QUICsand: Quantifying QUIC Reconnaissance Scans and DoS Flooding Events

Marcin Nawrocki, Raphael Hiesgen, Thomas C. Schmidt, Matthias Wählisch

{marcin.nawrocki, m.waehlisch}@fu-berlin.de
{raphael.hiesgen, t.schmidt}@haw-hamburg.de

Freie Universität Berlin

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

In a nutshell

Is QUIC used for DoS attacks?

Yes.

Network telescopes allow us to observe these attacks.

QUIC: New protocol, well-known foundations.

UDP

By implementation, based on UDP.

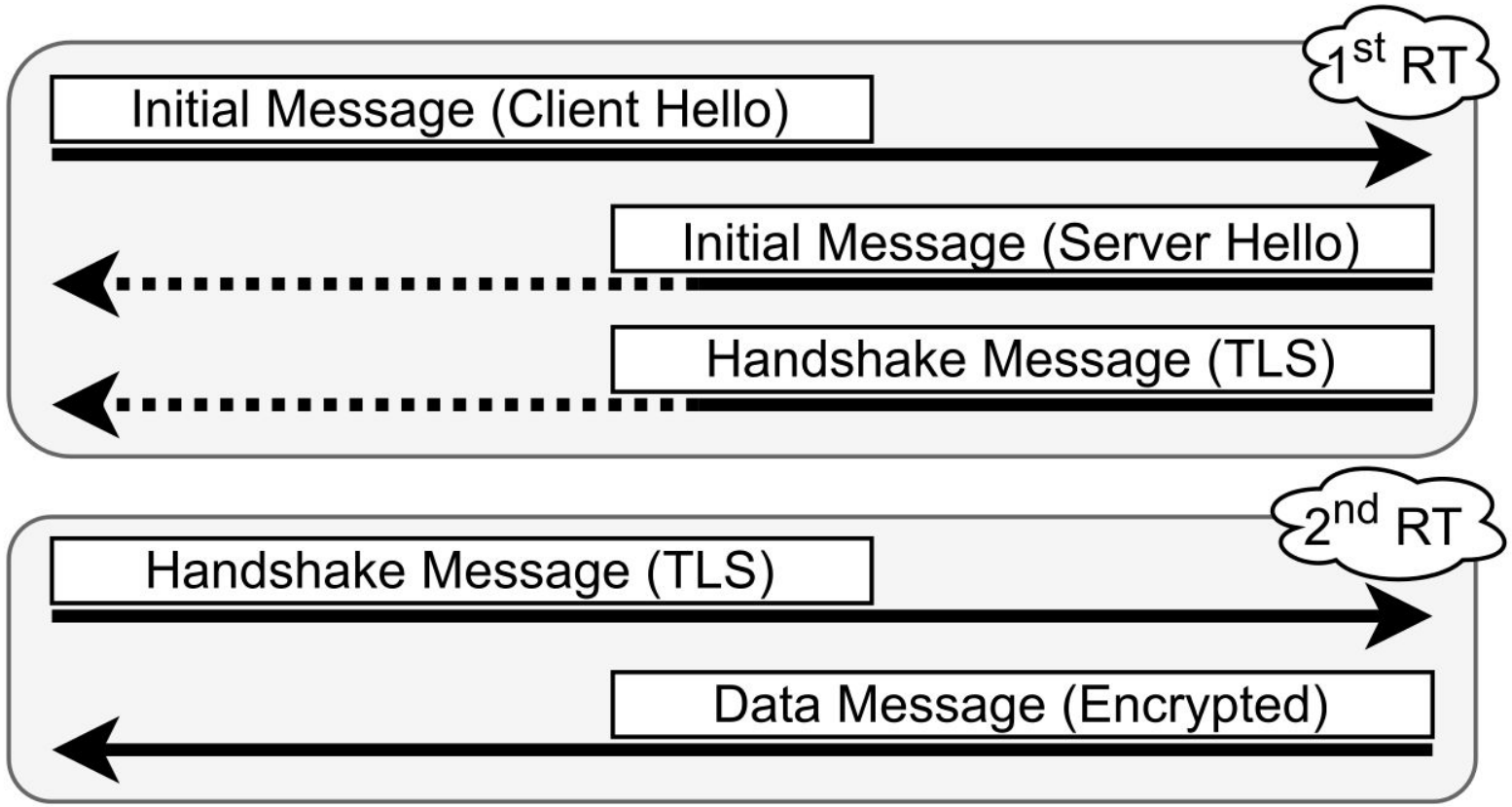
Prevents ossification by middleboxes.

TCP

By design, akin to TCP.

Connection-oriented, base for HTTP/3.

A *typical* QUIC handshake (1-RTT)

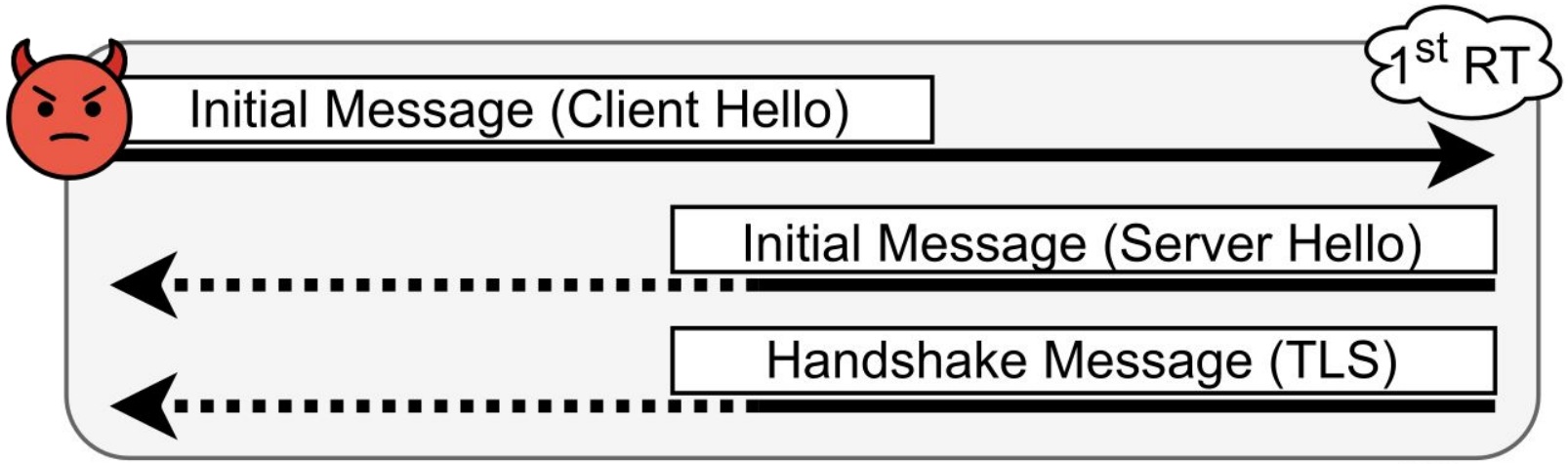


Problem?

During the first round-trip, the server responds to an **unverified** source.

Randomly spoofed QUIC INITIAL floods

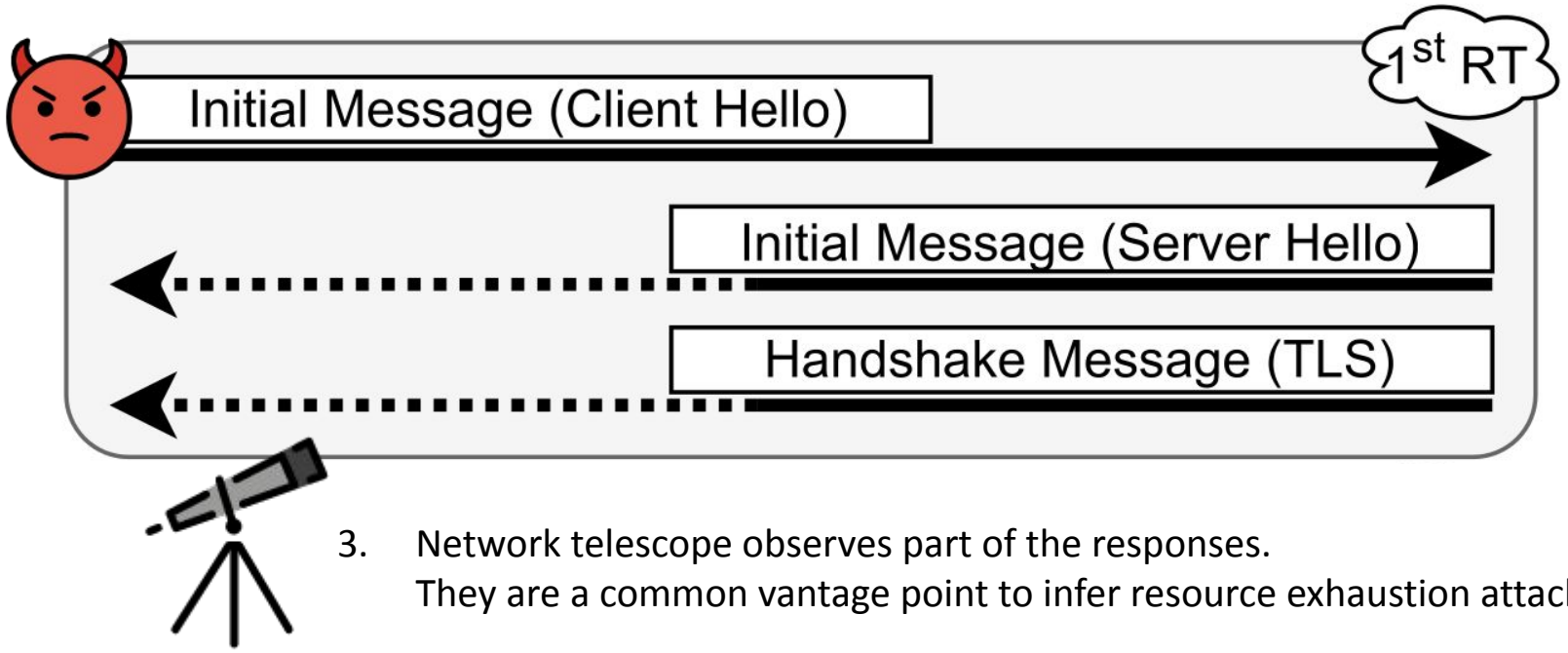
1. Attacker randomly spoofs IP addresses



2. Server reserves connection context & cryptographic computations

Randomly spoofed QUIC INITIAL floods

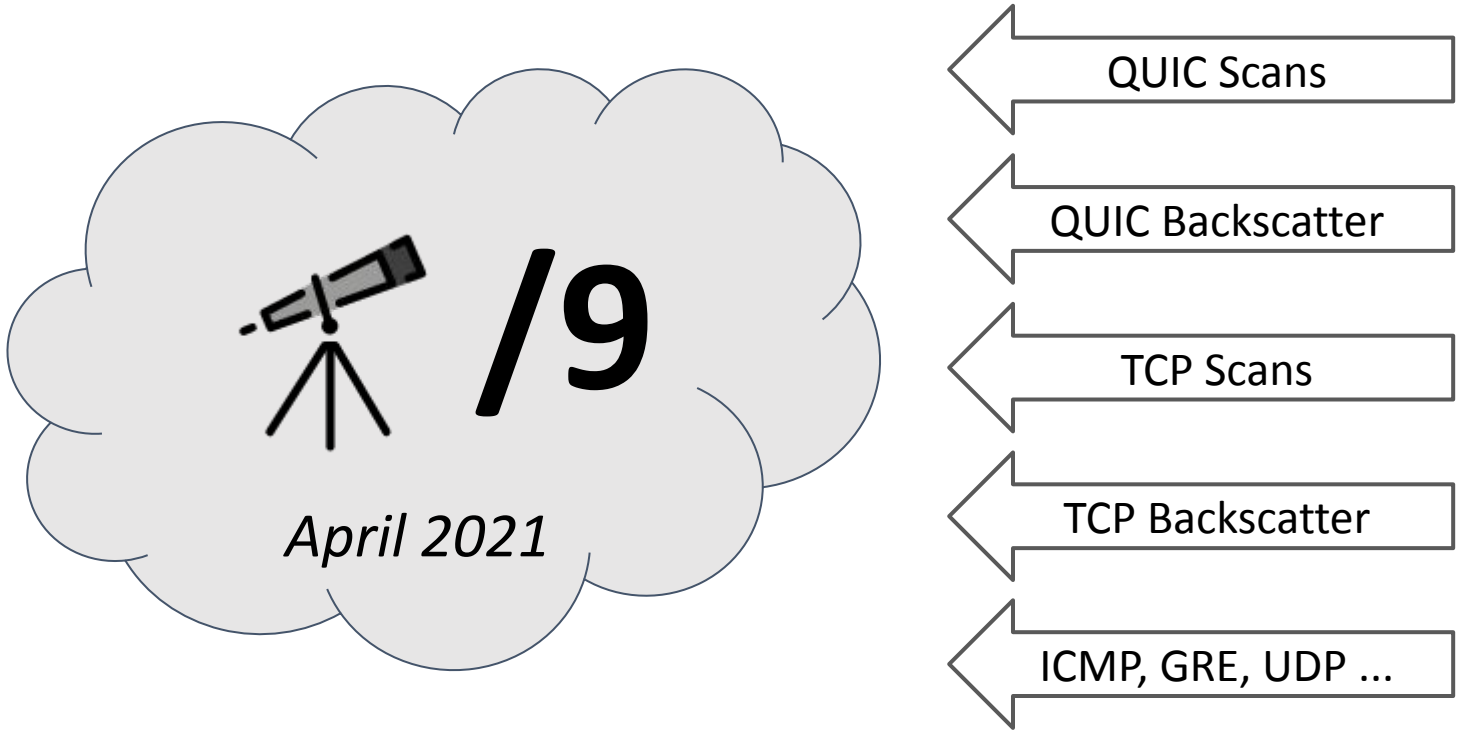
1. Attacker randomly spoofs IP addresses



2. Server reserves connection context & cryptographic computations

3. Network telescope observes part of the responses.
   They are a common vantage point to infer resource exhaustion attacks.

Setup: Passive traffic capture@UCSD telescope.



How to detect QUIC backscatter@telescope?

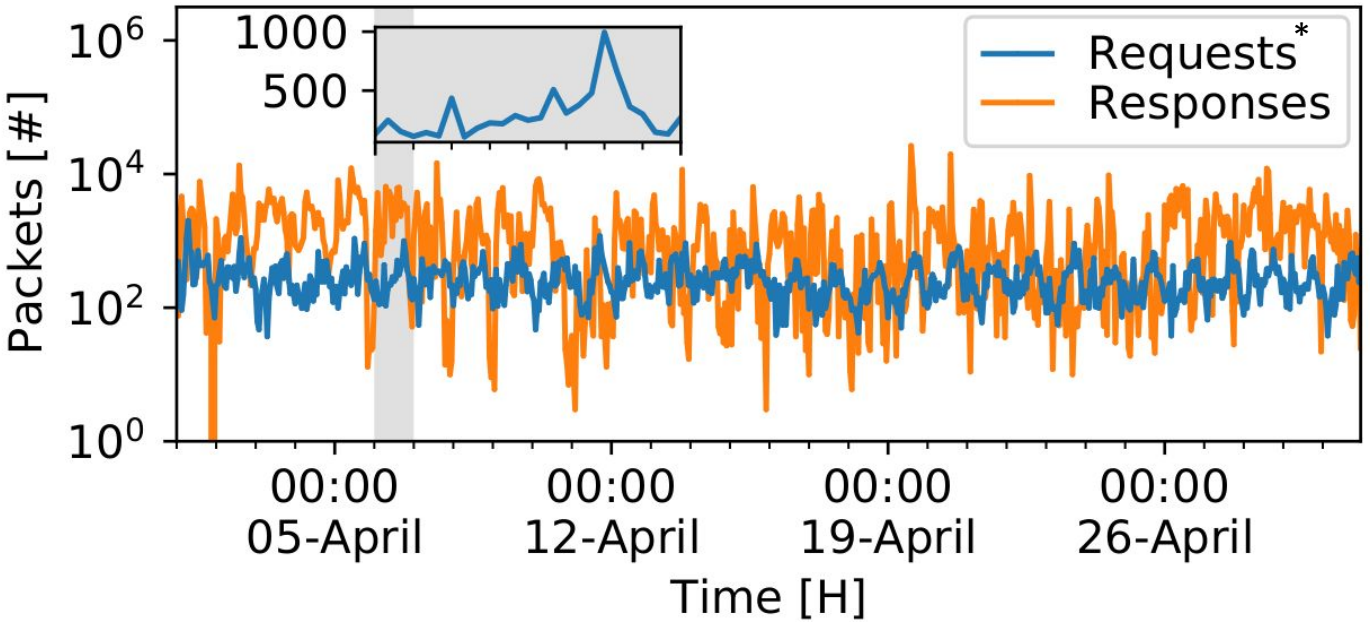
We use Wireshark to detect QUIC traffic based on the payload (DPI), *not only by ports*.

We detected 92M QUIC packets.

Then, we identify *scans* and *backscatter*:

a) QUIC requests are part of scanning activities.
b) QUIC responses are backscatter due to QUIC floods.

Erratic response traffic hints at DoS events



*sanitized

How to infer DoS attacks?

We apply a common* method and thresholds to identify attacks.

1. Group packets from the same source into sessions:
   idle timeout == 5 minutes

2. Response (backscatter) sessions are an attack if:
   > 25 packets, > 60 seconds, and maximum PPS > 0.5

* Moore, David, et al. "Inferring internet denial-of-service activity." *ACM Transactions on Computer Systems (TOCS)* 24.2 (2006): 115-139.

How many attacks did you find?

2905

QUIC floods in April 2021.

How many attacks did you find?

2905

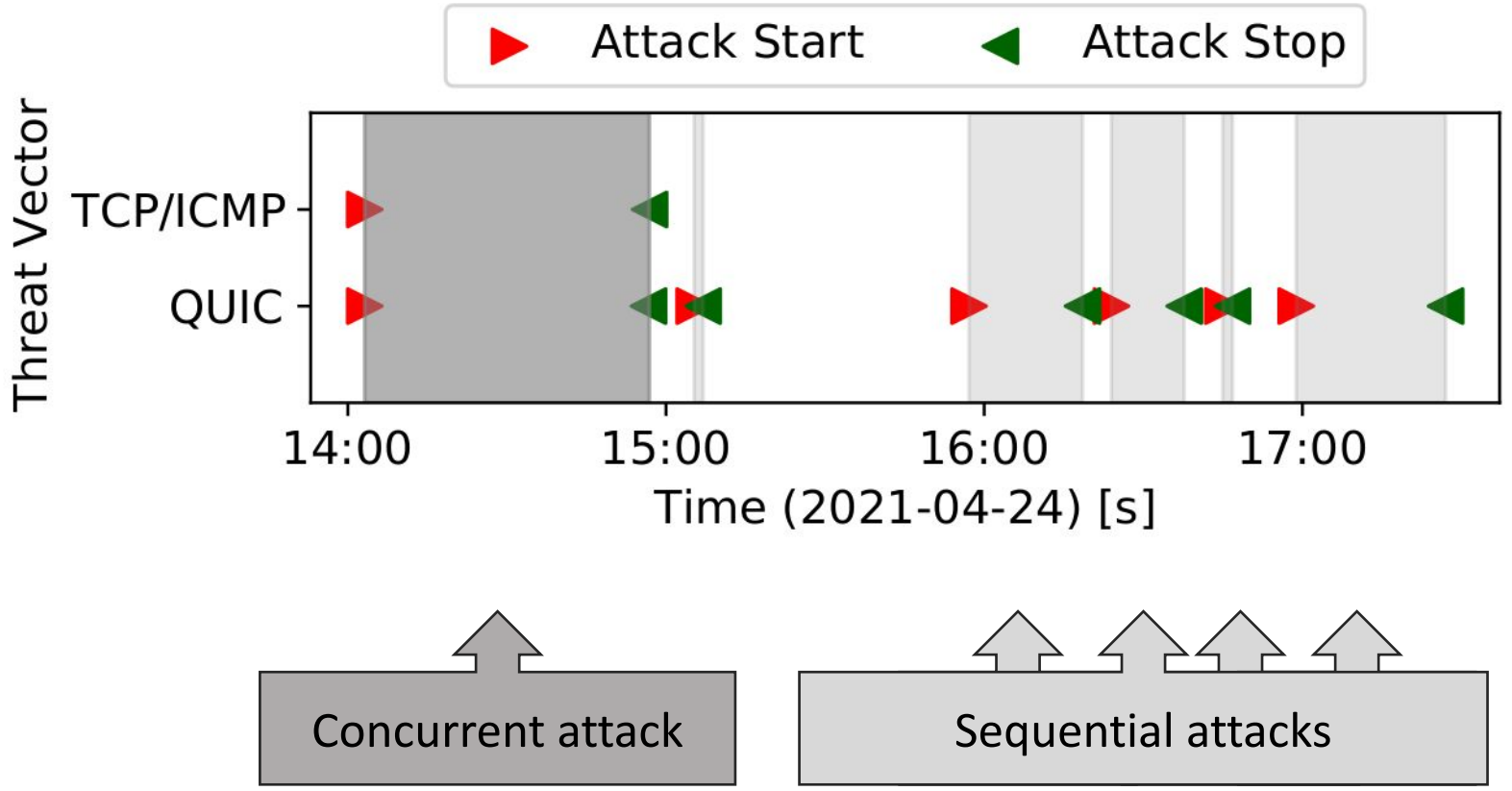
QUIC floods in April 2021.

58%

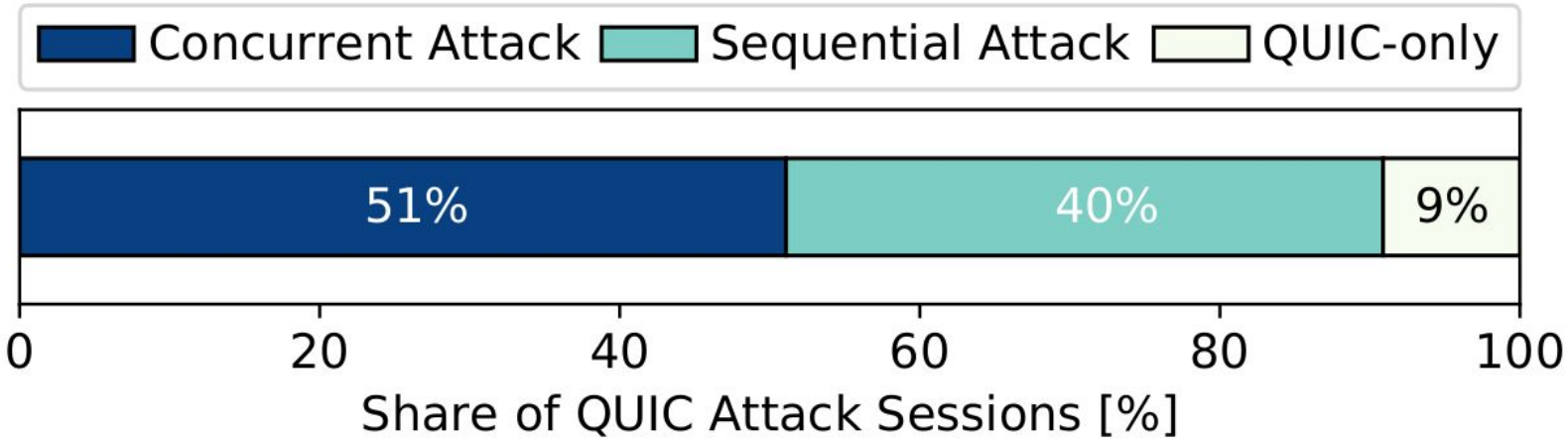
25%

Victims

A closer look at a single victim



Multi-vector attacks are common: QUIC INITIAL and TCP SYN floods co-occur



A mitigation option: QUIC RETRY.

Similar to TCP SYN cookies, RETRY messages force the client to return with a unique token.

This proves its authenticity but adds a full round-trip to the connection setup.

https://www.br.de/nachrichten/bayern/anstehen-statt-einloggen-warum-behoerden-noch-kaum-digital-sind

Do QUIC floods really work? Yes, NGINX is vulnerable without RETRY.

Attack	NGINX Config		Results			
Volume [pps]	QUIC Retry	Workers [#]	Client [# Req]	Server [# Resp]	Service Available	Extra RTT
10	✘	4	3,001	12,004	100%	✘
100	✘	4	30,001	81,520	68%	✘
1,000	✘	4	300,001	81,520	7%	✘

Do QUIC floods really work? Yes, NGINX is vulnerable without RETRY.

More CPUs just **delay** the problem

Attack		NGINX Config	Results			
Volume [pps]	QUIC Retry	Workers [#]	Client [# Req]	Server [# Resp]	Service Available	Extra RTT
10	✘	4	3,001	12,004	100%	✘
100	✘	4	30,001	81,520	68%	✘
1,000	✘	4	300,001	81,520	7%	✘
1,000	✘	auto=128	300,001	1,200,004	100%	✘
10,000	✘	auto=128	499,798	521,728	26%	✘
100,000	✘	auto=128	498,505	320,222	26%	✘

Do QUIC floods really work? Yes, NGINX is vulnerable without RETRY.

More CPUs just **delay** the problem

Attack	NGINX Config		Results			
Volume [pps]	QUIC Retry	Workers [#]	Client [# Req]	Server [# Resp]	Service Available	Extra RTT
10	✘	4	3,001	12,004	100%	✘
100	✘	4	30,001	81,520	68%	✘
1,000	✘	4	300,001	81,520	7%	✘
1,000	✘	auto=128	300,001	1,200,004	100%	✘
10,000	✘	auto=128	499,798	521,728	26%	✘
100,000	✘	auto=128	498,505	320,222	26%	✘
1,000	✓	4	300,001	300,001	100%	✓
10,000	✓	4	499,798	499,798	100%	✓
100,000	✓	4	499,798	499,798	100%	✓

Enabling RETRY **prevents** the DoS

Do QUIC floods really work? Yes, NGINX is vulnerable without RETRY.

Attack	NGINX Config		Results			
Volume	QUIC	Workers	Client	Server	Service	Extra
1,000	✓	4	300,001	300,001	100%	✓
10,000	✓	4	499,798	499,798	100%	✓
100,000	✓	4	499,798	499,798	100%	✓

We did not find any RETRY packets in the DoS backscatter.

RETRY is not used by the large content providers under attack.

Conclusion & Outlook

QUIC INITIAL floods are an actively misused (multi-)attack vector.

We detected and quantified QUIC DoS attacks using a network telescope.

Can we fine-tune the DoS thresholds?

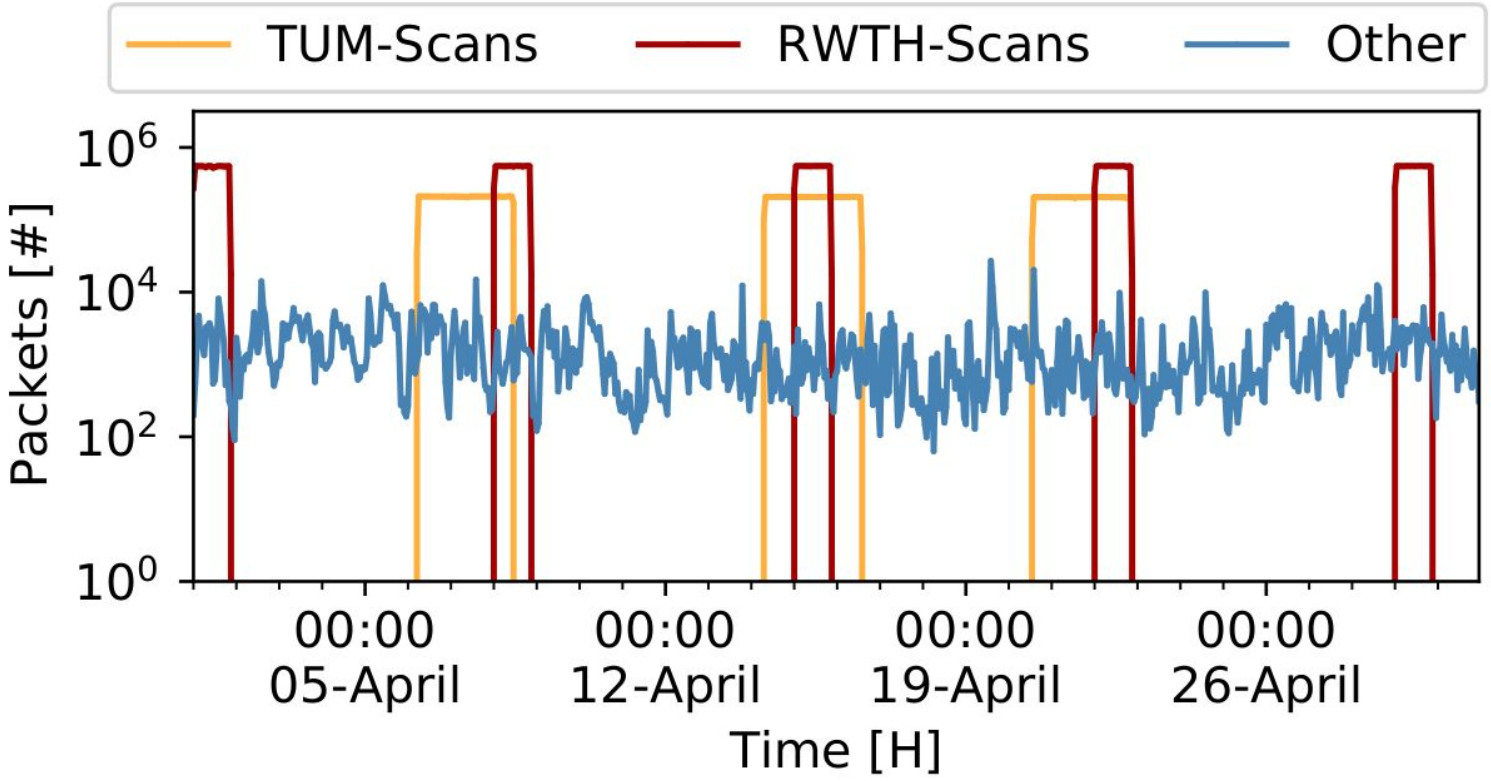
Is the deployment of RETRY worth the cost?

[Artifact: https://doi.org/10.5281/zenodo.5504169]

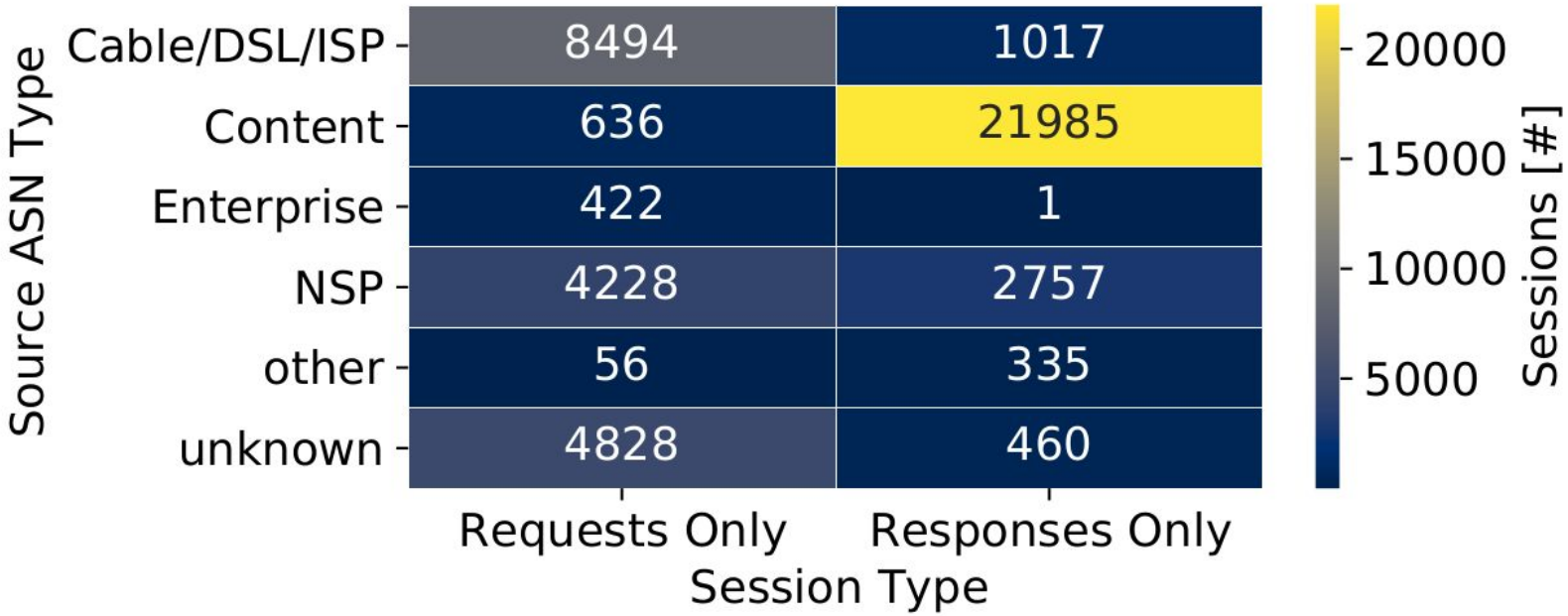
Backup Slides

++

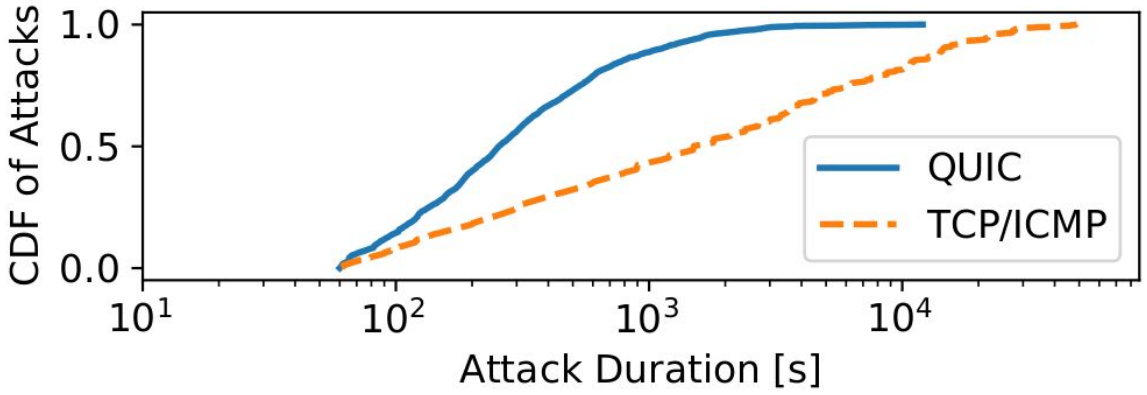
Was data sanitization necessary? Yes: Research scanners dominate QUIC IBR



ASN types differ per session type

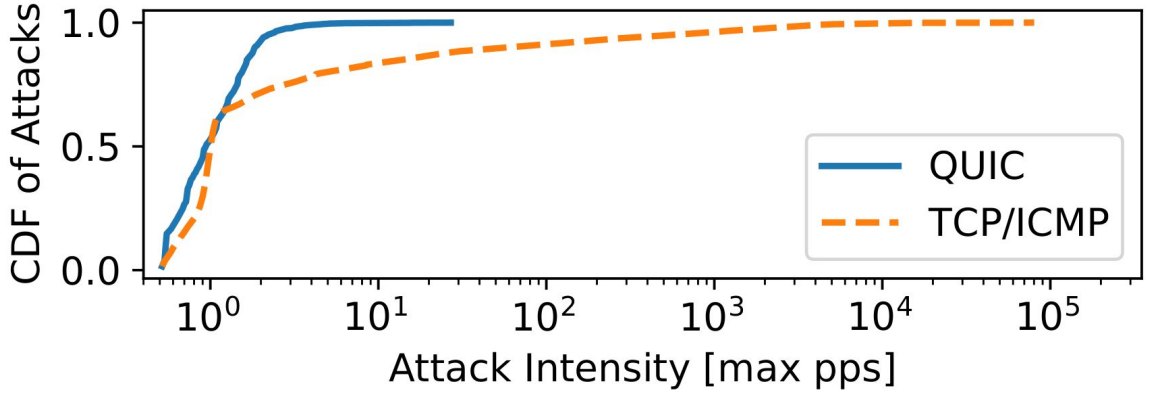


Comparison with common protocols



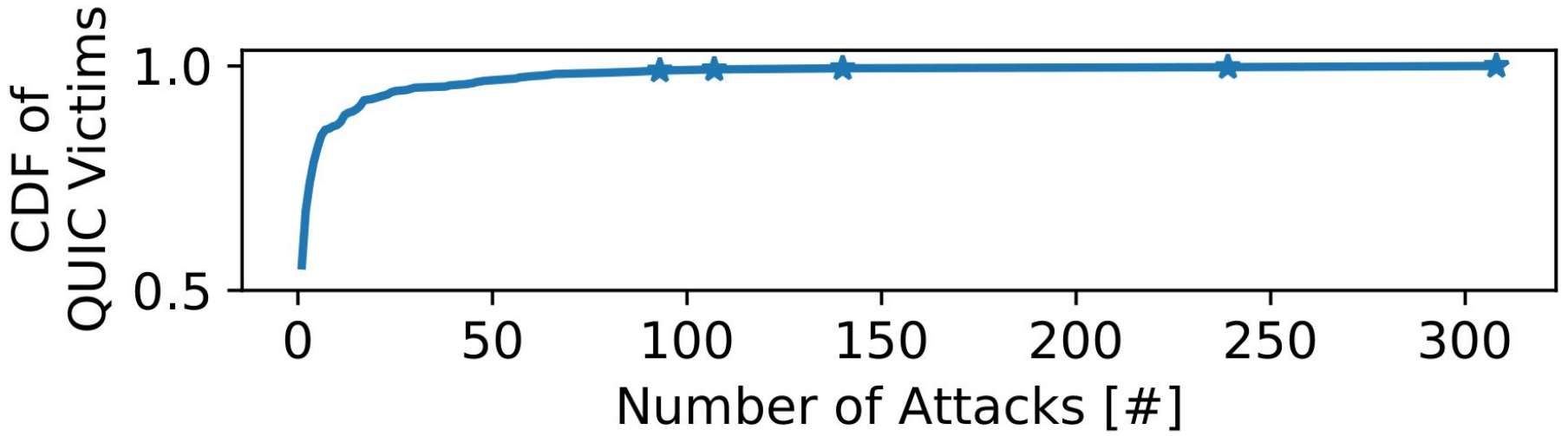
Backscatter events stop for various reasons:

1. the attack has ended
2. a mitigation was initiated
3. victims service is completely unresponsive

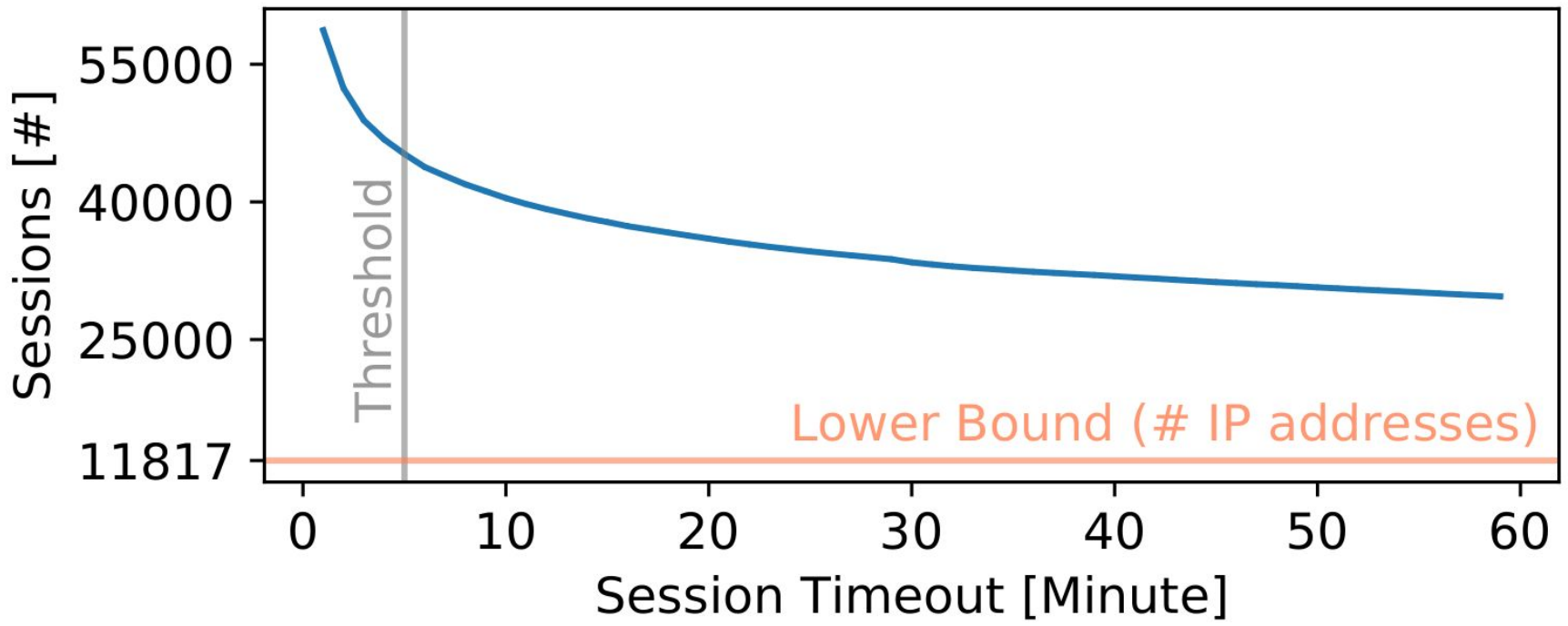


Note that the same maximum PPS might induce different loads for TCP and QUIC.

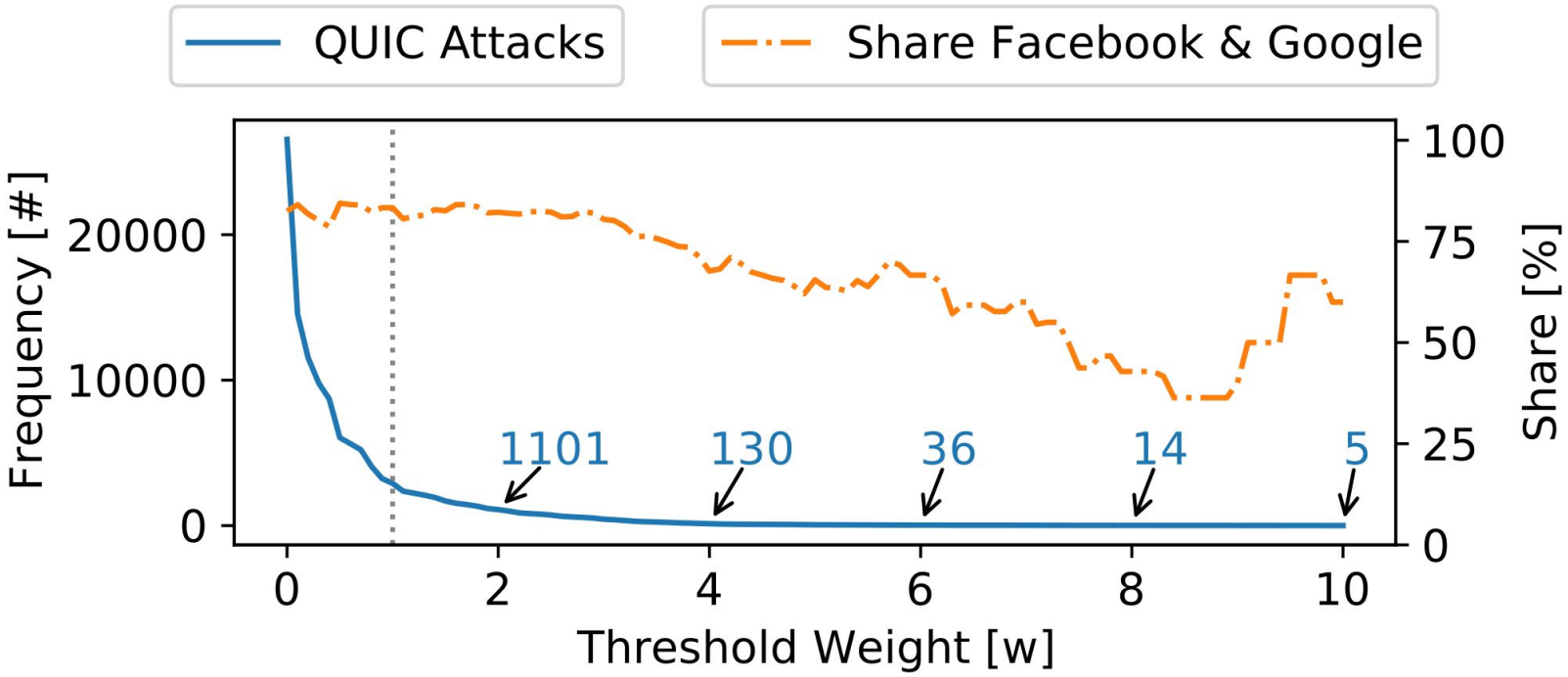
Always the same victim?



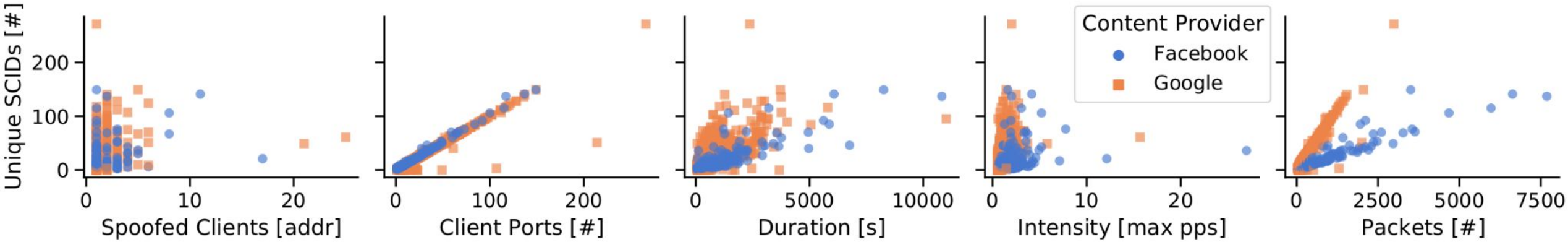
'Old' idle timeout is still appropriate



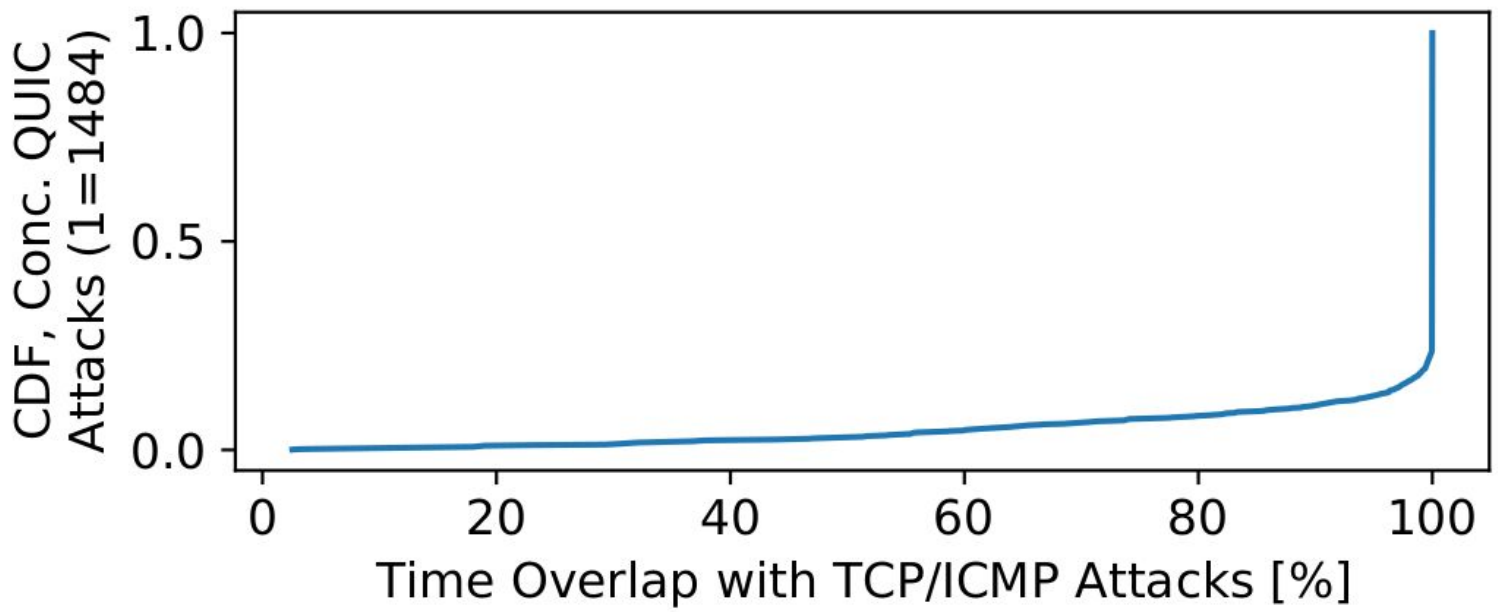
What is the effect of your thresholds?



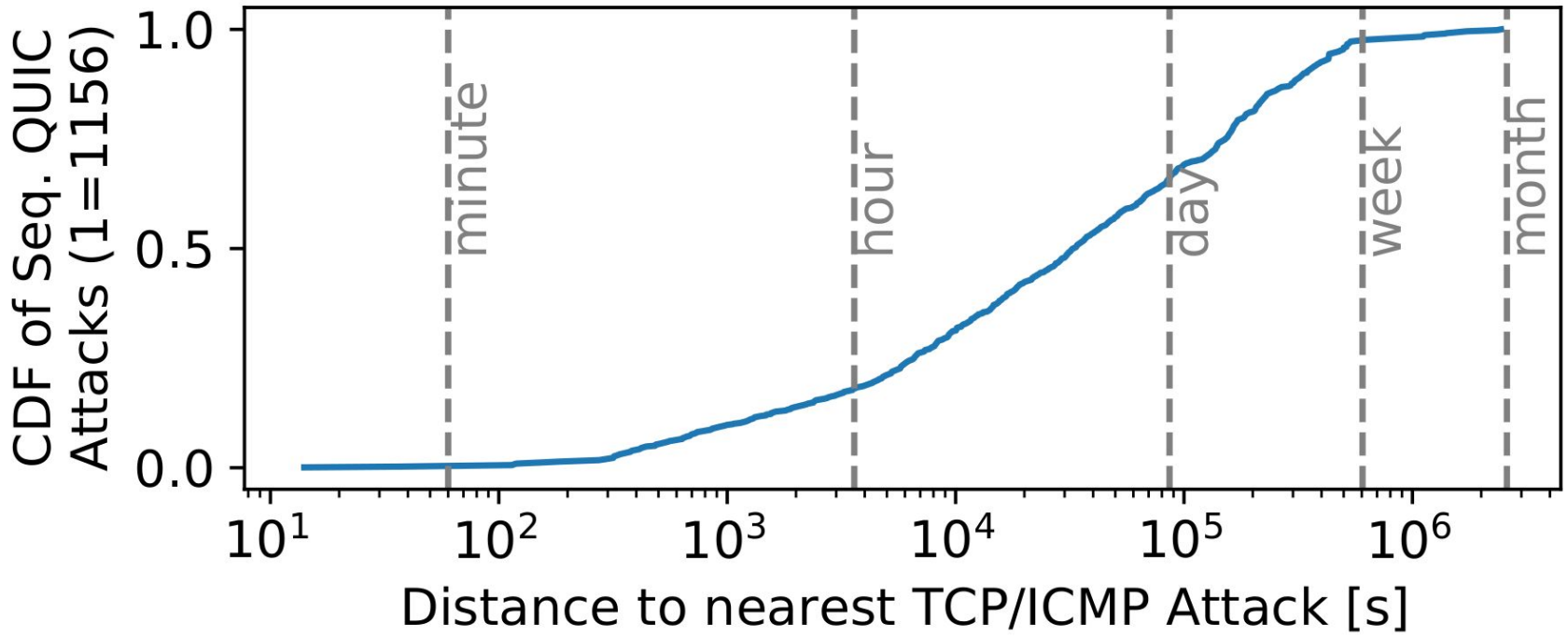
Do attacks differ between content providers?



How much do multi-vector attacks overlap?



Temporal distance between sequential attacks?



Did you implement a QUIC attack tool?

No. We reused common tools to prevent implementation mistakes:

- `nginx` as a webserver
- `quiche` as a HTTP/3 client
- `tcpdump` to record 500k complete QUIC handshakes
- Restart server, resend recorded client INITIALs with `tcpreplay`

How do you classify QUIC messages?

QUIC requests are sent to UDP port 443, INITIALS contain a `TLS CLIENT HELLO`

QUIC responses originate from UDP port 443, INITIALS contain an (encrypted) `TLS SERVER HELLO`

How does your toolchain look like?

I worked on a CAIDA-VM (8 CPU cores, 32 GB).
It took me one week to parse one month of (complete) PCAP data:

1. Linux parallelization (e.g. `parallel` with 4 workers)
2. `swift` downloads with a bandwidth limiter (`trickle`)
3. Parallelized (de-) compressing of zip-files with `pigz`
4. `tcpdump` as a PCAP pre-filter (`not icmp and udp and port 443`)
5. Wireshark as a DPI filter (`quic or gquic`)

   --> Optimized to be as stateless as possible (Wireshark follows flows and creates *unneeded* statistics)
6. Export to CSV and Python/Pandas as post analysis stack