DPIFuzz: A Differential Fuzzing Framework to Detect DPI Elusion Strategies for QUIC

Gaganjeet Singh Reen and Christian Rossow
CISPA Helmholtz Center for Information Security

ACSAC 2020

Minkyung Park

mkpark@mmlab.snu.ac.kr

June 29, 2022

Contents

Deep Packet Inspection Elusion Attack

- DPI (Deep Packet Inspection) detects various security-critical incidents by inspecting application-layer communication content
- DPI systems are typically deployed with man-in-the-middle proxies that assist in intercepting encrypted channel (e.g., TLS)
- Elusion attacks against DPI systems fool their TCP and/or HTTP inspections
- **<u>It uses differences on protocol implementations</u>**
  - How the DPI system and the actual data recipient have implemented a protocol

Deep Packet Inspection Elusion Attack

- The actual recipient of data and the DPI system reassemble different payload data

- Why such differences may occur
  - Protocol specifications (deliberately or not) leave some details out
  - The DPI system may simplify the state machine of stateful transport protocol
  - The DPI system may perform a lower number of checks to validate packets

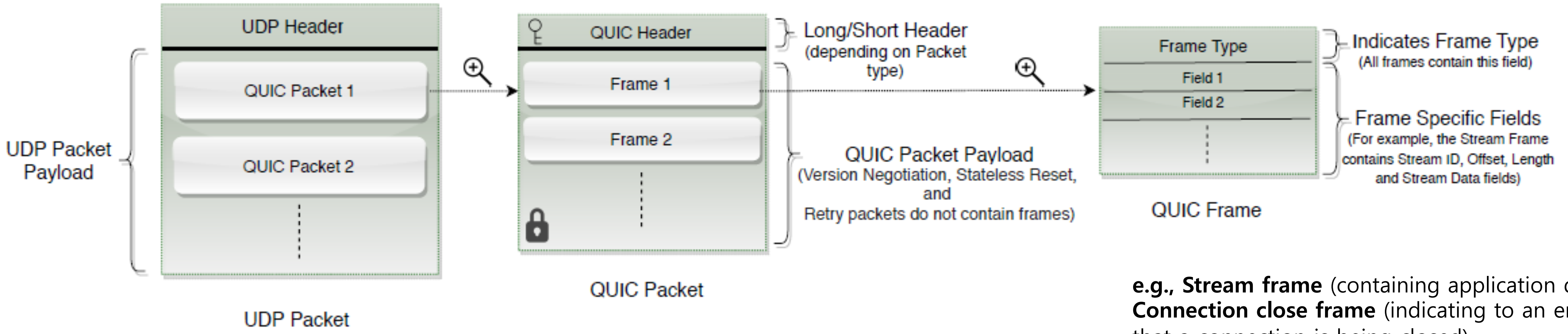
DPI elusion attacks for QUIC

- While QUIC is adopted widely, DPI elusion attacks for QUIC is not sufficiently investigated

- DPIFuzz is a differential fuzzing framework to automatically uncover potential differences between QUIC implementations
  - Uncovers two distinct strategies
  - Uncovers four security-critical vulnerabilities

Background: QUIC Protocol

- QUIC is an encrypted-by-default Internet transport protocol
- It is similar to a combination of TCP, TLS, and HTTP/2 implemented on **UDP**


UDP Header
QUIC Packet 1
QUIC Packet 2
UDP Packet Payload
UDP Packet
QUIC Header
Frame 1
Frame 2
QUIC Packet
Long/Short Header (depending on Packet type)
QUIC Packet Payload (Version Negotiation, Stateless Reset, and Retry packets do not contain frames)
Frame Type
Field 1
Field 2
QUIC Frame
Indicates Frame Type (All frames contain this field)
Frame Specific Fields (For example, the Stream Frame contains Stream ID, Offset, Length and Stream Data fields)

e.g., Stream frame (containing application data), or **Connection close frame** (indicating to an end point that a connection is being closed)

Background: QUIC Protocol

- QUIC is stateful
  - Each connection starts with a handshake phase (using TLS)
- Stream is ordered byte-stream abstraction to an application
  - Each stream is identified by “Stream ID”
- Multiple streams can be used to send data in a connection

Background: Differential Fuzzing

- Fuzzing is the process of providing randomized inputs to programs and observing their behaviors

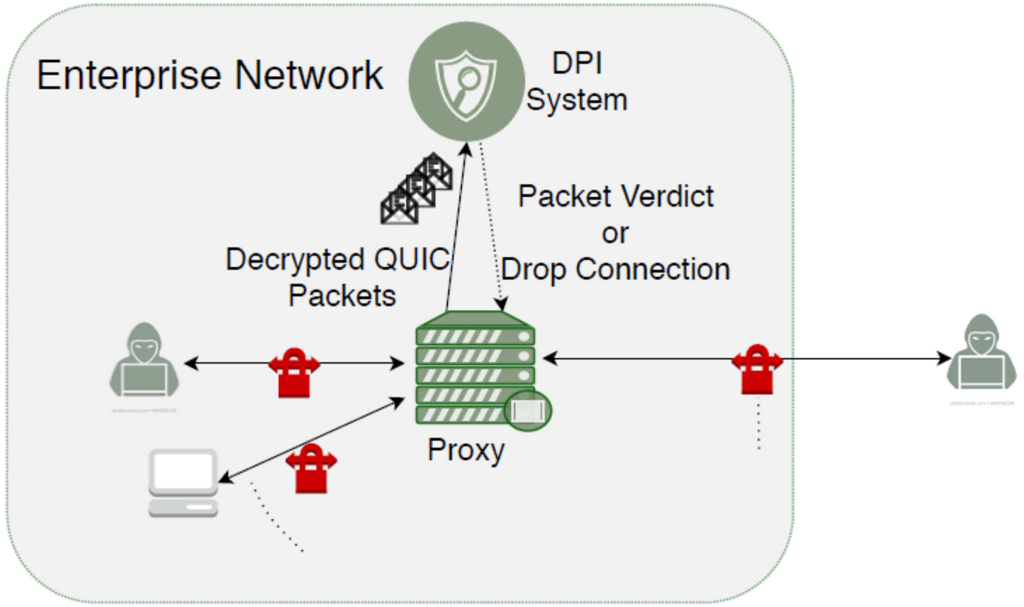
- Differential fuzzing is a type of fuzzing techniques
  - The same input is provided to different implementations
    - e.g., QUICHE by cloudflare, MVFST by Facebook, or QUANT by NetApp
    - It is expected that they behave identical given the same input
  - Their behaviors and/or responses are compared

Goal

- The goal is to detect strategies of DPI elusion attacks of QUIC
- DPI elusion attacks can be caused by sequences of packets that contain denylisted payload, but remain unnoticed by the DPI system
- DPIFuzz aims to reveal sequences of QUIC packets that are reassembled differently by the server and the DPI system
    - Both use different QUIC implementations
- Those sequences are analyzed to find the underlying reason

Threat model

- An inline proxy
  - Establishes a QUIC connection with both the client and the server
  - Forwards the traffic between the two and to the DPI system



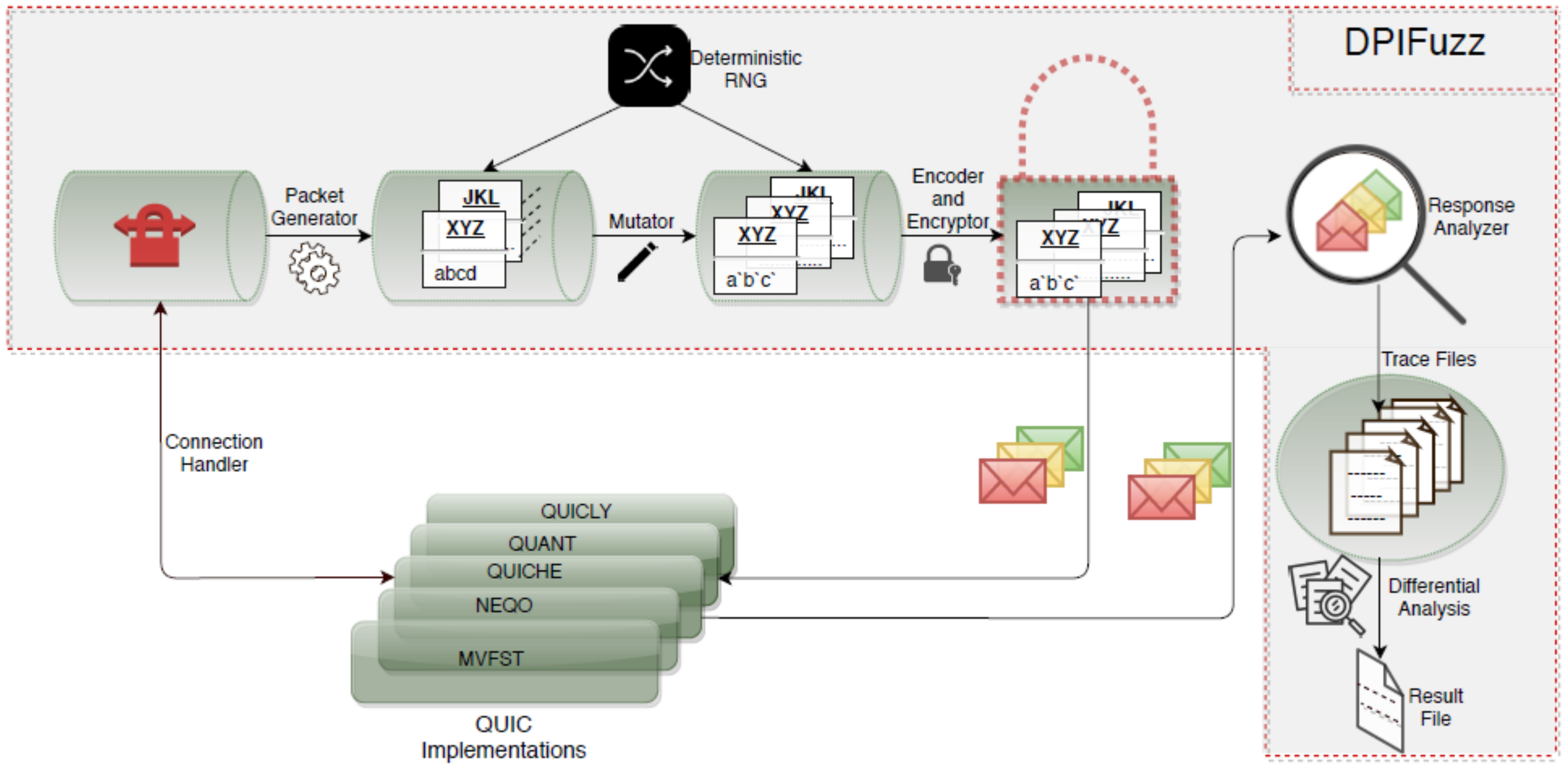
- DPI system
  - Reconstructs the streams and analyzes the packets for denylisted content
  - Either sends a verdict to the proxy for each packet or sends the proxy an asynchronous signal to drop a connection

Elusion strategies

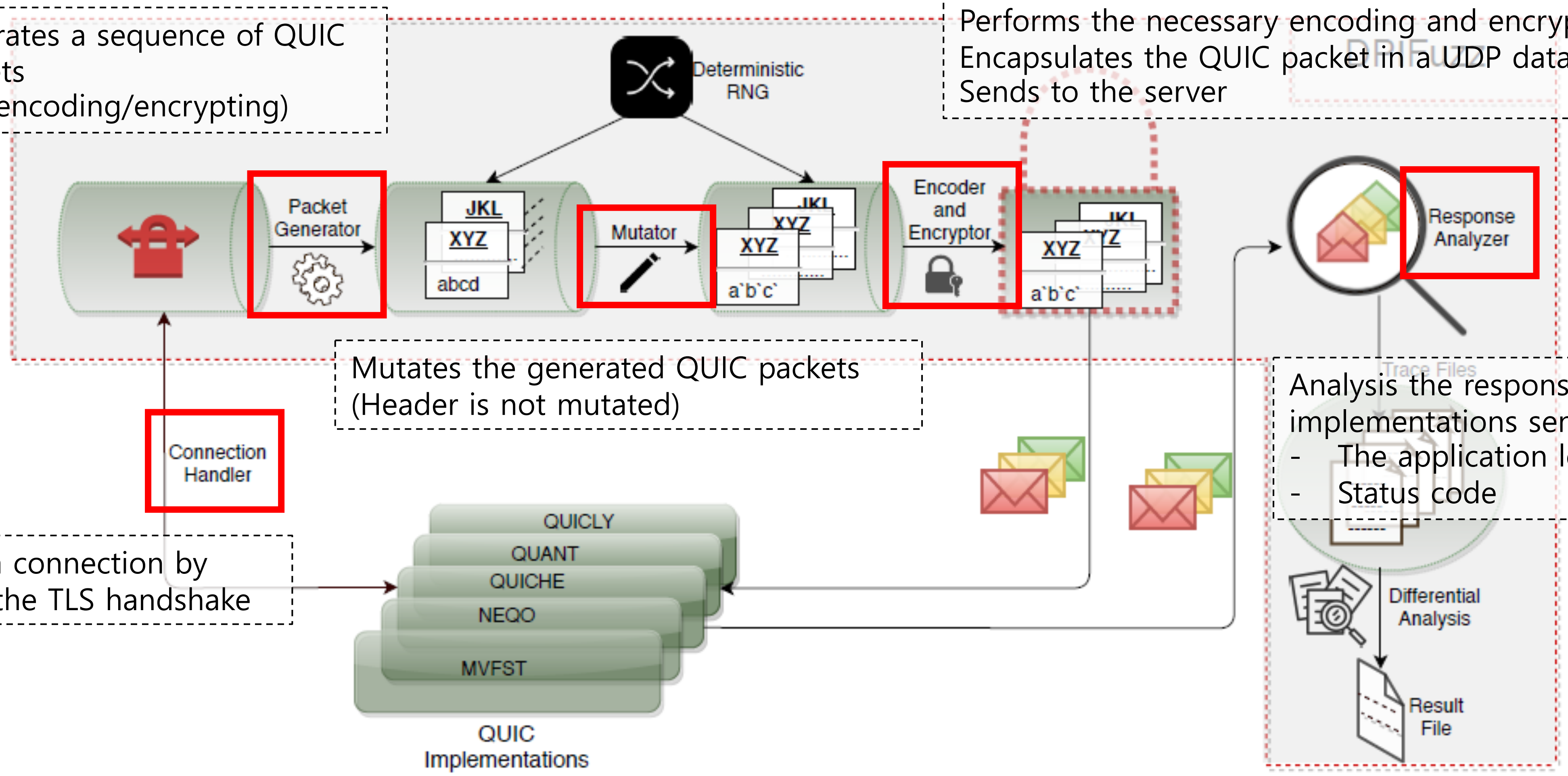
- Insertion packet/frame
  - Some packets/frames might be accepted by the DPI system and rejected by the server
  - It results in extra payload being registered at the DPI system

- Evasion packet/frame
  - Some packets/frames might be accepted by the server and rejected by the DPI system
  - It allows sending data to the server without it being registered at the DPI system

DPIFuzz: Overview



DPIFuzz: Overview



Generates a sequence of QUIC packets
(w/o encoding/encrypting)

Performs the necessary encoding and encryption
Encapsulates the QUIC packet in a UDP datagram
Sends to the server

Mutates the generated QUIC packets
(Header is not mutated)

Analysis the responses that the implementations sends
- The application level data
- Status code

Establishes a connection by completing the TLS handshake

Packet Generator

- Packet generator create streams and packets in the first place (then mutated)

- The generators can or cannot have a control over the generated sequence
  - (1) Randomly decide which types of frames/packets to create and randomly group frames into packet payload
    - No consideration of whether the specification allows a particular packet to have specific frame types of if a client is even allowed to send specific frames
  - (2) **Create specific types of packet sequences**
    - The type of packets and frames and the grouping of frames into packet payload are controlled

Mutations

- **Sequence-level mutations** affect the sequence of packets
  - Shuffle: the order of packets in a sequence is randomly shuffled
  - Duplicate: randomly selected packets are duplicated
  - Drop: randomly selected packets are dropped
  - A sequence can undergo the three mutations with a probability α_1, α_2 and α_3

- **Packet-level mutations** affect an individual packet payload with probability γ
  - **Payload mutations** consider QUIC packet payload simply as a collection of bytes (probability ω)
  - **Frame mutations** are defined for the individual frames contained in the QUIC packet payload (probability $1 - \omega$)

Packet-level Mutations

- **Payload mutations** do not consider the structure of the frame
  - Repeat payload: a random substring is selected and injected at a random position
  - Alter payload: for each byte, a random decision is made about whether to fuzz the byte or not
  - Add random payload: a random payload is selected and inserted at a random position without overwriting
  - Drop random payload: a random number of bytes from a randomly selected offset are dropped

- **Frame mutations** is to fuzz each field of a frame based on its structure

Differential Analysis

- For analysis, echo servers are implemented to see the application-level payload they received from the fuzzer (i.e., client)

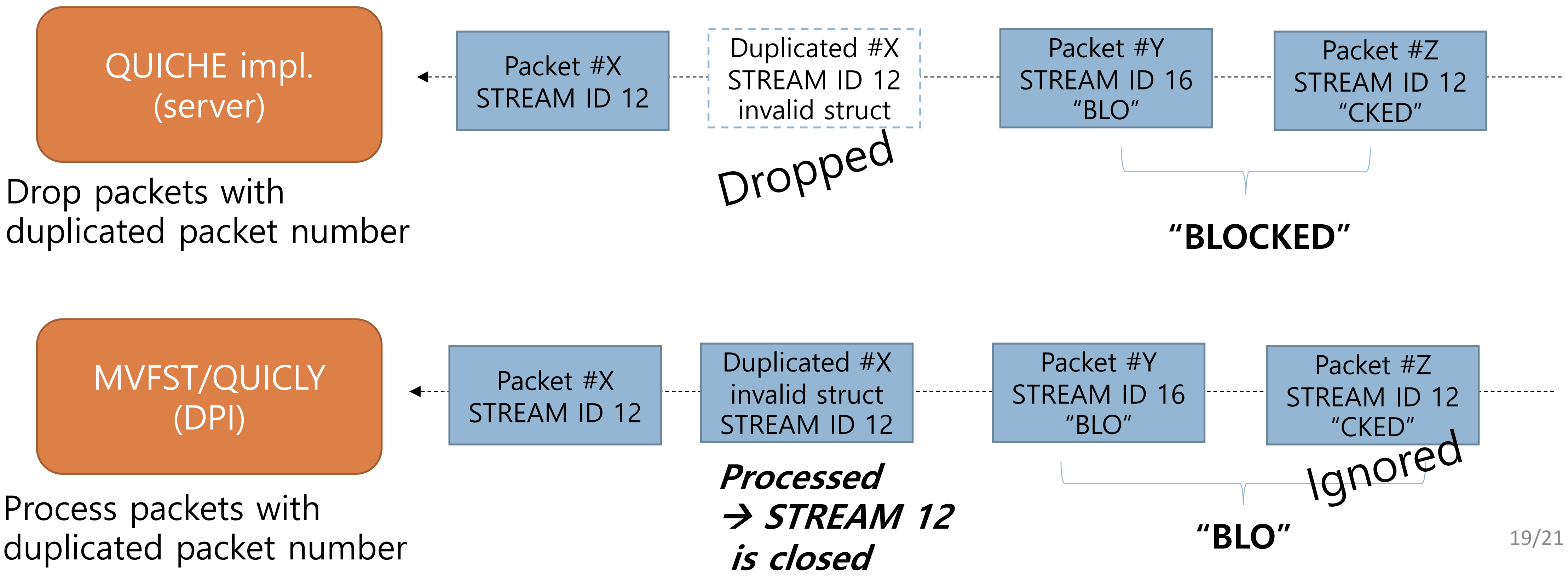
- The fuzzer records application-level data as well as status code
  - ServerTimeOut
  - TLSHandshakeFail
  - ServerDidNotRespond
  - ServerIsAlive

Experiment overview

- Target implementations
  - QUICHE by cloudflare, MVFST by Facebook, QUANT by NetApp, NEQO by Mozilla and QUICLY by Fastly
- Echo servers and the client run on a locally hosted VM
  - The implementations being compared are fed the exact same logical sequence of packets
  - The reassembly differences are a consequence of design differences
- DPIFuzz created 600 unique sequences and the test takes about 2.5 hours
- DPIFuzz found **two implementation differences** and four vulnerabilities

Reassembly Differences 1)

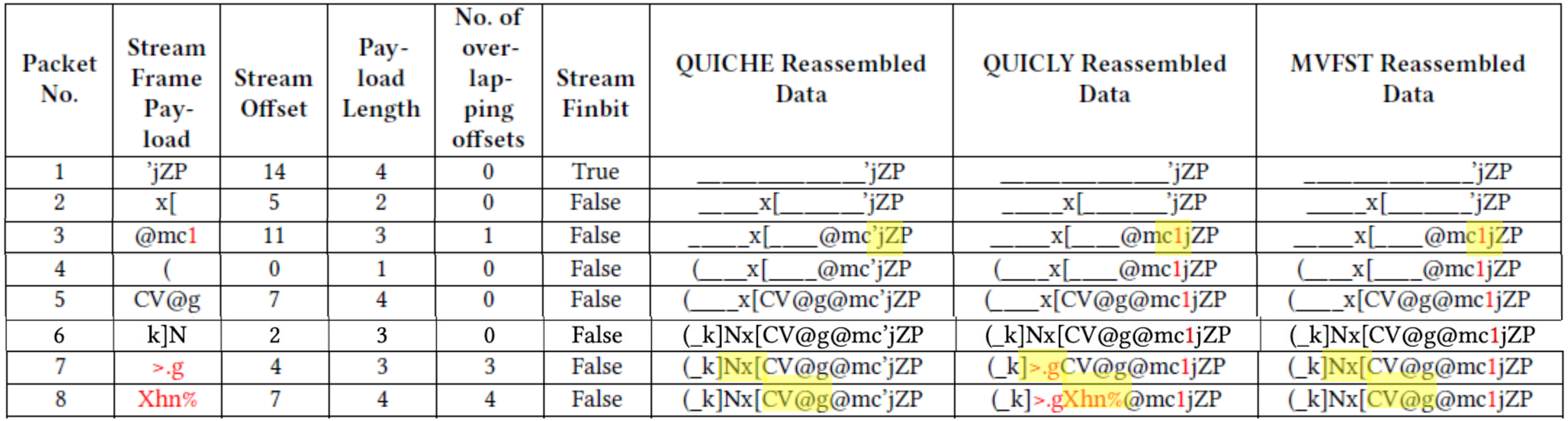
- Inserting packets with duplicate packet numbers are ignored by some IUTs, but not by others
  - e.g., "BLOCKED" is denylisted



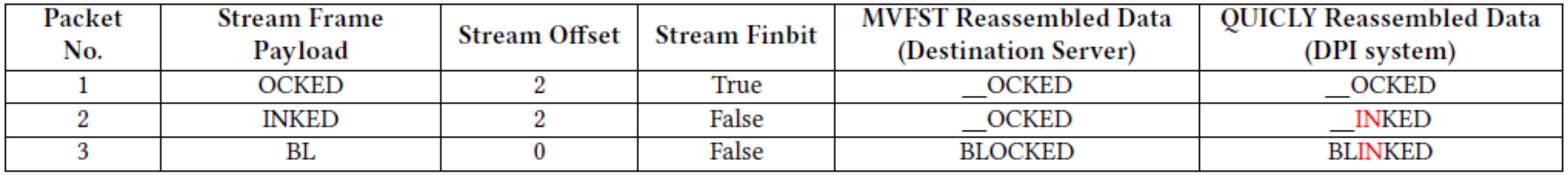
QUICHE impl. (server)
Packet #X STREAM ID 12
Duplicated #X STREAM ID 12 invalid struct
Packet #Y STREAM ID 16 "BLO"
Packet #Z STREAM ID 12 "CKED"
Dropped
"BLOCKED"
Drop packets with duplicated packet number
MVFST/QUICLY (DPI)
Packet #X STREAM ID 12
Duplicated #X invalid struct STREAM ID 12
Packet #Y STREAM ID 16 "BLO"
Packet #Z STREAM ID 12 "CKED"
Processed → STREAM 12 is closed
Ignored
"BLO"
Process packets with duplicated packet number

Reassembly Differences 2)

- The implementations handle receiving data at overlapping offsets in different ways



Packet No.	Stream Frame Pay-load	Stream Offset	Pay-load Length	No. of over-lap-ping offsets	Stream Finbit	QUICHE Reassembled Data	QUICLY Reassembled Data	MVFST Reassembled Data
1	’jZP	14	4	0	True	______________’jZP	______________’jZP	______________’jZP
2	x[	5	2	0	False	_____x[_______’jZP	_____x[_______’jZP	_____x[_______’jZP
3	@mc1	11	3	1	False	_____x[____@mc’jZP	_____x[____@mc1jZP	_____x[____@mc1jZP
4	(	0	1	0	False	(____x[____@mc’jZP	(____x[____@mc1jZP	(____x[____@mc1jZP
5	CV@g	7	4	0	False	(____x[CV@g@mc’jZP	(____x[CV@g@mc1jZP	(____x[CV@g@mc1jZP
6	k]N	2	3	0	False	(_k]Nx[CV@g@mc’jZP	(_k]Nx[CV@g@mc1jZP	(_k]Nx[CV@g@mc1jZP
7	>.g	4	3	3	False	(_k]Nx[CV@g@mc’jZP	(_k]>.gCV@g@mc1jZP	(_k]Nx[CV@g@mc1jZP
8	Xhn%	7	4	4	False	(_k]Nx[CV@g@mc’jZP	(_k]>.gXhn%@mc1jZP	(_k]Nx[CV@g@mc1jZP



Packet No.	Stream Frame Payload	Stream Offset	Stream Finbit	MVFST Reassembled Data (Destination Server)	QUICLY Reassembled Data (DPI system)
1	OCKED	2	True	__OCKED	__OCKED
2	INKED	2	False	__OCKED	__INKED
3	BL	0	False	BLOCKED	BLINKED

Conclusion

- Although QUIC is widely adopted, its DPI elusion attacks are rarely investigated
- The paper presented a differential fuzzing framework, DPIFuzz, to detect DPI elusion strategies for the QUIC protocol
- DPIFuzz leverages differential fuzzing, which compares behaviors of various QUIC implementations given the same input
- Finally, it uncovers two strategies and four security vulnerabilities
  - The bugs and vulnerabilities are disclosed to the developers of the implementations