

Dissertação de Mestrado



UM MODELO PARA NAVEGAÇÃO WEB USANDO NOMEAÇÃO AUTO-CERTIFICÁVEL E PARADIGMA PÚBLICA/ASSINA

Daniel Lopes Fussia

Setembro de 2016

**Um Modelo para Navegação Web usando Nomeação
Auto-Certificável e Paradigma Pública/Assina**

Daniel Lopes Fussia

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

2016

FOLHA DE APROVAÇÃO

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

Prof. Dr. Antônio Marcos Alberti

INATEL

Prof. Dr. Christian Esteve Rothenberg

UNICAMP

Prof. Dr. Joel José Puga Coelho Rodrigues

INATEL

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

“Nenhuma grande descoberta foi feita jamais sem ousadia”.

-- Isaac Newton

Agradecimentos

Agradeço muito a Deus pelo que vivenciei, foram momentos difíceis, mas valorosos. Agradeço também aqueles que estiveram próximos e me deram palavras de apoio, também ao meu orientador, Prof. Dr. Antônio Marcos Alberti, que me ajudou muito nessa caminhada e teve toda a paciência de um grande educador. E ao Inatel por me fornecer a oportunidade de realizar este curso acadêmico que vou levar para a vida toda.

Índice

Lista de Figuras	viii
Lista de Abreviaturas e Siglas	x
Publicação.....	xii
Resumo	xiii
Abstract.....	xiv
Capítulo 1. Introdução	15
Capítulo 2. NovaGenesis	24
2.1. Principais Componentes da Implementação Atual.....	27
2.1.1. Hash Table Service (HTS).....	27
2.1.2. Generic Indirection Resolution Service (GIRS).....	27
2.1.3. Publish/Subscribe Service (PSS).....	28
2.1.4. Proxy/Gateway Service (PGS)	28
2.5. NovaGenesis Web	28
Capítulo 3. Armazenamento de Conteúdo.....	31
3.1 Modelos de Armazenamento da Internet.....	31
3.1.1 Tempo de expiração.....	32
3.1.2 Validação	33
3.2 Modelo de Armazenamento do Navegador NovaGenesis.....	34
Capítulo 4. Implementação da NovaGenesis Web	36
4.1. Componentes de implementação	36
4.1.1. QtWebKit	37
4.1.2 D-Bus.....	37
4.1.2.1 QtDBus	37
4.2. Componentes da <i>web</i>	38
4.2.1. NGConverter	38
4.2.1.1 Exemplo de conversão de um único <i>site</i>	39
4.2.2. NGAppPublisher	40
4.2.3. NGAppCommunicator.....	41
4.2.4. NGBrowser.....	42
4.2.4.1 NovaGenesis Search (<i>ngs://</i>)	42

4.2.4.2 NovaGenesis Unique (ngu://)	42
Capítulo 5. Resultados Experimentais.....	44
5.1 Preparação das páginas <i>web</i>	44
5.2 Inicialização da NovaGenesis.....	44
5.3 Publicação das páginas <i>web</i>	45
5.4 Navegação	47
5.5. Eficiência do <i>cache</i>	48
5.6. Teste em escala.....	49
5.6.1 Validação do algoritmo	49
5.6.2 Influência dos dados probabilísticos	52
5.6.3 Simulação do teste em escala	56
Capítulo 6. Conclusão	59
Referências	60
Anexos.....	64

Lista de Figuras

Figura 1 – Arpanet	15
Figura 2 - Visualização em um browser (direita) de um HTML (esquerda).....	16
Figura 3 - Quantidade de páginas indexadas pelo Google e Bing.....	17
Figura 4 - Acesso ao cache pelo browser	18
Figura 5 - Acesso ao cache via servidor proxy.....	18
Figura 6 - Modelo da Arquitetura NGBrowser e NovaGenesis.	21
Figura 7 - Fluxo de entrada e saída da conversão e publicação de sites.....	22
Figura 8 – Princípios de design da NovaGenesis.	24
Figura 9 – Arquitetura NovaGenesis ilustrada para uma rede local.....	25
Figura 10 - Cenário de publica/assina de objetos web.	30
Figura 11 - Demonstração do modelo de cache tempo de expiração.	32
Figura 12 - Fluxo de mensagens na primeira requisição de um objeto web.	33
Figura 13 - Fluxo de mensagens da segunda requisição do objeto web da Figura 12.....	34
Figura 14 - Demonstração do modelo de armazenamento do navegador NovaGenesis.	35
Figura 15 - Processo de conversão de site.....	39
Figura 16 - Conteúdo de um descritor.....	39
Figura 17 - Demonstração do antes e depois da conversão do Site Exemplo 01.	40
Figura 18 – Conversão do Site Exemplo 01.....	40
Figura 19 - Resultado da conversão dos 11 sites.....	44
Figura 20 - Quatro principais processos da NovaGenesis.....	45
Figura 21 – Log de execução do NGAppPublisher.....	46
Figura 22 - Mensagem NovaGenesis de publicação de arquivo.	46
Figura 23 - Associação de palavras-chave a um descritor.....	46
Figura 24 - HTS após o término das publicações.....	46
Figura 25 - NGAppCommunicator em espera por navegadores.	47
Figura 26 - Assinatura encontrada e obtida com sucesso.....	47
Figura 27 - Resultado do NGS de um site (esquerda) e de vários sites (direita).....	48
Figura 28 - Exibição da página HTML (esquerda) e o acesso com o NGU (direita).....	48
Figura 29 - Eficiência do cache do NGBrowser.....	49
Figura 30 - Dados do experimento do Capítulo 5.	50
Figura 31 - Teste do algoritmo de simulação utilizando dados experimentais do Capítulo 5..	50

Figura 32 - Relatório de saída em console do algoritmo	51
Figura 33 - Resultado da simulação para $p1=0,2$ e $p2=0,2$	52
Figura 34 - Resultado da simulação para $p1=0,8$ e $p2=0,2$	53
Figura 35 - Resultado da simulação para $p1=0,2$ e $p2=0,8$	54
Figura 36 - Resultado da simulação para $p1=0,8$ e $p2=0,8$	55
Figura 37 - Inicio da simulação do teste em escala.	56
Figura 38 - Eficiência no uso do cache do NGBrowser no teste em escala.	57
Figura 39 - Eficiência no uso do cache do navegador comum no teste em escala.	57
Figura 40 - Final da simulação do teste em escala.	58

Lista de Abreviaturas e Siglas

API	<i>Application Programming Interface</i>
ARPA	<i>Advanced Research Projects Agency</i>
CSS	<i>Cascading Style Sheets</i>
DNS	<i>Domain Name System</i>
ETag	<i>Entity Tag</i>
GIRS	<i>Generic Indirection Resolution Service</i>
HTML	<i>HyperText Markup Language</i>
HTS	<i>Hash Table Service</i>
IP	<i>Internet Protocol</i>
IMP	<i>Interface Message Processor</i>
IPC	<i>Inter-Process Communication</i>
IoT	<i>Internet of Things</i>
JSON	<i>JavaScript Object Notation</i>
LAN	<i>Local Area Network</i>
NB	<i>Name Binding</i>
NG	<i>NovaGenesis</i>
NGS	<i>NovaGenesis Search</i>
NGU	<i>NovaGenesis Unique</i>
NS	<i>Network Socket</i>
OSI	<i>Open Systems Interconnect</i>
P2P	<i>Peer-to-Peer</i>
PGS	<i>Proxy-Gateway Service</i>
PSS	<i>Publish/Subscribe Service</i>
QoS	<i>Qualidade de Serviço</i>
RMA	<i>Resource Management Agent</i>
RPC	<i>Remote Procedure Calling</i>
RWI	<i>Real-World Internet</i>
SCN	<i>Self-Certifying Name</i>
SLA	<i>Service Level Agreement</i>
TCP	<i>Transmission Control Protocol</i>
URI	<i>Uniform Resource Identifier</i>

URL	<i>Uniform Resource Local</i>
URN	<i>Uniform Resource Name</i>
WWW	<i>World Wide Web</i>

Publicação

ALBERTI, A.M.; FUSSIA, D.L.. Um Modelo para Navegação Web usando Nomeação Auto-Certificável e Publica/Assina. In: **Workshop de Redes P2P, Dinâmicas, Sociais e Orientadas a Conteúdo (WP2P+)**. XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos. Salvador, 2016.

Resumo

Este trabalho investiga um novo modelo para navegação na *world wide web*. A principal diferença é que ao invés de utilizarmos a arquitetura de protocolos da Internet atual, isto é o HTTP/TCP/IP, utilizamos uma arquitetura de Internet do futuro chamada NovaGenesis. A NovaGenesis encontra-se em desenvolvimento desde 2008 e atualmente conta com um protótipo em C++ para o Linux. O modelo proposto utiliza nomes auto-certificáveis no lugar de *hyperlinks* HTTP. Ao invés do modelo cliente/servidor, utilizamos comunicação que segue o paradigma publica/assina. O conteúdo das páginas *web* é armazenado em *caches* de rede, melhorando o desempenho da transferência de páginas. Reportamos uma implementação do modelo de *web* NovaGenesis com resultados de laboratório que comprovam suas vantagens em termos de eficiência, flexibilidade e segurança.

Palavras-chave: Internet do Futuro, NovaGenesis, Navegador, *Cache*, Publica e Assina.

Abstract

This thesis investigates a new model for world wide web navigation. The main difference is that instead of using protocols from current Internet architecture, i.e. HTTP/TCP/IP, we employ a future Internet initiative called NovaGenesis. NovaGenesis is being deployed since 2008 and currently has a C++ prototype for Linux. The proposed model employs self-certifying names instead of HTTP hyperlinks. Instead of client/server model, we employ a publish/subscribe paradigm. Web page contents are stored into network caches, improving performance. We report a successful implementation of NovaGenesis web through laboratory results that validate its advantages in terms of efficiency, flexibility and security.

Keywords: Future Internet, NovaGenesis, Browser, Cache, Publish and Subscribe.

Capítulo 1. Introdução

Se a tecnologia da informação é hoje o que a eletricidade foi na Era Industrial, em nossa época a Internet poderia ser equiparada tanto a uma rede elétrica quanto ao motor elétrico, em razão de sua capacidade de distribuir a força da informação por todo o domínio da atividade humana. Ademais, à medida que novas tecnologias de geração e distribuição de energia tornaram possível a fábrica e a grande corporação como os fundamentos organizacionais da sociedade industrial, a Internet passou a ser base tecnológica para a forma organizacional da Era da Informação: a rede (Castells, 2003, p.7).

A Arpanet deu origem a Internet que conhecemos, ela iniciou como uma rede de computadores da *Advanced Research Projects Agency* (ARPA). No início, a intenção era permitir que vários computadores compartilhassem informações entre eles. Esta rede deveria ser descentralizada, flexível e capaz de sobreviver a ataques nucleares. Nesta época a rede se utilizava da comutação de pacotes, tecnologia que era vista como revolucionária na área de telecomunicações. A Figura 1 demonstra a Arpanet em 1977 com *Interface Message Processor* (IMP) Names no lugar de Hostnames.

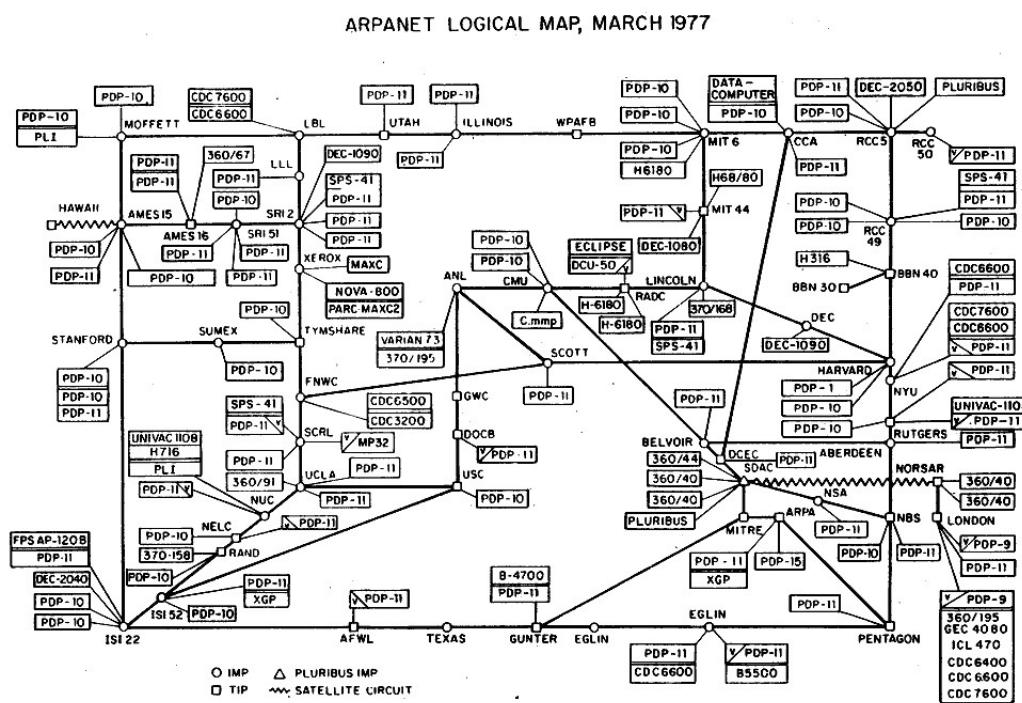


Figura 1 – Arpanet.

A Internet tomou forma quando a Arpanet começou a se expandir, várias redes começaram a se conectar e para que uma rede conversasse com outra era preciso padronizar os seus protocolos de comunicação, nascendo então o TCP/IP, o padrão que é utilizado até hoje. Muitos provedores aderiram à Internet e as forneciam mediante bases comerciais, isto permitiu que ela crescesse rapidamente [12].

A *World Wide Web* (WWW) é talvez a principal aplicação da Internet. Inventada por Sir Tim Berners-Lee em 1989, a *web* evoluiu muito desde a sua criação. Seu surgimento aconteceu devido as incompatibilidades das plataformas e ferramentas que existiam, tornavam impossível o acesso as informações existentes através de uma única interface. O *HyperText* foi a maneira encontrada para vincular e acessar informações de vários tipos, no qual o usuário poderia navegar em diversos conteúdos a procura da informação desejada através de diferentes associações de *links*, esta rede de *links* foi chamada de *web*. O navegador é o *software* cliente que permite acesso ao *HyperText* e quando se acessa o *link*, ele busca pelo *HyperText* associado [13]. Tanto o *HyperText* quanto o navegador foram sofrendo atualizações para suportar hoje a versão mais atual do *HyperText Markup Language* (HTML). A Figura 2 demonstra um exemplo da visualização em um browser atual de um HTML.

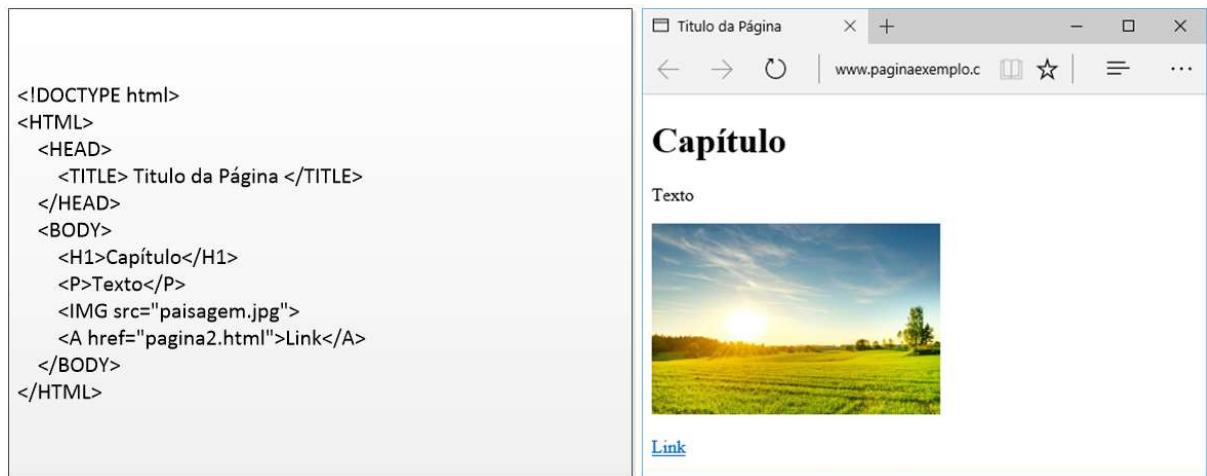
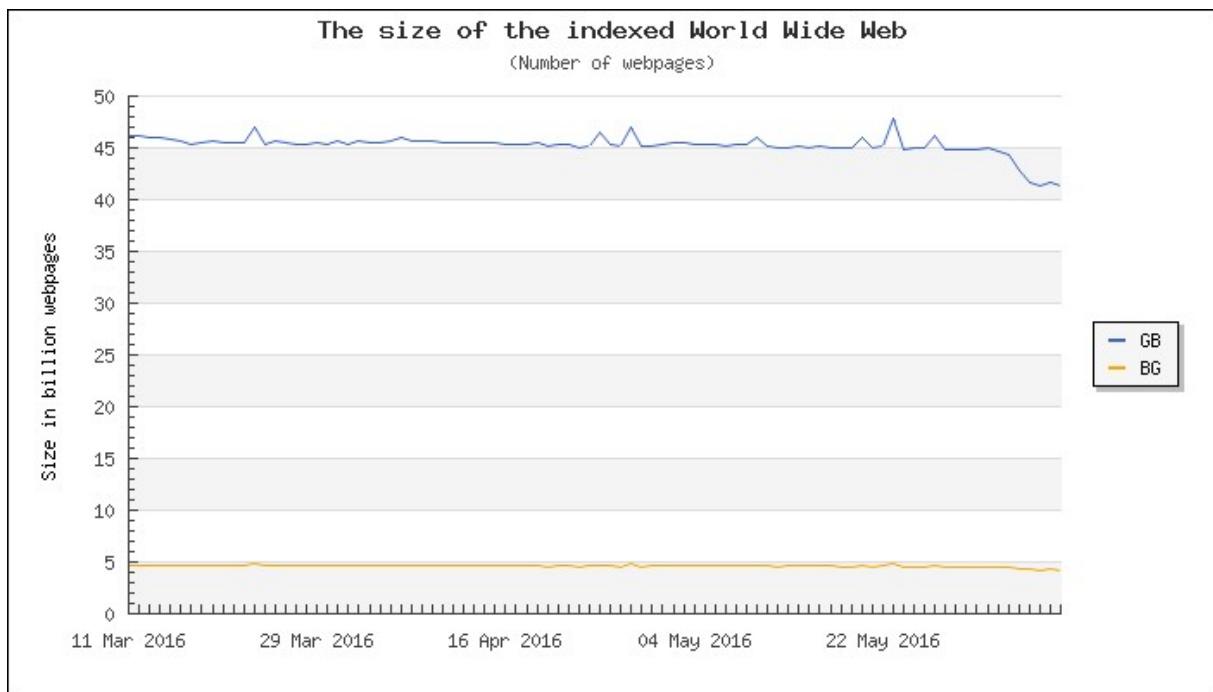


Figura 2 - Visualização em um browser (direita) de um HTML (esquerda).

A transferência de páginas em HTML entre dois computadores é feita através do protocolo *Hypertext Transfer Protocol* (HTTP). A proposta deste protocolo é ser leve, veloz, colaborativo, distributivo e funcional em sistemas de informação hipermídia. Protocolo de modelo cliente/servidor, considera cada transação como sendo independente (*stateless*) de outras anteriores. Também possui um conjunto de métodos que são usados para indicar o destino de um pedido, são eles [14]:

- *Uniform Resource Local* (URL) – A URL é utilizada para localizar um recurso na rede, pode-se usar barras para especificar pastas na localização do recurso. Em sua sintaxe, necessita especificar o protocolo que será utilizado na requisição e também características que forneçam acesso pelo protocolo [23].
- *Uniform Resource Identifier* (URI) – A URI é um método de acesso a objetos disponíveis na rede, a URL compõe parte de sua sintaxe, onde se especifica o protocolo de acesso e a localização deste objeto na rede [24].
- *Uniform Resource Name* (URN) – A URN é utilizada para obter um objeto através de um nome único, em sua sintaxe o nome não compõe a localização do objeto [25].

Com o aumento da Internet, a *web* cresce descontroladamente. Praticamente qualquer indivíduo ou instituição pode criar páginas com qualquer conteúdo ou *links*. A Figura 3 ilustra a quantidade de páginas indexadas em um período recente pelos motores de busca Google e Bing.



GB = Sorted on Google and Bing

BG = Sorted on Bing and Google

Figura 3 - Quantidade de páginas indexadas pelo Google e Bing.

Preocupados com o crescente aumento da *web*, algumas técnicas de armazenamento de conteúdo foram inventadas com o objetivo de reduzir o uso da rede e aumentar a taxa de

transmissão da web. Essas técnicas são chamadas de *cache*. Seu principal objetivo é armazenar uma cópia local para evitar a transferência futura durante uma requisição do mesmo objeto. Existem dois tipos de *cache*, os que são implementados no *browser* e os que são implementados em servidores fazendo intermediação durante uma requisição web. Estes servidores são chamados de servidores *proxy*. O servidor *proxy* quando intermedia uma requisição de um cliente *browser*, salva o objeto web para posteriormente entregá-lo durante requisições futuras vinda do mesmo ou de outros clientes *browser* [15]. Os dois tipos de *cache* são compatíveis e podem ser usados juntos, porém de forma hierárquica, em que o *cache* local tem maior prioridade. As Figuras 4 e 5 demonstram os cenários com os dois tipos de *cache*.

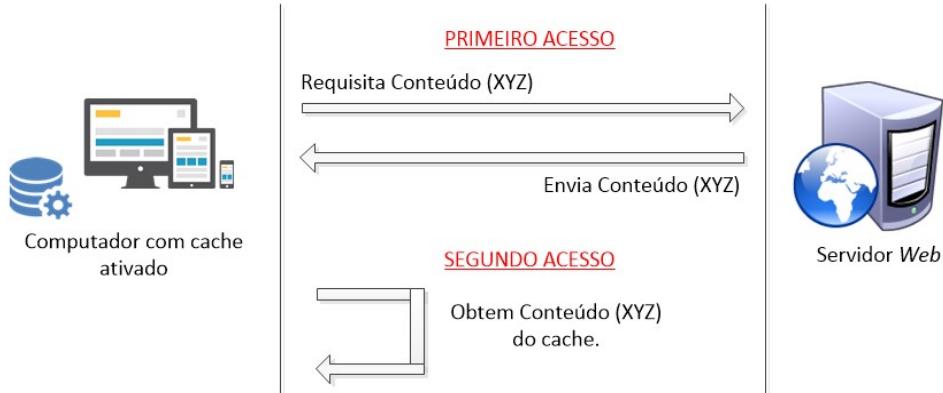


Figura 4 - Acesso ao cache pelo browser.

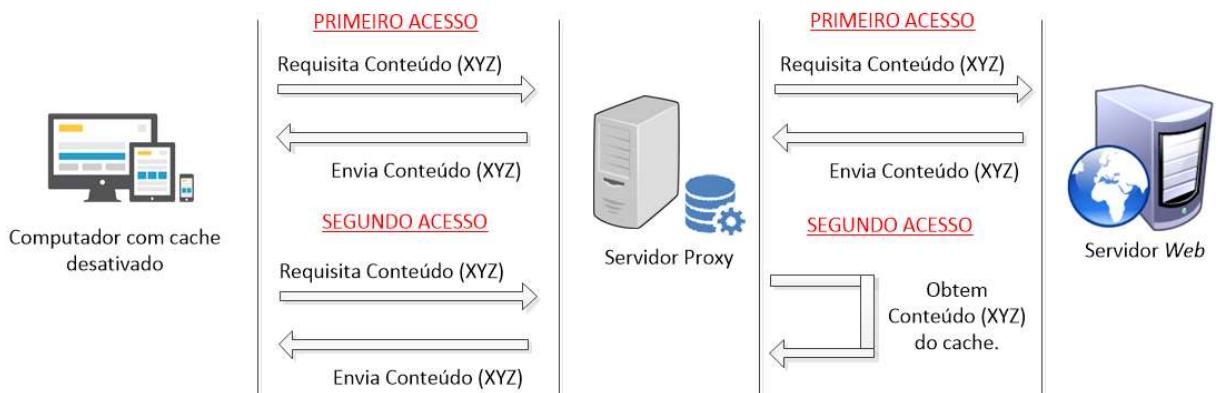


Figura 5 - Acesso ao cache via servidor proxy.

As técnicas de *cache* contribuíram de forma significativa para a melhoria da rede, pois a Internet tornou-se indispensável em nossa sociedade. Ela tem evoluído diariamente para

uma plataforma ampla e comercial. Diariamente, várias pessoas usufruem desta plataforma. A acessibilidade através de diversos recursos computacionais tem contribuído para um crescimento acelerado do número de dispositivos em rede. A Internet é um fenômeno social que vem mudando a forma como as pessoas se comunicam e de como fazemos negócio. Redefine como os computadores e seres humanos podem interagir. Praticamente, vários setores industriais tiram proveito disto, como grandes empresas de *software* (Microsoft, Google, SAP e etc.), empresas de manufatura (automobilística), empresas prestadoras de serviço (bancos e companhias de seguro) e empresas de entretenimento (Blizzard, Steam e etc.).

A Internet é hoje tão abrangente que afeta diversos tipos de pessoas e desta forma cada um a utiliza de forma de forma diferente, por exemplo, para alguns a Internet interessa as aplicações, como *web*, compartilhamento de arquivos, *chat*, voz sobre IP, mídias sociais, etc. Para outros são os protocolos de conexão, roteamento ou o tráfego e a qualidade da rede. Para outros consiste em elementos de rede, *hubs*, *switches*, *fibra*, etc. Ou para aqueles que se interessam no retorno financeiro que uma rede pode trazer [16], como as operadoras e outras empresas que dependem da rede para sobreviver. A Internet deve continuar em constante evolução para acomodar diversas inovações. Muitos desafios técnicos e não técnicos apareceram para suportar essas novas tecnologias e funcionalidades. Novas ideias e arquiteturas surgiram neste processo, dando origem a potenciais “novas arquiteturas de internet”. A Internet foi concebida há quase 40 anos para outras finalidades e propósitos. Entretanto, seu sucesso vem sendo prejudicado cada vez mais pelo seu atual design que contém, inúmeras falhas de segurança, mais e mais "remendos" são propostos e a complexidade do gerenciamento e a padronização do conjunto dos diversos protocolos suportados hoje tem sido outro desafio para quem usa ou implementa novas funcionalidades. Por isso, mediante a tantos desafios, mais e mais pesquisadores tem investigado novas arquiteturas de Internet a chamada Internet do futuro.

As arquiteturas de Internet do futuro não têm por objetivo solucionar um único cenário, topologia ou algo específico. São arquiteturas bem abrangentes, devem contemplar diversas tecnologias que são suportadas hoje e serem flexíveis o bastante para acomodar futuras ideias que possam surgir. Elas não devem ser rígidas, mas evolutivas, colaborativas, abrangentes, escaláveis, seguras, móveis, confiáveis, ágeis, etc [17]. A procura por novas arquiteturas gera muitas discussões na comunidade e foi assim que surgiram duas frentes de pesquisas onde diversos pesquisadores: (i) gostariam de evoluir a Internet atual; (ii) gostariam de começar uma nova arquitetura. Estas frentes de pesquisa foram chamadas de *Evolutionary*

e *Clean-Slate*, respectivamente. Também um “mix” destas pesquisas podem ser encontradas, embora ainda não se encontre um nome de referência para elas.

A pesquisa *Clean-Slate* é um grande desafio para os pesquisadores, suas arquiteturas precisam superar as atuais, mais que isso, precisa ser flexível o bastante para uma evolução constante, diversas análises e simulações precisam ser feitas para validar ideias inovadoras que promovam a nova arquitetura. Os protótipos são indispensáveis para este tipo de pesquisa. É quase impossível confirmar ideias plausíveis sem que tenham sido testadas devido à complexidade de uma rede e seus elementos. Algumas delas precisam ser testadas em condições reais ou em escala [18]. Alguns projetos surgiram para fornecer ao pesquisador infraestrutura para testes em escala, como: GENI, FIRE, OpenLab, FUTEBOL, etc.

Dentre diversas aplicações de rede, vale destacar a *web* que também sofre alterações por parte dos pesquisadores. Com a quantidade de páginas eletrônicas disponíveis e a facilidade com que uma entidade disponibiliza informação nelas, torna-se cada vez mais difícil localizar o conteúdo desejado [19]. Diversas páginas tentam fornecer meios de disponibilizar conteúdo específicos de forma agrupada para favorecer a navegação e a obtenção do conteúdo, ex.: Google, Uol, Yahoo, Bing, etc.

Ultimamente a *web* tem sido moldada bruscamente de seu objetivo inicial, em que um usuário estivesse interessado em encontrar um conteúdo qualquer, as empresas e usuários descobriram diversas oportunidades de negócios devido a abrangência da rede, diversas finalidades diferentes têm sido exploradas na *web* como o comércio eletrônico, *marketing*, mídias sociais, ensino a distância e banco eletrônico. Grandes empresas como Amazon, eBay, OLX, Mercado Livre, PagSeguro, Linkedin, Facebook, Google+, etc. Existem hoje, porque a *web* permitiu acomodar estas ideologias, a cada dia estas empresas vem crescendo e conquistando grande parte do mercado mundial. Desta forma, entendem-se as dificuldades de propor uma nova *web*, pois pesquisadores devem propor soluções que possam ser suportadas atualmente e ainda modelá-la de forma que possa acomodar ideias futuras.

Devido a diversidade e complexidade de propor novas soluções tanto para a Internet do futuro quanto para a *web* do futuro, pesquisadores tem adotado o modelo incremental de trabalho. No qual protótipos são construídos para validar essas ideias e então anexados junto ao projeto principal que contempla toda a proposta de Internet do futuro.

Neste trabalho, adotamos o modelo incremental de trabalho e reportamos os desafios e resultados obtidos na implementação de um modelo *Clean-Slate* para a navegação *web*. Esse modelo apoia-se sobre a arquitetura *Clean-Slate* chamada NovaGenesis [26]. Propomos e testamos um navegador *web* que segue os paradigmas da NovaGenesis, dentre eles: modelo

publica/assina com ciclo de vida integrado de serviços e conteúdos, espaços ilimitados de nomeação e resolução de nomes, nomeação auto-certificável, roteamento e encaminhamento de mensagens usando nomes, auto-certificáveis, armazenamento de conteúdos em *cache* de rede e operação baseada em contratos. Até onde sabemos não existe outro trabalho de *web* para Internet do futuro com essas características.

O modelo da *web* NovaGenesis consiste de: (i) um serviço que publica páginas *web* e seus conteúdos usando uma interface publica/assina distribuída – as páginas são armazenadas em *caches* de rede de forma distribuída usando estruturas de dados *hash table*; (ii) um *browser* usado para assinar páginas *web* e ligações entre nomes armazenados na rede. O navegador *web* é nomeado como *NGBrowser*, e é uma aplicação *Peer-to-Peer* (P2P) inserida no contexto da NovaGenesis, permitindo efetuar pesquisas e requisições de conteúdo, bem como páginas *web*, imagens, textos, entre outros objetos de informação. Tais objetos podem ter formatos binários, assim o usuário pode navegar pelo conteúdo com a mesma experiência da Internet atual. A Figura 6 ilustra a arquitetura desenvolvida neste trabalho e seus principais serviços e comunicações entre processos.

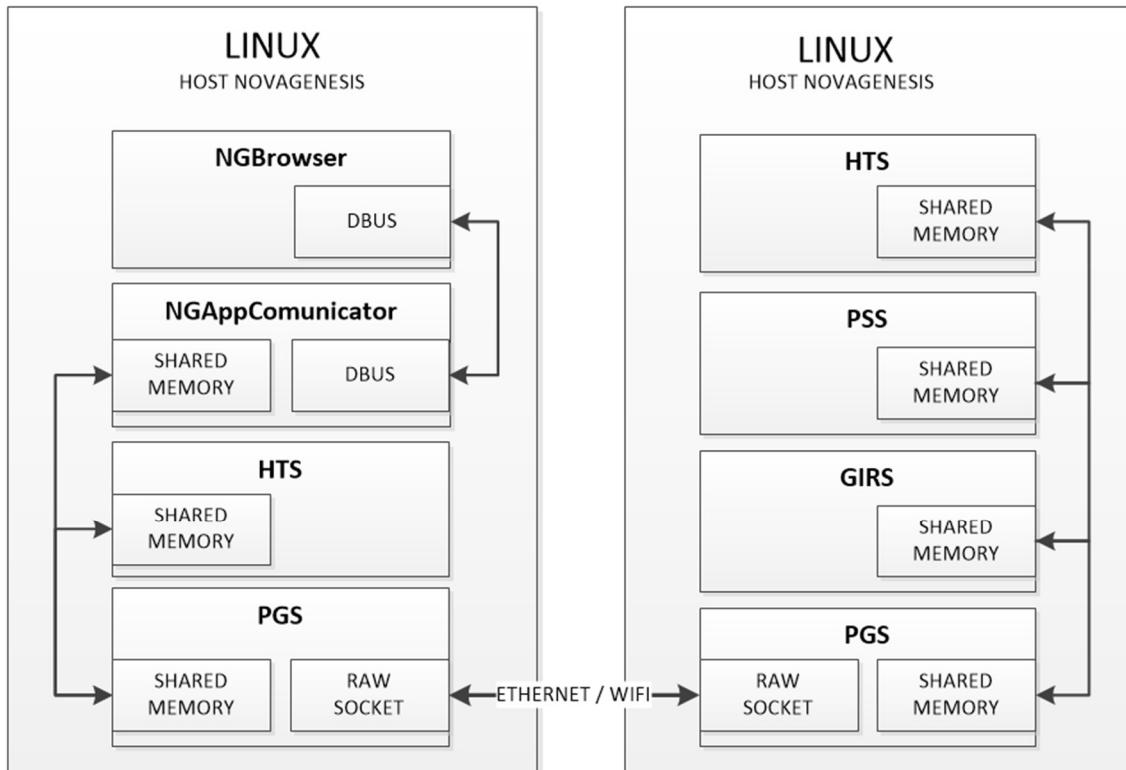


Figura 6 - Modelo da Arquitetura *NGBrowser* e *NovaGenesis*.

Diferente do modelo atual HTTP, a NovaGenesis utiliza o modelo publica/assina para envio e obtenção dos objetos *web*. O modelo publica/assina consiste em dois perfis: (i) publicador; (ii) assinante. No qual, o publicador consegue enviar qualquer conteúdo a vários assinantes sem especificá-los diretamente, basta que o assinante assine um tipo de "fila virtual" do publicador para receber o conteúdo dele. Os *hosts*, processos e conteúdo são todos nomeados por um código *hash* (sequência binária fixada gerada por um algoritmo baseado no conteúdo) em 128 *bits* e relacionados através de um grafo de ligações entre nomes que é armazenado de forma distribuída na rede. O código *hash* foi baseado no algoritmo *MurmurHash* por ser simples, ter alta resistência a colisão e alto desempenho[27].

Para a navegação *web* na arquitetura NovaGenesis foram desenvolvidas também ferramentas para adequação de páginas *web* já existentes para o novo modelo, chamadas de *NGConverter* e *NGAppPublisher*. Seus papéis são respectivamente os de transformar um *site web* comum em um formato de *site web* NovaGenesis e publicá-los a rede para que o *site* seja posteriormente acessado pelo *NGBrowser* através de um serviço intermediário chamado *NGAppComunicator*. A Figura 7 ilustra como esses componentes interagem entre si.

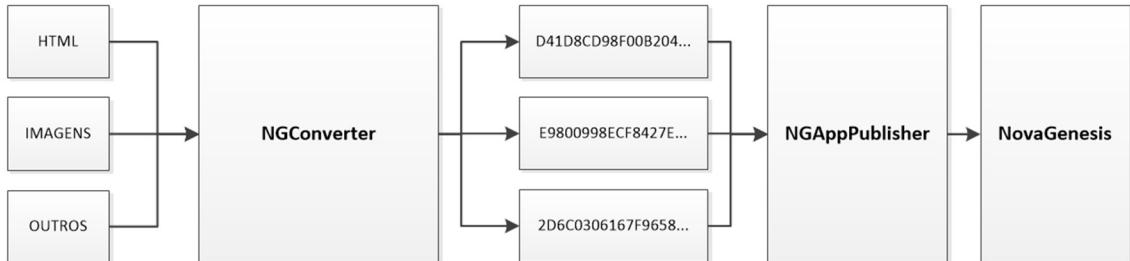


Figura 7 - Fluxo de entrada e saída da conversão e publicação de sites.

É esperado que o *NGBrowser* seja mais rápido e seguro que um navegador comum e que também faça uma economia de taxa de *bits*. No modelo proposto é possível obter um *hash* de qualquer conteúdo antes que seja efetuado o *download* do conteúdo. Desta forma, é possível se certificar de que o conteúdo solicitado é o mesmo que se encontra no *cache*, assim, os sistemas de *cache* serão mais eficientes descartando a necessidade de realizar um *download* de um arquivo já existente. Embora isso já exista na navegação *web* atual, na NovaGenesis o modelo publica/assina utiliza nomes auto-certificáveis que consiste em um nome único baseado na informação fornecida, desta forma é possível averiguar se o nome único tem procedência de uma determinada informação, porém de posse do nome único, não é possível saber a informação que foi utilizada para obtê-la, o que aumenta a segurança no

acesso aos objetos de informação [7], a utilização desta técnica permite o navegador efetuar o *hash* do conteúdo e se certificar de que o conteúdo obtido foi o solicitado. O *NGBrowser* prova a possibilidade de uma navegação em uma Internet do futuro através do modelo publica/assina, podendo obter mais vantagens como a de possuir um *cache* mais eficiente e uma navegação mais segura do que o HTTP da Internet atual.

O restante deste trabalho é organizado como segue. No Capítulo 2 é feita uma descrição da arquitetura NovaGenesis e os seus recursos para desenvolver um novo modelo de *web*. No Capítulo 3 apresentamos as técnicas de Armazenamento de Conteúdo existentes na Internet atual e a proposta deste trabalho. No Capítulo 4 apresentamos os componentes utilizados na implementação do novo modelo *web* proposto, componentes construídos para a navegação *web* com nomeação auto-certificável e comunicação via publica/assina. No Capítulo 5 reportamos os resultados dos testes feitos no ICT Lab. do Inatel. Por fim, no Capítulo 6 concluímos o trabalho.

Capítulo 2. NovaGenesis

A NovaGenesis é uma arquitetura convergente que integra a troca, processamento e armazenamento de informações. Pode ser vista como uma arquitetura de Internet do futuro integrada a computação em nuvem [26]. O projeto surgiu em 2008, mas foi em 2012 que houve a implementação de uma primeira prova de conceito. A NovaGenesis visa a integração dos princípios fundamentais de *design* que estão sendo considerados na Internet do futuro em um modelo único, convergente e auto similar à níveis de processos de um *host*, agregados deles ou em escalas maiores, por ex.: a publicação de conteúdos acontece entre processos, o mesmo entre *hosts* e também entre domínios. A Figura 8 demonstra os princípios adotados para incorporar a arquitetura NovaGenesis.

Princípios de Design	Escolha de Design
Selecionar ingredientes que favoreçam a flexibilidade e a generalidade.	Assine com nomes auto certificáveis para entidades.
Generalize a aplicação dos ingredientes selecionados o máximo possível.	Virtualização de tecnologias
Design para construção de uma composição dinâmica, modularidade hierárquica e recursividade.	Rede definida por software
Design para uma construção em segurança, privacidade, confiança e rastreabilidade.	Design de serviço orientado
Design para construção de uma mobilidade, evolução, sustentabilidade e um ecossistema de negócios.	Design orientado ao conteúdo.
Virtualizando o mundo real e integrando em os mundos virtuais.	Relacione nomes e objetos interativamente
Redução da intervenção humana, sustentando a ciência do conhecimento e facilitando a migração	Publica/Assina nomes e objetos.
Aja de acordo com o significado/contexto de informação/contexto.	Tabelas hash distribuídas.
Integre os ingredientes de forma coesa e sinérgica.	Ciclo de vida de entidades.
Faça um design "mais simples possível", mas não o mais simples, Einstein.	Orquestração de serviços.
	Auto organização de baixo para cima.
	Software fractal.
	Cognitivo e autonômico.

Figura 8 – Princípios de design da NovaGenesis.

A NovaGenesis adota vários paradigmas como o de auto-organização de baixo para cima (de pequenos programas para grandes sistemas distribuídos, isto significa que cada serviço pode cooperar com outros para fornecer um novo serviço que não precisa estar no mesmo *host*), exposição de recursos (permite fazer serviços possam anunciar funcionalidades), nomeação de serviços e conteúdos (com a nomeação é possível expô-los), virtualização de substrato (representação por *software* de alguns recursos físicos), *Self-Certifying Name* (representação por um *hash* de algum conteúdo obtido através de um

algoritmo que efetua o cálculo com os *bits* do conteúdo em questão), modelo de comunicação publica/assina, ciclo de vida de entidades física e virtuais, sejam processos, *hosts* ou objetos de informação (isto significa que as entidades não são permanentes no sistema, é normal que elas sejam criadas para uma finalidade e depois destruídas).

A NovaGenesis não implementa qualquer protocolo TCP/IP ou aplicações de rede conhecidas, baseando-se atualmente em quatro processos principais: HTS, PSS, GIRS e PGS. O *hash table service* (HTS) é um resolvedor distribuído de nomes e *cache* de rede, capaz de resolver nomes entre qualquer *namespace* usado e armazenar ligações entre nomes, bem como conteúdos em tabelas *hash*. O *generic indirection resolution service* (GIRS) é responsável por escolher o HTS que irá armazenar uma dada ligação entre nomes, armazenando o endereço dos HTS que guardam determinados pares < chave, valor(es) >. O *publish/subscribe service* (PSS) implementa uma interface publica/assina distribuída usada pelos demais serviços NovaGenesis para disponibilizar ligações entre nomes e conteúdos na rede [1], [2]. Por fim, o *proxy/gateway service* (PGS) encapsula mensagens NovaGenesis diretamente sobre um *Network Socket* (NS) do tipo *Raw* que recebe e envia datagramas que não incluem cabeçalhos de *link* [6], ou seja, não é necessário utilizar quaisquer protocolos da camada de rede já existentes. Tipicamente, as mensagens NovaGenesis são encapsuladas diretamente sobre tecnologias de enlace (tais com Ethernet e Wi-Fi). Porém, a versão corrente da arquitetura ainda carece de um protocolo de transporte como o TCP. A Figura 9 ilustra a arquitetura NovaGenesis para rede local.

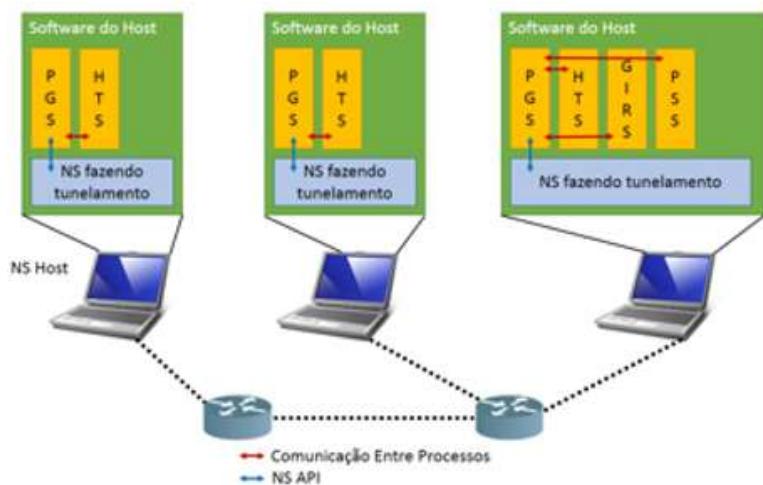


Figura 9 – Arquitetura NovaGenesis ilustrada para uma rede local.

A NovaGenesis traz requisitos que são importantes para a evolução da rede e suas aplicações. A mobilidade é um caso que a NovaGenesis trata em todas as escalas para todas as entidades, no entanto a rede atual tem trazido grandes dificuldades de elaboração e implementação por parte dos pesquisadores. Este tema é debatido até hoje com o objetivo de alcançar uma solução que seja plausível, pois algumas das ideias podem ser vistas como radicais. Por exemplo, uma das soluções propõe a independência das camadas do modelo OSI, pois segundo o pesquisador este seria o primeiro passo para permitir a mobilidade nas camadas superiores [28].

O conceito de redes centradas em informações é uma abordagem comum em pesquisas de Internet do futuro. A NovaGenesis busca complementar diversos elementos em seu projeto com a mesma abordagem utilizada em redes centradas em informação como *cache* em rede, comunicação multipartidária através da replicação e desacoplamento dos modelos de interação entre emissores e receptores, com o objetivo de fornecer um serviço de infraestrutura de rede adequado, resistente e robusto [1][29].

Tecnologias autonômicas e cognitivas são empregadas também em redes afim de melhorar alguns aspectos da arquitetura atual. Uma das propostas emprega estas tecnologias para tornar pacotes comuns em pacotes cognitivos. Em que roteamentos inteligentes são feitos por pacotes cognitivos afim de melhorar a confiabilidade, segurança, escalabilidade e Qualidade de Serviço (QoS) [30]. Nas pesquisas de Internet do futuro as tecnologias autonômicas e cognitivas também são empregadas. Na NovaGenesis essas tecnologias foram utilizadas em um serviço chamado de *Resource Management Agent* (RMA) que permite controlar e decidir o que deve ser feito baseado em informações geradas por serviços e em comunicações entre serviços "atuadores". O RMA se torna indispensável para auxiliar na operação de forma autônoma e integra diversos tipos de equipamentos em um ambiente *Internet of Things* (IoT). Este serviço é "puramente NovaGenesis" e não possui interface direta com qualquer artifício ou elemento externo a NovaGenesis. O objetivo de adotar essas tecnologias seria para reduzir a intervenção humana e os custos operacionais [31].

Outros requisitos devem ser levados em consideração como segurança, privacidade e confiança. A NovaGenesis utiliza do publica/assina para estabelecer uma comunicação entre diferentes serviços previamente autenticados e autorizados via um serviço intermediário, chamado de *Service Level Agreement* (SLA). A troca de conteúdo só acontece depois que o SLA esteja estabelecido. Com o contrato estabelecido, os serviços utilizam de uma criptografia assimétrica para a passagem de mensagens. Assim, tais processos garantem a troca de informações segura entre serviços NovaGenesis [32].

Algumas abordagens de IoT propõem um *framework* de virtualização para representar sensores reais em serviços *web* processando e raciocinando eventos provendo uma organização semântica formulando uma nuvem IoT com o objetivo de facilitar a monitoração de eventos, a procura por determinados sensores dentro de um contexto e a organização da rede [33]. A NovaGenesis aborda a IoT e *Real-World Internet* (RWI) com técnicas de virtualização que consiste em expor as funcionalidades reais do dispositivo para o *software*, assim torna possível a integração dos recursos do mundo real com os serviços virtuais. Isto permite a integração das abordagens de IoT citadas e também a interação com outros serviços cognitivos ou autônomos [1].

2.1. Principais Componentes da Implementação Atual

2.1.1. Hash Table Service (HTS)

O serviço de tabela *hash* é o processo responsável por armazenar (em *cache*) conteúdos, informações sobre *hosts*, processos ou relacionamentos entre eles. Os relacionamentos são efetuados através de uma chave associada a vários valores, onde as chaves ou valores assumem uma sequência em *bits* no formato de *String* (sequência de caracteres terminado com '\0'). Cada valor pode ser associado a outras chaves que por sua vez possuem outros valores. Desta forma os relacionamentos podem ser comparados a grafos cíclicos. Os arquivos também podem fazer parte dessa associação, de forma indireta, na qual um arquivo é representado por um SCN obtido através do *hash* do próprio arquivo e o valor é o caminho onde o arquivo é armazenado no sistema operacional.

2.1.2. Generic Indirection Resolution Service (GIRS)

O serviço de resolução de indireções genéricas é um processo que seleciona qual HTS irá guardar uma ligação entre nomes no formato < chave, valor(es) >, tal como descrito na subseção anterior. Seu papel é escolher a instância de HTS para cada par chave/valor, balanceando a carga, distribuindo igualmente as informações disponíveis entre as *hash tables*.

2.1.3. Publish/Subscribe Service (PSS)

O serviço de publica/assina é responsável por gerar permissões de acesso a ligações entre nomes publicadas. Portanto, ele faz o *rendezvous* entre serviços que publicam e assinam pares chave/valor. Todos os processos NovaGenesis devem utilizar o PSS para publicar/assinar qualquer conteúdo e/ou ligações entre nomes.

2.1.4. Proxy/Gateway Service (PGS)

O serviço *proxy/gateway* é o processo responsável por estabelecer a comunicação entre *hosts*. Cada PGS é executado em um *host* e efetua o protocolo de descobrimento da NovaGenesis para mapear e rotear dados entre eles utilizando as tecnologias de enlace.

2.5. NovaGenesis Web

A *web* do futuro ainda é discutida até hoje, não se sabe exatamente quais são as ideias que vão fazer parte da nova Internet, isto porque a evolução continua e rápida da *web* atual dificulta chegar em uma conclusão de como as coisas devem ser. Mais e mais tecnologias são associadas a *web*, como *e-commerce*, mídias sociais, vídeo sob demanda, *web tv*, bancos virtuais, etc. Mas, alguns pesquisadores dizem que a *web* deve ser organizada de forma a facilitar e agilizar a navegação por humanos e máquinas. Para isso técnicas de ontologia (consiste em um modelo de relacionamento entre diversos conceitos em um dado domínio, por ex. cadeira faz parte da mesa de jantar, mas a cadeira não faz parte do telhado), semântica (define uma linguagem, uma forma de comunicação) e reconhecimento de imagens (reconhece uma imagem através da busca por suas características) são empregadas [34]. Espera-se que através da análise de dados *web*, máquinas poderão nos ajudar a resolver problemas [35]. Também se espera que a *web* do futuro seja cada vez mais segura, o suficiente para fazer transações bancárias em alto volume ou consultar seus sensores e dados pessoais que estejam em nuvem sem se preocupar com intrusos ou a confiabilidade do local de acesso. Ser cada vez mais interativa através de outras interfaces além do teclado e mouse. Menos consumo de taxa em *bits* para realizar mesmas tarefas que são feitas hoje. Altamente compatível a vários tipos de dispositivos ou equipamentos para exibição do mesmo conteúdo.

Conter rastreabilidade de acesso para fins da justiça, por exemplo. Possuir uma infraestrutura ágil, robusta e de qualidade para suportar o crescimento exponencial de seu conteúdo.

O projeto NovaGenesis entende a necessidade de construir uma estrutura *web* abrangente, evolucionária, que suporte às tecnologias atuais e atenda as novas propostas *web*. O primeiro protótipo da NovaGenesis *web* é baseado neste trabalho e aborda dentro do contexto *web* o modelo de navegação e a exposição, pesquisa e obtenção do objeto *web*. Um objeto *web* pode ser uma imagem, um *javascript*, *stylesheet*, texto, HTML ou qualquer outro formato de arquivo. A rede possui meios de requerer ou enviar esses objetos através dos modelos de publica/assina. Cada objeto possui um SCN e pode ser associado a palavras-chaves, desta forma é possível requerer este objeto por SCN ou palavra-chave associada. Caso exista mais de um objeto associado a mesma palavra-chave, uma lista de objetos relacionados é entregue.

A Figura 10 ilustra o cenário da publicação de dois objetos *web* distintos (SCN diferentes) com uma lista de associações de palavras-chave em que uma delas é idêntica. Em seguida, a assinatura de um objeto *web* é feita através de seu SCN, como consequência o objeto *web* é retornado. Novamente uma assinatura é realizada, mas agora utilizando da palavra-chave “faculdade”, da qual ela está relacionada a dois objetos distintos. Como consequência uma lista de dois objetos é retornada.

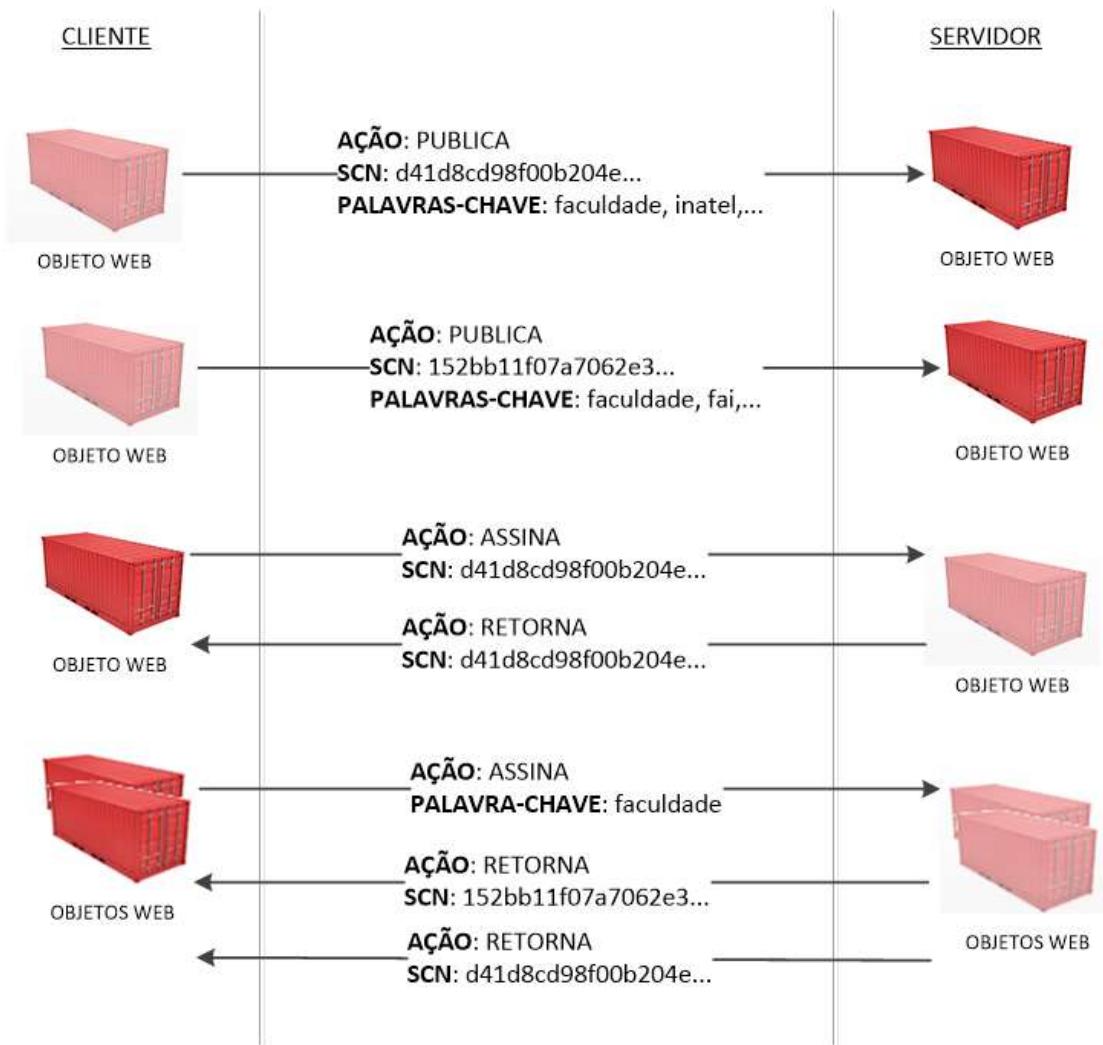


Figura 10 - Cenário de publica/assina de objetos web.

A NovaGenesis *web* se baseia nessas funcionalidades para realizar a navegação e a exposição, pesquisa e obtenção do objeto *web*. A navegação consiste da obtenção de objetos *web* através da assinatura de SCN's. A pesquisa da assinatura de palavras-chave que levam a uma lista de objetos *web* que são disponibilizados ao usuário. A exposição se baseia na publicação de objetos *web* e associação deles a palavras-chave.

Capítulo 3. Armazenamento de Conteúdo

Com o crescente e avanço da Internet, pesquisadores tem se esforçado em procurar por novas técnicas para diminuir o consumo de banda, baixar a latência da rede e agilizar o tempo de resposta do objeto *web* em questão [36]. Alguns modelos desenvolvidos se resumem a um procedimento chamado de *cache*, que consiste em ter uma cópia do objeto *web* (arquivos no formato de fotos, páginas, *javascript*, *stylesheet*, etc.) armazenado no computador local ou servidores *proxies* evitando a necessidade de novas requisições e uso da banda [37]. Os modelos utilizados hoje são apresentados a seguir, mas ainda possuem algumas desvantagens em comparação com este estudo.

3.1 Modelos de Armazenamento da Internet

O protocolo HTTP/1.1 adota o modelo de “Tempo de expiração” para reduzir a quantidade de requisições nos servidores. Já o modelo de “Validação” é utilizado para reduzir o uso de banda entre cliente e servidor. Ambos os modelos requerem transparência entre as pontas (que os *hosts* devem responder as requisições solicitadas e ainda de forma verídica) para que seja possível realizar a comunicação entre eles, a comunicação é essencial para que seja possível fazer as operações entre eles de ganho de desempenho, disponibilidade e desconexão. A não transparência pode gerar incompatibilidade entre diferentes aplicações e servidores. Portanto, o protocolo julga três elementos básicos importantes que devem ser implementados. São eles: (i) O protocolo deve prover total transparência quando requerido por qualquer uma das partes; (ii) demonstrar de forma clara as requisições do cliente e do servidor; (iii) controlar as operações que não são transparentes e permitir anexar alertas as respostas quanto aos pedidos de transparência.

Pode-se dizer que a negociação do *cache* ocorreu bem quando algumas destas condições é satisfeita: a revalidação do *cache* é confirmada pelo mesmo servidor de origem, atende as exigências de expiração tanto do servidor quanto dos clientes, utilizam de mensagens apropriadas (status 304, 305, 4xx ou 5xx), informam com alertas apropriados no cabeçalho ou responde com “conteúdo proibido” para refazer a negociação caso a veracidade e consistência do *cache* sejam duvidosos [20].

3.1.1 Tempo de expiração

Este modelo consiste em armazenar um objeto *web* em disco local por um determinado tempo, enquanto este tempo não é atingido, o objeto é utilizado nas requisições. Caso contrário, um novo objeto será requisitado durante o processo de consulta do *cache*. Este modelo apresenta melhor desempenho quando não necessita requisitar o servidor.

O tempo de expiração de um objeto *web* é estabelecido pelo aplicativo que o implementa, ex.: navegadores ou servidores *proxies*. Este tempo é sempre definido para o futuro, a menos que se deseje que o objeto em questão não seja armazenado em *cache*. Deve-se escolher bem os tempos para um melhor aproveitamento deste modelo, pois espera-se que durante o tempo de expiração o objeto não seja alterado. As recomendações são para que seja definido um tempo de até um ano de expiração. Isto pode causar problemas para arquivos textos utilizados em páginas, como *javascript* e *stylesheets*. Para eles deve-se associar em seus nomes, o *hash* de seu conteúdo, indicando um versionamento dele próprio. Desta forma, quando houver novas versões (novos *hashes*) do arquivo, o aplicativo será forçado a fazer novas requisições do objeto [22].

Um desenvolvedor de páginas *web* pode estabelecer o tempo de expiração para os objetos *web* através da meta tag *Cache-Control*, mas não é obrigatório que o aplicativo o obedeça, nem o servidor de página. Quando aplicado, o servidor de página responde com um cabeçalho “*max-age*”, informando o tempo em segundos para a expiração. A Figura 11 demonstra o funcionamento do modelo tempo de expiração quando o usuário requisita duas vezes o mesmo objeto *web*.

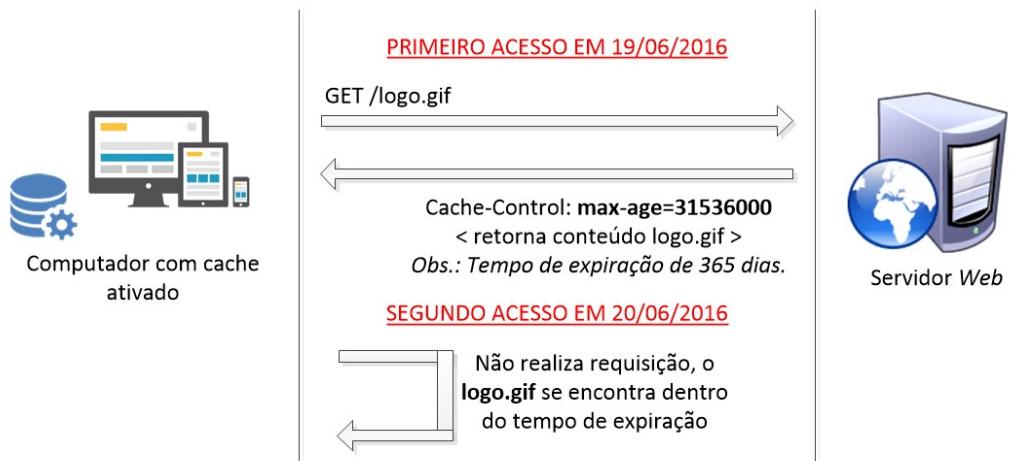


Figura 11 - Demonstração do modelo de cache tempo de expiração.

3.1.2 Validação

Este modelo busca validar se são idênticos os objetos *web* armazenado em *cache* com o objeto *web* armazenado no servidor remoto através de mensagens. Caso sejam idênticos, o objeto armazenado em *cache* é utilizado. Caso contrário o novo objeto é obtido.

O modelo de validação utiliza dos cabeçalhos do protocolo HTTP para trafegar informações que irão validar os objetos *web* armazenados. O cabeçalho “*ETag*” contém uma chave única que diz respeito a assinatura digital do arquivo. Caso um *bit* do arquivo seja alterado, uma nova chave é obtida. O cliente não precisa saber como esta chave é gerada, ele só precisa enviá-la durante a requisição do objeto. O servidor compara as chaves (local e remota) e retorna o status 304 (Não modificado) ou o novo objeto, caso as chaves sejam divergentes.

O envio da chave é realizado através do cabeçalho “*If-None-Match*” por parte do cliente. Não é obrigatório enviar este cabeçalho, mas se não for enviado, o objeto requisitado é retornado pelo servidor juntamente com o cabeçalho “*ETag*” [21]. A Figura 12 demonstra o fluxo de mensagens na primeira requisição de um objeto *web* qualquer. O objeto *web* é retornado neste ponto. Já a Figura 13 demonstra o fluxo de mensagens da segunda requisição do mesmo objeto *web*, desta vez a chave correspondente ao arquivo recebido anteriormente é enviada na requisição. O servidor o compara e sendo idêntica, não retorna o objeto *web*, somente o status 304.

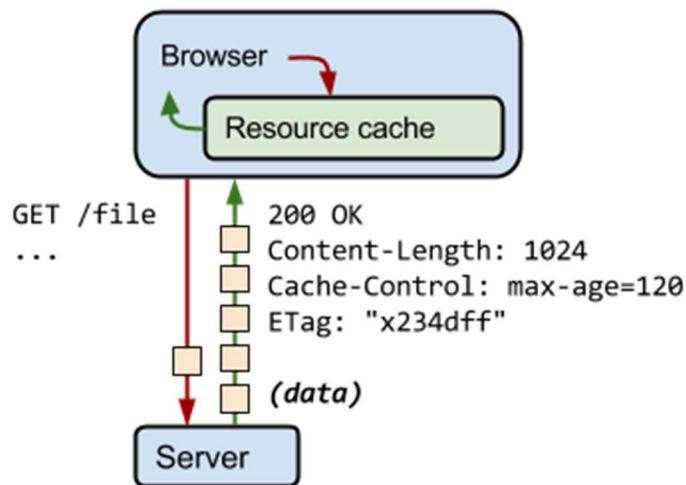


Figura 12 - Fluxo de mensagens na primeira requisição de um objeto web.

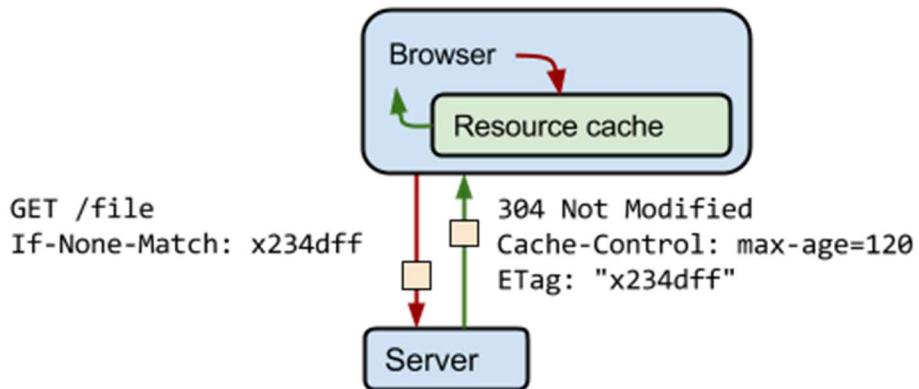


Figura 13 - Fluxo de mensagens da segunda requisição do objeto web da Figura 12.

3.2 Modelo de Armazenamento do Navegador NovaGenesis

Este modelo de armazenamento foi pensado para ser simples e requer que seja implementado somente no cliente. Seu ganho de desempenho beneficia tanto o cliente quanto o servidor, pois não necessita do *download* do objeto *web*, economizando o uso do *link* e também descartando a necessidade de validar o objeto junto ao servidor quanto sua veracidade ou variação. O desenho da NovaGenesis contribuiu para a eficiência deste modelo de armazenamento. O SCN permitiu a busca por objetos únicos e ajudou nos fatores de veracidade e variação, pois o objeto obtido é recertificado através do cálculo de seu SCN e comparado com o SCN requisitado. Cada objeto possui seu SCN. O SCN muda para qualquer alteração de *bit* no objeto. Portanto a variação é sempre tratada como um novo objeto e assim minimiza o erro em utilizar um objeto com conteúdo ultrapassado. Os objetos não expiram, pois espera-se que eles sejam imutáveis, a menos que sejam apagados pelo aplicativo para liberar espaço no disco local.

O sistema de *cache* funciona da seguinte forma: No momento que cada requisição é gerada, o aplicativo deve checar se existe o arquivo no disco local com o nome do SCN requisitante. Caso exista, o arquivo é utilizado na requisição como resposta. Caso não exista, a requisição prossegue e o objeto é obtido através da rede. A Figura 14 ilustra o funcionamento do *cache* do navegador NovaGenesis quanto o acesso a rede. No primeiro acesso ao objeto *web* de SCN igual a “adda24ad9c...”, o navegador efetua um pedido de assinatura do arquivo a rede NovaGenesis, que por sua vez o servidor o entrega. No segundo acesso, o navegador deixa de efetuar um pedido de assinatura do arquivo, pois o encontra em disco local.

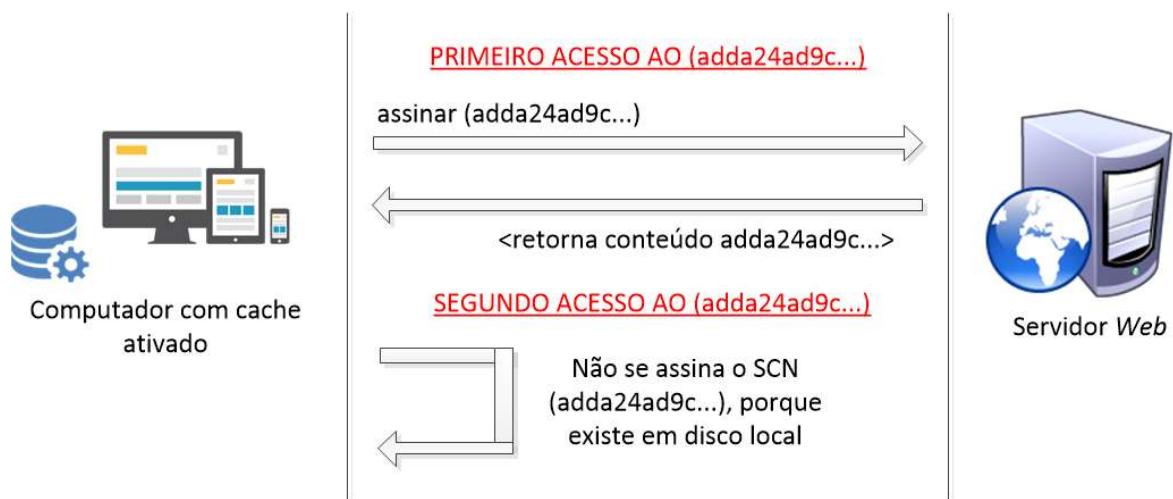


Figura 14 - Demonstração do modelo de armazenamento do navegador NovaGenesis.

Capítulo 4. Implementação da NovaGenesis Web

Para uma navegação *web* mínima, considere que há uma comunicação ponto-a-ponto entre dois *hosts* diferentes. O cliente requisita o conteúdo desejado a um servidor e o servidor fornece o conteúdo ao cliente, que por sua vez exibe o conteúdo requisitado. Fazendo uma comparação com a Internet atual e a NovaGenesis, a Internet atual precisa dos seguintes componentes para esse cenário: servidor DNS para resolução de nomes em IPs, servidor de página (hospedagem), protocolo de comunicação entre o cliente e servidor (HTTP) para efetuar a entrega do conteúdo solicitado sobre o TCP/IP [3]. Já a NovaGenesis utiliza dos processos PSS/GIRS/HTS para realizar as tarefas que seriam do DNS e servidor de página. O modelo publica/assina substitui o protocolo de comunicação (HTTP) para transferência de conteúdo. Na Internet atual, uma página *web* comum utiliza da linguagem de programação HTML para criar *sites* interativos que podem ser interpretados por navegadores [4]. A linguagem de programação HTML não é fortemente ligada ao HTTP ou TCP, portanto ela foi adotada neste trabalho para ser utilizada na NovaGenesis da mesma forma que é utilizada na Internet atual. Porém, os URIs, utilizados para encontrar um recurso na Internet, são modificados para acomodar a arquitetura de rede adotada.

4.1. Componentes de implementação

Os componentes de implementação dizem respeito a *framework* (ou conjunto de bibliotecas, que seriam um aglomerado de algoritmos no intuito de disponibilizar ao desenvolvedor de *software* meio práticos para resolver problemas complexos ou trabalhosos) e quais módulos foram utilizados para agilizar o desenvolvimento deste trabalho.

O Qt é um *framework* de desenvolvimento de aplicações multi-plataforma para *Desktop*, *Embedded* e *Mobile*. Atualmente, é suportado nas plataformas Linux, OS X, Windows, VxWorks, QNX, Android, iOS, BlackBerry, Sailfish OS e outros [8]. Escrito em C++ é utilizado como extensão a linguagem com recursos como *signals/slots*, também possui diversas APIs que facilitam o desenvolvimento de aplicações de forma abrangente, intuitiva e modularizada [9]. Neste trabalho foram utilizados os módulos QtWebKit e QtDBus.

4.1.1. QtWebKit

QtWebKit é um módulo de renderização de conteúdo *web* baseado no projeto *open source* WebKit que possui um amplo suporte para tecnologias padrão da *web* [11]. O WebKit pode ser encontrado em vários dispositivos como computadores, tablets e celulares. Com mais de 12 anos de história, este módulo possui uma comunidade imensa, colaborativa e com o apoio de grandes empresas como Google, Samsung, Nokia, RIM, HTC e outros. Com ele é possível tratar diversos eventos internos que um navegador comum pode possuir, como *request*, eventos de cliques, manipulação de conteúdo, entre outros. Este projeto se torna opção interessante para os projetos de Internet do futuro que desejam utilizar o HTML como linguagem *web*, devido sua flexibilidade e facilidade para aqueles que pretendem efetuar alterações significativas na camada de rede e requisições. Por este motivo este trabalho adotou este módulo para renderizar as páginas *web* da proposta NovaGenesis.

4.1.2 D-Bus

D-Bus é um mecanismo de *Inter-Process Communication* (IPC) e *Remote Procedure Calling* (RPC), originalmente desenvolvido para Linux para substituir e competir com soluções existentes de IPC utilizando um protocolo unificado. Ele também foi projetado para permitir a comunicação entre processos a nível de sistema e de usuários. Utiliza um rápido protocolo de passagem de mensagens binária, ideal para comunicação na mesma máquina devido à sua baixa latência e *overhead* [10].

4.1.2.1 QtDBus

QtDBus é um módulo que exporta as funcionalidades do D-Bus de forma amigável, ele encapsula completamente o conceito de baixo nível de mensagens em uma abordagem mais simples, orientada a objetos, familiar aos desenvolvedores do Qt [10]. Na maioria dos casos, o desenvolvedor não precisa se preocupar com o envio ou recebimento de mensagens. O QtDBus implementa o conceito de *Interfaces* e *Adaptors*. Baseia-se em objetos criados em tempo de execução que estabelecem conexões entre chamadores (*Interfaces*) e chamados (*Adaptors*) via D-Bus.

4.2. Componentes da *web*

Os componentes da *web* dizem respeito aos *softwares* que foram desenvolvidos para que fosse possível a navegação *web* pelo usuário através da rede NovaGenesis. Nem todos os *softwares* são utilizados simultaneamente, alguns deles são utilizados na etapa da preparação do ambiente *web*. Uma vez utilizados eles não são mais necessários. Os processos de criação do ambiente e navegação *web* foram quebrados em componentes com o intuito de reduzir a complexidade de cada um deles, bem como para possuir papéis claros. Também o reuso foi estudado para que estes componentes *web* pudessem ser utilizados para projetos futuros ou semelhantes.

4.2.1. NGConverter

O *NGConverter* é uma ferramenta utilizada para converter um *site web* utilizado na Internet atual em uma página para ser utilizado na NovaGenesis. Esta conversão se resume em cinco etapas que são executadas em ordem conforme ilustra a Figura 15. Cada página *web* tem um descriptor que é extraído da página e formatado em JSON, conforme ilustra a Figura 16. O principal objetivo do *NGConverter* é preparar o *site* de tal forma que simplifique o método de publicação executado pelo *NGAppPublisher*, discutindo logo em seguida. No final da execução desta ferramenta, é esperado que todos os arquivos do *site* utilizem apenas SCNs, bem como que todo o conteúdo de seus arquivos dependentes também esteja nomeado de forma auto-certificável. Ainda, é gerada uma lista de palavras chaves de cada página *web*. Todos os arquivos são convertidos e armazenados em uma única pasta, mesmo que um *site* contenha subpastas. Também não existem arquivos com nomes diferentes para um mesmo conteúdo, somente um SCN é utilizado em diferentes *sites*, uma tremenda vantagem de se usar os SCNs.

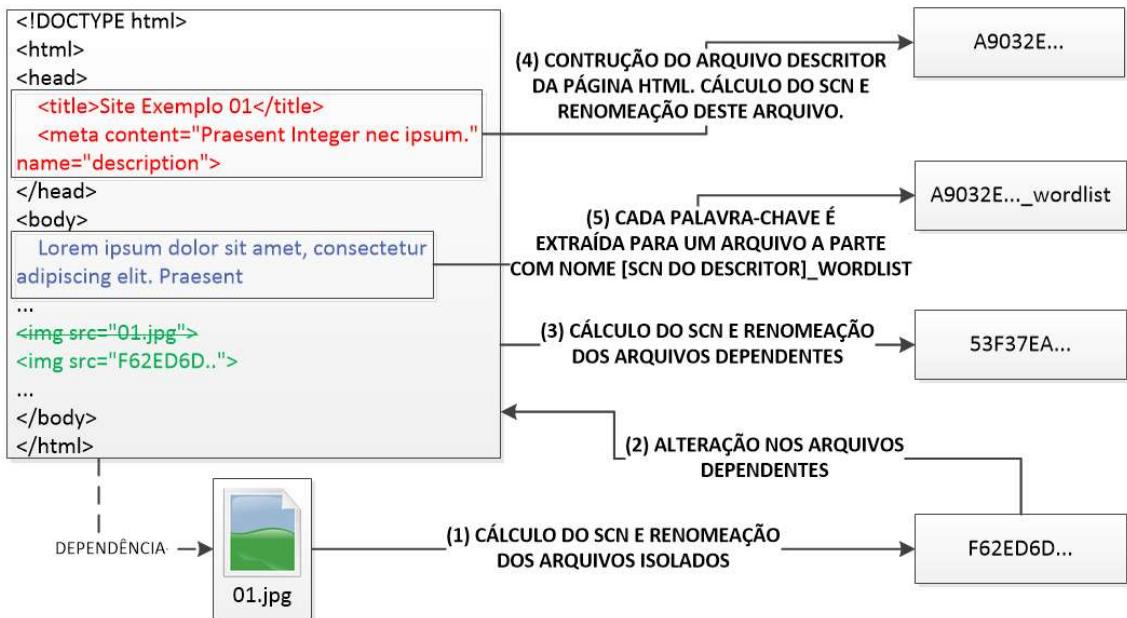


Figura 15 - Processo de conversão de site.

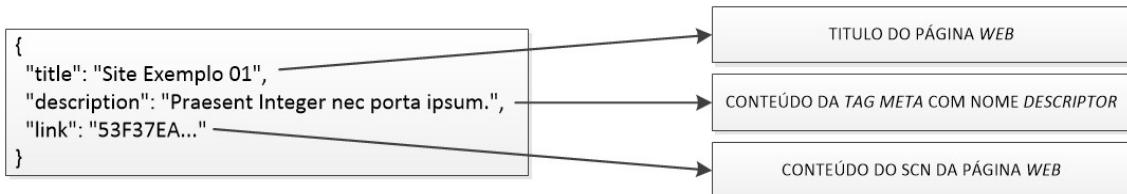


Figura 16 - Conteúdo de um descritor.

4.2.1.1 Exemplo de conversão de um único site

No lado esquerdo da Figura 17 demonstra-se um exemplo de uma estrutura de *site web* simples que poderia ser utilizado tanto na Internet atual, quanto em novas arquiteturas de rede. Esta estrutura serve de entrada para o *NGConverter*, que após convertido, resulta na parte direita da Figura 17. É esperado que após a conversão do *site*, esse tenha 2 arquivos a mais para cada arquivo HTML do *site* original. Eles representam o descritor e a lista de palavras-chaves do arquivo HTML. Em uma rápida análise a Figura 17, é possível observar o arquivo com extensão “*_wordlist*”, que representa a lista de palavras-chaves e o descritor deste HTML, representado pelo mesmo nome do arquivo da lista de palavras-chaves, porém sem a extensão “*_wordlist*”.

```

[fedora@localhost HTMLFiles]$ tree
.
└── site01
    ├── 01.jpg
    ├── 02.jpg
    ├── 03.jpg
    └── index.html

1 directory, 4 files
[fedora@localhost HTMLFiles]$

[ fedora@localhost NGFiles]$ tree
.
├── 09AA661FE5C84F646707B3B60ED673D5
├── 9789EF75DEAA6C6DB34C22EAA96D4A97
├── A1E0DBC95052460F910AC8EE7ECC3849
├── AA6C9E8353DDCE9E934EE2C77B7D4A72
├── AA6C9E8353DDCE9E934EE2C77B7D4A72_wordlist
└── D4CA335CAC00AEF6605746358777EAA6

0 directories, 6 files
[ fedora@localhost NGFiles]$

```

Figura 17 - Demonstração do antes e depois da conversão do Site Exemplo 01.

A Figura 18 reporta as atividades realizadas pelo *NGConverter* durante a conversão do Site Exemplo 01. Através dele é possível notar as tarefas descritas na Figura 15, onde os arquivos isolados neste exemplo representam as imagens: 01.jpg, 02.jpg e 03.jpg.

```

[fedora@localhost HtmlToHash]$ ./NGConverter.sh
index: ./HtmlToHash/SiteFiles/NGFiles/site01/03.jpg => A1E0DBC95052460F910AC8EE7ECC3849
index: ./HtmlToHash/SiteFiles/NGFiles/site01/02.jpg => D4CA335CAC00AEF6605746358777EAA6
index: ./HtmlToHash/SiteFiles/NGFiles/site01/01.jpg => 09AA661FE5C84F646707B3B60ED673D5
replace: 03.jpg => A1E0DBC95052460F910AC8EE7ECC3849 at ./HtmlToHash/SiteFiles/NGFiles/site01/index.html
replace: 02.jpg => D4CA335CAC00AEF6605746358777EAA6 at ./HtmlToHash/SiteFiles/NGFiles/site01/index.html
replace: 01.jpg => 09AA661FE5C84F646707B3B60ED673D5 at ./HtmlToHash/SiteFiles/NGFiles/site01/index.html
index: ./HtmlToHash/SiteFiles/NGFiles/site01/index.html => 9789EF75DEAA6C6DB34C22EAA96D4A97
building-descriptor: 9789EF75DEAA6C6DB34C22EAA96D4A97(index.html)
index-meta: ./HtmlToHash/SiteFiles/NGFiles/site01/9789EF75DEAA6C6DB34C22EAA96D4A97_metafile => AA6C9E8353DDCE9E934EE2C77B7D4A72
index-wordlist: ./HtmlToHash/SiteFiles/NGFiles/site01/9789EF75DEAA6C6DB34C22EAA96D4A97_wordlist => AA6C9E8353DDCE9E934EE2C77B7D4A72_wordlist
moving: ./HtmlToHash/SiteFiles/NGFiles/site01/* => ./HtmlToHash/SiteFiles/NGFiles/site01/..
delete: ./HtmlToHash/SiteFiles/NGFiles/site01
[fedora@localhost HtmlToHash]$

```

Figura 18 – Conversão do Site Exemplo 01.

4.2.2. NGAppPublisher

O *NGAppPublisher* é um processo que possui os protocolos NovaGenesis e assim se comunica com outros processos da rede. Dessa forma é possível realizar as operações de publicação e assinatura de ligações entre nomes e/ou objetos de informação. Este processo é temporário e é executado somente com o objetivo de ler uma pasta com os arquivos preparados pelo *NGConverter* e publicá-los na rede NovaGenesis. A associação efetuada na publicação é simples, do qual cada SCN é associado ao arquivo respectivo arquivo. Isto, torna possível o acesso direto ao arquivo apenas utilizando o SCN. Outra etapa realizada, é a associação das palavras-chaves extraídas das páginas para o respectivo SCN do descritor. Desta forma, é possível buscar uma palavra-chave e descobrir qual página ela está inserida. Cada palavra-chave pode ser associada a mais de um descritor, resultando na descoberta de

mais de uma página que está inserida a mesma palavra-chave, por ex.: comparado ao sistema de busca do Google que retorna ao usuário as páginas encontradas através da palavra-chave buscada.

4.2.3. NGAppCommunicator

O *NGAppCommunicator* é um processo que também possui os protocolos da NovaGenesis e assim como o *NGAppPublisher* também pode realizar as operações de publicação/assinatura. A diferença é que esse é executado durante o tempo em que o navegador *web* é usado. Sua finalidade é expor uma API de acesso simples ao *NGBrowser*, retirando toda a complexidade do protocolo na aplicação visual. A exposição desta API é efetuada através do *D-Bus*, portanto mais de uma aplicação visual diferente pode utilizar essa funcionalidade. O principal objetivo do *NGAppCommunicator* é intermediar a comunicação entre a aplicação visual e a arquitetura NovaGenesis, abstraindo funcionalidades simples e usuais das diferentes aplicações visuais com essa finalidade. Neste trabalho, dois métodos foram criados: (i) o *SearchByMurmur* para solicitar acesso direto ao conteúdo; e o (ii) *SearchByLiteral* para solicitar todos os descritores de uma ou mais palavras-chaves requisitas pelo usuário. Um método deve ser implementado no requisitante – chamado de “*complete*” – para receber uma lista de SCNs como resposta ao invocar os métodos *SearchByLiteral* ou *SearchByMurmur*. Os parâmetros para as chamadas aos métodos *SearchByLiteral* e *SearchByMurmur*, bem como o retorno através do “*complete*” são formatados usando JSON com um *array* de *strings*.

O *NGAppCommunicator* implementa o modelo de armazenamento do navegador NovaGenesis. Quando uma lista de SCNs é recebida como resultado de qualquer busca, esta lista é comparada com a que existe em disco. Caso exista, não é efetuada a assinatura do arquivo. O mesmo acontece quando ocorre um pedido de assinatura de um arquivo através do *browser*, sendo que ele já foi assinado anteriormente. Também implementa o sistema de segurança, em que recertifica o SCN do conteúdo com o SCN solicitante, caso sejam incoerentes o arquivo é descartado.

4.2.4. NGBrowser

O *NGBrowser* é um *software* aplicativo visual que o usuário utiliza para navegar na *web* NovaGenesis. A comunicação com a NovaGenesis é realizada através do *NGAppCommunicator* que emprega *D-Bus*. As URLs de acesso ao conteúdo definem o processo decisório que escolhe qual método do *NGAppCommunicator* será chamado. Neste trabalho, foram definidas somente dois tipos de URLs, as que começam com “ngs://” (*NovaGenesis Search*) e as que começam com “ngu://” (*NovaGenesis Unique*), outros tipos serão definidos futuramente para agregar várias funcionalidades ao usuário. O *NGBrowser* implementa a chamada recursiva de dados dependentes para exibir a página, por ex.: caso um HTML dependa de um arquivo CSS, é efetuada uma assinatura do conteúdo CSS através da URL NGU e se houver uma imagem atribuída dentro deste assinado, uma nova assinatura desta imagem será requisitada novamente através da URL NGU. As chamadas recursivas são identificadas somente após o *download* do conteúdo ser concluído.

4.2.4.1 NovaGenesis Search (ngs://)

As URLs do tipo “ngs://” são utilizadas para pesquisa e são compostas por palavras-chaves em linguagem natural podendo conter espaço entre elas. As palavras-chaves são as escolhidas pelo usuário com o desejo de encontrar páginas *web* que as contenham. Exemplo: “ngs://carro vermelho”, traz como resultados páginas que contêm as palavras-chaves carro ou vermelho. Na versão utilizada por este trabalho não existe uma ordem de classificação para exibir os resultados encontrados. Estas URLs invocam o método *SearchByLiteral* do *NGAppCommunicator*, que quando finalizado, invoca o método “*complete*”, passando uma lista de SCNs dos descritores das páginas. O navegador por sua vez abre cada descritor e formata o conteúdo para uma página comum, exibindo em cada linha o título, descrição da página *web* e um link de acesso a esta página encontrada.

4.2.4.2 NovaGenesis Unique (ngu://)

As URLs do tipo “ngu://” são utilizadas para requisitar um único conteúdo da rede, seja uma foto, página, *scripts*, *stylesheet*, etc. Qualquer conteúdo em formato de arquivo. O método “ngu://” é uma URL que referencia a um SCN. É uma URL geralmente utilizada para

um acesso direto ao arquivo, podendo ser usada em *tags* HTML como “``” ou “``” que exigem em seus parâmetros *src* ou *href*, respectivamente. Por definição, as *tags* que requisitam um conteúdo direto não necessitam da palavra “ngu://”, somente o SCN já é o suficiente para o *NGBrowser* identificar a requisição de único arquivo. O método “ngu://” pode ser digitado através da barra de endereços do navegador, ex.: “ngu://EC9E8A5F6354AD70812196ABB89ECAEB”, que carrega o conteúdo por trás desse código *hash* hexadecimal.

Capítulo 5. Resultados Experimentais

A arquitetura utilizada na avaliação experimental foi a mesma ilustrada na Figura 1, onde o meio de comunicação adotado foi a tecnologia *Ethernet*. Foram construídos 11 *sites* com uma página contendo palavras aleatórias e 3 fotos. Cada *site* teve 2 fotos repetidas dos outros 10 *sites*, seguindo a mesma estrutura do tópico “4.2.1.1”. Os próximos tópicos estão relacionados a preparação das páginas *web*, inicialização da NovaGenesis, publicação das páginas *web* e por fim a navegação utilizando as URLs “ngs://” e “ngu://”.

5.1 Preparação das páginas *web*

Com os *sites* de exemplo construídos, inicia-se a conversão através da ferramenta *NGConverter*. É esperado que tanto a pasta de origem quanto a pasta resultante da conversão contenham 45 arquivos no total. Pois, o conversor irá retirar 2 arquivos duplicados de cada *site* e gerar mais 2 arquivos de cada HTML, totalizando a mesma quantidade. A Figura 19 mostra a pasta resultante da conversão.

```
[fedora@localhost NGFiles]$ ls
@9AA661FE5C84F646707B3B60ED673D5
@BCE2FE16AA7672442E512FD6E916C80
165E75746BCC09B8A7A642C0C5BBDE2
1714334D85B9CFEA5568A1A51D1AF22F
1901D6A7B56307854C48034EFC9232
1A1304B6422F1B5552B986C620881873
1A1304B6422F1B5552B986C620881873_wordlist
2487295441CF4A33366A812F63B92931G
3679DEC5E0F96B36B4D1B5CFE091B590
41267271432A3054BF5A552C8495300
55EEFD60A282317A24FB65225980DAB
575B5B52C48F73EBEBEF9DD1694391C5
5E8199D17420DB3D315A4E42483FD0CF
5E8199D17420DB3D315A4E42483FD0CF_wordlist
6D6D620BE51BE854C11E65DEF2FF9A7
[fedora@localhost NGFiles]$ ls | wc -l
45
[fedora@localhost NGFiles]$
```

```
6D6D620BE51BE854C11E65DEF2FF9A7_wordlist
6E85E2B913D729870CFD3EC062A03F44
770516326045414C546BC749AB606B18
770516326045414C546BC749AB606B18_wordlist
7AC9FD032A95B24DACC8195B180986BE
7AC9FD032A95B24DACC8195B180986BE_wordlist
8323387C1F15CE9FB52B7641C0C95090
8DDC9C88974B6BEF41ECD54EBFA0397
9789EF75DEAA6C6D834C22EA96D4A97
A11BEE80D2B627C445F5FB908E2C5EE
A1E0DBC95052460F910AC8EE7ECC3849
A7AF50D17127D19D386CA686806B5EB8
A86EB041E0772BB06180C36C722A36FB
A86EB041E0772BB06180C36C722A36FB_wordlist
AA6C9E8353DDCE9E934EE2C77B7D4A72
AA6C9E8353DDCE9E934EE2C77B7D4A72_wordlist
B5126FF4D7B5B68EF7742062F3425714
BF2CCF196994DC6790873889DFED3C10
C1531F43B9614AA701DCE9005B6670E
C268098BCA38F4A931F08B0B7E561A16
C268098BCA38F4A931F08B0B7E561A16_wordlist
CBD5B23A0F9265053FF6443A822C5C2
CE58C9AF2DC32A87EA63EBB88099B89
CE58C9AF2DC32A87EA63EBB88099B89_wordlist
D4CA335CAC09AEF6605746358777EA6
D861C0EFE444E9113992962F81BAC72
DFO069473EB0FE0468F5D7214B8856480
DFO069473EB0FE0468F5D7214B8856480_wordlist
E077EEA4FB2D4EFA19151178874B525
E077EEA4FB2D4EFA19151178874B525_wordlist
```

Figura 19 - Resultado da conversão dos 11 sites.

5.2 Inicialização da NovaGenesis

Para que a NovaGenesis esteja funcionando em dois computadores diferentes, primeiro é necessário executar seus principais processos no *host* que assumirá o papel de servidor de páginas. Desta forma, os processos estarão aptos a se comunicarem entre si, inclusive pelas interfaces de rede. Logo em seguida, em outro *host*, já se pode iniciar o cliente que se conecta automaticamente a NovaGenesis. A Figura 20 mostra os quatro principais

processos da NovaGenesis em execução: PGS, GIRS, PSS, HTS da esquerda para a direita e de cima para baixo. Nesta tela, eles aguardam o cliente se conectar.

```

Atividades Terminal Dom, 21:48*
Terminal Arquivo Editar Ver Pesquisar Terminal Ajuda

```

The image shows four terminal windows side-by-side, each displaying command-line output from a NovaGenesis process. The processes are identified by their window titles: PGS, GIRS, PSS, and HTS. The output in each window includes various system commands and status messages, such as file paths, process IDs (PID), and sequence numbers (SCN). The HTS window contains a significant amount of text related to file operations and memory management.

Figura 20 - Quatro principais processos da NovaGenesis.

5.3 Publicação das páginas web

Toda a publicação é efetuada através do *NGAppPublisher* que se conecta à rede NovaGenesis. O processo *NGAppPublisher* utiliza os arquivos convertidos pelo *NGConverter* para iniciar as publicações. Na solução proposta neste trabalho, as páginas são publicadas no *cache* de rede da NovaGenesis (chamado HTS), ao invés de ficarem armazenadas em servidores *Web* acessíveis via HTTP. A Figura 21 reporta o processo *NGAppPublisher* comunicando-se com os outros processos NovaGenesis. A Figura 22 reporta a publicação da foto “02.jpg” do *site* Exemplo 01. A Figura 23 reporta a associação das palavras-chaves “vel”, “nunc”, “turpis”, “condimentum” e “premium” ao descritor do *site* Exemplo 01 com SCN “AA6C9E8353DDCE9E934EE2C77B7D4A72”. Após o término das publicações e o encerramento do processo *NGAppPublisher*, o HTS contém todos os arquivos dos *sites* gerados. Então, a Figura 24 reporta os arquivos publicados. Nota-se que o número de arquivos publicados (35 arquivos) é menor do que quando se considera os arquivos temporários, pois todos os arquivos com extensão “_wordlist” não foram publicados, mas sim lidos para criar a associação entre a palavra-chave e o descritor da página HTML em questão.

```

Arquivo Editar Ver Pesquisar Terminal Ajuda
> < 1 string 7B249B2B1577D901D2EA9C3065D5AD97_Core_BID >
ng -pub --bind 0.1 [ < 1 string 2 > < 1 string 55EEFD6DA282317A24FBB05225980DAB
> < 1 string 64EA88E95728B6E95E4ACEEDAE1FECD9_App_PID >
ng -pub --bind 0.1 [ < 1 string 2 > < 1 string 55EEFD6DA282317A24FBB05225980DAB
> < 1 string A4F2FA95596205CAB46A437B7ECEA30A_OSID >
ng -pub --bind 0.1 [ < 1 string 9 > < 1 string 55EEFD6DA282317A24FBB05225980DAB
> < 1 string 76B3622BE503236CF03F55CB32860108_HID >
ng -message --type 0.1 [ < 1 string 1 > ]
ng -message --seq 0.1 [ < 1 string 42 > ]
ng -scn --seq 0.1 [ < 1 string 7BC4D0B3878B525845DF8F5BF5D03723 > ]

There is a payload of 10385 bytes
)

-msg --cl 0.1
[6] (Pushing a message to the OutputQueue. Size = 0)

[3] (Finished processing)
. (Waiting 2.443373076 sec)
.....[5] (Writing a message to the shared memory)

[4] (Read a message from the OutputQueue)

```

Figura 21 – Log de execução do NGAppPublisher.

```

ng -store --bind 0.1 [ < 1 string 2 > < 1 string D4CA335CAC00AEF6605746358777EAA6 >
ng -info --payload 0.1 [ < 1 string D4CA335CAC00AEF6605746358777EAA6 > ]

```

Figura 22 - Mensagem NovaGenesis de publicação de arquivo.

```

ng -pub --bind 0.1 [ < 1 string 2 > < 1 string vel > < 1 string AA6C9E8353DDCE9E934EE2C77B7D4A72 > ]
ng -pub --bind 0.1 [ < 1 string 2 > < 1 string nunc > < 1 string AA6C9E8353DDCE9E934EE2C77B7D4A72 > ]
ng -pub --bind 0.1 [ < 1 string 2 > < 1 string turpis > < 1 string AA6C9E8353DDCE9E934EE2C77B7D4A72 > ]
ng -pub --bind 0.1 [ < 1 string 2 > < 1 string condimentum > < 1 string AA6C9E8353DDCE9E934EE2C77B7D4A72 > ]
ng -pub --bind 0.1 [ < 1 string 2 > < 1 string pretium > < 1 string AA6C9E8353DDCE9E934EE2C77B7D4A72 > ]

```

Figura 23 - Associação de palavras-chave a um descritor.

```

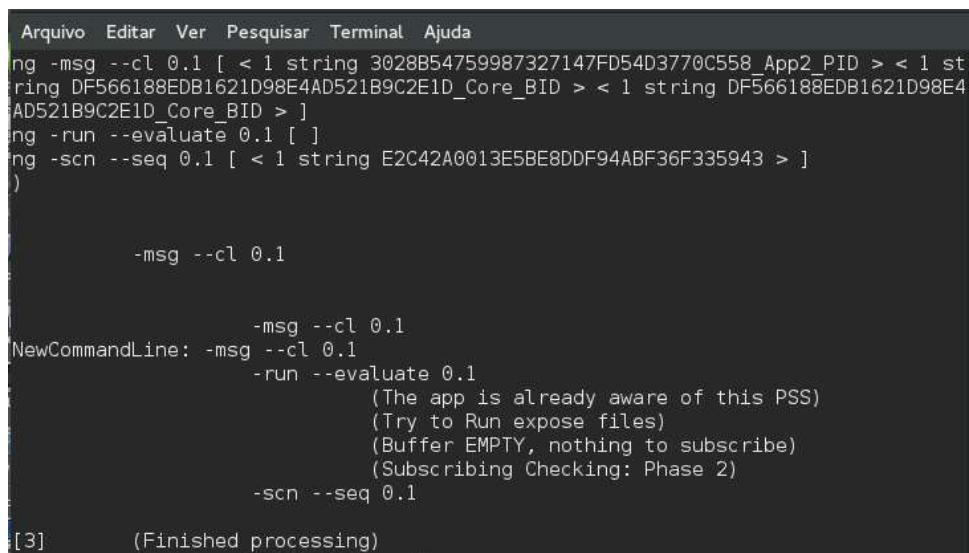
[fedor@localhost HTS]$ ls
09AA61F5EC84F646707B3B60ED673D5 55EEFD6DA282317A24FBB05225980DAB 9789E75DDEAA6C6DB34C22EAA96D4A97 C268098BCA38F4A931F0B8B0B7E561A16
08CE2FE16AA7672442E512FD6E916C80 575B5B52C48F73E8BE8F9DD1094391C5 A11EBEE0DD28627C445F5FB9D8E2C5EE CBBD5B23A0F9265053FF6443AB22C5C2
165E7746BCCF09B8A7A64200C56B0E2 5E8199D17420DB3D315A4E42483F00CF A1E00BC95652466F910AC8EE7ECC3849 CE58C9AF2DC32A87E6A3EB0B8809BB89
1714334D85B9CEA5568A1A51D1AF22F 6D6D629B1751BE854C11E65DEF2FF9A7 A7AF5D017127D19D386CA686806B5E8B D4CA335CAC00AEF6605746358777EAA6
1901D6A78A566307854C4B034EFC9232 6E85E2B913D720870CF3E062A03F44 A86EB041E0772BB0618DC30C722A36FB D861C0EEF444E9113992962F81BAFC72
1A1304B6422F1B5552B986C620881873 770516326045414C546BC749AB606B18 A46C9E8353DDCE9E934EE2C77B7D04472 DF0069473EB0FF046BF5D7214B850480
248729544FCFA33366A812F63B929310 7AC9FD032A95B24DACC9B195B180986BE B5126FF4D7B6B68EF7742C62F3425714 EC77EEA4FB2D4EFA19151178B74B525
36790EC5E0F96B3684D1B5CFE081B590 8323387C1F15CE9FB52B76410CC95090 BF2CCF1969940C679D873B890FED3C10 HTS.ini
41267271432A3054FBF5A552C8495300 8DDC0C88974B68EF41EC3D54EBFA0397 C1531F43B9614AA701DCCE9005B6070E
[ feder@localhost HTS]$ ls | wc -l
35

```

Figura 24 - HTS após o término das publicações.

5.4 Navegação

Para iniciar a navegação na NovaGenesis é preciso iniciar o processo *NGAppCommunicator* que irá intermediar a comunicação entre o *NGBrowser* e a NovaGenesis. Quando o processo estiver concluído, será exibida a mensagem: “*Buffer Empty, nothing to subscribe*”. A Figura 25 monstra um *report* de quando isso acontece.



```
Arquivo Editar Ver Pesquisar Terminal Ajuda
ng -msg --cl 0.1 [ < 1 string 302BB54759987327147FD54D3770C558 App2_PID > < 1 st
ring DF566188EDB1621D98E4AD521B9C2E1D_Core_BID > < 1 string DF566188EDB1621D98E4
AD521B9C2E1D_Core_BID > ]
ng -run --evaluate 0.1 [ ]
ng -scn --seq 0.1 [ < 1 string E2C42A0013E5BE8DDF94ABF36F335943 > ]
)

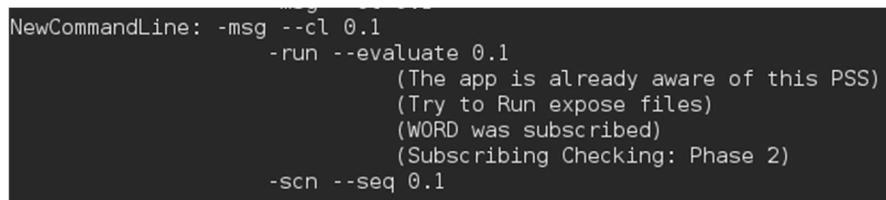
-msg --cl 0.1

NewCommandLine: -msg --cl 0.1
    -run --evaluate 0.1
        (The app is already aware of this PSS)
        (Try to Run expose files)
        (Buffer EMPTY, nothing to subscribe)
        (Subscribing Checking: Phase 2)
    -scn --seq 0.1

[3]      (Finished processing)
```

Figura 25 - *NGAppCommunicator* em espera por navegadores.

Logo, o *NGBrowser* é iniciado e os testes são realizados. Inicia-se por uma pesquisa pela palavra-chave “01” com a intenção de encontrar o “Site Exemplo 01” e depois a pesquisa pelas palavras-chaves “ipsum vac”, para encontrar todos os *sites* publicados. As Figuras 26 e 27 mostram o resultado das pesquisas. É possível observar o *NGAppCommunicator* exibindo uma nota de que os arquivos foram assinados “*WORD was subscribed*”. Em ambos os testes, as exibições dos resultados aconteceram de forma correta no *NGBrowser*.



```
NewCommandLine: -msg --cl 0.1
    -run --evaluate 0.1
        (The app is already aware of this PSS)
        (Try to Run expose files)
        (WORD was subscribed)
        (Subscribing Checking: Phase 2)
    -scn --seq 0.1

[3]      (Finished processing)
```

Figura 26 - Assinatura encontrada e obtida com sucesso.

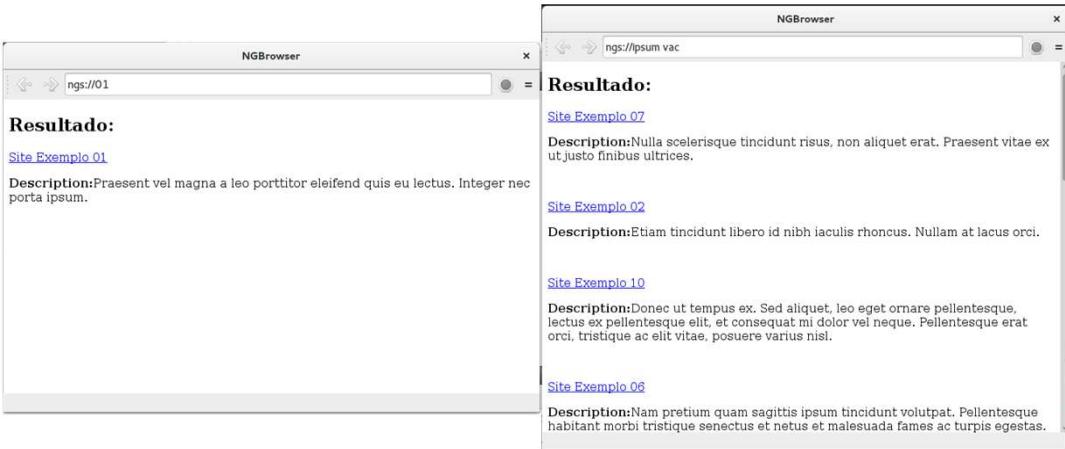


Figura 27 - Resultado do NGS de um site (esquerda) e de vários sites (direita).

Em seguida, a Figura 28 reporta outros dois testes. Primeiro, com um clique sobre o link de um site exemplo resultado do NGS e segundo a exibição de um conteúdo através da NGU.



Figura 28 - Exibição da página HTML (esquerda) e o acesso com o NGU (direita).

5.5. Eficiência do cache

Para validar o ganho de eficiência do *cache* NovaGenesis, foi realizada uma comparação entre o *NGBrowser* e um navegador comum, ignorando o *overhead* de ambos os protocolos das redes. A Figura 29 representa o acumulo em *bytes* trafegados começando do site 1 até 11.

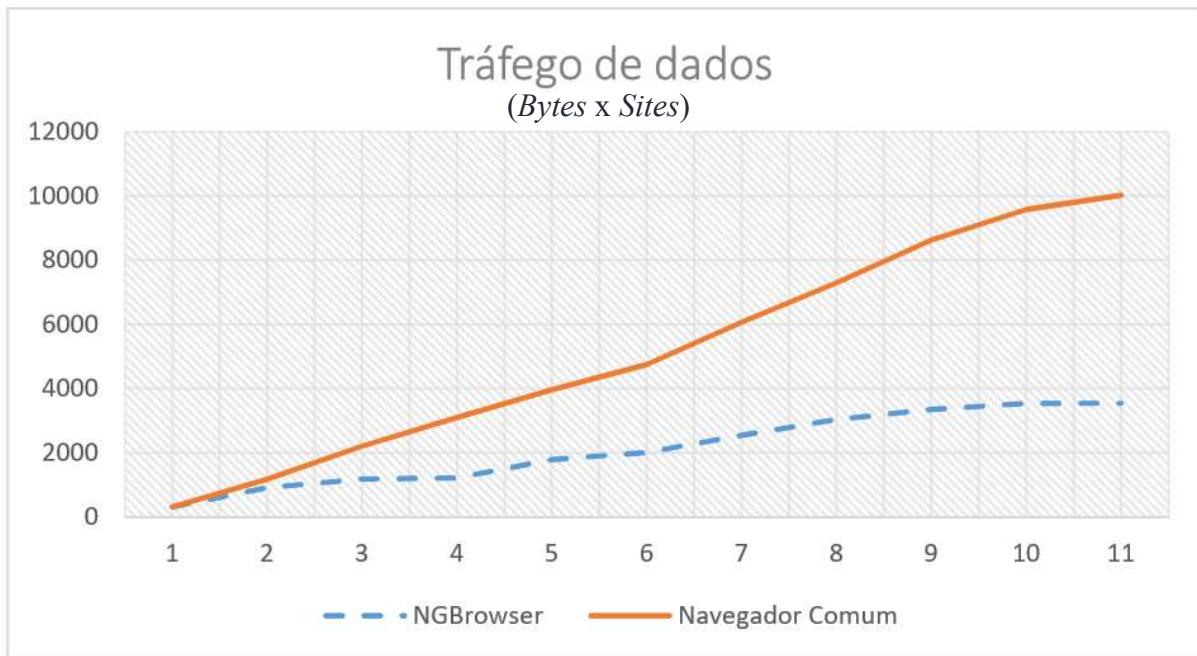


Figura 29 - Eficiência do cache do NGBrowser.

5.6. Teste em escala

A proposta do teste em escala é comprovar a eficiência do *cache* do *NGBrowser* quando se acessa uma grande quantidade de *sites*. Para isso, foi feito um algoritmo em Matlab que simulasse o funcionamento do *cache* do *NGBrowser* e de um navegador comum, em anexo. O algoritmo considera: (i) todos os objetos *web* acessados vão para o *cache*, (ii) os objetos *web* em *cache* não expiram (ou não são apagados), (iii) os *bytes* utilizados correspondem aos *bytes* que o navegador utilizou para abrir a página e sua origem pode ser da rede, do *cache* ou da rede e do *cache*, (iv) a probabilidade de acesso a mesma página (*p1*), (v) a probabilidade de uma página possuir objetos *web* repetidos (*p2*), (vi) distribuição uniforme na geração da aleatoriedade.

5.6.1 Validação do algoritmo

A validação do algoritmo foi baseado no experimento anterior, pois adotando os mesmos valores e realizando o mesmo comportamento do experimento é esperado que o algoritmo produza o mesmo resultado do experimento. Portanto, os dados ilustrados na Figura 30 foram adotados porque representam o tamanho dos objetos *web* utilizados no experimento, também a adoção de valores específicos de probabilidades para simular o mesmo

comportamento, são elas: (i) valor zero ($p1=0$) para a probabilidade de acesso ao mesmo *site*, desta forma o algoritmo acessa os *sites* de forma sequencial do 1 ao 11, (ii) valor cem por cento ($p2=1$) para que as páginas tivessem objetos repetidos, simulando a repetição de duas fotos para cada *site*. A Figura 31 é o resultado obtido do algoritmo, nota-se que o resultado é o mesmo da Figura 29.

```

54 % Col.1 (Web Data), Col.2 (Web Repeated Data)
55 - WebPagesData = [
56     [ 180 , 132 ] % Site 01
57     [ 596 , 260 ] % Site 02
58     [ 276 , 760 ] % Site 03
59     [ 28 , 856 ] % Site 04
60     [ 580 , 284 ] % Site 05
61     [ 212 , 588 ] % Site 06
62     [ 544 , 772 ] % Site 07
63     [ 492 , 736 ] % Site 08
64     [ 308 , 1016 ] % Site 09
65     [ 184 , 780 ] % Site 10
66     [ 12 , 428 ] % Site 11
67 ];

```

Figura 30 - Dados do experimento do Capítulo 5.

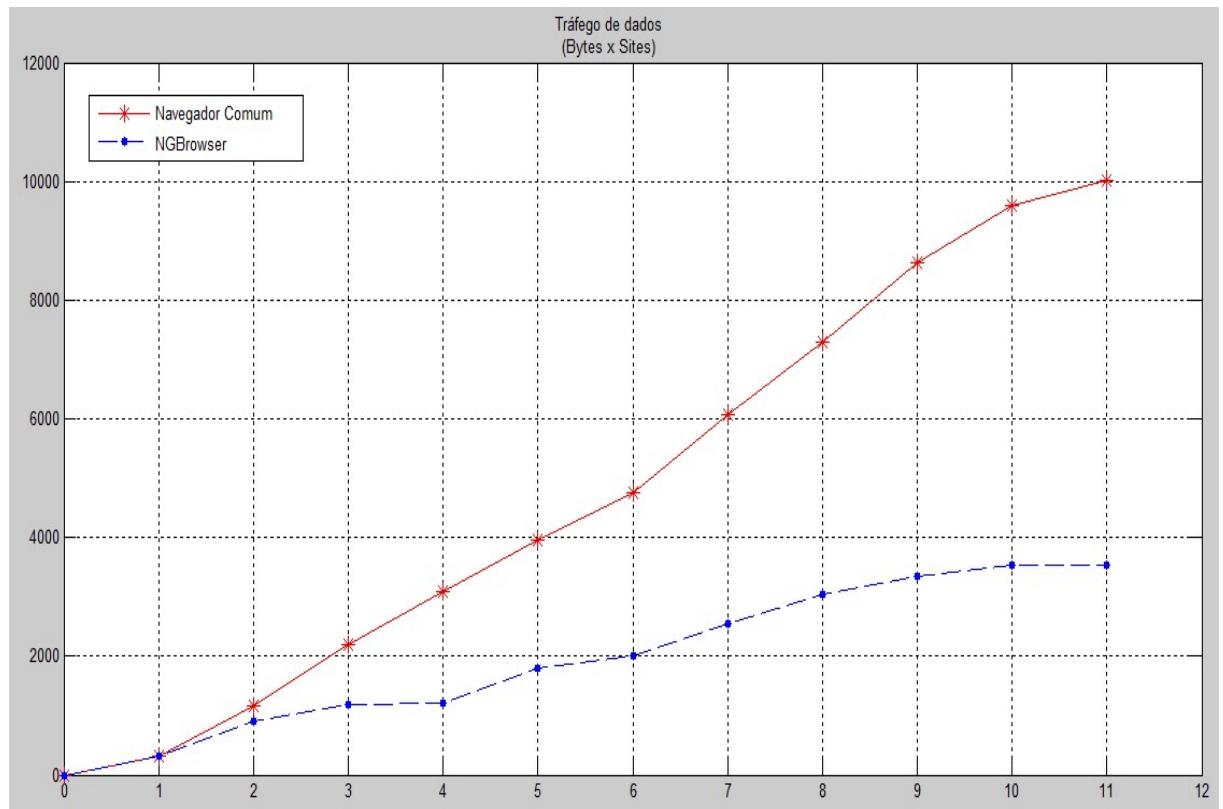


Figura 31 - Teste do algoritmo de simulação utilizando dados experimentais do Capítulo 5.

Ao final da execução do algoritmo, é gerado um *report* tal qual a Figura 32 mostra o relatório do experimento executado e possui as seguintes informações:

- *Interaction* – Quantidade de interações da simulação.
- *Bytes Transferred for Actual Web* – A quantidade em *bytes* transferidos considerando um navegador comum utilizando a rede atual.
- *Bytes Transferred for NovaGenesis Web* – A quantidade em *bytes* transferidos considerando o *NGBrowser* na rede NovaGenesis.
- *Bytes Used From AWCache* – A quantidade em *bytes* utilizados do *cache* de um navegador comum.
- *Bytes Used From NGBrowser* – A quantidade em *bytes* utilizados do *cache* do *NGBrowser*.
- *Total Bytes Used* – A quantidade em *bytes* que a simulação utilizou para abrir cada página *web*.
- *Access Sequence of the Sites* – A sequência dos índices acessados da matriz “*WebDataPages*” (Figura 30).

```
Report:  
- Interaction: 11  
- Bytes Transferred for:  
    Actual Web: 10024  
    NovaGenesis Web: 3544  
- Bytes Used from:  
    AWCache: 0  
    NGBrowser: 6480  
- Total Bytes Used: 10024  
- Access Sequence of the Sites:  
    1  
    2  
    3  
    4  
    5  
    6  
    7  
    8  
    9  
    10  
    11
```

>>

Figura 32 - Relatório de saída em console do algoritmo.

5.6.2 Influência dos dados probabilísticos

A configuração dos dados probabilísticos no algoritmo influencia grandemente em seu resultado. Para demonstrar essa afirmação foram feitas quatro simulações com as seguintes considerações: (i) mil interações, (ii) quantidade em *bytes* de cada objeto *web* e do objeto *web* repetido foram geradas aleatoriamente, (iii) a simulação considera que alguns dos objetos *web* já existem em *cache*, isto significa que cada acesso ao objeto *web* repetido não diz respeito ao objeto transferido durante a simulação. E as probabilidades de cada simulação foram, respectivamente: (i) $p1=0,2$ (20%) e $p2=0,2$ (20%); (ii) $p1=0,8$ (80%) e $p2=0,2$ (20%); (iii) $p1=0,2$ (20%) e $p2=0,8$ (80%); (iv) $p1=0,8$ (80%) e $p2=0,8$ (80%).

A Figura 33 reproduz o resultado da primeira simulação. A reta vermelha está abaixo da verde devido o *cache* do navegador comum entrar em ação e economizar na transferência de *bytes*. A relação entre a reta vermelha e verde é baixa devido a quantidade baixa ($p1=0,2$) do usuário acessar a mesma página. A reta azul mostra a eficiência baixa em relação ao navegador comum, porque a probabilidade de repetição de objetos *web* em *sites* distintos também é baixa ($p2=0,2$).

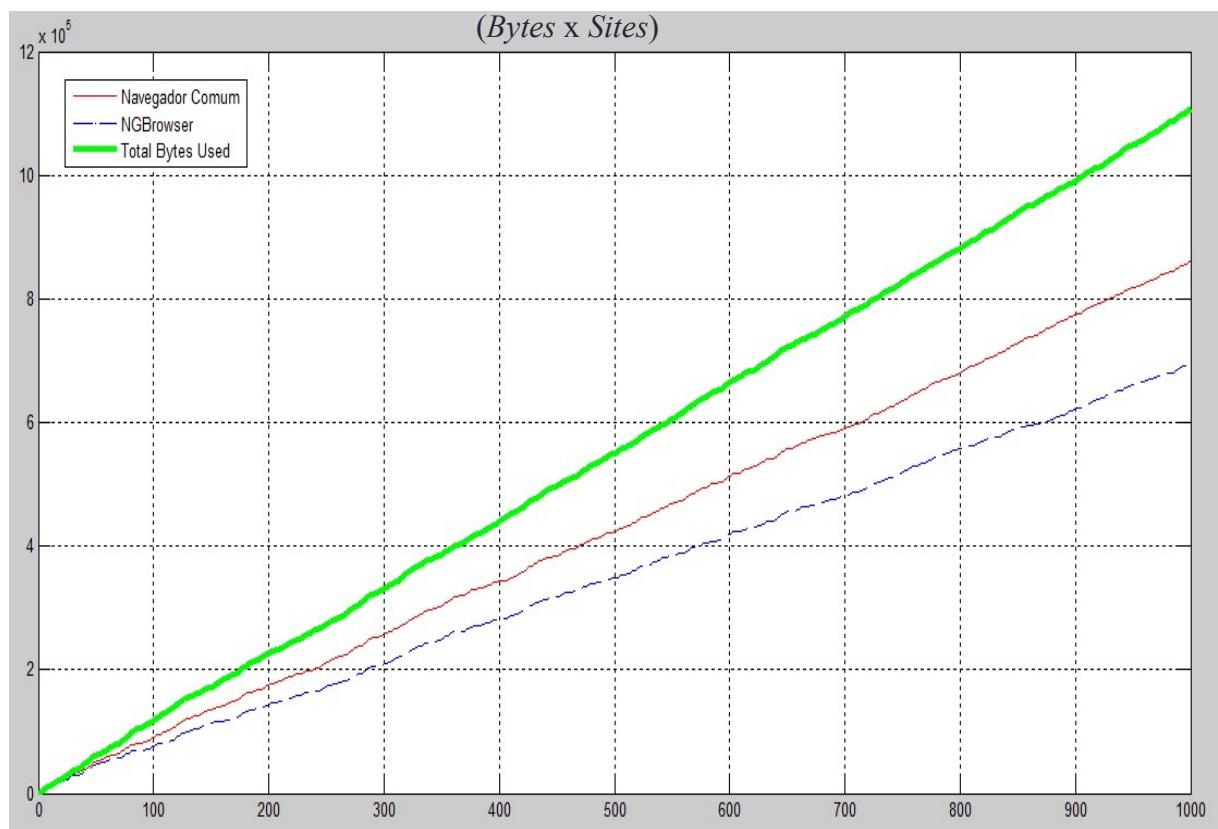


Figura 33 - Resultado da simulação para $p1=0,2$ e $p2=0,2$.

A Figura 34 reproduz o resultado da segunda simulação. Ambas as retas estão distantes da verde porque o *cache* foi bastante utilizado ($p1=0,8$). Mesmo que pequena ($p2=0,2$) a eficiência do *NGBrowser* ainda apresenta vantagens devido existir objetos *web* repetidos em páginas distintas.

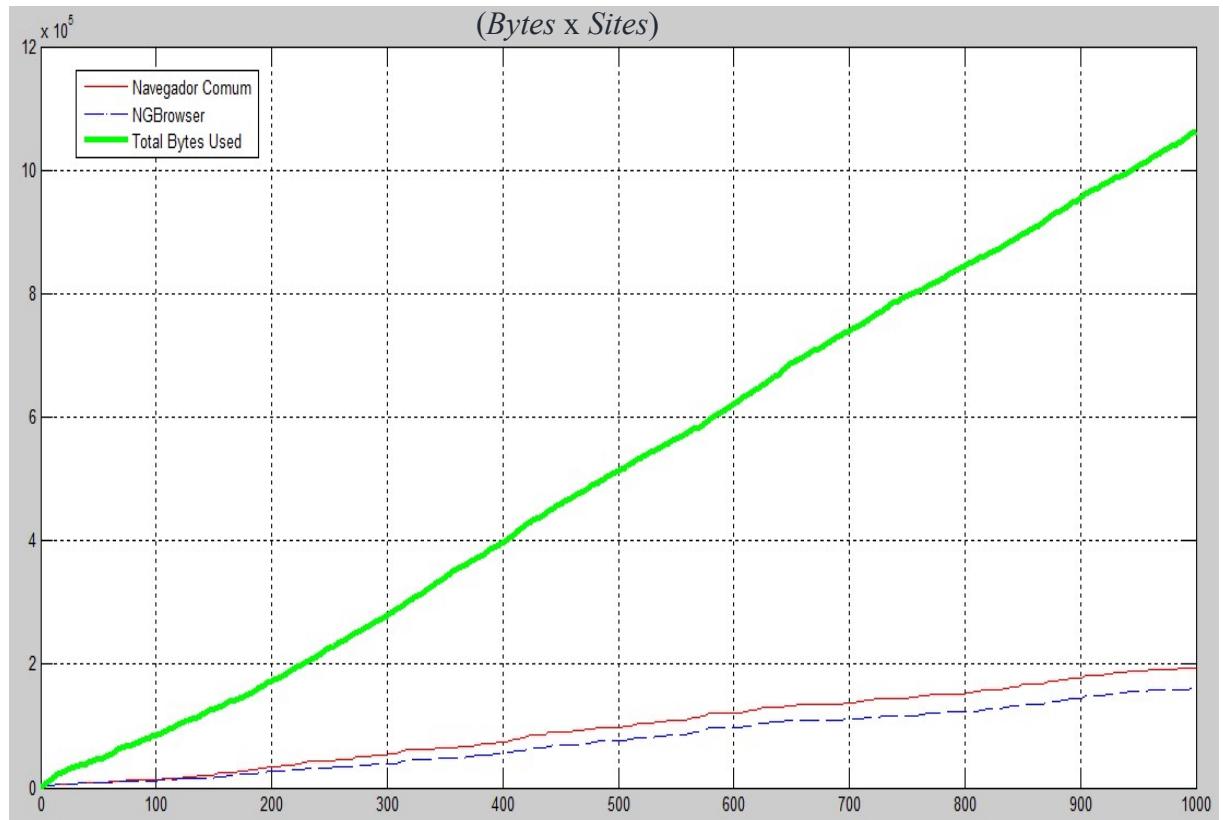


Figura 34 - Resultado da simulação para $p1=0,8$ e $p2=0,2$.

A Figura 35 mostra o resultado da terceira simulação. A reta verde se aproximou novamente como da Figura 33, porque o usuário começou a acessar menos vezes os mesmos *sites* ($p1=0,2$) e os *sites* que acessou tiveram uma quantidade alta de objetos *web* repetidos ($p2=0,8$).

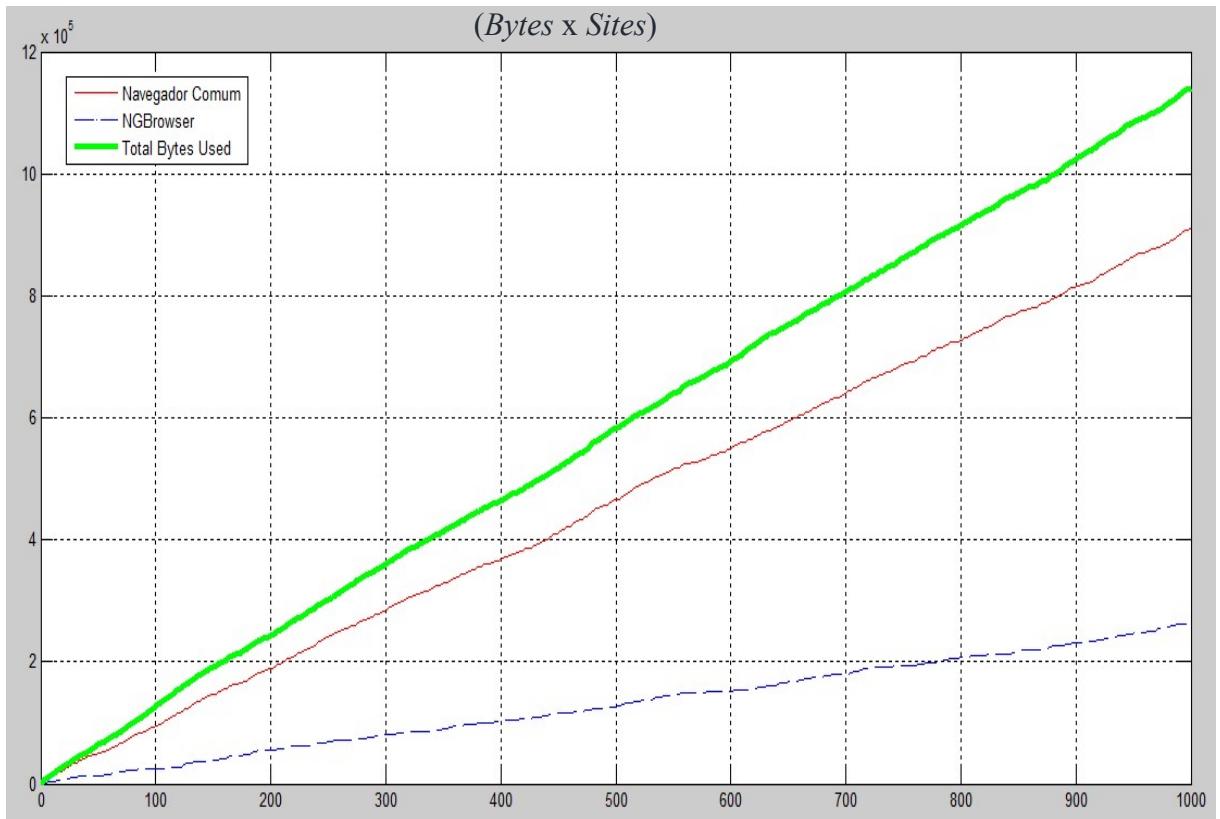


Figura 35 - Resultado da simulação para $p1=0,2$ e $p2=0,8$.

A Figura 36 mostra o resultado da quarta simulação. Ambas as retas vermelha e azul estão bem abaixo da reta verde porque o usuário optou em acessar o mesmo *site* várias vezes ($p1=0,8$). A reta azul ainda é eficiente com relação a vermelha, mas não muito eficiente porque os acessos aos mesmos *sites* são excessivos. Em consequência o acesso a *sites* distintos com objetos repetidos acontece com pouca frequência o que justifica a proximidade das retas vermelha e azul.

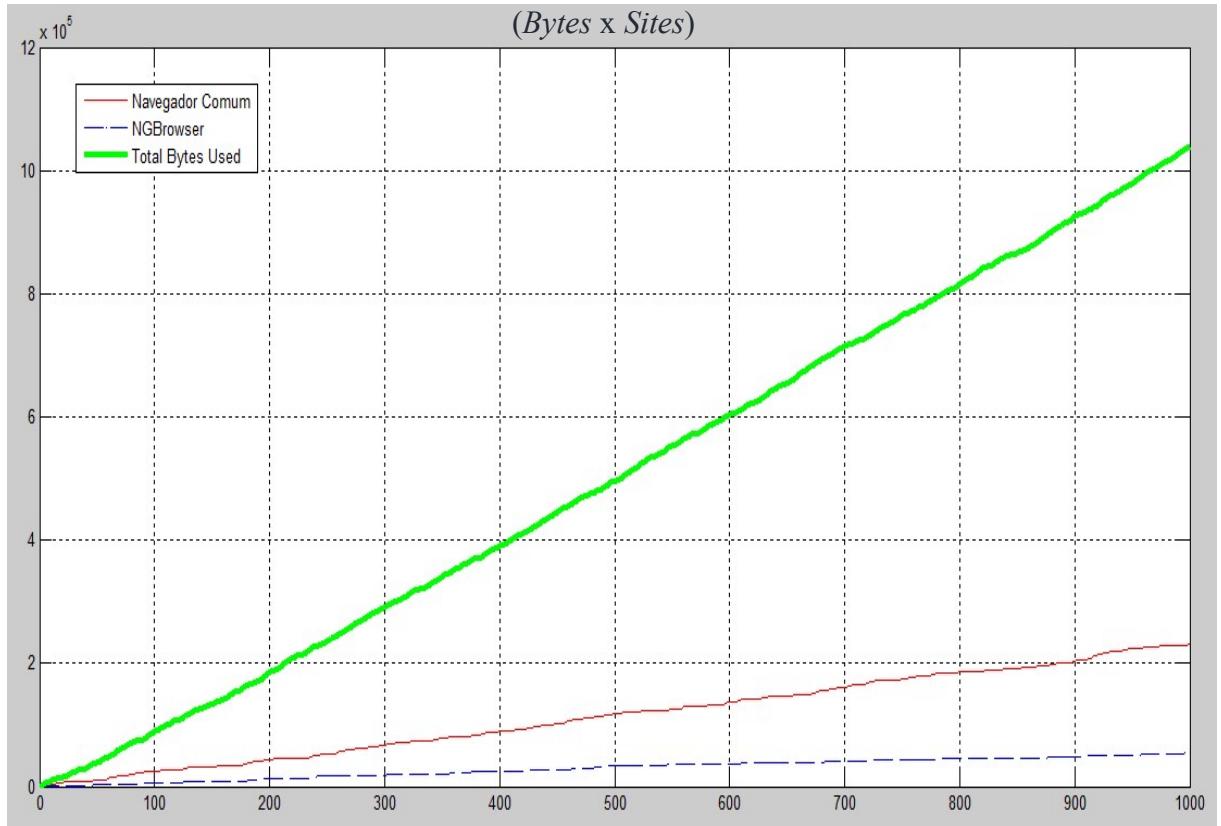


Figura 36 - Resultado da simulação para $p1=0,8$ e $p2=0,8$.

As simulações comprovam que o *NGBrowser* tem eficiência sobre qualquer valor de probabilidade, exceto para $p1=1$ e $p2=0$. Onde $p1=1$ e $p2=0$ tornaria ambos os navegadores equivalentes, pois $p1=1$, sinaliza que o usuário nunca acessaria *sites* diferentes e $p2=0$ indicaria que jamais existira objetos *web* repetidos na rede. O *NGBrowser* apresenta máxima eficiência quando $p1=0$ e $p2=1$, conforme mostra a Figura 31.

A principal vantagem do *NGBrowser* é o fato dele reusar os objetos *web* idênticos que são presentes em diferentes *sites*. Isto só é possível porque a NovaGenesis trata o conteúdo como único na rede.

Encontrar os valores reais de probabilidades de acesso e objetos repetidos não é uma tarefa simples, por isso a pesquisa pelos valores reais fará parte dos trabalhos futuros deste projeto. Embora não se tenha os valores reais, é possível notar que qualquer valor diferente de $p1=1$ e $p2=0$ valida a eficiência do *NGBrowser* sobre o navegador comum.

5.6.3 Simulação do teste em escala

O teste em escala comprova que o *NGBrowser* tem mais eficiência no uso do *cache* do que um navegador comum. Para provar essa afirmação foram adotadas probabilidades de pior caso (valores baixos para p_1 e p_2). A seguinte simulação foi feita com as seguintes considerações: (i) $p_1 = 0,001$ e $p_2 = 0,001$; (ii) um milhão de interações, (iii) quantidade em *bytes* de cada objeto *web* e do objeto *web* repetido foram geradas aleatoriamente, (iv) a simulação considera que alguns dos objetos *web* já existem em *cache*. As Figura 37, 38, 39 e 40 mostram o resultado da simulação. A Figura 37 mostra o início da simulação. Somente uma reta é exibida, isto significa que de início nenhum dos navegadores apresenta economia em *bytes* devido à baixa chance ($p_1=0,001$) do usuário acessar a mesma página e a baixa chance ($p_2=0,001$) das páginas possuírem objetos repetidos. Este comportamento começa a mudar na Figura 38, onde a partir de 71 *sites* acessados os objetos repetidos começam a aparecer nas páginas e o uso do *cache* do *NGBrowser* começa a entrar em ação e economizar a banda. A Figura 39 mostra quando a eficiência do *cache* no navegador comum começa a acontecer. Percebe-se que a eficiência só se torna efetiva depois de 428 *sites*. A Figura 40 mostra o final da simulação, onde as retas estão normalizando ficando similar ao cenário da Figura 33.

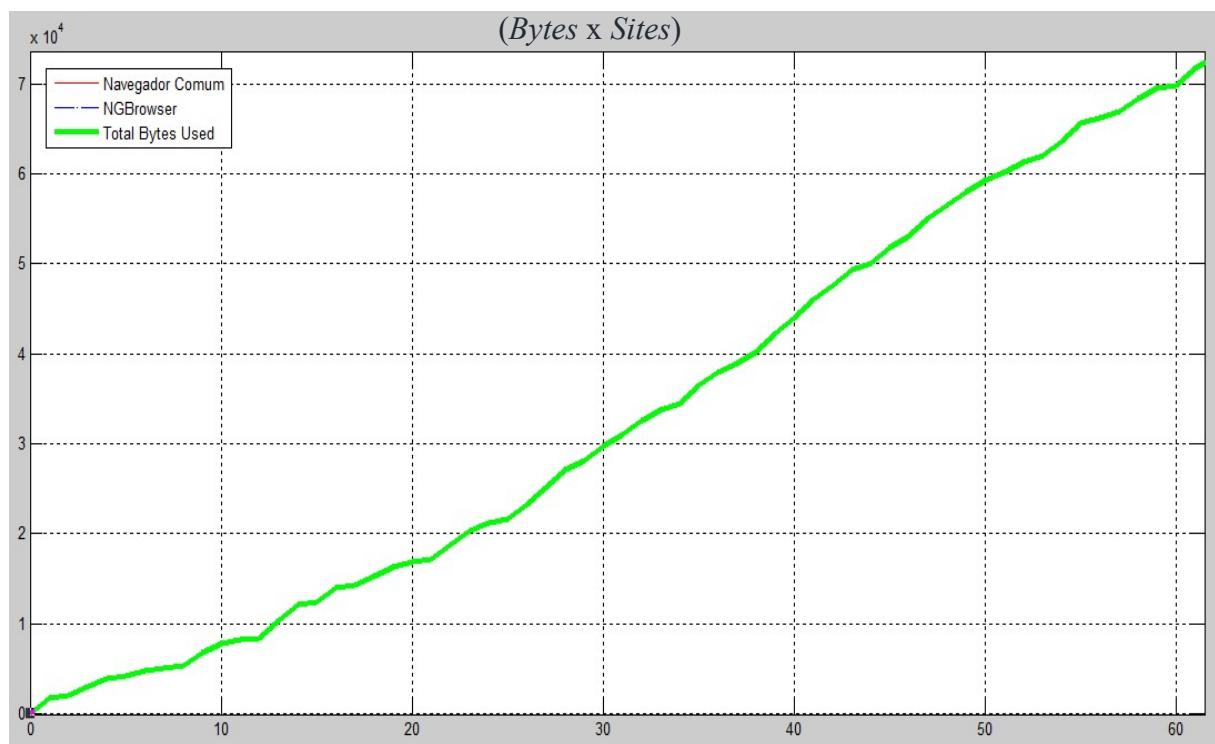


Figura 37 - Início da simulação do teste em escala.

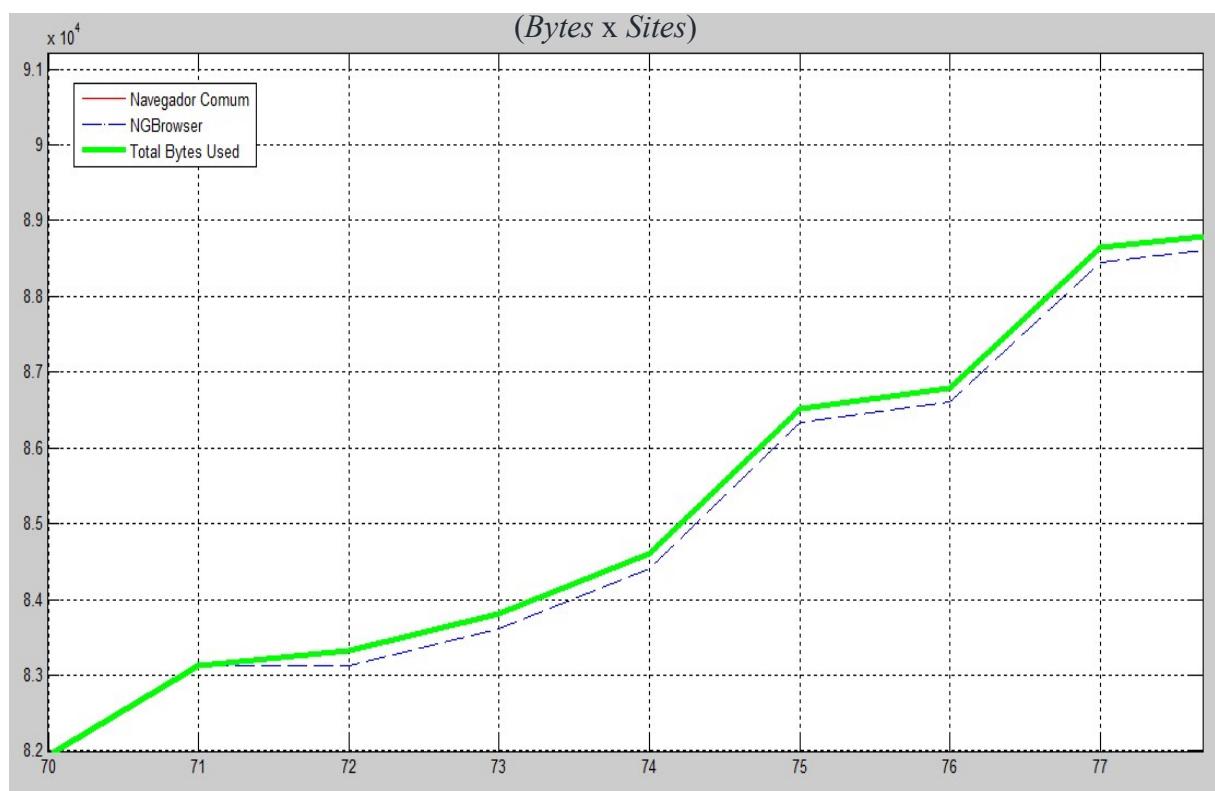


Figura 38 - Eficiência no uso do cache do NGBrowser no teste em escala.

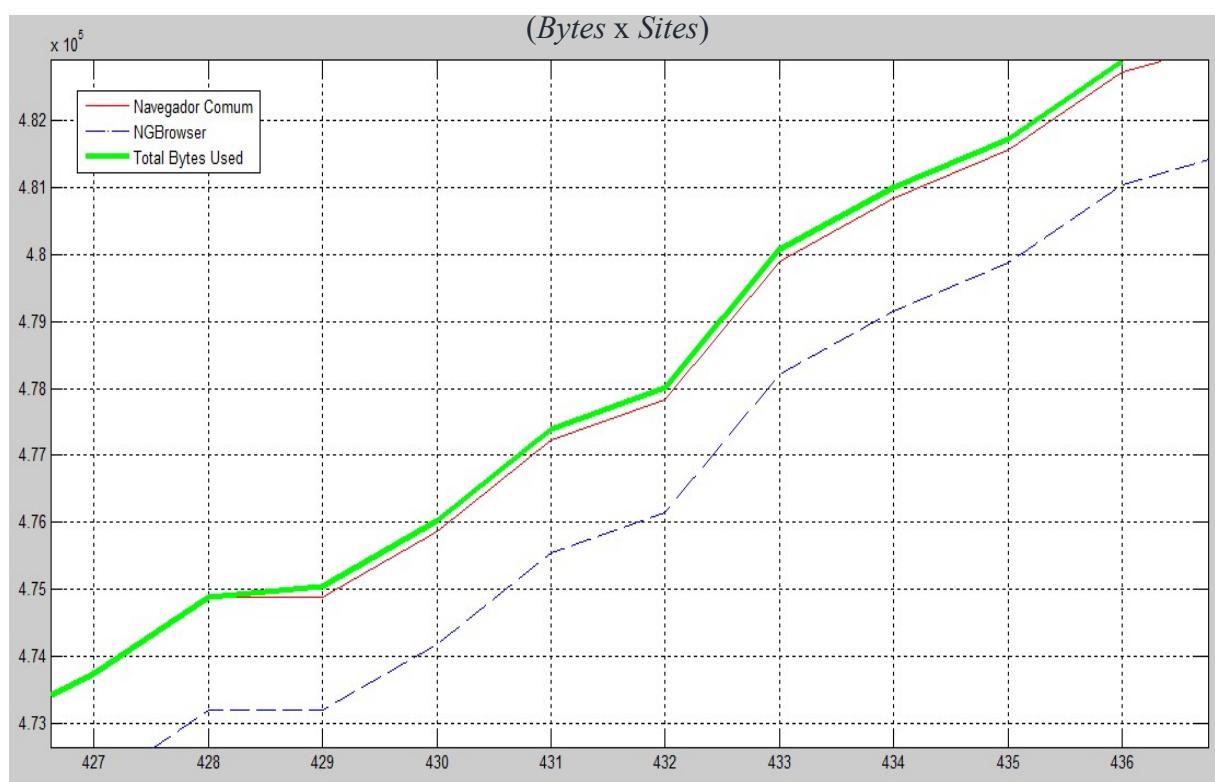


Figura 39 - Eficiência no uso do cache do navegador comum no teste em escala.

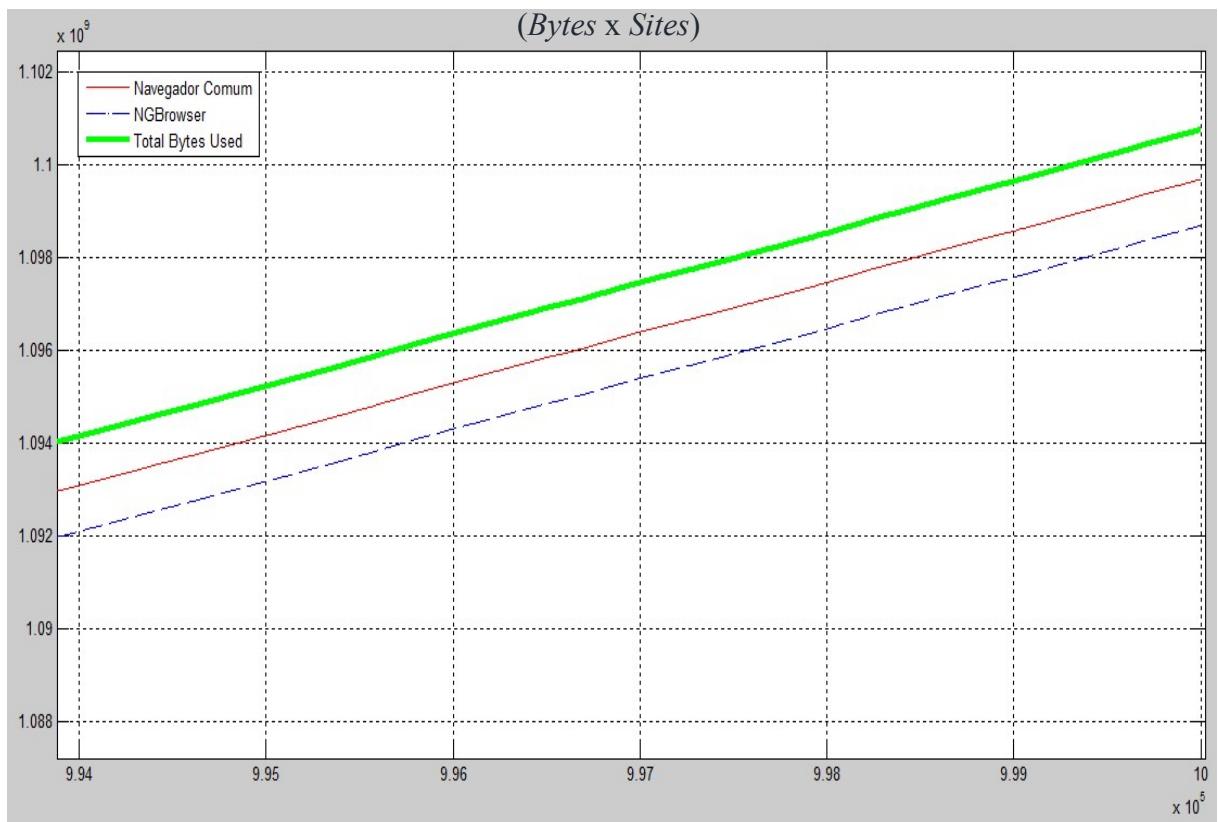


Figura 40 - Final da simulação do teste em escala.

A eficiência do *NGBrowser* começa a acontecer de forma antecipada devido à grande quantidade com que o usuário acessa páginas diferentes ($1-p_1$). Para cada página diferente a probabilidade de objetos repetidos (p_2), apesar de baixa, tem mais chance de acontecer do que a de acessar o mesmo *site* (p_1). Ao final da simulação, as retas convergem para terem a mesma distância entre si, porque as probabilidades (p_1 e p_2) possuem os mesmos valores.

Capítulo 6. Conclusão

Este trabalho reporta que é possível haver a navegação através de *websites* utilizando o modelo publica/assina proposto na NovaGenesis. Este modelo se mostrou eficaz, dispensando a necessidade da implementação do protocolo HTTP. Ferramentas podem ser desenvolvidas para facilitar a migração de *websites* de forma automática. O *NGAppCommunicator* é um processo que pode servir de exemplo para outros projetos de Internet do futuro com o âmbito de fornecer facilidades de acesso a diferentes redes através de APIs. Dentre as vantagens, estão a redução da complexidade de implementações de acesso a rede que os navegadores possuem e compartilhamento de um mesmo sistema de *cache*.

O uso de nomeação auto-certificável torna o sistema de *cache* eficiente devido a possibilidade de consultar conteúdos e certificá-los mesmo antes de realizar um *download*. Desta forma, é possível obter uma economia de dados significativa, conforme demonstrado em experimentos. Este valor depende da quantidade de repetições de conteúdo copiadas em diferentes *sites*. O sistema de *cache* dos navegadores atuais só são eficientes caso o conteúdo que os veiculam tenham a mesma URL [5]. A simulação do teste em escala demonstra que mesmo tendo baixo acesso a *sites* já acessados, o NGBrowser ainda apresenta eficiência sobre o navegador comum devido os *sites* possuírem objetos repetidos.

Os nomes auto-certificáveis ainda permitem que os dados sejam transferidos com verificação de integridade, pois conteúdos que tenham códigos *hash* diferentes são descartados pelo *NGAppCommunicator*. Este trabalho contribuí com a comunidade mediante a implementação deste *NGBrowser*. Futuramente, novos recursos serão implementados nesta aplicação para que ela alcance todas funcionalidades existentes hoje na Internet atual. Também, serão feitos testes em escalas maiores para aprofundar a análise de desempenho.

Referências

- [1] Pedro, E. e Junior, F. e Amorim, G. (2013), “Estudo e Análise da Proposta NovaGenesis para a Internet do Futuro”, NovaGenesis, Trabalho de Conclusão de Curso, p. 42-77.
- [2] Alberti, A. e Fernandes, V. e Casaroli, M. e Oliveira, L. e Júnior, F. e Singh, D. (2014), “A NovaGenesis Proxy/Gateway/Controller for OpenFlow Software Defined Networks”, NovaGenesis Architecture, p. 2-4.
- [3] Fielding, Reschke (2014) “HTTP/1.1 Message Syntax and Routing”, <http://tools.ietf.org/html/rfc7230#page-5>, Março 2016.
- [4] W3C (2014) “HTML5”, <https://www.w3.org/TR/html/introduction.html#a-quick-introduction-to-html>, Março 2016.
- [5] Ilya, G. (2014) “Armazenar HTTP em cache”, <https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching>, Março 2016.
- [6] die.net (1996) “Linux man page”, <http://linux.die.net/man/7/raw>, Março 2016.
- [7] Ghodsi, A. e Koponen, T. e Rajahalme, J. e Sarolahti, P. e Shenker, S. (2013), “Naming in Content-Oriented Architectures”, Introduction, p. 1-2.
- [8] Wieland (2015) “About Qt”, https://wiki.qt.io/About_Qt, Março 2016.
- [9] Qt (2015) “Qt Application Development”, <http://www.qt.io/application-development/>, Março 2016.
- [10] Qt (2015) “Qt D-Bus”, <http://doc.qt.io/qt-5/qtdbus-index.html>, Março 2016.
- [11] Qt (2015) “QtWebKit Guide”, <http://doc.qt.io/qt-4.8/qtwebkit-guide.html>, Março 2016.

- [12] Castells, M. (2003) “A galáxia da internet: Reflexões sobre a Internet, os negócios e a sociedade”, p.13-15.
- [13] Berners-Lee, T. (1990) “WorldWideWeb: Proposal for a HyperText Project”, <https://www.w3.org/Proposal>, Maio 2016.
- [14] Berners-Lee, T. e Fielding, R. e Frystyk, H. (1996) “Hypertext Transfer Protocol -- HTTP/1.0”, <http://www.hjp.at/doc/rfc/rfc1945.html>, Maio 2016.
- [15] Huston, G. (1999) “Web Caching”, <http://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-2/ipj-archive/article09186a00800c8903.html>, Maio 2016.
- [16] Feldmann A. (2007), “Internet Clean-Slate Design: What and Why?”, Introduction, p. 1-2.
- [17] Pan, J. e Paul, S. e Jain, R. (2011), “A Survey of the Research on Future Internet Architectures”, Introduction, p. 1-2.
- [18] Stuckmann, P. e Zimmermann, R. (2009), “European Research on Future Internet Design”, Introduction, p. 1-2.
- [19] Albert, R. e Jeong, H. e Barabási, A.L. (1999) “Diameter of the World-Wide Web”, p. 1.
- [20] Fielding R. e Gettys, J. e Mogul, J. e Frystyk, H. e Masinter, L. e Leach, P. e Berners-Lee, T. (1999) “Hypertext Transfer Protocol -- HTTP/1.1”, <https://www.ietf.org/rfc/rfc2616.txt>, Caching in HTTP, p. 74-99.
- [21] Grigorik, I. (2014) “Armazenar HTTP em cache”, <https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching?hl=pt-br>, Junho 2016.

- [22] Google Inc. (2014) "Aproveitar armazenamento em cache do navegador", <https://developers.google.com/speed/docs/insights/LeverageBrowserCaching#recomendaes>, Junho 2016.
- [23] Berners-Lee, T. e Masinter, L. e McCahill, M. (1994) "Uniform Resource Locators (URL)", <http://www.hjp.at/doc/rfc/rfc1738.html>, Agosto 2016.
- [24] Berners-Lee, T. (1994) "Universal Resource Identifiers in WWW", <http://www.hjp.at/doc/rfc/rfc1630.html>, Agosto 2016.
- [25] Sollins, K. e Masinter, L. (1994) "Functional Requirements for Uniform Resource Names", <http://www.hjp.at/doc/rfc/rfc1737.html>, Agosto 2016.
- [26] Alberti, A. e Moreira, W. e Righi, R. e Neto, F. e Dobre, C. e Singh, D. (2015), "Towards an Opportunistic, Socially-driven, Self-organizing, Cloud Networking Architecture with NovaGenesis", Introduction, p. 1-2.
- [27] Appleby, A. (2009) "MurmurHash", <https://sites.google.com/site/murmurhash/>, Agosto 2016.
- [28] Snoeren, A. e Balakrishnan, H. e Kaashoek, M. (2016), "Reconsidering Internet Mobility", Eliminate lower-layer dependence, p. 2.
- [29] Ahlgren, B. e Dannewitz, C. e Imbrenda, C. e Kutscher, D. e Ohlman, B. (2012), "A Survey of Information-Centric Networking", Introduction, p. 1.
- [30] Gelenbe, E. e Lent, R. e Montuori, A. e Xu, Z. (2002), "Cognitive Packet Networks: QoS and Performance", Introduction, p. 1.
- [31] Oliveira, L. (2015), "Contribuições no Desenvolvimento de Serviços de Representação, Controle e Alocação Dinâmica de Recursos de Substrato Para a NovaGenesis", Resource Management Agent, p. 34.

- [32] Alberti, A. e Moreira, W. e Righi, R. e Neto, F. e Dobre, C. e Singh, D. (2015), “Towards an Opportunistic, Socially-driven, Self-organizing, Cloud Networking Architecture with NovaGenesis”, Adapting NovaGenesis: Requirements and Challenges, p. 5-7.
- [33] Alam, S. e Chowdhury, M. e Noll, J. (2010), “SenaaS: An Event-driven Sensor Virtualization Approach for Internet of Things Cloud”, Introduction, p. 1
- [34] Berners-Lee, T. (2007) “Digital Future of the United States: Part I -- The Future of the World Wide Web”, <http://dig.csail.mit.edu/2007/03/01-ushouse-future-of-the-web.html>, Agosto 2016.
- [35] Berners-Lee, T. (1996) “The World Wide Web: Past, Present and Future”, <https://www.w3.org/People/Berners-Lee/1996/ppf.html>, Agosto 2016.
- [36] Iyer, S. e Rowstron, A. e Druschel, P. (2002), “Squirrel: A decentralized peer-to-peer web cache”, Introduction, p. 1
- [37] Fielding, R. e Gettys, J. e Mogul, J. e Frystyk, H. e Masinter, L. e Leach, P. e Berners-Lee, T. (1999) “Caching in HTTP”, <https://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html>, Agosto 2016.

ANEXO 1 – ALGORITMO EM MATLAB UTILIZADO NA SIMULAÇÃO

```
% Clear scope
clear;

% Simulation of Interaction
Interaction = 1000000;

% Max bytes per webpages
MaxBytesWebPages = 200;

% Max bytes for each object web
MaxBytesObjectWeb = 2000;

% Probabilities
% - Access the same webpage
ProbSameWebPage = 0.001;
% - Web object repeated
ProbRepeated = 0.001;

% Inicialization - Algorithm

% Number of webpages to simulate
NumWebPages = Interaction;
% Variables to save that web pages can
% access or have accessed
WebPagesToAccess = zeros(NumWebPages,1);
WebPagesAccessed = zeros(NumWebPages,1);

% Variables to plot
% Bytes Transferred
BytesTransferredAW = zeros(Interaction,1);
BytesTransferredNG = zeros(Interaction,1);
% Bytes used from each cache
BytesFromCacheAW = zeros(Interaction,1);
BytesFromCacheNG = zeros(Interaction,1);
% Bytes used in simulation
BytesUsed = zeros(Interaction,1);

% Bytes Transferred during the webpage access
% - For actual web
AWBytesTransferred = 0;
% - For NovaGenesis Web
NGBBytesTransferred = 0;

% Total Bytes Used from Cache
TotalBytesUsedInAWCache = 0;
TotalBytesUsedInNGCache = 0;
% Total Bytes Used
TotalBytesUsed = 0;

% Generation of webpages data
% RANDI - Uniformly distributed pseudorandom integers
WebPagesData = randi(MaxBytesWebPages,NumWebPages,1);
```

```

% For each webpage:
for i = 1:NumWebPages
    % - Assign a repeated object
    WebPagesData(i,2) = randi (MaxBytesObjectWeb);
    % - Set index to provide a webpage can be accessed
    WebPagesToAccess(i,1) = i;
end

% Size of webpages accessed
ka_size = 1;

% First Interaction of Simulation
% Get random webpage to access and Set it to accessed
kta = randi(size(WebPagesToAccess,1));
k = WebPagesToAccess(kta);
WebPagesAccessed(ka_size) = k;
ka_size = ka_size + 1;
WebPagesToAccess(kta,:) = [];
% Total Bytes Transferred by first interaction
BytesToTransfer = WebPagesData(k,1) + WebPagesData(k,2);
AWBytesTransferred = AWBytesTransferred + BytesToTransfer;
NGBBytesTransferred = NGBBytesTransferred + BytesToTransfer;
TotalBytesUsed = TotalBytesUsed + BytesToTransfer;
% To Plot
BytesTransferredAW(1) = AWBytesTransferred;
BytesTransferredNG(1) = NGBBytesTransferred;
BytesFromCacheAW(1) = TotalBytesUsedInAWCache;
BytesFromCacheNG(1) = TotalBytesUsedInNGCache;
BytesUsed(1) = TotalBytesUsed;

% Simulation for the rest of the interactions
for i = 1:Interaction-1
    UserAccess = rand();
    if( UserAccess <= ProbSameWebPage )
        % Access same webpage
        ka = randi(ka_size-1);
        k = WebPagesAccessed(ka,1);
        BytesInCache = WebPagesData(k,1) + WebPagesData(k,2);
        TotalBytesUsedInAWCache = TotalBytesUsedInAWCache + BytesInCache;
        TotalBytesUsedInNGCache = TotalBytesUsedInNGCache + BytesInCache;
        TotalBytesUsed = TotalBytesUsed + BytesInCache;
    else
        % Get random webpage to access and Set it to accessed
        kta = randi(size(WebPagesToAccess,1));
        k = WebPagesToAccess(kta);
        WebPagesAccessed(ka_size) = k;
        ka_size = ka_size + 1;
        WebPagesToAccess(kta,:) = [];

        ObjectRepeated = rand();
        if( ObjectRepeated <= ProbRepeated )
            % Accessing other webpage with objects repeated
            BytesToTransfer = WebPagesData(k,1) + WebPagesData(k,2);
            AWBytesTransferred = AWBytesTransferred + BytesToTransfer;
            TotalBytesUsed = TotalBytesUsed + BytesToTransfer;
            BytesToTransfer = WebPagesData(k,1);
            BytesInCache = WebPagesData(k,2);
            NGBBytesTransferred = NGBBytesTransferred + BytesToTransfer;
            TotalBytesUsedInNGCache = TotalBytesUsedInNGCache +
            BytesInCache;
        else

```

```

    % Accessing other webpage without objects repeated
    BytesToTransfer = WebPagesData(k,1) + WebPagesData(k,2);
    AWBytesTransferred = AWBytesTransferred + BytesToTransfer;
    NGBytesTransferred = NGBytesTransferred + BytesToTransfer;
    TotalBytesUsed = TotalBytesUsed + BytesToTransfer;
end
% Saving to plot
BytesTransferredAW(i+1) = AWBytesTransferred;
BytesTransferredNG(i+1) = NGBytesTransferred;
BytesFromCacheAW(i+1) = TotalBytesUsedInAWCache;
BytesFromCacheNG(i+1) = TotalBytesUsedInNGCache;
BytesUsed(i+1) = TotalBytesUsed;
end

%Plot
disp('Report: ')
disp(['- Interaction: ' num2str(Interaction)])
disp('- Bytes Transferred for:');
disp([' Actual Web: ' num2str(AWBytesTransferred)])
disp([' NovaGenesis Web: ' num2str(NGBytesTransferred)])
disp('- Bytes Used from:');
disp([' AWCache: ' num2str(TotalBytesUsedInAWCache)])
disp([' NGBrowser: ' num2str(TotalBytesUsedInNGCache)])
disp(['- Total Bytes Used: ' num2str(TotalBytesUsed)])
%disp('- Access Sequence of the Sites: ')
%disp(WebPagesAccessed);

% Set range and offset data
range = 0:1:Interaction;
BytesTransferredAW = [0;BytesTransferredAW];
BytesTransferredNG = [0;BytesTransferredNG];
BytesUsed = [0;BytesUsed];

plot(range,BytesTransferredAW,'r',range,BytesTransferredNG,'b',range,BytesUsed,'g');
%plot(range,BytesFromCacheAW,'b',range,BytesFromCacheNG,'r',range,BytesUsed,'g');

```