Uma proposta de endereçamento de rede aplicação-à-aplicação em arquiteturas clean-slate.

Alisson Oliveira Chaves

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2021

Alisson Oliveira Chaves

Uma proposta de endereçamento de rede aplicação-à-aplicação em arquiteturas clean-slate.

Dissertação de mestrado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Ph.D Flavio de Oliveira Silva

Uberlândia
2021

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

C512p
2021

Chaves, Alisson Oliveira, 1989-
Uma proposta de endereçamento de rede aplicação-à-aplicação em arquiteturas clean-slate [recurso eletrônico] / Alisson Oliveira Chaves. - 2021.

Orientador: Flavio de Oliveira Silva.
Dissertação (mestrado) - Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação.
Modo de acesso: Internet.
Disponível em: http://doi.org/10.14393/ufu.di.2022.5315
Inclui bibliografia.
Inclui ilustrações.

1. Computação. I. Silva, Flavio de Oliveira, 1970-, (Orient.). II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU: 681.3

Glória Aparecida
Bibliotecária - CRB-6/2047

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Coordenação do Programa de Pós-Graduação em Ciência da Computação
Av. João Naves de Ávila, nº 2121, Bloco 1A, Sala 243 - Bairro Santa Mônica, Uberlândia-MG, CEP 38400-902
Telefone: (34) 3239-4470 - www.ppgco.facom.ufu.br - cpgfacom@ufu.br

ATA DE DEFESA - PÓS-GRADUAÇÃO

Programa de Pós-Graduação em:	Ciência da Computação				
Defesa de:	Mestrado Acadêmico, 12/2021, PPGCO				
Data:	30 de junho de 2021	Hora de início:	09h35min	Hora de encerramento:	12h55min
Matrícula do Discente:	11812CCP004				
Nome do Discente	Alisson Oliveira Chaves				
Título do Trabalho:	Uma proposta de endereçamento de rede aplicação-à-aplicação em arquiteturas clean-slate				
Área de concentração:	Ciência da Computação				
Linha de pesquisa:	Sistemas de Computação				
Projeto de Pesquisa de vinculação:	-				

Reuniu-se, por videoconferência, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Ciência da Computação, assim composta: Professores Doutores: João Henrique de Souza Pereira - FACOM/UFU; Antônio Marcos Alberti - INATEL e Flávio de Oliveira Silva - FACOM/UFU, orientador do candidato.

Os examinadores participaram desde as seguintes localidades: Antônio Marcos Alberti - Santa Rita do Sapucaí/MG; João Henrique de Souza Pereira e Flávio de Oliveira Silva - Uberlândia/MG. O discente participou da cidade de Uberlândia/MG.

Iniciando os trabalhos o presidente da mesa, Prof. Dr. Flávio de Oliveira Silva, apresentou a Comissão Examinadora e o candidato, agradeceu a presença do público, e concedeu ao Discente a palavra para a exposição do seu trabalho. A duração da apresentação do Discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir o senhor presidente concedeu a palavra, pela ordem sucessivamente, aos examinadores, que passaram a arguir o candidato. Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando o candidato:

Aprovado.

Esta defesa faz parte dos requisitos necessários à obtenção do título de Mestre.

O competente diploma será expedido após cumprimento dos demais requisitos, conforme as normas do Programa, a legislação pertinente e a regulamentação interna da UFU.

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.

Documento assinado eletronicamente por **Flávio de Oliveira Silva**, **Professor(a) do Magistério Superior**, em 02/07/2021, às 16:54, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **João Henrique de Souza Pereira**, **Professor(a) do Magistério Superior**, em 02/07/2021, às 17:33, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **2870263** e o código CRC **6E39749D**.

Referência: Processo nº 23117.041919/2021-84 SEI nº 2870263

Este trabalho é dedicado àqueles que pouco sabem ou para àqueles que muito sabem, desde que anseiem a busca por aprender a cada dia mais.

Agradecimentos

Agradeço a Deus, coluna forte e base da minha existência. Agradeço também ao professor Pedro Frosi por me apresentar ao mundo da pesquisa. Agradeço ao professor Flávio Oliveira pela paciência em me guiar por este caminho difícil porém recompensador que é a pesquisa. Ao Centro de Tecnologia da Informação da UFU nas pessoas dos diretores Pedro Frosi e Luiz Fernando Faina, em especial, aos meus colegas de trabalho pelo apoio e incentivo e a todos aqueles que contribuíram para que este trabalho fosse executado.

"Quanto mais elaborados os nossos meios de comunicação são, menos comunicamos."
(Joseph Boython Priestey)

Resumo

O modelo de endereçamento presente nas redes atuais e que existe a cerca de 40 anos, tornou-se um padrão e segue o modelo de referência Open Systems Interconnection (OSI). Nesse sentido, temos um tipo diferente de endereçamento em cada camada. O modelo de endereçamento faz parte de algumas pesquisas sobre novas arquiteturas de Internet do futuro. Esses modelos permitem o uso unificado dos diferentes endereços utilizados, reduzindo a complexidade das redes de computadores e melhorando o suporte para a comunicação de novas aplicações nessas redes. Este trabalho visa contribuir para uma simplificação do processo de endereçamento na comunicação entre aplicações através do uso de uma interface de rede virtual para cada socket de aplicação. Com uma abordagem experimental, mostramos como este trabalho pode beneficiar arquiteturas de rede clean-slate como Entity Title Architecture (ETArch) e Named Data Network (NDN), e também demonstramos melhorias quantitativas em relação ao modelo Transmission Control Protocol (TCP)/Internet Protocol (IP).

Palavras-chave: Internet do Futuro. Endereçamento. Redes de computadores.

Abstract

The addressing model presented in current networks has been around for 40 years, has become a standard, and follows the OSI Reference Model. In this sense, we have a different type of addressing at each layer. The addressing model is part of some research on new Internet architectures of the future. These models allow the unified use of different addresses used, reducing computer networks' complexities and improving support for new applications' communication needs on these networks. This work aims to simplify the addressing process in communication between applications through the use of a virtual network interface for each application socket. With an experimental approach, we show how our work can benefit clean-slate network architectures like ETArch and NDN, and also demonstrate quantitative improvements over the TCP/IP.

Keywords: Future Internet. Addressing. Network computing.

Lista de ilustrações

Lista de tabelas

Lista de siglas

API Application Programming Interface

ARP Address Resolution Protocol

ASCII American Standard Code for Information Interchange

A2A Application-to-Application

B2B Business-to-Business

DNS Domain Name System

DTSA Domain Title Service Agents

ETArch Entity Title Architecture

FTP File Transfer Protocol

IANA Internet Assigned Numbers Authority

IP Internet Protocol

IPv6 Internet Protocol Version 6

MAC Media Access Control

MEHAR Mondial Entities Horizontally Addressed by Requirements

MIT Massachusetts Institute of Technology

MPLS Multi Protocol Label Switching

MTU Maximum Transmission Unit

NDN Named Data Network

NFD Named Data Networking Forwarding Daemon

NIC Network Interface Card

OSI Open Systems Interconnection

PDU Protocol Data Unit

PIT Pending Interest Table

QoS Quality of Service

RFC Request for Comments

RIR Regional Internet Registries

SRI-NIC Stanford Research Institute - Network Information Center

SO Sistema Operacional

TCP Transmission Control Protocol

VM Virtual Machine

vNIC Virtual Network Interface Card

VPN Virtual Private Network

Sumário

Capítulo 1

Introdução

O início do conceito de endereçamento em redes de computadores se deu através da necessidade humana de armazenar informações em sua memória. Na década de 70 já se tinha computadores clientes e alguns poucos servidores oferecendo serviços à rede de uma empresa. A tarefa de decorar um endereço IP destes servidores era de certa forma trivial.

Com o passar do tempo, a informatização global sistematizou processos através de soluções computacionais que demandaram a criação de mais servidores dentro das redes. Assim a tarefa de memorizar endereços IP começou a se tornar mais complexa sendo necessária uma solução mais simples onde endereços pudessem ser identificados com um nome (CARVALHO, 2013).

Por vários anos o endereçamento IP foi adaptado para suportar as necessidades tecnológicas que foram aparecendo, e que em sua raiz não foi projetado para suportar, mais sua flexibilidade permitia. Porém as limitações de desempenho, confiabilidade, escalabilidade e segurança, são algumas dos problemas que podem ser observados neste modelo de arquitetura.
Projetos de Internet do futuro tentar desenvolver métodos e arquiteturas de rede que possam superar as limitações atuais, permitindo o desenvolvimento de aplicações mais avançadas em termos tecnológicos (GOLDSMITH et al.,).

Propostas para desenvolvimento de novas arquiteturas de Internet do futuro foram publicadas demonstrando uma boa definição sobre os papéis do plano de controle e plano de dados, e em quais dispositivos influencia. Juntamente com essas propostas, o modelo de endereçamento tem sido redesenhado para suportar novas aplicações e reduzir a complexidade do endereçamento.

A tarefa de localizar um elemento participante em uma rede consiste em definir um modelo de endereçamento que possa ser estruturado com caminhos, origens e destinos, enquanto a identificação deve nomear o participante de forma única dentro de uma rede. Na atual arquitetura de Internet os conceitos de identificação (nome) e localização (endereço) de um dispositivo, estão contidos na camada de rede, e mais especificamente no endereço IP, o que torna a escalabilidade e mobilidade um desafio.

Os projetos de internet do futuro tem como característica, o desacoplamento destas funções de rede permitindo assim uma infraestrutura de roteamento altamente escalável, com mobilidade e funções de relacionamento *one-to-one* e *one-to-many* (CHOI et al., 2008).

O uso de cabeçalhos nas camadas de rede, aumentam a quantidade de dados não utilizados pela camada de aplicação no tráfego, elevando o uso de recursos computacionais para desencapsular o pacote e a tomada de decisão de encaminhar o pacote para o socket do sistema operacional (KUROSE; ROSS, 2017).

O desacoplamento da identificação e localização, manter uma forma simples de identificar um elemento na rede e, reduzir a quantidade de dados em comparação a atual arquitetura de internet, são desafios de pesquisa.

1.1 Motivação

Conforme citado anteriormente, as novas propostas de arquiteturas de Internet do futuro, trazem novos modelos de endereçamento com possibilidades de redução no uso de camadas do modelo de referência TCP/IP. Porém abordam somente a comunicação entre os dispositivos. Portanto, a principal motivação desta pesquisa foi investigar maneiras de reduzir ainda mais o uso dessas camadas, levando o processo de comunicação diretamente aos atores principais: as aplicações clientes e aplicações servidores.

1.2 Objetivos e Desafios da Pesquisa

O objetivo geral do projeto foi explorar novos métodos de envio de dados de aplicações em arquiteturas de Internet do futuro. Através do conhecimento obtido em trabalhos anteriores, foi possível verificar uma lacuna em relação ao socket da aplicação.
Ainda em análise de pesquisas sobre o assunto, foi possível observar que a utilização de virtualização de recursos de rede é um caminho que poderia trazer resultados para este trabalho.

De forma mais específica, a pesquisa focou nas seguintes ações:

- ❏ Investigar modelos de endereçamento em arquiteturas de Internet do futuro;
- ❏ Analisar ferramentas de virtualização de recursos de rede;
- ❏ Implementar um ambiente de experimentação para testes atuais e futuros trabalhos relacionados;
- ❏ Implementar um modelo de endereçamento que englobe todos os componentes em uma comunicação de rede;
- ❏ Reduzir o *overhead* do endereçamento e facilitar a comunicação entre aplicações.

Um grande desafio da pesquisa, foi desenvolver, implementar e testar toda a solução em ambientes reais, sem a utilização de ferramentas de simulação, ambientes virtualizados ou equipamentos virtualizados.

1.3 Hipótese

Este trabalho assume que é possível reduzir a complexidade do endereçamento e o overhead de dados de controle na rede através da criação de uma Virtual Network Interface Card (vNIC) específica para cada canal de comunicação entre aplicações cliente e aplicações servidor - um serviço conhecido como socket.

1.4 Contribuições

As aplicações práticas dessa solução são refletidas na forma do envio de dados pelas aplicações, contribuindo assim com o suporte no desenvolvimento de aplicações agnósticas à rede, reduzindo o overhead de dados de controle na rede, simplificando o endereçamento das aplicações e aumentando o número de hosts disponíveis na Internet através da utilização de nomes para identificação de um elemento na rede. A solução nativamente provê interoperabilidade entre diferentes arquiteturas e aplicações, desenvolvidas para possíveis ambientes heterogêneos.

1.5 Organização da Dissertação

O restante deste trabalho está organizado do seguinte modo: Capítulo 2 onde é apresentado um breve histórico do uso de nomes como representação do endereçamento, os conceitos das arquiteturas ETArch, NDN e trabalhos correlatos. O capítulo 3 descreve o modelo de endereçamento aplicação-à-aplicação proposto neste trabalho. O capítulo 4 apresenta os detalhes dos experimentos realizados e uma análise dos resultados. E por fim o capítulo 5 traz as conclusões deste trabalho, suas contribuições e indicações de futuros trabalhos.

Capítulo 2

Fundamentação Teórica

Neste capítulo, vamos falar sobre os conceitos, arquiteturas e análise de trabalhos correlatos usados como princípios no desenvolvimento deste projeto.

No trabalho YANAIL (CHUN; LEE; CHOI, 2011), é explorado uma definição para os conceitos de nomes, endereços, identificadores e localizadores. De forma resumida, o nome é um título qualquer, entregue a qualquer entidade pertencente à uma rede e o endereço é um rótulo que define a posição onde essas entidades estão na rede. Caso estes rótulos sejam usados como identificadores únicos e inequivocados, podemos chamá-los de identificadores. Os símbolos usados para definir um ponto específico em uma rede são chamados localizadores.

Algumas funcionalidades presentes no Domain Name System (DNS), foram utilizadas como base na definição do conceito de identificação de um elemento de rede em nossa pesquisa.

O endereço Media Access Control (MAC), utilizado como base para definição da forma de localização, traz um aumento na quantidade de elementos de rede que podem fazer parte dessa arquitetura, em relação ao atual endereçamento IP.

2.1 Endereçamento em TCP/IP e IEEE 802

O endereçamento IP resolve o problema de se encontrar um dispositivo presente na Internet. Ele representa usando números decimais o endereçamento na camada de rede do modelo de referência OSI. Em sua versão 4 permite aproximadamente 4 bilhões de endereços e desde 2014 vem se esgotando, assim após a criação da versão 6, temos mais de 340 undecilhões de endereços possíveis. Porém, o uso do novo protocolo só é possível através da atualização de todos os dispositivos e sistemas operacionais presentes na Internet, o que tornou o processo lento. Ainda hoje, existem várias redes que não possuem endereços Internet Protocol Version 6 (IPv6) (KUROSE; ROSS, 2017).

Com um tamanho de 65536 possibilidades, a porta é conhecida por representar através de números decimais, o contexto de uma aplicação dentro de um dispositivo. É conside-

rado um endereço de camada de aplicação, porém, também é utilizado pela camada de transporte. Juntamente com o endereço IP eles formam o socket, responsável por permitir a comunicação de diferentes aplicações entre um ou vários dispositivos cliente/servidor de maneira simultânea (KUROSE; ROSS, 2017).

O endereço MAC, que possui mais de 281 trilhões de endereços, representa o endereçamento da camada de enlace do modelo OSI através de números hexadecimais e, por definição, tem seu uso controlado para fabricantes de dispositivos de rede registrados, onde um único endereço MAC é disponibilizado para cada Network Interface Card (NIC) fabricada no mundo (KUROSE; ROSS, 2017).

2.2 Domain Name System

Datada do ano de 1982, uma solução foi introduzida pelo arquivo *hosts.txt*, facilitando a tarefa de decorar endereços IP para a grande quantidade de serviços presentes nas redes. É um arquivo de texto que contém uma lista correlacionando um endereço IP a um nome qualquer. Essa simples solução era implementada na Internet para todos os dispositivos conectados, e a sincronização era feita através da transferência do arquivo que ficava hospedado nos servidores do Stanford Research Institute - Network Information Center (SRI-NIC). A solução poderia ser utilizada também nas redes locais (MOCKAPETRIS; DUNLAP, 1988).

O conceito de endereçamento é bastante utilizado na tecnologia da informação, desde posições na memória, filas de processos, até armazenamento de informações em um disco. Após a solução ter sido definida, além do contínuo crescimento da quantidade de serviços disponibilizados pela rede, houve um grande crescimento nos tipos de componentes a serem endereçados em uma rede, como switches e roteadores, literalmente seria possível ter no mínimo um nome para cada um dos 4.294.967.296 endereços IP disponíveis no mundo.

A tarefa de gerenciar, manter, organizar e distribuir essa lista, se tornou um processo complexo e de alto custo para as redes. A transferência desse arquivo poderia causar degradação nos backbones da Internet, além da grande carga de trabalho aplicada no servidor do SRI-NIC e também a quantidade de alterações no arquivo *hosts.txt*. (MOCKAPETRIS; DUNLAP, 1988)

Trabalhos publicados no final da década de 70 e início da década de 80 originaram as Request for Comments (RFC) 882 e 883, que definem os conceitos, especificações e modelo de implementação de nomes de domínio, que não substituem o uso do arquivo *hosts.txt*, mais sim definem um padrão global de uso. (MOCKAPETRIS; DUNLAP, 1988).

Destas definições até hoje, o DNS passou por várias melhorias, porém, para este trabalho, levantamos algumas funcionalidades.

2.2.1 Linguagem

A linguagem adotada para representatividade digital da informação é baseada nos sinais representados na tabela American Standard Code for Information Interchange (ASCII), restringido à caracteres globalmente utilizados na grande maioria dos idiomas e reservas de alguns caracteres para funções especiais. Não existem limites de tamanho para os nomes de domínio.

2.2.2 Hierarquia

No modo de funcionamento hierárquico e em árvore, que além de possibilitar segmentar a gerência para os ramos, também nos permite consultas diretas em ramos, o que diminui a necessidade de consultar toda lista de relação nome-domínio, isso aumenta o desempenho da resolução de nomes e diminui a carga nos servidores de domínio e backbone da rede.

Os ramos são organizados através do uso do carácter ponto [.], por exemplo:
mehar.facom.ufu.br, onde *mehar* é um ramo de *facom* que faz parte do ramo *ufu* que participa do tronco *br*.

2.2.3 Exclusividade

O princípio da exclusividade foi necessário para o serviço de DNS garantindo um acesso único, imparcial e seguro. Raiz, troncos, ramos, sub-ramos e todas suas entradas devem ser únicas e exclusivas em toda a Internet. Para que o processo de registro desses nomes fosse organizado, o Internet Assigned Numbers Authority (IANA) foi criado, que, dentre várias funções, controla o registro de endereços IP na Internet através de Regional Internet Registries (RIR) e gerencia a atribuição dos servidores raiz de nomes de domínio, isso, de maneira profissional, justa e neutra (IANA. . . , 2021).

2.3 Arquitetura: Entity Title Architecture (ETArch)

O projeto de arquitetura ETArch utiliza o conceito de atribuição de títulos para identificar os componentes presentes em uma comunicação. A seguir, são definidas: as *entidades* que são os participantes da comunicação, o *título* que é a forma única de identificação independente da topologia e o *workspace* que é o meio de interligação para uma comunicação (CORUJO et al., 2013) (SILVA et al., 2014).

Tecnicamente, a implementação do ETArch é baseada no endereçamento de *entidade* e *workspace*. Através de uso de campos de endereçamento disponíveis na camada de enlace do modelo de referência OSI, as *entidades* e *workspace* são representados. Uma abstração das camadas de rede e transporte é realizada, simplificando o tráfego na rede e trazendo

ganhos em relação à performance e endereçamento. A Figura 1 demonstra a base da arquitetura ETArch e seus componentes.

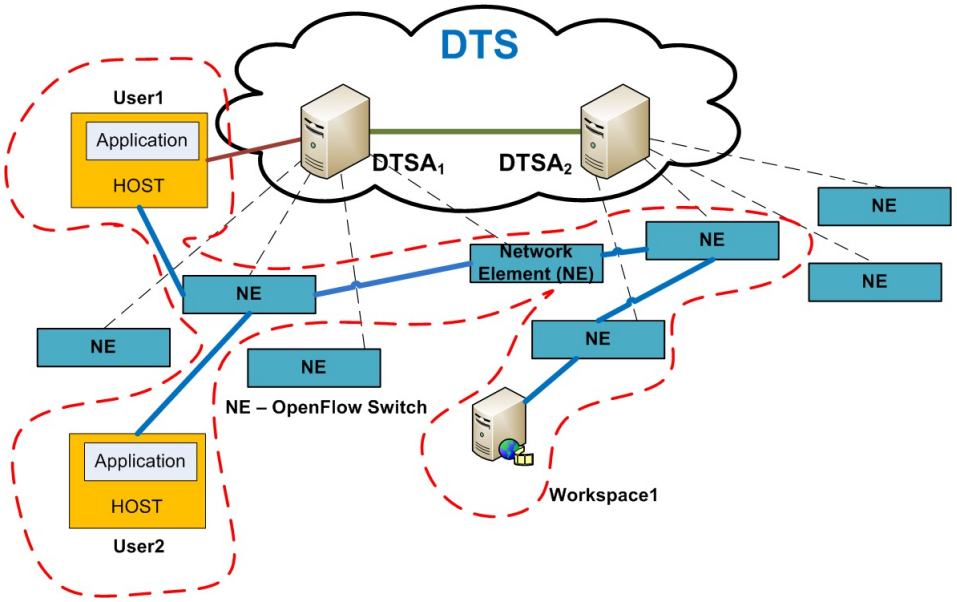

Figura 1 – Arquitetura ETArch. (SILVA, 2013)

Utilizando da implementação deste projeto, foram feitas contribuições para a definição de um modelo de endereçamento.

2.4 Arquitetura: Named Data Network (NDN)

Para o desenvolvimento do NDN alguns conceitos foram utilizados, como permitir que as tecnologias das camadas presentes no modelo de referência TCP/IP possam evoluir sem restrições, trazer uma segurança nativa através de assinatura de todos os pacotes e utilizar o princípio de ponta-a-ponta que permite o desenvolvimento de aplicações tolerante à falhas.

Como elementos da arquitetura, são definidos os elementos: *nomes* que tem sua nomenclatura sem significados e estruturada de forma hierárquica e exclusiva para um espaço de trabalho específico. O *roteamento e encaminhamento* são feitos através do nome eliminando assim problemas com esgotamento, NAT, mobilidade e gerenciamento de endereços. O *cache* de conteúdo que pode ser entregue à partir dos espaços de *buffer* dos roteadores trazendo ganhos em relação à performance. A *Pending Interest Table (PIT)* que traz o conceito de uma tabela de relação entre interface e *interests* agindo como uma tabela Address Resolution Protocol (ARP) e o *transporte*, aqui não tratado como uma camada separada, possui característica de entrega não confiável por padrão (ZHANG et al., 2010).

Com o objeto de testar a possibilidade de se aplicar este trabalho em diversas arquiteturas que não a ETArch, utilizando do Named Data Networking Forwarding Daemon (NFD), uma implementação do NDN, foram realizados testes de interoperabilidade.

2.5 Trabalhos Correlatos

Existem diferentes pesquisas relacionadas à Internet do Futuro. Vários trabalhos têm proposto um novo modelo de endereçamento em redes.

O trabalho de Seoul, (CHOI et al., 2008) propõe uma divisão na semântica dos localizadores e identificadores de uma rede, que hoje fazem parte do endereçamento da camada de rede. Essa união lógica faz com que as conexões se percam quando o endereço IP é modificado para atualizar a localização do dispositivo. O desacoplamento lógico das funções de localização e identificação traz melhorias no suporte à mobilidade multi-homing.

Geoff Houston (HUSTON, 2007) apresenta de maneira prática as propriedades Exclusividade (identificação única de entidades), Consistência, Persistência, Confiança e Robustez como essenciais para o funcionamento da rede. Ele descreve que a característica da combinação de funções presentes no endereço IP de *quem* (identificador), *onde*(localizador), não somente torna o uso da rede mais eficiente, mais também são a causa da complexidade criada pela necessidade de se entregar as funções descritas na atual arquitetura existente na Internet de hoje. Problemas atuais como mobilidade e a falta de identificação dos caminhos completos através da rede são desafios de pesquisa.

O projeto NDN (ZHANG et al., 2010) propõe a utilização de identificadores hierarquicamente estruturados em todos elementos de comunicação da rede. Em substituição, os nomes assumem o papel de identificação e podem referenciar os dados de maneira mais simples. Em um de seus princípios, ponta-a-ponta, permite o desenvolvimento de aplicações mais robustas em caso de falhas na rede. Algumas características são: nomes exclusivos e hierárquicos de comprimento variável e o endereçamento por nomes (títulos).

Pesquisadores do Massachusetts Institute of Technology (MIT) (SALTZER; REED; CLARK, 1984) apresentam o princípio do argumento ponta-a-ponta, que sugere que as funções aplicadas à rede podem ser redundantes ou de alto custo, visto que em muitos serviços disponibilizados na Internet hoje, realizam estas funções diretamente na camada de aplicação. Funções como garantia de entrega, transmissão segura de dados, supressão de mensagens duplicadas e identificação dos pontos são exemplos de possível uso desnecessário de recursos que realizamos hoje.

O mecanismo *macvlan*, um driver disponível no módulo de rede do kernel Linux permite uma comunicação à nível de endereçamento da camada de enlace e camada de rede em sistemas de virtualização e conteinerização das funções de um computador (processamento, armazenamento e entrada e saída de dados), e é descrito em diversos trabalhos, dos quais podemos citar (CLAASSEN; KONING; GROSSO, 2016), que demonstra em

testes de benchmark, que o *macvlan* performa melhor do que outros drivers semelhantes e também disponíveis no kernel Linux. Testbeds de pesquisas sobre roteamento e troca de tráfego foram adaptadas para cenários de conteinerização, mitigando a degradação de performance do tráfego no plano de dados (JUNIOR et al., 2018). No trabalho de (RATHORE; HIDELL; SJÖDIN, 2011), a proposta de estudos sobre virtualização de roteadores em ambientes virtualizados, conclui que o método *macvlan* é o mais promissor em relação à redução de overhead na rede.

O conceito aplicação-à-aplicação é descrito na pesquisa sobre desenvolvimento web (CURBERA; NAGY; WEERAWARANA, 2001), onde descreve a evolução da Internet, e demonstra que desde a antiga interação homem-aplicação-aplicação, até as atuais interações aplicação-à-aplicação através de marketplaces, transações Business-to-Business (B2B) automáticas e o compartilhamento de recursos, levantam questões que promovem a discussão sobre interoperabilidade entre aplicações, representação das informações e a padronização de requisitos relacionados ao tipo de comunicação descrito.

Propostas no desenvolvimento de novas arquiteturas *clean-slate* como MobiliyFirst (SESKAR et al., 2011), NEBULA (ANDERSON et al., 2013) e ETArch (SILVA, 2013), foram publicadas, demonstrando uma boa definição sobre os papéis do plano de controle e plano de dados e em quais dispositivos influencia.

Porém quando olhamos todos os envolvidos na comunicação, essas propostas estão focadas no endereçamento para dispositivos de rede, não contemplando o Sistema Operacional (SO) do dispositivo final e mais precisamente as aplicações disponíveis no dispositivo, obrigando assim, as aplicações que consomem serviços da rede a conhecerem as estruturas da rede e também a terem uma maior interação com o SO à nível de identificações na camada de aplicação.

Capítulo 3

Endereçamento A2A

Projetos de Internet do futuro, como ETArch, reduziram a quantidade de cabeçalhos para somente um, presente na camada de enlace, porém, por não interferir nos protocolos da camada de aplicação, os cabeçalhos desta camada devem ainda existir para que o tráfego seja redirecionado para a correta aplicação dentro do dispositivo.

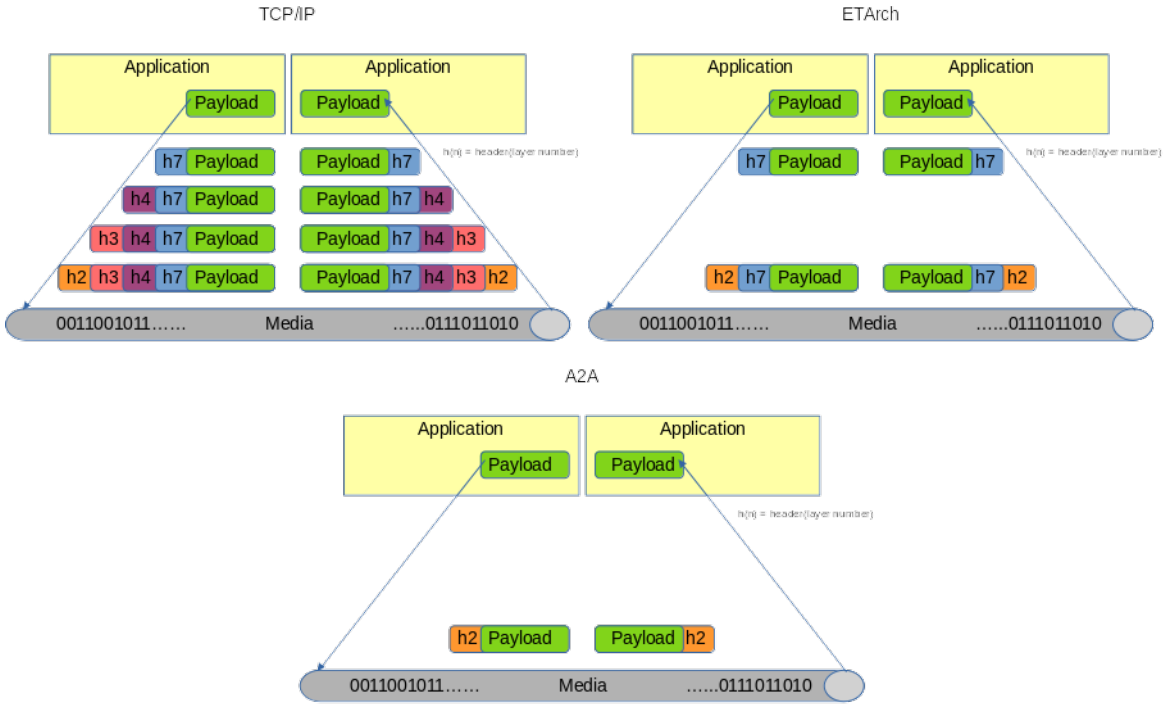

Figura 2 – Pilha de protocolos.

A figura 2 demonstra que na arquitetura TCP/IP os cabeçalhos das camadas 7, 4, 3 e 2 do modelo de referência OSI são incluídas ao payload no processo de comunicação entre a aplicação de origem e aplicação de destino. Na arquitetura ETArch, ao payload é adicionado os cabeçalhos das camadas 7 e 2. No endereçamento A2A, a proposta é que se crie um canal único de comunicação entre as aplicações, não sendo mais necessário adicionar cabeçalhos da camada 7.

Visando tornar a usabilidade em novas tecnologias de Internet do futuro mais simples, neste trabalho, contemplamos uma comunicação efetiva entre as novas arquiteturas de rede e os sistemas operacionais, de modo a prover transparência no tráfego de dados de aplicações.

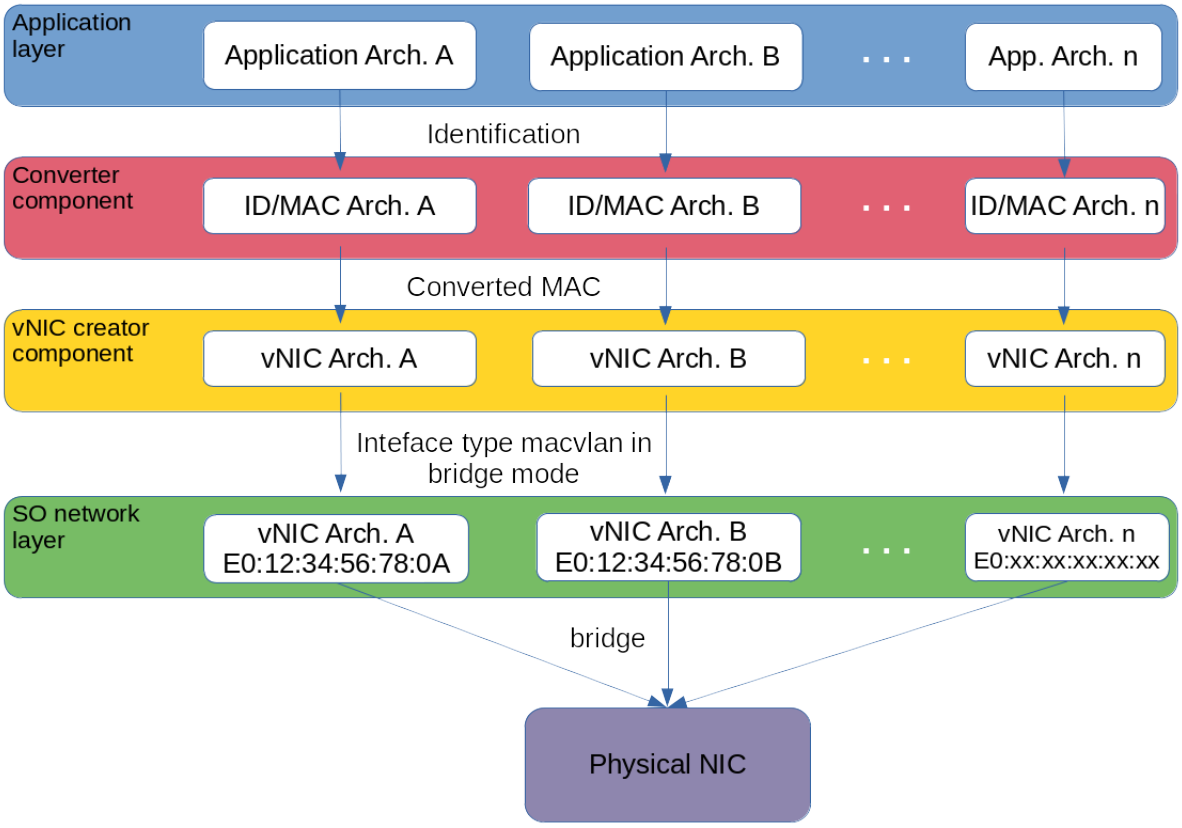

Figura 3 – Componentes do Endereçamento A2A.

A figura 3 apresenta uma visão geral dos componentes da proposta deste trabalho, chamada Endereçamento A2A. Nesta abordagem, separamos os componentes em camadas com suas respectivas funções.

Nos componentes da camada aplicação, temos várias aplicações desenvolvidas para diferentes tipos de arquiteturas. Como vimos na seção 2.2, para melhor identificação das aplicações, foi feita a escolha de uma linguagem globalmente adotada - ASCII - que combinada seja capaz de ter tamanhos diversos e que possa ser organizada de maneira hierárquica, o que melhora a performance de consultas e, para evitar duplicidades na identificação, também é necessário que este seja único.

A identificação da camada de aplicação é enviada para a camada de componentes de conversão, onde será convertida em um endereço MAC.

Durante anos, para suportar novas aplicações na arquitetura TCP/IP foram feitas várias adaptações, como por exemplo o uso de mais campos de controle em suas camadas, que assim reduz a eficiência da rede em transportar informação do usuário. Com a utilização do campo de endereçamento disponível no protocolo Ethernet, é possível reduzir

o uso destes campos de controle. Pesquisas sobre Internet do futuro, tendem a tornar o uso desta pilha de protocolos mais limpa.

A tabela 1 resume o estudo detalhado realizado em trinta e dois tipos de interface de comunicação de rede disponíveis no kernel (KERNEL..., 2021) do Linux em busca de um modelo que permita a comunicação de diferentes endereços de camada 2 (Ethernet) em cima de uma única interface física. O modelo *macvlan* exclusivamente trabalha da forma citada e foi utilizado na camada de componente de criação da vNIC que relaciona uma interface virtual com uma das aplicações descritas nos componentes da camada de aplicação. Por exemplo, para a aplicação de envio de envio de mensagens instantâneas teríamos um endereço MAC representado na interface física e para outra aplicação como por exemplo, de transmissão de vídeo, teríamos outro endereço MAC na mesma interface física.

Tabela 1 – Lista de tipos de interface de rede.

TIPO	**DESCRIÇÃO**
bridge	ponte Ethernet entre hosts, gateways e Virtual Machine (VM)
bond	agregação lógica de interfaces
dummy	teste local da pilha TCP/IP
hsr	redundância contínua em alta disponibilidade
ifb	concentrador para Quality of Service (QoS)
ipoib	protocolo IP sobre interfaces *InfiniBand*
macvlan	**interface virtual com endereço na camada 2**
macvtap	interface virtual com endereço na camada 2 tunelada
vcan	teste de barramento físico
vxcan	túnel entre vcan's
veth	tunel Ethernet local
vlan	rede virtual por tag de pacote (4 bits)
vxlan	rede virtual por tag de pacote (24 bits)
ip6tnl	encapsulamento IPv4 / IPv6 sobre IPv6
ipip	túnel IP sobre IP
sit	interconectar redes IPv6 em redes IPv4 globais
gre	tunelamento de tráfego
gretap	tunelamento de tráfego
erspan	tunelamento de tráfego com roteamento
ip6gretap	tunelamento de tráfego
ip6erspan	tunelamento de tráfego com roteamento
vti	tunelamento de tráfego
nlmon	monitor Netlink do sistema
ipvlan	igual a macvlan com mesmo endereço MAC nos terminais
ipvtap	interface virtual com endereço na camada 2 tunelada
lowpan	redes pessoais sem fio de baixa potência
geneve	tunelamento de tráfego
macsec	segurança na camada 2
vrf	virtualização de tabelas de roteamento (Virtual Private Network (VPN), Multi Protocol Label Switching (MPLS))
netdevsim	teste de Application Programming Interface (API) de rede
rmnet	framework ethernet para dispositivos móveis
xfrm	framework de modificação de pacotes

A publicação da identificação na rede, significa que ela se torna conhecida para inter-

faces físicas e virtuais em outras máquinas. Na camada de rede do SO, a identificação da camada de aplicação e sua conversão em endereço MAC é representada na informação da vNIC nos campos de nome e endereço MAC respectivamente.

O endereço MAC é usado para criar a vNIC do tipo *macvlan* em modo *bridge* que realiza uma ponte para cada comunicação da aplicação entre a interface virtual e a interface física garantindo que a identificação da interface virtual seja publicada na rede e agora os dados são enviados na interface física disponível no *host*.

Para novas aplicações funcionarem em paralelo nessa solução independente da arquitetura da rede, é necessário que as definições especificadas nas camadas citadas sejam executadas, e que o envio dos dados no SO se dê pela interface virtual referente à aplicação específica.

Sem ferir as funcionalidades descritas na seção 2.2, é proposto a criação de uma vNIC para cada socket de aplicação dentro do sistema operacional do dispositivo. Assim, quando uma aplicação invocar o uso de um socket, essa vNIC é criada e usada única e exclusivamente para esta aplicação.

Como o conceito de sockets (IP+porta) foi abstraído, o tráfego de uma aplicação através dessa vNIC pode ser em modo *raw*, ou seja, enviar somente os dados úteis referentes ao contexto da aplicação.

Por exemplo, após acessar o socket da aplicação e criar a vNIC, uma aplicação de chat pode trafegar somente os bits referentes ao texto da mensagem enviada. A gerência e uso dos sockets de rede do sistema operacional ficaria mais simples por não ser necessário manter informações de porta de origem e de destino, e o volume de dados para a mesma aplicação em comparação com as redes atuais seria menor, o que provê um acesso mais rápido aos serviços disponíveis nessa arquitetura.

Capítulo 4

Experimentos e Análise dos Resultados

Neste capítulo demonstramos que a solução proposta neste trabalho pode funcionar com arquiteturas de rede *clean-slate* como citadas na seção 2.3. Para realizar a avaliação experimental de nosso projeto, selecionamos a arquitetura de rede ETArch proposta por nosso grupo de pesquisa e também fizemos um teste de interoperabilidade com a arquitetura NDN.

4.1 Método para a Avaliação

A proposta de endereçamento aplicação-à-aplicação, define a possibilidade de não utilizar campos de cabeçalhos de pacotes em um socket da aplicação. Para avaliação da solução, os pacotes trafegados foram capturados com todos os dados possíveis sendo possível observar desde marcação de tempo, até quantidade de dados de controle.

Os testes executados com a implementação da arquitetura ETArch tem como objetivo comparar a quantidade de dados enviados e o desempenho de envio em arquiteturas de rede atual com o modelo proposto neste trabalho.

O funcionamento da solução proposta foi comprovada através do uso da arquitetura ETArch, e também, com a utilização de uma implementação da arquitetura NDN foi possível comprovar a interoperabilidade com outras arquiteturas de Internet do futuro.

4.2 Ambiente de Testes

O ambiente de testes para a proposta possui todos os participantes necessários para o completo funcionamento de uma rede ETArch. O Domain Title Service Agents (DTSA) (SILVA, 2013) está hospedado no datacenter de pesquisa do Mondial Entities Horizontally Addressed by Requirements (MEHAR) (MEHAR..., 2021). Em uma outra rede conectada através da Internet, foram usados 1 x roteador sem fio TPLink TL-WR1043ND com o SO modificado para OpenWRT 18.06 e 1 x roteador sem fio CISCO Linksys E9000 com o SO modificado para OpenWRT 18.06, ambos com a instalação do pacote OpenVSwitch

e protocolo OpenFlow 1.0 habilitado. Conectados em cada um desses roteadores sem fio, há um computador rodando sistema operacional Linux OpenSuSE Leap 15.0 64bits, onde foram implementadas as aplicações desenvolvidas. As configurações dos computadores estão descritas na tabela 2.

Tabela 2 – Configuração dos equipamentos utilizados nos testes.

Host	Processador	Memória
alisson:	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz	SODIMM DDR4 2400MHz de 8GB
hp:	Intel(R) Core(TM)2 Duo CPU T6600 @ 2.20GHz	SODIMM DDR3 800MHz de 4GB

A figura 4 detalha o modelo de ligação entre os dispositivos usados nos testes. Os links entre os roteadores e computadores estão trabalhando a 100 Mbps.

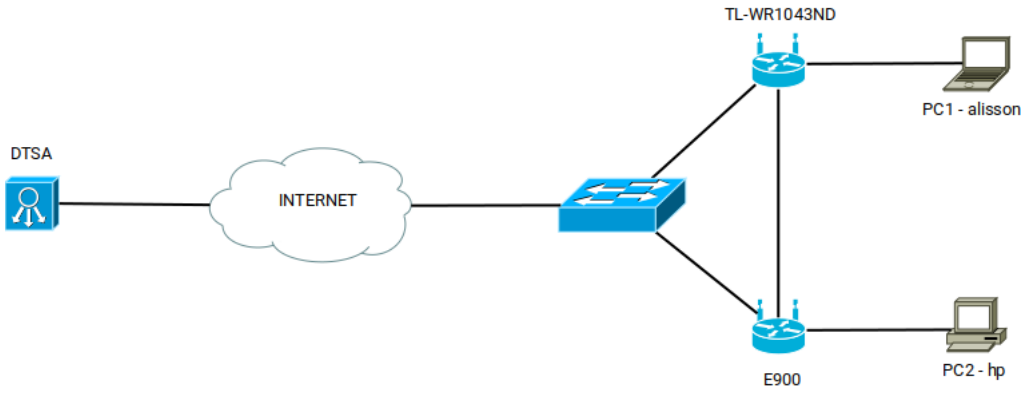

Figura 4 – Ambiente de testes.

4.3 Experimentos

Os campos de endereçamento disponíveis na arquitetura ETArch são os campos MAC de destino e MAC de origem, que possui um tamanho de 48 bits. Desses 48 bits, o sétimo bit é reservado para identificar se o endereço é globalmente único ou localmente administrado e o oitavo bit é usado para verificar se o endereço é unicast ou multicast.

Utilizando o sistema operacional Linux OpenSuSE Leap 15.0 64bits, foi definido o uso da biblioteca de virtualização de driver de rede chamada *macvlan*. A *macvlan* foi desenvolvida no intuito de criar um canal de comunicação entre a NIC física e máquinas virtuais (REDHAT..., 2021). Posteriormente os testes foram realizados com sucesso nas versões 15.1 e 15.2 do Linux OpenSuSE Leap.

Para este trabalho, o conceito de nomes de domínio foi usado nos *títulos* e *workspaces* com todas suas funções de linguagem, hierarquia e exclusividade. Um arquivo de configuração indica o título do dispositivo e em qual interface de rede física, a vNIC será atachada.

O software criado, lê essas informações e utiliza uma biblioteca de conversão de nome de domínio em endereço MAC para criar a vNIC correspondente ao socket da aplicação.

A biblioteca criada reserva os primeiros 8 bits como *1110 0000* que resultará em hexadecimal *E0*, e converte o nome de domínio em hexadecimal através da utilização da função de hash não segura com padrão *CRC32 CRC16* (MILLER; VANDOME; MCBREWSTER, 2009) com saída de 40 bits. Totalizando assim os 48 bits presentes nos campos de endereçamento da camada de enlace.

Após a criação do workspace, a aplicação deve realizar uma chamada informando o nome do workspace ao software de conversão que terá como entrada "<título_do_dispositivo>+<workspace>" e como saída um endereço de 48 bits.

Neste ponto o software desenvolvido cria uma vNIC do tipo *macvlan* em modo *bridge* e assim a aplicação pode abrir um socket e enviar os dados em modo *raw* através desta interface.

4.3.1 Experimento 1 (ETArch)

A figura 5 mostra a proposta de envio de dados *raw* em um socket de aplicação-à-aplicação. No dispositivo *alisson.ufu.br*, temos duas aplicações, e cada aplicação tem dois serviços diferentes, e para cada serviço uma vNIC correspondente com endereço único foi criada. Esses serviços comunicam com o dispositivo do cliente com acesso às respectivas aplicações e serviços disponibilizados. Assim, neste dispositivo, um pacote de dados tem como destino a identificação única do serviço entregue pela aplicação.

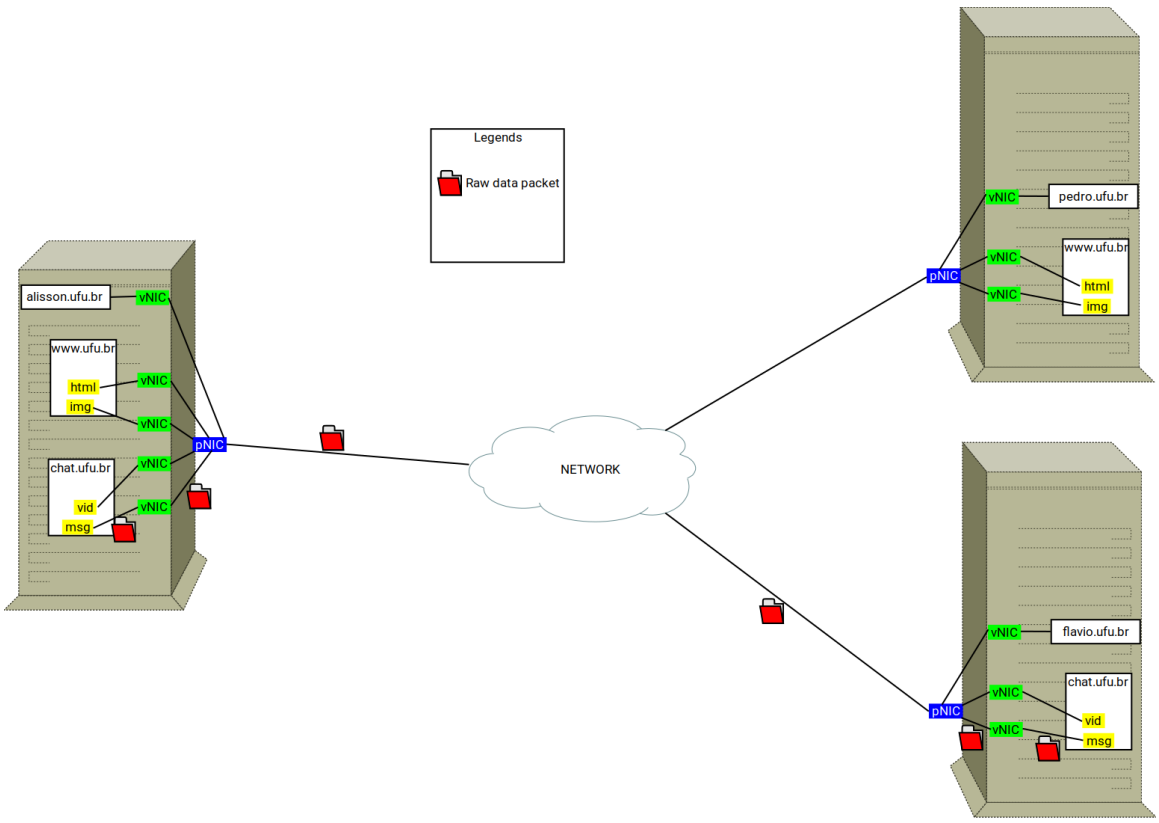

Figura 5 – Envio de dados *raw* em um socket aplicação-à-aplicação.

Para os testes, foi criada uma aplicação de chat simples, que envia através da interface destinada àquele socket de aplicação em modo *raw socket*, somente o texto puro a ser enviado na mensagem.

Para realização dos testes, criamos os componentes *vNIC Creator* e *vNIC Converter*. Também utilizamos um aplicativo de chat ETArch que realiza o envio de dados puros. A figura 6 mostra através do terminal do Linux, a execução do software desenvolvido em dois hosts. Ele verifica o arquivo de configuração, cria e verifica a interface relacionada à identificação do dispositivo. O software também cria e verifica o *workspace* e realiza a troca de mensagens na aplicação de chat.

```
alisson:~ # cat /etc/etarch/etarch.conf
p7p1 alisson.ufu.br
alisson:~ # etarch-int start
alisson:~ # ifconfig alisson.ufu.br
alisson.ufu.br: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::e233:aaff:fe1e:894b  prefixlen 64  scopeid 0x20<link>
        ether e0:33:aa:1e:89:4b  txqueuelen 1000  (Ethernet)
        RX packets 9  bytes 2546 (2.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 25  bytes 4524 (4.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

alisson:~ # etarch-ws start chat.ufu.br
alisson:~ # ifconfig chat.ufu.br
chat.ufu.br: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::e22d:3eff:fec4:a5c9  prefixlen 64  scopeid 0x20<link>
        ether e0:2d:3e:c4:a5:c9  txqueuelen 1000  (Ethernet)
        RX packets 15  bytes 3615 (3.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 24  bytes 4429 (4.3 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

alisson:~ # etarch-chat chat.ufu.br
RTNETLINK answers: File exists
Send a message:
First msg

Received Message:
Second msg

Send a message:
Third msg

Received Message:
Fourth msg

Send a message:
```

```
hp:~ # cat /etc/etarch/etarch.conf
eth0 flavio.ufu.br
hp:~ # etarch-int start
hp:~ # ifconfig flavio.ufu.br
flavio.ufu.br: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::e2d1:25ff:fe68:2a9a  prefixlen 64  scopeid 0x20<link>
        ether e0:d1:25:68:2a:9a  txqueuelen 1000  (Ethernet)
        RX packets 7  bytes 1610 (1.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 19  bytes 3402 (3.3 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

hp:~ # etarch-ws start chat.ufu.br
hp:~ # ifconfig chat.ufu.br
chat.ufu.br: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::e265:2dff:fe0b:9efb  prefixlen 64  scopeid 0x20<link>
        ether e0:65:2d:0b:9e:fb  txqueuelen 1000  (Ethernet)
        RX packets 7  bytes 1930 (1.8 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 18  bytes 3240 (3.1 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

hp:~ # etarch-chat chat.ufu.br
RTNETLINK answers: File exists
Send a message:
Second msg

Received Message:
Third msg

Send a message:
Fourth msg
```

Figura 6 – Configuração, execução e teste do chat no terminal.

4.3.2 Experimento 2 (NDN)

A solução proposta por este trabalho foi comprovada através da utilização da arquitetura ETArch, porém, como forma de comprovar a interoperabilidade desta solução com outras arquiteturas de Internet do futuro, utilizando as ferramentas disponibilizados no website do projeto NDN na versão 0.7.1 (NFD. . . , 2021), foi realizado um teste de interoperabilidade com a arquitetura. Para realização do testes, foi utilizada a estrutura descrita na seção 4.2, as aplicações desenvolvidas para este projeto e as aplicações desenvolvidas para o NFD.

Foi realizada a instalação da biblioteca *ndn-cxx*, a instalação e configuração da aplicação NFD que tem como funcionalidade a criação e manutenção de *canais*, *faces* e *rotas* e, a instalação da aplicação *ndn-traffic-generator* usada para simulações de tráfego na rede.

Uma das várias funcionalidades da arquitetura NDN é a definição do modelo de endereçamento, que assim como neste trabalho, permite uma simplificação no endereçamento

da rede. Outra funcionalidade é que o NFD permite o envio de dados através do protocolo Ethernet.

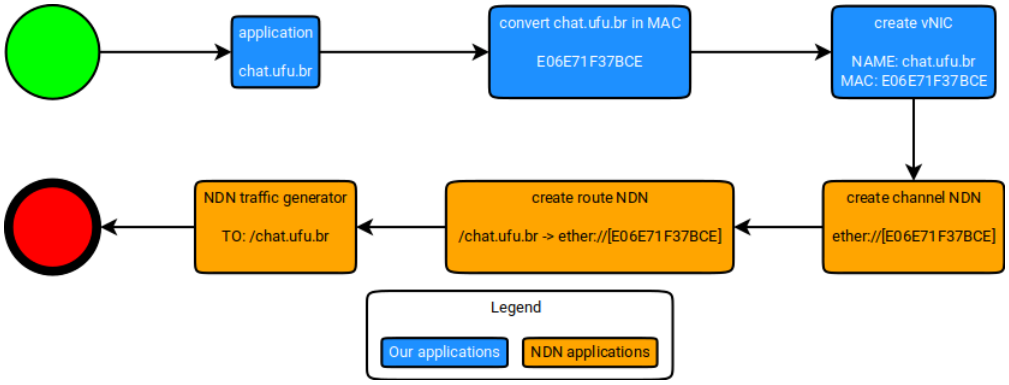

Figura 7 – Teste de interoperabilidade com NDN.

Um script para automação do teste foi desenvolvido utilizando as aplicações descritas na seção 4.3.1. A figura 7 descreve os passos do script criado onde, em cada computador, utilizando as aplicações propostas, é criado a interface referente ao dispositivo e a interface referente ao socket aplicação-à-aplicação da ferramenta de chat. Através das aplicações desenvolvidas no projeto NDN, foi criado o canal de comunicação com encaminhamento de pacotes para a interface aplicação-à-aplicação e adicionado uma rota usando o endereçamento NDN para o canal.

Para os testes foi utilizada a aplicação *ndn-traffic-generator* para transferência de dados através do endereçamento NDN.

4.4 Avaliação dos Resultados

Para avaliação dos resultados dos testes feitos com a arquitetura ETArch, foram realizados desde uma captura dos pacotes para contabilização, até uma análise da classificação entre dados de controle e dados puros.

As figuras 8 e 9 demonstram a captura dos pacotes enviados pela aplicação de chat em ambos os hosts, como visto, não é necessário o processo de *padding* da mensagem até completar 46 bytes no campo de dados, de acordo com os padrões do protocolo Ethernet (KUROSE; ROSS, 2017), visto que o envio é feito em modo “raw socket”, permitindo assim uma redução no envio de dados pela rede.

Como teste comparativo, foi utilizado o protocolo File Transfer Protocol (FTP) que realiza o envio de dados de forma trivial. A tabela 3 detalha a construção do tráfego transmitido, que são classificados como dados de controle e dados puros em uma transmissão FTP.

Como teste quantitativo, uma aplicação de envio de arquivos puros foi desenvolvida e utilizada. A tabela 4 detalha os dados transmitidos a partir da solução proposta por este

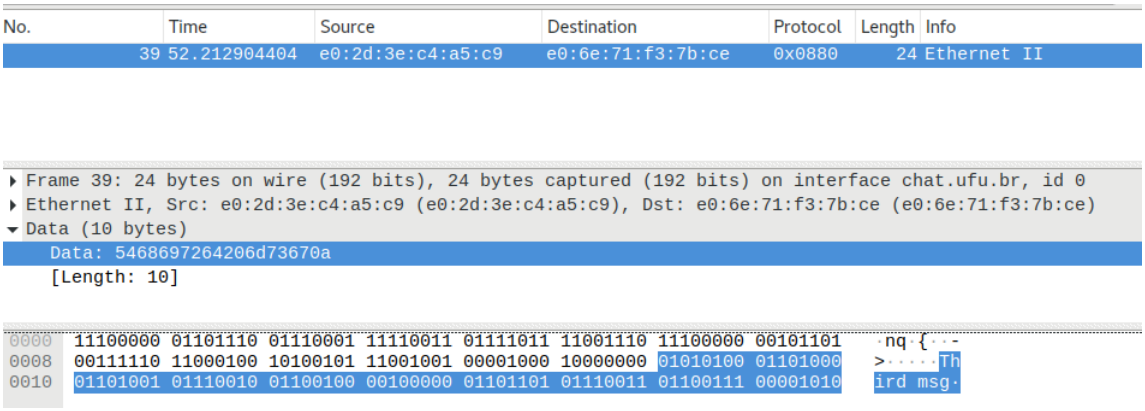

Figura 8 – Captura de pacotes enviados no primeiro computador (lab.alisson).

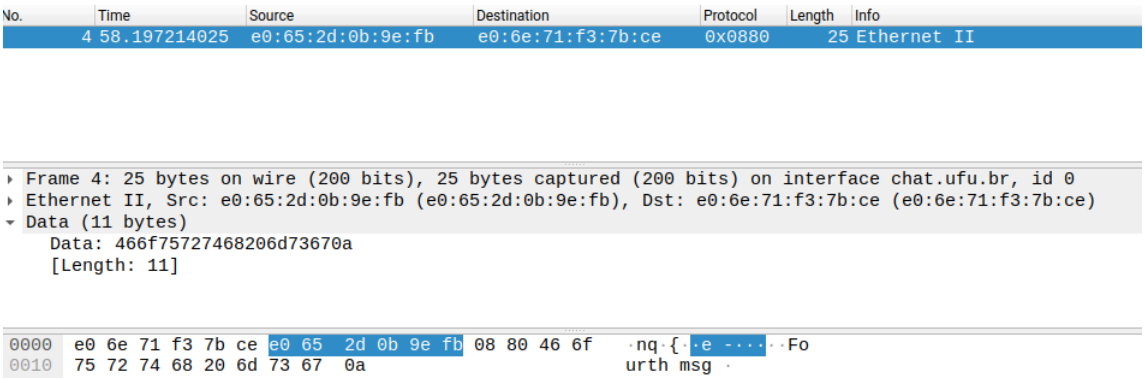

Figura 9 – Captura de pacotes enviados no segundo computador (lab.hp).

projeto. Para uma melhor acurácia nos resultados os valores são baseados em uma média ponderada de 10 execuções.

A figura 10 compara a quantidade de dados enviados pela rede utilizando o FTP x nossa proposta, em referência às informações contidas no arquivo original que será enviado. A quantidade de dados de controle do protocolo, mais o tamanho do Protocol Data Unit (PDU) das camadas do modelo TCP/IP representam o total de dados enviados na transmissão: 6,40%, 18,08% e 55,88%, com tamanhos de Maximum Transmission Unit (MTU) de 1500 bytes, 500 bytes e 130 bytes respectivamente.

No modelo proposto, não temos dados de controle de protocolo, e os campos usados combinados com o PDU das camadas utilizadas representam o total de dados enviados na transmissão: 0.92%, 2,72% e 9.72%, com tamanhos de MTU de 1500 bytes, 500 bytes e 130 bytes respectivamente.

A figura 11 compara o tempo de envio gasto para transmissão dos dados do mesmo arquivo através do FTP e da nossa proposta. Para uma melhor análise em cada cenário, este experimento foi replicado dez vezes (F1 a F10) e (A1 a A10), as médias são apresentadas utilizando um intervalo de confiança *T-Student* de 95%. Os tempos médios observados para o envio FTP são: $14.499s \pm 0.0085s$, $16.379s \pm 0.007s$ e $29.458s \pm 0.0248s$, com tamanhos de MTU de 1500 bytes, 500 bytes e 130 bytes respectivamente e os tempos médios

Tabela 3 – Envio FTP.

AVERAGE AMOUNT	SIZE PACKET	PDU SIZE (Bytes)	RAW DATA SIZE (Bytes)	TOTAL TRANSMITTED (Bytes)	% PDU per PACKET	AVERAGE TIME IN SECs	% CONTROL+PDU
MTU 1500							
117396	1514	66	1448	177737215	4.36	14.50	6.40
FTP CONTROL				2777			
TCP CONTROL				3874418			
TOTAL Bytes TRANSMITTED ->				**181614410**	**169990465**	**<- ORIGINAL FILE**	
MTU 500							
379350	514	66	448	194985829	12.84	16.38	18.08
FTP CONTROL				2645			
TCP CONTROL				12459026			
TOTAL Bytes TRANSMITTED ->				**207447500**	**169990465**	**<- ORIGINAL FILE**	
MTU 130							
2158875	144	66	78	310878025	45.83	29.46	55.88
FTP CONTROL				2778			
TCP CONTROL				70761458			
TOTAL Bytes TRANSMITTED ->				**381652261**	**169990465**	**<- ORIGINAL FILE**	

observados para o envio em nossa proposta são: $13.928s \pm 0.0129s$, $14.619s \pm 0.0186s$ e $15.567s \pm 0.0242s$, com tamanhos de MTU de 1500 bytes, 500 bytes e 130 bytes respectivamente.

A redução média do tempo gasto com envio dos dados no endereçamento A2A em relação ao envio FTP foi de: 3.93%, 10.74% e 40.36%, com tamanhos de MTU de 1500 bytes, 500 bytes e 130 bytes respectivamente.

Vislumbrando um futuro onde a solução proposta seja de fato um padrão para as redes de Internet do futuro, uma questão levantada, seria qual a quantidade possível de criação e uso de vNIC.

Em análise nos hosts descritos na seção 4.2, com padrões de uso comum do dispositivo por um administrador de redes, foi visto o uso aproximadamente de 300 conexões simultâneas. Em análise da literatura, para a biblioteca que cria a interface *macvlan* não existe limitações, porém a caráter de estudos, foi realizado o teste de criação de 10.000 vNIC nos dois hosts descritos na seção 4.2 com sucesso.

Um outro teste voltado para o desempenho da aplicação de criação de vNIC desen-

Tabela 4 – Envio utilizando Endereçamento A2A.

AVERAGE AMOUNT	SIZE PACKET	PDU SIZE (Bytes)	RAW DATA SIZE (Bytes)	TOTAL TRANSMITTED (Bytes)	% PDU per PACKET	AVERAGE TIME IN SECs	% CONTROL+PDU
MTU 1500							
113327	1514	14	1500	171577043	0.92	13.93	0.92
TOTAL Bytes TRANSMITTED ->				**171577043**	**169990465**	**<- ORIGINAL FILE**	
MTU 500							
339981	514	14	500	174750199	2.72	14.62	2.72
TOTAL Bytes TRANSMITTED ->				**174750199**	**169990465**	**<- ORIGINAL FILE**	
MTU 130							
1307619	144	14	130	188297131	9.72	17.57	9.72
TOTAL Bytes TRANSMITTED ->				**381652261**	**169990465**	**<- ORIGINAL FILE**	

volvida neste projeto foi realizado nos hosts descritos na seção 4.2. Para o host *alisson* o desempenho está em 41 vNIC por segundo, e para o host *hp* o desempenho está em 18 vNIC por segundo.

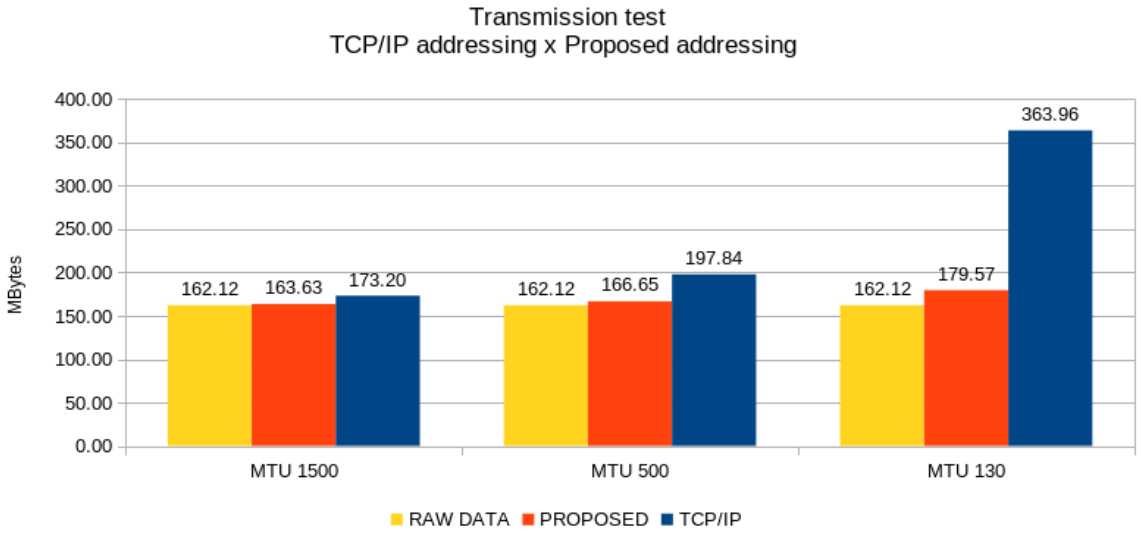

Figura 10 – Teste de transmissão. Endereçamento TCP/IP x Endereçamento A2A.

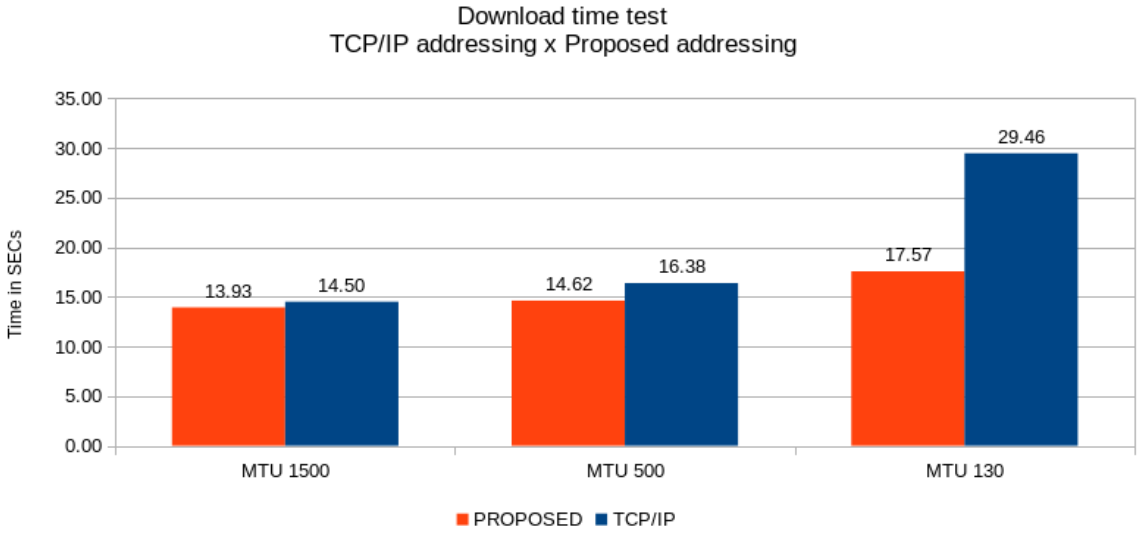

Figura 11 – Teste de tempo de transmissão. Endereçamento TCP/IP x Endereçamento A2A.

Capítulo 5

Conclusão

Esse capítulo apresenta as conclusões do trabalho proposto e as contribuições técnico e científicas alcançadas durante seu desenvolvimento. O objetivo geral de explorar novos métodos de envios de dados de aplicações em arquiteturas de Internet do futuro é alcançado e materializado na criação do Endereçamento A2A, um modelo que permite uma efetiva comunicação de aplicação à aplicação.

A literatura correlata foi revisada no intuito de que este modelo contribua nos pontos ainda não explorados na pesquisa e trouxe como carga de conhecimento, modelos de interfaces, métodos de endereçamento e métodos de virtualização de rede. Diversas propostas de Internet do futuro trazem um conceito de arquitetura limpa da rede, porém não ampliam o endereçamento além do host, até as aplicações ou instâncias das aplicações.

A solução demonstrada no capítulo 3 foi construída de acordo com os objetivos especificados na seção 1.2 e todos seu desenvolvimento demonstrado no capítulo 4. Em todos aspectos relacionados ao projeto como métodos, modelos, tecnologias, ferramentas, foi possível absorver e fixar grande aprendizado sobre os temas discutidos e principalmente sobre métodos de endereçamento.

O Endereçamento A2A propõe um modelo de endereçamento que cria um canal exclusivo de comunicação entre aplicações à nível da rede, o que reduz a complexidade da rede e diminui a quantidade de dados não-úteis trafegados, concluindo assim, que os objetivos foram alcançados.

5.1 Principais Contribuições

Como contribuição, um modelo de endereçamento único foi definido em substituição de toda a pilha TCP/IP através dos campos atualmente disponíveis em um protocolo ethernet, obtendo redução no tamanho mínimo de um frame ethernet. Como objetivo, também definimos um modelo para o desenvolvimento de novas aplicações em que é necessário, apenas enviar os dados puros através da interface de rede relacionada ao nome do socket da aplicação. Utilizando a aplicação desenvolvida, criamos um meio que

permite a criação e manutenção dessas interfaces. Uma aplicação simples de chat para uma arquitetura de rede *clean-slate* foi utilizada para testar as premissas deste trabalho e atingir os objetivos esperados.

5.2 Trabalhos Futuros

Trabalhos futuros devem adaptar o sistema de manutenção de interface como um módulo do sistema operacional Linux reavaliando o código e o tornando mais performático. O desenvolvimento de aplicações mais complexas em arquiteturas cliente/servidor seria um poderoso objeto de estudo. Outro objetivo futuro é aplicar este modelo a outras arquiteturas de *clean-slate*, garantindo assim a interoperabilidade entre os conceitos dos projetos. Outra possibilidade seria realizar testes em redes legadas e na atual Internet, para verificar a aplicabilidade dos conceitos aqui definidos com a carga histórica do desenvolvimento da Internet.

5.3 Contribuições em Produção Bibliográfica

Os resultados deste trabalho foram a base para a escrita do artigo intitulado *"A proposal for application-to-application network addressing in clean-slate architectures"* (CHAVES; ROSA; SILVA, 2021), o qual foi aceito para publicação nos anais da conferência *AINA - Advanced Information Networking and Applications* (Qualis A2 CAPES). O artigo foi apresentado em maio de 2021 e está disponível nos anais da conferência a partir dessa data.

Referências

ANDERSON, T. et al. The nebula future internet architecture. In: SPRINGER. **The Future Internet Assembly**. [S.l.], 2013. p. 16–26. https://doi.org/10.1007/978-3-642-38082-2_2.

CARVALHO, S. L. d. Dnssec: extensões de segurança para servidores dns. Universidade Tecnológica Federal do Paraná, 2013.

CHAVES, A. O.; ROSA, P. F.; SILVA, F. O. A proposal for application-to-application network addressing in clean-slate architectures. In: BAROLLI, L.; WOUNGANG, I.; ENOKIDO, T. (Ed.). **Advanced Information Networking and Applications**. Cham: Springer International Publishing, 2021. p. 783–793. ISBN 978-3-030-75075-6. https://doi.org/10.1007/978-3-030-75075-6_64.

CHOI, J. et al. Addressing in future internet: Problems, issues, and approaches. **Proc 3rd International CFI**, Citeseer, 2008.

CHUN, W.; LEE, T.-H.; CHOI, T. Yanail: yet another definition on names, addresses, identifiers, and locators. In: **Proceedings of the 6th International Conference on Future Internet Technologies**. [S.l.: s.n.], 2011. p. 8–12. https://doi.org/10.1145/2002396.2002399.

CLAASSEN, J.; KONING, R.; GROSSO, P. Linux containers networking: Performance and scalability of kernel modules. In: IEEE. **NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium**. [S.l.], 2016. p. 713–717. https://doi.org/10.1109/NOMS.2016.7502883.

CORUJO, D. et al. Enabling Network Mobility by Using IEEE 802.21 Integrated with the Entity Title Architecture. In: . [S.l.: s.n.], 2013.

CURBERA, F.; NAGY, W.; WEERAWARANA, S. Web services: Why and how. In: **Workshop on Object-Oriented Web Services-OOPSLA**. [S.l.: s.n.], 2001. v. 2001.

GOLDSMITH, M. H. et al. Clean-slate design for the internet.

HUSTON, G. Addressing the future internet. 2007.

IANA — About us. 2021. Disponível em: <https://www.iana.org/about>.

JUNIOR, B. et al. High-fidelity interdomain routing experiments. In: **Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos**. [S.l.: s.n.], 2018. p. 36–38. https://doi.org/10.1145/3234200.3241324.

KERNEL - The Linux Kernel documentation. 2021. Disponível em: <https://www.kernel.org/doc/html/latest/>.

KUROSE, J.; ROSS, K. **Computer networking : a top-down approach**. 2017. Disponível em: <https://cds.cern.ch/record/2252697>.

MEHAR - Mondial Entities Horizontally Addressed by Requirements. 2021. Disponível em: <http://mehar.facom.ufu.br>.

MILLER, F. P.; VANDOME, A. F.; MCBREWSTER, J. **Cyclic Redundancy Check: Computation of CRC, Mathematics of CRC, Error detection and correction, Cyclic code, List of hash functions, Parity bit, Information ... Cksum, Adler-32, Fletcher's checksum**. [S.l.]: Alpha Press, 2009. ISBN 9786130219741.

MOCKAPETRIS, P.; DUNLAP, K. J. Development of the domain name system. In: **Symposium proceedings on Communications architectures and protocols**. [S.l.: s.n.], 1988. p. 123–133. https://doi.org/10.1145/52324.52338.

NFD - Named Data Networking Forwarding Daemon. 2021. Disponível em: <https://named-data.net/doc/NFD/current/INSTALL.html>.

RATHORE, M. S.; HIDELL, M.; SJÖDIN, P. Data plane optimization in open virtual routers. In: SPRINGER. **International Conference on Research in Networking**. [S.l.], 2011. p. 379–392. https://doi.org/10.1007/978-3-642-20757-$0_3$0.

REDHAT - Introduction to Linux interfaces for virtual networking. 2021. Disponível em: <https://developers.redhat.com/blog/2018/10/22/introduction-to-linux-interfaces-for-virtual-networking/>.

SALTZER, J. H.; REED, D. P.; CLARK, D. D. End-to-end arguments in system design. **ACM Transactions on Computer Systems (TOCS)**, Acm New York, NY, USA, v. 2, n. 4, p. 277–288, 1984. https://doi.org/10.1145/357401.357402.

SESKAR, I. et al. Mobilityfirst future internet architecture project. In: **Proceedings of the 7th Asian Internet Engineering Conference**. [S.l.: s.n.], 2011. p. 1–3. https://doi.org/10.1145/2089016.2089017.

SILVA, F. et al. Entity title architecture extensions towards advanced quality-oriented mobility control capabilities. In: IEEE. **2014 IEEE Symposium on Computers and Communications (ISCC)**. [S.l.], 2014. p. 1–6. https://doi.org/10.1109/ISCC.2014.6912459.

SILVA, F. d. O. **Endereçamento por título: uma forma de encaminhamento multicast para a próxima geração de redes de computadores.** Tese (Doutorado) — Universidade de São Paulo, 2013. https://doi.org/10.11606/T.3.2013.tde-22092014-111409.

ZHANG, L. et al. Named data networking (ndn) project. **Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC**, Citeseer, v. 157, p. 158, 2010.

Apêndices

APÊNDICE A

etarch-chat.c

```
#include <linux/if_packet.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <net/if.h>
#include <netinet/ether.h>
#include <netinet/in.h>

#define MAXCHAR         1024
#define ETHER_TYPE     0x0880

/*C program to convert hexadecimal Byte to integer.*/
//function : getNum
//this function will return number corresponding
//0,1,2..,9,A,B,C,D,E,F

int getNum(char ch) {
    int num;
    if (ch >= '0' && ch <= '9') {
        num = ch - 0x30;
    } else {
        switch (ch) {
            case 'A':
            case 'a':
                num = 10;
```

```
                break;
            case 'B':
            case 'b':
                num = 11;
                break;
            case 'C':
            case 'c':
                num = 12;
                break;
            case 'D':
            case 'd':
                num = 13;
                break;
            case 'E':
            case 'e':
                num = 14;
                break;
            case 'F':
            case 'f':
                num = 15;
                break;
            default:
                num = 0;
        }
    }
    return num;
}


//function : hex2int
//this function will return integer value against
//hexValue - which is in string format

unsigned int hex2int(char hex[]) {
    unsigned int x;
    x = (getNum(hex[0])) * 16 + (getNum(hex[1]));
    return x;
}
```

```
int main(int argc, char *argv[]) {
    ssize_t numbytes;
    uint8_t buf[MAXCHAR];
    (void) argc;
    char iface[MAXCHAR], et[MAXCHAR], crc[MAXCHAR];

    // Open configuration file and get variables
    FILE *fp;
    char filename[] = "/etc/etarch/etarch.conf";

    fp = fopen(filename, "r");
    if (fp == NULL) {
        printf("Could not open file %s", filename);
        return 1;
    }

    // Construct and execute command workspace start
    char wsname[MAXCHAR];
    strcat(wsname, "etarch-ws start ");
    strcat(wsname, argv[1]);
    system(wsname);

    while ((fscanf(fp, "%s %s", iface, et)) == 2) {

        // INIT construct command conversion
        char ws[MAXCHAR];
        strcpy(ws, argv[1]);
        char exec1[] = "/usr/local/bin/etarch-cv ";
        strcat(exec1, ws);

        // INIT convert et
        FILE *fp1;

        if ((fp1 = popen(exec1, "r")) == NULL) {
            printf("Error opening pipe!\n");
            return -1;
        }
```

```
while (fgets(crc, MAXCHAR, fp1) != NULL) {
    // Do whatever you want here...
}

if (pclose(fp1)) {
    printf("Command not found or exited with error
       status\n");
    return -1;
}
//END convert et

// CONTINUE construct command conversion
char m0[20], m1[3], m2[3], m3[3], m4[3], m5[3];
sprintf(m0, "%.2s", &crc[0]);
sprintf(m1, "%.2s", &crc[2]);
sprintf(m2, "%.2s", &crc[4]);
sprintf(m3, "%.2s", &crc[6]);
sprintf(m4, "%.2s", &crc[8]);
sprintf(m5, "%.2s", &crc[10]);

unsigned int m0a; //can be stored in unsigned char
unsigned int m1a; //can be stored in unsigned char
unsigned int m2a; //can be stored in unsigned char
unsigned int m3a; //can be stored in unsigned char
unsigned int m4a; //can be stored in unsigned char
unsigned int m5a; //can be stored in unsigned char

m0a = hex2int(m0);
m1a = hex2int(m1);
m2a = hex2int(m2);
m3a = hex2int(m3);
m4a = hex2int(m4);
m5a = hex2int(m5);

int sockfd;
int sockfd2;
int sockopt;
struct ifreq if_idx;
struct ifreq if_mac;
```

```
int tx_len;
char sendbuf[MAXCHAR];

//RAW Socket send component
/* Header structures */
struct ether_header *eh = (struct ether_header *)
   sendbuf;
//      struct iphdr *iph = (struct iphdr *) (sendbuf +
   sizeof(struct ether_header));
//      struct udphdr *udph = (struct udphdr *) (buf +
   sizeof(struct iphdr) + sizeof(struct ether_header));
struct sockaddr_ll socket_address;


/* Open RAW socket to send on */
if ((sockfd = socket(AF_PACKET, SOCK_RAW, IPPROTO_RAW))
   == -1) {
    perror("socket");
}

/* Open PF_PACKET socket, listening for EtherType
   ETHER_TYPE */
if ((sockfd2 = socket(PF_PACKET, SOCK_RAW, htons(
   ETHER_TYPE))) == -1) {
    perror("listener: socket");
    return -1;
}

/* Allow the socket to be reused - incase connection is
   closed prematurely */
if (setsockopt(sockfd2, SOL_SOCKET, SO_REUSEADDR, &
   sockopt, sizeof sockopt) == -1) {
    perror("setsockopt");
    exit(EXIT_FAILURE);
}
/* Bind to device */
if (setsockopt(sockfd2, SOL_SOCKET, SO_BINDTODEVICE,
   argv[1], IFNAMSIZ - 1) == -1) {
    perror("SO_BINDTODEVICE");
```

```
    exit(EXIT_FAILURE);
}

/* Get the index of the interface to send on */
memset(&if_idx, 0, sizeof(struct ifreq));
strncpy(if_idx.ifr_name, argv[1], IFNAMSIZ - 1);
if (ioctl(sockfd, SIOCGIFINDEX, &if_idx) < 0)
    perror("SIOCGIFINDEX");
/* Get the MAC address of the interface to send on */
memset(&if_mac, 0, sizeof(struct ifreq));
strncpy(if_mac.ifr_name, argv[1], IFNAMSIZ - 1);
if (ioctl(sockfd, SIOCGIFHWADDR, &if_mac) < 0)
    perror("SIOCGIFHWADDR");

while (1) {

    /* Construct the Ethernet header */
    memset(sendbuf, 0, MAXCHAR);
    tx_len = 0;
    /* Ethernet header */
    eh->ether_shost[0] = ((uint8_t *) &if_mac.ifr_hwaddr
        .sa_data)[0];
    eh->ether_shost[1] = ((uint8_t *) &if_mac.ifr_hwaddr
        .sa_data)[1];
    eh->ether_shost[2] = ((uint8_t *) &if_mac.ifr_hwaddr
        .sa_data)[2];
    eh->ether_shost[3] = ((uint8_t *) &if_mac.ifr_hwaddr
        .sa_data)[3];
    eh->ether_shost[4] = ((uint8_t *) &if_mac.ifr_hwaddr
        .sa_data)[4];
    eh->ether_shost[5] = ((uint8_t *) &if_mac.ifr_hwaddr
        .sa_data)[5];
    eh->ether_dhost[0] = m0a;
    eh->ether_dhost[1] = m1a;
    eh->ether_dhost[2] = m2a;
    eh->ether_dhost[3] = m3a;
    eh->ether_dhost[4] = m4a;
    eh->ether_dhost[5] = m5a;
    /* Ethertype field */
```

```
eh->ether_type = htons(ETHER_TYPE);
tx_len += sizeof(struct ether_header);

/* Index of the network device */
socket_address.sll_ifindex = if_idx.ifr_ifindex;
/* Address length*/
socket_address.sll_halen = ETH_ALEN;
/* Destination MAC */
socket_address.sll_addr[0] = m0a;
socket_address.sll_addr[1] = m1a;
socket_address.sll_addr[2] = m2a;
socket_address.sll_addr[3] = m3a;
socket_address.sll_addr[4] = m4a;
socket_address.sll_addr[5] = m5a;

char message[MAXCHAR];
printf("Send a message: ");
printf("\n");
fgets(message, 100, stdin);

unsigned int i;

for (i = 0; i < strlen(message); i++) {
    sendbuf[tx_len++] = message[i];
}

/* Send packet */
if (sendto(sockfd, sendbuf, tx_len, 0, (struct
   sockaddr *) &socket_address, sizeof(struct
   sockaddr_ll)) < 0)
    printf("Send failed\n");

/* receive message*/
numbytes = recvfrom(sockfd2, buf, MAXCHAR, 0, NULL,
   NULL);

/* Print packet */
printf("\nReceived Message: ");
printf("\n");
```

```
            for (i = 14; i < numbytes; i++) { printf("%c", buf[i
                ]); }
            printf("\n");

        }
    }
}
```

APÊNDICE B

etarch-cv.c

```
#include <inttypes.h>
#include <stdint.h>
#include <stdio.h>
#include <string.h>

#include "../include/checksum.h"

#define MAX_STRING_SIZE     2048

/*
 * int main( int argc, char *argv[] );
 *
 * The function main() is the entry point of the example program
     which
 * calculates several CRC values from the contents of files, or
    data from
 * stdin.
 */

int main(int argc, char *argv[]) {

    unsigned char *ptr;
    unsigned char prev_byte;
    uint16_t crc_16_val;
    uint16_t crc_16_modbus_val;
    uint16_t crc_ccitt_ffff_val;
    uint16_t crc_ccitt_0000_val;
```

```
uint16_t crc_ccitt_1d0f_val;
uint16_t crc_dnp_val;
uint16_t crc_sick_val;
uint16_t crc_kermit_val;
uint32_t crc_32_val;
uint16_t low_byte;
uint16_t high_byte;

if (argc > 1) {
    ptr = (unsigned char *) argv[1];
    while (*ptr && *ptr != '\r' && *ptr != '\n') ptr++;
    *ptr = 0;

    crc_16_val = 0x0000;
    crc_16_modbus_val = 0xffff;
    crc_dnp_val = 0x0000;
    crc_sick_val = 0x0000;
    crc_ccitt_0000_val = 0x0000;
    crc_ccitt_ffff_val = 0xffff;
    crc_ccitt_1d0f_val = 0x1d0f;
    crc_kermit_val = 0x0000;
    crc_32_val = 0xffffffffL;

    prev_byte = 0;
    ptr = (unsigned char *) argv[1];

    while (*ptr) {

        crc_16_val = update_crc_16(crc_16_val, *ptr);
        crc_16_modbus_val = update_crc_16(crc_16_modbus_val,
            *ptr);
        crc_dnp_val = update_crc_dnp(crc_dnp_val, *ptr);
        crc_sick_val = update_crc_sick(crc_sick_val, *ptr,
           prev_byte);
        crc_ccitt_0000_val = update_crc_ccitt(
           crc_ccitt_0000_val, *ptr);
        crc_ccitt_ffff_val = update_crc_ccitt(
           crc_ccitt_ffff_val, *ptr);
        crc_ccitt_1d0f_val = update_crc_ccitt(
```

```
            crc_ccitt_1d0f_val, *ptr);
        crc_kermit_val = update_crc_kermit(crc_kermit_val, *
            ptr);
        crc_32_val = update_crc_32(crc_32_val, *ptr);

        prev_byte = *ptr;
        ptr++;
    }

    crc_32_val ^= 0xffffffffL;

    crc_dnp_val = ~crc_dnp_val;
    low_byte = (crc_dnp_val & 0xff00) >> 8;
    high_byte = (crc_dnp_val & 0x00ff) << 8;
    crc_dnp_val = low_byte | high_byte;

    low_byte = (crc_sick_val & 0xff00) >> 8;
    high_byte = (crc_sick_val & 0x00ff) << 8;
    crc_sick_val = low_byte | high_byte;

    low_byte = (crc_kermit_val & 0xff00) >> 8;
    high_byte = (crc_kermit_val & 0x00ff) << 8;
    crc_kermit_val = low_byte | high_byte;

    char crca[3] = "E0";
    char crcb[20];
    char crc32[20];
    char crc16[20];
    sprintf(crc32, "%08" PRIX32, crc_32_val);
    sprintf(crc16, "%04" PRIX32, crc_16_val);
    strcat(crcb, &crc32[2]);
    strcat(crca, crcb);
    strcat(crca, crc16);

    printf("%s", crca);
  }

}  /* main (tstcrc.c) */
```

APÊNDICE C

etarch-int.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAXCHAR 1000

int main(int argc, char *argv[]) {
    (void) argc;
    char iface[MAXCHAR], et[MAXCHAR], crc[MAXCHAR];

    // Open configuration file and get variables
    FILE *fp;
    char filename[] = "/etc/etarch/etarch.conf";

    fp = fopen(filename, "r");
    if (fp == NULL) {
        printf("Could not open file %s", filename);
        return 1;
    }

    // Construct and command conversion
    while ((fscanf(fp, "%s %s", iface, et)) == 2) {
        char exec1[] = "/usr/local/bin/etarch-cv ";
        strcat(exec1, et);

        // INIT convert et
        FILE *fp1;
```

```
        if ((fp1 = popen(exec1, "r")) == NULL) {
            printf("Error opening pipe!\n");
            return -1;
        }

        while (fgets(crc, MAXCHAR, fp1) != NULL) {
            // Do whatever you want here...
//          printf("%.2s", crc);
        }

        if (pclose(fp1)) {
            printf("Command not found or exited with error
               status\n");
            return -1;
        }
        //END convert et

        // Construct and execute command vNIC creator
        char t0[MAXCHAR] = "/bin/ip link ";
        char T0[MAXCHAR] = "/bin/ip link set ";

        if (strcmp(argv[1], "start") == 0) {
            char t1[MAXCHAR] = "add link ";
            char t2[] = " address ";
            char t3[] = " ";
            char t4[] = ":";
            strcat(t0, t1);
            strcat(t0, iface);
            strcat(t0, t3);
            strcat(t0, et);
            strcat(t0, t2);

            char m0[3], m1[3], m2[3], m3[3], m4[3], m5[3];
            sprintf(m0, "%.2s", crc);
            sprintf(m1, "%.2s", &crc[2]);
            sprintf(m2, "%.2s", &crc[4]);
            sprintf(m3, "%.2s", &crc[6]);
            sprintf(m4, "%.2s", &crc[8]);
```

```
            sprintf(m5, "%.2s", &crc[10]);

            strcat(t0, m0);
            strcat(t0, t4);
            strcat(t0, m1);
            strcat(t0, t4);
            strcat(t0, m2);
            strcat(t0, t4);
            strcat(t0, m3);
            strcat(t0, t4);
            strcat(t0, m4);
            strcat(t0, t4);
            strcat(t0, m5);

            char t5[] = " type macvlan mode bridge";
            strcat(t0, t5);

            strcat(T0, et);
            strcat(T0, " up");
            system(t0);
            system(T0);
        } else if (strcmp(argv[1], "stop") == 0) {
            char t1[MAXCHAR] = "delete ";
            strcat(t0, t1);
            strcat(t0, et);
            system(t0);
        }
    }
}
```

APÊNDICE D

etarch-sf.c

```
#include <linux/if_packet.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <net/if.h>
#include <netinet/ether.h>
#include <netinet/in.h>

#define MAXCHAR         999999
#define MAXALLOC     536870912
#define ETHER_TYPE     0x0880

/*C program to convert hexadecimal Byte to integer.*/
//function : getNum
//this function will return number corresponding
//0,1,2..,9,A,B,C,D,E,F

int getNum(char ch) {
    int num;
    if (ch >= '0' && ch <= '9') {
        num = ch - 0x30;
    } else {
        switch (ch) {
            case 'A':
            case 'a':
```

```
                num = 10;
                break;
            case 'B':
            case 'b':
                num = 11;
                break;
            case 'C':
            case 'c':
                num = 12;
                break;
            case 'D':
            case 'd':
                num = 13;
                break;
            case 'E':
            case 'e':
                num = 14;
                break;
            case 'F':
            case 'f':
                num = 15;
                break;
            default:
                num = 0;
        }
    }
    return num;
}


//function : hex2int
//this function will return integer value against
//hexValue - which is in string format

unsigned int hex2int(char hex[]) {
    unsigned int x;
    x = (getNum(hex[0])) * 16 + (getNum(hex[1]));
    return x;
//     printf("%s", x);
```

```
}


int main(int argc, char *argv[]) {

    (void) argc;
    char iface[MAXCHAR], et[MAXCHAR], crc[MAXCHAR];

    // Open configuration file and get variables
    FILE *fp;
    char filename[] = "/etc/etarch/etarch.conf";

    fp = fopen(filename, "r");
    if (fp == NULL) {
        printf("Could not open file %s", filename);
        return 1;
    }

    // Construct and execute command workspace start
    char wsname[MAXCHAR];
    strcat(wsname, "etarch-ws start ");
    strcat(wsname, argv[1]);
    system(wsname);

    while ((fscanf(fp, "%s %s", iface, et)) == 2) {

        char ws[MAXCHAR];
        strcpy(ws, argv[1]);

        // INIT construct command conversion
        char exec1[] = "/usr/local/bin/etarch-cv ";
        strcat(exec1, ws);

        // INIT convert et
        FILE *fp1;

        if ((fp1 = popen(exec1, "r")) == NULL) {
            printf("Error opening pipe!\n");
            return -1;
```

```
}

while (fgets(crc, MAXCHAR, fp1) != NULL) {
    // Do whatever you want here...
}

if (pclose(fp1)) {
    printf("Command not found or exited with error
       status\n");
    return -1;
}
//END convert et

// CONTINUE construct command conversion
char m0[20], m1[3], m2[3], m3[3], m4[3], m5[3];
sprintf(m0, "%.2s", &crc[0]);
sprintf(m1, "%.2s", &crc[2]);
sprintf(m2, "%.2s", &crc[4]);
sprintf(m3, "%.2s", &crc[6]);
sprintf(m4, "%.2s", &crc[8]);
sprintf(m5, "%.2s", &crc[10]);

unsigned int m0a; //can be stored in unsigned char
unsigned int m1a; //can be stored in unsigned char
unsigned int m2a; //can be stored in unsigned char
unsigned int m3a; //can be stored in unsigned char
unsigned int m4a; //can be stored in unsigned char
unsigned int m5a; //can be stored in unsigned char

m0a = hex2int(m0);
m1a = hex2int(m1);
m2a = hex2int(m2);
m3a = hex2int(m3);
m4a = hex2int(m4);
m5a = hex2int(m5);

int sockfd;
int sockfd2;
int sockopt;
```

```
struct ifreq if_idx;
struct ifreq if_mac;
int tx_len;
char *sendbuf = malloc(MAXALLOC);

//RAW Socket send component
/* Header structures */
struct ether_header *eh = (struct ether_header *)
   sendbuf;
//      struct iphdr *iph = (struct iphdr *) (sendbuf +
   sizeof(struct ether_header));
//      struct udphdr *udph = (struct udphdr *) (buf +
   sizeof(struct iphdr) + sizeof(struct ether_header));
struct sockaddr_ll socket_address;

/* Open RAW socket to send on */
if ((sockfd = socket(AF_PACKET, SOCK_RAW, IPPROTO_RAW))
   == -1) {
    perror("socket");
}

/* Open PF_PACKET socket, listening for EtherType
   ETHER_TYPE */
if ((sockfd2 = socket(PF_PACKET, SOCK_RAW, htons(
   ETHER_TYPE))) == -1) {
    perror("listener: socket");
    return -1;
}

/* Allow the socket to be reused - incase connection is
   closed prematurely */
if (setsockopt(sockfd2, SOL_SOCKET, SO_REUSEADDR, &
   sockopt, sizeof sockopt) == -1) {
    perror("setsockopt");
    //close(sockfd2);
    exit(EXIT_FAILURE);
}
/* Bind to device */
if (setsockopt(sockfd2, SOL_SOCKET, SO_BINDTODEVICE,
```

```
    argv[1], IFNAMSIZ - 1) == -1) {
      perror("SO_BINDTODEVICE");
      exit(EXIT_FAILURE);
}

/* Get the index of the interface to send on */
memset(&if_idx, 0, sizeof(struct ifreq));
strncpy(if_idx.ifr_name, argv[1], IFNAMSIZ - 1);
if (ioctl(sockfd, SIOCGIFINDEX, &if_idx) < 0)
    perror("SIOCGIFINDEX");
/* Get the MAC address of the interface to send on */
memset(&if_mac, 0, sizeof(struct ifreq));
strncpy(if_mac.ifr_name, argv[1], IFNAMSIZ - 1);
if (ioctl(sockfd, SIOCGIFHWADDR, &if_mac) < 0)
    perror("SIOCGIFHWADDR");

char *message = malloc(MAXALLOC);
// INIT - file
char *sndbuf;
FILE *fp2;
size_t size;

char openfile[MAXCHAR];
strcpy(openfile, argv[2]);
fp2 = fopen(openfile, "rb");

// Go to end file
fseek(fp2, 0, SEEK_END);
// Get size file
size = ftell(fp2);
// Go to back start file
rewind(fp2);
// Allocate buffer space with file size
sndbuf = malloc((size + 1) * sizeof(*sndbuf));
// Read start to end file
fread(sndbuf, size, 1, fp2);
sndbuf[size] = '\0';
// Copy file content/buffer to global variable
strcpy(message, sndbuf);
```

```
// END - file

unsigned int i = 0;
unsigned int j = 0;

while (i < strlen(message)) {

    /* Construct the Ethernet header */
    tx_len = 0;
    /* Ethernet header */
    eh->ether_shost[0] = ((uint8_t *) &if_mac.ifr_hwaddr
       .sa_data)[0];
    eh->ether_shost[1] = ((uint8_t *) &if_mac.ifr_hwaddr
       .sa_data)[1];
    eh->ether_shost[2] = ((uint8_t *) &if_mac.ifr_hwaddr
       .sa_data)[2];
    eh->ether_shost[3] = ((uint8_t *) &if_mac.ifr_hwaddr
       .sa_data)[3];
    eh->ether_shost[4] = ((uint8_t *) &if_mac.ifr_hwaddr
       .sa_data)[4];
    eh->ether_shost[5] = ((uint8_t *) &if_mac.ifr_hwaddr
       .sa_data)[5];
    eh->ether_dhost[0] = m0a;
    eh->ether_dhost[1] = m1a;
    eh->ether_dhost[2] = m2a;
    eh->ether_dhost[3] = m3a;
    eh->ether_dhost[4] = m4a;
    eh->ether_dhost[5] = m5a;
    /* Ethertype field */
    eh->ether_type = htons(ETHER_TYPE);
    tx_len += sizeof(struct ether_header);

    /* Index of the network device */
    socket_address.sll_ifindex = if_idx.ifr_ifindex;
    /* Address length*/
    socket_address.sll_halen = ETH_ALEN;
    /* Destination MAC */
    socket_address.sll_addr[0] = m0a;
    socket_address.sll_addr[1] = m1a;
```

```
            socket_address.sll_addr[2] = m2a;
            socket_address.sll_addr[3] = m3a;
            socket_address.sll_addr[4] = m4a;
            socket_address.sll_addr[5] = m5a;

            while (j < i + 130 && j < strlen(message)) {
                sendbuf[tx_len++] = message[j];
                j++;
            }
            if (sendto(sockfd, sendbuf, tx_len, 0, (struct
               sockaddr *) &socket_address, sizeof(struct
               sockaddr_ll)) < 0)
                printf("Send failed\n");
            i = j;
        }
        free(sndbuf);
        free(sendbuf);
        free(message);
        return 0;
    }
    return 0;
}
```

APÊNDICE E

etarch-ws.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAXCHAR 1000

int main(int argc, char *argv[]) {
    (void) argc;
    char iface[MAXCHAR], et[MAXCHAR], crc[MAXCHAR];

    // Open configuration file and get variables
    FILE *fp;
    char filename[] = "/etc/etarch/etarch.conf";

    fp = fopen(filename, "r");
    if (fp == NULL) {
        printf("Could not open file %s", filename);
        return 1;
    }

    // INIT construct command conversion
    while ((fscanf(fp, "%s %s", iface, et)) == 2) {
        char exec1[] = "/usr/local/bin/etarch-cv ";
        strcat(exec1, et);
        strcat(exec1, "+");
        strcat(exec1, argv[2]);
```

```
// INIT convert et
FILE *fp1;
if ((fp1 = popen(exec1, "r")) == NULL) {
    printf("Error opening pipe!\n");
    return -1;
}

while (fgets(crc, MAXCHAR, fp1) != NULL) {
    // Do whatever you want here...
}

if (pclose(fp1)) {
    printf("Command not found or exited with error
       status\n");
    return -1;
}
//END convert et

// Construct and execute command vNIC creator
char t0[MAXCHAR] = "/bin/ip link ";
char T0[MAXCHAR] = "/bin/ip link set ";

if (strcmp(argv[1], "start") == 0) {
    char t1[MAXCHAR] = "add link ";
    char t2[] = " address ";
    char t3[] = " ";
    char t4[] = ":";
    strcat(t0, t1);
    strcat(t0, iface);
    strcat(t0, t3);
    strcat(t0, argv[2]);
    strcat(t0, t2);

    char m0[3], m1[3], m2[3], m3[3], m4[3], m5[3];
    sprintf(m0, "%.2s", crc);
    sprintf(m1, "%.2s", &crc[2]);
    sprintf(m2, "%.2s", &crc[4]);
    sprintf(m3, "%.2s", &crc[6]);
    sprintf(m4, "%.2s", &crc[8]);
```

```
            sprintf(m5, "%.2s", &crc[10]);

            strcat(t0, m0);
            strcat(t0, t4);
            strcat(t0, m1);
            strcat(t0, t4);
            strcat(t0, m2);
            strcat(t0, t4);
            strcat(t0, m3);
            strcat(t0, t4);
            strcat(t0, m4);
            strcat(t0, t4);
            strcat(t0, m5);

            char t5[] = " type macvlan mode bridge";
            strcat(t0, t5);

            strcat(T0, argv[2]);
            strcat(T0, " up");
            system(t0);
            system(T0);
        } else if (strcmp(argv[1], "stop") == 0) {
            char t1[MAXCHAR] = "delete ";
            strcat(t0, t1);
            strcat(t0, argv[2]);
            system(t0);
        }
    }
}
```

APÊNDICE F

ndn_test.sh

```
#!/bin/bash

#variables
ws=chat.ufu.br
cv=`etarch-cv chat.ufu.br`
route=`nfdc face list | grep dev://chat.ufu.br | cut -f2 -d"="`

#Proposal applications start
etarch-int start
etarch-ws start $ws

#NDN start
nfd-start
nfdc face create remote ether://[$cv] local dev://$ws
    persistency permanent
nfdc route add /$ws $route

#NDN traffic configuration
echo -ne "Name=/$ws\nContent=HelloWord" > ndn-traffic-server.
    conf
echo -ne "TrafficPercentage=100\nName=/$ws\nExpectedContent=
    HelloWord" > ndn-traffic-client.conf

#Traffic generation on server
ndn-traffic-server ndn-traffic-server.conf

#Traffic generation on client
ndn-traffic-client ndn-traffic-client.conf
```