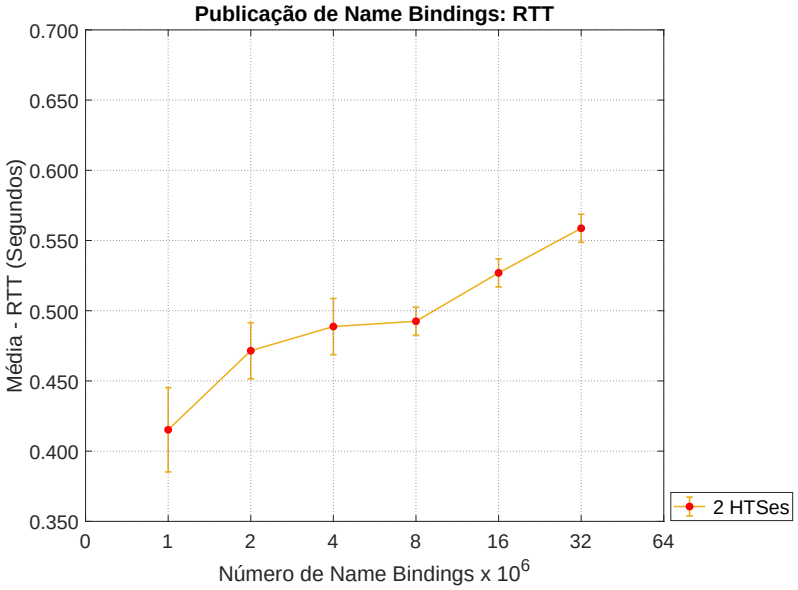


Figura 6.3: *Caso 2: RTT médio de publicações de 1000 NBs do NBSimpleTestApp até o NRNCS.*

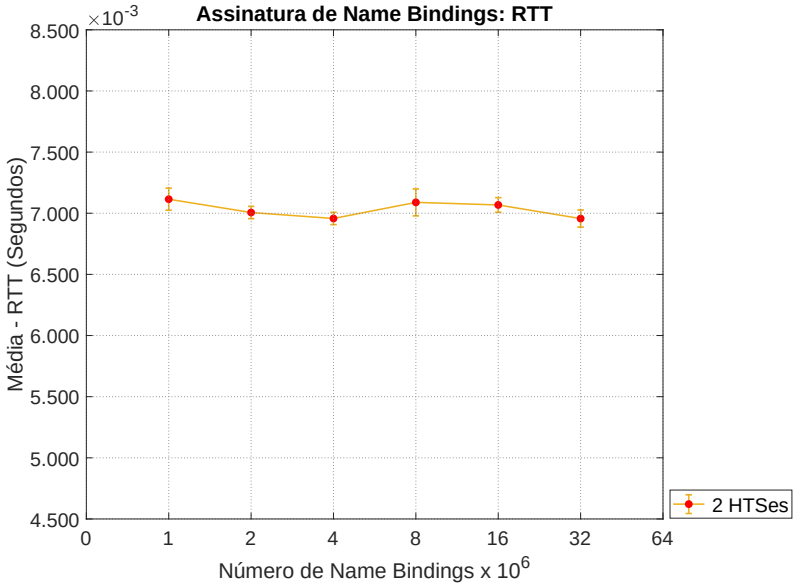


Figura 6.4: *Caso 2: RTT médio de assinatura de um único NB do NBSimpleTestApp até o NRNCS.*

A implementação de 3 HTSes nos experimentos realizados, caracterizando o Caso 3, ilustra os resultados observados nas Figuras 6.5 e 6.6. Na Figura 6.5, nota-se o aumento incremental do tempo médio de publicações de 1000 NBs com os 3 HTSes distribuídos na rede. No entanto, observa-se um decréscimo quando a quantidade total publicada é superior a 16 milhões. Já na Figura 6.6, o RTT médio da assinatura de um único NB decresce na faixa de 8 a 16 milhões de publicações NB. Para as demais quantidades de publicações, esse RTT médio tende a aumentar.

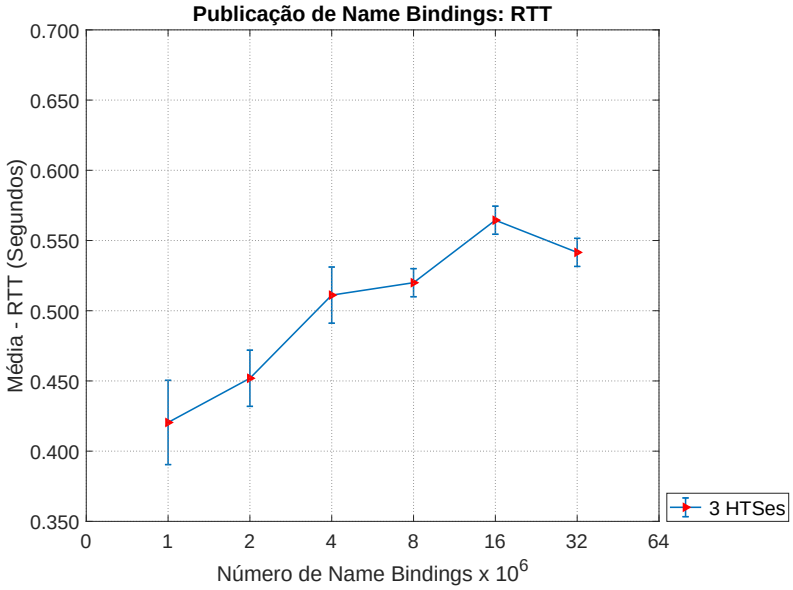


Figura 6.5: *Caso 3: RTT médio de publicações de 1000 NBs do NBSimpleTestApp até o NRNCS.*

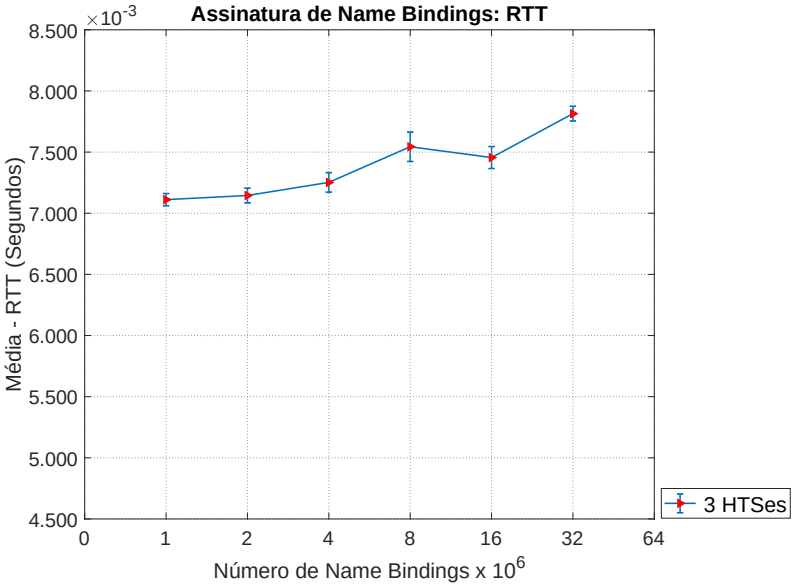


Figura 6.6: *Caso 3: RTT médio de assinatura de um único NB do NBSimpleTestApp até o NRNCS.*

Finalmente, para o Caso 4, as Figuras 6.7 e 6.8 mostram, respectivamente, os resultados dos valores de RTT médio das publicações e assinaturas de NBs utilizando 4 HTSes. Para este caso, observa-se um aumento contínuo no tempo médio de resposta de publicações em relação ao aumento da quantidade de NBs publicados, conforme ilustrado na Figura 6.7. Por outro lado, a Figura 6.8 apresenta uma oscilação no RTT médio das assinaturas de um único NB à medida que aumentamos o número de NBs total publicados. À medida que dobramos o número de publicações totais, o gráfico

responsável pelo tempo de resposta médio das assinaturas apresenta um comportamento inverso ao anterior. Por exemplo, ao incrementar o número de publicações de 1 para 2 milhões, há um aumento no RTT médio. Novamente, ao dobrar essa quantidade para 4 milhões, nota-se um decréscimo, e assim sucessivamente.

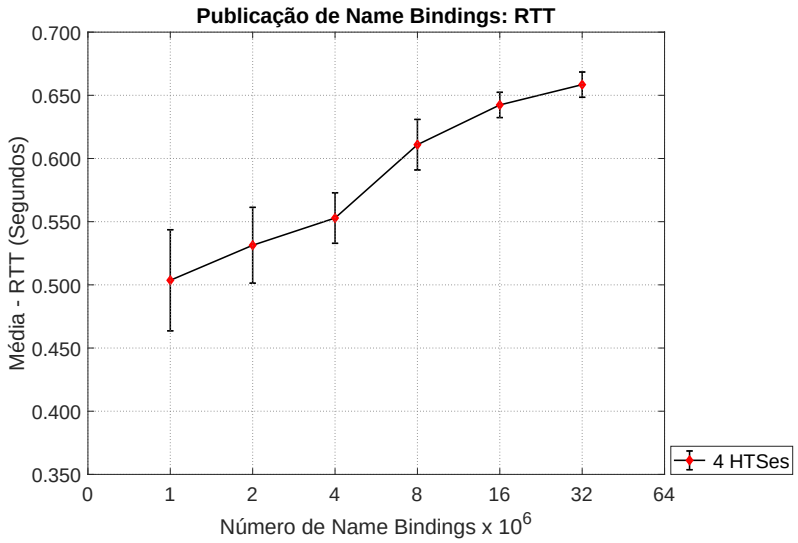


Figura 6.7: *Caso 4: RTT médio de publicações NBs do NBSimpleTestApp até NRNCS.*

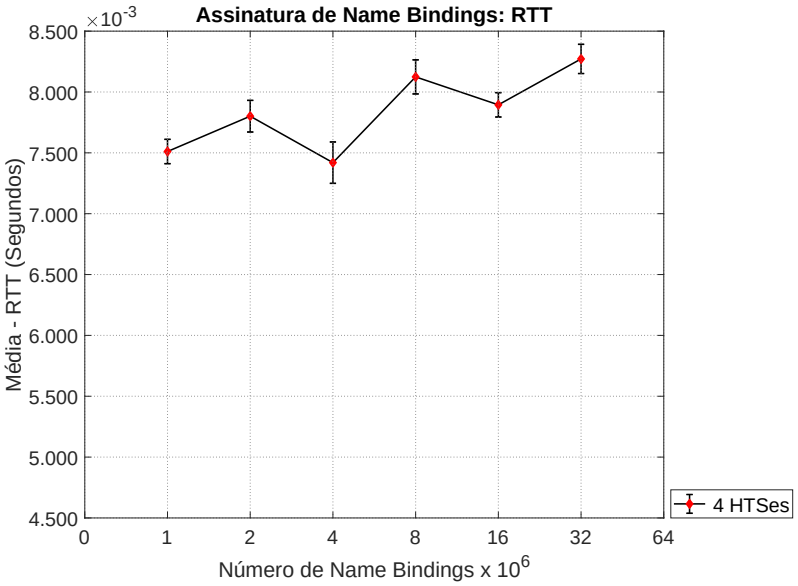


Figura 6.8: *Caso 4: RTT médio de assinatura NB do NBSimpleTestApp até NRNCS.*

Esse comportamento pode ser explicado analisando o *cgroups*, apresentado na Subseção 2.8.4. Lembre-se, o *cgroups* gerencia o uso de CPU e memória alocada em cada contêiner. A distribuição de recursos é automática aos contêineres *Docker*. O *Docker* permite o isolamento de aplicações e gerenciamento dos recursos. Quando um

contêiner utiliza baixa carga de CPU, o processado desse é liberado a outros. Portanto, no Caso 4 com 2 milhões e 4 milhões de publicações NBs, houve uma melhor utilização de recursos computacionais. Para realmente comprovar o melhor uso de recursos e o quanto de fato melhora ou piora os resultados, estudos futuros podem comprovar por meio de ferramentas de teste de CPU, memória, qual será o verdadeiro impacto de sobrecarga de recursos na avaliação da arquitetura.

Não é simples dizer o quanto dos recursos será atribuído para cada serviço NG, pois depende do comportamento de outros serviços pertencentes ao mesmo contêiner e de quantos recursos compartilhados estão alocados para ele.

A seguir, realizaremos uma comparação entre os 4 casos discutidos anteriormente, analisando o comportamento do gráfico do RTT médio de publicações e assinaturas ao aumentar a quantidade total de NBs publicados. Nesta Subseção, serão analisados os motivos de aumento e decréscimo desses tempos de resposta, levando em consideração os aspectos do uso da arquitetura NG e contêineres *Docker*.

6.1.1 Comparação dos Casos no Cenário 1

Compara-se neste ponto, os resultados do RTT médio das publicações e assinaturas de NBs nos quatro casos do Cenário 1. A Figura 6.9 mostra o RTT médio da publicação de NBs para os vários valores totais de publicação testados. Analisando o tempo de resposta médio de publicação de 1000 NBs para os 4 casos, podemos tirar as seguintes conclusões:

- Percebe-se uma grande discrepância de valores entre o RTT médio dos diferentes casos, com exceção dos casos 2 e 3 cujo processamento de resolução de nomes não variou muito, apresentando sobreposição em algumas regiões;
- O RTT de publicação é maior no Caso 4 (4 HTSes) comparado com os outros, devido ao maior número de HTSes e contêineres utilizados. Lembrado-se, no Caso 4 são criados seis contêineres para receber os serviços NG;
- Quanto maior a quantidade total de NBs publicados, maior é o RTT médio das publicações de 1000 NBs. O motivo para esse comportamento está relacionado à quantidade de HTSes instanciados, o número de NBs utilizados e também a quantidade de contêineres. Portanto, quanto maior o expresse de CPU, pior são os resultados.

Ainda na Figura 6.9, para alcançar 1 milhão de publicações no Caso 1 (1 HTS), um total de 20 *bursts* de publicação foram empregados. Em cada *burst*, 50 mensagens com 1000 NBs foram distribuídas em um intervalo de 60 segundos. Para alcançar 32 milhões de publicações, um total de 640 *bursts* de publicação formam realizados. No Caso 1, o RTT foi de 0,5320 segundos para publicar 32 milhões de NBs. Para o Caso 4, o RTT foi de 0,6584 segundos publicando a mesma quantidade de NBs, 32 milhões. Quando comparamos estes valores de tempo de resposta médio, observa-se um aumento de 23,76%.

A Figura 6.10 refere-se ao RTT médio em todos os Casos de assinatura de um único NB aleatório dentro de todo o conjunto publicado (“busca pela agulha no palheiro”). A quantidade total de assinaturas foi configurada para 360, com intervalo de um se-

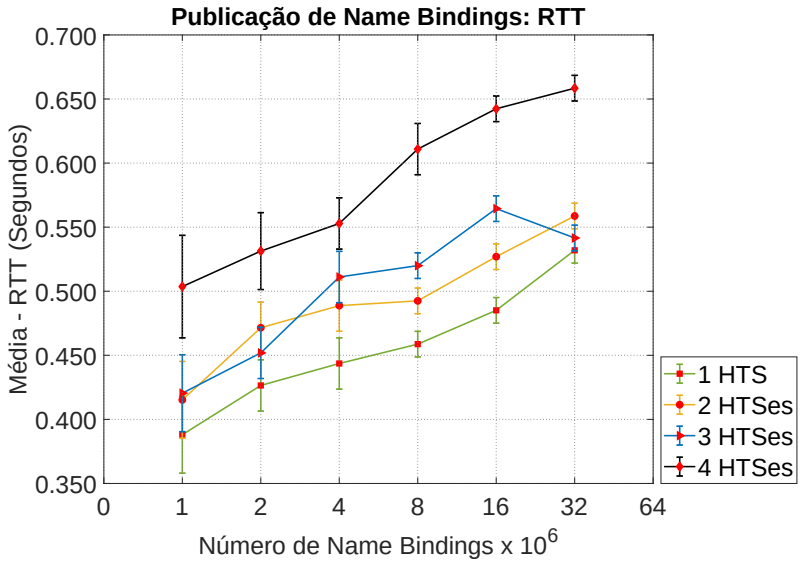


Figura 6.9: *Caso 1 a 4: RTT médio de publicações de 1000 NBs do NBSimpleTestApp até NRNCS.*

gundo entre duas consultas consecutivas. Deve-se considerar que, conforme discutido Subseção 5.3, foram realizados 10 ensaios de cada caso para comprovar uma melhor confiabilidade estatística. Os resultados referentes ao Caso 1 de assinatura NB foram os melhores, com RTT de 5,3135 ms para 1 milhão de publicações e 5,5668 ms para 32 milhões, obtendo um aumento de 4,77%.

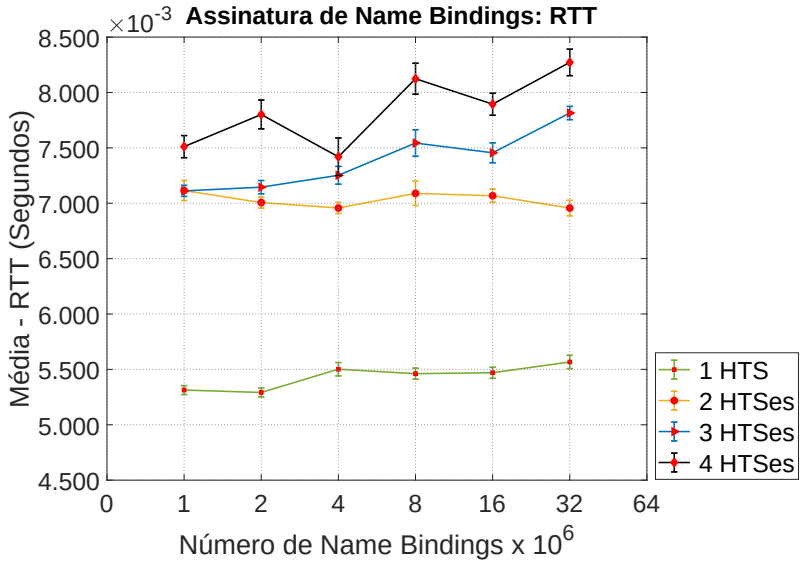


Figura 6.10: *Caso 1 a 4: RTT médio de assinatura NB do NBSimpleTestApp até NRNCS.*

Como esperado, há um aumento do tempo médio de publicação e assinatura à medida que foram adicionadas novas instâncias HTSes. Isso se deve ao fato de cada HTS

utilizar uma porcentagem de carga de CPU. Portanto, ao utilizar mais processamento no experimento, consequentemente tem-se um pior resultado.

Agora, vamos abordar a comparação do tempo total de execução dos casos, desde a criação dos contêineres, inicialização, envio de arquivos e *script* até execução de cada Caso experimental utilizando o AAAIF. A Tabela 6.1 mostra o tempo total de execução nos quatro Casos, com variação na quantidade de NBs publicados. Como ilustrado na Subseção 5.2.1 do Cenário 1, são utilizados: dois contêineres para o Caso 1, quatro para o Caso 2, cinco para o Caso 3 e seis para o Caso 4. Os tempos vão aumentando à medida que adicionamos novos contêineres com instâncias de HTSes. O maior tempo de execução foi observado no Caso 4 com 32 milhões de NBs publicados ao total, resultando em um tempo gasto de experimento de 15 horas e 3 minutos. Esse tempo se explica analisando a quantidade de publicações, sendo bem alta comparada à publicação de 1 milhão de NBs, e o maior processamento empregado ao utilizar novas instâncias de HTS distribuídas. A sobrecarga imposta na utilização de contêineres causa pouco impacto nos resultados. Novos trabalhos podem comparar a porcentagem de CPU alocada para experimentos com máquinas físicas, contêineres e VMs. Outro comparativo interessante foi cronometrar o tempo gasto sem o AAAIF, ou seja, execução manual de inicialização de todo experimento. Para cada contêiner em média, gastou-se 6 minutos. Portanto, foi contabilizado um total de 36 minutos quando implementou-se 6 contêineres. Em contrapartida, no ambiente AAAIF proposto, 2 minutos e 30 segundos foram usados para iniciar o Caso 4, resultando uma redução de 650% quando comparada ao AAAIF com experimentos manuais.

Tabela 6.1: *Comparativo do tempo gasto no experimento dos quatro Casos, com variação de quantidade de NBs*

NBs (Milhões)	Caso 1 (hh:mm)	Caso 2 (hh:mm)	Caso 3 (hh:mm)	Caso 4 (hh:mm)
1	00:30	00:32	00:34	00:36
2	00:52	00:54	00:56	01:08
4	1:40	1:48	01:57	02:19
8	3:00	3:25	03:52	04:11
16	6:25	6:35	06:58	07:33
32	12:58	13:30	14:17	15:03

6.2 Resultados de Avaliação dos Casos no Cenário 2

O Cenário 2 trata da implementação de uma rede de distribuição de conteúdos nomeados com NG. Nesta seção, serão apresentados os resultados obtidos nos experimentos com VMs, máquinas físicas e contêineres Docker. Lembrando que o objetivo é provar o conceito do ciclo de vida de serviços de mídia, incluindo a análise de desempenho da assinatura de conteúdos disponíveis em diferentes configurações de cache de redes (HTSes).

A seguir, serão analisadas as Figuras 6.11, 6.12 e 6.13, de modo a relacionar o RTT médio de assinatura de um único conteúdo de acordo com o número de fotos assinadas para cada cliente (*APPClient*), considerando respectivamente os ambientes físico (laboratório II-1) e virtual com contêineres e VMs. Deve-se destacar que para cada caso foram disponibilizados 25 clientes ao todo.

O gráfico da Figura 6.11 mostra os resultados dos experimentos no laboratório II-1 usando máquinas físicas. Ao aumentar a quantidade de 25 para 100 fotos, pode-se observar uma leve inclinação positiva, obtendo um aumento de 10%. Apesar do resultado positivo, este caso não é escalável devido a disponibilidade de computadores necessários.

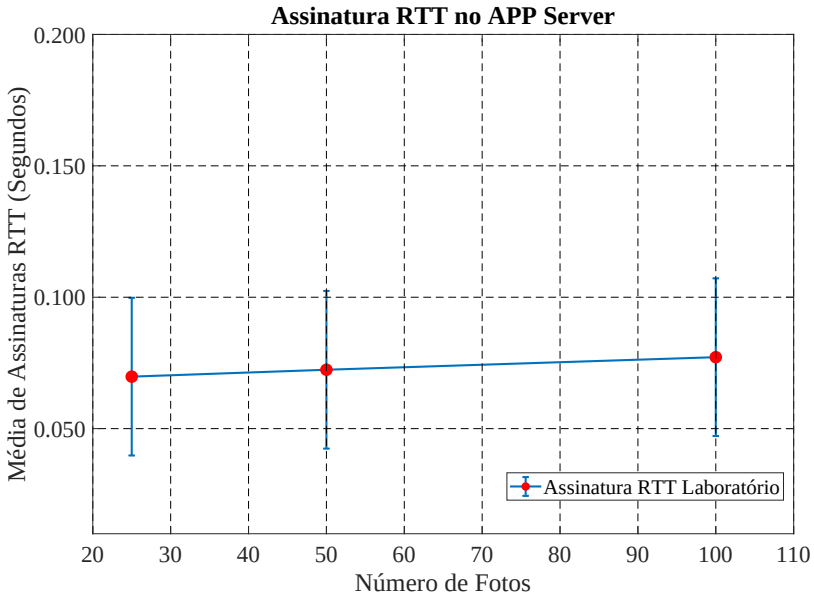


Figura 6.11: *Análise de desempenho do tempo médio RTT no laboratório II-1.*

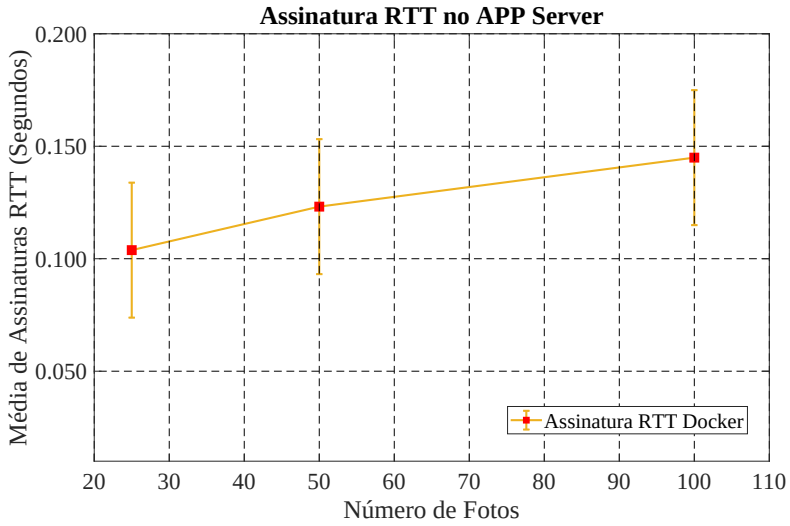


Figura 6.12: *Analise de desempenho do tempo médio RTT no Docker.*

A Figura 6.12 mostra de forma gráfica os resultados experimentais do ambiente virtual com contêineres, disponíveis no ICT Lab do Inatel. Quando é dobrado o número de fotos diferentes de 25 para 50 por cliente, a média de assinaturas RTT foi de 0.104

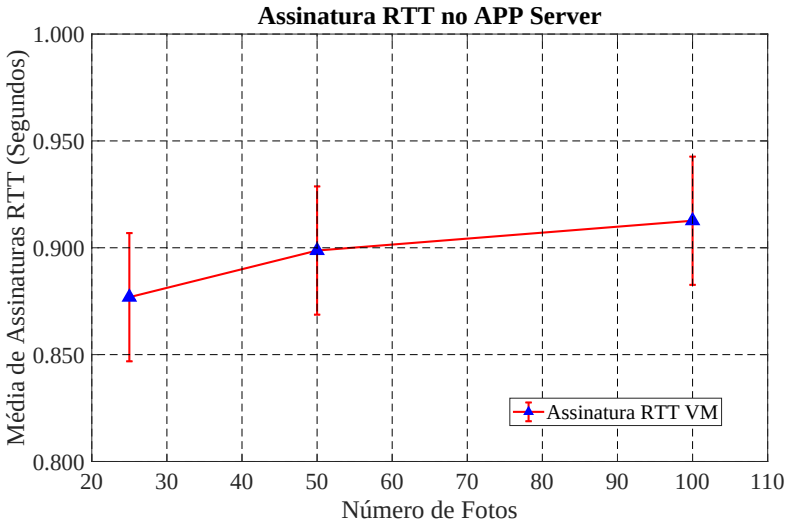


Figura 6.13: *Análise de desempenho do RTT médio de assinatura de uma única foto usando ambiente com VMs.*

a 0.123 segundos, aumento de 16%. Dobrando novamente a quantidade de fotos diferentes de 50 para 100, o tempo médio de assinaturas RTT passou de 0.123 para 0.145 segundos, ou seja, 15%. Por fim, comparando os resultados de 25 a 100 fotos, o aumento percentual foi de 28% com relação ao valor médio RTT de 0.103 a 0.145 segundos. Para realmente comprovar o melhor desempenho da arquitetura NG, novos experimentos comparando resultados com DNS e outras FIAs devem ser explorados.

O ambiente experimental com uso de contêineres *Docker* promete maior escalabilidade. Dependendo somente da capacidade computacional onde serão executados os experimentos e os recursos exigidos na arquitetura.

O gráfico na Figura 6.13 mostra os resultados obtidos nos experimentos virtuais com VMs. Observa-se uma tendência de crescimento no tempo médio de assinaturas à medida que dobramos o número de fotos diferentes.

Um comparativo entre os ambientes é realizado na Figura 6.14. Analisando os resultados, os experimentos no Laboratório II-1 apresentaram melhor desempenho. Os resultados com VMs apresentaram um tempo médio de assinaturas de um única foto com diferença significativa quando comparada com contêiner *Docker* e Laboratório II-1. Desta forma, para um experimento com 100 fotos, foi observado um aumento de 11,8 vezes em relação ao ambiente físico e 6,3 vezes quando comparado com contêineres. Isso pode ser explicado devido ao fato de que, à medida que usamos mais carga de CPU, têm-se um aumento do RTT para receber uma única foto no servidor a partir de um HTS. Como cada VM tem seu próprio SO, isso acaba impactando no desempenho. Por fim, a média de assinatura RTT para o *Docker* mostra resultados próximos de assinatura RTT ao Laboratório II-1. Porém, a vantagem do uso de contêineres *Docker* para experimentos é a facilidade de aumento de escala, agilidade para iniciar o cenário, executar os experimentos, salvar os resultados e por fim excluir todos os contêineres.

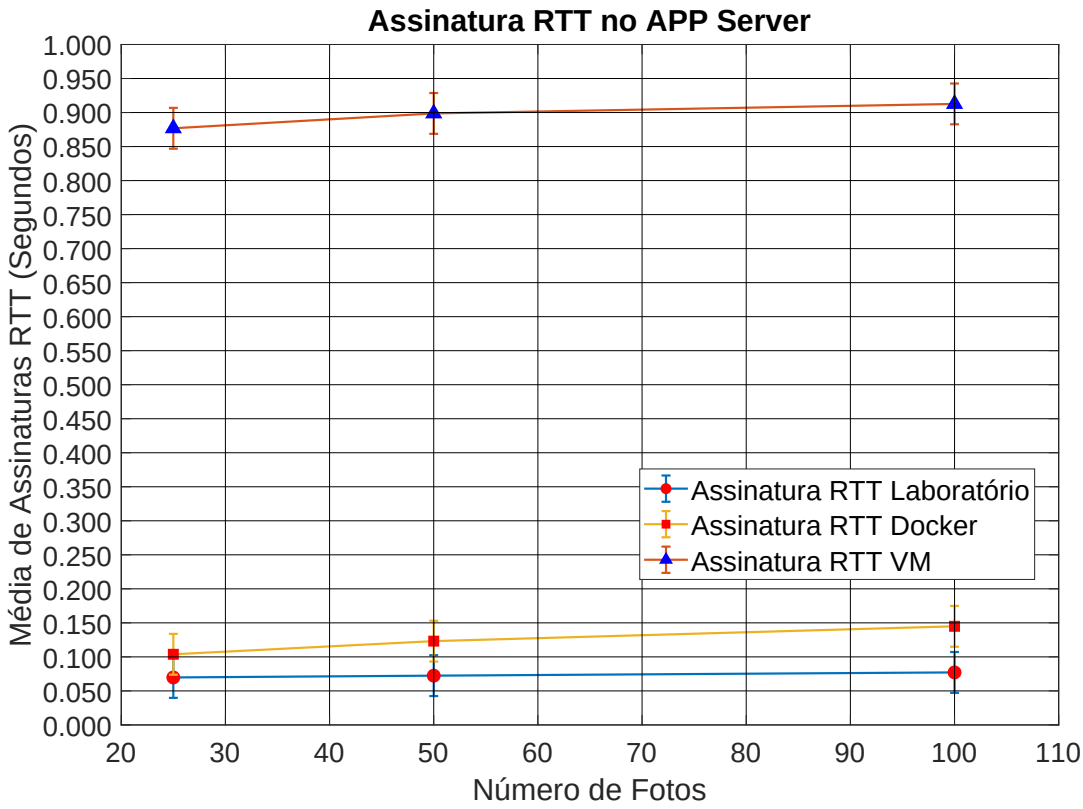


Figura 6.14: *Tempo médio de ida e volta de assinatura no aplicativo do servidor.*

Como comparativo cronometramos os experimentos em cada caso. Foram obtidos três tempos: (i) 3600 segundos para VMs; (ii) 1800 segundos com máquinas físicas e (iii) 1200 segundos com *Docker*. Essas medidas incluem o tempo de inicialização do cenário (ligar os computadores, criar e iniciar os contêineres, iniciar as VMs), execução dos aplicativos (com demandas equivalente de distribuição de conteúdos), salvar os resultados e finalizar todos os serviços. Os valores foram coletados usando o ambiente proposto (AAAIF).

Outro comparativo importante é a média do RTT de assinatura de uma única foto em segundos. A média do RTT de assinatura no laboratório aumenta à medida que dobramos o número de fotos (25, 50 e 100), resultando em (0,070, 0,072 e 0,077) segundos, seguindo a ordem crescente do número de fotos. Para *Docker* (0,104, 0,123, 0,145) segundos e VMs (0,877, 0,899, 0,913) segundos.

É possível analisar que os resultados do ambiente físico e *Docker* ficaram próximos. Destaca-se nos resultados obtidos, um melhor desempenho no cenário com *Dockers* em comparação à VMs. Como analisado no trabalho de Yadav et al. [124], os recursos computacionais são requisitados de uma forma mais eficiente em contêiner *Docker*. Credita-se esse mérito possivelmente ao *cgroups* apresentado na Subseção 2.8.4. Como o *Docker* usa *cgroups* no agrupamento dos processos em execução nos contêineres, há maior flexibilidade no gerenciamento dos recursos, sabendo que pode-se gerenciar cada grupo de processos individualmente.

6.3 Considerações Parciais

Os resultados dos experimentos no Cenário 1 (resolução de NBs em um domínio) apresentou um crescimento gradativo ao adicionar contêineres para instanciar novos HTSes. Este comportamento já era esperado, pois a medida que utiliza-se uma porcentagem maior de CPU, essa consequentemente impactou nos resultados. Mas, além disso, quando distribuímos vários HTSes como no Caso 4 (4 HTSes), é exigido um trabalho redobrado do serviço GIRS para calcular o resto da divisão de um número e escolher um dos HTSes. O interessante de inserir instâncias HTSes nos experimentos, é o balanceamento de carga, ou seja, quando os NBs chegam no PSS/GIRS são distribuídos entres os HTSes em cada contêiner. No entanto, infelizmente quando se distribui em um maior número de HTSes em conjunto com a quantidade de NBs, os resultando tendem a piorar devido ao maior processamento de CPU exigido pela arquitetura NG.

Para os experimentos do Cenário 2 (distribuição de fotos) o pior caso de RTT de assinatura de uma única foto foi com VMs. Como pode ser visto, ao utilizar um novo SO completo em cada VM piorou os resultados. A sobreposição de SOs (nativo e virtual) mais o processamento exigido da arquitetura NG, distanciou os resultados de 100 fotos (por cliente para o servidor) em 529,66% comparado com contêiner e 1082,64% comparado com máquinas física. Foi comprovado que o ambiente com contêineres *Docker* é escaláveis, pois os resultados ficaram próximos do ambiente com máquinas física e que o limite depende do poder de processamos do *HARDWARE* e melhorias no código NG.

Estudos posteriores revelaram que o problema está na troca de informações entre processos por meio de memória compartilhada, que permite a leitura de apenas uma mensagem por vez entre os processos, criando um importante gargalo. Além disso, uma única *thread* de execução é usada em cada processo do Linux para ler sequencialmente dos soquetes e da memória compartilhada. O ideal é paralelizar essa implementação. Ambos os aspectos são trabalhos futuros para melhorar o protótipo atual.

Por fim, o AAAIF provou ser ágil, preciso e escaláveis. Um ambiente que reduz em horas o trabalho manual de experimentação da NG. Também facilita a reprodução de resultados equiparáveis, não deixando a desejar erros de execução comparado a forma manual antes utilizada. Como pôde ser visto, tanto nos experimentos dos Casos do Cenário 1 quanto nos Casos do Cenário 2, em poucos segundos é possível iniciar uma avaliação da NG.

Capítulo 7

Conclusões e Trabalhos Futuros

7.1 Principais Conclusões

Essa dissertação teve como objetivo avaliar as soluções oferecidas pela arquitetura NG para lidar com as limitações da Internet atual no que tange a resolução de nomes e distribuição de conteúdos. Desta forma, foi desenvolvido um método e um novo ambiente de avaliação de arquitetura de Internet do Futuro para analisar o desempenho das propostas oferecidas pela NG, tais como o armazenamento temporário em *cache*, uso de sistema de resolução de nomes distribuído, nomeação auto-verificável e contratação dinâmica via auto-organização semântica.

Foi realizada uma pesquisa aprofundada relacionando as principais arquiteturas de Internet do Futuro e os ambientes utilizados para avaliação de desempenho. Baseado nesse estudo, pôde-se observar que a criação de grandes topologias de avaliação de FIAs consome muito tempo, e por isso é interessante utilizar ferramentas de automação de processos. Para isso, foi desenvolvido um novo ambiente de experimentação FIA nomeado AAAIF, utilizando como referência a arquitetura NG, com o diferencial de ser de código aberto, inicialmente preparado e configurado para executar experimentos com máquinas físicas, virtuais e contêineres. Não limitando a somente esses, pois o usuário pode implementar novas funcionalidades a qualquer momento, adaptando os cenários a suas necessidades.

Os experimentos mostraram que a escolha do protótipo NG como primeira FIA foi assertiva na utilização do ambiente proposto, pois não foi identificado nenhuma barreira quanto à execução dos serviços NG no AAAIF, principalmente em contêineres *Docker*. Mesmo sendo um protótipo, os Casos de uso experimentais demonstraram a NG como uma alternativa às arquiteturas atuais de DNS, IoT e redes programáveis.

Assim, esse trabalho se propôs a realizar experimentos com cenários de *Name Bindings* e distribuição de conteúdos (fotos), avaliando o desempenho da arquitetura FI NovaGenesis com foco na utilização de contêineres *Docker* e também comparando com outros ambientes (VMs, físico). Observamos que a utilização de contêineres oferece escalabilidade, agilidade e simplicidade nos processos de implantação, de construção e de entrega de serviços. Desta forma, esta dissertação foi dividida nos Capítulos descritos a seguir.

No Capítulo 1 mostrou-se de forma brevê as limitações da Internet relacionadas a nomeação, resolução de nomes e distribuição de conteúdo. Apresentou-se a motivação desse trabalho, o objetivo principal e publicações realizadas durante o período de estudos no ICT-LAB.

O Capítulo 2 explora as limitações da arquitetura atual de Internet. Limitações relacionadas a dupla semântica do modelo TCP/IP, resolução de nomes no DNS e distribuição de conteúdo. Em seguida, detalha como a arquitetura NovaGenesis implementa nomeação em Linguagem Natural, nomes auto-verificáveis, resolução de nomes, identificadores e localizadores e também os serviços PGCS, PSS, GIRS, HTS. Discutimos também os aplicativo *NBSimpleTestApp*, responsável por publicar/assinar NB e *APPClient*/*APPServer* distribuindo e guardando fotos. Demostrou-se as características únicas da NG. O Capítulo 2 também define as características relevantes do *Docker*, principal ferramenta na avaliação da NG neste trabalho.

O Capítulo 3 descreve os tralhados relacionados e as características das arquiteturas de internet do Futuro, utilizando nomeação, resolução de nome e distribuição de conteúdo em cache temporário de rede. Apresentamos também em qual ambiente cada arquitetura foi avaliada. Somente a XIA foi avaliada em contêineres *Docker* nos artigos localizados.

O Capítulo 4 explora a rede *Docker* e realiza uma analise de viabilidade com experimentos NG. Os testes comprovaram que a bridge docker0 não impactaria nos resultados das avaliações. Ainda neste Capítulo, apresentamos os algoritmos desenvolvido para o Ambiente de Avaliação de Arquiteturas de Internet do Futuro. O desenvolvimento do código foi trabalhoso, mas os estudos sobre a utilização do *Docker*, Linguagem Python e aprendizado da NG gerou grandes conhecimentos.

Para o Capítulo 5, ilustramos o Cenário 1 (resolução de NBs) nos quatro Casos e Cenário 2 (distribuição de conteúdo) em seus respectivos ambientes. Este Capítulo também descreve como serão conduzidos os experimentos.

Os resultados foram discutidos no Capítulo 6. Primeiramente apresentou-se as discussões dos quatro Casos do Cenário 1, quanto a publicações e assinaturas de NBs, comparou os resultados entre os Casos. Em seguida, discutiu-se os resultados dos experimentos do Cenário 2, com máquinas físicas, máquinas virtuais e contêineres *Docker*.

Nosso estudo mostra que a arquitetura NG é uma forte candidata para solucionar as limitações da Internet atual. Grandes desafios estão por vir, com relação as melhorias no código NG. O protótipo atual utiliza somente uma única *thread* de execução em cada processo, tanto para memória compartilhada quanto na leitura dos soquetes. O impacto disso pode ser visto na utilização de CPU. Nos experimentos, à medida que adicionamos novos *hosts* com serviços NG, a porcentagem de utilização de CPU aproximava de 100%. Nos experimentos com 27 VMs utilizou-se 98% da capacidade de processamento do ambiente virtual, ambiente apresentado na Tabela 5.1. Isso comprova o pior resultado utilizando VMs no Cenário 2.

7.2 Lições Aprendidas

Esta seção tem como objetivo de alertar novas pesquisas correlatas sobre algumas lições aprendidas durante o desenvolvimento e teste deste trabalho. As lições aprendidas foram:

- Documentação como facilitadora para usuários: o desenvolvedor deve garantir que qualquer usuário possa instalar e executar a solução desenvolvida, aumentando a porcentagem de que os interessados possam reproduzir a qualquer momento os resultados obtidos com o protótipo. Isso possibilita maior popularidade da ferramenta desenvolvida e utilização da arquitetura de FIA. Uma instalação de máquina física, virtual e/ou contêineres *Docker* com todos os pacotes e configurações essenciais, garante que isso não seja um problema ao tentar replicar os experimentos. Outro fator importante está relacionado aos passos a serem seguidos na execução dos experimentos, onde os usuários já concluíram as instalações/configurações, mas não sabem a ordem das operações.
- Planejamento do projeto: algo que pode ajudar muito o desenvolvimento de *script*, *software* e experimentos, é o diagrama de sequência. Antes de iniciar a programação, Python ou outra linguagem, desenhar o diagrama facilita a vida do próprio desenvolvedor e usuários envolvidos no projeto. Nossa experiência na criação dos *scripts* nos ensaios de experimentos em laboratório, comprovaram que teríamos economizado valorosas horas de estudos com o AAAIF e prática com o protótipo da NG.
- Experimentos com a NG em contêineres *Docker* no equipamento 1 (Tabela 5.1): Recomenda-se aos próximos pesquisadores não utilizar o terminal gnome no Linux. Nossa experiência em laboratório, comprovou que não é possível trabalhar com vários terminais *gnome* ao mesmo tempo em contêineres. Foram apresentados vários erros que para corrigir possivelmente impactaria na estrutura do SO. Por isso, utilizou-se terminais *xterm*. Com o *xterm* foram utilizados vários terminais, facilitando a visualização dos serviços NG individualmente. Mesmo utilizando o acesso remoto no equipamento 1, através de VPN não houve nenhum problema.
- Experimentos NG com máquinas físicas: Como o laboratório II-1 no INATEL era compartilhado com outros docentes, o agendamento diário muitas vezes não foi possível. Dessa forma, muitas vezes tivemos que executar os experimentos após as 22 horas ou nos finais de semana. Devido a esse compartilhamento, os equipamentos foram configurados com *dual boot*, ou seja, SO Linux e SO *Windows*. Muitas vezes, quanto tentou-se utilizar o laboratório, alguns computadores estavam com o SO corrompido, e nesse caso foi necessário uma nova instalação do SO. Outro fator importante é que toda vez que for repetir algum experimento recomenda-se reiniciar o computador, porque apesar de utilizar o mesmo *script* que exclui os serviços NG da lista de processos e limpa os semáforos, algum processo pode permanecer na lista de execução. Para não correr o risco de errar e colher resultados ruins, o melhor é reiniciar o computador. A mesma lógica pode ser aplicada quanto a utilização de VMs.

7.3 Trabalhos Futuros

Como trabalho futuro, seria interessante avaliar o desempenho de várias ICNs comparando com a NG. Utilizando contêineres *Docker* primeiramente em um ambiente controlado, ou seja, servidor local. Em seguida, realizar experimentos em plataformas de *testbed*. No entanto, nesses experimentos, muitos desafios ocorrem, especialmente nos ambientes não controlados, *testbed*. Entre outros prováveis desafios um deles seria a compatibilidade de execução das ICNs dentro de contêineres. No caso, é necessário um planejamento adicional, pois algumas ICNs necessitam de nós no caminho do percurso. Esses nós devem prover armazenamento de conteúdo em cache para comprovar o desempenho da arquitetura.

Outro estudo que seria relevante é repetir os experimentos neste trabalho dobrando a quantidade de HTSes no experimentos com NBs, dobrar o número de contêineres com aplicações clientes para publicação de fotos. Além disso, repetindo os experimentos presentes neste trabalho em novos cenários mesmo com o uso do TCP/IP. Para isso, devem realizar alterações no código NG. Por fim, seria interessante desenvolver roteadores e comutadores como no *mininet* que usa uma VM, mas com o uso de contêineres nas implementações.

Referências Bibliográficas

[1] J. Rexford and C. Dovrolis, "Future Internet Architecture: Clean-Slate versus Evolutionary Research," *Commun. ACM*, vol. 53, no. 9, pp. 36 - 40, September 2010.

[2] M. Handley, "Why the Internet only just works," *BT Technology J.*, vol. 24, no. 3, pp. 119–129, July 2006.

[3] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A Survey of Information-Centric Networking," *IEEE Commun. Mag.*, vol. 50, no. 7, pp. 26–36, July 2012.

[4] J. Pan, S. Paul, and R. Jain, "A Survey of the Research on Future Internet Architectures," *IEEE Communications Magazine*, vol. 49, no. 7, pp. 26–36, 2011.

[5] D. Griffin, M. Rio, P. Simoens, P. Smet, F. Vandeputte, L. Vermoesen, D. Bursztynowski, and F. Schamel, "Service Oriented Networking," in *2014 European Conference on Networks and Communications (EuCNC)*. IEEE, 2014, pp. 1–5.

[6] Q. Wu, Z. Li, J. Zhou, H. Jiang, Z. Hu, Y. Liu, and G. Xie, "Sofia: Toward Service-Oriented Information Centric Networking," *IEEE Network*, vol. 28, no. 3, pp. 12–18, 2014.

[7] D. Han, A. Anand, F. Dogar, B. Li, H. Lim, M. Machado, A. Mukundan, W. Wu, A. Akella, D. G. Andersen *et al.*, "XIA: Efficient Support for Evolvable Internetworking," in *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation NSDI 12*, 2012, pp. 309–322.

[8] 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 Network*, vol. 28, no. 2, pp. 20–25, 2014.

[9] G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, and G. C. Polyzos, "A Survey of Information-Centric Networking Research," *IEEE communications surveys & tutorials*, vol. 16, no. 2, pp. 1024–1049, 2013.

[10] 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*, 2011, pp. 1–6.

[11] R. Atkinson, "Naming in the Internet Architecture: Provenance and Current Issues," *RN*, vol. 5, no. 28, p. 1, 2005.

[12] P. Stuckmann and R. Zimmermann, "European Research on Future Internet Design," *IEEE Wireless Communications*, vol. 16, no. 5, pp. 14–22, 2009.

[13] S. Arshad, M. A. Azam, M. H. Rehmani, and J. Loo, "Recent Advances in Information-Centric Networking-Based Internet of Things (icn-iot)," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2128–2158, 2018.

[14] 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.

[15] J. Szefer and R. B. Lee, "Architectural Support for Hypervisor-Secure Virtualization," *ACM SIGPLAN Notices*, vol. 47, no. 4, pp. 437–450, 2012.

[16] N. Jain and S. Choudhary, "Overview of Virtualization in Cloud Computing," in *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*. IEEE, 2016, pp. 1–4.

[17] P. Padala, X. Zhu, Z. Wang, S. Singhal, K. G. Shin *et al.*, "Performance Evaluation of Virtualization Technologies for Server Consolidation," *HP Labs Tec. Report*, vol. 137, 2007.

[18] "The VmWare WebSite," https://www.vmware.com/br/solutions/virtualization.html, Acessado em: 15 de Janeiro de 2021.

[19] "The Microsoft Hyper-V WebSite," https://azure.microsoft.com/pt-br/services/virtual-machines/, Acessado em: 15 de Janeiro de 2021.

[20] "The XEN Project WebSite," https://xenproject.org/, Acessado em: 15 de Janeiro de 2021.

[21] "The RedHat WebSite," https://www.redhat.com/en/topics/virtualization/what-is-KVM, Acessado em: 15 de Janeiro de 2021.

[22] S. Singh and N. Singh, "Containers & Docker: Emerging Roles & Future of Cloud Technology," in *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*. IEEE, 2016, pp. 804–807.

[23] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. Lightweight Virtualization: A Performance Comparison," in *2015 IEEE International Conference on Cloud Engineering*. IEEE, 2015, pp. 386–393.

[24] "Docker Documentation," https://docs.docker.com/v17.09/, Acessado em: 15 de Março de 2021.

[25] A. Mouat, *Using Docker: Developing and Deploying Software with Containers*. "O'Reilly Media, Inc.", 2015.

[26] C. Guimarães, J. Quevedo, R. Ferreira, D. Corujo, and R. L. Aguiar, "Exploring Interoperability Assessment for Future Internet Architectures roll out," *Journal of Network and Computer Applications*, vol. 136, pp. 38–56, 2019.

[27] Y. Wang, F. Esposito, and I. Matta, "Demonstrating RINA Using the GENI Testbed," in *2013 Second GENI Research and Educational Experiment Workshop*. IEEE, 2013, pp. 93–96.

[28] E. Grasa, L. Bergesio, M. Tarzan, D. Lopez, J. Day, and L. Chitkushev, "Se-

amless Network Renumbering in RINA: Automate Address Changes without Breaking Flows!" in *2017 European Conference on Networks and Communications (EuCNC)*. IEEE, 2017, pp. 1–6.

[29] S. Vrijders, D. Staessens, M. Capitani, and V. Maffione, "Rumba: A Python Framework for Automating Large-Scale Recursive Internet Experiments on GENI and FIRE+," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2018, pp. 324–329.

[30] R. Atkinson, S. Bhatti, and S. Hailes, "Evolving the Internet Architecture Through Naming," *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 8, pp. 1319–1325, 2010.

[31] S. Al-Sarawi, M. Anbar, K. Alieyan, and M. Alzubaidi, "Internet of Things (IoT) Communication Protocols," in *2017 8th International Conference on Information Technology (ICIT)*. IEEE, 2017, pp. 685–690.

[32] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish, "A Layered Naming Architecture for the Internet," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 343–352, 2004.

[33] M. Walfish, H. Balakrishnan, and S. Shenker, "Untangling the Web from DNS," in *NSDI*, vol. 4, 2004, pp. 17–17.

[34] V. Pappas, D. Massey, A. Terzis, and L. Zhang, "A Comparative Study of the DNS Design with DHT-Based Alternatives," in *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*. IEEE, 2006, pp. 1–13.

[35] Anders Eriksson and Adeel Mohammad Malik, "A dns-based information-centric network architecture open to multiple protocols for transfer of data objects," in *2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, July 2018, pp. 01–08.

[36] G. Chuanxiong and Z. Shaoren, "Analysis and Evaluation of the TCP/IP Protocol Stack of Linux," in *WCC 2000-ICCT 2000. 2000 International Conference on Communication Technology Proceedings (Cat. No. 00EX420)*, vol. 1. IEEE, 2000, pp. 444–453.

[37] D. Massey, "A Comparative Study of the DNS Design with DHT-Based Alternatives," in *In the Proceedings of IEEE INFOCOM'06*. Citeseer, 2006.

[38] A. M. Alberti, D. Mazzer, M. M. Bontempo, L. H. de Oliveira, R. da Rosa Righi, and A. C. Sodre Jr, "Cognitive Radio in the Context of Internet of Things Using a Novel Future Internet Architecture Called NovaGenesis," *Computers & Electrical Engineering*, vol. 57, pp. 147–161, 2017.

[39] M. Komu, M. Sethi, and N. Beijar, "A Survey of Identifier-Locator Split Addressing Architectures," *Computer Science Review*, vol. 17, pp. 25–42, 2015.

[40] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking Named Content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, 2009, pp. 1–12.

[41] J. Day, *Patterns in Network Architecture: A Return to Fundamentals*.

[42] C. Guimarães, J. Quevedo, R. Ferreira, D. Corujo, and R. L. Aguiar, "Exploring Interoperability Assessment for Future Internet Architectures roll out," *Journal of Network and Computer Applications*, vol. 136, pp. 38–56, 2019.

[43] F. Bronzino, D. Raychaudhuri, and I. Seskar, "Experiences with Testbed Evaluation of the MobilityFirst Future Internet Architecture," in *2015 European Conference on Networks and Communications (EuCNC)*. IEEE, 2015, pp. 507–511.

[44] 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.

[45] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26–36, 2012.

[46] "The MD5 WebSite," https://www.miraclesalad.com/webtools/md5.php, acessado em: 2019.

[47] "A. Appleby," https://github.com/aappleby/smhasher/wiki/MurmurHash3, acessado em: 2019.

[48] D. Happ, N. Karowski, T. Menzel, V. Handziski, and A. Wolisz, "Meeting IoT Platform Requirements with Open Pub/Pub Solutions," *Annals of Telecommunications*, vol. 72, no. 1-2, pp. 41–52, 2017.

[49] W. Ramírez, X. Masip-Bruin, M. Yannuzzi, R. Serral-Gracia, A. Martínez, and M. S. Siddiqui, "A Survey and Taxonomy of ID/Locator Split Architectures," *Computer Networks*, vol. 60, pp. 13–33, 2014.

[50] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[51] V. Tiwari, "Study of Internet of things (IoT): A Vision, Architectural Elements, and Future Directions," *International Journal of Advanced Research in Computer Science*, vol. 7, no. 7, 2016.

[52] I. A. T. Hashem, V. Chang, N. B. Anuar, K. Adewole, I. Yaqoob, A. Gani, E. Ahmed, and H. Chiroma, "The Role of Big Data in Smart City," *International Journal of Information Management*, vol. 36, no. 5, pp. 748–758, 2016.

[53] A. Alberti, E. do Rosário, G. Cassiano, J. dos Santos, V. D'Ávila, and J. Carneiro, "Performance Evaluation of NovaGenesis Information-Centric Network," in *2017 2nd International Multidisciplinary Conference on Computer and Energy Science (SpliTech)*. IEEE, 2017, pp. 1–6.

[54] P. O. Abaev, "On SIP Session Setup Delay Modeling in Next Generation Networks," in *International Congress on Ultra Modern Telecommunications and Control Systems*. IEEE, 2010, pp. 1125–1131.

[55] A. Martin, S. Raponi, T. Combe, and R. Di Pietro, "Docker Ecosystem-Vulnerability Analysis," *Computer Communications*, vol. 122, pp. 30–43, 2018.

[56] T. Combe, A. Martin, and R. Di Pietro, "To Docker or not to Docker: A Security

Perspective," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 54–62, 2016.

[57] D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[58] "The OpenVz WebSite," https://openvz.org/, Acessado em: 15 de Janeiro de 2021.

[59] A. M. Joy, "Performance Comparison Between Linux Containers and Virtual Machines," in *2015 International Conference on Advances in Computer Engineering and Applications*. IEEE, 2015, pp. 342–346.

[60] Z. Zhuang, C. Tran, J. Weng, H. Ramachandra, and B. Sridharan, "Taming Memory Related Performance Pitfalls in Linux Cgroups," in *2017 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2017, pp. 531–535.

[61] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-Scale Cluster Management at Google with Borg," in *Proceedings of the Tenth European Conference on Computer Systems*, 2015, pp. 1–17.

[62] Z. Kozhirbayev and R. O. Sinnott, "A Performance Comparison of Container-Based Technologies for the Cloud," *Future Generation Computer Systems*, vol. 68, pp. 175–182, 2017.

[63] Á. Kovács, "Comparison of Different Linux Containers," in *2017 40th International Conference on Telecommunications and Signal Processing (TSP)*. IEEE, 2017, pp. 47–51.

[64] K. Indrasiri and P. Siriwardena, "Deploying and Running Microservices," in *Microservices for the Enterprise*. Springer, 2018, pp. 219–262.

[65] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, "Cloud Container Technologies: A State-Of-The-Art Review," *IEEE Transactions on Cloud Computing*, 2017.

[66] C. Lagoze and H. Van de Sompel, "The Open Archives Initiative: Building a Low-Barrier Interoperability Framework," in *Proceedings of the 1st ACM/IEEE-CS joint conference on Digital libraries*, 2001, pp. 54–62.

[67] "The Kubernetes WebSite, howpublished = https://kubernetes.io/, note = Acessado em: 15 de Março, 2020."

[68] C. Pahl, "Containerization and the PAAS Cloud," *IEEE Cloud Computing*, vol. 2, no. 3, pp. 24–31, 2015.

[69] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and [d]eployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[70] N. S. de Morais, A. C. Drummond, and A. P. F. de Araújo, "Proposal for a Virtualized Data Center Migration model for Private Cloud model for the new Brazilian Army Information Technology Pole," in *2017 12th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE, 2017, pp. 1–7.

[71] C. Strachey, "Time-Sharing in Large Fast Computers in Proceedings of the International Conference on Information Processing," *UNESCO, June*, vol. 1, p. 959, 1959.

[72] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An Updated Performance

Comparison of Virtual Machines and Linux Containers," in *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2015, pp. 171–172.

[73] Z. Zhuang, C. Tran, J. Weng, H. Ramachandra, and B. Sridharan, "Taming Memory Related Performance Pitfalls in Linux Cgroups," in *2017 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2017, pp. 531–535.

[74] P. Bellasi, G. Massari, and W. Fornaciari, "Effective Runtime Resource Management Using Linux Control Groups with the Barbequertrm Framework," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 14, no. 2, pp. 1–17, 2015.

[75] Q. Xu, M. Awasthi, K. T. Malladi, J. Bhimani, J. Yang, and M. Annavaram, "Docker Characterization on High Performance SSDs," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2017, pp. 133–134.

[76] H. Zeng, B. Wang, W. Deng, and W. Zhang, "Measurement and Evaluation for Docker Container Networking," in *2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*. IEEE, 2017, pp. 105–108.

[77] P. C. V. Varma, V. V. Kumari, S. V. Raju *et al.*, "Analysis of a Network io Bottleneck in Big Data Environments Based on Docker Containers," *Big Data Research*, vol. 3, pp. 24–28, 2016.

[78] I. Cerrato, F. Risso, R. Bonafiglia, K. Pentikousis, G. Pongrácz, and H. Woesner, "Composer: A Compact Open-Source Service Platform," *Computer Networks*, vol. 139, pp. 151–174, 2018.

[79] Y. Xu, V. Mahendran, and S. Radhakrishnan, "SDN Docker: Enabling Application Auto-Docking/Undocking in EDGE Switch," in *2016 IEEE conference on computer communications workshops (INFOCOM WKSHPS)*. IEEE, 2016, pp. 864–869.

[80] P. H. V. Guimaraes, L. H. G. Ferraz, J. V. Torres, D. M. Mattos, I. D. Alvarenga, C. S. Rodrigues, O. C. M. Duarte *et al.*, "Experimenting Content-Centric Networks in the Future Internet Testbed Environment," in *2013 IEEE International Conference on Communications Workshops (ICC)*. IEEE, 2013, pp. 1383–1387.

[81] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named Data Networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

[82] D. Barrera, L. Chuat, A. Perrig, R. M. Reischuk, and P. Szalachowski, "The SCION Internet Architecture," *Communications of the ACM*, vol. 60, no. 6, pp. 56–65, 2017.

[83] W. Ding, Z. Yan, and R. H. Deng, "A Survey on Future Internet Security Architectures," *IEEE Access*, vol. 4, pp. 4374–4393, 2016.

[84] W. K. Chai, G. Pavlou, G. Kamel, K. V. Katsaros, N. Wang, and S. Member,

"A Distributed Interdomain Control System for Information-Centric Content Delivery," *IEEE Systems Journal*, vol. PP, pp. 1–12, 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8430531/

[85] S. Duquennoy, A. Elsts, B. Al Nahas, and G. Oikonomo, "Tsch and 6tisch for Contiki: Challenges, Design and Evaluation," in *2017 13th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, 2017, pp. 11–18.

[86] "Recursive Internetwork Architecture (RINA)," http://csr.bu.edu/rina/index.html, acessado em: 15 de Março, 2020.

[87] E. Trouva, E. Grasa, J. Day, and S. Bunch, "Layer Discovery in RINA Networks," in *2012 IEEE 17th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. IEEE, 2012, pp. 368–372.

[88] S. Vrijders, D. Staessens, M. Capitani, and V. Maffione, "Rumba: A Python Framework for Automating Large-Scale Recursive Internet Experiments on GENI and FIRE+," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2018, pp. 324–329.

[89] E. Grasa, O. Rysavy, O. Lichtner, H. Asgari, J. Day, and L. Chitkushev, "From Protecting Protocols to Layers: Designing, Implementing and Experimenting with Security Policies in RINA," in *2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–7.

[90] B. Ahlgren, P. A. Aranda, P. Chemouil, S. Oueslati, L. M. Correia, H. Karl, M. Söllner, and A. Welin, "Content, Connectivity, and Cloud: Ingredients for the Network of the Future," *IEEE Communications Magazine*, vol. 49, no. 7, pp. 62–70, 2011.

[91] A. El Mougy, "On the Integration of Software-Defined and Information-Centric Networking Paradigms," in *2015 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*. IEEE, 2015, pp. 105–110.

[92] N. Fotiou, P. Nikander, D. Trossen, and G. C. Polyzos, "Developing Information Networking Further: From PSIRP to PURSUIT," in *International Conference on Broadband Communications, Networks and Systems*. Springer, 2010, pp. 1–13.

[93] C. Dannewitz, D. Kutscher, B. Ohlman, S. Farrell, B. Ahlgren, and H. Karl, "Network of Information NetInf-an Information-Centric Networking Architecture," *Computer Communications*, vol. 36, no. 7, pp. 721–735, 2013.

[94] T. Kaida and O. Mizuno, "Applying NetInf for the M2M Service Platform," in *The 16th Asia-Pacific Network Operations and Management Symposium*. IEEE, 2014, pp. 1–4.

[95] X. Liu, W. Trappe, and Y. Zhang, "Secure Name Resolution for Identifier-to-Locator Mappings in the Global Internet," in *2013 22nd International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2013, pp. 1–7.

[96] "The MobilityFirst WebSite," http://mobilityfirst.winlab.rutgers.edu/

TechApproach.html, acessado em: 19 de Março, 2020.

[97] A. Venkataramani, J. F. Kurose, D. Raychaudhuri, K. Nagaraja, M. Mao, and S. Banerjee, "MobilityFirst: A Mobility-Centric and Trustworthy Internet Architecture," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 74–80, 2014.

[98] R. Khondoker, B. Nugraha, R. Marx, and K. Bayarou, "Security of Selected Future Internet Architectures: A Survey," in *2014 Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. IEEE, 2014, pp. 433–440.

[99] W. Chai, N. Wang, I. Psaras, G. Pavlou, C. Wang, G. Garcia de Blas, F. Ramon-Salguero, L. Liang, S. Spirou, A. Beben, and E. Hadjioannou, "Curling: Content-Ubiquitous Resolution and Delivery Infrastructure for Next-Generation Services," *IEEE Communications Magazine*, vol. 49, no. 3, pp. 112–120, 3 2011. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5723808http://ieeexplore.ieee.org/document/5723808/

[100] A. Abidi, S. Gammar, F. Kamoun, W. Dabbous, and T. Turletti, "Memory Management Optimization for Content Routers in DONA," in *2015 IEEE 14th International Symposium on Network Computing and Applications*. IEEE, 2015, pp. 85–89.

[101] N. Fotiou, P. Nikander, D. Trossen, and G. C. Polyzos, "Developing Information Networking Further: From PSIRP to PURSUIT," in *International Conference on Broadband Communications, Networks and Systems*. Springer, 2010, pp. 1–13.

[102] R. P. Chaib and A. M. Alberti, "A Simple Solution for IoT Experimentation in the Context of Future Internet Architectures," in *Anais do VIII Workshop de Pesquisa Experimental da Internet do Futuro*. SBC, 2017.

[103] J. Hong, T.-W. You, and Y.-G. Hong, "Name Resolution Service for CCN," in *2017 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2017, pp. 1276–1279.

[104] "Manpage of TCPDUMP," https://www.tcpdump.org/tcpdump_man.html, acessado em: 15-03-2019.

[105] "Bridging Ethernet Connections (as of Ubuntu 16.04)," https://help.ubuntu.com/community/NetworkConnectionBridge, acessado em: 15-03-2019.

[106] Y. Peng and H. Wang, "Design and Implementation of Network Instruction Detection System Based on Snort and NTOP," in *2012 International Conference on Systems and Informatics (ICSAI2012)*. IEEE, 2012, pp. 116–120.

[107] V. Ndatinya, Z. Xiao, V. R. Manepalli, K. Meng, and Y. Xiao, "Network Forensics Analysis Using Wireshark," *International Journal of Security and Networks*, vol. 10, no. 2, pp. 91–106, 2015.

[108] P. Goyal and A. Goyal, "Comparative Study of Two most Popular Packet Sniffing Tools-Tcpdump and Wireshark," in *2017 9th International Conference on Computational Intelligence and Communication Networks (CICN)*. IEEE, 2017, pp. 77–81.

[109] "iPerf - The Ultimate Speed Test Tool for TCP, UDP and SCTP," https://iperf.fr/iperf-doc.php, acessado em: 15-03-2019.

[110] I. P. A. E. Pratama and I. M. A. Wikantyasa, "Implementasi dan Analisis Simulasi QOS dan Perfomance Device dengan Menggunakan ONOS dan Iperf3," *Jurnal Informatika Universitas Pamulang*, vol. 4, no. 2, pp. 57–64, 2019.

[111] S. H. Ali, S. A. Nasir, and S. Qazi, "Impact of Router Buffer Size on TCP/UDP Performance," in *2013 3rd IEEE International Conference on Computer, Control and Communication (IC4)*. IEEE, 2013, pp. 1–6.

[112] R. S. Oliveira, A. S. Carissimi, and S. S. Toscani, *Sistemas Operacionais-Vol. 11: Série Livros Didáticos Informática UFRGS*. Bookman Editora, 2009.

[113] H. Zhang and J. Yan, "Performance of SDN Routing in Comparison with Legacy Routing Protocols," in *2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*. IEEE, 2015, pp. 491–494.

[114] F. Godán, S. Colman, and E. Grampín, "Multicast BGP with SDN control plane," in *2016 7th International Conference on the Network of the Future (NOF)*. IEEE, 2016, pp. 1–5.

[115] A. Basit and N. Ahmed, "Path Diversity for Inter-Domain Routing Security," in *2017 14th international Bhurban conference on applied sciences and technology (IBCAST)*. IEEE, 2017, pp. 384–391.

[116] S. Salsano, P. L. Ventre, L. Prete, G. Siracusano, M. Gerola, and E. Salvadori, "OSHI-Open Source Hybrid IP/SDN networking (and its emulation on Mininet and on distributed SDN testbeds)," in *2014 Third European Workshop on Software Defined Networks*. IEEE, 2014, pp. 13–18.

[117] Z. Hongbo and N. Kitsuwan, "Measuring of failure switch-over time in software-defined network," in *2016 IEEE 17th International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 2016, pp. 118–119.

[118] P. Zeng, K. Nguyen, Y. Shen, and S. Yamada, "On the resilience of software defined routing platform," in *The 16th Asia-Pacific Network Operations and Management Symposium*. IEEE, 2014, pp. 1–4.

[119] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Springer Science & Business Media, 2012.

[120] A. M. Alberti, M. M. Bontempo, J. R. Dos Santos, A. C. Sodré, and R. D. R. Righi, "Novagenesis Aapplied to Information-Centric, Service-Defined, Trustable IoT/WSAN Control Plane and Spectrum Management," *Sensors*, vol. 18, no. 9, p. 3160, 2018.

[121] A. M. Alberti, G. D. Scarpioni, V. J. Magalhaes, A. Cerqueira, J. J. Rodrigues, and R. da Rosa Righi, "Advancing NovaGenesis Architecture Towards Future Internet of Things," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 215–229, 2017.

[122] E. C. do Rosário, V. H. D. D'Ávila, T. B. da Silva, and A. M. Alberti, "A Docker-Based Platform for Future Internet Experimentation: Testing NovaGenesis Name Resolution," in *2019 IEEE Latin-American Conference on Commu-*

nications (LATINCOM). IEEE, 2019, pp. 1–5.

[123] V. H. D. D. e. A. M. A. Elcio C. do Rosário, Tibério Tavares Rezende, "Distribuição de Conteúdos com NovaGenesis em Containers Docker," in *Anais do Workshop de Pesquisa Experimental da Internet do Futuro*. SBC, 2020.

[124] R. Yadav, E. Sousa, and G. Callou, "Performance Comparison Between Virtual Machines and Docker Containers," *IEEE Latin America Transactions*, vol. 16, no. 8, pp. 2282–2288, 2018.

Apêndice A

Imagem de Contêiner *Docker* e *Dockerfiles*

Este Apêndice apresenta como a imagem de contêiner *Docker* foi criada para o AAAIF. Na Seção A.1 será discutido o modo de preparação da imagem. Na Seção A.2 contém as linhas de comandos *Dockerfiles* da imagem nomeada ng-template. Nas Seções A.3, A.4 e A.5 estão os arquivos utilizados no *Dockerfiles*.

O código desse Apêndice foi citado na Seção 4.1.

A.1 Preparação de Imagens *Docker* para o Ambiente de Experimentação

Na criação de imagens para os experimentos foram analisadas quais bibliotecas são essenciais na execução da NG. Configuração de rede, acesso *ssh*, definição de usuário e senha de acesso também fazem parte da análise. A partir dessa análise foi identificado o que seria viável instalar e configurar de maneira a não criar uma imagem com elementos e configurações desnecessárias.

Como visto na Seção 2.5, uma imagem contém várias camadas. O sistema de arquivos da imagem pode usar várias variantes (Subseção 2.8.1), que por padrão utiliza o sistema de arquivo *Overlay2* no Linux Ubuntu 16.04 LTS.

O *Docker* cria imagens automaticamente seguindo as instruções do arquivo *Dockerfile*. O *Dockerfile* é um arquivo onde descrevemos todos os comandos que um usuário pode implementar para montar uma imagem. Cria uma compilação automatizada que executa várias instruções em sequência.

A Figura A.1 ilustra os passos para a criação de uma imagem da NG para o *Docker*. De forma didática dividimos em sete passos: (i) Efetuar o *download* da imagem base do Linux Ubuntu 16.04 LTS no repositório *Docker Hub*; (ii) copiar os arquivos (source.list, supervisord.conf, sshd_config) do diretório padrão que utilizamos no desenvolvimento; (iii) executar o *download* dos pacotes, ferramentas e realizar as configurações; (iv) enviar comando que habilita o uso de tecla de atalhos dentro do contêiner; (v) fazer exposição da porta (22) de acesso *ssh*; (vi) habilitar comando na

inicialização; e (vii) finalizar o processo gravando o contêiner em uma imagem [65].

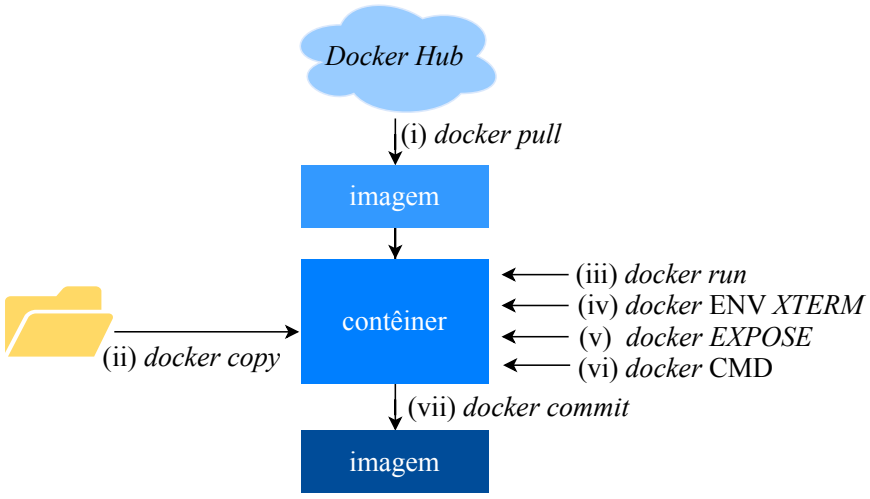


Figura A.1: *Processo de criação da imagem ng-template.*

O comando a seguir cria uma imagem da NG para o ambiente de experimentação desenvolvido nessa dissertação.

docker build -t ng-template .

Onde:

docker build: Permite construir uma imagem do *Docker* a partir de um arquivo *Dockerfile*.
-t: Habilita o uso de *tag* na imagem, que facilita sua identificação entre várias.
ng-template: Nome dado a imagem.
(.): O ponto indica que utilizamos o *Dockerfile* no diretório presente.

A.2 Imagem Base do Ubuntu, Bibliotecas e Configurações Essenciais

```
1  FROM ubuntu:16.04
2
3  MAINTAINER  Elcio Carlos do Rosário
   ↪  <elcio.carlos@mtel.inatel.br>
4
5  ENV DEBIAN_FRONTEND noninteractive
6
7  COPY sources.list /etc/apt/sources.list
8
9  #arquivo com script de ativação do acesso via ssh
10 COPY supervisord.conf
   ↪  /etc/supervisor/conf.d/supervisord.conf
11
12 #arquivo de configuração do acesso via ssh
```

```
13 COPY sshd_config /etc/ssh/sshd_config
14
15 RUN mkdir -p /var/log/supervisor
16
17 #cria usuário ng com senha gandalf e alterar senha do
   ↪ root para gandalf
18 RUN useradd -m ng && (echo gandalf ; echo gandalf) |
   ↪ passwd ng
19 RUN (echo gandalf ; echo gandalf) | passwd root
20
21 #atualização do sistema operacional, instalação do
   ↪ comando aptitude
22 #instalacao da ferramenta htop
23 RUN apt-get update -y
24 RUN apt-get install aptitude -y
25 RUN aptitude upgrade -y
26 RUN aptitude install htop -y
27
28 #instalação do editor nano, bibliotecas para o python
29 #instalação da ferramenta de acesso ssh
30 RUN aptitude install nano -y
31 RUN aptitude install build-essential -y
32 RUN aptitude install ssh -y
33 RUN aptitude install libssh-dev -y
34 RUN aptitude install python -y
35 RUN aptitude install python-pip -y
36 RUN pip install --upgrade pip
37
38 #bibliotecas compatíveis com os scprits NG
39 RUN aptitude install python-dev -y
40 RUN aptitude install psmisc -y
41 RUN aptitude install libffi-dev -y
42
43 #instalação da biblioteca paramiko e supervisor
44 RUN aptitude install python-paramiko -y
45 RUN aptitude install supervisor -y
46
47
48 #bibliotecas essenciais para os testes com a NovaGenesis
49 RUN aptitude install libssl-dev -y
50 RUN aptitude install libzmq-dev -y
51 RUN aptitude install libzmq5 -y
52 RUN aptitude install gdb -y
53 RUN aptitude install g++ -y
54 RUN aptitude install gcc -y
55 RUN aptitude install nmap -y
```

```
56
57 #instalação da ferramenta ping e ifconfig
58 RUN apt-get install inetutils-ping -y
59 RUN apt-get install iputils-ping -y
60 RUN apt-get install net-tools -y
61 RUN apt-get install sudo -y
62
63 #ferramentas, biblioteca de uso do python
64 #criptografia, criação de imagens, acesso remoto via ssh
65 RUN pip install -U pip setuptools
66 RUN pip install --upgrade paramiko
67 RUN pip install cryptography --force-reinstall
68 RUN pip install numpy
69 RUN pip install image
70
71 #limpa e remove arquivos deixando a imagem mais leve
72 RUN apt-get clean autoclean && apt-get autoremove -y
73 RUN rm -rf /var/lib/{apt,dpkg,cache,log}/
74
75 #ativar o uso de teclas de atalho
76 ENV TERM xterm
77
78 #exposição da porta 22 para acesso remoto
79 EXPOSE 22
80
81 #ao executar o container o comando supervisord ativa a
   ↪ comunicação ssh
82 CMD ["/usr/bin/supervisord"]
```

A.3 Código do Arquivo source.list

```
1 deb http://br.archive.ubuntu.com/ubuntu/ trusty main
   ↪ restricted
2 deb-src http://br.archive.ubuntu.com/ubuntu/ trusty main
   ↪ restricted
3 deb http://br.archive.ubuntu.com/ubuntu/ trusty-updates
   ↪ main restricted
4 deb-src http://br.archive.ubuntu.com/ubuntu/
   ↪ trusty-updates main restricted
5 deb http://br.archive.ubuntu.com/ubuntu/ trusty universe
6 deb-src http://br.archive.ubuntu.com/ubuntu/ trusty
   ↪ universe
7 deb http://br.archive.ubuntu.com/ubuntu/ trusty-updates
   ↪ universe
8 deb-src http://br.archive.ubuntu.com/ubuntu/
   ↪ trusty-updates universe
```

```
deb http://br.archive.ubuntu.com/ubuntu/ trusty
 ↪ multiverse
deb-src http://br.archive.ubuntu.com/ubuntu/ trusty
 ↪ multiverse
deb http://br.archive.ubuntu.com/ubuntu/ trusty-updates
 ↪ multiverse
deb-src http://br.archive.ubuntu.com/ubuntu/
 ↪ trusty-updates multiverse
deb http://br.archive.ubuntu.com/ubuntu/ trusty-backports
 ↪ main restricted universe multiverse
deb-src http://br.archive.ubuntu.com/ubuntu/
 ↪ trusty-backports main restricted universe multiverse
deb http://security.ubuntu.com/ubuntu trusty-security
 ↪ main restricted
deb-src http://security.ubuntu.com/ubuntu trusty-security
 ↪ main restricted
deb http://security.ubuntu.com/ubuntu trusty-security
 ↪ universe
deb-src http://security.ubuntu.com/ubuntu trusty-security
 ↪ universe
deb http://security.ubuntu.com/ubuntu trusty-security
 ↪ multiverse
deb-src http://security.ubuntu.com/ubuntu trusty-security
 ↪ multiverse
```

A.4 Código do Arquivo supervisord.conf

```
[supervisord]
nodaemon=true

[program:sshd]
command=/usr/sbin/service ssh start
```

A.5 Código do Arquivo sshd_config

```
# Package generated configuration file
# See the sshd_config(5) manpage for details

# What ports, IPs and protocols we listen for
Port 22
# Use these options to restrict which
 ↪ interfaces/protocols sshd will bind to
#ListenAddress ::
#ListenAddress 0.0.0.0
Protocol 2
```

```
10 # HostKeys for protocol version 2
11 HostKey /etc/ssh/ssh_host_rsa_key
12 HostKey /etc/ssh/ssh_host_dsa_key
13 HostKey /etc/ssh/ssh_host_ecdsa_key
14 HostKey /etc/ssh/ssh_host_ed25519_key
15 #Privilege Separation is turned on for security
16 UsePrivilegeSeparation yes
17
18 # Lifetime and size of ephemeral version 1 server key
19 KeyRegenerationInterval 3600
20 ServerKeyBits 1024
21
22 # Logging
23 SyslogFacility AUTH
24 LogLevel INFO
25
26 # Authentication:
27 LoginGraceTime 120
28 PermitRootLogin yes
29 StrictModes yes
30
31 RSAAuthentication yes
32 PubkeyAuthentication yes
33 #AuthorizedKeysFile        %h/.ssh/authorized_keys
34
35 # Don't read the user's ~/.rhosts and ~/.shosts files
36 IgnoreRhosts yes
37 # For this to work you will also need host keys in
    ↪ /etc/ssh_known_hosts
38 RhostsRSAAuthentication no
39 # similar for protocol version 2
40 HostbasedAuthentication no
41 # Uncomment if you don't trust ~/.ssh/known_hosts for
    ↪ RhostsRSAAuthentication
42 #IgnoreUserKnownHosts yes
43
44 # To enable empty passwords, change to yes (NOT
    ↪ RECOMMENDED)
45 PermitEmptyPasswords no
46
47 # Change to yes to enable challenge-response passwords
    ↪ (beware issues with
48 # some PAM modules and threads)
49 ChallengeResponseAuthentication no
50
51 # Change to no to disable tunnelled clear text passwords
```

```
52 #PasswordAuthentication yes
53
54 # Kerberos options
55 #KerberosAuthentication no
56 #KerberosGetAFSToken no
57 #KerberosOrLocalPasswd yes
58 #KerberosTicketCleanup yes
59
60 # GSSAPI options
61 #GSSAPIAuthentication no
62 #GSSAPICleanupCredentials yes
63
64 X11Forwarding yes
65 X11DisplayOffset 10
66 PrintMotd no
67 PrintLastLog yes
68 TCPKeepAlive yes
69 #UseLogin no
70
71 #MaxStartups 10:30:60
72 #Banner /etc/issue.net
73
74 # Allow client to pass locale environment variables
75 AcceptEnv LANG LC_*
76
77 Subsystem sftp /usr/lib/openssh/sftp-server
78
79 # Set this to 'yes' to enable PAM authentication, account
    ↪ processing,
80 # and session processing. If this is enabled, PAM
    ↪ authentication will
81 # be allowed through the ChallengeResponseAuthentication
    ↪ and
82 # PasswordAuthentication.  Depending on your PAM
    ↪ configuration,
83 # PAM authentication via ChallengeResponseAuthentication
    ↪ may bypass
84 # the setting of "PermitRootLogin without-password".
85 # If you just want the PAM account and session checks to
    ↪ run without
86 # PAM authentication, then enable this but set
    ↪ PasswordAuthentication
87 # and ChallengeResponseAuthentication to 'no'.
88 UsePAM yes
```

Apêndice B

Preparando o Sistema Operacional Linux Ubuntu 16.04 na máquina física

Este Apêndice contém o *script* de instalação das bibliotecas essenciais tanto para execução da arquitetura NovaGenesis, quanto para execução dos *scripts* Python de automatização dos experimentos.

O Apêndice foi citado na Seção 4.3.

B.1 Código do script libray.sh

```
#!/bin/bash
#@Elcio Rosário

#Bibliotecas apt-get install
lista1=( "python" "libzmq5" "python-pip" " python-dev"
 ↪  "psmisc" "libssl-dev" "libzmq-dev" "libffi-dev"
                "gcc" "g++" "nmap" "xterm" )

#Bibliotecas pip install
lista2=( "-U pip setuptools" "--upgrade pip " "paramiko"
 ↪  "--upgrade paramiko" "cryptography --force-reinstall"
 ↪  "numpy" "image" )

#
#
#

if [ "$1" == "" ]; then
        echo "Entre com o argumeto"
        echo " -all [Para Atualizar repositorios, pacotes
         ↪  e libs para NG]"
        echo " -ng [para instalar somente libs para NG]"
        exit 1
```

```
20 fi
21
22 if [ "$1" == "-all" ]; then
23         echo "Atualizando repositórios.."
24         if ! apt-get update
25         then
26             echo -e "\033[05;31m \nNão foi possível
                ↪  atualizar os repositórios. Verifique seu
                ↪  arquivo /etc/apt/sources.list\n"; tput
                ↪  sgr0
27
28     else
29             echo -e "\033[05;35m \nAtualização feita com
                ↪  sucesso\n"; tput sgr0
30         fi
31
32         echo "Atualizando pacotes já instalados"
33         if ! apt-get dist-upgrade -y
34         then
35             echo -e "\033[05;31m \nNão foi possível
                ↪  atualizar pacotes.\n"; tput sgr0
36             #exit 1
37         else
38                 echo -e "\033[05;35m \nAtualização de
                    ↪  pacotes feita com sucesso\n"; tput
                    ↪  sgr0
39         fi
40
41 fi
42
43 if [ "$1" == "-all" ] || [ "$1" == "-ng" ]; then
44         for i in ${lista1[@]}
45         do
46                 if ! apt-get install $i -y
47                 then
48                     echo -e "\03[05;31m********** Não foi
                        ↪  possível instalar o pacote $i
                        ↪  ***************"; tput sgr0
49
50             else
51                     echo -e "\033[05;35m Instalação $i
                        ↪  finalizada\n"; tput sgr0
52                 fi
53         done
54
55         for j in "${lista2[@]}"
```

```
56          do
57                  if ! pip install $j
58                  then
59                      echo -e "\033[05;31m********** Não
                        ↪  foi possível instalar o pacote $j
                        ↪  ****************\n"; tput sgr0
60
61              else
62                      echo -e "\033[05;35m Instalação $j
                        ↪  finalizada\n"; tput sgr0
63                  fi
64
65          done
66
67          else
68                  echo "Comando Invalido, use -all ou -ng"
69
70  fi
```

Apêndice C

Menu de Gerência de Contêineres

Este Apêndice contém as linhas de código Python hyperDocker.py. Ele foi criado com o objetivo de automatizar as tarefas antes manuais no uso de contêineres *Docker*. Na Seção C.1 estão as linhas do código. A Seção C.2 mostra o fluxograma completo do gerenciador dos contêineres desenvolvido, composto por dez opções.

O código desse Apêndice foi citado na Seção 4.1.

C.1 Código do script hyperDocker.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#@Elcio Rosário
from serviceNG import *
from nmap import *
import random


######### Definir parâmetros Iniciais ################
def case_1():

        #recebe os atributos da rede
        net.pull_network(manager._network)

        op = raw_input("Utilizar todos os host listados?
         ↪  (y/n): ")
        if op == 'y':
                #Define parametro do arquivo
                 ↪  parametrosPGCS e printa a interface
                 ↪  de rede
                manager.define_pgcs(net._mac)

        if op == 'n' :
```

```
21                  x = raw_input("Entre com o ip inicial e
                     ↪  final ou com o ip unico: ")
22                  x = x.split()
23
24                  #Define range dos ips listados
25                  if len(x) == 1:
26                          net.range(x[0],x[0])
27                  else:
28                          net.range(x[0],x[1])
29                  #Define parametro do arquivo
                     ↪  parametrosPGCS e printa a interface
                     ↪  de rede
30                  manager.define_pgcs(net._mac)
31
32
33          #Escreve o endereco mac do server no arquivo
             ↪  /PGCS/macServer
34          manager.write_mac()
35          '''if ips not in getParam.ipServidor:
36                  print "Erro - IPs do Servidor Fotos não
 ↪  estam contidos nos IPs encontrados"
37                  exit()'''
38
39  ######### Enviar NG para cliente ######################
40  def case_2():
41
42          manager.threads('sendNovagenesis',net._ip,
             ↪  waiting = 'yes')
43
44  ######### Apagar pasta        NG no cliente
     ↪  ###################
45  def case_3():
46
47          manager.threads('removeNovagenesis',net._ip,
             ↪  waiting = 'yes')
48
49  ######### Executar NG Core  ####################
50  def case_4():
51
52          cmd1 = "ipcmk -M 1024 ;
             ↪  ./novagenesis_central/clean.sh"
53          cmd2 = "cd novagenesis_central/; sh runCore.sh"
54          #cmd3 = "cd novagenesis_files; sh watch_info.sh >
             ↪  /dev/null 2>&1&"
55          a = subprocess.check_output(cmd1,shell=True)
56
```

```
57          #manager.exe_shell(cmd3)
58          manager.exe_shell(cmd1)
59          manager.exe_shell(cmd2)
60
61
62
63  ######### Executar PGCS no cliente ####################
64  def case_5():
65
66          #c = "cd " + manager._pathRemote + "; ipcmk -M
             ↪  1024 ;./clean.sh; ./watch_info.sh"
67          c = "cd " + manager._pathRemote + "; ipcmk -M
             ↪  1024 ;./clean.sh"
68
69
70          manager.threads('exe_cmd_root', net._ip, cmd = c,
             ↪  delay = 0.2)
71
72          #c = "cd " + manager._pathRemote + ";
             ↪  ./watch_info.sh > /dev/null 2>&1&;"
73          #manager.threads('exe_cmd_root', net._ip, cmd =
             ↪  c, delay = 0.2)
74
75          print "Executando PGCS"
76          cmd = ['xterm']
77          p = "sshpass -p 'gandalf' ssh -o
             ↪  StrictHostKeychecking=no root@" +
             ↪  str(manager._ipServer[0]) + " 'cd
78          /home/ng/workspace/novagenesis_files/PGCS_files/;
 ↪  ./runPGCS.sh';"
79          cmd.extend(['-e', p])
80          subprocess.Popen(cmd, stdout=subprocess.PIPE)
81
82  ######## Executar PGCS e HTS ###################
83  def case_6():
84
85
86          #c = "cd " + manager._pathRemote + "; ipcmk -M
             ↪  1024 ;./clean.sh; ./watch_info.sh"
87          c = "cd " + manager._pathRemote + "; ipcmk -M
             ↪  1024 ;./clean.sh"
88
89          manager.threads('exe_cmd_root', net._ip, cmd = c,
             ↪  delay = 0.2)
90
91          for i in range(1,len(net._ip)):
```

```
92
93                  print "Executando PGCS e
                     ↪  HTS"
94                  cmd = ['xterm']
95                  p = "sshpass -p 'gandalf' ssh -o
                     ↪  StrictHostKeychecking=no root@" +
                     ↪  str(net._ip[i]) + "
96                  'cd
    ↪  /home/ng/workspace/novagenesis_files/HTS_files/;
    ↪  ./runHTS.sh ; cd
    ↪  /home/ng/workspace/novagenesis_files/PGCS_files/;
    ↪  ./runPGCS.sh';"
97                  cmd.extend(['-e', p])
98                  subprocess.Popen(cmd,
                     ↪  stdout=subprocess.PIPE)
99
100
101 ######### Executar NB ####################
102
103 def case_7():
104
105         print "Executando NB"
106         cmd = ['xterm']
107         p = "sshpass -p 'gandalf' ssh -o
             ↪  StrictHostKeychecking=no root@" +
             ↪  str(manager._ipServer[0]) + " 'cd
             ↪  /home/ng/workspace/novagenesis_files/;
             ↪  ./runClient.sh';"
108         cmd.extend(['-e', p])
109         subprocess.Popen(cmd, stdout=subprocess.PIPE)
110
111
112
113 ######## Download Relatorios ######################
114 def case_8():
115
116         c1 = "cd " + manager._pathRemote + "; mv memo.txt
             ↪  net.txt cpu.txt ./NBSimpleTest/"
117         manager.threads('exe_cmd_root', net._ip, cmd =
             ↪  c1, delay = 0.2)
118
119         ips = net._ip
120         name_dir = raw_input("Salvar relatorio com nome:
             ↪  ")
121         c = "mkdir /home/ng/ngDocker/relatorios_pub_sub/"
122         +name_dir manager.exe_shell(c)
```

```
123
124         manager.exe_shell("sshpass -p 'gandalf' scp
             ↪  root@"+str(manager._ipServer[0])+":/home/ng/
125         workspace/novagenesis_files/NBSimpleTest/*"+ "
 ↪  /home/ng/ngDocker/relatorios_pub_sub/"+name_dir)
126
127         #manager.exe_shell("cd
             ↪  /home/ng/ngDocker/novagenesis_files; mv
             ↪  memo.txt net.txt cpu.txt
             ↪  /home/ng/ngDocker/relatorios_pub_sub/;")
128         #manager.exe_shell("cd
             ↪  /home/ng/ngDocker/relatorios_pub_sub; mv
             ↪  memo.txt memo-server.txt; mv cpu.txt
             ↪  cpu-server.txt;mv net.txt net-server.txt; mv
             ↪  memo-server.txt cpu-server.txt net-server.txt
             ↪  ./"+name_dir)
129
130
131 ###### SubMenu  Serviços
 ↪  ###################################
132 def case_9():
133         subDict = {1: subcase_1, 2: subcase_2, 3:
             ↪  subcase_3}
134         subDict[input('1 - Excluir todos PGCSes\n' + '2 -
             ↪  Excluir todos HTSes\n'+'3 - Excluir
             ↪  NBSimpleTest\n'
135                 + 'Escolhar uma opção: ')]()
136
137 def subcase_1():
138
139         c =  "sudo -S killall PGCS;sudo -S killall
             ↪  watch_info;"
140
141         manager.threads('exe_cmd_root',net._ip,cmd = c,
             ↪  waiting = 'yes')
142
143 def subcase_2():
144
145         c = "sudo -S killall HTS;"
146         manager.threads ('exec_cmd_root', net._ip, cmd =
             ↪  c, waiting = 'yes')
147
148 def subcase_3():
149
150         c = "sudo -S pkill NBSimpleTestApp;"
151
```

```
152         manager.threads ('exe_cmd_root', net._ip, cmd =
             ↪  c, waiting = 'yes')
153
154 ######## Finalizar Experimentos
     ↪  ##########################
155
156 def case_10():
157
158         print "--------------- GOOD BYE
             ↪  ------------------------"
159         exit()
160
161
162 def switch(x):
163         dict[x]()
164
165 ########## -   MENU    -
     ↪  ################################
166
167 if __name__ == '__main__':
168
169         dict = {1 : case_1, 2 : case_2, 3 : case_3, 4 :
             ↪  case_4,5: case_5, 6 : case_6, 7 : case_7, 8 :
             ↪  case_8,9: case_9,10 : case_10}
170
171         #Inicializa class Network.nmap
172         net = Network()
173         #Inicializa class Config.serviceNG
174         manager = Config()
175         #Lê configuracoes iniciais
176         manager.read()
177         #printa configuracoes iniciais
178         print (manager)
179
180         while True:
181
182                 time.sleep(0.5)
183                 print "\n************ NOVAGENESIS
                     ↪  ***************************\n"
184                 switch(input('1 - Inicializar
                     ↪  Parâmetros\n'+'2 - Enviar arquivos
                     ↪  NG\n'+ '3 - Apagar Diretório\n' +
185                 '4 - Executar Core NG\n' + '5 - Executar
                     ↪  PGCS no cliente\n'+ '6 - Executar
                     ↪  PGCSes e HTSes\n' +
```

```
186                 '7 - Executar NBSimpleTest\n' + '8 -
                    ↪  Salvar Relatório\n' +'9 - Serviços\n'
                    ↪  +'10 - Finalizar Experimentos\n' +
187                 'Entre com a Opção: '))
188
189
190
```

C.2 Fluxograma Gerenciador de Contêineres

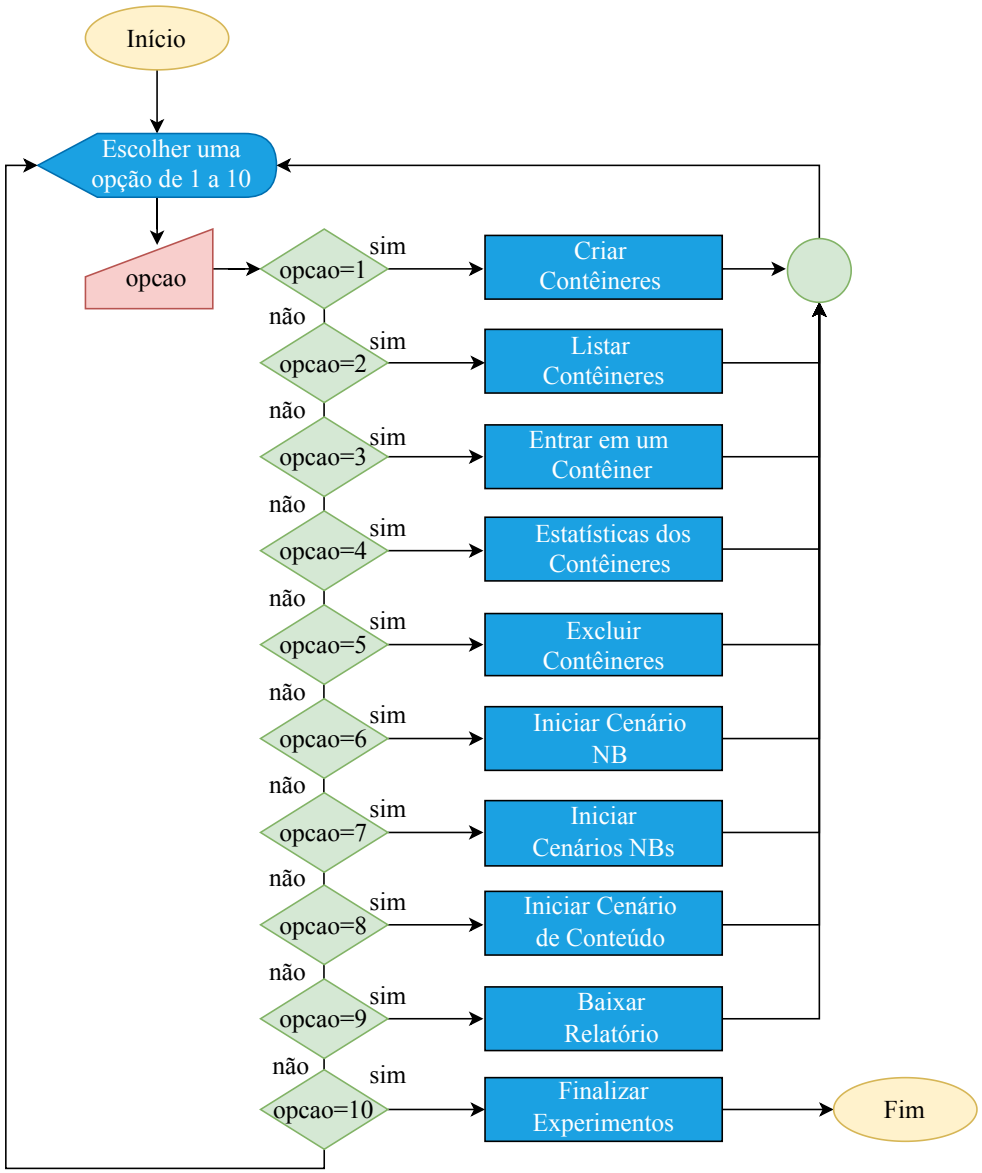


Figura C.1: *Fluxograma completo do algoritmo do gerenciador de contêineres desenvolvido para o AAAIF.*

C.3 Código do script serviceNG.py

```
# -*- coding: utf-8 -*-
#!/usr/bin/env python
#@Elcio Rosário
import os,threading,subprocess,time
from sshClient import sshClient

#####################################################

class Config(object,):

        def __init__ (self,pathRemote = None, username =
        ↪ None, password = None, delay_timer = None,
        ↪ network = None, ipServer = []):
                self._pathlocal =  os.getcwd() +
                ↪ "/novagenesis_files"
                self._pathRemote = pathRemote #Local da
                ↪ pasta aonde sera enviado os arquivos
                self._username = username #nome do
                ↪ usuario remoto
                self._password = password #senha de
                ↪ usuario remoto
                self._delay_timer = delay_timer #Delay
                ↪ para executar App Fotos em cada um
                ↪ dos Clientes
                self._network = network
                self._ipServer = ipServer


        def read(self):
                try:
                inputFile = open('config.ng', 'rt')
                args = inputFile.read()
                args = args.split()

                if('pathRemote' in args):
                        self._pathRemote =
                        ↪ args[args.index('pathRemote')+
                        ↪ 2]

                if('username' in args):
                        self._username =
                        ↪ args[args.index('username')+
                        ↪ 2]

```

```
33                  if('password' in args):
34                          self._password =
                             ↪  args[args.index('password')+
                             ↪  2]
35
36                  if('delay_timer' in args):
37                          self._delay_timer =
                             ↪  args[args.index('delay_timer')+
                             ↪  2]
38
39                  if('network' in args):
40                          self._network =
                             ↪  args[args.index('network')+
                             ↪  2]
41
42                  if('ipServer' in args):
43                          args[:args.index('ipServer') + 2]
                             ↪  = []
44                          self._ipServer =
                             ↪  args
45
46                  inputFile.close()
47                  except:
48                          print "Erro, arquivo 'config.ng'
                             ↪  nao encontrado"
49                          exit()
50
51          def
             ↪  sendNovagenesis(self,ip,username,password,pathlocal,
52          pathRemote):
53
54                  ssh =
                     ↪  sshClient('root',ip,username,password)
55                  ssh.connect()
56                  ssh.push(pathlocal,pathRemote)
57                  ssh.write_root('chmod -R 777 ' +
                     ↪  pathRemote)
58                  ssh.close()
59
60          def removeNovagenesis(self,ip,username,password,
61          pathRemote):
62                  ssh =
                     ↪  sshClient('root',ip,username,password)
63                  ssh.connect()
64                  ssh.write_root("rm -Rf "+ pathRemote)
65                  ssh.close()
```

```
66
67          def exe_cmd(self,ip,username,password,cmd):
68                  ssh =
                     ↪  sshClient('root',ip,username,password)
69                  ssh.connect()
70                  ssh.write_root(cmd)
71                  ssh.close()
72
73          def exe_cmd_root(self,ip,username,password,cmd):
74                  ssh =
                     ↪  sshClient('root',ip,username,password)
75                  ssh.connect()
76                  ssh.write_root(cmd)
77                  ssh.close()
78
79          def exe_shell(self,cmd):
80                  subprocess.call(cmd, shell=True)
81
82          def report(self,ip,username,password,pathRemote,
             ↪  name_dir):
83                  report_files =
                     ↪  ['imgs/App_Core_pubrtt{}.dat',
84                  'imgs/App_Core_subrtt{}.dat']
85                  report_server =
                     ↪  ['imgs_received/App_Core_subrtt{}.dat']
86
87                  ssh =
                     ↪  sshClient('root',ip,username,password)
88                  ssh.connect()
89
90
91                  mkd = "/"+ os.getcwd()
                     ↪  +'/relatorios_pub_sub/' + name_dir
92
93                  try:
94                          os.mkdir(mkd)
95                  except OSError:
96                          pass
97
98
99  if ip not in self._ipServer:
100         for name in report_files:
101                 #adiciona no nome do arquivo o numero do
                     ↪  pc retirando a ultima parte do IP
102                 report_name =
                     ↪  name.format('_pc'+str(ip.split('.')
```

```
103                     [-1])+'_')
104                     #retira apenas o nome do arquivo, para
                         ↪  salvar na pasta relatorios_pub_sub
105                     report_name = report_name.split('/')[-1]
106                     ssh.pull( pathRemote + name.format(''),
                         ↪  mkd[1:] +'/'+ report_name )
107
108 else:
109         for name in report_server:
110                     #adiciona no nome do arquivo o numero do
                         ↪  pc retirando a ultima parte do IP
111                     report_name =
                         ↪  name.format('_pc'+str(ip.split('.')
112                     [-1])+'_')
113                     #retira apenas o nome do arquivo, para
                         ↪  salvar na pasta relatorios_pub_sub
114                     report_name = report_ame.split('/')[-1]
115                     ssh.pull( pathRemote + name.format(''),
                         ↪  mkd[1:] +'/'+ report_name )
116
117
118                     self.exe_shell("chmod -R 777 ./")
119                     #Limpa pasta HTS
120                     self.exe_shell("cd " + os.getcwd() +
                         ↪  "/novagenesis_central/HTS_files;
121                     rm -f *.jpg")
122                     #Fecha conexao
123                     ssh.close()
124
125
126         def define_pgcs(self,listMacs):
127                     aux = " 0 Intra_Domain -p "
128                     file =
                         ↪  open("novagenesis_central/parametrosPGCS",
129                     "w") interface = self.pull_interface()
130                     print "interface da rede: " +
                         ↪  str(interface)
131
132                     for mac in listMacs:
133                             aux += "Ethernet Intra_Domain "+
                                 ↪  str(interface) +" "+ str(mac)
                                 ↪  +" 1200 "
134
135                     file.write(aux)
136                     file.close()
137
```

```
138         def write_mac(self):
139
140                 aux = os.popen("ifconfig")
141                 args = aux.read()
142                 aux.close()
143                 args = args.split()
144                 mac = args[args.index("HW")+1]
145
146                 file =
                     ↪  open("novagenesis_files/PGCS_files/
147                 macServer", "w")
148                 file.write(mac)
149                 file.close()
150
151
152         def pull_interface(self):
153
154                 aux = os.popen("ifconfig")
155                 args = aux.read()
156                 aux.close()
157                 args = args.split()
158
159                 return args[0]
160
161         def __str__ (self):
162
163                 return ("************Configurações
                     ↪  Iniciais**************************"
                     ↪  +
164                         "\nPathRemote: " + str
                             ↪  (self._pathRemote) +
165                         "\nUsername: " +
                             ↪  str(self._username) +
166                         "\nPassword: " +
                             ↪  str(self._password) +
167                         "\nDelay Timer: " +
                             ↪  str(self._delay_timer) +
168                         "\nNetwork: " +
                             ↪  str(self._network) +
169                         "\nIpServidor: " +
                             ↪  str(self._ipServer) +
170                         "\n*****************************")
171
172         def threads(self,name, ips, cmd = None, waiting =
             ↪  None, delay = 0.0):
173                 monitores = []
```

```
174                 for ip in ips:
175                         monitores.append(self.Monitor(name,
                             ↪  ip, self._username,
                             ↪  self._password,
                             ↪  self._pathlocal
                             ↪  ,self._pathRemote, cmd))
176
177                 # Execute as Threads
178                 for monitor in monitores:
179                         monitor.start()
180                         time.sleep(float(delay))
181
182                 if waiting:
183                         for t in monitores:
184                                 t.join()
185
186         class Monitor(threading.Thread):
187
188                 def
                     ↪  __init__(self,name,ip,username,password,
189                 pathlocal,pathRemote, cmd = ''):
190                         threading.Thread.__init__(self)
191                         self.name = name
192                         self.ip = ip
193                         self.username = username
194                         self.password = password
195                         self.pathlocal = pathlocal
196                         self.pathRemote = pathRemote
197                         self.cmd = cmd
198
199         def run(self):
200
201                 print "Starting " + str(self.name) + " -
                     ↪  " + str(self.ip)
202
203                 if self.name in 'sendNovagenesis':
204                         Config().sendNovagenesis(self.ip,
205                         self.username,self.password,
                             ↪  self.pathlocal,
                             ↪  self.pathRemote)
206
207                 if self.name in 'removeNovagenesis':
208                         Config().removeNovagenesis(self.ip,
                             ↪  self.username, self.password,
                             ↪  self.pathRemote)
209
```

```
210                 if self.name in 'exe_cmd_root':
211                         Config().exe_cmd_root(self.ip,
                             ↪  self.username, self.password,
                             ↪  self.cmd)
212
213                 if self.name in 'exe_cmd':
214                         Config().exe_cmd(self.ip,
                             ↪  self.username, self.password,
                             ↪  self.cmd)
215
216                 if self.name in 'report':
217                         Config().report(self.ip,
                             ↪  self.username, self.password,
                             ↪  self.pathRemote, self.cmd)
218
219                         print "Exiting " + str(self.ip)
220
```

C.4 Código do script nmap.py

```
1  # -*- coding: utf-8 -*-
2  #@Elcio Rosário
3  import os,threading,subprocess
4
5  class Network(object):
6
7          def __init__(self):
8                  self._ip = []
9                  self._mac = []
10                 self._myip = None
11
12
13         def pull_network(self,rede):
14                 print 'Buscando na rede...'
15                 aux = subprocess.check_output("hostname
                    ↪  -I", shell=True)
16
17                 if len(aux) > 1:
18                         self._myip = aux.split()[0]
19                         print "IP local: "+ self._myip
20
21                 else:
22                         print "Erro ao buscar IP,
                            ↪  verifique sua conexao de
                            ↪  rede"
23                         exit()
```

```
24
25                  p = subprocess.check_output("nmap -sP -n
                     ↪  "+ rede +" | egrep -i '(scan|MAC)'",
                     ↪  shell=True)
26                  p = p.split()
27                  bridge = 1
28                  while(1):
29                          x = p.index("for")
30                          p[:x+1] = []
31
32                          if self._myip == p[0]:
33                                  return
34
35                          if bridge == 0:
36
37                                  print p[0]
38                                  self._ip.append(p[0])
39                                  x = p.index("Address:")
40                                  p[:x+1] = []
41                                  self._mac.append(p[0].lower())
42
43                          bridge = 0
44
45          def __str__ (self):
46                  return ("IP local: " + str(self._myip)+
47                                  "\nIPs Disponiveis: " +
                                     ↪  str(self._ip))
48
49          def range(self,st,end):
50                  try:
51                          x = self._ip.index(st)
52                          y = self._ip.index(end)+1
53                          self._ip = self._ip[x:y]
54                          self._mac = self._mac[x:y]
55                  except:
56                          print "faixa de ip nao existe"
57                          exit()
58
59
```

C.5 Código do script sshClient.py

```
1  #@Elcio Rosário
2  import paramiko,os
3
4  class sshClient:
```

```
5
6  def __init__(self,userRemote,ip,username,password):
7  #construtor cliente ssh
8          self.ssh = paramiko.SSHClient()#cria uma instacia
           ↪  do cliente ssh
9          self.ssh.set_missing_host_key_policy
10         (paramiko.AutoAddPolicy())#parametro para aceitar
           ↪  automaticamente as chaves
11         self.username = username
12         self.password = password
13         self.ip = ip
14         self.userRemote = userRemote
15         self.connected = False
16
17         def connect(self):
18                 try:
19                         self.ssh.connect(self.ip,
                           ↪  username =
                           ↪  self.username,password
                           ↪  =self.password ,timeout = 5)
20                         self.connected = True
21                         self.sftp =
                           ↪  self.ssh.open_sftp()#protoco
                           ↪  sftp para transferencia
                           ↪  usando biblioteca paramiko
22
23                 except:
24                         print self.ip + ' Time out '
25                         self.connected = False
26
27
28         def write(self,cmd):
29
30                 if not self.connected:
31                         print "ERRO AO SE CONECTAR"
32                         return
33                 else:
34                         stdin,stdout,stderr =
                           ↪  self.ssh.exec_command(cmd)
35                         if not stdout.read() == "":
36                                 print stdout.read()
37
38         def write_root(self,cmd):
39
40                 if not self.connected:
41                         print "ERRO AO SE CONECTAR"
```

```
42                          return
43                  else:
44                          stdin,stdout,stderr =
                             ↪  self.ssh.exec_command(cmd,
45                          get_pty = True)
46                          stdin.write(self.password+"\n")
47                          if not stdout.read() == "":
48                                  print stdout.read()
49
50          def read():
51                  pass
52
53          #envia todos os arquivos do diretorio selecionado
54          def push(self,localfile,remotefile):
55                  try:
56                  for root, dirs, files in
                     ↪  os.walk(localfile, topdown=False):
                     ↪  #gera os nome dos arquivos no
                     ↪  diretório localfile
57                  for name in dirs:
58                          #print 'CRIANDO DIRETORIO: ' +
                             ↪  remotefile + name
59                          self.ssh.exec_command('mkdir -p
                             ↪  '+ remotefile+ name +' ;cd ;
                             ↪  chmod -R 777 '+ remotefile +
                             ↪  name)
60
61                  print 'TRANSFERINDO
                     ↪  ARQUIVOS......'
62                  for root, dirs, files in
                     ↪  os.walk("./novagenesis_files",
                     ↪  topdown=False):
63                          for name in files:
64                                  fname =
                                     ↪  os.path.join(root,
                                     ↪  name)[2:]
65
66                  self.sftp.put(localfile[:-17]+fname,
67                  remotefile[:-18]+fname)
68
69                  '''for file in os.listdir(localfile):
70                          path  =
↪   os.path.join(localfile,file)
71                          if not os.path.isdir(path):
72                          self.sftp.put(localfile
↪   +file,remotefile+file)
```

```
                                '''
                        except:
                                print "ERRO - CAMINHO
                                 ↪  INVALIDO"


        def pull(self,remote,local):
                self.sftp.get(remote, local)

        def close(self):
                self.ssh.close()
```

C.6 Código do script runCore.sh

```
var1=`pwd`;
PGCS=`cat parametrosPGCS`

xterm -title "PGCS" -e "cd $var1/PGCS_files; ./PGCS .
 ↪  $PGCS;" &
xterm -title "GIRS" -e "cd $var1/GIRS_files/; sleep
 ↪  10;./GIRS ./;" &
xterm -title "PSS" -e "cd $var1/PSS_files/; sleep
 ↪  10;./PSS ./;" &
xterm -title "HTS" -e "cd $var1/HTS_files/; sleep
 ↪  10;./HTS ./;" &

#cd $var1/PGCS_files/; ./PGCS . $PGCS > $var1/pgcs.log &
#cd $var1/GIRS_files/; sleep 10;./GIRS ./ >
 ↪  $var1/girs.log &
#cd $var1/PSS_files/; sleep 10;./PSS ./ > $var1/pss.log &
#cd $var1/HTS_files/; sleep 10;./HTS ./ > $var1/hts.log &
```

C.7 Código do script clean.sh

```
#!/bin/bash
#@Elcio Rosario

ME=`whoami`

IPCS_S=`ipcs -s | egrep "0x[0-9a-f]+ [0-9]+" | grep $ME |
 ↪  cut -f2 -d" "`
IPCS_M=`ipcs -m | egrep "0x[0-9a-f]+ [0-9]+" | grep $ME |
 ↪  cut -f2 -d" "`
IPCS_Q=`ipcs -q | egrep "0x[0-9a-f]+ [0-9]+" | grep $ME |
 ↪  cut -f2 -d" "`
```

```
9
10
11 for id in $IPCS_M; do
12   ipcrm -m $id;
13 done
14
15 for id in $IPCS_S; do
16   ipcrm -s $id;
17 done
18
19 for id in $IPCS_Q; do
20   ipcrm -q $id;
21 done
22
23 rm -rfv /dev/shm/sem.*
```

C.8 Arquivo parametrosPGCS

```
1  0 Intra_Domain -p Ethernet Intra_Domain eth0
    ↪ 02:42:ac:11:00:03 1200 Ethernet Intra_Domain eth0
    ↪ 02:42:ac:11:00:04 1200
```

Apêndice D

Menu de Gerência de Experimentos de Name Bindings

Este Apêndice contém as linhas de código do *script* dos experimentos de *Name Bindings*. Foi criado com o objetivo de automatizar as tarefas antes manuais na execução dos serviços da NovaGenesis. Na Seção 4.1 foi discutido as operações dos códigos desse Apêndice.

O fluxograma completo do código para experimentos de resolução de nomes e avaliação de desempenho é apresentado na Figura D.1.

O código desse Apêndice foi citado na Seção 4.1.

D.1 Código do script run.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#@Elcio Rosario
from serviceNG import *
from nmap import *
import random


######### Definir parametros Iniciais ################
def case_1():

        #recebe os atributos da rede
        net.pull_network(manager._network)

        op = raw_input("Utilizar todos os host listados?
         ↪  (y/n): ")
        if op == 'y':
                #Define parametro do arquivo
                 ↪  parametrosPGCS e printa a interface
                 ↪  de rede
```

```
18                  manager.define_pgcs(net._mac)
19
20          if op == 'n' :
21                  x = raw_input("Entre com o ip inicial e
                     ↪  final ou com o ip unico: ")
22                  x = x.split()
23
24                  #Define range dos ips listados
25                  if len(x) == 1:
26                          net.range(x[0],x[0])
27                  else:
28                          net.range(x[0],x[1])
29                  #Define parametro do arquivo
                     ↪  parametrosPGCS e printa a interface
                     ↪  de rede
30                  manager.define_pgcs(net._mac)
31
32
33          #Escreve o endereco mac do server no arquivo
             ↪  /PGCS/macServer
34          manager.write_mac()
35          '''if ips not in getParam.ipServidor:
36                  print "Erro - IPs do Servidor Fotos não
 ↪  estam contidos nos IPs encontrados"
37                  exit()'''
38
39  ######### Enviar NG para cliente ######################
40  def case_2():
41
42          manager.threads('sendNovagenesis',net._ip,
             ↪  waiting = 'yes')
43
44  ######### Apagar pasta        NG no cliente
     ↪  ###################
45  def case_3():
46
47          manager.threads('removeNovagenesis',net._ip,
             ↪  waiting = 'yes')
48
49  ######### Executar NG Core  ####################
50  def case_4():
51
52          cmd1 = "ipcmk -M 1024 ;
             ↪  ./novagenesis_central/clean.sh"
53          cmd2 = "cd novagenesis_central/; sh runCore.sh"
```

```
54          #cmd3 = "cd novagenesis_files; sh watch_info.sh >
             ↪  /dev/null 2>&1&"
55          a = subprocess.check_output(cmd1,shell=True)
56
57          #manager.exe_shell(cmd3)
58          manager.exe_shell(cmd1)
59          manager.exe_shell(cmd2)
60
61
62
63  ######### Executar PGCS no cliente ####################
64  def case_5():
65
66          #c = "cd " + manager._pathRemote + "; ipcmk -M
             ↪  1024 ;./clean.sh; ./watch_info.sh"
67          c = "cd " + manager._pathRemote + "; ipcmk -M
             ↪  1024 ;./clean.sh"
68
69
70          manager.threads('exe_cmd_root', net._ip, cmd = c,
             ↪  delay = 0.2)
71
72          #c = "cd " + manager._pathRemote + ";
             ↪  ./watch_info.sh > /dev/null 2>&1&;"
73          #manager.threads('exe_cmd_root', net._ip, cmd =
             ↪  c, delay = 0.2)
74
75          print "Executando PGCS"
76          cmd = ['xterm']
77          p = "sshpass -p 'gandalf' ssh -o
             ↪  StrictHostKeychecking=no root@" +
             ↪  str(manager._ipServer[0]) + " 'cd
78          /home/ng/workspace/novagenesis_files/PGCS_files/;
 ↪  ./runPGCS.sh';"
79          cmd.extend(['-e', p])
80          subprocess.Popen(cmd, stdout=subprocess.PIPE)
81
82  ######## Executar PGCS e HTS ###################
83  def case_6():
84
85
86          #c = "cd " + manager._pathRemote + "; ipcmk -M
             ↪  1024 ;./clean.sh; ./watch_info.sh"
87          c = "cd " + manager._pathRemote + "; ipcmk -M
             ↪  1024 ;./clean.sh"
88
```

```
89          manager.threads('exe_cmd_root', net._ip, cmd = c,
             ↪  delay = 0.2)
90
91          for i in range(1,len(net._ip)):
92
93                  print "Executando PGCS e
                     ↪  HTS"
94                  cmd = ['xterm']
95                  p = "sshpass -p 'gandalf' ssh -o
                     ↪  StrictHostKeychecking=no root@" +
                     ↪  str(net._ip[i]) + " 'cd
                    /home/ng/workspace/novagenesis_files/
97                  HTS_files/; ./runHTS.sh ; cd
98                  /home/ng/workspace/novagenesis_files/
99                  PGCS_files/; ./runPGCS.sh';"
100                 cmd.extend(['-e', p])
101                 subprocess.Popen(cmd,
                     ↪  stdout=subprocess.PIPE)
102
103
104 ######### Executar NB ####################
105
106 def case_7():
107
108         print "Executando NB"
109         cmd = ['xterm']
110         p = "sshpass -p 'gandalf' ssh -o
             ↪  StrictHostKeychecking=no root@" +
             ↪  str(manager._ipServer[0]) + " 'cd
             ↪  /home/ng/workspace/novagenesis_files/;
             ↪  ./runClient.sh';"
111         cmd.extend(['-e', p])
112         subprocess.Popen(cmd, stdout=subprocess.PIPE)
113
114
115
116 ######## Download Relatorios ######################
117 def case_8():
118
119         c1 = "cd " + manager._pathRemote + "; mv memo.txt
             ↪  net.txt cpu.txt ./NBSimpleTest/"
120         manager.threads('exe_cmd_root', net._ip, cmd =
             ↪  c1, delay = 0.2)
121
122         ips = net._ip
```

```
123         name_dir = raw_input("Salvar relatorio com nome:
             ↪ ")
124         c = "mkdir
             ↪ /home/ng/ngDocker/relatorios_pub_sub/"+name_dir
125         manager.exe_shell(c)
126
127         manager.exe_shell("sshpass -p 'gandalf' scp
             ↪ root@"+str(manager._ipServer[0])+":/home/ng/
128         workspace/novagenesis_files/NBSimpleTest/*"+
129         " /home/ng/ngDocker/relatorios_pub_sub/
130         "+name_dir)
131
132         #manager.exe_shell("cd
             ↪ /home/ng/ngDocker/novagenesis_files; mv
             ↪ memo.txt net.txt cpu.txt
             ↪ /home/ng/ngDocker/relatorios_pub_sub/;")
133         #manager.exe_shell("cd
             ↪ /home/ng/ngDocker/relatorios_pub_sub; mv
             ↪ memo.txt memo-server.txt; mv cpu.txt
             ↪ cpu-server.txt;mv net.txt net-server.txt; mv
             ↪ memo-server.txt cpu-server.txt net-server.txt
             ↪ ./"+name_dir)
134
135
136 ###### SubMenu  Serviços
     ↪ ###################################
137 def case_9():
138         subDict = {1: subcase_1, 2: subcase_2, 3:
             ↪ subcase_3}
139         subDict[input('1 - Excluir todos PGCSes\n' + '2 -
             ↪ Excluir todos HTSes\n'+'3 - Excluir
             ↪ NBSimpleTest\n'
140                 + 'Escolhar uma opção: ')]()
141
142 def subcase_1():
143
144         c =  "sudo -S killall PGCS;sudo -S killall
             ↪ watch_info;"
145
146         manager.threads('exe_cmd_root',net._ip,cmd = c,
             ↪ waiting = 'yes')
147
148 def subcase_2():
149
150         c = "sudo -S killall HTS;"
```

```
151         manager.threads ('exec_cmd_root', net._ip, cmd =
             ↪  c, waiting = 'yes')
152
153 def subcase_3():
154
155         c = "sudo -S pkill NBSimpleTestApp;"
156
157         manager.threads ('exe_cmd_root', net._ip, cmd =
             ↪  c, waiting = 'yes')
158
159 ######## Finalizar Experimentos
     ↪  ##########################
160
161 def case_10():
162
163         print "---------------- GOOD BYE
             ↪  ------------------------"
164         exit()
165
166
167 def switch(x):
168         dict[x]()
169
170 ########## -   MENU    -
     ↪  ################################
171
172 if __name__ == '__main__':
173
174         dict = {1 : case_1, 2 : case_2, 3 : case_3, 4 :
             ↪  case_4,5: case_5, 6 : case_6, 7 : case_7, 8 :
             ↪  case_8,9: case_9,10 : case_10}
175
176         #Inicializa class Network.nmap
177         net = Network()
178         #Inicializa class Config.serviceNG
179         manager = Config()
180         #Lê configuracoes iniciais
181         manager.read()
182         #printa configuracoes iniciais
183         print (manager)
184
185         while True:
186
187                 time.sleep(0.5)
188                 print "\n************ NOVAGENESIS
                     ↪  **************************\n"
```

```
189                     switch(input('1 - Inicializar
                         ↪  Parâmetros\n'+'2 - Enviar arquivos
                         ↪  NG\n'+ '3 - Apagar Diretório\n' +
190                     '4 - Executar Core NG\n' + '5 - Executar
                         ↪  PGCS no cliente\n'+ '6 - Executar
                         ↪  PGCSes e HTSes\n' +
191                     '7 - Executar NBSimpleTest\n' + '8 -
                         ↪  Salvar Relatório\n' +'9 - Serviços\n'
                         ↪  +'10 - Finalizar Experimentos\n' +
192                     'Entre com a Opção: '))
193
```

D.2 Código do arquivo config.ng

```
1  pathRemote = /home/ng/workspace/novagenesis_files/
2  username = root
3  password = gandalf
4  delay_timer = 10
5  network = 172.17.0.0/24
6  ipServer = 172.17.0.3
```

D.3 Código do script nmap.py

```
1  # -*- coding: utf-8 -*-
2  #@Elcio Rosario
3  import os,threading,subprocess
4
5  class Network(object):
6
7  def __init__(self):
8          self._ip = []
9          self._mac = []
10         self._myip = None
11
12
13 def pull_network(self,rede):
14         print 'Buscando na rede...'
15         aux = subprocess.check_output("hostname -I",
            ↪  shell=True)
16
17         if len(aux) > 1:
18                 self._myip = aux.split()[0]
19                 print "IP local: "+ self._myip
20
21         else:
```

```
22                  print "Erro ao buscar IP, verifique sua
                     ↪  conexao de rede"
23                  exit()
24
25          p = subprocess.check_output("nmap -sP -n "+ rede
             ↪  +" | egrep -i '(scan|MAC)'", shell=True)
26          p = p.split()
27          bridge = 1
28          while(1):
29                  x = p.index("for")
30                  p[:x+1] = []
31
32          if self._myip == p[0]:
33                  return
34
35          if bridge == 0:
36
37                  print p[0]
38                  self._ip.append(p[0])
39                  x = p.index("Address:")
40                  p[:x+1] = []
41                  self._mac.append(p[0].lower())
42
43                  bridge = 0
44
45  def __str__ (self):
46          return ("IP local: " + str(self._myip)+
47                              "\nIPs Disponiveis: " +
                                 ↪  str(self._ip))
48
49  def range(self,st,end):
50          try:
51                  x = self._ip.index(st)
52                  y = self._ip.index(end)+1
53                  self._ip = self._ip[x:y]
54                  self._mac = self._mac[x:y]
55          except:
56                  print "faixa de ip nao existe"
57                  exit()
```

D.4 Código do script serviceNG.py

```
1  # -*- coding: utf-8 -*-
2  #!/usr/bin/env python
3  #@Elcio Rosario
4  import os,threading,subprocess,time
```

```
5  from sshClient import sshClient
6
7  ####################################
8
9  class Config(object,):
10
11 def __init__ (self,pathRemote = None, username = None,
    ↪  password = None, delay_timer = None, network = None,
    ↪  ipServer = []):
12 self._pathlocal =  os.getcwd() + "/novagenesis_files"
13 self._pathRemote = pathRemote #Local da pasta aonde sera
    ↪  enviado os arquivos
14 self._username = username #nome do usuario remoto
15 self._password = password #senha de usuario remoto
16 self._delay_timer = delay_timer #Delay para executar App
    ↪  Fotos em cada um dos Clientes
17 self._network = network
18 self._ipServer = ipServer
19
20
21 def read(self):
22 try:
23         inputFile = open('config.ng', 'rt')
24         args = inputFile.read()
25         args = args.split()
26
27         if('pathRemote' in args):
28                 self._pathRemote =
                    ↪  args[args.index('pathRemote')+ 2]
29
30         if('username' in args):
31                 self._username =
                    ↪  args[args.index('username')+ 2]
32
33         if('password' in args):
34                 self._password =
                    ↪  args[args.index('password')+ 2]
35
36         if('delay_timer' in args):
37                 self._delay_timer =
                    ↪  args[args.index('delay_timer')+ 2]
38
39         if('network' in args):
40                 self._network =
                    ↪  args[args.index('network')+ 2]
41
```

```
42          if('ipServer' in args):
43                  args[:args.index('ipServer') + 2] = []
44                  self._ipServer =
                     ↪  args
45
46                  inputFile.close()
47          except:
48                  print "Erro, arquivo 'config.ng' nao
                     ↪  encontrado"
49                  exit()
50  def sendNovagenesis(self,ip,username,password,pathlocal,
51  pathRemote):
52
53          ssh = sshClient('root',ip,username,password)
54          ssh.connect()
55          ssh.push(pathlocal,pathRemote)
56          ssh.write_root('chmod -R 777 ' + pathRemote)
57          ssh.close()
58
59  def
     ↪  removeNovagenesis(self,ip,username,password,pathRemote):
60          ssh = sshClient('root',ip,username,password)
61          ssh.connect()
62          ssh.write_root("rm -Rf "+ pathRemote)
63          ssh.close()
64
65  def exe_cmd(self,ip,username,password,cmd):
66          ssh = sshClient('root',ip,username,password)
67          ssh.connect()
68          ssh.write_root(cmd)
69          ssh.close()
70
71  def exe_cmd_root(self,ip,username,password,cmd):
72      ssh = sshClient('root',ip,username,password)
73          ssh.connect()
74          ssh.write_root(cmd)
75          ssh.close()
76
77  def exe_shell(self,cmd):
78          subprocess.call(cmd, shell=True)
79
80  def report(self,ip,username,password,pathRemote,
     ↪  name_dir):
81          report_files = ['imgs/App_Core_pubrtt{}.dat',
82          'imgs/App_Core_subrtt{}.dat']
```

```
83          report_server =
             ↪  ['imgs_received/App_Core_subrtt{}.dat']
84
85          ssh = sshClient('root',ip,username,password)
86          ssh.connect()
87
88
89          mkd = "/"+ os.getcwd() +'/relatorios_pub_sub/' +
             ↪  name_dir
90
91          try:
92                  os.mkdir(mkd)
93                  except OSError:
94                  pass
95
96
97  if ip not in self._ipServer:
98          for name in report_files:
99          #adiciona no nome do arquivo o numero do pc
             ↪  retirando a ultima parte do IP
100         report_name =
             ↪  name.format('_pc'+str(ip.split('.')[-1])+'_')
101         #retira apenas o nome do arquivo, para salvar na
             ↪  pasta relatorios_pub_sub
102         report_name = report_name.split('/')[-1]
103         ssh.pull( pathRemote + name.format(''), mkd[1:]
             ↪  +'/'+ report_name )
104
105 else:
106         for name in report_server:
107         #adiciona no nome do arquivo o numero do pc
             ↪  retirando a ultima parte do IP
108         report_name =
             ↪  name.format('_pc'+str(ip.split('.')[-1])+'_')
109         #retira apenas o nome do arquivo, para salvar na
             ↪  pasta relatorios_pub_sub
110         report_name = report_ame.split('/')[-1]
111         ssh.pull( pathRemote + name.format(''), mkd[1:]
             ↪  +'/'+ report_name )
112
113
114 self.exe_shell("chmod -R 777 ./")
115         #Limpa pasta HTS
116 self.exe_shell("cd " + os.getcwd() +
     ↪  "/novagenesis_central/HTS_files;rm -f *.jpg")
117 #Fecha conexao
```

```
118         ssh.close()
119
120
121 def define_pgcs(self,listMacs):
122         aux = " 0 Intra_Domain -p "
123         file =
            ↪  open("novagenesis_central/parametrosPGCS","w")
124         interface = self.pull_interface()
125         print "interface da rede: " + str(interface)
126
127         for mac in listMacs:
128                 aux += "Ethernet Intra_Domain "+
                    ↪  str(interface) +" "+ str(mac) +" 1200
                    ↪  "
129
130         file.write(aux)
131         file.close()
132
133 def write_mac(self):
134
135         aux = os.popen("ifconfig")
136         args = aux.read()
137         aux.close()
138         args = args.split()
139         mac = args[args.index("HW")+1]
140
141         file =
            ↪  open("novagenesis_files/PGCS_files/macServer",
            ↪  "w")
142         file.write(mac)
143         file.close()
144
145
146 def pull_interface(self):
147
148         aux = os.popen("ifconfig")
149         args = aux.read()
150         aux.close()
151         args = args.split()
152
153         return args[0]
154
155 def __str__ (self):
156
157         return ("************Configurações
            ↪  Iniciais**************************" +
```

```
158                             "\nPathRemote: " + str
                                ↪  (self._pathRemote) +
159                             "\nUsername: " +
                                ↪  str(self._username) +
160                             "\nPassword: " +
                                ↪  str(self._password) +
161                             "\nDelay Timer: " +
                                ↪  str(self._delay_timer) +
162                             "\nNetwork: " +
                                ↪  str(self._network) +
163                             "\nIpServidor: " +
                                ↪  str(self._ipServer) +
164                             "\n*****************")
165
166 def threads(self,name, ips, cmd = None, waiting = None,
    ↪  delay = 0.0):
167         monitores = []
168         for ip in ips:
169                 monitores.append(self.Monitor(name, ip,
                    ↪  self._username, self._password,
                    ↪  self._pathlocal ,self._pathRemote,
                    ↪  cmd))
170
171         # Execute as Threads
172         for monitor in monitores:
173         monitor.start()
174                 time.sleep(float(delay))
175
176         if waiting:
177                 for t in monitores:
178                         t.join()
179
180 class Monitor(threading.Thread):
181
182 def __init__(self,name,ip,username,password,
183 pathlocal,pathRemote, cmd = ''):
184         threading.Thread.__init__(self)
185         self.name = name
186         self.ip = ip
187         self.username = username
188         self.password = password
189         self.pathlocal = pathlocal
190         self.pathRemote = pathRemote
191         self.cmd = cmd
192
193 def run(self):
```

```
194
195 print "Starting " + str(self.name) + " - " + str(self.ip)
196
197 if self.name in 'sendNovagenesis':
198 Config().sendNovagenesis(self.ip,self.username,
199 self.password, self.pathlocal, self.pathRemote)
200
201 if self.name in 'removeNovagenesis':
202 Config().removeNovagenesis(self.ip, self.username,
    ↪  self.password, self.pathRemote)
203
204 if self.name in 'exe_cmd_root':
205 Config().exe_cmd_root(self.ip, self.username,
    ↪  self.password, self.cmd)
206
207 if self.name in 'exe_cmd':
208 Config().exe_cmd(self.ip, self.username, self.password,
    ↪  self.cmd)
209
210 if self.name in 'report':
211 Config().report(self.ip, self.username, self.password,
    ↪  self.pathRemote, self.cmd)
212
213                 print "Exiting " + str(self.ip)
214
```

D.5 Código do script sshClient.py

```
1  #@Elcio Rosario
2  import paramiko,os
3
4  class sshClient:
5
6  #construtor cliente ssh
7  def __init__(self,userRemote,ip,username,password):
8  #cria uma instacia do cliente ssh
9  self.ssh = paramiko.SSHClient()
10 #parametro para aceitar automaticamente as chaves
11 self.ssh.set_missing_host_key_policy
12 (paramiko.AutoAddPolicy())
13 self.username = username
14 self.password = password
15 self.ip = ip
16 self.userRemote = userRemote
17 self.connected = False
18
```

```
19 def connect(self):
20 try:
21         self.ssh.connect(self.ip, username =
            ↪  self.username,password =self.password
            ↪  ,timeout = 5)
22         self.connected = True
23         #protoco sftp para transferencia usando
            ↪  biblioteca paramiko
24         self.sftp = self.ssh.open_sftp()
25
26 except:
27         print self.ip + ' Time out '
28         self.connected = False
29
30
31 def write(self,cmd):
32
33 if not self.connected:
34         print "ERRO AO SE CONECTAR"
35         return
36 else:
37         stdin,stdout,stderr = self.ssh.exec_command(cmd)
38 if not stdout.read() == "":
39         print stdout.read()
40
41 def write_root(self,cmd):
42
43 if not self.connected:
44         print "ERRO AO SE CONECTAR"
45         return
46         else:
47         stdin,stdout,stderr =
            ↪  self.ssh.exec_command(cmd,get_pty = True)
48         stdin.write(self.password+"\n")
49
50 if not stdout.read() == "":
51         print stdout.read()
52
53 def read():
54         pass
55
56 #envia todos os arquivos do diretorio selecionado
57 def push(self,localfile,remotefile):
58 try:
59     #gera os nome dos arquivos no diretorio localfile
```

```
60          for root, dirs, files in os.walk(localfile,
             ↪  topdown=False):
61          for name in dirs:
62          #print 'CRIANDO DIRETORIO: ' + remotefile + name
63          self.ssh.exec_command('mkdir -p '+ remotefile+
             ↪  name +' ;cd ; chmod -R 777 '+ remotefile +
             ↪  name)
64
65          print 'TRANSFERINDO
             ↪  ARQUIVOS......'
66          for root, dirs, files in
             ↪  os.walk("./novagenesis_files",
             ↪  topdown=False):
67          for name in files:
68          fname = os.path.join(root, name)[2:]
69
70  self.sftp.put(localfile[:-17]+fname,remotefile[:-18]+fname)
71
72          '''for file in os.listdir(localfile):
73          path  = os.path.join(localfile,file)
74          if not os.path.isdir(path):
75
76          self.sftp.put(localfile +file,remotefile+file)
77                                      '''
78          except:
79                  print "ERRO - CAMINHO INVALIDO"
80
81
82  def pull(self,remote,local):
83          self.sftp.get(remote, local)
84
85  def close(self):
86          self.ssh.close()
```

D.6 Fluxograma de Experimentos de NBs

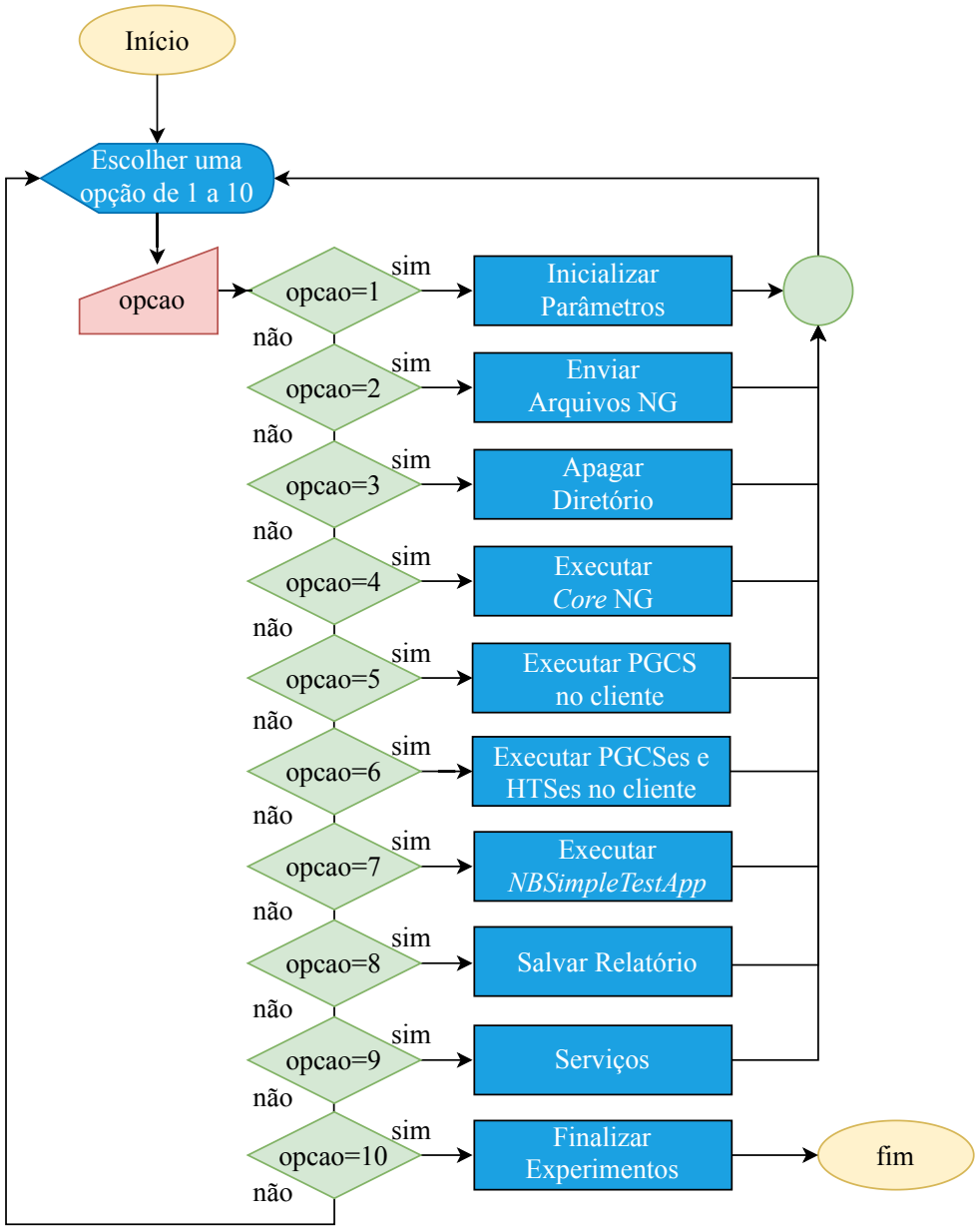


Figura D.1: *Fluxograma completo do algoritmo para cenário de resolução de nomes NovaGenesis em um domínio.*

Apêndice E

Menu de Gerência de Distribuição de conteúdo NG

Este Apêndice contém as linhas de código do *script* run.py. Foi criado com o objetivo de automatizar os experimentos de distribuição de conteúdo (fotos) da NovaGenesis. Também foi discutido as operações desse Apêndice na Seção 4.1.

E.1 Código do script run.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#@Elcio Rosario
from serviceNG import *
from nmap import *
import random


######### Definir parametros Iniciais #################
def case_1():

        #recebe os atributos da rede
        net.pull_network(manager._network)

        op = raw_input("Utilizar todos os host listados?
         ↪  (y/n): ")
        if op == 'y':
                #Define parametro do arquivo
                 ↪  parametrosPGCS e printa a interface
                 ↪  de rede
                manager.define_pgcs(net._mac)

        if op == 'n' :
```

```
21                  x = raw_input("Entre com o ip inicial e
                     ↪  final ou com o ip unico: ")
22                  x = x.split()
23
24                  #Define range dos ips listados
25                  if len(x) == 1:
26                          net.range(x[0],x[0])
27                  else:
28                          net.range(x[0],x[1])
29                  #Define parametro do arquivo
                     ↪  parametrosPGCS e printa a interface
                     ↪  de rede
30                  manager.define_pgcs(net._mac)
31
32
33          #Escreve o endereco mac do server no arquivo
             ↪  /PGCS/macServer
34          manager.write_mac()
35          '''if ips not in getParam.ipServidor:
36                  print "Erro - IPs do Servidor Fotos não
 ↪  estam contidos nos IPs encontrados"
37                  exit()'''
38
39  ######### Enviar NG para cliente ######################
40  def case_2():
41
42          manager.threads('sendNovagenesis',net._ip,
             ↪  waiting = 'yes')
43
44  ######### Apagar pasta        NG no cliente
     ↪  ###################
45  def case_3():
46
47          manager.threads('removeNovagenesis',net._ip,
             ↪  waiting = 'yes')
48
49  ######### Executar NG Core  ####################
50  def case_4():
51
52          cmd1 = "ipcmk -M 1024 ;
             ↪  ./novagenesis_central/clean.sh"
53          cmd2 = "cd novagenesis_central/; sh runCore.sh"
54          a = subprocess.check_output(cmd1,shell=True)
55
56          manager.exe_shell(cmd1)
57          manager.exe_shell(cmd2)
```

```
58
59
60
61
62  ######### Executar Servidor Fotos ####################
63  def case_5():
64
65          if not net._myip == manager._ipServer[0]:
66                  print "Servidor de Fotos
                     ↪  remoto"
67                  cmd = ['xterm']
68                  p = "ssh -Y root@" +
                     ↪  str(manager._ipServer[0]) + " 'cd
                     ↪  /home/ng/workspace/novagenesis_files/;
69                  ./runServerFotos.sh'"
70                  cmd.extend(['-e', p])
71                  subprocess.Popen(cmd,
                     ↪  stdout=subprocess.PIPE)
72
73          ########## Servidor de fotos junto com core NG
             ↪  ##############
74          else:
75                  print "Servidor Fotos Local"
76                  #cmd = ['xterm']
77                  #p = 'cd
                     ↪  novagenesis_files/;./runServerFotos.sh'
78                  #cmd.extend(['-e', p])
79                  cmd = "cd novagenesis_files/; sh
                     ↪  runServerFotos.sh"
80                  manager.exe_shell(cmd)
81                  #subprocess.Popen(cmd,
                     ↪  stdout=subprocess.PIPE)
82
83
84  ######### Executar PGCS no cliente ####################
85  def case_6():
86
87          c = "cd " + manager._pathRemote + "; ipcmk -M
             ↪  1024 ;./clean.sh; cd PGCS_files/; sudo -S sh
             ↪  runPGCS.sh > /dev/null;"
88
89          manager.threads('exe_cmd_root', net._ip, cmd = c,
             ↪  delay = 0.2)
90
91  ######### Executar App no cliente ####################
92  def case_7():
```

```
93
94          c = "cd " + manager._pathRemote + ";
             ↪  ./runClient.sh > /dev/null;"
95          ips = net._ip
96
97          if manager._ipServer[0] in ips:
98                  #exclui ip do servidor
99                  aux_ips =
                     ↪  ips[ips.index(manager._ipServer[0])+1
                     ↪  : ]
100                 manager.threads('exe_cmd_root', aux_ips,
                     ↪  cmd = c, delay =
                     ↪  manager._delay_timer)
101         else:
102                 manager.threads('exe_cmd_root', ips, cmd
                     ↪  = c, delay = manager._delay_timer)
103
104 ######## Criar Fotos no Cliente ##########################
105 def case_8():
106
107         while True:
108                 op = raw_input("Criar fotos:\n1 -
                     ↪  Diferentes\n2 - Iguais\n3 -
                     ↪  Mista\nEntre com a Opcao:")
109                 if (op == '1') or (op == '2') or (op ==
                     ↪  '3'):
110                         break
111                 else:
112                         print 'Opcao Invalida!'
113
114
115         qtd = raw_input("Entre com a quantidade de
             ↪  Fotos:")
116         c1 = "cd " + manager._pathRemote + "imgs; python
             ↪  CriarFotos.py "+ "different " + qtd + ";"
117         c2 = "cd " + manager._pathRemote + "imgs; python
             ↪  CriarFotos.py "+ "equal " + qtd + ";"
118
119         if (op == '1'):
120                 manager.threads ('exe_cmd_root', net._ip,
                     ↪  cmd = c1, waiting = 'yes' )
121
122         elif (op == '2'):
123                 manager.threads ('exe_cmd_root', net._ip,
                     ↪  cmd = c2, waiting = 'yes' )
124
```

```
125 ###### Limpar pasta Imagens ############
126 def case_9():
127
128         c = "cd " + manager._pathRemote + "imgs;rm -f
            ↪  *.jpg *.txt *.dat" #rm -f !(*.py|*.ini)
129
130         manager.threads('exe_cmd_root',net._ip, cmd = c,
            ↪  waiting = 'yes')
131
132 ######## Download Relatorios ######################
133 def case_10():
134
135         ips = net._ip
136         name_dir = raw_input("Salvar relatorio com nome:
            ↪  ")
137
138
139         if manager._ipServer[0] in ips:
140                 aux_ips =
                    ↪  ips[ips.index(manager._ipServer[0])+1
                    ↪  : ]
141                 manager.threads('report',aux_ips, waiting
                    ↪  = 'yes', cmd = name_dir)
142                 print "Digite a senha:"
143                 manager.exe_shell("scp
                    ↪  root@"+str(manager._ipServer[0])+
144                 ":/home/ng/workspace/novagenesis_files/
145                 imgs_received/*"+
146                 " /home/ng/ngDocker/
147                 relatorios_pub_sub/"+name_dir)
148
149         else:
150                 manager.threads('report',ips, waiting =
                    ↪  'yes', cmd = name_dir)
151
152                 thisReport =
                    ↪  'novagenesis_files/imgs_received/*'
153                 if(os.path.isfile(thisReport)):
154                         manager.exe_shell("cp " +
                            ↪  thisReport + "
                            ↪  relatorios_pub_sub/" +
                            ↪  name_dir)
155
156
157
158
```

```
159
160 ######## Encerrar PGCS no cliente ######################
161 def case_11():
162
163         c =  "sudo -S killall PGCS;"
164
165         manager.threads('exe_cmd_root',net._ip,cmd = c,
            ↪  waiting = 'yes')
166
167
168 ####### Encerrar App no cliente ####################
169 def case_12():
170
171         c = "sudo -S pkill App;"
172
173         manager.threads ('exe_cmd_root', net._ip, cmd =
            ↪  c, waiting = 'yes')
174
175 ###### Editar arquivo config.ng #############
176 def case_13():
177         manager.exe_shell("nano config.ng")
178         exit()
179
180 ###### SubMenu  Servicos
    ↪  ##################################
181 def case_14():
182         subDict = {1: subcase_1, 2: subcase_2, 3:
            ↪  subcase_3}
183         subDict[input('1 - LIGAR TODOS PCS\n' + '2 -
            ↪  REINICIAR TODOS PCS\n'+'3 - DESLIGAR TODOS
            ↪  PCS\n'
184                 + 'Entre com a opcao: ')]()
185
186 def subcase_1():
187
188         print "EM CONSTRUCAO"
189
190 def subcase_2():
191
192         c = "sudo -S reboot"
193         manager.threads ('exec_cmd_root', net._ip, cmd =
            ↪  c, waiting = 'yes')
194
195 def subcase_3():
196
197         c = "sudo -S poweroff"
```

```
198         manager.threads ('exe_cmd_root', net._ip, cmd =
             ↪  c, waiting = 'yes')
199
200 def case_15():
201
202         print "--------------- GOOD BYE
             ↪  ------------------------"
203         exit()
204
205
206 def switch(x):
207         dict[x]()
208
209 ########## -   MENU    -
     ↪  ##################################
210
211 if __name__ == '__main__':
212
213         dict = {1 : case_1, 2 : case_2, 3 : case_3, 4 :
             ↪  case_4,5: case_5, 6 : case_6, 7 : case_7, 8 :
             ↪  case_8,9: case_9,10 : case_10,
214                         11 : case_11, 12 : case_12, 13 :
                             ↪  case_13, 14 : case_14, 15 :
                             ↪  case_15}
215
216         #Inicializa class Network.nmap
217         net = Network()
218         #Inicializa class Config.serviceNG
219         manager = Config()
220         #Lê configuracoes iniciais
221         manager.read()
222         #printa configuracoes iniciais iniciais
223         print (manager)
224
225         while True:
226
227                 time.sleep(0.5)
228                 print "\n************ NOVA GENESIS
                     ↪  ***************************\n"
229                 switch(input('1 - INICIALIZAR
                     ↪  PARAMETROS\n'+'2 - ENVIAR NOVA
                     ↪  GENESIS\n'+ '3 - APAGAR DIRETORIO\n'
                     ↪  +
230                 '4 - EXECUTAR CORE NG\n' + '5 - EXECUTAR
                     ↪  SERVIDOR FOTOS\n'+'6 - EXECUTAR PGCS
                     ↪  NO CLIENT\n'+
```

```
231                 '7 - EXECUTAR APP FOTOS NO CLIENT\n' + '8
                     ↪  - CRIAR FOTOS\n' + '9 - LIMPAR PASTA
                     ↪  IMAGENS\n'+
232                 '10 - BAIXAR RELATORIOS\n' +'11 -
                     ↪  FINALIZAR PGCS\n' + '12 - FINALIZAR
                     ↪  CLIENT\n'+
233                 '13 - ALTERAR CONFIGURAÇOES\n' +'14 -
                     ↪  SERVICOS\n' +'15 - SAIR\n' + 'Entre
                     ↪  com a Opcao: '))
234
```

E.2 Código do script config.ng

```
1 pathRemote = /home/ng/workspace/novagenesis_files/
2 username = root
3 password = gandalf
4 delay_timer = 10
5 network = 172.17.0.0/24
6 ipServer = 172.17.0.3
```

E.3 Código do script nmap.py

```
1  # -*- coding: utf-8 -*-
2  #@Elcio Rosario
3  import os,threading,subprocess
4
5  class Network(object):
6
7  def __init__(self):
8          self._ip = []
9          self._mac = []
10         self._myip = None
11
12
13 def pull_network(self,rede):
14         print 'Buscando na rede...'
15         aux = subprocess.check_output("hostname -I",
            ↪  shell=True)
16
17         if len(aux) > 1:
18                 self._myip = aux.split()[0]
19                 print "IP local: "+ self._myip
20
21         else:
```

```
22                  print "Erro ao buscar IP, verifique sua
                     ↪  conexao de rede"
23                  exit()
24
25                  p = subprocess.check_output("nmap -sP -n
                     ↪  "+ rede +" | egrep -i '(scan|MAC)'",
                     ↪  shell=True)
26          p = p.split()
27          bridge = 1
28          while(1):
29                  x = p.index("for")
30                  p[:x+1] = []
31
32                  if self._myip == p[0]:
33                          return
34
35                  if bridge == 0:
36
37                          print p[0]
38                          self._ip.append(p[0])
39                          x = p.index("Address:")
40                          p[:x+1] = []
41                          self._mac.append(p[0].lower())
42
43                  bridge = 0
44
45  def __str__ (self):
46          return ("IP local: " + str(self._myip)+
47                          "\nIPs Disponiveis: " +
                             ↪  str(self._ip))
48
49  def range(self,st,end):
50          try:
51                  x = self._ip.index(st)
52                  y = self._ip.index(end)+1
53                  self._ip = self._ip[x:y]
54                  self._mac = self._mac[x:y]
55          except:
56                  print "faixa de ip nao existe"
57                  exit()
58
59
```

E.4 Código do script serviceNG.py

```
# -*- coding: utf-8 -*-
#!/usr/bin/env python
#@Elcio Rosario
import os,threading,subprocess,time
from sshClient import sshClient

###############################################

class Config(object,):

def __init__ (self,pathRemote = None, username = None,
 ↪  password = None, delay_timer = None, network = None,
 ↪  ipServer = []):
        self._pathlocal =  os.getcwd() +
         ↪  "/novagenesis_files"
        self._pathRemote = pathRemote #Local da pasta
         ↪  aonde sera enviado os arquivos
        self._username = username #nome do usuario remoto
        self._password = password #senha de usuario
         ↪  remoto
        self._delay_timer = delay_timer #Delay para
         ↪  executar App Fotos em cada um dos Clientes
        self._network = network
        self._ipServer = ipServer


def read(self):
try:
        inputFile = open('config.ng', 'rt')
        args = inputFile.read()
        args = args.split()

        if('pathRemote' in args):
                self._pathRemote =
                 ↪  args[args.index('pathRemote')+ 2]

        if('username' in args):
                self._username =
                 ↪  args[args.index('username')+ 2]

        if('password' in args):
                self._password =
                 ↪  args[args.index('password')+ 2]

```

```
36          if('delay_timer' in args):
37                  self._delay_timer =
                     ↪  args[args.index('delay_timer')+ 2]
38
39          if('network' in args):
40                  self._network =
                     ↪  args[args.index('network')+ 2]
41
42          if('ipServer' in args):
43                          args[:args.index('ipServer') + 2]
                             ↪  = []
44                  self._ipServer =
                     ↪  args
45
46  inputFile.close()
47  except:
48          print "Erro, arquivo 'config.ng' nao encontrado"
49          exit()
50
51  def sendNovagenesis(self,ip,username,password,
52  pathlocal,pathRemote):
53
54  ssh = sshClient('root',ip,username,password)
55  ssh.connect()
56  ssh.push(pathlocal,pathRemote)
57  ssh.write_root('chmod -R 777 ' + pathRemote)
58  ssh.close()
59
60  def removeNovagenesis(self,ip,username,password,
61  pathRemote):
62  ssh = sshClient('root',ip,username,password)
63  ssh.connect()
64  ssh.write_root("rm -Rf "+ pathRemote)
65  ssh.close()
66
67  def exe_cmd(self,ip,username,password,cmd):
68  ssh = sshClient('root',ip,username,password)
69  ssh.connect()
70  ssh.write_root(cmd)
71  ssh.close()
72
73  def exe_cmd_root(self,ip,username,password,cmd):
74  ssh = sshClient('root',ip,username,password)
75  ssh.connect()
76  ssh.write_root(cmd)
77  ssh.close()
```

```
78
79  def exe_shell(self,cmd):
80  subprocess.call(cmd, shell=True)
81
82  def report(self,ip,username,password,pathRemote,
     ↪  name_dir):
83  report_files = ['imgs/App_Core_pubrtt{}.dat',
84  'imgs/App_Core_subrtt{}.dat']
85  report_server = ['imgs_received/App_Core_subrtt{}.dat']
86
87  ssh = sshClient('root',ip,username,password)
88  ssh.connect()
89
90
91  mkd = "/"+ os.getcwd() +'/relatorios_pub_sub/' + name_dir
92
93  try:
94  os.mkdir(mkd)
95  except OSError:
96  pass
97
98
99  if ip not in self._ipServer:
100 for name in report_files:
101 #adiciona no nome do arquivo o numero do pc retirando a
     ↪  ultima parte do IP
102 report_name =
     ↪  name.format('_pc'+str(ip.split('.')[-1])+'_')
103 #retira apenas o nome do arquivo, para salvar na pasta
     ↪  relatorios_pub_sub
104         report_name = report_name.split('/')[-1]
105         ssh.pull( pathRemote + name.format(''), mkd[1:]
             ↪  +'/'+ report_name )
106
107 else:
108 for name in report_server:
109         #adiciona no nome do arquivo o numero do pc
             ↪  retirando a ultima parte do IP
110         report_name =
             ↪  name.format('_pc'+str(ip.split('.')[-1])+'_')
111         #retira apenas o nome do arquivo, para salvar na
             ↪  pasta relatorios_pub_sub
112         report_name = report_ame.split('/')[-1]
113         ssh.pull( pathRemote + name.format(''), mkd[1:]
             ↪  +'/'+ report_name )
114
```

```
115
116 self.exe_shell("chmod -R 777 ./")
117 #Limpa pasta HTS
118 self.exe_shell("cd " + os.getcwd() +
    ↪ "/novagenesis_central/HTS_files;rm -f *.jpg")
119 #Fecha conexao
120 ssh.close()
121
122
123 def define_pgcs(self,listMacs):
124 aux = " 0 Intra_Domain -p "
125 file = open("novagenesis_central/parametrosPGCS","w")
126 interface = self.pull_interface()
127 print "interface da rede: " + str(interface)
128
129 for mac in listMacs:
130         aux += "Ethernet Intra_Domain "+ str(interface)
            ↪ +" "+ str(mac) +" 1200 "
131
132 file.write(aux)
133 file.close()
134
135 def write_mac(self):
136
137 aux = os.popen("ifconfig")
138 args = aux.read()
139 aux.close()
140 args = args.split()
141 mac = args[args.index("HW")+1]
142
143 file = open("novagenesis_files/PGCS_files/macServer",
    ↪ "w")
144         file.write(mac)
145         file.close()
146
147
148 def pull_interface(self):
149
150 aux = os.popen("ifconfig")
151 args = aux.read()
152 aux.close()
153 args = args.split()
154
155 return args[0]
156
157 def __str__ (self):
```

```
158
159 return ("***Configurações Iniciais***" +
160                         "\nPathRemote: " + str
                             ↪  (self._pathRemote) +
161                 "\nUsername: " + str(self._username) +
162                 "\nPassword: " + str(self._password) +
163                 "\nDelay Timer: " +
                     ↪  str(self._delay_timer) +
164                         "\nNetwork: " +
                             ↪  str(self._network) +
165                         "\nIpServidor: " +
                             ↪  str(self._ipServer) +
166                 "\n**********************")
167
168 def threads(self,name, ips, cmd = None, waiting = None,
    ↪  delay = 0.0):
169 monitores = []
170 for ip in ips:
171         monitores.append(self.Monitor(name, ip,
             ↪  self._username, self._password,
             ↪  self._pathlocal ,self._pathRemote, cmd))
172
173 # Execute as Threads
174 for monitor in monitores:
175         monitor.start()
176         time.sleep(float(delay))
177
178 if waiting:
179         for t in monitores:
180                 t.join()
181
182 class Monitor(threading.Thread):
183
184 def __init__(self,name,ip,username,password,pathlocal,
185 pathRemote, cmd = ''):
186         threading.Thread.__init__(self)
187         self.name = name
188         self.ip = ip
189         self.username = username
190         self.password = password
191         self.pathlocal = pathlocal
192         self.pathRemote = pathRemote
193         self.cmd = cmd
194
195 def run(self):
196
```

```
197         print "Starting " + str(self.name) + " - " +
             ↪  str(self.ip)
198
199 if self.name in 'sendNovagenesis':
200         Config().sendNovagenesis(self.ip,self.username,
201         self.password, self.pathlocal, self.pathRemote)
202
203 if self.name in 'removeNovagenesis':
204         Config().removeNovagenesis(self.ip,
             ↪  self.username, self.password,
             ↪  self.pathRemote)
205
206 if self.name in 'exe_cmd_root':
207         Config().exe_cmd_root(self.ip, self.username,
             ↪  self.password, self.cmd)
208
209 if self.name in 'exe_cmd':
210         Config().exe_cmd(self.ip, self.username,
             ↪  self.password, self.cmd)
211
212 if self.name in 'report':
213         Config().report(self.ip, self.username,
             ↪  self.password, self.pathRemote, self.cmd)
214
215         print "Exiting " + str(self.ip)
216
```

E.5 Código do script sshClient.py

```
1  #@Elcio Rosario
2  import paramiko,os
3
4  class sshClient:
5
6  #construtor cliente ssh
7  def __init__(self,userRemote,ip,username,password):
8  #cria uma instacia do cliente ssh
9  self.ssh = paramiko.SSHClient()
10 #parametro para aceitar automaticamente as chaves
11 self.ssh.set_missing_host_key_policy
12 (paramiko.AutoAddPolicy())
13 self.username = username
14 self.password = password
15 self.ip = ip
16 self.userRemote = userRemote
17 self.connected = False
```

```
18
19 def connect(self):
20 try:
21         self.ssh.connect(self.ip, username =
            ↪  self.username,password =self.password
            ↪  ,timeout = 5)
22         self.connected = True
23         #protoco sftp para transferencia usando
            ↪  biblioteca paramiko
24         self.sftp = self.ssh.open_sftp()
25
26 except:
27         print self.ip + ' Time out '
28         self.connected = False
29
30
31 def write(self,cmd):
32
33 if not self.connected:
34         print "ERRO AO SE CONECTAR"
35         return
36         else:
37         stdin,stdout,stderr = self.ssh.exec_command(cmd)
38         if not stdout.read() == "":
39                 print stdout.read()
40
41 def write_root(self,cmd):
42
43 if not self.connected:
44         print "ERRO AO SE CONECTAR"
45         return
46 else:
47         stdin,stdout,stderr =
            ↪  self.ssh.exec_command(cmd,get_pty = True)
48         stdin.write(self.password+"\n")
49         if not stdout.read() == "":
50                 print stdout.read()
51
52 def read():
53 pass
54
55 #envia todos os arquivos do diretorio selecionado
56 def push(self,localfile,remotefile):
57 try:
```

```
58 for root, dirs, files in os.walk(localfile,
    ↪ topdown=False): #gera os nome dos arquivos no
    ↪ diretorio localfile
59         for name in dirs:
60         #print 'CRIANDO DIRETORIO: ' + remotefile + name
61                 self.ssh.exec_command('mkdir -p '+
                    ↪ remotefile+ name +' ;cd ; chmod -R
                    ↪ 777 '+ remotefile + name)
62
63 print 'TRANSFERINDO
    ↪ ARQUIVOS......'
64 for root, dirs, files in os.walk("./novagenesis_files",
    ↪ topdown=False):
65 for name in files:
66 fname = os.path.join(root, name)[2:]
67 self.sftp.put(localfile[:-17]+fname,
68 remotefile[:-18]+fname)
69
70 '''for file in os.listdir(localfile):
71                 path  = os.path.join(localfile,file)
72                 if not os.path.isdir(path):
73                         self.sftp.put(localfile
    ↪ +file,remotefile+file)
74 '''
75         except:
76                 print "ERRO - CAMINHO INVALIDO"
77
78
79 def pull(self,remote,local):
80 self.sftp.get(remote, local)
81
82 def close(self):
83 self.ssh.close()
84
```

E.6 Código do script clean.sh

```
1 #!/bin/bash
2
3 ME=`whoami`
4
5 IPCS_S=`ipcs -s | egrep "0x[0-9a-f]+ [0-9]+" | grep $ME |
    ↪ cut -f2 -d" "`
6 IPCS_M=`ipcs -m | egrep "0x[0-9a-f]+ [0-9]+" | grep $ME |
    ↪ cut -f2 -d" "`
```

```
IPCS_Q=`ipcs -q | egrep "0x[0-9a-f]+ [0-9]+" | grep $ME |
 ↪  cut -f2 -d" "`


for id in $IPCS_M; do
  ipcrm -m $id;
done

for id in $IPCS_S; do
  ipcrm -s $id;
done

for id in $IPCS_Q; do
  ipcrm -q $id;
done

rm -rfv /dev/shm/sem.*
```

E.7 Código do script runCore.sh

```
var1=`pwd`;
PGCS=`cat parametrosPGCS`

xterm -title "PGCS" -e "cd $var1/PGCS_files; ./PGCS .
 ↪  $PGCS;" &
xterm -title "GIRS" -e "cd $var1/GIRS_files/; sleep
 ↪  10;./GIRS ./;" &
xterm -title "PSS" -e "cd $var1/PSS_files/; sleep
 ↪  10;./PSS ./;" &
xterm -title "HTS" -e "cd $var1/HTS_files/; sleep
 ↪  10;./HTS ./;" &

#cd $var1/PGCS_files/; ./PGCS . $PGCS > $var1/pgcs.log &
#cd $var1/GIRS_files/; sleep 10;./GIRS ./ >
 ↪  $var1/girs.log &
#cd $var1/PSS_files/; sleep 10;./PSS ./ > $var1/pss.log &
#cd $var1/HTS_files/; sleep 10;./HTS ./ > $var1/hts.log &
```

E.8 Código do arquivo parametrosPGCS

```
 0 Intra_Domain -p Ethernet Intra_Domain eth0
 ↪  02:42:ac:11:00:03 1200 Ethernet Intra_Domain eth0
 ↪  02:42:ac:11:00:04 1200
```

E.9 Fluxograma de Experimentos com Fotos

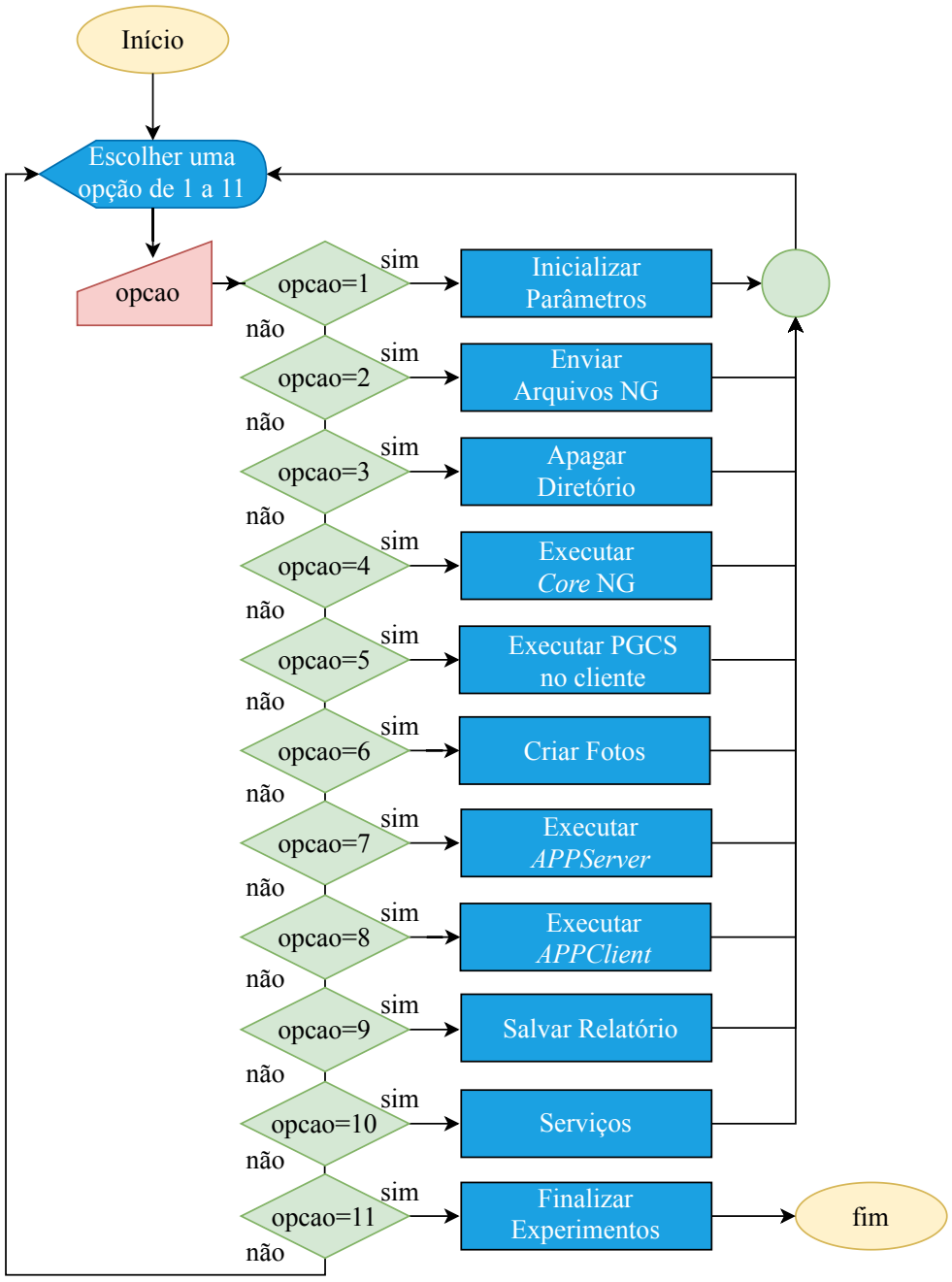


Figura E.1: *Fluxograma completo do algoritmo de experimentos com fotos.*