UNIVERSIDADE FEDERAL DE SÃO CARLOS
PROGRAMA DE PÓS GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Guilherme Mendes Vieira de Matos

In-Network IdentifiCation and chAining: um mecanismo de identificação de tráfego e encadeamento no plano de dados

São Carlos - SP

2021

Guilherme Mendes Vieira de Matos

In-Network IdentifiCation and chAining: um mecanismo de identificação de tráfego e encadeamento no plano de dados

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação.

Orientação Prof. Dr. Fábio Luciano Verdi

São Carlos - SP

2021

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências em Gestão e Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Defesa de Dissertação de Mestrado do candidato Guilherme Mendes Vieira de Matos, realizada em 27/10/2021.

Comissão Julgadora:

Prof. Dr. Fabio Luciano Verdi (UFSCar)

Prof. Dr. Christian Esteve Rothenberg (UNICAMP)

Prof. Dr Hélio Crestana Guardia (UFSCar)

O Relatório de Defesa assinado pelos membros da Comissão Julgadora encontra-se arquivado junto ao Programa de Pós-Graduação em Ciência da Computação

Dedico este trabalho a minha família. Dedico também àqueles que me chamaram de louco, mas principalmente a quem não acreditava.

Agradecimentos

Agradeço,

à minha mulher Patricia Angelucci de Lima, pelo apoio incondicional, sacrifício e paciência.

aos meus filhos, por suportar a ausência.

aos meus pais e avós pelo incentivo e ajuda.

ao meu irmão Julio César, por me abrir as portas de sua casa quando precisei.

ao Prof. Dr. Fábio Luciano Verdi pelo tempo e por tanto ter me ensinado.

ao Prof. Ms. Paulo Eduardo Cardoso pelo incentivo e apoio.

aos meus colegas de pesquisa Leandro, André e Rodrigo, por compartilharem desde feedbacks a trechos de códigos.

Todos estes contribuíram de forma significativa para a realização deste trabalho. Ao final, refiz e fortaleci laços de sangue e criei novos laços de amizade.

"Isto é para os loucos. Os desajustados. Os rebeldes. Os desordeiros. Para os peixes fora d'agua. Para aqueles que vêem as coisas de forma diferente. Eles não gostam de regras. E não nutrem o menor respeito pelo status quo. Você pode citá-los, discordar deles, glorificá-los ou difamá-los. Mas a única coisa que você não pode fazer é ignorá-los. Porque eles transformam as coisas. Eles empurram a raça humana para frente. Enquanto alguns podem vê-los como loucos, nós vemos o gênio."

(Steve Jobs)

Resumo

Dentro do 5G *Core* (5GC), uma miríade de funções (virtuais) podem ser implementadas para que diferentes tratamentos possam ser dados para o tráfego entre a *Radio Access Network* (RAN) e *User Plane Function* (UPF). Uma dessas funções principais é o encadeamento de funções de serviço (SFC, do inglês *Service Function Chaining*), compatível com o protocolo SRv6 de última geração. Neste trabalho, apresentamos uma solução baseada em P4 para identificação e encadeamento de tráfego usando SRv6. Chamamos essa função de *In-Network Classification and chAining* (INCA), que é totalmente implantada no plano de dados usando uma placa de rede Netronome Agilo SmartNIC. O INCA é implantado imediatamente antes da UPF, dentro do 5GC e é capaz de observar o tráfego que entra e sai da RAN para classificar e criar a sequência adequada de serviços a ser seguida por cada fluxo específico. Nossos resultados mostram que o INCA realiza a tarefa de classificação e encadeamento de pacotes perfeitamente com um impacto mínimo de *Flow Completion Time* (FCT) quando comparado ao mesmo ambiente sem ele.

Palavras-chave: 5G, *network slice*, P4, plano de dados, *service function chaining*, dispositivos programáveis.

Abstract

Inside the 5G *Core* (5GC) a myriad of (virtual) functions may be deployed so that different treatment can be given for traffic coming in and out to/from the *Radio Access Network* (RAN). One of these key functions is *Service Function Chaining* (SFC) supported by SRv6 state-of-the-art protocol. In this work, we present an SFC P4-based solution for traffic identification and chaining using SRv6. We call this function *In-Network Classification and chAining* (INCA), which is deployed entirely in the data plane using a Netronome Agilo SmartNIC. The INCA is deployed just before the *User Plane Function* (UPF) inside the 5GC and is capable of observing the traffic coming from and going to the RAN so that it classifies and creates the proper sequence of services to be followed by every specific flow. Our results show that INCA performs the task of packet classification and chaining perfectly with a minimal *Flow Completion Time* (FCT) impact when compared to the same environment without it.

Keywords: 5G, *network slice*, P4, dataplane, *service function chaining*, programmable devices.

Lista de ilustrações

Lista de tabelas

Lista de acrônimos

3GPP *3rd Generation Partnership Project*

5GC 5G *Core*

AF *Application Function*

AMF *Access and Mobility Management Function*

AN *Access Network*

ASIC *Application Specific Integrated Circuits*

BMv2 *Behavioral Model version 2*

CP *Control Plane*

CUPS *Control and User Plane Separation*

DN *Data Network*

DNN *Data Network Name*

DPI *Deep Packet Inspection*

DRB *Data Radio Bearer*

eMBB *Enhanced Mobile Broadband*

ETSI *European Telecommunications Standards Institute*

FCT *Flow Completion Time*

FPGA *Field-Programmable Gate Array*

GSM *Global System for Mobile Communications*

GTP *GPRS Tunneling Protocol*

GTP-U *GPRS Tunnelling Protocol for the user plane*

IDS *Intrusion Detection System*

INCA *In-Network Classification and chAining*

IoT *Internet of Things*

IPS *Intrusion Prevention System*

LTE *Long Term Evolution*

LTE EPC *Long Term Evolution Evolved Packet Core*

MANO *Management and Orchestration*

MEC *Mobile Edge Computing*acroMIoT*Massive* IoT

NFV *Network Function Virtualization*

NF *Network Function*

NS *Network Slice*acroNSI*Network Slice Instance*

NSSAI *Network Slice Selection Assistance Information*

NSSF *Network Slice Selection Function*

ONF *Open Networking Foundation*

PDU *Protocol Data Unit*

PF *Phisical Function*

QoS *Quality of Service*

RAN *Radio Access Network*

SBA *Service-Based Architecture*

SDN *Software Defined Network*

SFC *Service Function Chaining*

SGW-U *Serving Gateway User plane function*

SL *Segment Left*

SMF *Session Management Function*

SR *Segment Routing*

SRH *Segment Routing Header*

SR-I/OV *Single-root input/output virtualization*

SRv6 *Segment Routing over IPv6 dataplane*

TEID *Tunnel Endpoint Identifier*

UE *User Equipment*

UMTS *Universal Mobile Telecommunications System*

UP *User Plane*

UPF *User Plane Function*

URLLC *Ultra Reliable Low Latency Communication*

VF *Virtual Function*

VNF *Virtual Network Function*

Sumário

1 Introdução

Há algum tempo estamos presenciando o crescimento de tecnologias relacionadas a internet das coisas (do inglês, *Internet of Things* (IoT)), serviços de transporte e carros autônomos, cirurgia remota e controle remoto de operações robóticas em indústrias. Esse crescimento ocorre principalmente em países desenvolvidos, com conexões consideradas ótimas e que suportam essa demanda. A 5ª geração (5G) do sistema de comunicações móveis (2411-0, 2017) já está saindo do papel em muitos países, e sua arquitetura foi especialmente desenhada para suportar essa demanda. Não só isso, existe também a pretensão de operadoras de 5G por uma fatia do mercado que hoje é quase que exclusividade das operadoras de computação em nuvem. Mas para isto, esta arquitetura deve ser elástica, ágil, programável e terá que suportar serviços diversificados e requisitos técnicos no que diz respeito à vazão, latência, disponibilidade e confiabilidade (ROST et al., 2017).

Para lidar com este cenário, a arquitetura 5G foi projetada com mudanças estruturais muito importantes já implementadas desde sua concepção, adicionando os conceitos mais modernos do estado da arte, como *network slicing* (ORDONEZ-LUCENA et al., 2018b), arquitetura baseada em serviços (SBA, do inglês, *Service-Based Architecture*) e também Rede Definida por Software (SDN, do inglês, *Software Defined Network*) (XIA et al., 2014). Este último, conhecido como *Control and User Plane Separation* (CUPS) dentro da arquitetura 5G, permite a separação do plano de controle e dados (na arquitetura 5G o plano de dados é referenciado como plano de usuário, ou *User Plane* (UP)), que é um conceito chave desde a origem do OpenFlow em 2008.

A arquitetura 5G foi desenhada para ser uma plataforma baseada em nuvem, tendo assim todas suas funções *softwarizadas*. Com isto, existe uma ênfase na virtualização de funções de rede (NFV, do inglês, *Network Function Virtualization*) (JOSHI; BENSON, 2016) como um conceito de projeto integral com funções de software virtualizadas, desacoplando o software do hardware, substituindo diversas funções de rede, como *firewalls*, balanceadores de carga e roteadores, com instâncias virtualizadas, operando como software. A NFV pode tratar outros desafios do 5G por meio de computação virtualizada, armazenamento e recursos de rede que são personalizados com base nas aplicações e casos de uso específicos dos clientes.

Desta forma, criam-se novas oportunidades de modelo de negócios, como alocação de funções ou recursos específicos de acordo com a necessidade do cliente dentro do próprio 5GC, provendo agregação de valor a seus usuários. Esta oportunidade nos remete ao conceito de SFC (HALPERN; PIGNATARO et al., 2015), que se apresenta como um mecanismo quase indispensável para suportar serviços complexos, já que uma *network slice*

pode atravessar várias NFVs como servidores de cache, *firewalls*, *Deep Packet Inspection* (DPI) e outros para prover agregação de valor a seus usuários. Um mecanismo de SFC auxiliaria a transformar esse processamento de dados em uma sequência de serviços (HOMMA; FOY; GALIS, 2018).

Alinhados com as tecnologias mencionadas acima, atualmente também temos testemunhado uma tendência de adoção de soluções em rede aproveitando o hardware programável que surgiu nos últimos anos, bem como o uso de linguagens de programação de rede que permitem a criação de novos protocolos e comportamentos diretamente no equipamento de rede.

Levando em consideração que podemos usar tecnologias de software de rede como *Software Defined Network* (SDN) e NFV para criar soluções com alto nível de programabilidade, flexibilidade e modularidade, e também que identifica-se a necessidade de uma função para SFC que leve em consideração as limitações referentes à dependência do 5GC ao protocolo *GPRS Tunneling Protocol* (GTP), projetamos e implementamos uma solução que utiliza o protocolo *Segment Routing over IPv6 dataplane* (SRv6) (FILSFILS et al., 2018) para resolver a falta de um mecanismo capaz de encadear funções de serviço dentro do 5GC inteiramente no plano de dados.

O INCA é uma solução baseada em P4 capaz de realizar a identificação do tráfego e construir um cabeçalho SRv6 contendo um SRH. Este, por sua vez, contém uma lista de endereços IPv6 (chamados de segmentos) que representam as funções que estão sendo encadeadas. Para realizar esta identificação, o INCA é capaz de analisar diversos campos da pilha de protocolos, como cabeçalhos IPv6 (interno/externo), *Tunnel Endpoint Identifier* (TEID), QoS ID, entre outros.

Utilizando uma placa de rede SmartNIC Netronome Agilio CX 2x10Gbe, esta compatível com a linguagem P4, realizamos a implementação do INCA em um ambiente 5G emulado, onde o INCA foi capaz de realizar o encadeamento de funções. Realizamos a avaliação de desempenho levando em consideração o FCT. Fizemos vários testes enviando diferentes fluxos de tráfego para dois ambientes distintos, um com e outro sem INCA para que pudéssemos comparar o tempo de conclusão de cada fluxo para cada ambiente. Nossos testes mostram que o uso do INCA possibilita encadear funções de forma eficiente. Quanto ao desempenho, com a utilização do INCA houve um aumento máximo da FCT de apenas 1%, o que mostra que a sua utilização tem um impacto mínimo na arquitetura 5G.

Além do INCA, este trabalho também realiza outras duas contribuições: (1) implementação do *PDU Session User Plane Protocol*, novo protocolo especificado pelo 3GPP para carregar informações referentes a QoS no sistema 5G e (2) solução para desencapsulamento do cabeçalho SRv6 no modo *inline*, este ainda indisponível no estado da arte.

Organizamos o restante deste trabalho da seguinte maneira. Na Seção 2 apresentamos alguns conceitos importantes para o restante deste trabalho. Na Seção 3, apresentamos alguns trabalhos relacionados encontrados na literatura. Na Seção 4, apresentamos como nossa proposta foi desenhada e implementada. Na Seção 5, mostramos a implantação e avaliação. Por fim, na Seção 6, apresentamos as considerações finais.

2 Conceitos Básicos

Nesta seção, apresentaremos os conceitos básicos relacionados a este trabalho. No entanto, dado ao grande volume de informações referentes a alguns destes conceitos, serão apresentados apenas as informações úteis ao entendimento deste trabalho, e abordarão arquitetura do sistema 5G, *Service Function Chaining* (SFC), P4 e *Segment Routing over IPv6 dataplane* (SRv6).

2.1 Arquitetura 5G

O 5G é visto como um sistema fim a fim que foi desenhado para agregar valor a vários segmentos melhorando produtividade, sustentabilidade e bem estar para uma sociedade totalmente móvel e conectada. O 5G é definido de várias maneiras diferentes em vários projetos. No entanto, como este trabalho está inserido no contexto de 5G como definido pela *3rd Generation Partnership Project* (3GPP), este capítulo trará a definição e arquitetura conforme este modelo.

O 5G permitirá uma nova variedade de casos de uso, através da expansão da capacidade, como taxas de dados, latência e densidade. Como cada caso de uso terá seus próprios requisitos de serviço, o 5G precisará ter um *design* baseado na flexibilidade e escalabilidade de capacidades. Utilizando redes virtuais e programáveis, funções podem ser projetadas de forma modular, para que possam ser implantadas sob demanda (Iwamura, 2015).

De acordo com 3GPP (2019), a arquitetura do sistema 5G é definida para suportar conectividade de dados e serviços permitindo que as implantações usem técnicas como NFV e SDN. Sua arquitetura utilizará interações baseadas em serviço entre o plano de controle (CP, do inglês, *Control Plane*) e plano de usuário (UP, do inglês, *User Plane*). Além disto, este sistema conta com funções de CP e UP separadas, permitindo escalabilidade independente e implantações flexíveis. Um ponto importante é a possibilidade de funções UP serem implantadas próximas à rede de acesso (AN, do inglês, *Access Network*).

A Figura 1 apresenta a arquitetura do sistema 5G. Cada item representa um serviço ou função dentro da arquitetura. As caixas em azul fazem parte do CP e as caixas em vermelho do UP. Podemos notar também que há elementos que estão inseridos em ambos os planos. Na Tabela 1, são detalhados os serviços e funções que devem ser destacados no contexto deste projeto.

Um dos aspectos mais inovadores da arquitetura 5G dependerá do fatiamento da rede (do inglês, *Network slice*), que permitirá que operadores ofereçam porções de sua rede

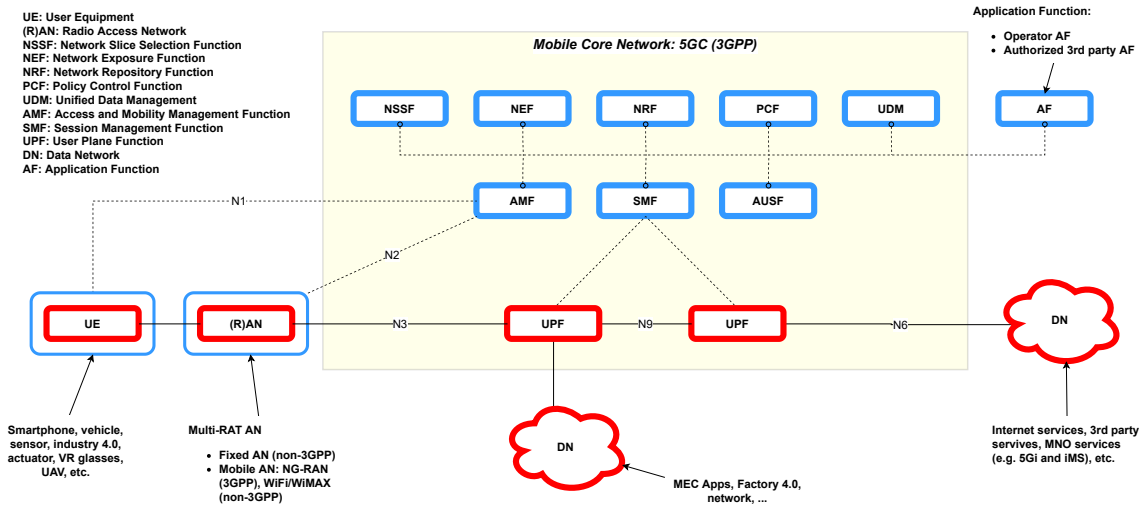


Figura 1 – Componentes do sistema 5G (3GPP, 2019).

Tabela 1 – Componentes e funções 5G (adaptado de 3GPP (2019).

UE	*User Equipment*: Equipamento que o usuário esteja utilizando para acessar os serviços 5G.
RAN	*Radio Access Network*: Rede de acesso ao 5GC.
UPF	*User Plane Function*: Componente fundamental com diversas funções, entre elas: ponto de interconexão entre entre a infraestrutura 5G e DN, encapsulamento e desencapsulamento do GTP-U (*GPRS Tunnelling Protocol for the user plane*), encaminhamento e roteamento de pacotes
DN	*Data Network*: Internet ou serviços de terceiros.
AMF	*Access and Mobility Management Function*: Função responsável pelo gerenciamento de registro, conexão, e outras tarefas, sendo assim, responsável por gerenciar o acesso ao CP.
SMF	*Session Management Function*: Responsável por interagir com CP e UP, criando, atualizando e removendo PDU *Sessions* (*Protocol Data Unit*) e gerenciando contexto de sessões com UP.
NSSF	*Network Slice Selection Function*: Seleciona NSI (*Network Slice Instance*) baseado nas informações fornecidas durante o registro do *User Equipment* (UE).
AF	*Application Function*: Permite que uma aplicação influencie o roteamento de tráfego, interagindo com as políticas de controle.

para casos de uso específicos (IoT, carros autonômos, etc.). Cada caso de uso receberá um conjunto de recursos e topologia de rede otimizado, atingindo as necessidades de cada aplicação.

Network slice, no caso da 3GPP, é uma arquitetura de rede virtual que é alcançável utilizando a flexibilidade oferecida por tecnologias modernas como NFV e SDN. Em essência, *network slice* permite a criação de múltiplas redes virtuais em uma infraestrutura física compartilhada.

De maneira geral, um recurso pode ser definido como uma unidade gerenciável,

esta, caracterizada por um conjunto de atributos ou capacidades que podem ser usadas para entregar um serviço. *Network slice* é composta por uma coleção de recursos que se combinados de forma apropriada, preenchem os requisitos de caso de uso específico de cada *slice* (ORDONEZ-LUCENA et al., 2017).

Foram definidas três categorias principais de *slices*, que são relacionadas a três serviços genéricos. A Tabela 2 mostra os três tipos de *Network Slices* e suas características de acordo com a 3GPP:

Tabela 2 – Tipos de *slice* (3GPP, 2019).

Slice/tipo de serviço	**Características**	**Requisitos**
Enhanced Mobile Broadband (eMBB)	Geralmente utilizada pelas aplicações móveis incluindo *streaming* de vídeos de alta qualidade e transferências de arquivos muito grandes. Desenvolvida para suportar altas taxas de dados e densidade de tráfego.	10Gbps
Ultra Reliable Low Latency Communication (URLLC)	Suporta comunicações ultra confiáveis de baixa latência incluindo automação industrial e sistemas de controle remoto.	1 ms
Massive IoT (MIoT)	Suporta um grande número e alta densidade de dispositivos IoT.	1000K conexões por quilometro quadrado

O aspecto mais importante da *network slice* é a virtualização dos recursos de rede, inclusive suas *Network Functions* (NFs). A Figura 2 mostra três dispositivos conectados em duas *slices*. Cada *slice* foi instanciada apenas com as funções de rede necessárias, o que mostra a flexibilidade e modularidade deste recurso. Essas *slices* compartilham a mesma infraestrutura física, no entanto, cada *slice* trafega de maneira isolada, mantendo assim a segurança e isonomia da rede (Jiang; Condoluci; Mahmoodi, 2016).

Dentro do sistema 5G existe um catálogo de serviço onde podem ser encontrados os conjuntos de modelos de serviços que descrevem cada um dos diferentes serviços que são ofertados. Estes modelos contêm todas as informações que são necessárias para a implantação de uma *network slice*, incluindo topologia e funções que serão implantadas, requerimentos de rede, temporais, de geolocalização e operacionais, formando assim um *template*, conforme Figura 3. Durante o estabelecimento da conexão são trocadas informações entre o UE e *Access and Mobility Management Function* (AMF). Durante esta troca, o UE fornece um número *Network Slice Selection Assistance Information* (NSSAI) e, com este número, o AMF aciona a função *Network Slice Selection Function* (NSSF), o qual verifica se este UE tem permissões para utilizar os recursos solicitados. Em caso positivo, com base no tipo de *slice* solicitada, esta é criada e disponibilizada para o *tenant*. Durante

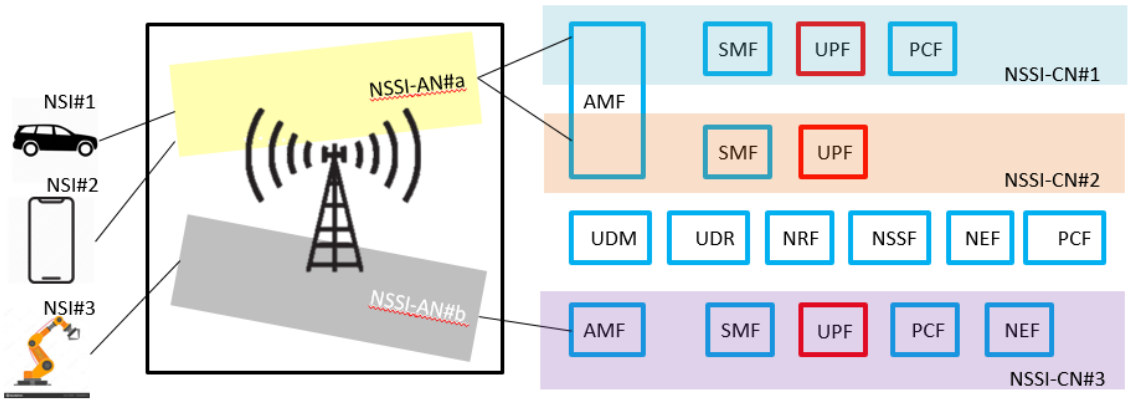


Figura 2 – *Network Slice no 5G* (Elaborada pelo autor).

a criação da *slice*, apenas os serviços e funções necessárias são implantadas, conforme o *template* selecionado com base no tipo de *slice* relacionado com o NSSAI fornecido pelo UE.

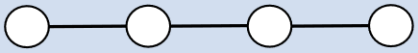
Fields	Attributes
NSL Topology	
NSL Network Requirements	• **Effective Throughput** • **Latency** • **Reliability** • **Number of devices** • **Security:** Confidentiality, integrity, ... • **Coverage** • **Mobility** • ...
NSL Temporal Requirements	• **Time intervals to be active**: From [dd/mm/yyyy] to [dd/mm/yyyy] • **Time intervals to be inactive**: From [dd/mm/yyyy] to [dd/mm/yyyy] • ....
NSL Geolocation Requirements	• **Location**: City (Cities), Country • ....
NSL Operational Requirements	• **Capability exposure for visibility and management**: Only monitoring / monitoring + limited management / monitoring + full management • **Priority level** • **KPI monitoring**: metric presentation, reporting period, ... • **Accounting**: online / offline • ....

Figura 3 – *Slice template* (ORDONEZ-LUCENA et al., 2018a).

O sistema 5G suporta serviço de transmissão de PDU, que provê troca de PDUs entre UE e uma rede de dados identificada pelo nome da rede de dados (do inglês *Data Network Name* (DNN)).

Na Figura 4, podemos ver que uma PDU é transportada entre UE para a RAN utilizando um *Data Radio Bearer* (DRB). É importante citar que já na RAN serão criados e anexados à pilha de protocolos do usuário um cabeçalho IPv6 (o qual contém endereçamento IPv6 referente à rede 5G), um cabeçalho TCP com porta de destino 2152 (porta de identificação do protocolo subjacente *GPRS Tunnelling Protocol for the user plane* (GTP-U)) e por fim um cabeçalho GTP-U. Somente após esta operação o pacote é

encaminhado à UPF via GTP. Todos estes cabeçalhos são necessários apenas dentro do 5GC, e a UPF é responsável por retirar estes cabeçalhos antes de encaminhar o pacote para o seu destino (3GPP, 2019).

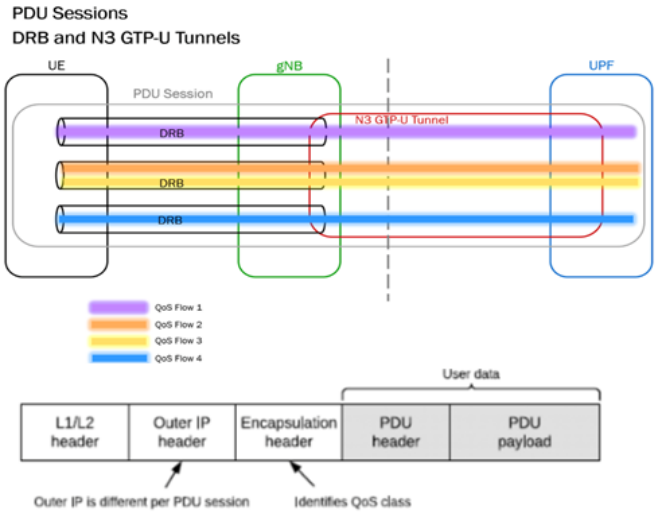


Figura 4 – PDU *sessions and tunnels* (3GPP, 2019).

O GTP *tunnel* é identificado por um campo no cabeçalho chamado TEID. Desde a RAN até o último UPF o pacote trafegará dentro destes túneis. Cada túnel identificado com um TEID representa também uma *slice*.

O DRB e GTP *tunnels* são pré-estabelecidos durante a conexão entre UE e 5GC, tendo assim uma conectividade fim a fim entre UE e DNN, como podemos ver na Figura 5. A instanciação e sincronismo destes serviços é chamada de PDU *session*, e pode ser criada, atualizada ou excluída de acordo com requisitos pré-estabelecidos (3GPP, 2019).

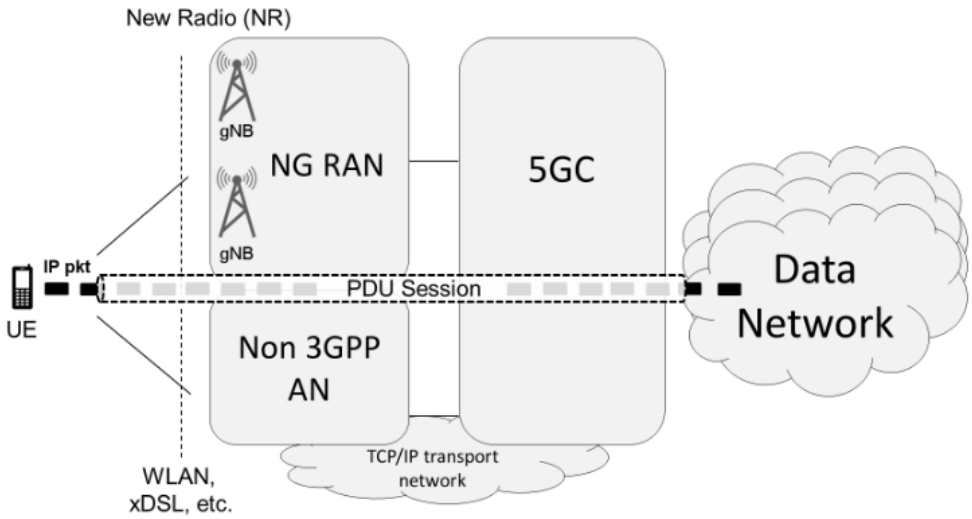


Figura 5 – PDU *sessions* (DETTI, 2018).

2.2 Service function chaining

Para garantir serviços de comunicação ultra confiáveis e latências ultra baixas, as arquiteturas 5G devem ser elásticas, ágeis, programáveis, eficientes e também devem suportar o crescimento do tráfego de dados. Atualmente, ainda existem muitas arquiteturas que utilizam dispositivos de *hardware* aplicados na rede para oferecer serviços específicos (*firewalls, WAN optimizer, proxies*, etc). Esses dispositivos são chamados de *middleboxes* e são tipicamente caros, proprietários, e devem ser implantados em lugares específicos exigindo treinamento para implantação e manutenção (Alameddine; Qu; Assi, 2017).

*Network Function Virtualization*s (NFVs) têm sido propostas como uma solução às limitações dos *middleboxes* dentro das redes 5G. NFVs oferecem funções de redes virtuais, ou VNFs (do inglês *Virtual Network Function*s), e podem exercer as mesmas funções oferecidas pelos *middleboxes*, mas desacopladas do *hardware*. Desta maneira, VNFs podem ser implantadas em qualquer lugar da rede a qualquer momento, sem necessidade de instalação de um novo *hardware*, garantindo flexibilidade e custo benefício viáveis (Taleb; Ksentini; Jantti, 2016).

Para oferecer comunicação fim a fim com latências ultra baixas, VNFs precisam garantir o processamento rápido de dados em tempo real. Esse processamento pode precisar de uma ou mais funções de rede, ou NFs (do inglês *Network Function*s). O encadeamento de NFs é conhecido como encadeamento de funções de serviço, ou SFC (Han et al., 2015).

Uma SFC consiste em um conjunto de *Virtual Network Function*s (VNFs) interconectadas por enlaces lógicos. Múltiplos SFCs de clientes distintos podem compartilhar recursos de rede e computação para melhorar a utilização de recursos (Fan et al., 2017).

A vantagem do uso de SFCs é a redução da complexidade na implantação e gerenciamento de funções virtuais que devem ser executadas em ordem específica para a valoração de serviços na rede. Também é possível automatizar de forma dinâmica a ordem dos serviços e suas interconexões (BHAMARE et al., 2016).

Podemos visualizar todos estes conceitos na Figura 6. XYZ.com é um *web service* (ou seja, um *network service*). Este serviço conta com duas *network functions* (*proxy* e *web server*). Por último, observando-se o fluxo dos pacotes, observamos que trata-se de uma SFC.

2.3 Segment routing over IPv6 (SRv6)

SRv6 é a instanciação do *Segment Routing* (SR) no plano de dados IPv6 (FILSFILS et al., 2018). Com o constante crescimento do tráfego e requisitos de latência mais exigentes, têm-se tornado um desafio operar as redes móveis. Além disto, o plano de usuário rígido tem tornado ainda mais difícil para operadoras otimizar e operar o plano de dados. Em paralelo,

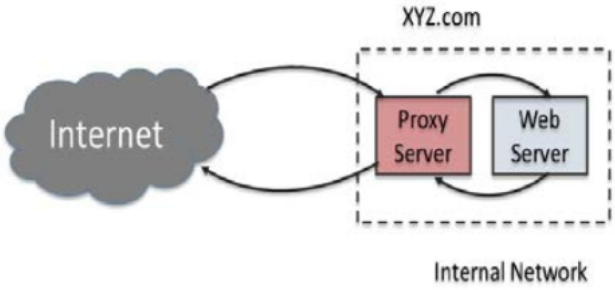


Figura 6 – *Network service, network functions e service function chain* (Elaborada pelo autor).

o IPv6 têm sido adotado pelas operadoras de rede como seu protocolo de transporte. SRv6 integra a aplicação e a camada de transporte subjacente em um único protocolo, permitindo aos operadores otimizar suas redes de forma simples e retirando o estado do encaminhamento da rede (MATSUSHIMA et al., 2019).

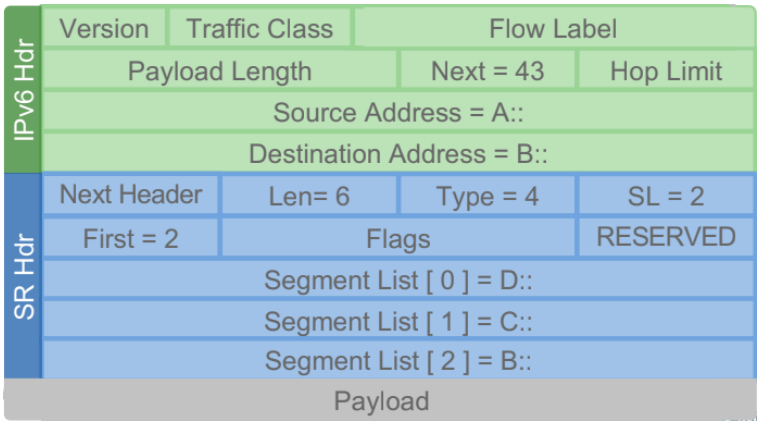


Figura 7 – IPv6 + SRv6 Header (Elaborada pelo autor).

Como podemos ver na Figura 7, o cabeçalho SRv6, chamado de *Segment Routing Header* (SRH), é uma extensão do protocolo IPv6. o SRH contém uma *segment list*, no qual cada segmento nesta lista, também chamado de SID (do inglês, *Segment Identification*), corresponde a um endereço IPv6 pelo qual o pacote transitará. Esses endereços são representados na Figura 7 como D::, C:: e B::. No SRH também existe um apontador (*Segment Left* (SL)) para identificar qual segmento está ativo. Quando este pacote passa por um *segment endpoint* (nó *SR-capable* que corresponde ao segmento ativo) o ponteiro é decrementado, e o novo segmento é então copiado para o campo endereço de destino do IPv6. Portanto, a ordem dos segmentos pelos quais o pacote passará é inverso à ordem numérica da *segment list*. Caso o pacote passe por um dispositivo que não reconheça o protocolo SRv6, o encaminhamento é feito normalmente utilizando o endereço de destino presente no IPv6. Outros campos que estão presentes no SRv6 são:

Para ilustrar o funcionamento do SRv6, vamos analisar a Figura 8. Esta figura

Tabela 3 – Campos presentes no SRH (adaptada de Bosshart et al. (2014)).

Next header	Identifica o tipo de cabeçalho subjacente ao SRv6.
Lenght	Tamanho do cabeçalho de roteamento em unidades de 8 octetos, não incluindo os 8 primeiros.
Routing type	Identifica a variação de cabeçalho de roteamento em particular.
First	Identifica o primeiro segmento da lista.
Flags	Espaço reservado para definição de novas *flags*.
Reserved	Espaço reservado.

apresenta os SR *nodes* A, B, C e D. Além disto, há também um nó que não reconhece SRv6, entre os nós A e B. Os números 1, 2 e 3 no canto superior de cada quadro representam um salto no encaminhamento. Na linha em verde temos o IPv6 com SA e DA sendo *source address* e *destination address*, respectivamente. Na linha em azul temos o SRv6. Entre parênteses, temos a *segment list*, ou seja, os nós pelos quais o pacote passará. Os nós são apresentados sempre do último para o primeiro. Neste exemplo, o pacote passará pelo nó B, depois pelo C e por último, pelo D. SL representa o nó ativo, e seu valor inicial sempre será o numero de nós (n) menos 1, e este valor é decrementado a medida que passa por um SR *node*. Cada vez que um nó passa por um SR *node*, além do decremento em SL, o segmento referente àquele SL é copiado para o IPv6 DA. Por exemplo, no quadro 1, o pacote deixa o nó A com os campos DA = B e SL = 2. Como o próximo nó não reconhece o SRH, ele apenas encaminha para B, pois este é o endereço de destino do IPv6. Em B, já no quadro 2, há um decréscimo em SL, e o valor referente à este apontador (neste caso, o endereço C) é então copiado para DA, e o pacote é encaminhado para o próximo nó, e assim sucessivamente.

A criação do SRH é chamada de encapsulamento. Existem duas formas básicas, *encap* e *inline*. Levando-se em consideração um ambiente onde temos na rede um cabeçalho IPv6 já em tráfego, o modo *inline* aproveita este cabeçalho já existente e cria o SRH logo após ele. Já no modo *encap*, ao invés aproveitar o cabeçalho IPv6 já existente, é realizada a criação de um novo cabeçalho IPv6, o qual é estendido com o SRH. O protocolo SRv6 está disponível no kernel linux a partir da versão 4.10. Apesar de ser possível utilizar o encapsulamento utilizando os dois modos, é necessário também realizar o desencapsulamento, ou seja, retirar o cabeçalho SRH após a ultima função. Atualmente, apenas o desencapsulamento do modo *encap* está implementado, inviabilizando, de certo modo, o modo *inline*.

2.4 Programming protocol-independent packet processor - P4

Pelo grande volume de dados que precisam ser transportados atualmente pelas redes de computadores surge a necessidade de tornar as redes flexíveis e programáveis. Até

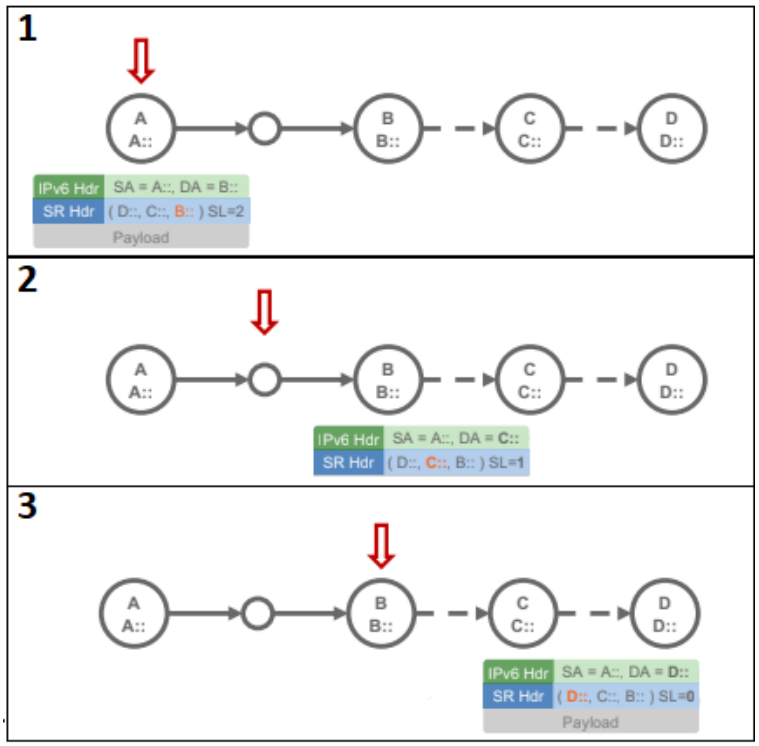


Figura 8 – Exemplo de encaminhamento SRv6 (Elaborada pelo autor).

bem pouco tempo atrás tínhamos um plano de dados fixo e engessado, que impossibilitava a construção de redes de acordo com requisitos próprios. Neste cenário surgiu a SDN, que é definida pela *Open Networking Foundation* (ONF) como uma arquitetura emergente, dinâmica, gerenciável e adaptável, permitindo a separação entre o controle da rede e as funções de encaminhamento. Essa separação faz com que o controle da rede seja diretamente programável e a infraestrutura subjacente possa ser abstraída para o desenvolvimento de novas aplicações e serviços.

Dentro deste paradigma existe o protocolo *Open Flow* (MCKEOWN et al., 2008), que permite que o plano de dados da rede seja configurado ou manipulado através de programação via software. O *Open Flow* especifica explicitamente os cabeçalhos de protocolo nos quais ele opera. Porém ainda não oferece flexibilidade suficiente para se adicionar novos cabeçalhos e definir novas ações após um *flow matching*.

Neste cenário surge a linguagem P4, que é uma proposta a permitir a programação do plano de dados de dispositivos de rede e também a programação de processadores de pacotes independentes de protocolo, que trabalha em conjunto com protocolos de controle de SDN, permitindo criar soluções customizadas (BOSSHART et al., 2014).

A linguagem P4 ajuda a superar as limitações do *Open Flow*, aumentando a programabilidade das SDNs. Ela permite que o projetista recupere o controle do plano de dados e considere funções otimizadas para uma rede específica e possui três objetivos principais, conforme Tabela 4.

Tabela 4 – Objetivos da linguagem P4 (Adaptada de Bosshart et al. (2014)).

Reconfigurabilidade	Está relacionada com a habilidade do programador redefinir o processamento dos pacotes nos dispositivos de rede.
Independência do Protocolo	Define que os dispositivos de rede não devem ser amarrados a formatos de pacotes específicos.
Independência do Alvo	Relaciona-se com o fato de que o compilador da linguagem de programação de redes deve gerar uma descrição independente do dispositivo, porém levando em conta as capacidades do hardware alvo específico na hora de configurá-lo.

Com base neste contexto, vários benefícios da programabilidade do plano de dados podem ser percebidos, dentre eles:

- O controle total da programabilidade do plano de dados, de tal forma que o dispositivo de rede se comporta exatamente como desejado;
- Capacidade de adicionar novas funções, uma vez que o dispositivo é completamente programável;
- Exclusividade, porque os usuários têm a capacidade de implementar protocolos personalizados;
- Eficiência, uma vez que, considerando que os dispositivos atualmente têm capacidade limitada e funções fixas, muitos dessas não são usadas e, portanto, consomem recursos sem nenhuma finalidade;
- A confiabilidade constitui outro benefício, porque é possível não usar funções implementadas por terceiros;
- O monitoramento no plano de dados permite olhar para dentro do dispositivo. Por exemplo, é possível ver os cabeçalhos dos pacotes que estão sendo processados e exportar esses dados para obter indicadores de desempenho.

A linguagem P4 possibilita a implementação de *switches* mais flexíveis, possibilitando ao programador decidir como o plano de dados processa os pacotes. Para isto, utiliza-se das definições na Tabela 5.

Os itens da tabela acima podem ser visualizadas na Figura 9, onde também fica claro a profundidade das possibilidades oferecidas pela P4, onde na parte superior da figura (*Switch configuration*) podemos ver o que é permitido que o programador controle via código P4.

Tabela 5 – Definições de elementos (Adaptada de Bosshart et al. (2014)).

Cabeçalhos	Descrevem a sequência e a estrutura de uma série de campos.
Analisadores (*parsers*)	Definem como os campos de cada cabeçalho são identificados e validados.
Tabelas (*tables*)	São mecanismos utilizados para realizar o processamento dos pacotes
Implementação de programas de controle (*control programs*)	Determinam a ordem das tabelas que são aplicadas a um pacote.

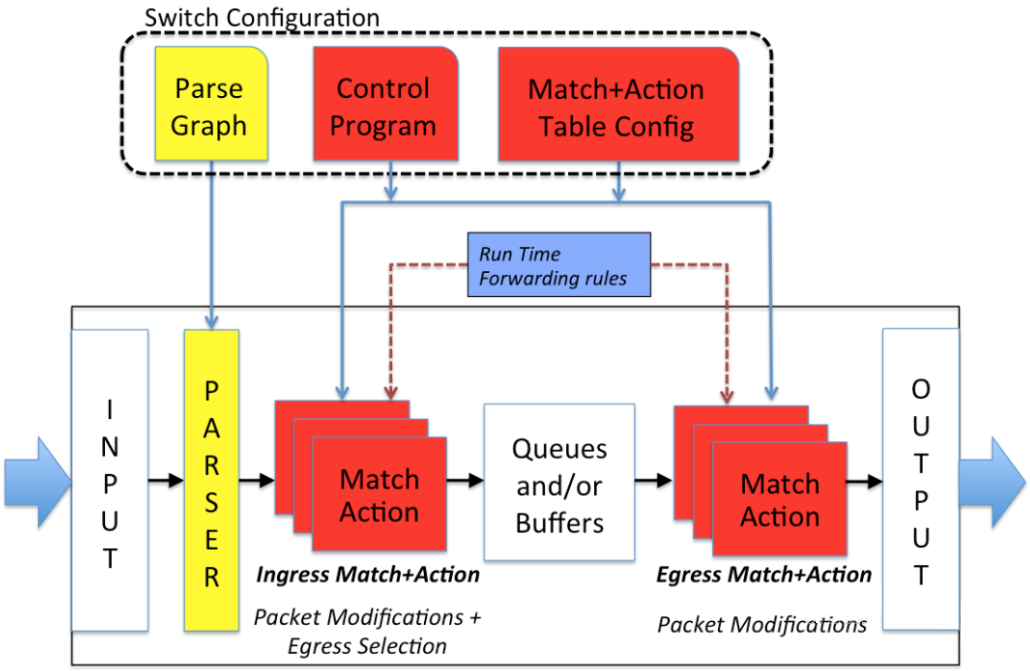


Figura 9 – Modelo abstrato (The P4 Language Consortium, 2015).

A linguagem P4 permite definir como os pacotes devem ser processados. Um dispositivo que possa executar um código P4 é chamado de alvo (do inglês *target*), e este dispositivo deve oferecer um compilador, que transformará as configurações do código P4 para o código de máquina. Com isto, o código P4 é independente de alvo, pois o mesmo código pode ser compilado por vários compiladores diferentes e ser executados em dispositivos com diferentes arquiteturas. A Figura 10 mostra este caminho de forma clara, além de um exemplo de código em P4. Podemos notar nesta figura a presença do P4 *program*, que é fornecido pelo usuário, e também do P4 *architectural model*, ou seja, um compilador, que é fornecido pelo fabricante do *target*. Quando o programa é compilado, gera-se o plano de dados definido pelo programa escrito pelo usuário, com a possibilidade de atualizações em tempo real, além de um *driver* que será utilizado para comunicação entre o plano de controle o plano de dados.

A linguagem P4 tem crescido de forma eloquente por ser muito adaptável, podemos por exemplo criar um *switch*, testá-lo em um ambiente emulado no *Mininet* (KAUR; SINGH;

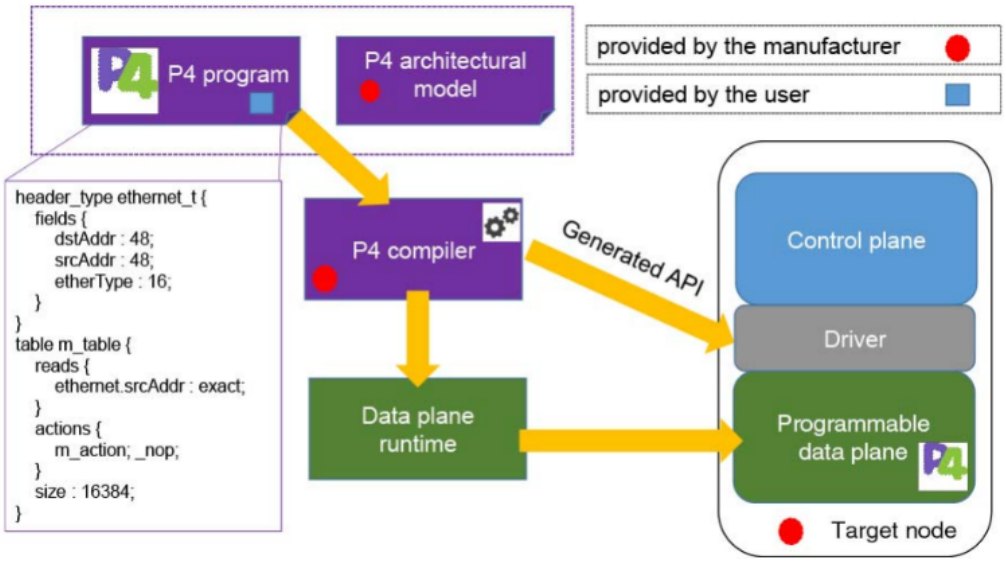


Figura 10 – Compilação P4 (PAOLUCCI et al., 2019).

GHUMMAN, 2014) e colocá-lo em produção virtualizando um *switch BMv2* (Behavioral Model v2, 2021). Para os casos mais extremos, os próprios autores da linguagem P4 criaram um chip ASIC (*Application Specific Integrated Circuits*) chamado *Tofino* (Barefoot (Intel) Torino Chip, 2021), o qual é usado em *switches* programáveis de alta performance por empresas como *Edgecore*, *Inventec*, *Stordis* e WNC. Alguns exemplos de *switches* que utilizam ASIC *Tofino* são STORDIS BF6064X-T (APS-NETWORKS, 2021) e Winstron OSW1800 (Wistron NeWeb Corp., 2021).

3 Trabalhos Relacionados

Este capítulo apresenta alguns artigos relacionados com grande relevância para o tema desta dissertação.

O protocolo de tunelamento GTP-U foi implementado no plano de usuários dos sistemas *Global System for Mobile Communications* (GSM), *Universal Mobile Telecommunications System* (UMTS) e 4G *Long Term Evolution* (LTE). Apesar do protocolo SRv6 ter sido proposto como um protocolo alternativo no UP pelas vantagens que oferece, não é viável trocar o prototocolo GTP-U pelo SRv6 por conta da dependência ainda existente de vários nós pelo GTP-U. Para suportar a coexistência destes dois mecanismos no UP, soluções que realizam tradução sem armazenamento de estados foram propostas. O trabalho Lee et al. (2019) faz uma avaliação quantitativa para validar a tradução de SRv6 para GTP-U e vice-versa. Os autores desenvolveram e implementaram funções de tradução sem armazenamento de estado entre GTP-U e SRv6 com mínimo de recursos em um *switch* programável. Os autores injetaram *timestamps* nos cabeçalhos dos pacotes para servir de telemetria para a medição. Também foram utilizadas métricas conhecidas como vazão e perda de pacotes por segundo em condições leves (100mbps) e pesadas (100gbps). As funções utilizadas fazem tradução de GTP-U *over* IPv4 ou IPv6 para SRv6 e vice-versa. Os autores utilizaram um *switch* com placa ASIC Tofino no qual foi implantado o código P4 desenvolvido. Os testes mostram que não houve perda de dados e não há mudanças abruptas de desempenho. Os autores consideraram apenas a tradução do protocolo GTP-U para SRv6 de forma que o pacote transitasse no UP dentro do protocolo SRv6. Nosso trabalho não fará esta tradução, e sim construirá um cabeçalho SRv6 estendendo o IPv6 entre a RAN e UPF para facilitar o SFC entre estes dois pontos. O SRv6 será retirado ao final do encadeamento das funções e o pacote em seu formato original será encaminhado ao UPF, mantendo-se então o GTP-U no UP.

Com o surgimento de paradigmas como SDN, que propõem a separação dos planos de dados e controle, surgiram também projetos deste paradigma aplicados às redes móveis, conhecido como *Control and User Plane Separation* (CUPS). Um exemplo é o 4G *Long Term Evolution Evolved Packet Core* (LTE EPC), que consiste em funções de controle para autenticação de usuários e configuração de sessões de dados e funcionalidades no UP para encaminhamento de pacotes. Com o 5G, o UP pode ficar ainda mais perto dos usuários finais, para diminuir a latência no encaminhamento. O trabalho de Shah et al. (2020) faz uma revisão das implicações deste modelo tendo como motivação o crescimento rápido do tráfego de sinalização e também a alta taxa de procedimentos de sinalização que ocorrem no 5GC . Os autores propõem um *offloading* dos procedimentos de sinalização que mais acontecem no 5GC para o plano de dados, de modo a melhorar a vazão e diminuir

a latência da sinalização. Para isto, foram utilizados *switches* P4 programáveis, os quais foram programados para armazenar estados e processar mensagens de sinalização ainda dentro do plano de dados. Este trabalho tem como ponto em comum a utilização de *switches* P4 programáveis no plano de dados, mas diferentemente deste trabalho, nos propomos a criar um ponto de entrada entre a RAN e UPF e UPF e DNN, trabalhando com pacotes SRv6, para facilitar o SFC.

De acordo com o paradigma de 5G *slices*, uma única infraestrutura de rede é particionada em sub-redes virtuais adaptadas aos requisitos das aplicações, atingindo um balanceamento entre custo (compartilhado) e performance (dedicado). *Slices* são implementadas como o agrupamento de componentes executados em diferentes plataformas e interconectadas pela rede de transporte. O trabalho de Borsatti et al. (2019) utiliza SFC para gerenciar o ciclo de vida das *slices*, por meio da adotação do *European Telecommunications Standards Institute* (ETSI) NFV *Management and Orchestration* (MANO) *framework*, sendo o SRv6 utilizado para alcançar este objetivo. Utilizaremos SRv6 para alcançar a técnica de SFC para encaminhamento do pacote para outras NFs que agregarão valor ao serviço, deixando a criação e gerenciamento de *slices* sob responsabilidade do sistema 5G existente. Além disto, utilizaremos a linguagem P4 para criação de um dispositivo programável, o que não foi utilizado no trabalho.

O trabalho de Shen et al. (2019) utiliza P4 para construir *switches* programavéis que estão localizados entre o *backhaul* (rede de acesso) e o *Serving Gateway User plane function* (SGW-U) (que nas redes 4G desempenham funções similares ao UPF). Estes *switches* possuem capacidade para realizar *offloading* das funções do GTP, ou seja, sendo capazes de realizar o todo o processo de encapsulamento e desencapsulamento dos pacotes em túneis GTP, aliviando assim as CPUs das VMs. Resultados experimentais verificaram a funcionalidade da proposta. Também foi visto que este modelo alcançou vazão de 10gbps por porta. Apesar do texto citar que o trabalho é aplicado às redes 5G, na verdade, pela verificação da infraestrutura citada, notamos que se trata de uma rede 4G LTE. Apesar disto, o trabalho tem como ponto em comum a aplicação de *switch* programável no UP. Nosso trabalho se diferencia pois é focado em construir um cabeçalho SRv6 estendendo o IPv6, e com isto, possibilitar o SFC dentro do 5GC.

No trabalho de Ricart-Sanchez et al. (2019), os autores implementaram um sistema baseado em NetFPGA (*Field-Programmable Gate Array*) e NetFPGA-SUME para suportar um *framework* de definição e controle de network *slices* em uma arquitetura 5G *Mobile Edge Computing* (MEC). Este sistema é responsável por criar e gerenciar *slices* entre a RAN e 5GC. Muito embora o trabalho citado utilize também a linguagem P4, utilizaremos o campo TEID presente no cabeçalho GTP para identificar uma *slice* dentro do 5GC. A escolha por este campo foi definido após longa pesquisa e é de suma importância para este trabalho. O TEID é o único meio para identificar uma *network slice* no UP. por isto,

faremos uso deste recurso para identificar e tratar o encaminhamento de *network slices*, mas mantendo a criação e gerenciamento delas por conta de seus provedores.

Confome podemos ver na Tabela 6, os trabalhos anteriormente apresentados mostram o uso de várias técnicas e tecnologias, mas apesar de várias trabalharem com SRv6 ou GTP de variadas formas, nenhum deles se propõe a criar um mecanismo para habilitar o SFC utilizando SRv6, mantendo-se o GTP.

Identifica-se a necessidade do avanço proposto pelo INCA pela apresentação de uma função para SFC que leve em consideração as limitações referentes à dependência do 5GC ao protocolo GTP, já que criaremos o SRv6 apenas estendendo o protocolo IP já existente e mantendo o GTP.

Tabela 6 – Tecnologias e conceitos utilizados (Elaborada pelo autor).

Autores	Lee et al. (2019)	Shah et al. (2020)	Borsati et al (2019)	Shen et al (2019)	Ricart-Sanches et al (2019)	INCA
5G	X	X	X		X	X
GTP	X	X		X	X	X
SRv6	X		X			X
SFC			X			X
DATAPLANE with P4	X	X		X	X	X

4 Projeto e Implementação

Com o objetivo de realizar SFC dentro do 5GC de forma totalmente transparente, todo o projeto e implementação do INCA foi desenhado de forma que não houvesse impacto às funções adjacentes. Além disto, utilizamos apenas softwares *open source* no desenvolvimento do INCA.

O SFC fornece suporte para a criação de serviços compostos que consistem em um conjunto ordenado de funções de rede. O SFC pode ser implementado com base em várias tecnologias, como *Network Service Header* (NSH) e SRv6. Dessa forma, uma aplicação clara do modelo de protocolo SRv6 consiste em direcionar pacotes por meio de um SFC (ABDELSALAM et al., 2017). Por isto, escolhemos usar este protocolo no desenvolvimento do INCA.

4.1 Visão geral da proposta

Este trabalho propõe a construção de um dispositivo que faz uso da linguagem P4 para construir uma solução que utiliza o protocolo SRv6 para resolver a falta de um mecanismo capaz de encadear funções de serviço no 5GC inteiramente no plano de dados.

De acordo com os principais aspectos do 5G, certas funções podem ou não ser alocadas aos clientes, e o que determina quais são alocadas é o tipo de tráfego, *slice* ID e outras necessidades relacionadas ao serviço solicitado. Nossa solução se destina a ser implantada dentro do 5GC, de forma que possa ser alocada sob demanda, ou seja, somente se houver necessidade de realização de SFC. Para isto, é necessário um certo nível de autonomia, de forma que a criação e destruição do SRH e o roteamento para funções não dependam ou impactem as funções já existentes. Portanto, o cabeçalho SRv6 é um candidato chave para manter a transparência respeitando a pilha TCP/IP existente.

Um ponto fundamental levado em consideração é que SRv6 não pode ser usado como protocolo de transporte de plano de usuário, uma vez que outras funções ainda dependem do GTP, que é o protocolo mais comum usado de comunicação entre RAN e UPF. Como tal, o INCA é responsável por criar e remover o SRH antes de encaminhar o tráfego GTP original para o UPF. O SRv6 é criado, a lista de segmentos é adicionada e o fluxo de tráfego é encaminhado para as *Virtual Function*s (VFs) de forma totalmente independente e transparente. Fazendo isso antes do UPF, habilitamos o SFC dentro do 5GC sem a necessidade de alterar o protocolo de transporte original ou outras funcionalidades predefinidas.

Na construção do INCA optamos pelo modo SRv6 *inline*, pois oferece a vantagem

de utilizar o cabeçalho IPv6 externo, que já existe no 5GC, evitando assim a criação de outro cabeçalho IPv6. Embora o modo *inline* seja mais interessante dentro do 5GC, estendendo o SRH a partir do cabeçalho IPv6 existente, evitando a criação de um novo, este modo também impõe um desafio. Como uma escolha natural, o sistema operacional Linux é usado para hospedar as VFs, agindo como o roteador compatível com SRv6 e host para as VFs. Por padrão, a última função é sempre responsável por remover o SRH. Porém, o kernel Linux atualmente suporta apenas o modo *encap*, ou seja, quando o SRH é removido, o cabeçalho IPv6 também é removido, o que causaria problemas dentro do 5GC. Resolvemos este problema codificando o modo *inline* em P4 e deixando-o sob a responsabilidade do INCA, de forma que agora é possível remover o cabeçalho SRv6 sem alterar o cabeçalho IPv6 original.

4.2 Implementação

O INCA foi totalmente implementado utilizando a linguagem P4. Adicionamos todos os cabeçalhos usados no 5GC e também o SRv6. Configuramos o *parser* para que o INCA aceite pacotes no padrão 5G, com ou sem o cabeçalho SRv6. A Figura 11 mostra o fluxo de processamento do INCA, que indica as ações a serem tomadas: adicionar o SRH (Add SRv6) ou eliminar o SRH (Drop SRv6). Os modelos de pacote que são suportados podem ser vistos na Figura 16, itens 2 e 4, que são respectivamente a pilha de protocolo 5GC padrão e a pilha padrão com a adição do SRH. Para simplificar, o cabeçalho Ethernet não aparece na Figura 16, no entanto, é o primeiro cabeçalho a ser aberto pelo INCA. Após a identificação dos cabeçalhos, caso esteja dentro dos padrões citados acima, o pacote é aceito e então entra na etapa de processamento de ingresso, onde para cada situação há uma ação:

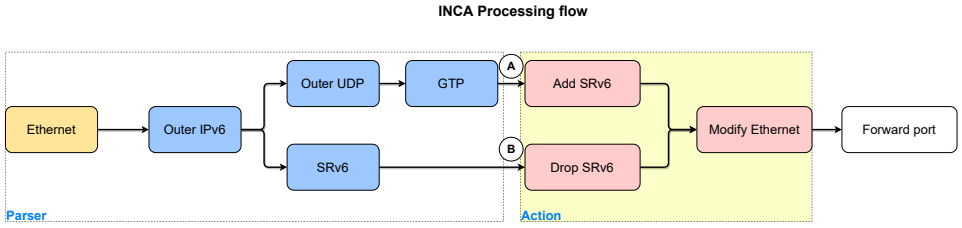


Figura 11 – INCA *processing flow* (Elaborada pelo autor).

- Caso A - Se o pacote não contém um SRH, é feita uma busca nas tabelas configuradas pelo plano de controle para verificar se existem regras que se aplicam a este pacote. Se existir, o SRH é criado e o pacote é encaminhado para a primeira VF. Neste caso, a tabela em questão chama-se *main*, tal como pode ser vista na Figura 12. Caso aconteça um *matching*, a função *build_srv63* (Figura 13) será chamada e o SRv6 será criado.

```
236     table main {
237         key = {
238             hdr.gtp.teid: ternary;
239             hdr.pdu_container.qosid: ternary;
240             hdr.ipv6_inner.dst_addr: ternary;
241             hdr.ipv6_inner.src_addr: ternary;
242             hdr.ipv6_inner.next_hdr: ternary;
243             hdr.tcp_inner.dstPort: ternary;
244             hdr.tcp_inner.srcPort: ternary;
245             hdr.udp_inner.dport: ternary;
246             hdr.udp_inner.sport: ternary;
247         }
248         actions = {
249             build_srv63;
250             srv6_pop;
251         }
252         default_action = srv6_pop(64);
253         /*size = 1024;*/
254     }
```

Figura 12 – Tabela de verificação de políticas (Elaborada pelo autor).

```
215     action build_srv63(ip6Addr_t s2, ip6Addr_t s3) {
216         hdr.srv63.setValid();
217         hdr.srv63.next_hdr = hdr.ipv6_outer.next_hdr;
218         hdr.srv63.hdr_ext_len =  LEN;
219         hdr.srv63.routing_type = TYPE_SR;
220         hdr.srv63.segment_left = SL;
221         hdr.srv63.last_entry = 2;
222         hdr.srv63.flags = 0;
223         hdr.srv63.tag = 1;
224         hdr.srv63.segment_id1 = hdr.ipv6_outer.dst_addr;
225         hdr.srv63.segment_id2 = s2;
226         hdr.srv63.segment_id3 = s3;
227         hdr.ipv6_outer.next_hdr = TYPE_SRV6;
228         hdr.ipv6_outer.dst_addr = s3;
229         hdr.ipv6_outer.payload_len = hdr.ipv6_outer.payload_len + 56;
230     }
```

Figura 13 – Função de construção do SRv6 (Elaborada pelo autor).

- Caso B - Se o pacote contém um SRH com campo SL igual a zero, significa que o pacote já passou por todas as funções. Neste caso o INCA descarta o SRH (modo *inline*) e encaminha o pacote para o UPF. Como estamos utilizando o modo *inline* e o próprio INCA é responsável por realizar a remoção do SRH, o INCA será sempre o último SID configurado. Desta forma, foi criada uma tabela (Figura 14) onde é realizada a verificação do endereço de destino IPv6 atual. Para que a função de remoção do SRH seja realizada, este deve ser igual ao endereço IPv6 referente ao INCA. Neste caso, a função *srv6_pop* (Figura 15) é chamada. Esta função remove o SRv6 e configura o IPv6 de forma que fique idêntico ao pacote recebido.

```
265     table pop {
266         key = {
267             hdr.ipv6_outer.dst_addr:exact;
268         }
269         actions = {
270             srv6_pop;
271             drop;
272         }
273         size = 1024;
274         default_action = drop();
275     }
```

Figura 14 – Tabela de verificação de última função (Elaborada pelo autor).

```
207     action srv6_pop (bit<8> hop) {
208         hdr.ipv6_outer.next_hdr = hdr.srv63.next_hdr;
209         hdr.ipv6_outer.payload_len = hdr.ipv6_outer.payload_len - 56;
210         hdr.srv63.setInvalid();
211         hdr.ipv6_outer.hop_limit = hop;
212         hdr.gtp.spare = 1;
213     }
```

Figura 15 – Função para retirar SRH (Elaborada pelo autor).

A Figura 16 mostra como nossa solução funciona. Após o usuário (UE) enviar um pacote (1) e este pacote ser encapsulado pela RAN (2), o INCA o recebe. Este cenário se encaixa no Caso A: um SRH é criado e o pacote é enviado para a primeira VF (3). Esta VF identifica o SRH, realiza sua operação (executa a VF), decrementa o campo SL, atualiza o destino IPv6 com base no novo SID (4) e envia o pacote para a próxima VF (5). Esta seqüência é então repetida, porém, como a VF B:: é a último da sequência, o pacote é então enviado de volta ao INCA (6). O INCA detecta que o campo SL é igual a zero (Caso B) e que não há mais VFs para percorrer. Assim, o INCA retira o SRH e encaminha o pacote para a UPF (7) que o encaminha para o destino final (8). Observe que o pacote encaminhado ao UPF é idêntico ao pacote recebido pela RAN.

Ainda no Caso A, como podemos ver na Figura 12, foram configuradas várias possíveis *keys*, ou seja, vários campos das camadas de transporte e rede do pacote do usuário ou do 5GC que podem ser usados como referências pelo plano de controle para identificar um tráfego. O *PDU Session User Plane Protocol* (ETSI, 2021) (nova extensão para o GTP criado especificamente para o 5GC) também foi implementado como um desses campos. O INCA já tem implementado todos os mecanismos necessários para a leitura deste protocolo e até mesmo para realizar o *matching* usando QoS ID, criando assim o SFC com base neste campo. Desta forma, o INCA é capaz de realizar SFC baseado em endereços IPv6, portas UDP ou TCP, slice ID, QoS ID ou qualquer combinação destes.

Todo o código e instruções para replicação do ambiente foram divididos em

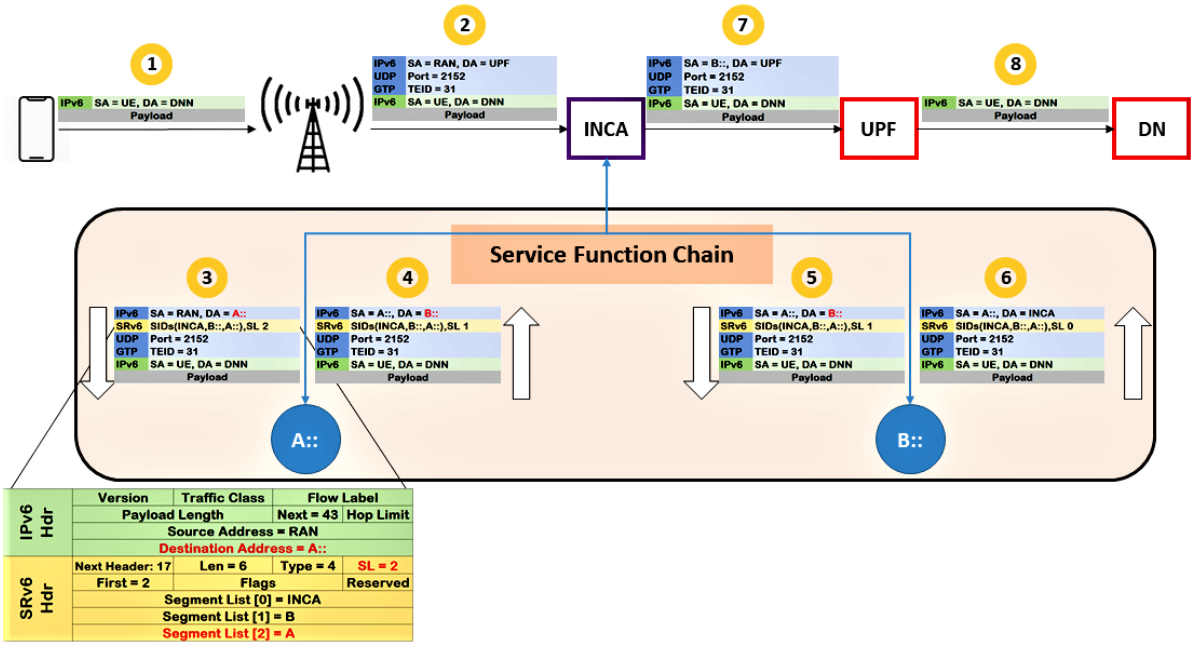


Figura 16 – INCA *working flow* (Elaborada pelo autor).

3 repositórios públicos. O primeiro (https://github.com/guimvmatos/P4-INCA.) contém o código do INCA, que pode ser implantado em BMv2 ou SmartNIC. O segundo (https://github.com/guimvmatos/P4-BMv2-RAN-UPF) repositório contém o código para construção de RAN e UPF utilizando *vagrant/virtualbox*. No terceiro (https://github.com/guimvmatos/SRv6-topology) encontra-se o código para construção do ambiente contendo as funções, cliente e servidor, completando assim o ambiente.

5 Implantação e avaliação

Neste capítulo, apresentamos os detalhes de como foi realizada a implantação e avaliação do INCA.

5.1 Implantação

O INCA foi implantado em uma Netronome Agilio CX SmartNIC compatível com P4 contendo duas portas de 10 Gbps cada (Figura 17). Também usamos máquinas virtuais (VMs) com sistema operacional Linux para o emular o ambiente 5G e para hospedar as VFs.

Usando *Single-root input/output virtualization* (SR-I/OV), a placa Netronome é capaz de criar VFs logicamente isolados de *Phisical Function* (PF) para compartilhar os recursos físicos. Normalmente, VFs são atribuídas às VMs. O acesso às VMs é feito diretamente via PCI, evitando o host do kernel. Nossa solução usa cinco interfaces virtuais (VF0_1 - VF0_5) compartilhadas a partir das duas interfaces físicas existentes.

Figura 17 – Netronome Agilio CX 2x10GbE SmartNIC (Open-NFP, 2021).

De acordo com a Figura 18, foi definida a seguinte configuração: sete VMs, três delas são funções de rede (NFV1 é um *Intrusion Detection System* (IDS), NFV2 é um *Intrusion Prevention System* (IPS) e NFV3 é um *Firewall*). Uma VM para o UE, que envia tráfego para DN, e outra VM para DN, que responde às solicitações enviadas pelo UE. Por último, temos as funções nativas do 5GC: RAN e UPF. Esses dois últimos realizam o encapsulamento e o desencapsulamento de pacotes em túneis GTP e, para isso, utilizamos um código P4 implantado em um switch virtual *Behavioral Model version 2* (BMv2) em execusão nessas duas VMs.

Foram configurados diferentes fluxos de forma a abranger a identificação do tráfego baseado em diferentes políticas, como criação do SRH baseado em endereçamento IPv6, 5-tupla, portas de serviços e identificador de slice (TEID).

As funções de rede foram configuradas de forma a exemplificar várias opções de encadeamento de diferentes cenários. Assim, foram montados dois cenários. No primeiro, a *SFC* é formada pelas funções de rede NFV1 e NFV2, que desempenham suas funções de IDS e IPS e também de processamento de *SRH*. No segundo cenário, o *SFC* é formado por NFV2 e NFV3. O último é um *firewall* configurado para bloquear o tráfego proveniente do UE. Usando esses dois cenários, podemos enviar fluxos e alterar dinamicamente as funções de rede para que um determinado tráfego seja permitido enquanto outro é bloqueado pelo *firewall*.

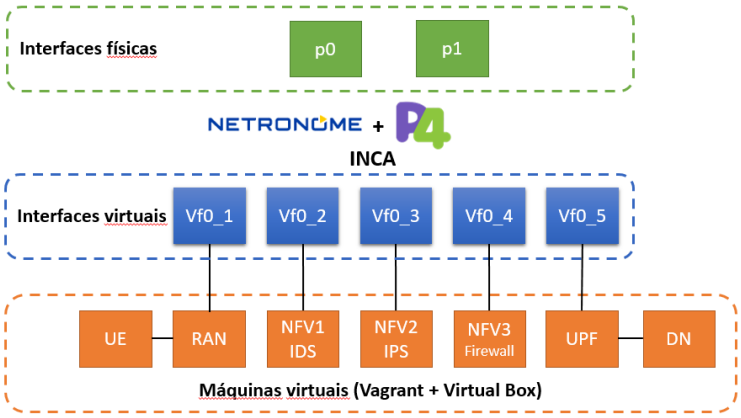


Figura 18 – Configuração do testbed utilizado nos testes (Elaborada pelo autor).

5.2 Avaliação

Nossos testes baseiam-se na avaliação do impacto do INCA no FCT em relação ao mesmo ambiente sem o uso do INCA. Configuramos o INCA para que as ações de encapsulamento (criação) e desencapsulamento (destruição) de SRH sejam feitas em sequência. Para que seja possível estabelecer um fluxo de dados idêntico entre o ambiente com e sem INCA, os pacotes não passam pelas funções de rede. Em vez disso, o pacote chega ao INCA onde o SRH é criado no processamento de ingresso. Logo após o processo de criação do SRH, ele é destruído e, em seguida, o pacote é encaminhado para o UPF.

Para realizar a avaliação, usamos Iperf (TIRUMALA, 1999) para enviar fluxos TCP de diferentes tamanhos: 10 MB, 50 MB, 100 MB, 150 MB e 300 MB. Os experimentos foram realizados 30 vezes para cada tamanho de fluxo em cada ambiente.

Para este trabalho, levamos em consideração um número razoável de três SIDs. As regras e políticas sobre o que fazer com os fluxos de tráfego podem ser configuradas pelo plano de controle para construir o cabeçalho SRv6 com um, dois ou três SIDs.

Nossos testes mostraram que a variação no número de SIDs criados não tem influência no FCT. Assim, escolhemos o ambiente onde são criados dois SIDs para realizar a comparação com um ambiente sem INCA.

As tabelas 7 e 8 mostram o resumo dos testes realizados e a Figura 19 mostra a distribuição e a comparação da FCT para cada tamanho de fluxo, com e sem INCA . No eixo X, temos o tamanho dos fluxos, e no eixo Y, os tempos de conclusão de cada teste em segundos. Os pontos em verde referem-se ao ambiente com resultados do INCA e em vermelho o ambiente sem ele.

Tabela 7 – Resumo dos testes sem INCA (Elaborada pelo autor).

Flow Size	**Min**	**Max**	**Mean**	**Median**	**Std**
10MB	57.6	66.4	61.37	61.3	1.77
50MB	295.5	310.7	302.92	302.65	4.82
100MB	578.8	622	601.9	602.55	11.07
150MB	862.9	942.7	903.46	903.4	17.71
300MB	1607	1829.5	1776.49	1784.85	49.03

Tabela 8 – Resumo dos testes com INCA (Elaborada pelo autor).

Flow Size	**Min**	**Max**	**Mean**	**Median**	**Std**
10MB	57.9	65.5	61.88	62.15	1.8
50MB	290.1	315.5	305.73	305.65	5.9
100MB	575.9	623.9	607.19	609.5	12.81
150MB	878.7	940	910.37	912	15.17
300MB	1532.5	1890.5	1791.89	1800.5	70.27

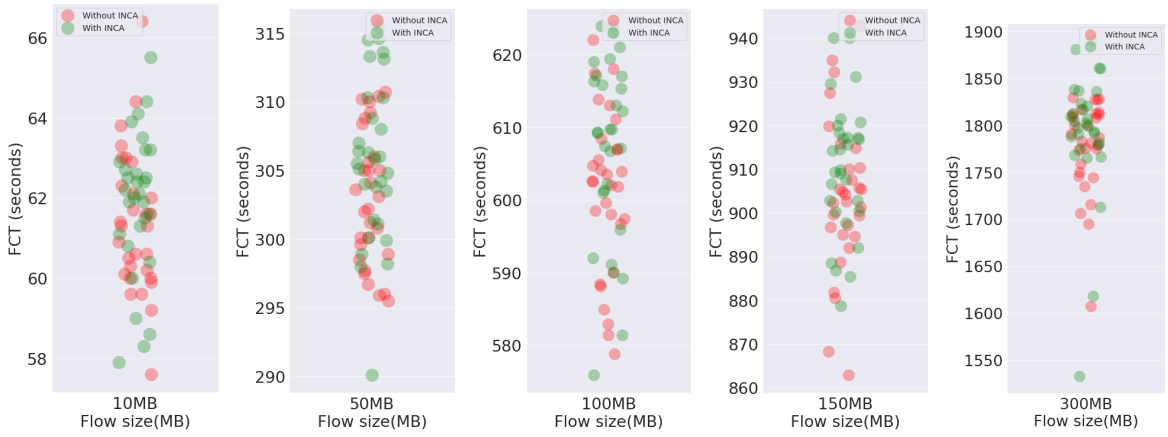


Figura 19 – Distribuição de FCTs por tamanho de fluxo (Elaborada pelo autor).

Analisando as Tabelas 7 e 8, podemos observar que a média, mediana e desvio padrão do INCA foram em sua maioria maiores quando comparados ao mesmo ambiente sem INCA, o que indica um FCT maior.

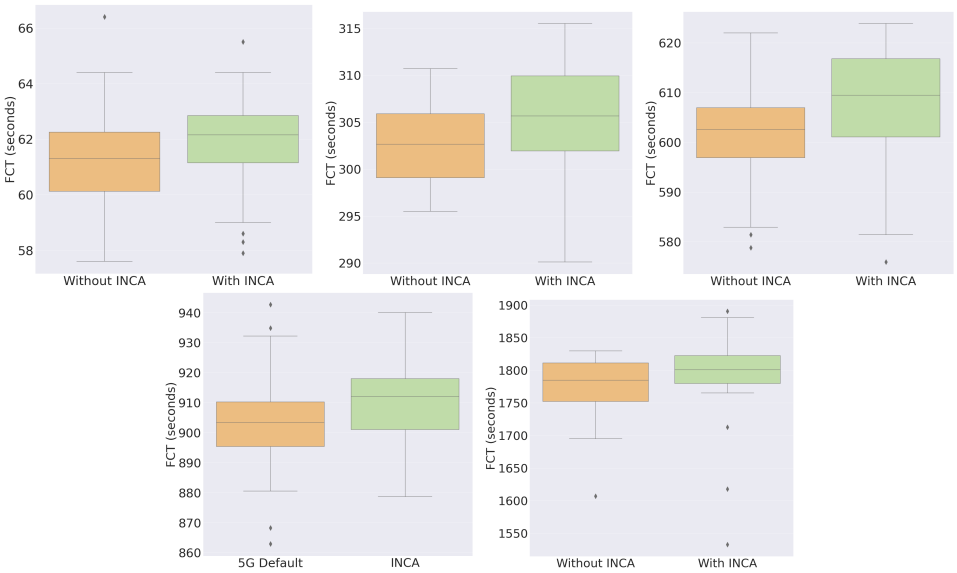


Figura 20 – Comparativo FCT (Elaborada pelo autor).

Quanto à amplitude dos resultados, ou seja, a diferença entre o maior e o menor resultado de cada fluxo, é possível observar na Figura 19 que ambos os ambientes apresentaram variações. Isso demonstra que as taxas de transmissão não permaneceram constantes durante os testes.

A Figura 20 mostra o FCT com e sem INCA. Podemos visualizar a mediana, dispersão dos resultados, intervalo interquartil e outliers. Este gráfico mostra que em todos os casos a mediana referente ao INCA é maior do que sem INCA. Além disso, podemos notar os seguintes padrões: a mediana do INCA está sempre próxima a Q3 (borda superior do retângulo) do ambiente sem INCA e Q1 (borda inferior do retângulo) do INCA sempre começa próximo à mediana do ambiente sem INCA. Isso demonstra que os intervalos interquartis (ou seja, a dispersão dos resultados) de ambos os testes são muito próximos.

Na Figura 21, é apresentada a distribuição dos valores relativos à diferença entre o ambiente com e sem INCA. No eixo X podemos ver a distribuição da diferença e no eixo Y os valores estão em porcentagem. De fato, olhando para esta figura, é possível observar que há um pequeno aumento entre 0,77% e 1,01% na FCT quando se utiliza o INCA.

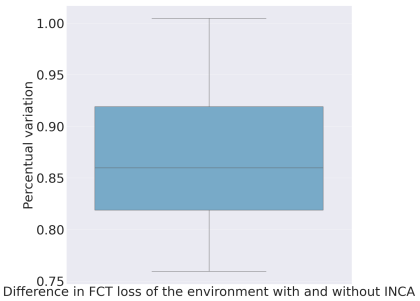


Figura 21 – Variação percentual (Elaborada pelo autor).

Os resultados mostram que o INCA é capaz de executar SFC dentro da arquitetura

5G de forma eficaz. Também foi possível observar que a criação de até três SIDs não altera a FCT. O INCA tem um acréscimo de 1% na FCT quando comparado com o cenário sem o INCA. Percebemos também que existe um padrão, onde os valores de FCT com e sem INCA são sobrescritos, de forma que os resultados sempre ficam dentro de uma faixa pré-definida onde é possível prever o comportamento, o que é uma característica importante para SLAs (Acordo de Nível de Serviço, do inglês, *Service Level Agreement*)

5.3 Limitações

Em um estudo preliminar não foi percebi diferença no FCT entre tráfegos com duas ou três funções. Portanto escolhemos para realização da avaliação utilizar duas funções. No entanto, a cada função pela qual os pacotes devem passar são acrescidos 128 bits no cabeçalho SRv6. Portanto, faz-se necessário um estudo para determinar até quantas funções o SRv6 pode suportar antes de tornar-se proibitivo em relação ao tamanho máximo de segmento.

6 Conclusão

A evolução das redes de computadores atualmente exige a existência de uma rede que atenda as demandas relacionadas à internet das coisas, serviço de transporte autônomo, cirurgia remota e controle remoto de operações industriais. Com isto surgiu a quinta geração do sistema de comunicações móveis, também chamado de 5G. Apesar do 5G contar com o que há de mais atual no estado da arte, como arquitetura baseada em serviços, redes definidas por software e separação de plano de controle e dados, ainda são necessárias algumas adequações para que alguns serviços possam ser ofertados.

Dentro do 5GC, uma miríade de funções (virtuais) podem ser implementadas para que diferentes tratamentos possam ser dados para o tráfego entre RAN e UPF. Portanto, é necessário uma função de encadeamento de serviços, que reduz a complexidade de gerenciamento das funções virtuais para que essas possam ser executadas em uma ordem específica, valorizando assim os serviços da rede.

Neste trabalho, apresentamos o desenho, implantação e avaliação de uma solução desenvolvida em P4 para identificação e encadeamento de tráfego usando SRv6. Chamamos essa função de INCA (In-Network Classification and ChAining). Esta função foi totalmente implementada no plano de dados usando uma placa de rede Netronome Agilo SmartNIC. O INCA é implantado imediatamente antes da UPF dentro do 5GC e é capaz de observar o tráfego que entra e sai da RAN para classificar e criar a sequência adequada de serviços a ser seguida por cada fluxo específico.

Nossos testes mostram que o INCA é capaz de realizar a identificação de pacotes, construindo e removendo o SRH, bem como realizar o roteamento entre as funções da rede. Os testes de desempenho mostram que o impacto do INCA no FCT é mínimo quando comparado com o mesmo ambiente sem INCA. Além disso, o uso de SRv6 sem alterar o GTP-U existente permite o uso irrestrito em arquiteturas 5G.

Adicionalmente, também realizamos a implementação do *PDU Session User Plane Protocol*, o novo protocolo definido para 3GPP para carregar as informações de QoS. Também criamos uma solução para realizar o desencapsulamento do SRH no modo *inline*.

Trabalhos futuros incluem a implantação do INCA utilizando um ambiente 5G real, além da realização de testes e avaliação da implantação do INCA em ambiente de alta performance. Também realizaremos um estudo para verificar a possibilidade de implementação do INCA na borda, de forma a adiantar o SFC.

Referências

2411-0, R. M. Requirements, evaluation criteria and submission templates for the development of imt-2020. 2017. Citado na página 14.

3GPP. *System architecture for the 5G System (5GS)*. [S.l.], 2019. Version 16.3.0. Citado 6 vezes nas páginas 8, 9, 17, 18, 19 e 21.

ABDELSALAM, A. et al. Implementation of virtual network function chaining through segment routing in a linux-based nfv infrastructure. In: *2017 IEEE Conference on Network Softwarization (NetSoft)*. [S.l.: s.n.], 2017. p. 1–5. Citado na página 32.

Alameddine, H. A.; Qu, L.; Assi, C. Scheduling service function chains for ultra-low latency network services. In: *2017 13th International Conference on Network and Service Management (CNSM)*. [S.l.: s.n.], 2017. p. 1–9. ISSN 2165-963X. Citado na página 22.

APS-NETWORKS. 2021. Disponível em: <https://www.aps-networks.com/products/bf6064x-t>. Acesso em: 21 set. 2021. Citado na página 28.

Barefoot (Intel) Torino Chip. 2021. Disponível em: <https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series/tofino.html>. Acesso em: 21 set. 2021. Citado na página 28.

Behavioral Model v2. 2021. Disponível em: <https://github.com/p4lang/behavioral-model>. Acesso em: 21 set. 2021. Citado na página 28.

BHAMARE, D. et al. A survey on service function chaining. *Journal of Network and Computer Applications*, v. 75, p. 138 – 155, 2016. ISSN 1084-8045. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1084804516301989>. Citado na página 22.

BORSATTI, D. et al. Service function chaining leveraging segment routing for 5g network slicing. In: IEEE. *2019 15th International Conference on Network and Service Management (CNSM)*. [S.l.], 2019. p. 1–6. Citado na página 30.

BOSSHART, P. et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, ACM, v. 44, n. 3, p. 87–95, 2014. Citado 5 vezes nas páginas 9, 24, 25, 26 e 27.

DETTI, A. Functional architecture. p. 50–57, 2018. ISSN 00136158. Citado 2 vezes nas páginas 8 e 21.

ETSI. *5G; NG-RAN; PDU session user plane protocol (3GPP TS 38.415 version 16.4.0 Release 16)*. [S.l.], 2021. Version 16.4.0. Citado na página 35.

Fan, J. et al. Availability-aware mapping of service function chains. In: *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*. [S.l.: s.n.], 2017. p. 1–9. ISSN null. Citado na página 22.

FILSFILS, C. et al. *Segment Routing Architecture*. RFC Editor, 2018. RFC 8402. (Request for Comments, 8402). Disponível em: <https://rfc-editor.org/rfc/rfc8402.txt>. Citado 2 vezes nas páginas 15 e 22.

HALPERN, J.; PIGNATARO, C. et al. Service function chaining (sfc) architecture. In: *RFC 7665*. [S.l.: s.n.], 2015. Citado na página 14.

Han, B. et al. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, v. 53, n. 2, p. 90–97, Feb 2015. ISSN 1558-1896. Citado na página 22.

HOMMA, S.; FOY, X. de; GALIS, A. *Gateway Function for Network Slicing*. [S.l.], 2018. Work in Progress. Disponível em: <https://datatracker.ietf.org/doc/html/draft-homma-nfvrg-slice-gateway-00>. Citado na página 15.

Iwamura, M. Ngmn view on 5g architecture. In: *2015 IEEE 81st Vehicular Technology Conference (VTC Spring)*. [S.l.: s.n.], 2015. p. 1–5. ISSN 1550-2252. Citado na página 17.

Jiang, M.; Condoluci, M.; Mahmoodi, T. Network slicing management prioritization in 5g mobile systems. In: *European Wireless 2016; 22th European Wireless Conference*. [S.l.: s.n.], 2016. p. 1–6. ISSN null. Citado na página 19.

JOSHI, K.; BENSON, T. Network function virtualization. *IEEE Internet Computing*, IEEE, v. 20, n. 6, p. 7–9, 2016. Citado na página 14.

KAUR, K.; SINGH, J.; GHUMMAN, N. Mininet as software defined networking testing platform. In: . [S.l.: s.n.], 2014. Citado na página 28.

Lee, C. et al. Performance evaluation of gtp-u and srv6 stateless translation. In: *2019 15th International Conference on Network and Service Management (CNSM)*. [S.l.: s.n.], 2019. p. 1–6. ISSN 2165-9605. Citado na página 29.

MATSUSHIMA, S. et al. *Segment Routing IPv6 for Mobile User Plane*. [S.l.], 2019. Work in Progress. Disponível em: <https://datatracker.ietf.org/doc/html/draft-ietf-dmm-srv6-mobile-uplane-07>. Citado na página 23.

MCKEOWN, N. et al. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, Association for Computing Machinery, New York, NY, USA, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833. Disponível em: <https://doi.org/10.1145/1355734.1355746>. Citado na página 25.

Open-NFP. 2021. Disponível em: <https://cdn.open-nfp.org/media/documents/PB_Agilio_CX_2x10GbE.pdf>. Acesso em: 21 set. 2021. Citado 2 vezes nas páginas 8 e 37.

ORDONEZ-LUCENA, J. et al. The creation phase in network slicing: From a service order to an operative network slice. In: IEEE. *2018 European Conference on Networks and Communications (EuCNC)*. [S.l.], 2018. p. 1–36. Citado 2 vezes nas páginas 8 e 20.

ORDONEZ-LUCENA, J. et al. *The Creation Phase in Network Slicing: From a Service Order to an Operative Network Slice*. 2018. Disponível em: <http://arxiv.org/abs/1804.09642https://arxiv.org/abs/1804.09642>. Citado na página 14.

ORDONEZ-LUCENA, J. et al. Network slicing for 5g with sdn/nfv: Concepts, architectures, and challenges. *IEEE Communications Magazine*, v. 55, n. 5, p. 80–87, 2017. Citado na página 19.

PAOLUCCI, F. et al. P4 edge node enabling stateful traffic engineering and cyber security. *Journal of Optical Communications and Networking*, Optical Society of America, v. 11, n. 1, p. A84–A95, 2019. Citado 2 vezes nas páginas 8 e 28.

RICART-SANCHEZ, R. et al. P4-netfpga-based network slicing solution for 5g mec architectures. In: IEEE. *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. [S.l.], 2019. p. 1–2. Citado na página 30.

ROST, P. et al. Network Slicing to Enable Scalability and Flexibility in 5G Mobile Networks. *IEEE Communications Magazine*, Institute of Electrical and Electronics Engineers Inc., v. 55, n. 5, p. 72–79, may 2017. ISSN 01636804. Disponível em: <https://ieeexplore.ieee.org/document/7926920>. Citado na página 14.

SHAH, R. et al. Turboepc: Leveraging dataplane programmability to accelerate the mobile packet core. In: *Proceedings of the Symposium on SDN Research*. [S.l.: s.n.], 2020. p. 83–95. Citado na página 29.

SHEN, C.-A. et al. A programmable and fpga-accelerated gtp offloading engine for mobile edge computing in 5g networks. In: IEEE. *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. [S.l.], 2019. p. 1021–1022. Citado na página 30.

Taleb, T.; Ksentini, A.; Jantti, R. "anything as a service"for 5g mobile systems. *IEEE Network*, v. 30, n. 6, p. 84–91, November 2016. ISSN 1558-156X. Citado na página 22.

The P4 Language Consortium. The P4 Language Specification. *IEEE Transactions on Mobile Computing*, p. 1–90, 2015. Citado 2 vezes nas páginas 8 e 27.

TIRUMALA, A. Iperf: The tcp/udp bandwidth measurement tool. *http://dast. nlanr. net/Projects/Iperf/*, 1999. Citado na página 38.

Wistron NeWeb Corp. 2021. Disponível em: <https://www.wnc.com.tw/index.php?action=pro_detail&top_id=106&scid=142&tid=99&id=353>. Acesso em: 21 set. 2021. Citado na página 28.

XIA, W. et al. A survey on software-defined networking. *IEEE Communications Surveys & Tutorials*, IEEE, v. 17, n. 1, p. 27–51, 2014. Citado na página 14.

APÊNDICE A – Código fonte

A.1 Declaração de cabeçalhos

```
typedef bit<16> egressSpec_t;
typedef bit<48> macAddr_t;
typedef bit<128> ip6Addr_t;
```

Code A.1 – Declaração de tipos de dados

```
header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16> etherType;
}
```

Code A.2 – Header *Ethernet*

```
header ipv6_t {
    bit<4> version;
    bit<8> traffic_class;
    bit<20> flow_label;
    bit<16> payload_len;
    bit<8> next_hdr;
    bit<8> hop_limit;
    bit<128> src_addr;
    bit<128> dst_addr;
}
```

Code A.3 – Header IPv6

```
header srv6_t3 {
    bit<8> next_hdr;
    bit<8> hdr_ext_len;
    bit<8> routing_type;
    bit<8> segment_left;
    bit<8> last_entry;
    bit<8> flags;
    bit<16> tag;
    bit<128> segment_id1;
    bit<128> segment_id2;
    bit<128> segment_id3;
}
```

Code A.4 – Header SRv3

```
header udp_t {
    bit<16> sport;
    bit<16> dport;
    bit<16> len;
    bit<16> checksum;
}
```

Code A.5 – Header UDP

```
header tcp_t {
    bit<16> srcPort;
    bit<16> dstPort;
    bit<32> seqNo;
    bit<32> ackNo;
    bit<4>  dataOffset;
    bit<3>  res;
    bit<3>  ecn;
    bit<6>  ctrl;
    bit<16> window;
    bit<16> checksum;
    bit<16> urgentPtr;
}
```

Code A.6 – Header TCP

```
header gtp_t {
    bit<3>  version_field_id;
    bit<1>  proto_type_id;
    bit<1>  spare;
    bit<1>  extension_header_flag_id;
    bit<1>  sequence_number_flag_id;
    bit<1>  npdu_number_flag_id;
    bit<8>  msgtype;
    bit<16> msglen;
    bit<32> teid;
}
```

Code A.7 – Header GTP

```
header gtp_ext_t {
  bit<8> next_extension;
}
```

Code A.8 – Cabeçalho de tipo de extensão GTP

```
header pdu_container_t {
  bit<4> pdu_type;
  bit<5> spare;
```

```
    bit<1> rqi;
    bit<6> qosid;
    bit<8> padding;
}
```

Code A.9 – Cabeçalho *PDU Session User Plane Protocol*

```
struct headers {
    ethernet_t              ethernet;
    ipv6_t                  ipv6_outer;
    srv6_t3                 srv63;
    udp_t                   udp;
    tcp_t                   tcp;
    gtp_t                   gtp;
    gtp_ext_t               gtp_ext;
    pdu_container_t         pdu_container;
    ipv6_t                  ipv6_inner;
    udp_t                   udp_inner;
    tcp_t                   tcp_inner;
}
```

Code A.10 – Estrutura de cabeçalhos

A.2 *Parser*

```
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {
```

Code A.11 – Declaração *parser*

```
    state start {
        transition parse_ethernet;
    }
```

Code A.12 – *start*

```
    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            TYPE_IPV6: parse_ipv6_outer;
        }
    }
```

Code A.13 – *Parse ethernet*

```
    state parse_ipv6_outer {
        packet.extract(hdr.ipv6_outer);
        transition select(hdr.ipv6_outer.next_hdr){
            TYPE_UDP: parse_udp_outer;
            TYPE_TCP: parse_tcp_outer;
            TYPE_SRV6: parse_srv63;
            default: accept;
        }
    }
```

Code A.14 – *Parse IPv6 outer*

```
    state parse_srv63 {
        packet.extract(hdr.srv63);
        transition parse_udp_outer;
    }
```

Code A.15 – *Parse* SRv3

```
    state parse_udp_outer {
        packet.extract(hdr.udp);
        transition select (hdr.udp.dport){
            TYPE_GTP: parse_gtp;
            default: accept;
        }
    }
```

Code A.16 – *Parse* UDP *outer*

```
    state parse_tcp_outer {
        packet.extract(hdr.tcp);
        transition accept;
    }
```

Code A.17 – *Parse* TCP *outer*

```
    state parse_gtp {
        packet.extract(hdr.gtp);
        transition select(hdr.gtp.extension_header_flag_id){
            1: parse_gtp_ext;
            0: parse_ipv6_inner;
        }
    }
```

Code A.18 – *Parse* GTP

```
    state parse_gtp_ext{
        packet.extract(hdr.gtp_ext);
```

```
3           transition select(hdr.gtp_ext.next_extension){
4               pdu_container: parse_pdu_container;
5               }
6           }
```

Code A.19 – *Parse* extensão GTP

```
1       state parse_pdu_container{
2           packet.extract(hdr.pdu_container);
3           transition parse_ipv6_inner;
4       }
```

Code A.20 – *Parse PDU Session User Plane Protocol*

```
1       state parse_ipv6_inner{
2           packet.extract(hdr.ipv6_inner);
3           transition select(hdr.ipv6_inner.next_hdr){
4               TYPE_UDP: parse_udp_inner;
5               TYPE_TCP: parse_tcp_inner;
6               default: accept;
7           }
8       }
```

Code A.21 – *Parse* IPv6 *inner*

```
1       state parse_udp_inner{
2           packet.extract(hdr.udp_inner);
3           transition accept;
4       }
```

Code A.22 – *Parse* UDP *inner*

```
1       state parse_tcp_inner{
2           packet.extract(hdr.tcp_inner);
3           transition accept;
4       }
5   }
```

Code A.23 – *Parse* TCP *inner*

A.3 *Ingress processing*

```
1   control MyIngress (inout headers hdr,
2                      inout metadata meta,
3                      inout standard_metadata_t standard_metadata) {
```

Code A.24 – Declaração *ingress processing function*

```
1     action drop() {
2         mark_to_drop();
3     }
```

Code A.25 – *Drop*

```
1     action ipv6_forward (macAddr_t dstAddr, egressSpec_t port) {
2         standard_metadata.egress_spec = port;
3         hdr.ethernet.dstAddr = dstAddr;
4     }
```

Code A.26 – IPv6 *forward*

```
1     action srv6_pop (bit<8> hop) {
2         hdr.ipv6_outer.next_hdr = hdr.srv63.next_hdr;
3         hdr.ipv6_outer.payload_len = hdr.ipv6_outer.payload_len - 56;
4         hdr.srv63.setInvalid();
5         hdr.ipv6_outer.hop_limit = hop;
6         hdr.gtp.spare = 1;
7     }
```

Code A.27 – SRv3 *pop*

```
1     action build_srv63(ip6Addr_t s2, ip6Addr_t s3) {
2         hdr.srv63.setValid();
3         hdr.srv63.next_hdr = hdr.ipv6_outer.next_hdr;
4         hdr.srv63.hdr_ext_len =  LEN;
5         hdr.srv63.routing_type = TYPE_SR;
6         hdr.srv63.segment_left = SL;
7         hdr.srv63.last_entry = 2;
8         hdr.srv63.flags = 0;
9         hdr.srv63.tag = 1;
10        hdr.srv63.segment_id1 = hdr.ipv6_outer.dst_addr;
11        hdr.srv63.segment_id2 = s2;
12        hdr.srv63.segment_id3 = s3;
13        hdr.ipv6_outer.next_hdr = TYPE_SRV6;
14        hdr.ipv6_outer.dst_addr = s3;
15        hdr.ipv6_outer.payload_len = hdr.ipv6_outer.payload_len + 56;
16    }
```

Code A.28 – *build* SRv3

```
1     table ipv6_outer_lpm {
2         key = {
3             hdr.ipv6_outer.dst_addr:exact;
4         }
5         actions = {
6             ipv6_forward;
```

```
            drop;
            NoAction;
        }
        size = 1024;
        default_action = drop();
    }
```

Code A.29 – Tabela de roteamento IPv6

```
    table main {
        key = {
            hdr.gtp.teid: ternary;
            hdr.pdu_container.qosid: ternary;
            hdr.ipv6_inner.dst_addr: ternary;
            hdr.ipv6_inner.src_addr: ternary;
            hdr.ipv6_inner.next_hdr: ternary;
            hdr.tcp_inner.dstPort: ternary;
            hdr.tcp_inner.srcPort: ternary;
            hdr.udp_inner.dport: ternary;
            hdr.udp_inner.sport: ternary;
        }
        actions = {
            build_srv63;
            srv6_pop;
        }
        default_action = srv6_pop(64);
    }
```

Code A.30 – Tabela de *matching* para construção do SRv3

```
    table pop {
        key = {
            hdr.ipv6_outer.dst_addr:exact;
        }
        actions = {
            srv6_pop;
            drop;
        }
        size = 1024;
        default_action = drop();
    }
```

Code A.31 – Tabela de *matching* para *drop* do SRv3

```
    apply {
        if (hdr.srv63.isValid() && hdr.srv63.segment_left == 2 && hdr.srv63
            .tag == 0){
            main.apply();
        } else if (hdr.srv63.isValid() && hdr.srv63.segment_left == 0){
```

```
5                pop.apply();
6            }
7            ipv6_outer_lpm.apply();
8        }
9    }
```

Code A.32 – Lógica condicional

A.4 *Deparser*

```
1  control MyDeparser (packet_out packet,
2                      in headers hdr) {
3      apply {
4          packet.emit(hdr.ethernet);
5          packet.emit(hdr.ipv6_outer);
6          packet.emit(hdr.srv63);
7          packet.emit(hdr.udp);
8          packet.emit(hdr.tcp);
9          packet.emit(hdr.gtp);
10         packet.emit(hdr.gtp_ext);
11         packet.emit(hdr.pdu_container);
12         packet.emit(hdr.ipv6_inner);
13         packet.emit(hdr.udp_inner);
14         packet.emit(hdr.tcp_inner);
15     }
16 }
```

Code A.33 – *Deparser*