Um Mecanismo de Aprendizado Incremental para Detecção e Bloqueio de Mineração de Criptomoedas em Redes Definidas por Software

Helio N. C. Neto[1], **Martin Andreoni Lopez**[2], **Natalia C. Fernandes**[1], **Diogo M. F. Mattos**[1]

[1]MídiaCom - PPGEET/TET/UFF
Universidade Federal Fluminense - UFF
[2]Samsung Research Institute - Brasil

Resumo. *A mineração não autorizada de criptomoedas implica o uso de valiosos recursos de computação e o alto consumo de energia. Este artigo propõe o mecanismo MineCap, um mecanismo dinâmico e em linha para detectar e bloquear fluxos de mineração não autorizada de criptomoedas, usando o aprendizado de máquina em redes definidas por software. O MineCap desenvolve a técnica de super aprendizado incremental, uma variante do super learner aplicada ao aprendizado incremental. O super aprendizado incremental proporciona ao MineCap precisão para classificar os fluxos de mineração ao passo que o mecanismo aprende com dados recebidos. Os resultados revelam que o mecanismo alcança 98% de acurácia, 99% de precisão, 97% de sensibilidade e 99,9% de especificidade e evita problemas relacionados ao desvio de conceito.*

Abstract. *Covert mining of cryptocurrency implies the use of valuable computing resources and high energy consumption. In this paper, we propose MineCap, a dynamic online mechanism for detecting and blocking covert cryptocurrency mining flows, using machine learning on software-defined networking. MineCap uses a novel technique called super incremental learning, a variant of the super learner with incremental learning. Hence, we design an accurate mechanism to classify mining flows that learn with incoming data with an average of 98% accuracy, 99% accuracy, 97% sensitivity and 99.9% specificity and avoid concept drift-related issues.*

1. Introdução

As redes corporativas são alvos constantes de ataques [Ingols, 2009, Porras e Valdes, 2001] e a mineração não autorizada de criptomoedas é uma nova ameaça a essas redes. Tal atividade prejudica o ambiente corporativo, devido ao consumo excessivo de recursos de computação e energia, embora nem sempre a mineração seja explicitamente proibida na rede [de Oliveira et al., 2019, Tahir et al., 2017]. Uma variação dessa ameaça ocorre em aplicativos que mineram criptomoedas sem o conhecimento ou consentimento do usuário, como *malwares* ou injeção de *scripts* em páginas Web.

Este trabalho foi realizado com recursos da CNPq, CAPES, FAPERJ, RNP e TAESA.

Em ambientes corporativos, a ameaça da mineração de criptomoedas pode ser interna ou externa. A ameaça interna consiste em usuários internos que usam conscientemente o poder computacional da infraestrutura da empresa para realizar a mineração para fins particulares. Por outro lado, as ameaças externas consistem em entidades que exploram vulnerabilidades da infraestrutura para executar códigos de mineração de criptomoeda. Portanto, identificar o tráfego de mineração torna-se essencial para garantir o uso eficiente e seguro dos recursos computacionais da empresa. Contudo, esta é uma tarefa complexa, já que as características de fluxo de mineração geralmente são semelhantes às do tráfego criptografado de navegação Web, tornando-se simples ocultar o tráfego malintencionado. Assim, há uma clara necessidade de ferramentas para detectar e impedir o tráfego não autorizado de mineração em ambientes em que as políticas de rede proíbem a atividade de mineração de criptomoeda.

Este artigo propõe o MineCap, um mecanismo que utiliza algoritmos de aprendizado de máquina para realizar a classificação em linha do tráfego de mineração de criptomoeda em redes definidas por *software* (*Software-Defined Networks* - SDN). Nas redes definidas por *software*, o plano de controle e encaminhamento são desacoplados e introduz-se uma nova entidade, o controlador SDN [Bannour et al., 2018], responsável por executar todas as ações de controle na rede. O controlador SDN permite que a rede se torne programável e que os aplicativos de controle interajam com os elementos de rede, como os comutadores, de forma logicamente centralizada [Mattos et al., 2016]. Assim, as SDN permitem um bloqueio efetivo dos fluxos de mineração, por meio de uma interação direta entre o MineCap e o controlador da rede. Assim, o MineCap consegue bloquear dinamicamente os fluxos de mineração que são reconhecidos com o algoritmo de aprendizado de máquina.

O artigo propõe ainda a técnica de super aprendizado incremental que é uma variante da técnica *super learner* aplicada ao aprendizado incremental. A ideia chave da proposta é realizar o aprendizado incremental sobre as classificações providas por um agregado de classificadores de menor acurácia. Para tanto, avaliou-se o desempenho de quatro classificadores: Floresta Aleatória, *Naive Bayes*, Regressão Logística e *Gradient Boosted Tree*. Os dois algoritmos com melhor desempenho foram usados na técnica de super aprendizado incremental, com o objetivo de aumentar a acurácia da detecção. O super aprendizado incremental utiliza a técnica de *super learner* [Van der Laan et al., 2007] combinada com o aprendizado incremental, de forma que o super modelo aprende com novos dados sem o esquecimento catastrófico e consequente descarte do classificador.

O MineCap executa sobre o arcabouço de processamento Apache Spark, através das bibliotecas Spark *Streaming*, que manipula o fluxo de dados, e a MLlib, que fornece os algoritmos de aprendizado de máquina na classificação em linha. A avaliação do protótipo do MineCap foi realizada em uma rede emulada pela plataforma Mininet, utilizando o controlador SDN Ryu com a API REST habilitada para instalar regras de bloqueio dos fluxos. Os resultados das avaliações mostram que o MineCap tem baixo consumo dos recursos de rede e computacional enquanto analisa todo o tráfego de rede. Trabalhos relacionados se concentram em identificar o tráfego de mineração usando o aprendizado de máquina [Tahir et al., 2017, Konoth et al., 2018]. Contudo, eles protegem

Disponível em https://github.com/mininet/mininet.
Disponível em https://osrg.github.io/ryu/.

um único *host*, enquanto o MineCap protege toda a rede e, por ser baseado no paradigma SDN, permite a definição de políticas de bloqueio em alto nível [Mattos et al., 2016] diretamente nos elementos de rede mais próximos às fontes de tráfego.

O restante do artigo está organizado da seguinte forma. A Seção 2 discute os trabalhos relacionados. A Seção 3 apresenta as técnicas para o processamento de grandes massas de dados, e a Seção 4 propõe a técnica de super aprendizado incremental. O mecanismo MineCap é definido na Seção 5 e a Seção 6 apresenta os resultados da avaliação. Por fim, a Seção 7 conclui o artigo.

2. Trabalhos Relacionados

Tahir *et al.* propõem a ferramenta MineGuard para detectar em tempo real o comportamento de processos de mineração em máquinas virtuais [Tahir et al., 2017]. MineGuard utiliza contadores de desempenho de *hardware* (*Hardware Performance Counters* - HPCs), um conjunto de registradores integrados em processadores modernos, para armazenar as contagens de atividades relacionadas ao *hardware*. Os contadores permitem rastrear com precisão as operações de mineração de baixo nível ou eventos dentro da CPU e GPU com sobrecarga mínima, dando à ferramenta a capacidade de detectar com precisão, em tempo real, se uma máquina virtual está tentando minerar criptomoedas. A MineGuard baseia-se na observação de que, para minerar criptomoedas, é necessário executar repetidamente o algoritmo de *Proof-of-Work* (PoW).

O MineSweeper [Konoth et al., 2018] usa a URL do sítio *Web* como entrada para verificar se há algum *software* de mineração de criptomoedas oculto. O MineSweeper usa uma métrica para identificar funções criptográficas, medindo o número de operações executadas por um aplicativo. Os autores analisam o algoritmo CryptoNight e suas variantes, mas argumentam que adicionar outros algoritmos é uma tarefa trivial. O MineSweeper concentra-se na mineração embarcada em páginas Web e seu principal objetivo é identificar um novo ataque, o *drive-by mining*, no qual um sítio infectado executa secretamente código JavaScript ou um módulo WebAssembly no navegador do usuário para minerar criptomoedas sem o seu consentimento. Diferentemente do MineSweeper, o MineCap concentra-se em detectar e bloquear tráfego de mineração de criptomoeda na rede.

Andreoni *et al.* propõem a CATRACA, uma ferramenta em linha para a detecção e prevenção de ataques em uma função virtual de rede [Andreoni Lopez et al., 2019]. Assim como o MineCap, a ferramenta proposta utiliza um sistema de processamento de fluxo para as grandes massas de dados (*Big Data*), o Apache Spark, fornecendo um serviço de detecção de ameaças em tempo real. A ferramenta CATRACA opera em dois modos: *online* ou *offline*. O processo de detecção inclui algoritmos de seleção de características e aprendizado de máquina para discriminar tráfego normal, negação de serviço (*Denial of Service* - DoS) e varredura de portas. Em uma detecção de ameaça, o sistema pode reagir prontamente e criar regras de bloqueio em um *firewall*. O MineCap usa uma abordagem semelhante, mas integra a capacidade de bloqueio de fluxos utilizando uma rede definida por *software* sem a necessidade de usar outras ferramentas.

3. Processamento de Grandes Massas de Dados

As técnicas tradicionais de banco de dados não permitem o processamento de dados estruturados e não estruturados em grandes volume, variedade e velocidade.

Na maioria dos ambientes corporativos, o volume e a velocidade de geração dos dados excede a taxa de transferência dos gerenciadores de banco de dados relacionais [McAfee et al., 2012]. Para a extração de conhecimentos úteis dos metadados para aumentar a receita, conquistar ou reter clientes e melhorar as operações, as empresas coletam dados de várias fontes, incluindo e-mails, dispositivos móveis, aplicativos, bancos de dados, servidores e mídias diversas. Os dados obtidos devem ser extraídos, formatados, manipulados, armazenados e analisados. A análise das grandes massas de dados pode ocorrer em lote ou em fluxo [Medeiros et al., 2019]. O processamento em lote é a forma mais tradicional de processamento de grandes massas de dados. Normalmente, os dados são armazenados para serem processados posteriormente por meio de realização de consultas [Carbone et al., 2015]. No processamento por lotes, é comum usar *Online Analytical Processing* (OLAP), sistemas capazes de analisar grandes volumes de dados de forma rápida e eficiente, técnicas de mineração de dados e processamento paralelo com ferramentas como o Apache Hadoop. O processamento em fluxo é comumente usado por aplicativos que exigem baixo tempo de resposta para combinar a captura de dados em fluxo (*in-stream*) com o uso de técnicas de inferência e correlação [Medeiros et al., 2019].

No MineCap, todos os pacotes na saída da rede são processados, a fim de identificar os fluxos de mineração de criptomoeda. O MineCap utiliza o processamento de dados em fluxo devido ao grande número de pacotes na saída da rede, que combina características de alto volume de dados e alta velocidade de geração. Existem ferramentas de processamento distribuído em fluxo de código aberto, como a Apache Flink, a Apache Storm e a Spark Streaming, que funcionam em aglomerados computacionais, mas a única que garante a entrega de 100% das mensagens em cenários de falha é a Spark Streaming [Andreoni Lopez et al., 2019]. A Spark Streaming permite a programação de aplicações nas linguagens Scala, Java, Python e R. Por essas vantagens o MineCap utiliza o Spark Streaming para processamento em fluxo de grandes massas de dados.

4. Proposta de Super Aprendizado Incremental

A técnica de super aprendizado incremental, proposta neste artigo, é a combinação das técnicas *super learner* [Van der Laan et al., 2007] e aprendizado incremental [Fei-Fei et al., 2007]. Na proposta, algoritmos de aprendizado de máquina são usados como modelos candidatos que alimentam o super modelo. Os modelos candidatos são treinados e realizam as previsões que são entregues como entrada para o super modelo. Então, posteriormente o super modelo é treinado parcialmente utilizando o aprendizado incremental.

A técnica do *super learner* consiste em treinar um modelo no qual as amostras de entrada são a saída de outros modelos, semelhante a um sistema hierárquico. Van der Laan *et al.* [Van der Laan et al., 2007] foram os primeiros a usar esta técnica, utilizando a validação cruzada, para propor um novo método de previsão criado a partir da combinação ponderada de diversos modelos candidatos. O primeiro passo do *super learner* proposto por van der Laan *et al.* é treinar n modelos utilizando todo o conjunto de dados. No segundo passo, é feita a divisão do conjunto de dados original em V blocos e, depois, cada modelo candidato é treinado com um desses blocos, chamados de blocos de validação. Logo, é feita a previsão dos blocos de dados de validação correspondente a cada bloco de treinamento. Na sequência, são feitos ajustes do resultado observado dos resultados previstos de cada modelo candidato e, por fim, é feito o treinamento do super modelo

com os resultados previstos pelos modelos candidatos. Pode-se avaliar o super modelo comparando com as previsões de cada modelo candidato, feito no primeiro passo, com a saída do super modelo. A Figura 1 mostra o diagrama da técnica original do *super learner*.

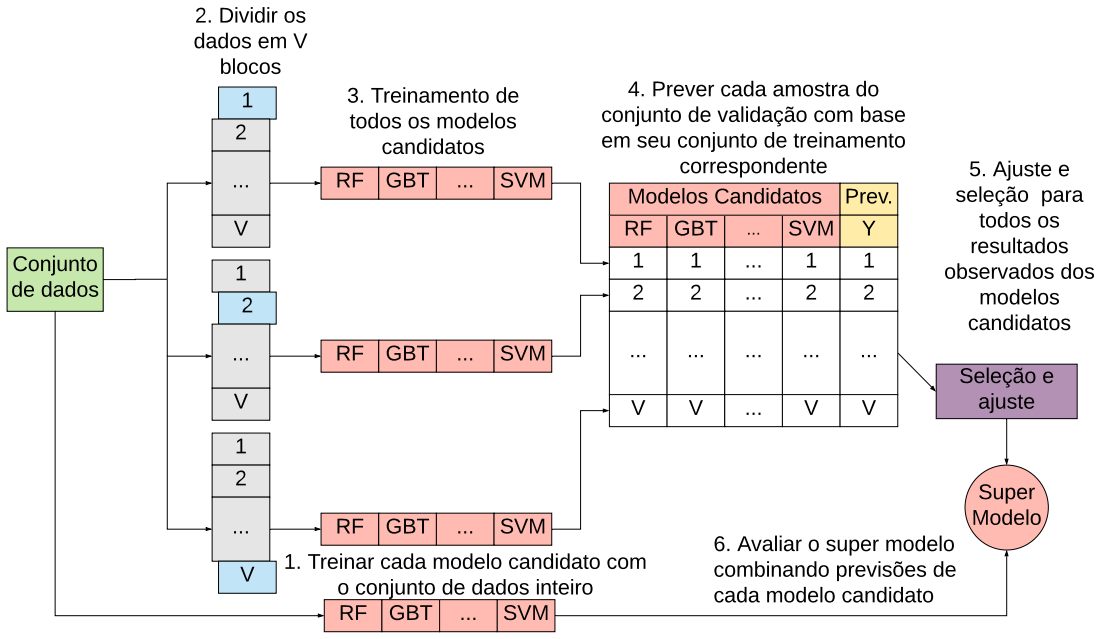


Figura 1. Diagrama de fluxo com as etapas da técnica *Super Learner*. Adaptado de [Van der Laan et al., 2007]

A ideia principal do aprendizado incremental, também conhecido como aprendizado em linha, é limitar a perda de informação para o uso de dados finitos em relação a modelos com dados infinitos. Para tanto, a perda de informação é feita em função do número de amostras em cada estágio de aprendizado e, assim, o número de amostras utilizadas em cada um dos estágios é mínimo e mantido livre do limiar de perdas preservadas [Medeiros et al., 2019]. A resolução do problema de quanta informação é perdida diminuindo o número de amostras é dada usando o limite de Hoeffding [Gama et al., 2014]. Considerando uma variável aleatória real x, cujo valor está contido no intervalo R, presume-se que n observações independentes da variável são feitas e a média $\bar{r}$ é computada. O limite de Hoeffding garante com probabilidade $1 - \delta$ que a verdadeira média da variável é dada por pelo menos $\bar{x} - \epsilon$, em que

$$\epsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}}. \tag{1}$$

O limite de Hoeffding é independente da distribuição que gera a variável x. A partir desse resultado, os algoritmos de aprendizado de máquina para treinamento de dados em fluxos são desenvolvidos. No entanto, assume-se que os valores gerados pela variável x vêm de um processo estocástico constante. Nos casos em que há uma mudança no processo que gera a variável usada no treinamento de métodos de aprendizado em fluxo, é dito que ocorre uma mudança de conceito (*concept drift*) e, portanto, faz-se necessário um novo treinamento do método de aprendizado [Wang et al., 2013].

Abordagens típicas para o aprendizado de novas informações envolvem manter o comportamento estocástico dos dados ou realizar o descarte do classificador existente e re-treinar com os dados acumulados até o momento. Nas abordagens que consideram o fim da estabilidade estatística dos dados, o processo deixa de ser estocástico, resultando na perda de todas as informações previamente adquiridas, o que é conhecido como esquecimento catastrófico. Assim, Polikar *et al.* [Polikar et al., 2001] definem que os algoritmos de aprendizado incremental devem satisfazer os requisitos a seguir ao obter informações adicionais sobre novos dados: i) não requerer acesso aos dados originais usados para treinar o classificador existente; ii) preservar o conhecimento previamente adquirido, isto é, o classificador não deve sofrer o esquecimento catastrófico; e iii) ter capacidade de acomodar novas classes que podem ser introduzidas por novos dados. Assim, os classificadores que adotam o aprendizado incremental não requerem o treinamento total do classificador no caso de uma mudança no comportamento do fluxo de dados.

Na proposta super aprendizado incremental é feita uma mesclagem da variante do *super learner* e, para gerar o super modelo, são usadas redes neurais com aprendizado incremental para o aprendizado em linha. Na variante do *super learner*, é feita a divisão do conjunto de dados em um conjunto de treinamento e um conjunto de teste. Em seguida, utiliza-se o conjunto de treinamento para treinar os modelos candidatos. No caso do MineCap, foram utilizados os algoritmos floresta aleatória e *Gradient Boosted Tree*, treinando ambos com o mesmo conjunto de dados. Cada amostra do conjunto de dados de entrada para o super modelo tem como atributos as probabilidades da amostra ser da classe fluxo normal, classificado como 0, ou fluxo de mineração, classificado como 1, geradas a partir das previsões feitas dos modelos candidatos aplicados sobre o conjunto de dados de teste.

Os dois modelos candidatos geram a lista $W = (w_1, w_2, w_3, w_4, y)$ em que w_1 é a probabilidade do primeiro modelo candidato retornar como saída a classe tráfego normal, w_2 é a probabilidade do primeiro modelo candidato retornar como saída a classe de trafego de mineração, w_3 e w_4 têm a mesma representação para o segundo modelo candidato. Finalmente, y é a saída desejada de cada amostra, ou seja a classe alvo dessa amostra. A lista W gerada pelos modelos candidatos é passada como entrada para treinar a rede neural incremental utilizando o algoritmo de Perceptron Multi-Camadas (*Multi-Layer Perceptron* - MLP). A Figura 2 mostra o diagrama da técnica de super aprendizado incremental desenvolvida no MineCap.

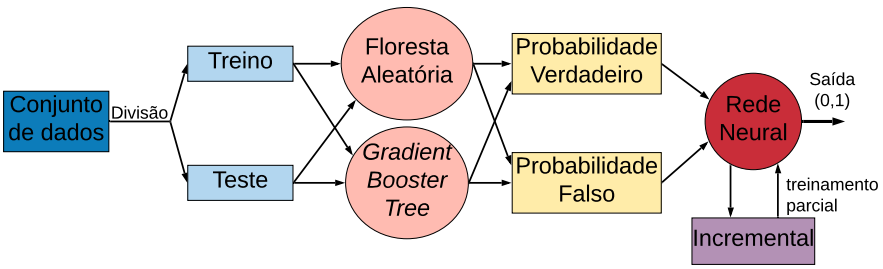


Figura 2. Diagrama de fluxo da técnica de super aprendizado incremental.

Quando o MineCap recebe novos dados, os envia aos modelos candidatos para classificação e, em seguida, armazena as saídas em um arquivo chamado *incremental*. Somente depois, envia os dados como entrada para a rede neural para realizar a previsão.

A rede neural classifica cada amostra recebida em linha e anexa a saída prevista à entrada da amostra no arquivo *incremental*. Cada entrada do arquivo *incremental* possui a lista $W = (w_1, w_2, w_3, w_4, y)$, em que todos os w são a saída dos modelos candidatos, floresta aleatória e *Gradient Boosted Tree* e, no caso do MineCap, y é a saída do super modelo. Quando um número k suficiente de amostras é acumulado, o processo incremental verifica cada amostra e coleta apenas as amostras que possuem maior probabilidade de serem fluxo de mineração e menor probabilidade de serem fluxo normal ou o inverso para depois realizar a aprendizagem parcial. A quantidade k de amostras acumuladas para o treinamento parcial é proporcional ao tempo para a aprendizagem com novos dados. Quanto maior a quantidade, maior o tempo para a aprendizagem. No protótipo do mecanismo propostos, é usado o valor de $k = 50$, pois verificou-se que com 50 amostras há o compromisso de executar o treinamento parcial do super modelo entre 1 e 2 minutos, no caso de ocorrência constante de mineração na rede.

5. Mecanismo MineCap para Detecção e Bloqueio de Mineração

O mecanismo MineCap é executado em uma estação (*host*) ou em um aglomerado computacional (*cluster*) separado do controlador de rede. A execução do MineCap em uma estação separada do controlador de rede evita a sobrecarga no controlador. O controlador da rede redireciona todos os pacotes de saída da rede local para o MineCap, aplicando a técnica de espelhamento de porta no *gateway* de rede. Conforme mostrado na Figura 3, o mecanismo proposto apresenta uma arquitetura de três camadas: captura, processamento e bloqueio.

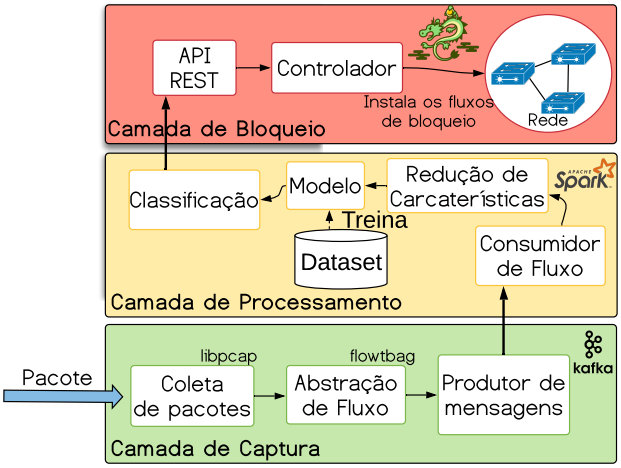


Figura 3. A arquitetura da ferramenta MineCap. Os pacotes chegam na interface de rede, são abstraídos em fluxos que, por sua vez, são publicados no serviço de mensagens Apache Kafka. O Spark Streaming consome a mensagem, pré-processa os dados e envia para o modelo treinado para classificação.

A **camada de captura** tem como objetivo capturar os dados e prepará-los para a camada superior. O primeiro passo é capturar os pacotes de rede através da execução da biblioteca `libpcap`. Esses pacotes capturados são resumidos em fluxos de rede. Define-se fluxo de rede como a sequência de pacotes que possuem o mesmo IP de origem, IP de destino, porta de origem, porta de destino e protocolo de transporte. Foi desenvolvida uma aplicação na linguagem Python, baseada na aplicação `flowtbag`, para realizar a

Disponível em https://github.com/DanielArndt/flowtbag.

abstração de pacotes em fluxos. Os fluxos são publicados no serviço de mensagens Apache Kafka, um sistema de publicação e assinatura (*publisher/subscriber*) que garante o armazenamento e a entrega confiável das mensagens.

A **camada de processamento** é responsável por consumir, processar e classificar os fluxos da rede que o Apache Kafka fornece como dados em fluxo. O mecanismo MineCap adota o Apache Spark como sua plataforma de processamento de fluxo em linha. O Apache Spark Streaming apresenta melhor desempenho em relação à tolerância a falhas quando comparado a outras plataformas de processamento de fluxo [Andreoni Lopez et al., 2016], adicionando robustez e resiliência ao processamento. O MineCap evita assim a perda de informação. O Spark consome o conteúdo do Kafka com a biblioteca Spark Streaming. O Apache Kafka é um intermediário de mensagens que entrega mensagens aos processos assinantes, mas não suporta execução de algoritmos mais complexos, como o aprendizado de máquina. O MineCap incorpora o algoritmo de Análise de Componente Principal (*Principal Component Analysis* - PCA) [Andreoni Lopez et al., 2016], que reduz um grande conjunto de atributos a um conjunto menor de características artificiais que ainda contém a maior parte da quantidade de informação do conjunto original. A `MLlib` é a biblioteca de aprendizado de máquina escalável do Spark que consiste em algoritmos e utilitários comuns para o aprendizado de máquina. Neste trabalho, são avaliados quatro algoritmos de aprendizado de máquina: Floresta Aleatória, *Gradient Boosted Tree*, *Naive Bayes* e Regressão Logística.

A **camada de bloqueio** recebe a saída do classificador e instala uma regra de bloqueio para fluxos rotulados como mineração de criptomoeda. O OpenFlow 1.3 [Open Networking Foundation, 2012] é o protocolo de rede definida por *software* utilizado pelo MineCap. O Ryu é baseado em Python e foi escolhido como o controlador SDN para o protótipo desenvolvido, devido à sua facilidade de implantação e baixo tempo para o desenvolvimento de aplicações. No entanto, qualquer outro controlador poderia ser utilizado. A integração entre o MineCap e o controlador é agnóstica, o que possibilita a integração ou atualização de outros controladores SDN. O MineCap se comunica com o controlador por meio de um interface de Transferência de Estado Representacional(*Representational State Transfer* - REST) executada no controlador, que recebe uma mensagem criptografada, na qual os fluxos de mineração são identificados e, então, as regras de bloqueio são instaladas na rede. Essa é a camada de integração com a rede, para o MineCap funcionar com as redes tradicionais basta modificar essa camada. Por exemplo, pode ser substituída a chamada REST para o controlador SDN por um comando via ssh para o bloqueio no firewall da rede.

6. Avaliação do Protótipo

Um protótipo do MineCap foi desenvolvido para avaliar a proposta. Os experimentos foram realizados em um computador equipado com um processador Intel Core i7 7700 a 3,60 GHz, com oito núcleos e 16 GB de RAM. A avaliação foi realizada em uma rede emulada, usando a plataforma de emulação `Mininet`, com 16 *hosts* em uma topologia em árvore personalizada, usando sete comutadores executando o protocolo OpenFlow 1.3 [Open Networking Foundation, 2012].

Na primeira avaliação, entre os 16 *hosts* do ambiente, quatro deles executavam

Disponível em https://osrg.github.io/ryu/.

aplicativos de mineração de criptomoedas em linha de comando populares para mineração utilizando CPU e o restante estava repetindo um tráfego de 30 minutos contido em um arquivo de captura real, gerado por usuários reais, de tráfego de rede, usando `tcpreplay`. Nessa etapa, é avaliada a diferença entre os dois melhores modelos dentre os demais. O tráfego contido no arquivo de captura foi previamente classificado para ser comparado com a saída de modelos de aprendizado de máquina, para ser utilizado como linha de base. Utilizou-se *pools* de mineração com portas TCP conhecidas, para posterior classificação manual do conjunto de dados utilizado para treinar os modelos candidatos. Nos testes de avaliação, utilizaram-se *pools* diferentes das utilizadas no conjunto de dados de treinamento. O *gateway* de rede teve sua porta espelhada para o MineCap para garantir que todo o tráfego de rede passasse pela classificação. O MineCap instala um fluxo, com duração de 5 minutos, no controlador SDN bloqueando todo o tráfego classificado como mineração de criptomoeda e, portanto, os pacotes são rejeitados diretamente no comutador OpenFlow da rede. O conjunto de dados utilizado para treinar e testar os modelos foi criado em laboratório, em um ambiente estilo "mundo fechado", em que o tráfego foi capturado em um ambiente controlado com um comutador, computadores atuando como mineradores e outros utilizando diferentes perfis de navegação como *streaming* de vídeo, descarga de arquivos e acesso a sítios Web. Os aplicativos de mineração utilizados foram o MinerGate e o GuiMiner, executando em máquinas com sistema operacional Windows. Antes de particionar o conjunto de dados em conjunto de treinamento e teste, o conjunto de dados foi embaralhado para evitar qualquer elemento de viés ou padrões nos conjuntos de dados. Assim, com o conjunto de dados embaralhado, a qualidade e o desempenho dos modelos são melhoradas. O conjunto de dados passou pelo processo de *oversampling*, pois a técnica é frequentemente usada para balancear a quantidade de classes do conjunto de dados [Luengo et al., 2011].

A Figura 4 mostra a curva de características operacionais do receptor (*Receiver Operating Characteristic* - ROC), a precisão, a sensibilidade e a especificidade de cada algoritmo de classificação testado. A curva ROC mede e especifica o desempenho dos algoritmos testados através do relacionamento entre verdadeiros positivos e falsos positivos em diversos pontos de corte na probabilidade de uma amostra pertencer a uma classe. É um método gráfico robusto e direto que permite estudar a variação da sensibilidade e especificidade para diferentes valores de corte.

Os algoritmos de aprendizado de máquina Floresta Aleatória e *Gradient Boosted Tree* superaram os outros algoritmos, com boa precisão e sensibilidade. Regressão Logística e Naive Bayes tiveram 0 % de precisão e sensibilidade, por isso ficaram sem barras na Figura 4(b). A Regressão Logística classificou cada fluxo como fluxo normal e o *Naive Bayes* classificou alguns fluxos como normal e outros como fluxos de mineração de criptomoedas, assim, eles não serão utilizados como modelos candidatos do super aprendizado incremental.

Além dos testes de avaliação de aprendizado de máquina, são apresentados outros

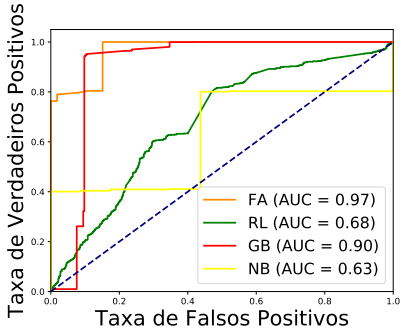
---

O tráfego de mineração usado para treinar os algoritmos de aprendizado de máquina se originam da execução dos aplicativos de mineração `cpuminer` e `xmrig`.

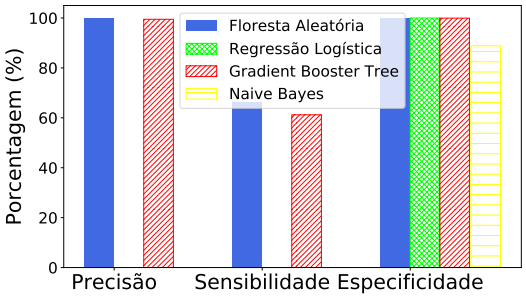
Disponível em https://github.com/appneta/tcpreplay.

Disponível em https://minergate.com.

Disponível em https://guiminer.org.



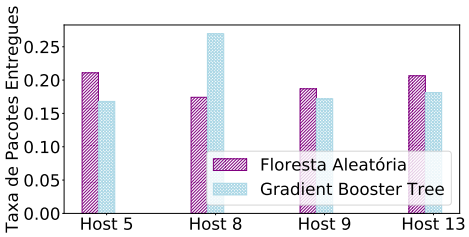
(a) Curva de características operacionais do receptor (ROC).



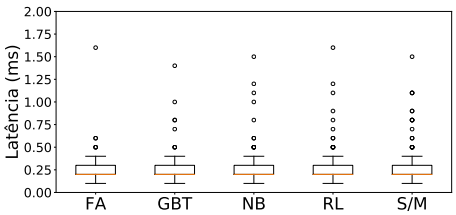
(b) Precisão, sensibilidade e especificidade dos algoritmos de classificação avaliados.

Figura 4. Avaliação dos algoritmos de aprendizado de máquina. a) O algoritmo de Floresta Aleatória apresenta uma Área Abaixo da Curva (AUC) de 0,97 e, assim, apresenta a melhor relação de compromisso entre sensibilidade e especificidade. b) *Gradient Boosted Tree* também apresenta altas taxas de sensibilidade e especificidade, porém são inferiores às da Floresta Aleatória.

dois testes de desempenho. O primeiro é a relação dos pacotes entregues e os pacotes gerados. O segundo consiste em um teste de latência da rede para verificar se o MineCap gera sobrecarga. Como os algoritmos de aprendizado de máquina *Naive Bayes* e Regressão Logística obtiveram mal desempenho, apenas os algoritmos de Floresta Aleatória e *Gradient Boosted Tree* são apresentados nas próximas avaliações. A Figura 5(a) mostra a taxa de tráfego entregue para os quatro *hosts* que estavam minerando criptomoedas. Vale a pena observar que o algoritmo de Floresta Aleatória bloqueia pelo menos 80% do tráfego de mineração, enquanto o *Gradient Boosted Tree* atinge 25% do tráfego de mineração entregue no `Host 8`. Em média, o algoritmo de Floresta Aleatória bloqueia mais tráfego na rede do que o *Gradient Boosted Tree*, devido sua maior capacidade de generalização do conhecimento obtido. Destaca-se também que os algoritmos baseados em árvore tiveram melhor desempenho do que outros devido à natureza discreta dos dados de rede [Andreoni Lopez et al., 2016].



(a) Taxa de pacotes de mineração de criptomoedas entregues em um tráfego gerado de 30 min.



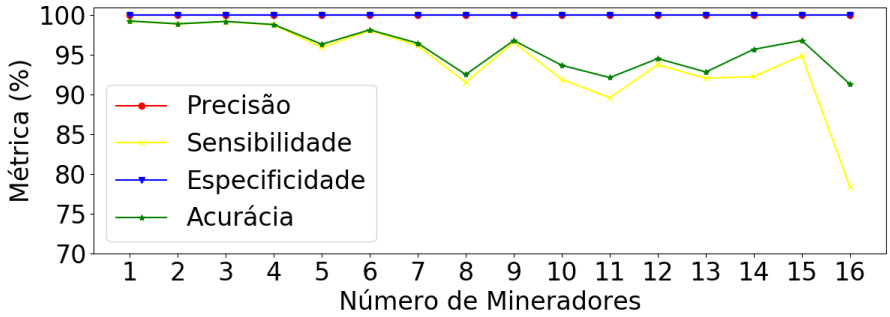
(b) Latência de comunicação no cenário avaliado.

Figura 5. Avaliação da latência na rede com e sem a ferramenta MineCap e gráfico da taxa de pacotes de mineração entregue. a) O classificador Floresta Aleatória bloqueou 80% do tráfego de mineração enquanto o *Gradient Boosted Tree* encaminhou mais que 25% do tráfego de mineração. b) O MineCap não acrescenta latência na rede. Floresta Aleatória (FA), *Gradiente Boosted Tree* (GBT), *Naive Bayes* (NB), Regressão Logística (RL), Sem MineCap (S/M).

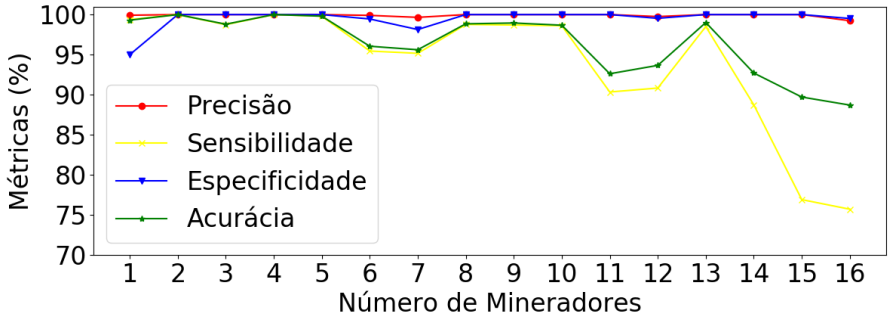
A latência da rede foi analisada enquanto os testes eram realizados gerando pa-

cotes ICMP (Internet Control Message Protocol) de um *host* na rede para o controlador. O teste de latência foi executado nos testes com todos os algoritmos e, também, em um novo cenário reproduzindo o mesmo tráfego sem intervenção do MineCap. Os resultados mostram que o mecanismo MineCap não implica mais latência no encaminhamento de pacotes e, assim, mantém o desempenho da rede do cenário sem a adoção do MineCap, como mostra a Figura 5(b). Também foi avaliado, separadamente, o tempo de resposta de chamada REST que foi insignificante, mostrando um atraso de menos de 1 ms em uma rede local, pois o tamanho dos pacotes nas chamadas é muito pequeno. O tempo de bloqueio é em média de 2 minutos, usando ambos os algoritmos, Floresta Aleatória e *Gradient Boosted Tree*, porque às vezes o algoritmo classifica fluxos de mineração de criptomoedas como fluxos normais. Os modelos de aprendizado de máquina são passíveis de falhas, já que falsos positivos e negativos existem nos problemas de aprendizado de máquina em que não há sobreajuste [Pietraszek e Tanner, 2005].

Floresta Aleatória e *Gradient Boosted Tree* obtiveram resultados similares, porém os testes anteriores utilizavam uma quantidade estática de mineradores. É importante avaliar se o aumento na quantidade de mineradores na rede impacta o desempenho dos classificadores. Assim, o teste com 30 minutos de tráfego foi realizado variando a quantidade de mineradores.



(a) Precisão, sensibilidade, especificidade e precisão do modelo Floresta Aleatória para cenários com diferentes números de mineradores.

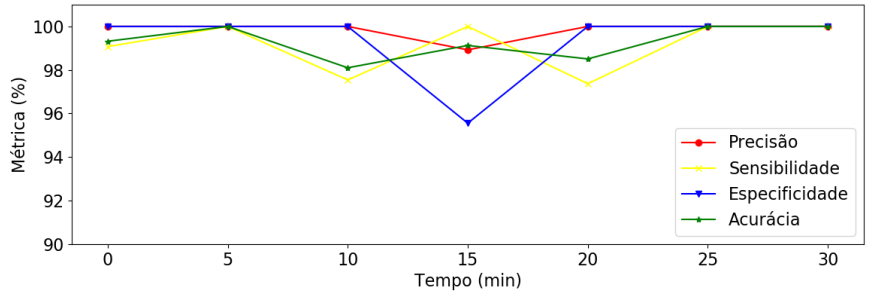


(b) Precisão, sensibilidade, especificidade e precisão do modelo *Gradient Boosted Tree* para cenários com diferentes números de mineradores.

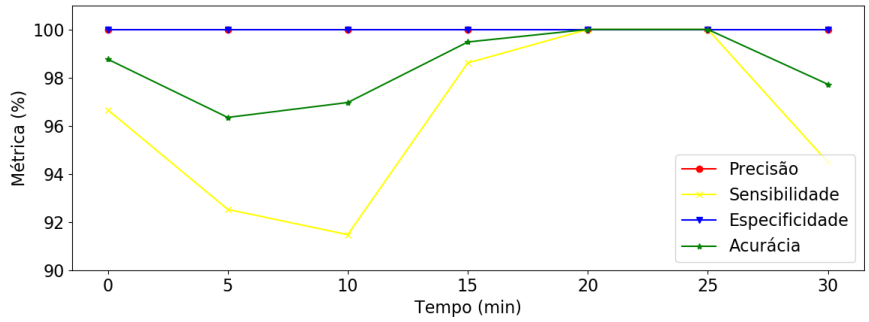
Figura 6. Avaliação dos algoritmos de aprendizado de máquina de acordo com o crescimento do número de mineradores.

A Figura 6 mostra que, à medida que mais mineradores são adicionados à rede,

a precisão e a sensibilidade dos modelos diminuem, mas continuam com resultados aceitáveis, superiores a 75%. Paralelamente, é proposto uma variante do *super learner* [Van der Laan et al., 2007] usando esses dois modelos pelos seguintes motivos: i) é desejado ter um desempenho igual ou superior ao dos algoritmos usados utilizando no *super learner*; ii) é importante o uso de algoritmos de aprendizado de máquina presentes na biblioteca MLlib [Meng et al., 2016] do Spark, pois obtêm-se melhores aproveitamentos da abstração de dados do Spark e a biblioteca não possui suporte para o aprendizado incremental. Assim, utiliza-se uma rede neural como o super modelo, pois suporta o aprendizado incremental. Para testar a eficácia do super aprendizado incremental, utilizou-se o super modelo em dois cenários diferentes. O cenário *a)* com cinco mineradores, e o cenário *b)* com quinze mineradores. Esses cenários representam ambientes com poucos mineradores e muitos mineradores, proporcionalmente ao total de dezesseis *hosts*. Em cada cenário, executou-se o mesmo padrão de 30 minutos de tráfego de rede reproduzido. Esse tempo foi discretizado em fatias de cinco minutos. A cada cinco minutos são calculadas a precisão, a sensibilidade, a especificidade e a precisão do modelo.



(a) Precisão, sensibilidade, especificidade e precisão do super aprendizado incremental com poucos mineradores durante o tempo.



(b) Precisão, sensibilidade, especificidade e precisão do super aprendizado incremental com muitos mineradores durante o tempo.

Figura 7. Avaliação da técnica de super aprendizado incremental durante reproduzindo um tráfego de 30 min.

É possível verificar na Figura 7 que em algumas fatias de tempo o modelo se comporta mal, mas melhora quando aprende com os novos dados. Ainda, em alguns momentos, acontece o desvio de conceito como é possível ver na fatia de tempo de 15 minutos da Figura 7(a) e nas fatias de tempo 5, 10 e 30 minutos da Figura 7(b), mas o super modelo se adapta como é possível identificar nos 20 e 25 minutos na Figura 7(a) e 15

e 20 minutos na Figura 7(b), pois é treinado incrementalmente com as melhores amostras selecionadas de acordo com sua probabilidade de classificação e, então, é garantido que o super modelo aprenderá com dados que possuem alta probabilidade de estarem corretos.

7. Conclusão

A tendência da mineração de criptomoeda é crescer proporcionalmente ao valor monetário das moedas digitais. O mecanismo MineCap foi desenvolvido para identificar e bloquear os fluxos em linha de mineração de criptomoeda em uma rede definida por *software*. Foi desenvolvido um protótipo do mecanismo para avaliação. O artigo propôs a técnica de super aprendizado incremental que combina modelos candidatos como entrada de um super modelo de aprendizado incremental. Os resultados das avaliações mostram que os algoritmos de aprendizado de máquina mais adequados a serem modelos candidatos a fornecerem atributos ao aprendizado incremental foram a Floresta Aleatória e o *Gradient Boosted Tree*. Ambos apresentaram boa capacidade de generalização, com precisão e especificidade de cerca de 100% e sensibilidade de 66,5%. A técnica de super aprendizado incremental obteve bons resultados e demonstrou um correto funcionamento em relação a fatias de tempo com diferentes quantidades de mineradores na rede. A técnica aprende com novos dados, mantendo alto desempenho desde o início da execução e, em alguns casos, melhorando com o tempo.

Referências

Andreoni Lopez, M., Lobato, A. G. P. e Duarte, O. C. M. B. (2016). A performance comparison of open-source stream processing platforms. Em *2016 IEEE Global Communications Conference (GLOBECOM)*, p. 1–6.

Andreoni Lopez, M., Mattos, D. M. F., Duarte, O. C. M. B. e Pujolle, G. (2019). Toward a monitoring and threat detection system based on stream processing as a virtual network function for big data. *Concurrency and Computation: Practice and Experience*, 0(0):1–17.

Bannour, F., Souihi, S. e Mellouk, A. (2018). Distributed SDN control: Survey, taxonomy, and challenges. *IEEE Communications Surveys Tutorials*, 20(1):333–354.

Carbone, P., Ewen, S., Haridi, S., Katsifodimos, A., Markl, V. e Tzoumas, K. (2015). Apache Flink: Unified Stream and Batch Processing in a Single Engine. *Data Engineering*, p. 28–38.

de Oliveira, M. T., Carrara, G. R., Fernandes, N. C., Albuquerque, C. V. N., Carrano, R. C., de Medeiros, D. S. V. e Mattos, D. M. F. (2019). Towards a performance evaluation of private blockchain frameworks using a realistic workload. Em *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, Paris.

Fei-Fei, L., Fergus, R. e Perona, P. (2007). Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding*, 106(1):59 – 70. Special issue on Generative Model Based Vision.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M. e Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):44.

Ingols, K. (2009). Modeling modern network attacks and countermeasures using attack graphs. *Computer Security Applications Conference*.

Konoth, R. K., Vineti, E., Moonsamy, V., Lindorfer, M., Kruegel, C., Bos, H. e Vigna, G. (2018). Minesweeper: An in-depth look into drive-by cryptocurrency mining and its defense. Em *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, p. 1714–1730. ACM.

Luengo, J., Fernández, A., García, S. e Herrera, F. (2011). Addressing data complexity for imbalanced data sets: analysis of smote-based oversampling and evolutionary undersampling. *Soft Computing*, 15(10):1909–1936.

Mattos, D. M. F., Duarte, O. C. M. B. e Pujolle, G. (2016). Reverse update: A consistent policy update scheme for software-defined networking. *IEEE Communications Letters*, 20(5):886–889.

McAfee, A., Brynjolfsson, E., Davenport, T. H., Patil, D. e Barton, D. (2012). Big data: the management revolution. *Harvard business review*, 90(10):60–68.

Medeiros, D. S. V., Cunha Neto, H. N., Andreoni Lopez, M., Magalhães, L. C. S., Silva, E. F., Vieira, A. B., Fernandes, N. C. e Mattos, D. M. F. (2019). Análise de dados em redes sem fio de grande porte: Processamento em fluxo em tempo real, tendências e desafios. *Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC*, 2019:142–195.

Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S. et al. (2016). Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241.

Open Networking Foundation (2012). *OpenFlow Switch Specification Version 1.3.0 (Wire Protocol 0x04)*. The OpenFlow Consortium.

Pietraszek, T. e Tanner, A. (2005). Data mining and machine learning—towards reducing false positives in intrusion detection. *Information security technical report*, 10(3):169–183.

Polikar, R., Upda, L., Upda, S. S. e Honavar, V. (2001). Learn++: an incremental learning algorithm for supervised neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 31(4):497–508.

Porras, P. A. e Valdes, A. (2001). Network surveillance. US Patent 6,321,338.

Tahir, R., Huzaifa, M., Das, A., Ahmad, M., Gunter, C., Zaffar, F., Caesar, M. e Borisov, N. (2017). Mining on someone else's dime: Mitigating covert mining operations in clouds and enterprises. Em *International Symposium on Research in Attacks, Intrusions, and Defenses*, p. 287–310. Springer.

Van der Laan, M. J., Polley, E. C. e Hubbard, A. E. (2007). Super learner. *Statistical applications in genetics and molecular biology*, 6(1).

Wang, S., Minku, L. L., Ghezzi, D., Caltabiano, D., Tino, P. e Yao, X. (2013). Concept drift detection for online class imbalance learning. Em *The 2013 Int. Joint Conference on Neural Networks (IJCNN)*, p. 1–10.