Telemetria Adaptativa Usando Aprendizado por Reforço Profundo em Redes Definidas por Software

Debora H. Job[1], **Sidney C. de Lucena** [1], **Pedro Nuno Moura** [1]

[1]Programa de Pós–Graduação em Informática
Universidade Federal do Estado do Rio de Janeiro (UNIRIO)
CEP 22290–255-– Rio de Janeiro, RJ -– Brasil

{debora.job,sidney,pedro.moura}@uniriotec.br

Abstract. *The use of software-defined networks leveraged the adoption of telemetry techniques for fine-grained monitoring. However, its indiscriminate adoption generates additional costs. It may degrade network performance, increasing the amount of the stored data to be processed, making its benefits unfeasible. The adoption of adaptive telemetry then emerges as a way to circumvent this problem. This work proposes a deep reinforcement learning engine to provide a data plane adaptive telemetry that monitors congestion. As a proof-of-concept, we developed an environment on a version of the ONOS platform enabled with the In-Band Network Telemetry (INT) engine and P4 switches. Experiments with varying traffic profiles and hyperparameters confirm the proposal's benefits and explore its limitations.*

Resumo. *O uso de redes definidas por software alavancou a adoção de técnicas de telemetria para um monitoramento de alta granularidade. Sua adoção indiscriminada, contudo, gera custos adicionais que podem degradar o desempenho da rede, provocar um volume exagerado de dados a serem armazenados e processados e, assim, inviabilizar seus benefícios. A adoção de telemetria adaptativa surge então como uma forma de contornar esse problema. Este trabalho propõe o uso de aprendizado por reforço profundo para prover uma telemetria adaptativa do plano de dados que monitore congestionamentos. Um ambiente para prova de conceito foi desenvolvido sobre uma versão da plataforma ONOS habilitada com o mecanismo In-Band Network Telemetry (INT) e switches P4. Experimentos variando perfis de tráfego e hiperparâmetros do mecanismo de aprendizado confirmam os benefícios da proposta e exploram suas limitações.*

1. Introdução

Na última década, a programabilidade do plano de controle das redes de computadores, alavancado pelas redes definidas por software (SDN), estendeu-se também para o plano de dados graças ao surgimento da linguagem P4 [Bosshart et al. 2014] e de outras técnicas como o BPFabric, que faz uso de eBPF [Jouet and Pezaros 2017]. Tal avanço ampliou ainda mais as possibilidades de automação dos diversos processos de gerenciamento das redes, dentre eles aqueles voltados para um monitoramento mais especializado, capaz de verificar problemas de desempenho mais pontuais e em escalas de tempo menor. Esse tipo de monitoramento especializado tem sido chamado de *telemetria* pelo mercado e pela academia.

Em linhas gerais, a telemetria de redes pode ser descrita como um processo no qual medidas são coletadas em curtos intervalos de tempo para um dispositivo de rede, e enviadas em tempo real para um receptor. Tal envio pode se dar através de técnicas que façam uso da própria rede de dados (*in-band telemetry*), que é a forma que mais atrai o interesse da comunidade [Yu 2019]. Em SDN, a telemetria *in-band* pode ser obtida pela programação do plano de dados dos switches, como no *In-band Network Telemetry* (INT) [Van Tu et al. 2017], que faz uso da linguagem P4 para medir ocupação nas filas de saída, latência salto a salto e/ou latência de fluxo, dentre outras medidas. Tais medidas estão vinculadas à presença de fluxos de dados pré-configurados no controlador SDN.

A instrumentação de uma SDN objetivando telemetria *in-band* pode gerar um elevado fluxo de informações com alto grau de especificidade, o que pode elevar substancialmente o custo de processamento, armazenamento e uso de banda. Por outro lado, essa riqueza de informação pode ser usada por mecanismos de aprendizado de máquina para agregar inteligência a essas redes. No aprendizado por reforço profundo [Graesser and Keng 2020], por exemplo, sistemas são treinados a partir de padrões obtidos por algoritmos de inferência baseados em modelos matemáticos [Xie et al. 2019]. O resultado desta união entre programabilidade de redes com telemetria *in-band* e mecanismos de inteligência artificial pode ser inserido no escopo das chamadas Redes Definidas pelo Conhecimento, ou *Knowledge-Defined Networking* (KDN) [Mestres et al. 2017].

Este trabalho propõe o uso de técnicas de aprendizado por reforço profundo para criar um agente inteligente capaz de reconfigurar, conforme a necessidade, um mecanismo de telemetria *in-band* em uma SDN. Com isso, objetiva-se um melhor equilíbrio entre o benefício de uma telemetria *in-band* gerando mais ou menos informações e o consumo de recursos decorrente disso. Como prova de conceito, construiu-se um agente capaz de perceber padrões que indiquem, ou não, saturação na fila de saída de um *switch* e, conforme o caso, aumente ou diminua a quantidade de metadados do *framework* INT [Tu et al. 2018]. Esse *framework* inclui o controlador ONOS, *software switches* bmv2 programados em P4, o emulador de redes *mininet* e o *INTCollector*, que recebe e armazena os dados de telemetria numa base InfluxDB que serve de fonte para o aprendizado por reforço. Esses dados de telemetria são unicamente associados a um fluxo *monitor*, cuja rota passa pelos dispositivos que se deseja monitorar e cuja vazão confere o grau de amostragem desejado. Ao longo dos experimentos, o tráfego do fluxo monitor e as recompensas do modelo foram empiricamente ajustadas, assim como se variou o tráfego de *background* da rede e os hiperparâmetros do modelo de aprendizado. Os resultados obtidos mostram que o agente inteligente é capaz de perceber situações de congestionamento dentro de uma escala de tempo desejável e, assim, adaptar em tempo real o nível de telemetria da rede.

O restante deste artigo traz os trabalhos relacionados na Seção 2; a Seção 3 aborda SDN e telemetria *in-band*; a Seção 4 fundamenta aprendizado por reforço profundo; a Seção 5 apresenta o cenário experimental e as implementações adotadas; a Seção 6 discute os resultados obtidos; por fim, a Seção 7 traz a conclusão e trabalhos futuros.

2. Trabalhos Relacionados

O *Self-tuning Adaptive Monitoring* (SAM) é proposto em [Tangari et al. 2018] como uma abordagem descentralizada e adaptativa para o monitoramento de redes SDN, através da qual se busca limitar o consumo de recursos desta ação. O sistema se auto-ajusta conforme

as características do tráfego, modificando em tempo real a taxa de leitura dos *switches* da rede e a agregação de medidas, porém mantendo um adequado compromisso entre precisão e consumo de banda. Esse ajuste é dado por um mecanismo de escalonamento de leituras que controla a carga demandada para cada *switch* numa dada janela de tempo, conforme estimativas de consumo de banda causado por essas leituras a partir do número de registros a serem lidos. Esse mecanismo adota também limites inferiores e superiores para cada taxa de leitura, associados a uma técnica de predição baseada na estatística das últimas n leituras de uma dada medição.

O conceito de KDN foi proposto por [Mestres et al. 2017] e combina técnicas de SDN e de aprendizado de máquina para fins de controle automatizado da rede. Nessa mesma abordagem, [Yao et al. 2018] propõe o NetworkAI, uma arquitetura inteligente que emprega aprendizado por reforço profundo e tecnologias de monitoramento de rede para gerar políticas de controle em redes SDN. Nele, um agente inteligente centralizado interage com a rede num sistema em malha fechada, no qual o estado da rede é enviado para o agente por um *upload link* e as decisões de controle são enviadas para a rede por um *download link*.

A proposta de [Hyun et al. 2018] também se baseia nas premissas de KDN. Nela, os autores apresentam um sistema de monitoramento de granularidade fina, baseado no sistema proposto em [Van Tu et al. 2017], no qual os metadados de telemetria gerados por cada pacote são extraídos e transmitidos ao plano de gerenciamento. Essa extração é coordenada por um controlador SDN que atua sobre dispositivos programáveis, influenciando assim o monitoramento da rede. No plano do conhecimento, os dados coletados servem de entrada para algoritmos de aprendizado de máquina que fornecem como saída o modelo da rede. Essas saídas são então convertidas nos chamados *intents*, que são parâmetros de alto nível usados para alterar a configuração da rede.

Este trabalho se diferencia das propostas de [Yao et al. 2018] e de [Hyun et al. 2018], que usam inteligência artificial para alterar o comportamento da rede. Em nossa proposta, objetivamos o autoajuste do próprio monitoramento, assim como em [Tangari et al. 2018], porém através de um mecanismo de aprendizado por reforço profundo atuando sobre uma solução de telemetria *in-band*. Dado que uma telemetria exaustiva pode ter forte impacto no funcionamento da rede, o mecanismo de aprendizado proposto deve ser capaz de predizer a ocorrência ou não de eventos de interesse, como rajadas de tráfego em escalas de tempo pequenas, e assim aumentar ou reduzir a telemetria para permitir o monitoramento desses eventos de forma econômica para a rede.

3. Telemetria do Plano de Dados com INT

A telemetria é uma especialização do monitoramento que oportuniza obter dados de pacotes e de fluxos com granularidade fina de tempo, possibilitando uma visão mais acurada do comportamento da rede. Com o advento da progamabilidade do plano de dados, tornou-se possível realizar essa telemetria "por dentro da banda", ou seja, tendo suas medições encapsuladas e transmitidas dentro dos pacotes de dados da rede. O *framework In-band Network Telemetry* (INT) foi projetado para viabilizar essa abordagem, gerando relatórios que sumarizam essas medições a não requererem intervenção do plano de controle. O INT permite um monitoramento com acurácia e granularidade fina, aplicado a redes que ope-

ram dispositivos programados em linguagem P4.

A estratégia do INT consiste em acrescentar metadados aos cabeçalhos dos pacotes de um ou mais fluxos previamente associados ao disparo da telemetria. Esses metadados acumulam, ao longo do caminho desses fluxos, as medidas coletadas por cada dispositivo que tenha o INT configurado. Quando os fluxos passam pelo dispositivo configurado como "ponto de egresso", os metadados são retirados dos cabeçalhos dos pacotes desses fluxos, restaurando assim suas estruturas originais, e exportados para um servidor "coletor". Nesse servidor, essas informações de telemetria são sumarizadas e organizadas numa base de dados [Tu et al. 2018].

Conforme especificado na padronização do INT [Van Tu et al. 2017], os *switches* responsáveis por manipular os metadados nos cabeçalhos dos pacotes são denominados de *INT Source*, *INT Transit Hop* ou *INT Sink*, conforme sua função. O *INT Source* realiza a inserção de metadados nos cabeçalhos, conforme as instruções pertinentes, e os *INT Transit Hops* os atualizam. Já o *INT Sink* se responsabiliza pela remoção desses metadados e envio de seus valores para o coletor. Há um conjunto de metadados padrão definido na especificação do INT que são elencados a seguir:

- Switch ID: informação pertinente ao *switch*, que deve ser única dentro de um domínio de gerenciamento;
- Ingress Port ID: Referente à porta na qual o pacote INT é recebido;
- Ingress Timestamp: indica o tempo local do dispositivo quando o pacote INT foi recebido na porta de entrada;
- Egress Port ID: referente à porta que o pacote INT saiu;
- Egress Timestamp: indica o tempo local do dispositivo quando o pacote INT deixou porta de saída;
- Hop Latency: indica o tempo levado para o pacote INT ser comutado dentro do dispositivo;
- Egress port TX Link Utilization: informa o percentual de utilização da porta de saída no momento em que o pacote INT é comutado para ela;
- Queue occupancy: informa a ocupação da fila na porta de saída do dispositivo no momento em que o pacote INT é comutado para ela.

O coletor INT pode processar os metadados de telemetria e inserir métricas consolidadas por fluxo e/ou por tempo na base de dados. É o caso do pacote *open source* BPFCollector[1], usado neste trabalho. O BPFCollector gera diferentes métricas, a partir dos metadados de telemetria, e as armazena numa base Influxdb[2].

4. Aprendizado por Reforço Profundo

O aprendizado de máquina é uma subárea da Inteligência Artificial que lida com algoritmos que aprendem a partir de experiências relacionadas a uma determinada tarefa, melhorando seu desempenho nesta tarefa segundo uma métrica pré-definida. As subdivisões clássicas nesse campo são: aprendizado supervisionado, geralmente usado para classificação e regressão, onde há um atributo de rótulo a ser predito; aprendizado não supervisionado, usado para redução de dimensionalidade, clusterização e estimativa de

[1]https://gitlab.com/ebpf/BPFCollector
[2]https://www.influxdata.com/

densidade; e aprendizado por reforço, empregado em problemas de controle em *closed-loop*, pois suas ações de aprendizagem afetam as entradas posteriores [Yao et al. 2018].

O aprendizado por reforço exige que um agente aprenda como atuar, mapeando situações em ações e sem conhecimento prévio. Diferente do aprendizado supervisionado, onde se tem a ação correta para cada exemplo fornecido, no aprendizado por reforço isso se dá por tentativa e erro. O agente interage com o ambiente executando ações durante um certo tempo, objetivando descobrir quais ações levam às maiores recompensas e, assim, aprender uma política de comportamento ótima [Sutton and Barto 2018]. Essa política visa, portanto, maximizar as recompensas acumuladas ao longo do tempo.

Como explica [Sutton and Barto 2018], o agente e o ambiente interagem em uma sequência de etapas de tempo discreto t. A cada etapa, o agente recebe uma representação do ambiente denominada **estado** $S_t \in S$, onde S é um conjunto de estados possíveis, e seleciona uma **ação** $A_t \in A(S_t)$, onde $A(S_t)$ é um conjunto de ações disponíveis em S_t. Após a ação ser tomada, o agente receberá uma **recompensa imediata** $R_{t+1} \in R$, onde R é o conjunto de possíveis recompensas, indo para um novo estado S_{t+1}. O objetivo é aprender a política de controle $\pi : S \rightarrow A$ que maximiza a soma esperada das recompensas.

O agente inicia o treinamento com a estratégia de *exploration*, que o permite vivenciar o impacto de suas ações através das recompensas retornadas pelo ambiente [Krohn and Beyleveld 2020]. Para tanto, com uma probabilidade definida pelo hiperparâmetro *epsilon*, denominado de taxa de exploração, o agente toma uma ação aleatória a cada estado recebido. Já na estratégia de *exploitation*, o agente usa o conhecimento já adquirido por meio do aprendizado para tomar ações que retornem as maiores recompensas. A cada etapa de tempo, *epsilon* é reduzido ao ser multiplicado pelo hiperparâmetro *epsilon_decay*. Quando o valor de *epsilon* atinge o limiar dado pelo hiperparâmetro *epsilon_min*, o agente fica essencialmente em *exploitation*, de forma estável.

Formalmente, o aprendizado por reforço pode ser descrito como um *Markov Decision Process* (MDP) [Arulkumaran et al. 2017] contendo uma **função de transição de estados** P e uma **função da recompensa imediata esperada** R relacionadas à tripla (s, a, s'), onde s é o estado atual, a a ação a ser executada e s' o próximo estado. A função P fornece a probabilidade de um agente no estado s, ao executar a ação a, atingir o estado s'. A função R retorna a recompensa esperada nesse passo.

Nesse MDP(S, A, P, R), o agente executa uma política $\pi(s, a)$, $s \in S$ e $a \in A$, que busca maximizar, a longo prazo, o valor esperado da soma das recompensas futuras e, com isso, alcançar uma política de ação ótima. Esse cálculo considera ainda um desconto exponencial de recompensas futuras pelo seu atraso. A *função valor-estado* $V^{\pi}(s_t)$ (Equação 1) representa a quantidade de recompensas acumuladas para uma política arbitrária π a partir de um estado s em diferentes trajetórias [Graesser and Keng 2020]:

$$V^{\pi}(s) = \mathbb{E}_{\pi, s_0=s}\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots + \gamma^T r_T\right] = \mathbb{E}_{\pi, s_0=s}\left[\sum_{t=0}^{T} \gamma^t r_t\right] \qquad (1)$$

onde $\gamma \in [0, 1]$ serve para penalizar as recompensas atrasadas em relação às imediatas.

Semelhante à função V^{π}(s), a *função valor-ação* $Q^{\pi}(s, a)$ descreve a quantidade esperada de recompensas acumuladas a partir do instante em que se toma a ação a no estado s para uma política π, em diferentes trajetórias [Sutton and Barto 2018]. Isso permite obter a função de valor ótimo $Q^{*}(s, a)$ para todas as políticas possíveis, o que determina a política que informará ao agente qual ação tomar em uma dada circunstância [Sutton and Barto 2018].

O algoritmo *Q-learning* computa $Q(s, a)$ iterativamente, armazenando em uma tabela indexada pelos valores das tuplas estado-ação as suas respectivas recompensas durante o aprendizado. O cálculo explícito de $Q(s, a)$ pode ser inviável se houver um número expressivo de pares estado–ação, tornando-se ineficiente para problemas de alta complexidade computacional de memória, além de implicar em baixas taxas de convergência para obtenção do comportamento ótimo [Xie et al. 2019]. Como alternativa, adota-se uma rede neural profunda como método de aproximação da função valor-ação $Q(s, a)$ [Arulkumaran et al. 2017, Yao et al. 2018].

O trabalho de [Mnih et al. 2015] deu origem ao chamado *Aprendizado por Reforço Profundo* ao usar uma rede neural convolucional capaz de aprender uma política de controle para lidar com entradas de alta dimensão. A rede usava uma variação do algoritmo *Q-Learning* para treinamento do agente e foi testada em jogos da plataforma de videogame Atari, obtendo resultados melhores do que todos os algoritmos que existiam previamente, além de desempenho sobre-humano para diversos jogos testados. Desde então, todos os grandes avanços no campo de aprendizado por reforço têm utilizado redes neurais para aproximar a função $Q(s, a)$ [Graesser and Keng 2020].

Resolver problemas usando o aprendizado por reforço requer modelar, conforme o domínio de aplicação, estados, ações e recompensas. Inicialmente, modela-se o ambiente para definir quais informações são disponibilizadas para o agente em cada etapa de tempo, constituindo seu espaço de estados. Em seguida, planejam-se as ações possíveis que o agente pode executar. Por fim, define-se a estratégia de recompensa que será adotada.

5. Proposta de Solução

O *framework* proposto é detalhado na Figura 1, onde se pode visualizar o agente interagindo com um componente denominado de **SmartMon**. O SmartMon foi criado para manipular os metadados de telemetria, criar os estados que representam o ambiente e orquestrar as ações sobre ele. Portanto, o SmartMon providencia a interação do agente com o ambiente, consumindo os dados de telemetria reportados pelo plano de dados e adaptando o nível dessa telemetria para reduzir seus custos.

A seguir são descritas as implementações dos elementos do *framework*, aqui denominado ***framework SmartMon***[3], objetivando a criação de um protótipo experimental.

O **ambiente** em questão é constituído por uma rede SDN com o INT habilitado. Ele é emulado usando o Mininet[4] e um controlador ONOS, com o plano de dados formado por ao menos dois *switches* bmv2[5], sendo um no papel de *source* e o outro no de *sink*. O coletor INT, implementado pelo BPFCollector, gera métricas que são armazenadas em

[3]https://github.com/unirio/SmartMon

[4]http://mininet.org/

[5]https://github.com/p4lang/behavioral-model

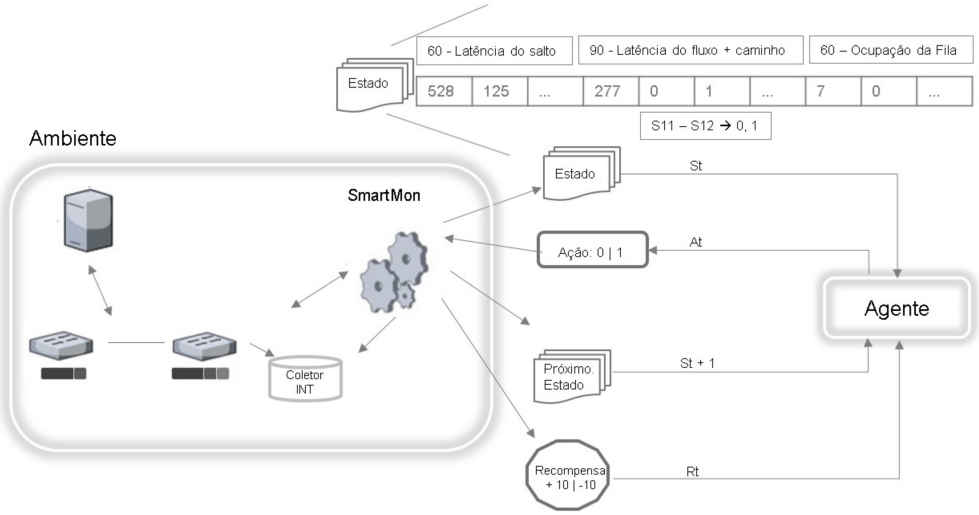

Figura 1. O *framework* SmartMon.

tabelas do Influxdb: latência do salto (*timestamp, latência*), latência do fluxo (*timestamp, latência do fluxo, caminho percorrido*) e profundidade da fila (*switch ID, fila*), sendo que *caminho percorrido* contém os IDs dos *switches* no caminho do fluxo associado à telemetria. O componente *SmartMon* lê essas sequências de tuplas para então criar um **estado** S_t que reflita o ambiente no mecanismo de aprendizado por reforço.

No SmartMon, o **estado** S_t representa um intervalo de tempo fixo de atividade da rede. Para tal, todas as sequências de tuplas armazenadas no Influxdb que refletem as medidas de telemetria nesse período de atividade são lidas e inseridas num vetor de dimensão fixa, usado na representação de S_t na rede neural. O tamanho desse vetor, portanto, deve ser calibrado conforme a quantidade de entradas geradas nas tabelas do Influxdb pelos fluxos que contêm os metadados de telemetria ao longo da janela de tempo que se deseja representar em S_t, para qualquer t. No protótipo implementado neste trabalho, adotou-se uma dimensão de vetor capaz de capturar, aproximadamente, 30 segundos de medidas.

O **agente** do *framework* SmartMon é baseado no código *opensource*[6] de [Krohn and Beyleveld 2020], que implementa o algoritmo *Deep Q-Learning* para cenários de jogos. No presente trabalho, adaptou-se esse código para o agente escolher, com base nas observações do ambiente SDN, a ação a ser tomada sobre a rede: acionar telemetria **alta** (ação 1) ou telemetria **baixa** (ação 0). A telemetria **alta** usa todos os metadados gerados pelo INT e a telemetria **baixa** usa apenas os metadados de *switch id*, *port ids* e *hop latency*.

Portanto, o *framework* SmartMon não se limita à implementação *open-source* do *Deep Q-Learning*, envolve também a coleta estruturada de informações do Influxdb, montagem do estado S_t usado no MDP do Aprendizado por Reforço (AR), toda a modelagem e a lógica do AR profundo, a consequente adaptação da telemetria e a implementação das recompensas.

O sistema de **recompensa** deste trabalho baseia-se na observação do comportamento do ambiente em teste. Quando há enlace sobrecarregado e o agente ativou a **te-**

[6]https://github.com/keon/deep-q-learning/blob/master/dqn.py

lemetria alta para investigar mais detalhadamente o que está ocorrendo no ambiente, o agente é recompensado positivamente (+10). Entretanto, o agente é penalizado (-10) se ativar **telemetria baixa** nesses casos. Quando o ambiente não está sobrecarregado, o agente recebe uma recompensa positiva (+10) se ativou **telemetria baixa** e é penalizado (-10) se ativou **telemetria alta**. A sobrecarga é "percebida", ou não, através da sequência de valores de latência de salto e de latência de fluxo, quando a telemetria está baixa, e também da ocupação da fila, quando a telemetria está alta, conforme limites pré-definidos empiricamente para cada caso.

6. Avaliação da Proposta

Usar aprendizado por reforço para gerar uma telemetria adaptativa da rede não é tarefa trivial, dado que o agente aprende enquanto interage com o ambiente a partir das ações tomadas e recompensas recebidas. Como o objetivo macro é maximizar tais recompensas ajustando o aprendizado, é necessário então construir uma modelagem adequada. Para tanto, desenvolveu-se o cenário experimental descrito a seguir.

6.1. Cenário experimental

A Figura 2 ilustra a rede emulada com Mininet para prova de conceito e avaliações preliminares. Nos experimentos, dois fluxos de tráfego UDP com pacotes de 1500 bytes foram gerados usando o *software* iperf[7]: **Fluxo Monitor INT** e **Tráfego de Fundo**. Os enlaces dessa topologia têm capacidade de 50 Mbps.

O **Tráfego de Fundo** serve para estabelecer situações de folga ou sobrecarga no ambiente para verificarmos como o SmartMon se comporta. Para tal, sua banda variou aleatoriamente a cada 60 segundos entre os seguintes valores: 5 Mbps, 10 Mbps, 20 Mbps, 25 Mbps, 30 Mbps, 35 Mbps, 45 Mbps e 51 Mbps.

O **Fluxo Monitor INT** é aquele para o qual se tem a telemetria habilitada, o que é determinado por intermédio da interface de configuração do ONOS. É a rota desse fluxo e a taxa de envio de seus pacotes que definirá em quais *switches* haverá coleta de telemetria e com qual frequência. Para a rede trivial deste experimento, ambos os *switches* farão a inserção de metadados nos pacotes desse fluxo e o *switch* S2 realizará também a extração desses metadados para envio ao coletor.

A banda do fluxo monitor foi empiricamente dimensionada em 3 Mbps para ocupar uma pequena fração da capacidade entre S1 e S2 e ainda ser capaz de produzir uma telemetria que permita uma leitura efetiva do ambiente. Como se pode observar na Figura 2, ambos os tráfegos compartilham o enlace entre S1 e S2. Portanto, os efeitos causados pelo tráfego de fundo, conforme a banda usada, se refletirão nos metadados de telemetria embutidos nos pacotes do fluxo monitor.

O *workflow* dos experimentos envolve os seguintes passos: (i) o tráfego de fundo fica alternando sua banda a cada 60s ao longo de todo o experimento; (ii) injeta-se 30s de fluxo monitor para produzir as métricas de telemetria que são armazenadas no Influxdb; (iii) o SmartMon lê essas métricas e computa S_t para passá-lo ao **Agente**; (iv) o **Agente** define a telemetria que deve ser usada, com base em seu aprendizado, e a informa ao SmartMon; (v) o SmartMon computa R_t, passa para o **Agente** e aplica a telemetria na

[7]https://iperf.fr/iperf-doc.php

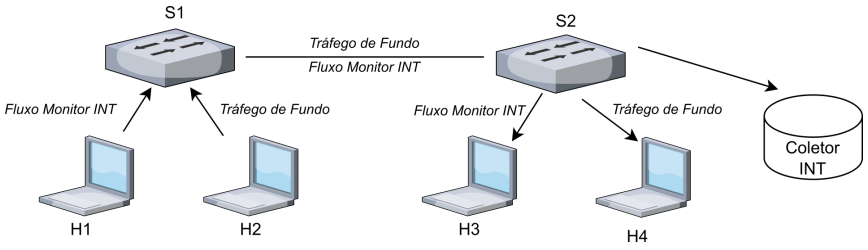

Figura 2. Ambiente utilizado nos experimentos.

rede, e volta-se ao passo (ii). O tempo discreto t compreende o *loop* entre os passos (ii) e (v).

Para avaliarmos o agente do SmartMon, implementou-se um **agente aleatório** para servir de *baseline*. Este agente realiza uma escolha aleatória das ações de aumentar ou diminuir a telemetria vinculada ao fluxo monitor, sem nenhuma atividade de aprendizado associada a isso. Nos experimentos, ambos os agentes foram testados com os mesmos tipos de tráfegos.

6.2. Hiperparâmetros de treinamento e arquitetura da rede neural do agente

No aprendizado por reforço profundo, o treinamento da rede neural do agente se dá por meio da repetição de memórias vivenciadas e da aplicação do algoritmo *Deep Q-Learning*, influenciado pelos valores de um conjunto de hiperparâmetros. O *batch size* representa o número de exemplos utilizados em cada rodada de treinamento, portanto o agente só começa a aprender, de fato, a partir do acúmulo de um número de experiências equivalente ao *batch size*. O hiperparâmetro *episódios* delimita o conjunto de experiências (*timesteps*) usadas no treinamento do modelo, para os quais se aplica o desconto de *gamma* evidenciado na Equação 1. A cada episódio, o valor de *timesteps* indica o número de passos do algoritmo de treinamento durante o aprendizado. Esses três hiperparâmetros, juntamente com *epsilon*, *epsilon_decay*, *epsilon_min* e *gamma*, formam o conjunto de hiperparâmetros usados no SmartMon. Para *epsilon*, *epsilon_min* e *gamma* foram fixados os valores 1, 0,01 e 0,95, respectivamente.

Foram experimentadas diferentes configurações para a rede neural do agente contendo duas camadas ocultas. Aquela que se saiu melhor foi a seguinte: (i) camada de entrada com 210 dados, referentes ao tamanho dos estados do agente; (ii) primeira camada oculta densa com 128 neurônios com ativação ReLU; (iii) segunda camada oculta densa com 64 neurônios com ativação ReLU; e (iv) camada de saída com dois neurônios de ativação linear, referentes às duas possíveis ações do agente (telemetria baixa ou alta). Por fim, a função de custo utilizada foi o Erro Médio Quadrático e o otimizador foi o Adam.

6.3. Métricas

A avaliação experimental se baseou em três medidas: **Probabilidade de Acertar Ligar (PAL)**, **Probabilidade de Acertar Desligar (PAD)** e **Recompensa Acumulada**. No caso, ***ligar*** e ***desligar*** referem-se, respectivamente, a passar a telemetria de ***baixa para alta*** e de ***alta para baixa***. O cálculo de PAL e PAD envolve verificar a fração de vezes que o agente foi recompensado positivamente quando o ambiente necessitava de telemetria alta ou baixa, respectivamente.

A Recompensa Acumulada foi proposta em [Poole and Mackworth 2017] para comparar e avaliar algoritmos baseados em aprendizado por reforço. Essa medida acumula todas as recompensas recebidas pelo agente durante todos os episódios de treinamento e as seguintes métricas são observadas: o ponto de mínimo da curva, relacionado às recompensas negativas que o algoritmo recebe antes de estabilizar o aprendizado; o ponto onde a curva cruza ascendentemente o nível zero, indicando quanto tempo leva até que o algoritmo recupere seu custo de aprendizado; e o crescimento da curva após o algoritmo estabilizar.

6.4. Resultados

Inicialmente serão apresentados os resultados que servem de *baseline* para a avaliação do SmartMon. A Figura 3 apresenta a evolução de PAL e PAD para o agente aleatório em dois experimentos com 20 episódios cada. Cada experimento contou com um número diferente de *timesteps* por episódio (32 e 64). Pode-se verificar que os valores de PAL e PAD variaram bastante ao longo dos episódios, sinalizando um comportamento esperado conforme o caráter aleatório do tráfego de fundo e a proporção de vezes em que havia saturação de enlace. Quanto maior o número de *timesteps*, menores são as variações dessas métricas.

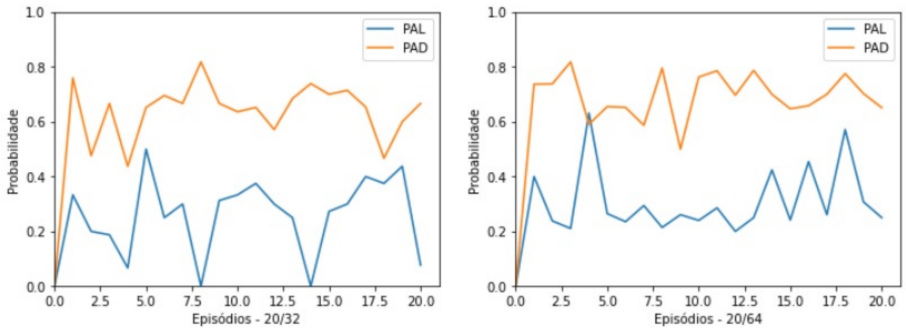

Figura 3. Evolução de PAL e PAD para o Agente Aleatório.

A Figura 4 mostra a evolução da recompensa acumulada ao longo dos episódios para o agente aleatório nos dois experimentos realizados. Como se pode ver, as curvas apresentam um crescimento errático, especialmente quando o número de *timesteps* por episódio é 32. Como os PADs se mostraram superiores a 0,5 em ambos os casos, isso justifica a tendência de crescimento das recompensas acumuladas para o agente aleatório.

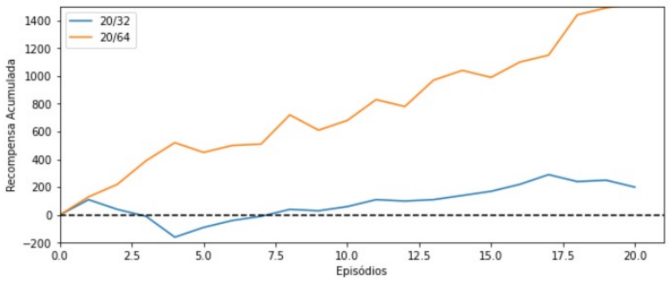

Figura 4. Evolução da recompensa acumulada para o Agente Aleatório.

Para o SmartMon, foram realizados experimentos de 20 episódios cada, variando-se o número de *timesteps* por episódio, o *batch size* e o *epsilon_decay*. A Figura 5 e a Figura 6 mostram a evolução de PAL e PAD ao longo de 20 episódios de 32 *timesteps* e *batch size* de 16 para *epsilon_decay* de 0,97 e 0,985, respectivamente. Cada figura exibe dois experimentos para uma mesma configuração e a linha tracejada vertical indica o momento em que *epsilon* atinge *epsilon_min*, sendo menor quanto menor for o *epsilon_decay*. Observa-se que as evoluções de PAL e PAD são similares a partir desses pontos.

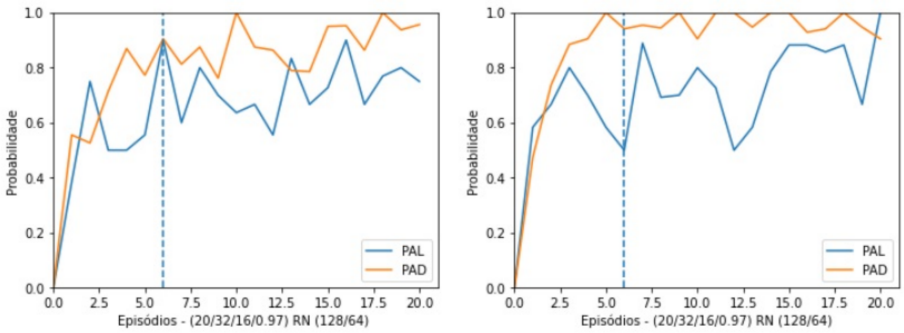

Figura 5. Evolução de PAL e PAD para o SmartMon com *epsilon_decay* de 0,97.

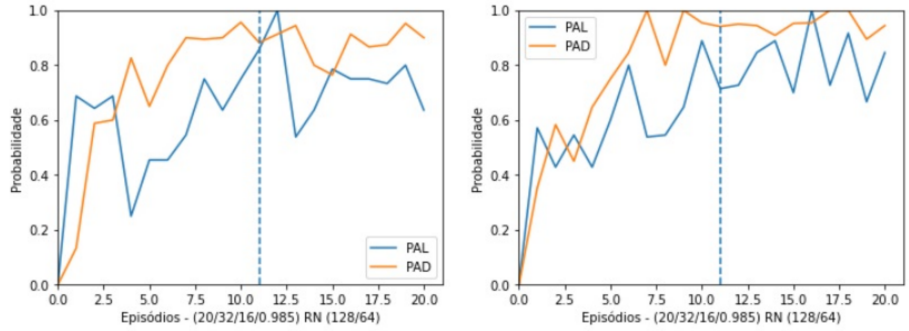

Figura 6. Evolução de PAL e PAD para o SmartMon com *epsilon_decay* de 0,985.

A Figura 7 mostra as evoluções da recompensa acumulada para o SmartMon nos dois experimentos de cada configuração, cada qual associado a uma diferente curva. Cada gráfico dessa figura refere-se a um *epsilon_decay* diferente (0,97 e 0,985). Ao se comparar essas evoluções com as dos experimentos envolvendo o agente aleatório, verifica-se um crescimento muito mais consistente e rápido, com as recompensas acumuladas atingindo níveis muito maiores após 20 episódios.

As Figuras 8 e 9 mostram a evolução de PAL e PAD para 64 *timesteps* por episódio e, respectivamente, *epsilon_decay* de 0,97 e 0,985. Em cada caso, mostra-se o efeito para um *batch size* de 16 e de 32. Observa-se que um *batch size* maior, combinado com um *epsilon_decay* maior, aponta para uma maior consistência na evolução dessas métricas.

A Figura 10 mostra a evolução da recompensa acumulada para o SmartMon nos cenários experimentados. As legendas das curvas trazem os hiperparâmetros usados: número de episódios, quantidade de *timesteps*, *batch size* e *epsilon_decay*. Nos casos onde houve dois experimentos para uma mesma configuração, apresenta-se o melhor resultado

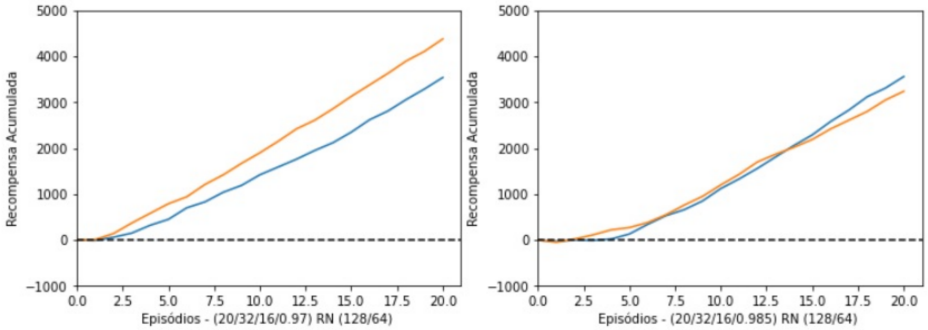

Figura 7. Evolução da recompensa acumulada em 20 episódios para o SmartMon: *epsilon_decay* de 0,97 e 0,985.

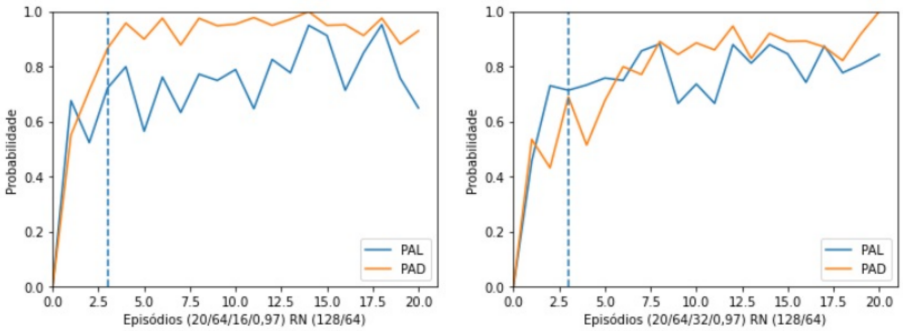

Figura 8. Evolução de PAL e PAD para o SmartMon com 64 *timesteps* e *epsilon_decay* de 0,97: *batch size* de 16 e 32.

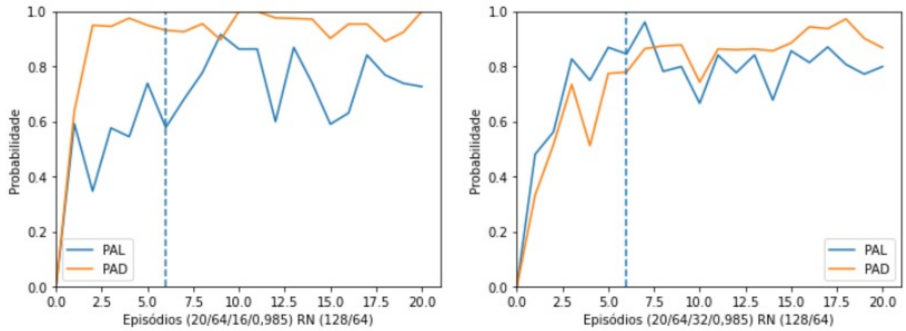

Figura 9. Evolução de PAL e PAD para o SmartMon com 64 *timesteps* e *epsilon_decay* de 0,985: *batch size* de 16 e 32.

para a recompensa acumulada. Pode-se notar que os experimentos com 32 *timesteps* por episódio acumularam menos recompensas do que aqueles que usaram 64 *timesteps*. É possível também observar que o SmartMon teve o seu melhor desempenho quando usou 64 *timesteps*, *batch size* de 16 e *epsilon_decay* de 0,97 ou de 0,985, com uma ligeira vantagem para o último.

Todas as curvas apresentam crescimento assintótico acentuado, demonstrando que o agente aprende uma boa política para escolha das ações a partir do ponto de estabilização. Os pontos de mínimo para as curvas se localizam praticamente na origem, já que as funções possuem um comportamento similar ao de uma função monótona

crescente. Além disso, no experimento no qual a curva apresenta-se sinuosa antes do cruzamento do nível zero, isso é explicado pelo maior tempo que o algoritmo levou explorando o ambiente e acumulando memórias antes de tomar as ações aprendidas, devido às configurações de *timesteps* (64), *batch size* (32) e *epsilon_decay* (0,985).

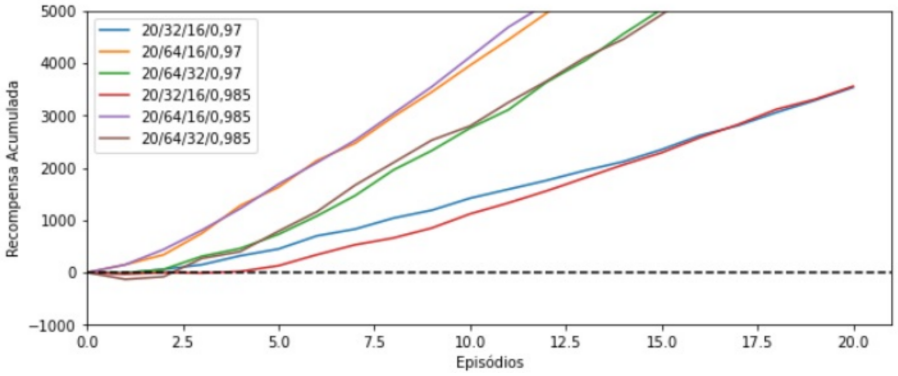

Figura 10. Evolução da recompensa acumulada para o SmartMon.

7. Conclusão

A evolução das SDNs e da programabilidade do plano de dados propiciou a especialização do monitoramento das redes por meio de telemetria. Os dados gerados por esses mecanismos servem de subsídio para algoritmos de aprendizado de máquina, agregando inteligência às chamadas redes definidas por conhecimento. Com o intuito de evitar um consumo exagerado de recursos nas ações de telemetria, este trabalho propõe o SmartMon: um *framework* de telemetria adaptativa capaz de antever problemas pontuais de congestionamento que demandem a intensificação do monitoramento. O SmartMon usa aprendizado por reforço profundo para aprender o comportamento na rede e decidir quando aumentar ou diminuir a telemetria. As análises experimentais comprovaram a eficácia do SmartMon e permitiram avaliá-lo para diferentes configurações de seus hiperparâmetros.

No estudo de caso deste trabalho, que serviu para prova de conceito e avaliações preliminares, a economia provida pela telemetria adaptativa pode não ser suficiente para justificar o arcabouço proposto. Entretanto, num cenário bem maior onde a decisão de aumento ou diminuição de telemetria *in-band* envolva também o número de pontos de coleta desses metadados, assim como o conjunto de fluxos monitores que os carregam, poderá haver um ganho de escala significativo nesta economia. Tal estudo será alvo de trabalhos futuros. Além disso, pretendemos aplicar a técnica de *Double Deep Q-Learning* ao *framework* e estudar o uso de múltiplos agentes em colaboração.

Referências

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38.

Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. (2014). P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95.

Graesser, L. and Keng, W. L. (2020). *Foundations of Deep Reinforcement Learning Theory and Practice in Python*. Addison Wesley Data & Analytics Series.

Hyun, J., Van Tu, N., and Hong, J. W.-K. (2018). Towards knowledge-defined networking using in-band network telemetry. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–7.

Jouet, S. and Pezaros, D. P. (2017). Bpfabric: Data plane programmability for software defined networks. In *2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 38–48.

Krohn, J. and Beyleveld, G. ang Bassens, A. (2020). *Deep Learning Illustrated. A Visual, Interactive Guide to Artificial Intelligence*. Addison Wesley Data & Analytics Series.

Mestres, A., Rodriguez-Natal, A., Carner, J., Barlet-Ros, P., Alarcó n, E., Solé, M., Muntés-Mulero, V., Meyer, D., Barkai, S., Hibbett, M. J., Estrada, G., Ma'ruf, K., Coras, F., Ermagan, V., Latapie, H., Cassar, C., Evans, J., Maino, F., Walrand, J., and Cabellos, A. (2017). Knowledge-defined networking. *ACM SIGCOMM Computer Communication Review*, 47(3):2–10.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Poole, D. L. and Mackworth, A. K. (2017). *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press, USA, 2nd edition.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

Tangari, G., Tuncer, D., Charalambides, M., Qi, Y., and Pavlou, G. (2018). Self-adaptive decentralized monitoring in software-defined networks. *IEEE Transactions on Network and Service Management*, 15(4):1277–1291.

Tu, N. V., Hyun, J., Kim, G. Y., Yoo, J.-H., and Hong, J. W.-K. (2018). Intcollector: A high-performance collector for in-band network telemetry. In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 10–18.

Van Tu, N., Hyun, J., and Hong, J. W.-K. (2017). Towards onos-based sdn monitoring using in-band network telemetry. In *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 76–81.

Xie, J., Yu, F. R., Huang, T., Xie, R., Liu, J., Wang, C., and Liu, Y. (2019). A survey of machine learning techniques applied to software defined networking (sdn): Research issues and challenges. *IEEE Communications Surveys & Tutorials*, 21(1):393–430.

Yao, H., Mai, T., Xu, X., Zhang, P., Li, M., and Liu, Y. (2018). Networkai: An intelligent network architecture for self-learning control strategies in software defined networks. *IEEE Internet of Things Journal*, 5(6):4319–4327.

Yu, M. (2019). Network telemetry: Towards a top-down approach. *SIGCOMM Comput. Commun. Rev.*, 49(1):11–17.