Uma nova abordagem de bits efetivos (BE) para endereços IPv4 com estudo de caso: NeuroCuts.

Eliton Luiz Scardin Perin
UFMS

Jonathan Aldori Alves Oliveira
UFMS

ABSTRACT

One of the fundamental problems in the computer network is how we approach packet classification (PC). Currently, there are numerous solutions that seek to optimize the classification time, memory usage or both. In this context, we use HiCuts, HyperCuts, EffiCuts and CutS-plit algorithms as a comparison parameter for the improvements we will make to NeuroCuts. Our main objective is to improve the use of NeuroCuts memory by using techniques that make use of only the effective bits (EB) necessary to identify the set of rules. Unlike the Multibit decision trees, which are proposed by [3] and use EB to filter most rules, we will use EB to optimize rule entries. This approach is justified given the growing number of connections and rules required for PC. Using dynamic memory allocation, the EB approach obtained an average value of 12% lower when compared to the original value. With static allocation of IPv4 addresses, the average value was 8% lower when compared to the original value. We observed that the gain changes according to the types of Class-Bench[11] analyzed (acl, fw, ipc), and the rules of the "fw"type are those that the EB approach obtains the best gain, whether with dynamic allocation or static. The use of EB makes the NeuroCuts algorithm produce in the majority of experimentsstests smaller trees and uses less bytes per rule.

Keywords: packet classification, deep reinforcement learning, trade-off time-memory, classification time, memory footprint, effective bits.

RESUMO

Um dos problemas fundamentais em rede de computadores é como abordamos a classificação do pacotes (CP). Atualmente existem inúmeras soluções que buscam otimizar o tempo de classificação, uso da memória ou ambos. Neste contexto utilizamos como base da pesquisa os algoritmos HiCuts, HyperCuts, EffiCuts e CutSplit como parâmetro de comparação para as melhorias que faremos no NeuroCuts. Nosso objetivo principal é melhorar a utilização da memória do NeuroCuts com a utilização de técnicas que façam uso apenas do bits efetivos (BE) necessários para identificar o conjunto de regras. Diferente das árvores de decisão Multibit, que é proposto por [3] e utilizam os BE para filtrar a maioria das regras, nós utilizaremos os BE para otimizar as entradas da regra. Esta abordagem se justifica dada o número crescente de conexões e regras necessárias para a CP. Com o uso de alocação dinâmica de memória, a abordagem BE obteve o valor médio de 12% menor quando comparado a valor original. Já com alocação estática dos endereços IPv4, obteve o valor médio de 8% menor quando comparado a valor original. Observamos que o ganho muda de acordo com os tipos de Class-Bench[11] analisados (acl, fw, ipc), sendo que as regras do tipo "fw", são as que a abordagem BE, obtém o melhor ganho, seja com alocação dinâmica ou estática. A utilização de BE faz com que o algoritmo NeuroCuts produza na maioria dos experimentos testados árvores menores e utilize menos bytes por regra.

Palavras chaves: classificação de pacotes, aprendizado por reforço profundo, trade-off, tempo de classificação, uso de memória, bits efetivos.

UFMS Redes de Computadores, Mar - Jul, 2020, Campo Grande, Brasil
2020. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUÇÃO

Aplicações de redes de computadores como firewall, detecção de intrusos e serviços similares em sua maioria são realizados através de classificação de pacotes. Um conjunto de regras estabelece o que deve ser feito com o pacote, a regra define um conjunto de valores ou intervalos de valores. As informações analisadas para classificar o pacote podem ser, principalmente, os valores do endereço de IP (Internet Protocol) de origem e destino, seguido das portas de entrada e saída e também o protocolo de rede.

A classificação de pacotes é similar ao problema de localizar um ponto em um espaço geométrico multidimensional. O problema apresenta uma difícil barganha entre consumo de memória e complexidade de execução, que o torna desafiador. Para solucionar este problema, existem abordagens que se baseiam no uso de árvores de decisão, decomposição, também pode ser resolvida usando probabilidade de prefixo, entropia nas estruturas de dados ou ainda compressão de tabelas. As técnicas utilizam heurísticas ou soluções que se aproximam de uma resposta, mas não resolvem com uma solução ótima para determinadas características de regras.

Na comunidade científica da computação, vem surgindo cada vez mais a necessidade de validação dos experimentos (algoritmos) implementados e publicados pelos autores em diferentes periódicos, por outros pesquisadores familiarizados com os assuntos abordados na pesquisa.

Tal fato se justifica, afim de validar a pesquisa e permitir sua replicabilidade, tornando seus resultados seguros e válidos, para que possam ser referenciados e utilizados por outros pesquisadores ou na solução que se propõem a resolver.

Neste trabalho foi usado os métodos de entropia nas estruturas para melhorar o desempenho no tempo e no uso de memória na classificação de pacotes usando também o algoritmo de aprendizado por reforço profundo proposto em [5] denominado NeuroCuts. Outro trabalho que também explora este mesmo caminho é o contido em [3] e serviu de inspiração para construção deste projeto. Também foram vislumbradas melhorias para o desempenho usando outros métodos, porém não foram implementadas ou estudadas a fundo.

O presente artigo está dividido em 8 seções, contando com esta introdução. Na Seção 2 está relatada as técnicas que são necessária para entender melhor o algoritmo do NeuroCuts, assim como os trabalhos relacionados. A implementação do projeto e demais implementações relacionadas estão descritos na Seção 3. Na Seção 4 apresentamos os resultados encontrados com os experimentos. Na Seção 5 é apresentado os algoritmos e artigos relacionados ao NeuroCuts. Na Seção 6 é exibido os resultados esperados para o projeto.

Por fim, temos as duas Seções 7 e 8 finais, que se referem a trabalhos futuros e há um Apêndice com a tentativa de contato com os autores do NeuroCuts [5] e MultiBit [3].

2 CONTEXTUALIZAÇÃO

Dentre as soluções exploradas por [5] é a utilização de árvores de decisão juntamento com aprendizado por reforço profundo. As duas maneiras básicas de construção da árvore de decisão para classificação de pacotes são:

Corte de nó. É a técnicas que subdivide a árvore em nós ao longo de uma ou mais dimensões de modo que abrangem vários intervalos conduzindo a criação de nós com várias regras. A raiz desta árvore contém todas das regras. Para definir a qual regra pertence um pacote, é necessário percorrer desde a raiz até uma folha. Na Figura 1 está apresentada como é a árvore resultante da sua construção usando a técnica por corte de nó.

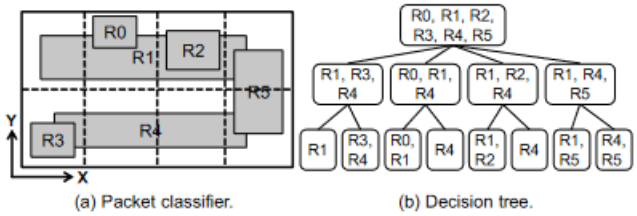


Figura 1: Representação do corte do nó. Imagem extraída de [5].

Regra por partição. Esta abordagem determina uma subdivisão que separa as regras em nós distintos, mesmo que seja necessário replicar intervalos produzindo mais árvores. Isto pode levar ao problema de construção de várias árvores porém com profundidades menores e menor quantidade de comparação [5]. Está apresentada na Figura 2 como são as árvores construídas usando a técnica por partição.

Em resumo, ambas se diferem em três pontos de suas decisões:

(1) Em qual nó aplicar a ação
(2) Qual ação aplicar
(3) Como aplicar

Pensando nas duas primeiras maneiras, o uso do aprendizado por reforço foi proposto para a construção da árvore. O aprendizado por reforço simula a interação de um agente com o ambiente. O agente toma ações dentro do ambiente simulado e este obtêm algum valor em recompensa, podendo ser a favor ou contra o progresso do agente. O algoritmo procura pelas políticas de ações que conduzem para o melhor caminho de acordo com as recompensas que o ambiente lhe devolve. Na Figura 3 está apresentado o diagrama de funcionamento do algoritmo de aprendizado por reforço.

O aprendizado por reforço é conhecido por obter resultados que superam a performance humana em diversos jogos de videogame, como o Atari [6], ou computador [1], e também de tabuleiro como xadrez, shogi, Go [8], onde dos agentes ocorrem jogando entre si.

No caso do NeuroCuts é representado pela Figura 3(b) que leva em consideração a classificação do pacote para realizar a ação da

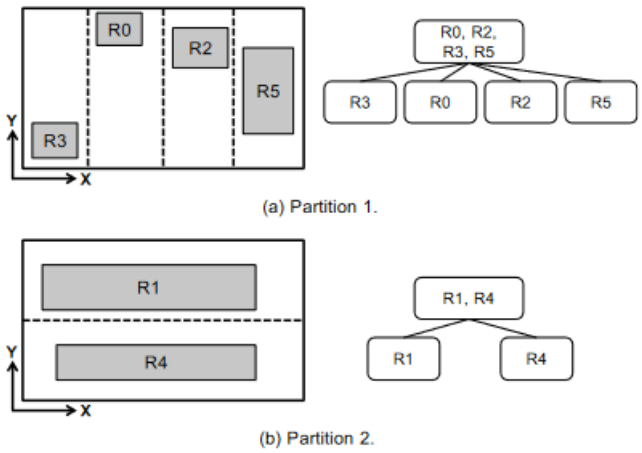


Figura 2: Representação da regra por partição. Imagem extraída de [5].

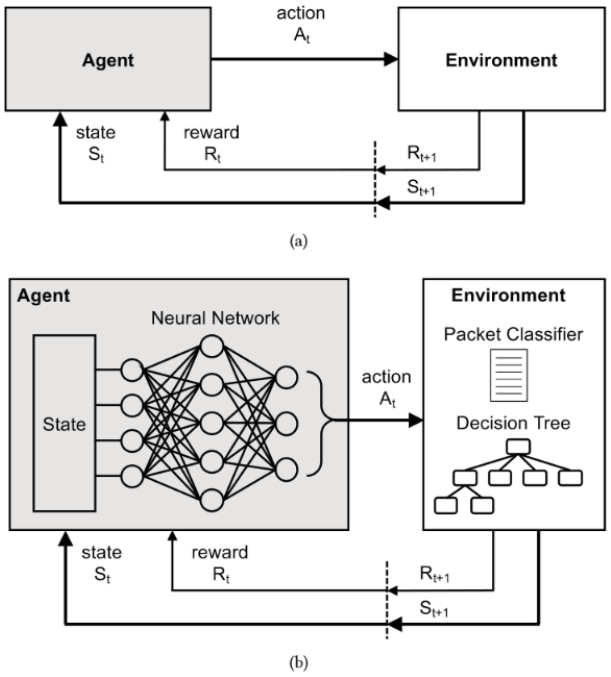


Figura 3: Diagrama do funcionamento do algoritmo de aprendizado por reforço. Retirado de [5].

construção da árvore de decisão por corte ou partição do nó obtido da classificação do pacote. A recompensa é recebida de acordo com os critérios de altura e quantidades de bytes por regra, quanto maior a altura da árvore ou quantidade de bytes por regras menor é a recompensa. O objetivo é aprender as melhores políticas de que o conduzem a receber as maiores recompensas. Para isso o algoritmo constrói várias árvores para verificar quais ações levam a melhor recompensa.

3 IMPLEMENTAÇÃO

A implementação original do artigo está disponível no github: https://github.com/neurocuts/neurocuts. O código está implementado em Python e possui bibliotecas que auxiliam na paralelização do algoritmo e uso de técnicas de aprendizado de máquina. Uma destas bibliotecas é a RLlib ray[1], feita para paralelização de diversos algoritmos, incluindo o aprendizado por reforço. Outra a biblioteca que é uma dependência do RLlib é a Tensorflow[2] para desenvolver algoritmos de aprendizado profundo com redes neurais.

A implementaçãodeste artigo também será em Python pois é feita usando como base o código do NeuroCuts original.

Podemos adotar diferentes abordagens para melhorar CP, com por exemplo:

(1) Sintetizar a árvore gerada pelo NeuroCuts usando BE, o processo é melhor descrito na Seção 3.1 e a avaliação e comparação dos dois métodos é feita na Seção 4.1.
(2) Produzir um autoajuste no aprendizado por reforço no NeuroCuts, afim de encontrar um melhor trade-off entre utilização da memória e desempenho no tempo para classificação do pacote.
(3) Usar outras estruturas de dados com heurísticas de desempenho, por exemplo indexação de banco de dados espaciais.
(4) Uma nova reformulação para a algoritmo de aprendizado por reforço de NeuroCuts.

Como BE é um abordagem de autoria nossa, optamos por utilizá-la e avaliar seu impacto no algoritmo NeuroCuts.

3.1 Algoritmos para seleção do BE

Nesta seção descreveremos cada um dos algoritmos utilizados nos experimentos com o intuito de possibilitar utilização pela comunidade acadêmica, afim de validar sua aplicação neste e em outros experimentos.

Na Figura 4 é apresentado o modelo proposto neste trabalho. A ideia é modificar todos os pacotes a serem classificados pelo algoritmo do NeuroCuts. A modificação é feita nos 4 endereços IPv4 de início e fim do intervalo da regra, tanto para os IPs de origem e destino. Os números da porta e de protocolo não são alterados. Em seguida o pacote é passado para o ambiente de aprendizado por reforço e segue o fluxo do algoritmo do NeuroCuts.

O pseudo-código para encontrar os BE de um endereço IPv4 está descrito abaixo.

Algoritmo 1: Algoritmo que retorna o bits efetivos do endereço IPv4

Entrada: Valores do endereço de origem inicial e final, endereço de destino inicial e final.
Saída: Valor do endereço de origem inicial e final, endereço de destino inicial e final com os BE.

```
1  s = retornaBinario(enderecoIP);
2  s1 = converteDecimal(s[0:8]);
3  s2 = converteDecimal(s[8:16]);
4  s3 = converteDecimal(s[16:24]);
5  s4 = converteDecimal(s[24:32]);
6  enderecoBE = 0;
7  if s4==0 then
8  |  if s4==0 and S3==0 then
9  |  |  if s4 == 0 and s3 == 0 and s2 == 0 then
10 |  |  |  enderecoBE = removeNBE(s[0:8]);
11 |  |  else
12 |  |  |  enderecoBE = s[0:8] + removeNBE(s[8:16]);
13 |  |  end
14 |  else
15 |  |  enderecoBE = s[0:16] + removeNBE(s[16:24]);
16 |  end
17 else
18 |  enderecoBE = s[0:24] + removeNBE(s[24:36]);
19 end
   /* Problema da conversão de decimal com zeros a
      esquerda e colisão de endereços            */
20 enderecoBE = '1' + enderecoBE;
21 retorna converteDecimal(enderecoBE)
```

O pseudo-código para remover os bits que não são efetivos(NBE) de um endereço IPv4 está descrito abaixo.

Algoritmo 2: Algoritmo que retorna o bits efetivos do endereço IPv4 (removeNBE)

Entrada: Valor do octeto de bits que será avaliado
Saída: bits efetivos (BE) do octeto recebido

```
1  valor = 0;
2  comparar = verdadeiro;
3  valorBE = '';
4  foreach caracter in octeto do
5  |  if comparar then
6  |  |  valor = int(caracter);
7  |  |  if valor==1 then
8  |  |  |  comparar = Falso;
9  |  |  |  valorBE += caracter;
10 |  |  else
11 |  |  |  valorBE += caracter;
12 |  |  end
13 end
14 retorna valorBE;
```

[1]https://docs.ray.io/en/master/rllib.html
[2]https://www.tensorflow.org/

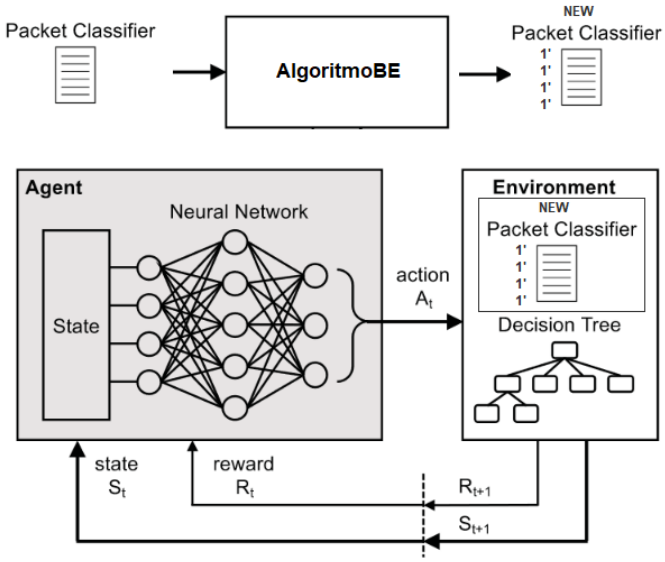


Figura 4: Transformação do pacote a ser classificado com o algoritmo BE. Figura adaptada de [5].

O pseudo-código para retornar os bits que não são efetivos(NBE) de um endereço IPv4 está descrito abaixo.

Algoritmo 3: Algoritmo que retorna O endereço IPv4 original (retornaOriginal)

Entrada: Valores do endereço com os BE
Saída: Valores do endereço orginal sem os BE

```
Remove o 1 inserido para resolver o problema da conversão de decimal com zeros a esquerda e colisão de endereços s = retornaBinario(enderecoIPBE);
s = s[1:];
if len(s) <= 8 then
    s = retornaB(s) + '00000000000000000000000';
else
    if len(s) > 8 and len(s) <= 16 then
        s = s[0:8] + retornaB(s[8:len(s)]) + '0000000000000000';
    else
        if len(s) > 16 and len(s) <= 24 then
            s = s[0:16] + retornaB(s[16:len(s)]) + '00000000';
        else
            s = s[0:24] + retornaB(s[24:len(s)]);
        end
    end
end
retorna converteDecimal(s)
```

O pseudo-código para retornar os bits que foram removidos dos octeto está descrito abaixo.

Algoritmo 4: Algoritmo que retorna o bits do octeto original do endereço IPv4 (retornaB)

Entrada: Valor do octeto de bits que será avaliado
Saída: octeto original

```
valor = '00000000' + octeto;
valor = valor[len(valor)-8:len(valor)];
retorna valor;
```

3.2 Compreenda endereços IP

O endereço IP (Internet Protocol) é um endereço usado para identificar de maneira única um dispositivo em uma rede IP. O endereço na sua versão 4 é composto de 32 bits. Os 32 bits são divididos em quatro octetos (1 octeto = 8 bits). Cada octeto é convertido em decimal e separado por um ponto final (ponto). Por esse motivo, um endereço IPv4 deve ser expressado no formato decimal pontuado (por exemplo, 192.168.0.1). O valor em cada octeto varia de 0 a 255 decimais ou de 00000000 a 11111111 em binário.

Para convertemos cada octetos binários em decimal: começamos do bit da direita, ou bit menos significativo, de um octeto guarda um valor de

$$2^0 \tag{1}$$

, vamos aumentado em 1 o exponencial até chegarmos o o bit mais à esquerda com valor de

$$2^7 \tag{2}$$

. Dessa forma, se todos os bits forem um, o equivalente em decimal seria 255 conforme mostrado na Figura 5:

1	1	1	1	1	1	1	1	Soma:
128	64	32	16	8	4	2	1	255
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	

Figura 5: Exemplo do cálculo em decimal do octeto.

Na Figura 6 está uma conversão de octeto de exemplo no qual nem todos os bits são 1.

0	0	0	1	1	0	0	0	Soma:
0	0	0	16	8	0	0	0	24

Figura 6: Exemplo do cálculo em decimal do octeto onde nem todos são 1.

Na Figura 7 está uma conversão de um endereço IP de exemplo.

10.	1.	1.	19	Decimal
00001010	00000001	00000001	00010011	Binário

Figura 7: Exemplo do cálculo de um endereço IP.

Para exemplificarmos nossa abordagem do BE, a Figura 8 demonstra o funcionamento do algoritmo e o número de bits que serão necessários para cada um dos endereços.

Podemos observar que foi necessário a inclusão de um bit com valor 1 no início de cada endereço, isso se faz necessário dado o fato que podemos perder os bits zerados mais a esquerda nos casos de conversão para decimal e podemos ainda ter colisão de endereços

1	10.	1.	1.	0	Decimal	EB
1	00001010	00000001	1		Binário	18

1	10.	1.	0.	0	Decimal	EB
1	00001010	1			Binário	10

1	10.	0.	0.	0	Decimal	EB
1	1010				Binário	5

1	10.	19.	0.	0	Decimal	EB
1	00001010	10011			Binário	14

Figura 8: Exemplo do funcionamento do algoritmo de EB

com a aplicação de BE. Por isso inserimos este bit que permitirá a mudança de base e eliminará os problemas de colisão.

Esta abordagem nos traz um problema que deverá ser aferido, pois na pior dos cenários, precisaremos de 33 bits para representar um endereço IP. E caso o endereçamento não seja dinâmica, para utilizar apenas os bits necessários, precisaremos de 40 bits para representar o endereço estaticamente.

Para que nossa técnica se torne interessante, é preciso economizar efetivamente mais bits do que os inseridos, seja dinamicamente ou estaticamente alocados. Os resultados do experimentos serão apresentados na seção a seguir.

4 AVALIAÇÃO EXPERIMENTAL

Foram usados para realizar os experimentos o benchmark do ClassBench[11], ele contém pacotes para serem classificados de diversos tamanhos e características. O ClassBench é bem conhecido para comparar desempenho de classificadores de pacotes.

Devido ao pequeno poder computacional e ao tempo restrito para produção do projeto não poderemos realizar testes para o algoritmo do NeuroCutsBE com todos os dados do benchmark e vamos nos restringir aos dados de:

- acl2_1k
- acl4_1k
- fw1_1k
- fw2_1k
- fw3_1k
- fw4_1k
- fw5_1k
- ipc1_1k
- ipc2_1k

Os dados possuem em torno de 1000 regras, por isso o final 1k. O ClassBench possui conjuntos com 10 mil (10k) e 100 mil (100k) também para a mesma inicial citada na lista acima.

Como a aplicação de BE não necessita que seja executado o NeuroCutsBE, podemos avaliar a aplicação de BE nos diferentes Class-Bench[11], para isso, analisamos os endereços de origem e destino, inicial e final para todos os arquivos.

Podemos observar na Figura 9, onde estão listados todos os resultados dos experimentos, no qual foram aplicados o algoritmo BE para todos os endereços IPv4, que se alocados dinamicamente os endereços IP originais e BE, ainda assim em todos os ClassBench[11], o algoritmo BE utiliza-se de menos memória para alocação dos endereços, obtendo o valor médio de aproximadamente 12%, o que representa o valor médio de 3 bits a menos por endereço.

Arquivo	Número Endereços	Qtde BE(bits)	Qtde Normal(bits)	Percentual	Média por Endereço
acl1_1k	3.768	110.162	117.337	6,51%	1,90
acl1_10k	39.096	1.154.671	1.208.708	4,68%	1,38
acl1_100k	397.276	11.776.029	12.341.996	4,81%	1,42
acl2_1k	3.844	92.379	111.997	21,24%	5,10
acl2_10k	37.904	925.947	1.120.045	20,96%	5,12
acl2_100k	300.400	7.518.932	9.052.628	20,40%	5,11
acl3_1k	3.960	107.634	116.483	8,22%	2,23
acl3_10k	37.676	1.017.018	1.114.259	9,56%	2,58
acl3_100k	397.924	10.742.042	11.725.325	9,15%	2,47
acl4_1k	3.960	102.460	113.940	11,20%	2,90
acl4_10k	38.374	1.029.452	1.136.335	10,38%	2,79
acl4_100k	396.208	10.776.680	11.810.560	9,59%	2,61
acl5_1k	3.732	103.770	106.900	3,02%	0,84
acl5_10k	29.192	878.034	909.920	3,63%	1,09
acl5_100k	392.720	11.862.394	12.175.364	2,64%	0,80
fw1_1k	3.428	75.279	86.877	15,41%	3,38
fw1_10k	37.516	843.120	952.787	13,01%	2,92
fw1_100k	352.324	8.101.409	9.019.880	11,34%	2,61
fw2_1k	3.884	71.376	98.539	38,06%	6,99
fw2_10k	38.620	735.252	1.008.630	37,18%	7,08
fw2_100k	384.312	7.287.230	10.017.125	37,46%	7,10
fw3_1k	3.196	66.921	77.907	16,42%	3,44
fw3_10k	36.148	786.534	898.529	14,24%	3,10
fw3_100k	335.008	7.595.968	8.493.956	11,82%	2,68
fw4_1k	3.388	74.861	88.130	17,72%	3,92
fw4_10k	35.100	787.793	917.572	16,47%	3,70
fw4_100k	335.868	7.697.019	8.861.503	15,13%	3,47
fw5_1k	3.456	70.431	84.965	20,64%	4,21
fw5_10k	35.332	748.257	884.688	18,23%	3,86
fw5_100k	335.188	7.247.511	8.440.090	16,46%	3,56
ipc1_1k	3.896	102.133	116.773	14,33%	3,76
ipc1_10k	38.072	992.554	1.128.445	13,69%	3,57
ipc1_100k	397.300	10.401.774	11.792.772	13,37%	3,50
ipc2_1k	2.784	72.326	78.821	8,98%	2,33
ipc2_10k	40.000	1.044.050	1.096.735	5,05%	1,32
ipc2_100k	400.000	10.518.669	11.012.623	4,70%	1,23
Total	**4.910.854**	**123.518.071**	**138.319.144**	**11,98%**	**3,01**

Figura 9: Ganho percentual de espaço necessário para alocação dinâmica em memória e média de bits economizado por endereço IPv4.

Como podemos observar, o ganho muda de acordo com os tipos de ClassBench[11] analisados (acl, fw, ipc). Sendo que as regras do tipo "fw", são as que o algoritmo BE, obtém o melhor ganho. O resultado do agrupamento médio pode ser observado na Figura 10.

Arquivo	Número Endereços	Qtde BE(bits)	Qtde Normal(bits)	Percentual	Média por Endereço
acl	2.086.034	58.197.604	63.161.797	8,53%	2,38
fw	1.942.768	42.188.961	49.931.178	18,35%	3,99
ipc	882.052	23.131.506	25.226.169	9,06%	2,37

Figura 10: Média de Ganho percentual de espaço necessário para alocação dinâmica em memória e média de bits economizado por endereço IPv4 por tipo de regra.

Podemos representar os resultado obtidos graficamente, o que facilita nossa comparação, conforme Figuras 11 à 14. Para melhor

visualização dos gráficos alguns nomes de arquivos foram ocultados, mas seguem a mesma ordem listada na Figura 9.

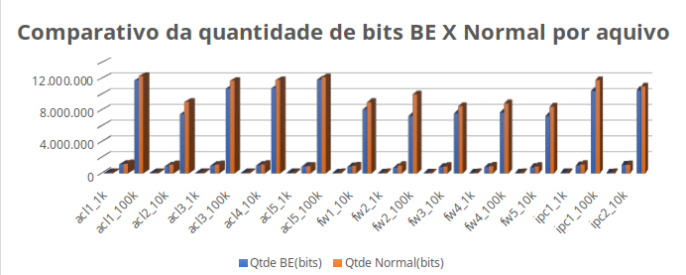


Figura 11: Gráfico comparativo da quantidade de bits necessários dinamicamente (BE vs Normal) por aquivo.

Figura 12: Gráfico do percentual de melhora na utilização do espaço com o algoritmo BE por arquivo.

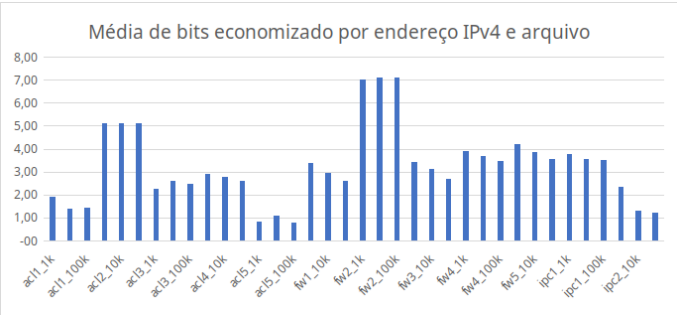


Figura 13: Gráfico de bits economizados por endereço IPv4 e arquivo.

Partindo do pressuposto que alocar memória dinamicamente por endereço seria um trabalho custoso. Executamos os experimentos para calcular quanto de memória seria necessário se alocarmos estaticamente em múltiplos de 8 bits (1 byte) apenas os byte(s) necessários com a aplicação do algoritmo BE, para o endereço original, neste caso, utilizamos o valor fixo de 32 bits, ou 4 bytes.

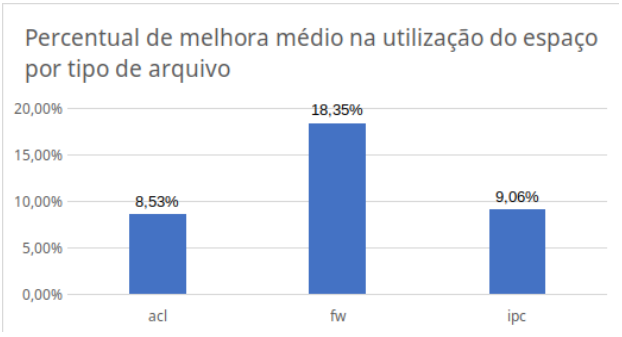


Figura 14: Gráfico do percentual de melhora médio na utilização do espaço por tipo de arquivo

Com essa alteração, agora o algoritmo BE, poderá alocar para cada endereço, no mínimo 1 byte e no pior caso 5 bytes. Neste caso para que o algoritmo BE seja mais eficiente que o método de endereçamento atual, será necessários que os número de endereços com 5 bytes sejam compensados pelos endereços com 1, 2 e 3 bytes. E que o somatório desses espaços seja menor que o somatório dos 4 bytes por endereço original. O resultado dos experimentos podem ser observados na Figura 15.

Arquivo	Número Endereços	Endereço com 1 byte	Endereço com 2 byte	Endereço com 3 byte	Endereço com 4 byte	Endereço com 5 byte	Limitação da Alocação (bits)	Qtde Normal(bits)	Qtde Memória Excedente
acl1_1k	3.768	31	87	145	3.193	312	119.776	120.576	-0,66%
acl1_10k	39.096	198	960	3.317	24.604	10.017	1.284.560	1.251.072	2,68%
acl1_100k	397.276	1.786	8.149	21.570	285.005	80.766	13.013.152	12.712.832	2,36%
acl2_1k	3.844	310	380	852	1.518	784	108.944	123.008	-11,43%
acl2_10k	37.904	2.570	5.106	6.676	16.228	7.324	1.074.736	1.212.928	-11,39%
acl2_100k	300.400	13.723	38.881	55.742	137.263	54.791	8.653.744	9.612.800	-9,98%
acl3_1k	3.960	336	157	371	1.876	1.220	122.936	126.720	-2,99%
acl3_10k	37.676	2.867	1.628	4.251	18.915	10.015	1.156.888	1.205.632	-4,04%
acl3_100k	397.924	31.251	15.950	37.862	218.471	94.390	12.180.568	12.733.568	-4,34%
acl4_1k	3.960	308	244	312	2.875	221	114.696	126.720	-9,49%
acl4_10k	38.374	2.438	2.322	4.327	20.289	8.998	1.169.672	1.227.968	-4,75%
acl4_100k	396.208	25.367	21.115	37.051	221.226	91.449	12.167.192	12.678.656	-4,03%
acl5_1k	3.732	0	63	420	2.878	371	118.024	119.424	-1,17%
acl5_10k	29.192	0	636	2.238	18.133	8.185	971.544	934.144	4,00%
acl5_100k	392.720	9	7.111	21.520	267.369	96.711	13.054.576	12.567.040	3,88%
fw1_1k	3.428	899	212	104	1.284	929	91.328	109.696	-16,74%
fw1_10k	37.516	9.248	2.281	840	15.343	9.804	1.013.776	1.200.512	-15,55%
fw1_100k	352.324	81.238	21.080	7.680	147.180	95.146	9.687.104	11.274.368	-14,08%
fw2_1k	3.884	1.271	289	449	1.249	626	90.576	124.288	-27,12%
fw2_10k	38.620	12.302	2.697	4.325	12.980	6.316	913.368	1.235.840	-26,09%
fw2_100k	384.312	123.818	25.943	44.022	127.342	63.187	9.064.584	12.297.984	-26,29%
fw3_1k	3.196	972	205	52	987	980	83.088	102.272	-18,76%
fw3_10k	36.148	9.363	2.480	962	14.193	9.150	957.848	1.156.736	-17,19%
fw3_100k	335.008	82.568	20.242	6.701	130.226	95.271	9.123.312	10.720.256	-14,90%
fw4_1k	3.388	783	201	475	1.054	875	89.608	108.416	-17,35%
fw4_10k	35.100	7.535	1.908	4.056	13.777	7.824	941.976	1.123.200	-16,13%
fw4_100k	335.868	64.367	15.859	41.107	140.239	74.296	9.214.736	10.747.776	-14,26%
fw5_1k	3.456	1.059	255	82	1.200	860	87.320	110.592	-21,04%
fw5_10k	35.332	9.848	2.641	878	13.230	8.735	914.872	1.130.624	-19,08%
fw5_100k	335.188	89.515	23.597	8.575	129.036	84.465	8.807.224	10.726.016	-17,89%
ipc1_1k	3.896	222	220	836	1.720	898	116.320	124.672	-6,70%
ipc1_10k	38.072	2.439	2.150	6.614	18.693	8.176	1.137.864	1.218.304	-6,60%
ipc1_100k	397.300	24.686	20.235	70.782	199.317	82.280	11.889.360	12.713.600	-6,48%
ipc2_1k	2.784	400	80	75	1.457	772	83.784	89.088	-5,95%
ipc2_10k	40.000	6.292	1.141	719	17.393	14.455	1.220.624	1.280.000	-4,64%
ipc2_100k	400.000	64.309	9.976	5.730	174.466	145.519	12.215.280	12.800.000	-4,57%
Total	**4.910.854**	**674.328**	**256.481**	**401.718**	**2.402.209**	**1.176.118**	**143.054.960**	**157.147.328**	**-8,97%**

Figura 15: Ganho percentual de espaço necessário para alocação estática em memória, com o número de endereços por bytes necessário para representa-los utilizando ao Algoritmo BE.

Podemos observar na Figura 15, onde estão listados todos os resultados dos experimentos, no qual foram aplicados o algoritmo BE para todos os endereços IPv4, que se alocados estaticamente os endereços IP originais e BE, ainda assim na maioria dos ClassBench[11]

(exceção: acl1_10k, acl1_100k, acl5_10k e acl5_100k), o algoritmo BE utiliza-se de menos memória para alocação dos endereços, obtendo o valor médio de aproximadamente 8%.

Apenas para os ClassBench[11](acl1_10k, acl1_100k, acl5_10k e acl5_100k) que a alocação original utiliza-se de menos memória para representar os mesmos endereços. Como é possível avaliar antecipadamente, conforme demonstrados na Figura 15, o ganho ou não do uso do algoritmo de BE, o usuário pode avaliar e optar por utilizar ou não o algoritmo BE nestes casos.

Novamente podemos observar, que o ganho muda de acordo com os tipos de ClassBench[11] analisados (acl, fw, ipc). Sendo que as regras do tipo "fw", são as que o algoritmo BE, obtém o melhor ganho com alocação estática. O resultado do agrupamento médio pode ser observado na Figura 16.

Arquivo	Número Endereços	Endereço com 1 byte	Endereço com 2 byte	Endereço com 3 byte	Endereço com 4 byte	Endereço com 5 byte	Limitação da Alocação (bits)	Qtde Normal(bits)	Qtde Memória Excedente
acl	2.086.034	81.194	102.789	196.654	1.239.843	465.554	65.311.008	66.753.088	-2,16%
fw	1.942.768	494.786	119.890	120.308	749.320	458.464	51.080.720	62.168.576	-17,84%
ipc	882.052	98.348	33.802	84.756	413.046	252.100	26.663.232	28.225.664	-5,54%

Figura 16: Média de Ganho percentual de espaço necessário para alocação estática em memória e média de bits economizado por endereço IPv4 por tipo de regra.

São apresentados a seguir, os experimentos no algoritmo NeuroCuts original e com o nosso algoritmo BE para representar os endereços IPv4, que denominamos NeuroCutsBE. As métrica avaliadas são a quantidade de bytes por regra e o tempo de classificação, que é dado pela profundidade máxima da árvore nos ClassBench listados anteriormente.

Primeiro, realizamos testes para confirmar o desempenho do algoritmo do NeuroCuts nos datasets utilizados para retirar avaliação de tempo de classificação das regras (profundidade da árvore) para geração da árvore de decisão conforme a Figura 17. Este experimento serviu de base para validar os resultado apresentado pelos autores e para validar que as correções que fizemos no algoritmo original, que não compilava. Por isso, conforme relatado no apêndice tentamos contato com o autor original, porém não obtivemos sucesso.

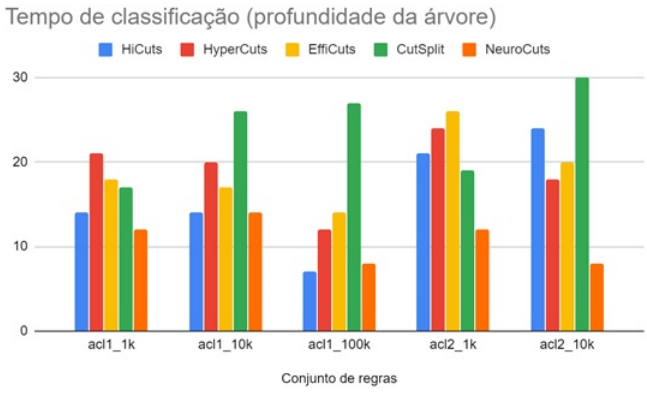


Figura 17: Tempo de classificação do NeuroCuts original.

Na seção a seguir está apresentado os resultados obtidos na execução do NeuroCuts original e com o algoritmo BE que denominamos NeuroCutsBE.

4.1 Avaliação do BE (NeuroCutsBE) quando comparado ao NeuroCuts

Conforme proposto em [5] com o NeuroCuts, aplicamos os algoritmos HiCuts, HyperCuts, EffiCuts, CutSplit e NeuroCuts como parâmetro de comparação para as melhorias que fizemos em NeuroCutsBE. Primeiramente é importante salientar que utilizamos como parâmetros:

- –partition-mode : None
- –depth-weight: 0 e 1 (uma rodada para cada)

Não utilizamos as demais variações possíveis ao parâmetro –partition-mode (simple, efficuts, cutsplit), pois como já dito anteriormente, nosso tempo e poder computacional foi limitado.

Como resultado das métricas avaliadas, quantidade de bytes por regra e profundidade da arvore, obtivemos os resultados apresentados nas Figuras 18 e 19.

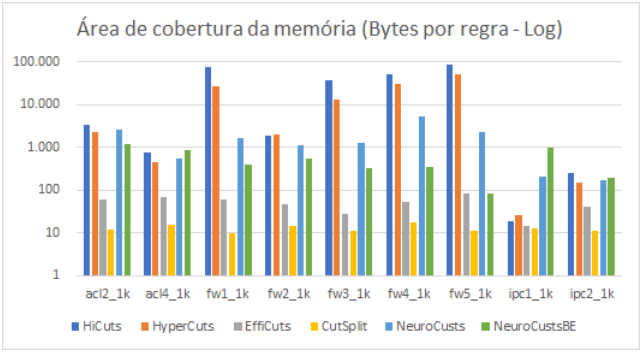


Figura 18: Bytes por regra em escala logarítmica.

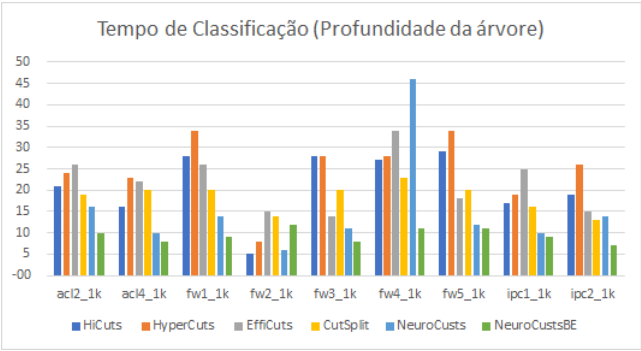


Figura 19: Profundidade da árvores.

Podemos observar na Figura 18 que nos ClassBench (acl4_1k, ipc1_1k e ipc2_1k) o NeuroCuts obteve resultado melhor que NeuroCutsBE, nos demais o resultado do NeuroCutsBE foi superior ao NeuroCuts original. Já na Figura 19 onde avaliamos a profundidade das árvores geradas, podemos observar que somente no ClassBench (fw2_1k) o NeuroCuts obteve resultado melhor que NeuroCutsBE, nos demais o NeuroCutsBE produziu árvores menores que o NeuroCuts original e os demais algoritmos comparados. Esse resultado

reforça que a utilização de BE faz com que o algoritmo produza árvores menores e utilize menos bytes por regra.

Precisamos citar algumas ressalvas:

- Que o algoritmo NeuroCuts com BE deve apresentar melhor resultado em conjunto de dados maiores.
- Não comparamos os algoritmos NeuroCuts e NeuroCutsBE com os parâmetros: simple, efficuts e cutsplit, para adicionarmos outras ações de partição de regra, além da ação de corte implementada do NeuroCuts, que poderia melhorar os resultados.

Para facilitar a comparação entre o algoritmo do NeuroCuts original e o NeuroCutsBE, separamos os resultado de ambos. Isso pode ser observado nas Figuras 20 e 21.

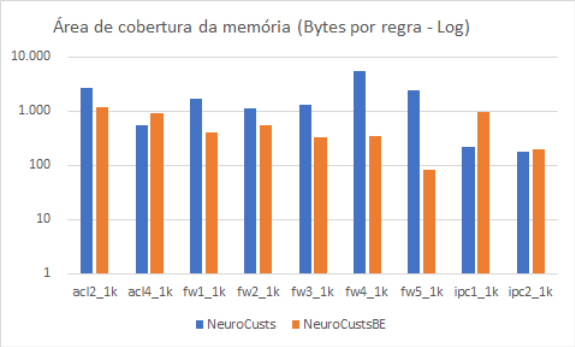


Figura 20: Bytes por regra em escala logarítmica.

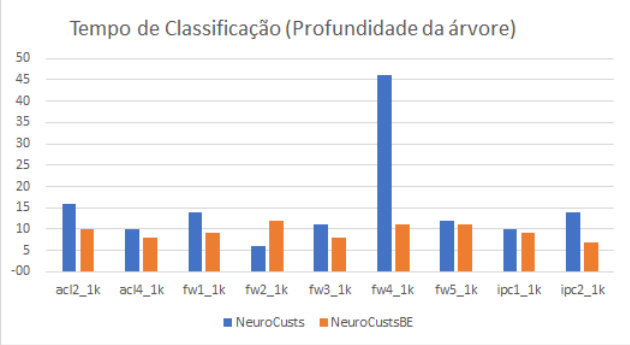


Figura 21: Profundidade da árvores.

5 TRABALHOS RELACIONADOS

Como mencionado na Seção 2, algoritmos de construção de árvore de decisão para classificação se baseiam em cortar ou particionar os nós para a construção da árvore.

O HiCuts apresenta uma heurística para construção de uma árvore [2] [3]. Algoritmos como o HyperCuts [9] [4], o CutSplit [4] [5] e HiCuts são os mais antigos e foram utilizados para compor o benchmark de comparações.

O algoritmo do EffiCuts é uma heurística que constrói a árvore por meio de regra de partição [12] [6]. Seus resultados superam as técnicas de HiCuts em menor tempo de acesso e HyperCuts em fatores de memória utilizada.

O NeuroCuts é o primeiro trabalho reportado a utilizar aprendizado profundo por reforço para resolver heuristicamente o problema de classificação de pacotes [5][7]. O trabalho apresenta uma abordagem que produz a árvore de modo autônomo, ou seja, sem auxílio humano para construção e pode ser usada para aprimorar árvores que já estão em produção.

Logo, surgiram trabalhos que otimizam o NeuroCuts como o artigo Multibit Tries Packet Classification with Deep Reinforcement Learning [3] [8] que melhora o desempenho usando menos bits para fazer a classificação, alcançando 55% de melhora no tempo de classificação. Outros trabalhos que citam o NeuroCuts também foram encontrados, os exemplificam como implementações que exploram o poder de uso de aprendizado profundo por reforço dentro de demais contexto de rede de computadores. Por exemplo, em Solving System Problems with Machine Learning [10][9] é exibido aplicações diversas de Aprendizado de Máquina para melhorar soluções, ou ainda dentro do domínio de classificação de pacotes, no trabalho A Computational Approach to Packet Classification [7] [10] que faz comparações com o NeuroCuts.

6 RESULTADOS

O algoritmo BE utiliza-se de menos memória para alocação dinâmica dos endereços IPv4, obtendo o valor médio de a aproximadamente 12% quando comparado a valor original, o que representa o valor médio de aproximadamente 3 bits a menos por endereço.

Com a alocação estática, o algoritmo BE também utiliza-se de menos memória para alocar os endereços IPv4, obtendo o valor médio de aproximadamente 8% quando comparado a valor original, mesmo que no pior caso precise de 5 bytes (40 bits) para representar o endereço.

Observamos que o ganho muda de acordo com os tipos de ClassBench[11] analisados (acl, fw, ipc). Sendo que as regras do tipo "fw", são as que o algoritmo BE, obtém o melhor ganho, seja com alocação dinâmica ou estática.

Nossa abordagem de BE pode ser aplicada a outras heurísticas e algoritmos que precisam trabalhar com o conjunto de 32 bits dos endereços e máscaras de rede do protocolo IPv4. A otimização do consumo de memória dependerá do número de regras e o a distribuição dos endereços IPv4.

Podemos observar que nos ClassBench (acl4_1k, ipc1_1k e ipc2_1k) o NeuroCuts obteve resultado melhor que NeuroCutsBE, nos demais casos o resultado do NeuroCutsBE foi superior ao NeuroCuts original.

Quando avaliamos a profundidade das árvores geradas, podemos observar que somente no ClassBench (fw2_1k) o NeuroCuts obteve resultado melhor que NeuroCutsBE, nos demais o NeuroCutsBE

[3]http://www.academia.edu/download/32951877/HOTI_99.pdf
[4]https://dl.acm.org/doi/pdf/10.1145/863955.863980
[5]https://ieeexplore.ieee.org/abstract/document/8485947
[6]https://dl.acm.org/doi/pdf/10.1145/1851275.1851208
[7]https://dl.acm.org/doi/abs/10.1145/3341302.3342221
[8]https://ieeexplore.ieee.org/abstract/document/9098974
[9]https://sic.ici.ro/wp-content/uploads/2019/06/Art.-1-Issue-2-SIC-2019.pdf
[10]https://arxiv.org/pdf/2002.07584.pdf

produziu árvores menores que o NeuroCuts original e os demais algoritmos comparados. Esse resultado reforça que a utilização de BE faz com que o algoritmo produza árvores menores e utilize menos bytes por regra.

Com as reformulações inseridas no NeuroCuts, tentamos propor uma solução mais eficaz para o problema de CP. Nossa ideia é fornecer melhorias significativas no consumo de memória em comparação com os algoritmos citados e com a versão original do NeuroCuts.

Se partimos do pressuposto que agora necessitamos de menos bits para representar o endereço IPv4 original, podemos construir árvores binárias que precisarão percorrer menos nós para chegar ao nó procurado.

7 TRABALHOS FUTUROS

Nossa abordagem de BE pode ser aplicada em algoritmos de construção de árvores binárias de endereços IPv4, para chegar mais rapidamente ao nó de destino que representa o endereço, quando comparado aos 32 bits dos endereços de IP original. Deixamos este exemplo para que os pesquisadores validem a eficácia de BE em árvores binárias para endereços IP.

Testar o algoritmo NeuroCutsBE em conjunto de dados maiores para validar se efetivamente ele apresenta um resultado ainda melhor nesses conjunto de dados.

Testar e comparar os algoritmos NeuroCuts e NeuroCutsBE com os parâmetros: simple, efficuts e cutsplit, para adicionarmos outras ações de partição de regra, além da ação de corte implementada do NeuroCuts.

REFERÊNCIAS

[1] Christopher Amato and Guy Shani. 2010. High-Level Reinforcement Learning in Strategy Games. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1 (AAMAS '10)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 75–82.

[2] Pankaj Gupta and Nick McKeown. 1999. Packet classification using hierarchical intelligent cuttings. In *Hot Interconnects VII*, Vol. 40.

[3] Hasibul Jamil and Ning Weng. 2020. Multibit Tries Packet Classification with Deep Reinforcement Learning. In *2020 IEEE 21st International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 1–6.

[4] Wenjun Li, Xianfeng Li, Hui Li, and Gaogang Xie. 2018. Cutsplit: A decision-tree combining cutting and splitting for scalable packet classification. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2645–2653.

[5] Eric Liang, Hang Zhu, Xin Jin, and Ion Stoica. 2019. Neural packet classification. In *Proceedings of the ACM Special Interest Group on Data Communication*. 256–269.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[7] Alon Rashelbach, Ori Rottenstreich, and Mark Silberstein. 2020. A Computational Approach to Packet Classification. *arXiv preprint arXiv:2002.07584* (2020).

[8] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144.

[9] Sumeet Singh, Florin Baboescu, George Varghese, and Jia Wang. 2003. Packet classification using multidimensional cutting. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. 213–224.

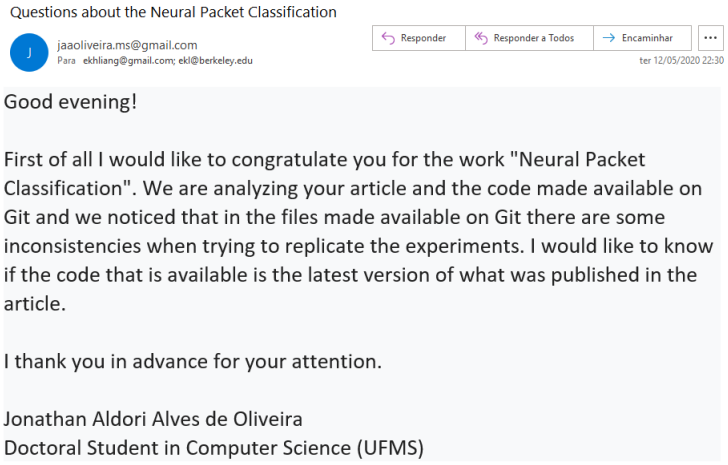
[10] Ion Stoica. 2019. Solving System Problems with Machine Learning. *Studies in Informatics and Control* 28, 2 (2019), 119–132.

[11] David E Taylor and Jonathan S Turner. 2007. Classbench: A packet classification benchmark. *IEEE/ACM transactions on networking* 15, 3 (2007), 499–511.

[12] Balajee Vamanan, Gwendolyn Voskuilen, and TN Vijaykumar. 2010. EffiCuts: optimizing packet classification for memory and throughput. *ACM SIGCOMM Computer Communication Review* 40, 4 (2010), 207–218.

8 APÊNDICE - RESTRIÇÕES DA PESQUISA

O principal problema encontrado por nós foram algumas inconsistências do código fonte disponibilizado por meio dos autores [5], no endereço do github: https://github.com/neurocuts/neurocuts. Percebemos que não se trata da última implementação referenciada no artigo original. Tentamos entrar em contato com o autor, porem até a presente data não obtivemos retorno (Figura 22).


Questions about the Neural Packet Classification

jaaoliveira.ms@gmail.com
Para ekhliang@gmail.com; ekl@berkeley.edu

Responder | Responder a Todos | Encaminhar

ter 12/05/2020 22:30

Good evening!

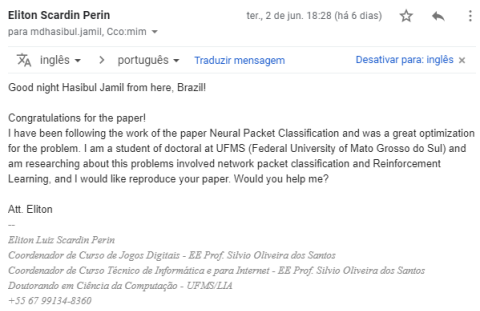
First of all I would like to congratulate you for the work "Neural Packet Classification". We are analyzing your article and the code made available on Git and we noticed that in the files made available on Git there are some inconsistencies when trying to replicate the experiments. I would like to know if the code that is available is the latest version of what was published in the article.

I thank you in advance for your attention.

Jonathan Aldori Alves de Oliveira
Doctoral Student in Computer Science (UFMS)

Figura 22: E-mail enviado para os autores do NeuroCuts.

Já os autores de do Multibit [3], nos responderam, mas não disponibilizaram o código fonte de sua implementação. O argumento utilizado foi que estão trabalhando ainda no algoritmo (Figura 23, Figura 24).


Eliton Scardin Perin ter., 2 de jun. 18:28 (há 6 dias)
para mdhasibul.jamil, Cco:mim

inglês › português Traduzir mensagem Desativar para: inglês

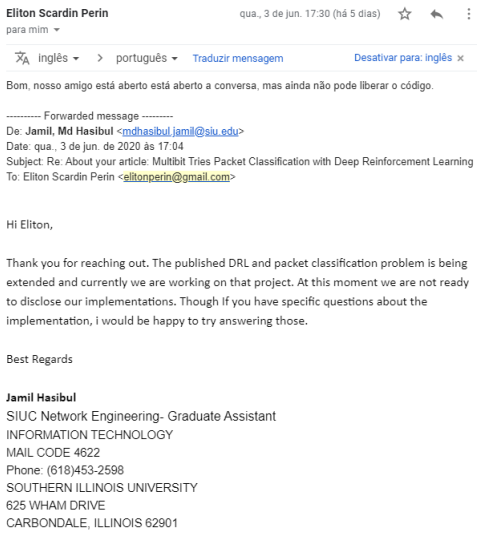
Good night Hasibul Jamil from here, Brazil!

Congratulations for the paper!
I have been following the work of the paper Neural Packet Classification and was a great optimization for the problem. I am a student of doctoral at UFMS (Federal University of Mato Grosso do Sul) and am researching about this problems involved network packet classification and Reinforcement Learning, and I would like reproduce your paper. Would you help me?

Att. Eliton
--
Eliton Luiz Scardin Perin
Coordenador de Curso de Jogos Digitais - EE Prof. Silvio Oliveira dos Santos
Coordenador de Curso Técnico de Informática e para Internet - EE Prof. Silvio Oliveira dos Santos
Doutorando em Ciência da Computação - UFMS/LIA
+55 67 99134-8360

Figura 23: E-mail enviado para os autores do Multibit.

Devido ao poder computacional e ao tempo restrito para a produção do projeto, não foi possível realizar testes com conjunto de dados grandes, maiores que 10k dos benchmark's.



Eliton Scardin Perin qua., 3 de jun. 17:30 (há 5 dias)
para mim

inglês > português Traduzir mensagem Desativar para: inglês ×

Bom, nosso amigo está aberto está aberto a conversa, mas ainda não pode liberar o código.

---------- Forwarded message ---------
De: **Jamil, Md Hasibul** <mdhasibul.jamil@siu.edu>
Date: qua., 3 de jun. de 2020 às 17:04
Subject: Re: About your article: Multibit Tries Packet Classification with Deep Reinforcement Learning
To: Eliton Scardin Perin <elitonperin@gmail.com>

Hi Eliton,

Thank you for reaching out. The published DRL and packet classification problem is being extended and currently we are working on that project. At this moment we are not ready to disclose our implementations. Though If you have specific questions about the implementation, i would be happy to try answering those.

Best Regards

Jamil Hasibul
SIUC Network Engineering- Graduate Assistant
INFORMATION TECHNOLOGY
MAIL CODE 4622
Phone: (618)453-2598
SOUTHERN ILLINOIS UNIVERSITY
625 WHAM DRIVE
CARBONDALE, ILLINOIS 62901

Figura 24: E-mail de resposta dos autores do Multibit.