QUIC-Tr4ck: Mitigando Ataques QUIC-Flood usando Planos de Dados Programáveis

Nicolle P. Favero[1], **Ricardo Parizotto**[1], **Alberto E. Schaeffer-Filho**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{npfavero, rparizotto, alberto}@inf.ufrgs.br

Abstract. *The QUIC protocol aims to improve the performance of Web applications, allowing the establishment of multiplexed connections over the UDP protocol. However, with the popularization of QUIC-based services, these have become the target of malicious attacks, mainly Denial of Service (DoS) attacks. Although there are already solutions to mitigate attacks against QUIC that operate on endpoints, solutions based on programmable data planes (PDP) to detect and mitigate attacks against the QUIC protocol are not largely investigated. In this paper, we present QUIC-Tr4ck, a system for tracking QUIC packets using a hybrid approach that combines programmable data planes (using P4 and sketches) with an SDN controller (consolidating snapshots for a more holistic network state analysis). QUIC-Tr4ck allows switches to intercept QUIC connections and identify malicious clients proactively and preemptively, that is, before they exhaust a server's resources.*

Resumo. *O protocolo QUIC tem como propósito melhorar o desempenho de aplicações Web, permitindo o estabelecimento de conexões multiplexadas sobre o protocolo UDP. No entanto, com a popularização de serviços baseados em QUIC, esses passaram a ser alvo de tentativas de ataques maliciosos, principalmente de ataques de negação de serviço (DoS). Apesar de já existirem soluções para mitigação de ataques contra QUIC operando em endpoints, soluções baseadas em planos de dados programáveis (PDP) para detectar e mitigar ataques ao protocolo QUIC são pouco exploradas. Neste artigo, é apresentado o QUIC-Tr4ck, um sistema para rastreamento de pacotes QUIC que combina de forma híbrida estratégias empregadas no plano de dados programável (utilizando a linguagem P4 e sketches) e estratégias executando em um controlador SDN (consolidando snapshots para uma análise holística do estado da rede). QUIC-Tr4ck permite que switches interceptem conexões QUIC e identifiquem clientes maliciosos de maneira proativa e preventiva, isto é, antes que eles exauram os recursos de um servidor.*

1. Introdução

Apesar da consolidação do TCP como protocolo de camada de transporte para aplicações orientadas à conexão, recentemente o protocolo QUIC vem ganhando visibilidade devido a sua eficiência e segurança [Langley et al. 2017]. O protocolo QUIC foi proposto pelo Google para aprimorar o desempenho de aplicações Web baseadas em TCP, permitindo o estabelecimento de diversas conexões multiplexadas entre dois *endpoints*

sobre o protocolo UDP. Atualmente o QUIC é amplamente investigado em trabalhos acadêmicos [Basyoni et al. 2021, Shreedhar et al. 2021] e já começou a ser adotado pela indústria [Mahindra and Guo 2019, Joras and Chi 2020]. O protocolo QUIC é considerado mais seguro que o TCP, pois seu *handshake* para estabelecimento de conexão já inclui o handshake do protocolo TLS; e também é considerado mais eficiente devido ao seu *handshake* levar menos RTTs (*round-trip times*) que o handshake do TCP.

Recentemente, com a popularização de serviços utilizando QUIC, esses passaram a ser alvo de tentativas de ataques maliciosos, principalmente de ataques de negação de serviço (DoS) [Nawrocki et al. 2021, Chatzoglou et al. 2022]. Nesse contexto, um atacante pode explorar características do protocolo QUIC a fim de realizar ataques DoS de diversas formas, e.g., ataques de exaustão de estado e ataques de reflexão. Tais ataques são difíceis de evitar por dois motivos: (1) pelo QUIC ser implementado sobre o protocolo UDP, os cabeçalhos dos pacotes fornecem poucas informações sobre o próprio conteúdo do pacote; (2) pelos pacotes QUIC serem criptografados, o que dificulta manter o controle do estado das conexões, sejam elas legítimas ou maliciosas [Cloudflare 2022]. Com isso, a eficiência e segurança, características valiosas mencionadas anteriormente, se tornam um obstáculo quando se trata de identificar ataques DoS contra QUIC.

Estudos sobre a segurança do QUIC vêm se popularizando nos últimos anos [Lychev et al. 2015, Jager et al. 2015, Fischlin and Günther 2017, Cao et al. 2019, Hiba et al. 2020], os quais fazem análises sobre o funcionamento e a segurança do protocolo. Embora existam soluções que visam mitigar ataques direcionados ao QUIC, muitas delas atuam nos *endpoints*, o que, no entanto, pode deixar os servidores vulneráveis. Isso ocorre porque esses ataques podem sobrecarregar os *buffers* dos servidores e interromper a prestação contínua do serviço. Portanto, há uma necessidade de soluções que possam conter ataques antes que eles comprometam a disponibilidade dos servidores.

Uma solução promissora é utilizar o plano de dados programável (PDP), que permite alterar a funcionalidade dos dispositivos de encaminhamento. Dispositivos do PDP permitem a definição de *parsers* de pacotes, de tabelas de *match+action*, e a execução de operações com estado. Tal programabilidade permite o descarregamento de funções ao plano de dados, podendo reduzir a carga nos servidores e mantendo um uso melhorado de largura de banda entre os switches e servidores. Dessa forma, uma abordagem que poderia ser adotada para prevenir e combater ataques ao QUIC seria a detecção e mitigação proativa de ataques diretamente nos dispositivos de rede, integrando essa funcionalidade ao plano de dados. Isso permitiria identificar e neutralizar ataques ao QUIC antes mesmo que eles afetem os *endhosts*, contribuindo para uma proteção mais eficaz e imediata.

Neste artigo, é apresentado o QUIC-Tr4ck, um sistema para rastrear e identificar ataques de exaustão de estado ao protocolo QUIC. O sistema utiliza uma abordagem híbrida, que combina planos de dados programáveis e um controlador SDN, para rastrear tanto conexões simétricas quanto assimétricas. Em particular, o QUIC-Tr4ck permite que switches interceptem conexões QUIC e identifiquem proativamente clientes maliciosos antes que eles exauram os recursos de um servidor. Os switches do QUIC-Tr4ck extraem informações não criptografadas dos cabeçalhos dos pacotes durante o handshake QUIC, subsequentemente processando e consolidando essas informações em *sketches* implementadas com a linguagem P4. Isso possibilita identificar clientes que iniciam muitas conexões QUIC não estabelecidas e, em seguida, impedir novas tentativas de conexão

desses clientes. A fim de garantir que este mecanismo funcione mesmo em casos de conexões assimétricas, um controlador SDN consolida o estado de *sketches* de diferentes dispositivos de encaminhamento, mantendo uma visão holística e sincronizada do estado da rede e assegurando uma detecção precisa em casos de roteamento assimétrico.

Contribuições. As contribuições do artigo podem ser sumarizadas da seguinte maneira: (1) proposta de um mecanismo para descarregar a identificação e mitigação de ataques de exaustão de estado ao QUIC; (2) desenvolvimento de uma prova de conceito de um mecanismo para permitir o descarregamento em casos de roteamento assimétrico, evitando sobrecarregar o plano de controle.

2. Fundamentação Teórica

Nesta seção, são discutidos conceitos fundamentais sobre o protocolo QUIC e sobre planos de dados programáveis.

2.1. Protocolo QUIC

QUIC [Langley et al. 2017] é um protocolo de camada de transporte proposto visando ser confiável, seguro e rápido, substituindo partes das funcionalidades implementadas pelos protocolos TCP e TLS. Além disso, o QUIC é um protocolo implementado em *userspace*, executando sobre o protocolo UDP, estratégia essa que visa facilitar a sua implantação e manter a portabilidade com *middleboxes* já utilizadas nas redes.

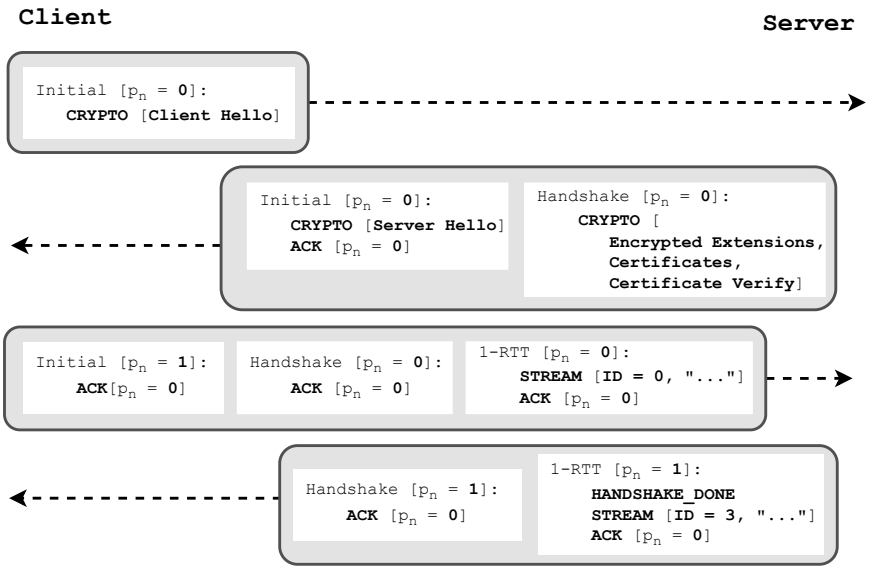


Figura 1. *Handshake* Padrão do QUIC

A Figura 1 ilustra o *handshake* padrão do protocolo QUIC. Via uma única conexão, o QUIC permite multiplexar múltiplos *streams*, usados nas interações de *requests*/*responses* entre o cliente e o servidor [Iyengar and M. Thomson 2021]. Diferentemente da forma como ocorre na pilha de protocolos tradicional, o QUIC realiza de forma combinada os *handshakes* de transporte e de criptografia para minimizar o número de RTTs. Isso resulta em uma operação muito semelhante a do TLS 1.3, com apenas 1-

RTT[1]. Para isso, o cliente envia um pacote do tipo `Initial` com um `Client Hello`. O servidor responde com as seguintes informações: a confirmação de recebimento (ACK) do pacote do cliente, o `Server Hello`, seus certificados, a confirmação de veracidade dos certificados, e um pacote com informações para uma possível reconexão futura por um *handshake* 0-RTT, que reutiliza as credenciais para reestabelecer a conexão novamente. Além disso, na sequência com os pacotes ACK enviados pelo cliente, também são enviados pacotes com o conteúdo da comunicação em si, antes mesmo de o servidor considerar o *handshake* terminado (*frame HANDSHAKE_DONE*).

2.2. Planos de Dados Programáveis

Planos de dados programáveis (PDP) oferecem flexibilidade no desenvolvimento de protocolos e funcionalidades de rede, permitindo a programação e configuração de dispositivos de encaminhamento (e.g., reescrita de cabeçalhos, ações customizadas).

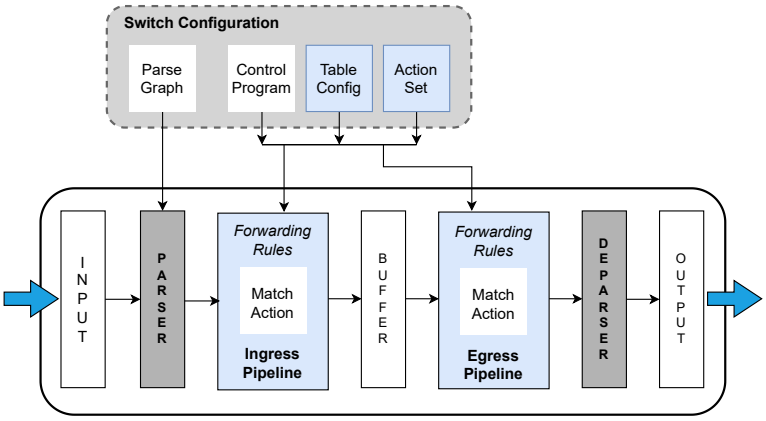


Figura 2. P4 Pipeline (baseada em [Bosshart et al. 2014])

P4 (*Programming Protocol-independent Packet Processors*) [Bosshart et al. 2014] é uma linguagem de programação cujo objetivo é promover controle e flexibilidade sobre o plano de dados de dispositivos de encaminhamento na rede. Por meio de um programa escrito em P4 é possível especificar instruções de como cabeçalhos de pacotes devem ser manipulados e como devem refletir no estado interno do dispositivo. PISA (*Protocol Independent Switch Architecture*) é uma arquitetura para a qual a linguagem P4 pode ser usada. Equipamentos que operam nessa arquitetura processam pacotes em um *pipeline* programável (Figura 2). Dentre os estágios desse *pipeline* estão: o *parser*, que separa os dados dos cabeçalhos dos protocolos usados pelo pacote de acordo com uma máquina de estados; a tabela *match+action*, que recebe os dados coletados pelo parser e realiza o processamento conforme as ações com as quais os dados fazem *match* na tabela; o bloco de controle, que especifica o fluxo de execução entre as tabelas *match+action*; e o *deparser*, que recoloca os cabeçalhos do pacote, com qualquer alteração feita pelo programa P4 e encaminha o pacote para a porta de saída.

3. Motivação: QUIC Flood

Flooding (ou inundação) é um termo recorrente na literatura de ataques DoS/DDoS [Dalmazo et al. 2021, da Silva et al. 2015]. Exemplos de ataques de inundação são o *UDP*

[1]Por mais que o servidor necessite enviar diversos pacotes, o envio de cada um é feito com uma diferença de tempo muito pequena e, portanto, o tempo total do *handshake* é considerado de 1-RTT.

Tabela 1. Ataques DoS de exaustão de estados ao QUIC

	[Nawrocki et al. 2021]	[Iyengar and M. Thomson 2021]	[Chatzoglou et al. 2022]
Descrição	Atacante abre muitas conexões para desperdiçar recursos do servidor.	Atacante abre muitos *streams* em uma conexão para desperdiçar recursos do servidor.	Atacante abre muitas conexões 0-RTT para desperdiçar recursos do servidor.
Características do QUIC	Connections	Streams	0-RTT Connections
Técnicas usadas	Botnet	Botnet	Não disponível
Técnicas de mitigação	Limitar o número de conexões por cliente.	Limitar o número de *streams* por conexão.	Limitar o número de conexões por cliente.

Flood e o *SYN Flood*, sendo o último realizado sobre o *handshake* TCP. Embora o QUIC seja um protocolo relativamente novo, já existem esforços que buscam identificar tipos de ataques contra ele [Chatzoglou et al. 2022, Nawrocki et al. 2021, Cloudflare 2022]. Ataques DoS contra QUIC podem ser melhor categorizados quando baseados em duas das categorias já existentes de ataques DoS: ataques volumétricos e ataques de exaustão de estado (sendo o último o foco deste trabalho).

No QUIC, os ataques de exaustão de estados podem tomar várias formas e ter diferentes objetivos. Por exemplo, o atacante pode atuar como cliente e estabelecer várias conexões (realizando o *handshake* por completo, nesse caso), visando fazer o servidor QUIC se sobrecarregar ao alocar recursos para manter o estado de todas essas conexões [Nawrocki et al. 2021]. O atacante pode usar essa mesma estratégia, porém abrindo muitos *streams* em uma mesma conexão, que é o caso conhecido por *Stream Commitment Attack* em [Iyengar and M. Thomson 2021]. Ambos os ataques são similares ao ataque *TCP SYN Flood*. Outra alternativa de ataque de exaustão de recursos seria o atacante abrir o máximo de conexões 0-RTT possíveis a fim de esgotar os recursos do servidor. Apesar de diferentes vetores de ataque, o objetivo deles é um só: sobrecarregar o servidor a fim de que ele não consiga atender usuários legítimos. Sob essa perspectiva, todos esses ataques recebem o nome de *QUIC Flood Attack*, e são sumarizados na Tabela 1.

4. Trabalhos Relacionados

Nessa sessão, são discutidos trabalhos relacionados que: *(i)* investigam ataques contra o protocolo QUIC; e *(ii)* investigam técnicas de mitigação de ataques DoS no plano de dados programável.

4.1. Ataques Contra QUIC

Em [Chatzoglou et al. 2022] os autores apresentam uma visão geral do atual cenário da segurança do protocolo QUIC, além de uma avaliação do comportamento de diversos servidores QUIC quando sob ataques de diversos tipos, como *QUIC flood*, *Slowloris* e *Version Downgrade*. Os resultados apresentados classificam quais ataques merecem um estudo mais aprofundado e quais não apresentam perigo. Entre os ataques do primeiro grupo está o *QUIC flood*, que é o tópico deste trabalho.

Em [Nawrocki et al. 2021] é apresentada uma categorização de ataques DoS contra QUIC, além de uma análise sobre o tráfego dos pacotes QUIC na Internet. A análise mostra que ataques de exaustão de estado sobre o QUIC são tão frequentes e tão impactantes quanto no TCP, porém com uma duração mais curta. A análise mostra, também,

que servidores QUIC de grandes provedores de conteúdo, como Google e Facebook, estão sob ataques e que esses muitas vezes são parte de ataques multi-vetor.

Em [Gbur and Tschorsch 2021] é feito um estudo sobre o protocolo QUIC no contexto de *firewalls*. O trabalho avalia as possibilidades e os desafios de um *firewall* ao manter o rastreamento de uma conexão QUIC através de uma comparação com um *firewall* de estados TCP. Além disso, o trabalho também traz semelhanças do *firewall* QUIC teórico com um *firewall* de estados UDP, e mostra como as vulnerabilidades do segundo permanecem válidas para o primeiro. Diferentemente, o QUIC-Tr4ck move parte da mitigação de ataques para um switch programável, evitando a necessidade de processar pacotes em uma solução baseada apenas em servidores. No entanto, a dificuldade de se lidar com pacotes QUIC encriptados ainda permanece.

4.2. DDoS no Plano de Dados Programável

O uso de *switches* programáveis para detectar e mitigar ataques DoS que utilizem protocolos mais tradicionais, como o TCP, foi investigado por trabalhos relacionados, em particular usando switches para identificar DoS caracterizado por grandes volumes de tráfego no plano de dados. Estes incluem trabalhos que utilizam técnicas de *machine learning*, como árvores de decisão [Xavier et al. 2021, Coelho and Schaeffer-Filho 2022] ou *clustering* [Alcoz et al. 2022], para identificar um ataque.

Em [Scholz et al. 2020], é apresentada uma solução em plano de dados programáveis para detecção de ataques DoS *SYN flood*, através da implementação de uma *"SYN proxy"* em P4. A solução apresentada faz o *parser* do protocolo TCP e calcula as hashes criptográficas no próprio plano de dados, permitindo que os SYN *cookies* sejam calculados no próprio switch, ao invés de no servidor. Os cookies no switch servem para autenticar as novas conexões e impedir clientes maliciosos de abrirem diversas conexões, mas nunca estabelecê-las efetivamente, o que diminui o impacto nos servidores TCP. Diferentemente, QUIC-Tr4ck mitiga o ataque através de um *threshold* e de análise dos pacotes que passam, enquanto neste trabalho os *cookies* fazem parte do *handshake*.

SACK3 [Tavares and Ferreto 2019] é um sistema que permite identificar ataques *SYN flood* utilizando switches programáveis. O SACK3 mantém informações de conexões em *switches* usando *sketches*. De maneira similar ao SACK3, nosso trabalho também utiliza sketches para salvar estado no *switch*, porém, o foco do QUIC-Tr4ck é em ataques de exaustão de estado contra o protocolo QUIC e não contra TCP. Isso exige interceptar e interpretar informações específicas deste protocolo no próprio *switch* e definir mecanismos para lidar com as informações criptografadas dos pacotes QUIC.

5. QUIC-Tr4ck

O QUIC-Tr4ck é um sistema proposto para mitigar ataques que exploram vulnerabilidades do protocolo QUIC. O sistema utiliza uma abordagem híbrida, que combina o plano de dados programável com um controlador SDN para auxiliar na prevenção de ataques DoS por exaustão de estado ao QUIC.

5.1. Visão geral

A visão geral do QUIC-Tr4ck é apresentada na Figura 3. O núcleo do QUIC-Tr4ck está em *switches* programáveis que ficam localizados entre os clientes e servidores QUIC.

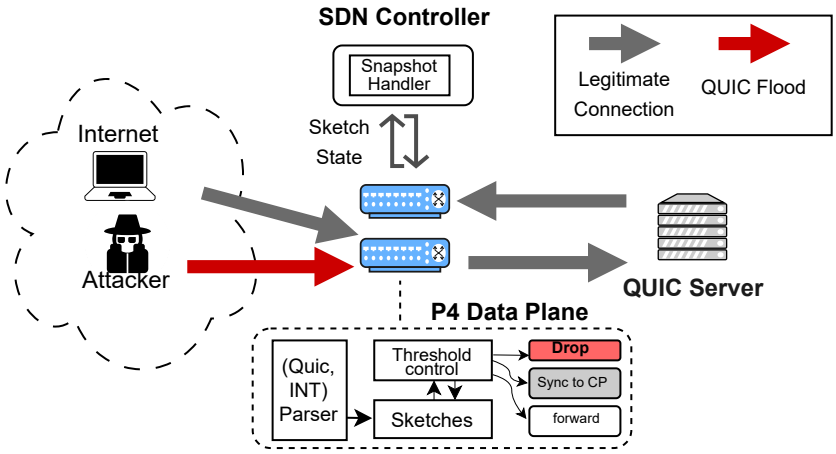


Figura 3. Visão geral do QUIC-Tr4ck

Os switches interceptam tentativas de conexões, extraem e armazenam informações de estado sobre o cliente por meio de estruturas de dados chamadas de *sketches*. As informações coletadas pelos *switches* são submetidas a um processo onde são comparadas com *thresholds*. Isso permite identificar clientes maliciosos e, posteriormente, descartar novas tentativas destes clientes. A solução foi concebida para operar de forma combinada em *duas fases*: além da operação realizada no próprio plano de dados, que permite a detecção de ataques com rapidez, na taxa de linha, o QUICK-Tr4ck também emprega um modo de operação envolvendo o plano de controle. Em particular, esse modo de operação permite sincronizar as informações de estado provenientes de um conjunto de switches, de modo a obter uma visão holística do estado da rede. A sincronização de informações acontece utilizando um mecanismo de telemetria, que sincroniza apenas informações de fluxos críticos. Apesar dessa segunda fase não atuar na mesma escala de tempo que a primeira, essa abordagem híbrida tem por objetivo consolidar as informações de múltiplos switches, permitindo identificações mais precisas em situações onde conexões possuem requisições e respostas que seguem por caminhos distintos.

5.2. Extraindo informações do cabeçalho QUIC

Para identificar quando uma conexão é estabelecida, é necessário permitir que o switch consiga fazer a análise de cabeçalhos QUIC, em particular definindo um *parser* para o cabeçalho dos pacotes `Initial` e `Handshake`. O pacote `Initial` é utilizado pelo cliente para solicitar o estabelecimento de nova conexão, enquanto o pacote `Handshake` carrega as informações que serão utilizadas para a criptografia dos pacotes. Como ambos são pacotes QUIC de cabeçalho longo, foi definida uma mesma estrutura *header* agrupando os valores de tamanho fixo comuns a ambos os tipos de pacotes. No futuro pode-se estender a estrutura para pacotes `Retry` e `0-RTT` que também fazem parte do grupo de cabeçalhos longos. A definição da estrutura *header* é apresentada na Figura 4.

Para ter acesso aos primeiros campos de informação do cabeçalho do pacote QUIC, o QUIC-Tr4ck faz a análise do cabeçalho `quic_long_t`. Com isso, é possível identificar se o pacote é um pacote de cabeçalho longo (`headerForm == 1`) ou de cabeçalho curto/1-RTT (`headerForm == 0`) e, se ele for longo, de qual tipo ele é (`Initial`, `Handshake`, `Retry`, `0-RTT`). Essas informações serão utilizadas posteriormente para o registro da quantidade de *handshakes* iniciados por um mesmo cliente.

```
header quic_long_t {
    bit<1> headerForm;
    bit<1> fixedBit;
    bit<2> longPacketType;
    bit<2> reservedBits;
    bit<2> packetNumberLength;
    bit<32> version;
}
```

Figura 4. QUIC long header

5.3. Armazenando informações em Sketches

Após definir como extrair as informações necessárias, nesta seção, são apresentadas maneiras de armazenar essas informações em variáveis de estados do switch. Isso é desafiador, uma vez que switches programáveis possuem memória limitada (cerca de 10MB) de modo que, geralmente, são utilizadas estruturas de dados compactas, como *hash maps* ou *bloom filters*. No entanto, ambas são estruturas estatísticas e possuem probabilidade de colisão elevada. Por isso, uma alternativa é a utilização de *sketches*.

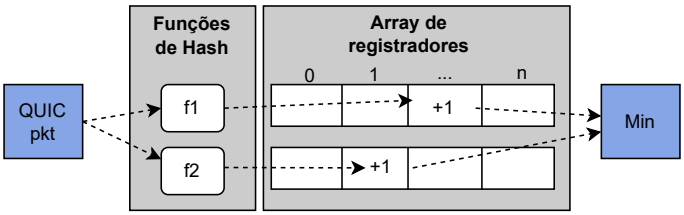


Figura 5. *Sketches* no plano de dados

A Figura 5 (inspirada por [Namkung et al. 2022]) ilustra como *sketches* são usados para manter informações de conexões [Charikar et al. 2002]. Ao chegar um pacote de uma conexão QUIC, informações do seu cabeçalho são utilizadas como entrada para duas funções *hash*. Com o objetivo de reduzir a probabilidade de colisões, a entrada de cada função é composta por informações tanto do cabeçalho UDP, quanto do cabeçalho IP:

```
UDP = <hdr.udp.src_addr, hdr.udp.dst_addr>

IP = <hdr.ipv4.src_addr, hdr.ipv4.dst_addr>
```

Após computar as funções de *hash*, o QUIC-Tr4ck atribui o resultado da computação de cada função para um conjunto de registradores diferentes. São utilizados dois *arrays* de registradores, onde cada um é responsável por armazenar informações mapeadas por uma das funções de *hash*. Utilizar arrays distintos é uma característica típica de *sketches*, que tem por objetivo evitar colisões entre múltiplas funções, diferentemente do que acontece com estruturas como *bloom filters*.

Dado o *handshake* padrão do QUIC ilustrado na Figura 1, quando um pacote `Initial` é recebido por um cliente, os valores das funções de *hash* são computados e é somamo um ao registrador correspondente. Caso contrário, se for um pacote de

`Handshake`, é decrementado um do registrador, pois isso significa que o cliente já cumpriu sua parte no *handshake*. Portanto, uma *sketch* é usada para registrar quantas tentativas de conexões foram realizadas, mas não foram estabelecidas. Esse valor é usado para impedir que clientes que tentaram abrir um número excessivo de conexões, mas não as estabeleceram, possam fazer novas conexões QUIC.

5.4. Sincronizando múltiplos switches

Para o correto funcionamento do sistema, é necessário que os switches observem ambas as direções de uma conexão QUIC. Isso ocorre pois, em casos de roteamento assimétrico, a resposta de um pacote pode passar por um caminho diferente [Chen et al. 2020].

Alternativas. Uma solução para resolver este problema de sincronismo é realizar capturas do estado das *sketches* e unir suas informações. Existem duas opções para realizar essa captura [Zeno et al. 2022]: (1) utilizar a interface PCIe entre o switch e o controlador e ler as informações dos registradores que armazenam as *sketches*; ou (2) enviar as informações pelo próprio plano de dados usando abordagens de telemetria. Capturar o conteúdo completo de uma *sketch* pode resultar em atrasos na leitura no caso da PCIe, ou requerer primitivas de recirculação, o que acarreta perda de desempenho no processamento de pacotes. Como alternativa, foi adotada uma solução baseada em telemetria, inspirada em [Castanheira et al. 2019], porém aplicando-se otimizações onde apenas é sincronizado o estado de fluxos QUIC considerados perigosos, o que evita atrasos relacionados à leitura de toda a sketch.

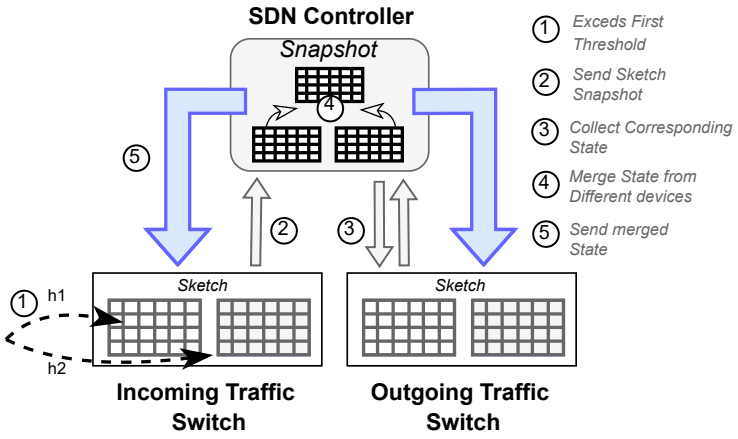


Figura 6. Sincronizando *sketches* com auxílio do plano de controle

A Figura 6 ilustra como o QUIC-Tr4ck soluciona o problema de assincronismo nas *sketches* dos *switches*. O QUIC-Tr4ck utiliza dois *thresholds* no plano de dados: o primeiro *threshold* é um aviso para sincronização, enquanto o segundo *threshold* é o limite máximo de conexões que um cliente pode iniciar. Quando uma conexão QUIC ultrapassa o primeiro *threshold*, o *switch* envia proativamente capturas do estado correspondente para um módulo no plano de controle. A premissa é de que o controlador já tem conhecimento das rotas e, após receber o aviso, coleta o estado correspondente das *sketches* dos caminhos assimétricos[2]. Após coletar o estado das *sketches*, o controlador as agrega em

[2]Não é necessário sincronização em caso de roteamento simétrico, uma vez que nesses casos o switch é capaz de realizar detecções de forma *stand-alone* e sem o auxílio do plano de controle.

uma captura global, combinando as informações de ambos os caminhos. Após unir as informações, o estado é enviado de volta para os *switches*, que adotam os novos valores computados em suas respectivas *sketches*. Caso a conexão cruze um segundo *threshold*, a origem é considerada maliciosa e novas tentativas de estabelecimento de conexão deste cliente serão impedidas. Esse mecanismo permite que os *switches* calculem os *thresholds* e identifiquem ameaças com maior precisão do que de maneira centralizada, e permite evitar a sobrecarga de realizar uma captura de toda a estrutura de dados.

Apesar de o protótipo atual do QUIC-Tr4ck ser baseado em uma noção simples de *threshold* do número de conexões QUIC não completadas para identificar comportamentos abusivos relacionados à exaustão de recursos, mecanismos mais sofisticados podem ser considerados em trabalhos futuros, incluindo: *(i)* utilização de técnicas baseadas em aprendizado de máquina para redimensionamento do *threshold*; *(ii)* análise de cabeçalhos e *features* adicionais para acompanhar etapas subsequentes do *handshake* do QUIC; e *(iii)* definição de mecanismos de quarentena e listas de suspeitos que estarão sujeitos a filtragem parcial dos pacotes até que a legitimidade seja averiguada.

6. Avaliação Preliminar

Nesta seção, é descrita a implementação do protótipo e um conjunto de experimentos para uma avaliação preliminar do QUIC-Tr4ck. Os experimentos focam em medir o consumo de memória de um servidor QUIC durante uma simulação de ataque QUIC Flood quando utilizamos o QUIC-Tr4ck. Também é realizada uma análise do funcionamento do QUIC-Tr4ck operando para conexões QUIC em casos de roteamento assimétrico.

6.1. Ambiente de Experimentação

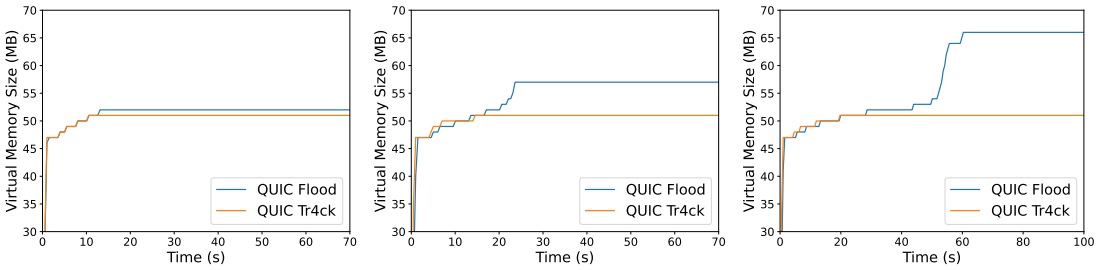
O sistema foi prototipado utilizando o emulador BMv2 na linguagem P4-16. Os experimentos foram conduzidos em uma VM com 2GB de RAM com 2 cores. As cargas de conexões QUIC foram geradas com a biblioteca Python `aioquic`[3]. Por fim, foi usada a biblioteca `psutil` para realizar a medição de recursos utilizados, como memória e CPU.

6.2. Metodologia

Inicialmente foi configurado um ambiente com um *host* atuando como cliente e outro como servidor. Apenas um switch é considerado nesse experimento, sem a necessidade do mecanismo de sincronização para conexões assimétricas. O *switch* foi ajustado para permitir um limite de no máximo 10 conexões não concluídas. Além disso, a configuração do cliente foi feita de modo a executar repetidas tentativas de estabelecimento de conexões QUIC. Contudo, de forma intencional, foram descartados os pacotes `Handshake` antes da entrega, impedindo a conclusão da conexão e simulando um ataque com o objetivo de explorar os recursos do servidor.

Com o objetivo de avaliar o potencial do QUIC-Tr4ck, foi configurada a frequência de tentativas maliciosas de estabelecimento de conexões por meio de *bursts* (rajadas de conexões). Foram realizados experimentos com 8 *bursts*, disparados em um intervalo de 1 segundo, variando o número de requisições por *burst* em cada experimento (4, 8 e 16). Em cada requisição, são realizadas 9 tentativas de envio de pacotes `Initial`. Adicionalmente, foi incluída uma requisição extra no final do experimento, visando melhorar o desempenho e obter uma saída mais organizada.

[3]https://aioquic.readthedocs.io



(a) 4 requisições por *burst*

(b) 8 requisições por *burst*

(c) 16 requisições por *burst*

Figura 7. Memória Virtual (em MB) utilizada pelo servidor QUIC

6.3. Consumo de Recursos

A Figura 7 apresenta a quantidade de memória virtual utilizada pelo servidor QUIC durante a simulação dos ataques. Ao aumentar o número de requisições por *burst* ou o número de *bursts*, a quantidade de conexões em aberto cresce de forma considerável, assim como o uso da memória do servidor. Porém, nos três cenários com quantidades diferentes de bursts por segundo, o QUIC-Tr4ck reduz a quantidade de memória virtual gasta pelo servidor quando sob ataque. Este resultado é satisfatório, mostrando o potencial de usar *switches* programáveis para mitigar este tipo de ataque, mesmo com um mecanismo de limite simples.

6.4. Avaliação Funcional

Esta seção apresenta um caso de estudo visando avaliar a funcionalidade do QUIC-Tr4ck. O ambiente de experimentação considera o emulador BMv2 em uma VM com dois 2GB de RAM. É considerado um cenário com tráfego assimétrico, apresentado na Figura 8. Foram executados quatro experimentos diferentes, onde cada experimento dispara *bursts* de tráfego de clientes legítimos (usando o `aioquic`) até um servidor, percorrendo um caminho de ida diferente do caminho de volta. Cada experimento gera *bursts* contendo 5, 10, 15 e 20 conexões legítimas, respectivamente. Os *switches* foram configurados com o primeiro *threshold* em 5 e o segundo em 10. Esses valores foram utilizados pois o objetivo é analisar a funcionalidade do sistema, e não o desempenho em cenários de estresse.

Cada experimento foi executado com duas abordagens: o QuickTr4ck e o *baseline*. O QuickTr4ck utiliza o controlador que realiza a sincronização, enquanto a abordagem *baseline* implementa apenas limites individuais em cada *switch*, sem empregar qualquer mecanismo de coordenação entre eles. Para avaliar os resultados, foi medida a quantidade de fluxos legítimos completos, apresentada na Figura 9. Como é possível perceber, sem o uso do QUIC-Tr4ck, apenas 10 fluxos puderam ser estabelecidos, mesmo que tenham ocorrido 20 tentativas legítimas. Por outro lado, ao utilizar QUIC-Tr4ck, todas as tentativas de conexão foram estabelecidas. Isso se deve ao fato de que, ao disparar-se o número de conexões que ultrapassa o primeiro *threshold*, o sistema sincroniza as posições das *sketches* dos *switches* de ambos os caminhos, assegurando que pacotes de uma conexão legítima não sejam erroneamente marcados como um potencial ataque. Por outro lado, o mecanismo sem qualquer coordenação descarta pacotes de tentativas de conexões que ultrapassam o *threshold*, menos que pertençam a conexões legítimas.

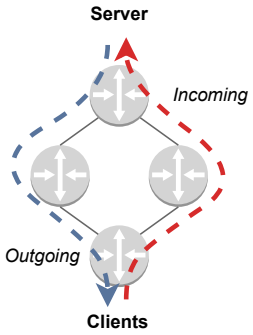


Figura 8. Topologia de teste

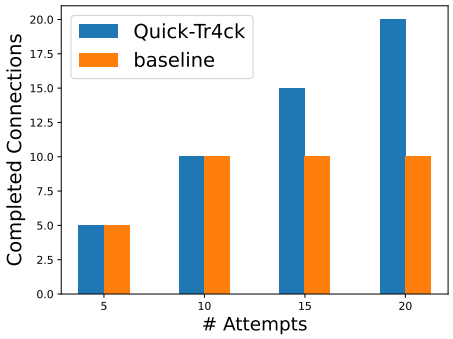


Figura 9. Análise funcional

7. Conclusões

Neste artigo foi apresentado o QUIC-Tr4ck, um sistema para rastrear conexões QUIC que usa uma abordagem híbrida, combinando o plano de dados programável com o plano de controle SDN. O sistema utiliza *sketches* para armazenar informações de conexões QUIC em um *switch* programável, de modo a identificar clientes maliciosos de maneira proativa. As sketches do plano de dados são sincronizadas com o auxílio do plano de controle quando necessário, permitindo que a detecção seja precisa mesmo em casos de roteamento assimétrico. Os resultados mostram que é possível reduzir o impacto que ataques QUIC-Flood geram aos servidores QUIC utilizando nosso sistema.

Como trabalhos futuros, uma possibilidade é a investigação do desempenho do sistema em hardware e *workloads* mais realísticos. Também serão estudadas abordagens de mitigação que vão além da utilização do limiar (*threshold*), abrangendo métodos para a identificação de fluxos suspeitos através de aprendizado de máquina. Pretende-se também aprofundar a investigação de estruturas de dados mais eficientes para o armazenamento das informações de estado nos *switches*. Além disso, outra possibilidade é investigar técnicas para posicionar (e.g., [Xu et al. 2023]) o mecanismo de detecção conforme os recursos disponíveis nos *switches* e interfaces de rede.

Agradecimentos

Esse trabalho foi financiado em parte pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001, CNPq (#311276/2021-0) e FAPESP (#2020/05152-7 - PROFISSA).

Referências

Alcoz, A. G., Strohmeier, M., Lenders, V., and Vanbever, L. (2022). Aggregate-based congestion control for pulse-wave ddos defense. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 693–706, New York, NY, USA. Association for Computing Machinery.

Basyoni, L., Erbad, A., Alsabah, M., Fetais, N., Mohamed, A., and Guizani, M. (2021). Quictor: Enhancing tor for real-time communication using quic transport protocol. *IEEE Access*, 9:28769–28784.

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.

Cao, X., Zhao, S., and Zhang, Y. (2019). 0-rtt attack and defense of quic protocol. In *2019 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6.

Castanheira, L., Parizotto, R., and Schaeffer-Filho, A. E. (2019). Flowstalker: Comprehensive traffic flow monitoring on the data plane using p4. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pages 1–6.

Charikar, M., Chen, K., and Farach-Colton, M. (2002). Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer.

Chatzoglou, E., Kouliaridis, V., Karopoulos, G., and Kambourakis, G. (2022). Revisiting quic attacks: A comprehensive review on quic security and a hands-on study.

Chen, X., Kim, H., Aman, J. M., Chang, W., Lee, M., and Rexford, J. (2020). Measuring tcp round-trip time in the data plane. In *Proceedings of the Workshop on Secure Programmable Network Infrastructure*, pages 35–41.

Cloudflare (2022). What is a QUIC flood DDoS attack? | QUIC and UDP floods. Cloudflare Website. Accessed in: December 2022, https://www.cloudflare.com/learning/ddos/what-is-a-quic-flood/.

Coelho, B. and Schaeffer-Filho, A. (2022). Backorders: using random forests to detect ddos attacks in programmable data planes. In *Proceedings of the 5th International Workshop on P4 in Europe*, pages 1–7.

da Silva, A. S., Smith, P., Mauthe, A., and Schaeffer-Filho, A. (2015). Resilience support in software-defined networking: A survey. *Computer Networks*, 92:189–207.

Dalmazo, B. L., Marques, J. A., Costa, L. R., Bonfim, M. S., Carvalho, R. N., da Silva, A. S., Fernandes, S., Bordim, J. L., Alchieri, E., Schaeffer-Filho, A., Paschoal Gaspary, L., and Cordeiro, W. (2021). A systematic review on distributed denial of service attack defense mechanisms in programmable networks. *International Journal of Network Management*, 31(6):e2163.

Fischlin, M. and Günther, F. (2017). Replay attacks on zero round-trip time: The case of the tls 1.3 handshake candidates. In *2017 IEEE European Symposium on Security and Privacy (EuroSP)*, pages 60–75.

Gbur, K. and Tschorsch, F. (2021). A quic(k) way through your firewall?

Hiba, O., Leibowitz, H., and Herzberg, A. (2020). Quicr: Quic resiliency to bw-dos attacks.

Iyengar, J., E. and M. Thomson, E. (2021). Quic: A udp-based multiplexed and secure transport. RFC 9000. https://www.rfc-editor.org/info/rfc9000.

Jager, T., Schwenk, J., and Somorovsky, J. (2015). On the security of tls 1.3 and quic against weaknesses in pkcs1 v1.5 encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 1185–1196, New York, NY, USA. Association for Computing Machinery.

Joras, M. and Chi, Y. (2020). How Facebook is bringing QUIC to billions. Engineering at Meta. https://engineering.fb.com/2020/10/21/networking-traffic/how-facebook-is-bringing-quic-to-billions/.

Langley, A., Riddoch, A., Wilk, A., Vicente, A., Krasic, C., Zhang, D., Yang, F., Kouranov, F., Swett, I., Iyengar, J., Bailey, J., Dorfman, J., Roskind, J., Kulik, J., Westin, P., Tenneti, R., Shade, R., Hamilton, R., Vasiliev, V., Chang, W.-T., and Shi, Z. (2017). The quic transport protocol: Design and internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'17)*, SIGCOMM '17, page 183–196, New York, NY, USA. Association for Computing Machinery.

Lychev, R., Jero, S., Boldyreva, A., and Nita-Rotaru, C. (2015). How secure and quick is quic? provable security and performance analyses. In *2015 IEEE Symposium on Security and Privacy*, pages 214–231.

Mahindra, R. and Guo, E. (2019). Employing QUIC protocol to optimize Uber's app performance. Uber Blog. https://www.uber.com/en-TT/blog/employing-quic-protocol/.

Namkung, H., Liu, Z., Kim, D., Sekar, V., and Steenkiste, P. (2022). {SketchLib}: Enabling efficient sketch-based monitoring on programmable switches. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 743–759.

Nawrocki, M., Hiesgen, R., Schmidt, T. C., and Wählisch, M. (2021). Quicsand: Quantifying quic reconnaissance scans and dos flooding events. In *Proceedings of the 21st ACM Internet Measurement Conference*, IMC '21, page 283–291, New York, NY, USA. Association for Computing Machinery.

Scholz, D., Gallenmüller, S., Stubbe, H., and Carle, G. (2020). Syn flood defense in programmable data planes. In *Proceedings of the 3rd P4 Workshop in Europe*, EuroP4'20, page 13–20, New York, NY, USA. Association for Computing Machinery.

Shreedhar, T., Panda, R., Podanev, S., and Bajpai, V. (2021). Evaluating quic performance over web, cloud storage and video workloads. *IEEE Transactions on Network and Service Management*.

Tavares, K. and Ferreto, T. (2019). Ddos on sketch: Spoofed ddos attack defense with programmable data planes using sketches in sdn. In *Anais do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 805–819, Porto Alegre, RS, Brasil. SBC.

Xavier, B. M., Guimarães, R. S., Comarela, G., and Martinello, M. (2021). Programmable switches for in-networking classification. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE.

Xu, W., Zhang, Z., Feng, Y., Song, H., Chen, Z., Wu, W., Liu, G., Zhang, Y., Liu, S., Tian, Z., et al. (2023). Clickinc: In-network computing as a service in heterogeneous programmable data-center networks. *arXiv preprint arXiv:2307.11359*.

Zeno, L., Ports, D. R., Nelson, J., Kim, D., Landau-Feibish, S., Keidar, I., Rinberg, A., Rashelbach, A., De-Paula, I., and Silberstein, M. (2022). {SwiSh}: Distributed shared state abstractions for programmable switches. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 171–191.