Passthru: Protocol Omni-multiplexer

CSCI-ECEN 5273 Network Systems, 2022 Fall

Gaukas Wang, Parker Morris, Sharon Moses Jangam

University of Colorado Boulder

Special Thanks

- Thanks to Bailey
  - for switching presentation timeslot with us!

Overview

- Background
- Original Problem – Inefficiency in proxying TLS traffics
- A smarter way – Passthru
  - Design
  - Implementation
  - Application Scenario

Background

- Reverse Proxy
  - Sits in front of back-end application servers and forward requests
- CDN
  - Geographically distributed network of (reverse) proxy servers
- International Private Leased Circuit (IPLC)
- International Ethernet Private Line (IEPL)

Background

- Transport Layer Security (TLS)
  - CA-signed certificate-based cryptographic protocol
  - Integrity, Authenticity and Privacy per se, MITM attack prevented
  - The foundation of modern HTTP-over-TLS (https://)
  - Designed for TCP. UDP adaptations: DTLS(WebRTC), QUIC(HTTP3)

Traditional ways to proxy TLS

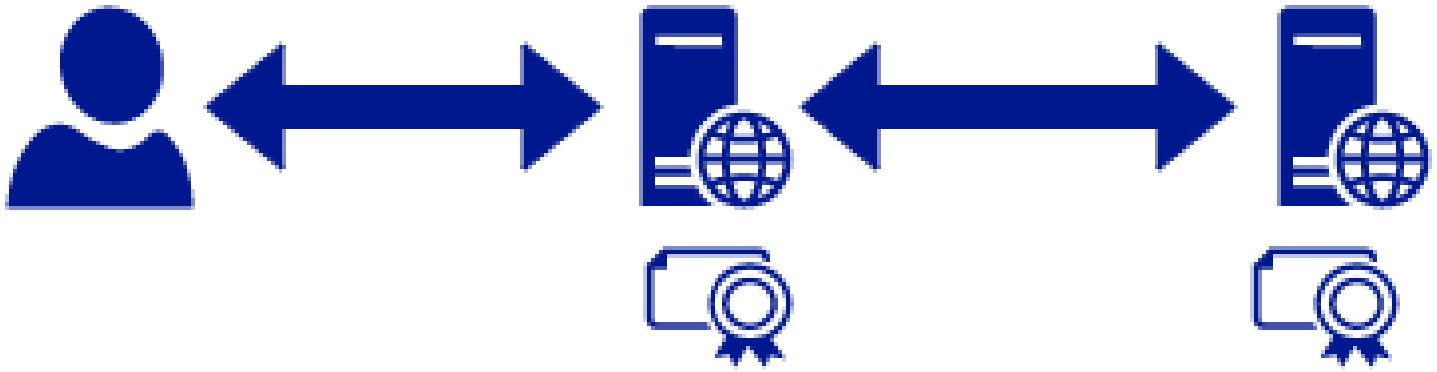
- Reverse Proxy for a TLS service, e.g., HAProxy
  - 2 TCP Connections, 1 TLS Session
  - Port Mapping/Forwarding + optional Load Balancer

Traditional ways to proxy TLS

- Reverse Proxy for a TLS service, e.g., HAProxy
  - 2 TCP Connections, 1 TLS Session
  - Port Mapping/Forwarding + optional Load Balancer
  - Basically, 1:1 NAT -- which is expensive

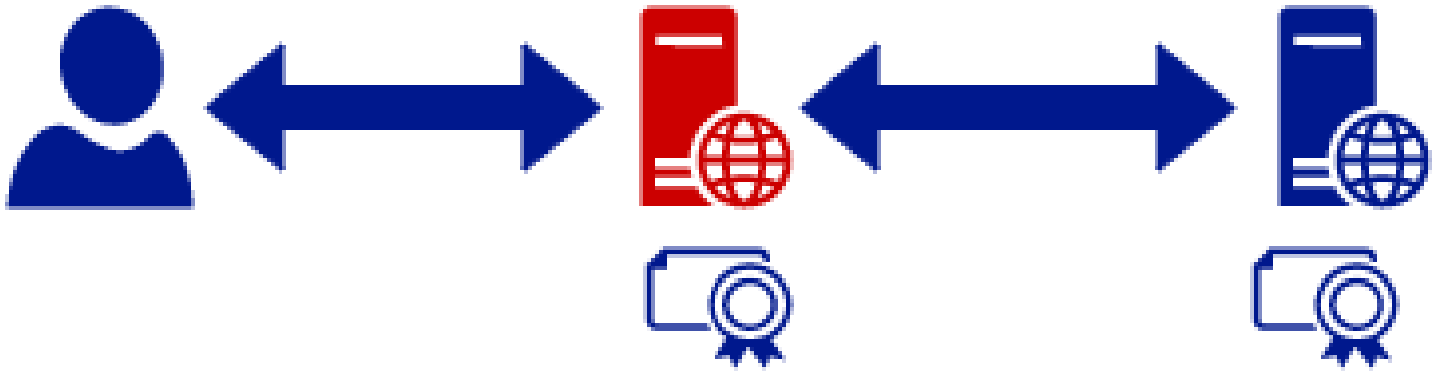
Traditional ways to proxy TLS

- CDN for a TLS Website (via HTTPS), e.g., Cloudflare
  - 2 TCP Connections, 2 TLS Sessions
  - Caching + Load Balancing + optional Attack Mitigation

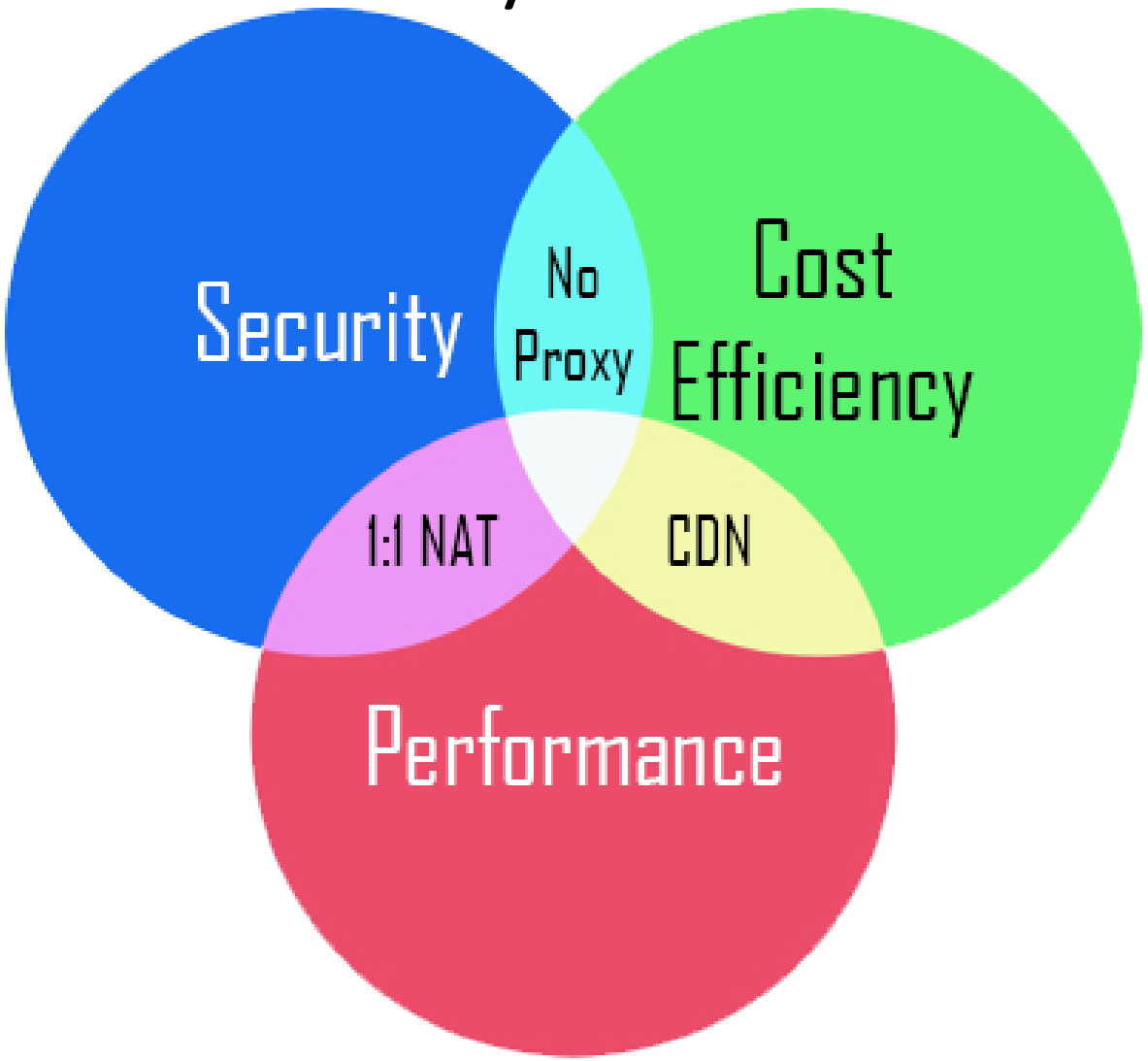


Traditional ways to proxy TLS

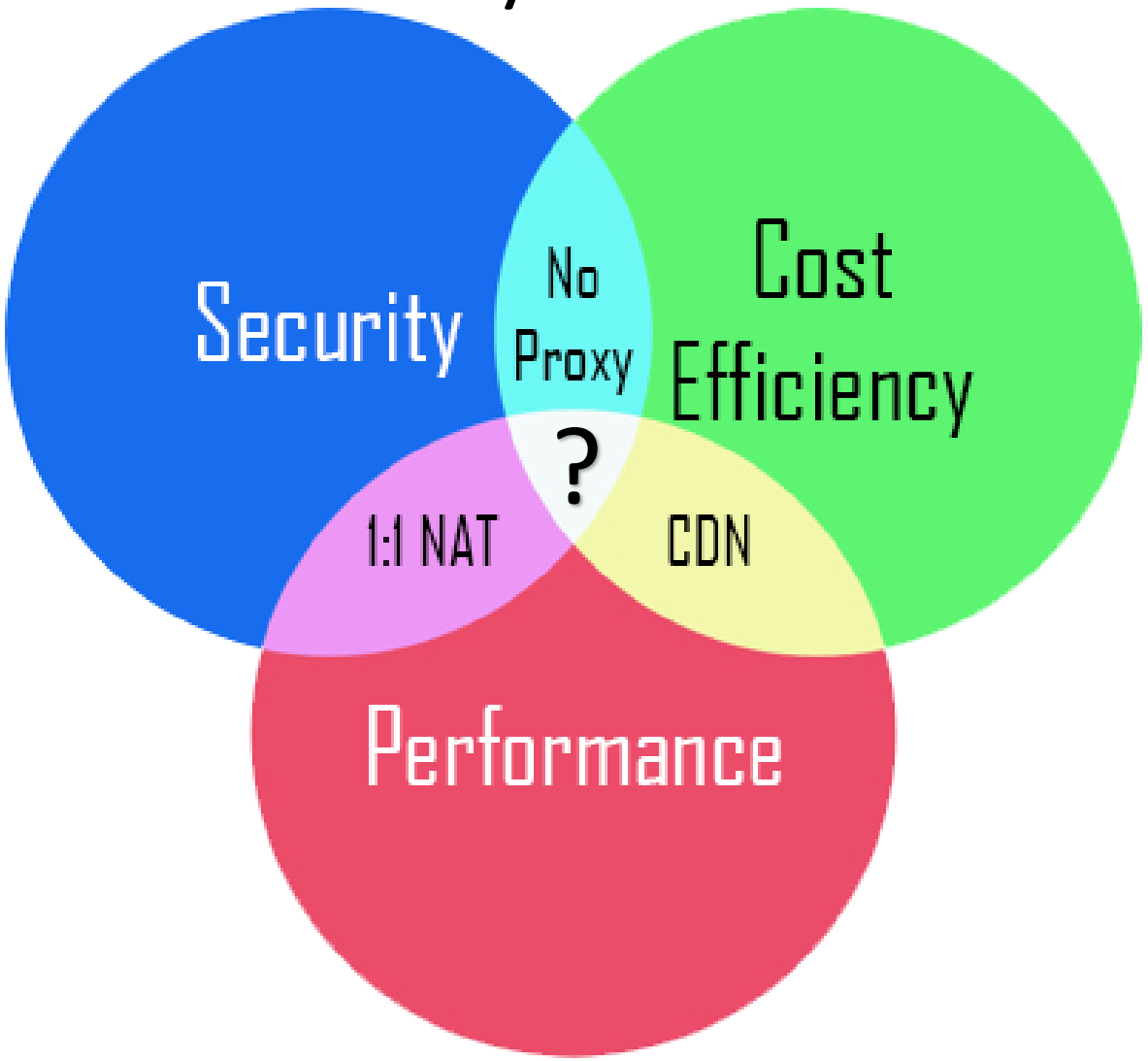
- CDN for a TLS Website (via HTTPS), e.g., Cloudflare
  - 2 TCP Connections, 2 TLS Sessions
  - Caching + Load Balancing + optional Attack Mitigation



Trilemma: Security vs. Cost vs. Performance



Trilemma: Security vs. Cost vs. Performance



A smarter (yet simple) way

- Deep Packet Inspection (DPI)

- Assumptions
  - Protocols MUST be distinguishable before a server responds
  - Protocol Identifications MUST be deterministic and exclusive
  - Protocol identifications SHOULD happen at line speed

Passthru

- Ad hoc DPI-based Protocol Omni-multiplexer
  - DPI: for Application Data Sniffing
  - Omni: Highly programmable/customizable
  - Protocols are identified at Transport Layer (or lower)

- Prototype
  - Unprivileged user space application at Transport Layer written in Go
  - Detects TCP Protocols only
  - TLS Protocol Detection Module implemented as example, AND
  - Allows custom pluggable Detection Modules to be added by user

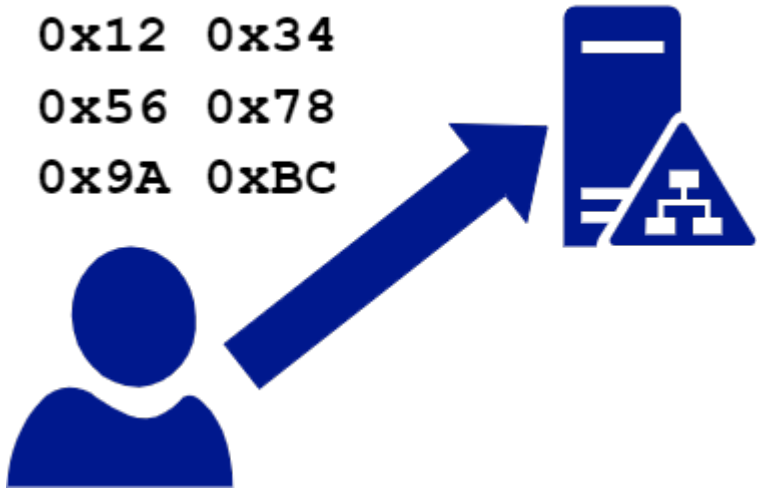
Definition: Objects

- `Server`
  - Accepts incoming connections
  - Dials outgoing connections
  - Relays data between clients and destination hosts

- `Protocol`
  - Performs DPI on incoming bytes
  - Tells `Server` if matching

Design

Design



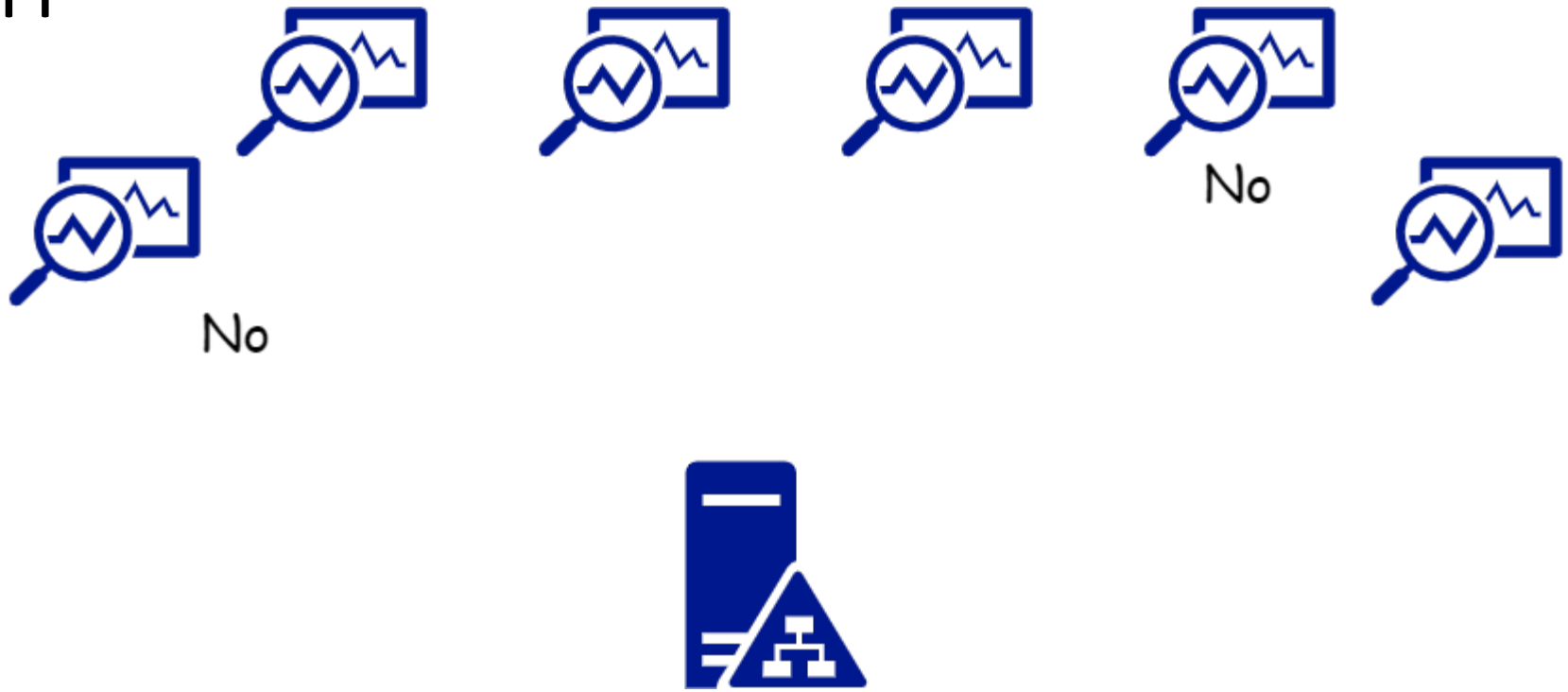
Design

```
0x12  0x34
0x56  0x78
0x9A  0xBC
```

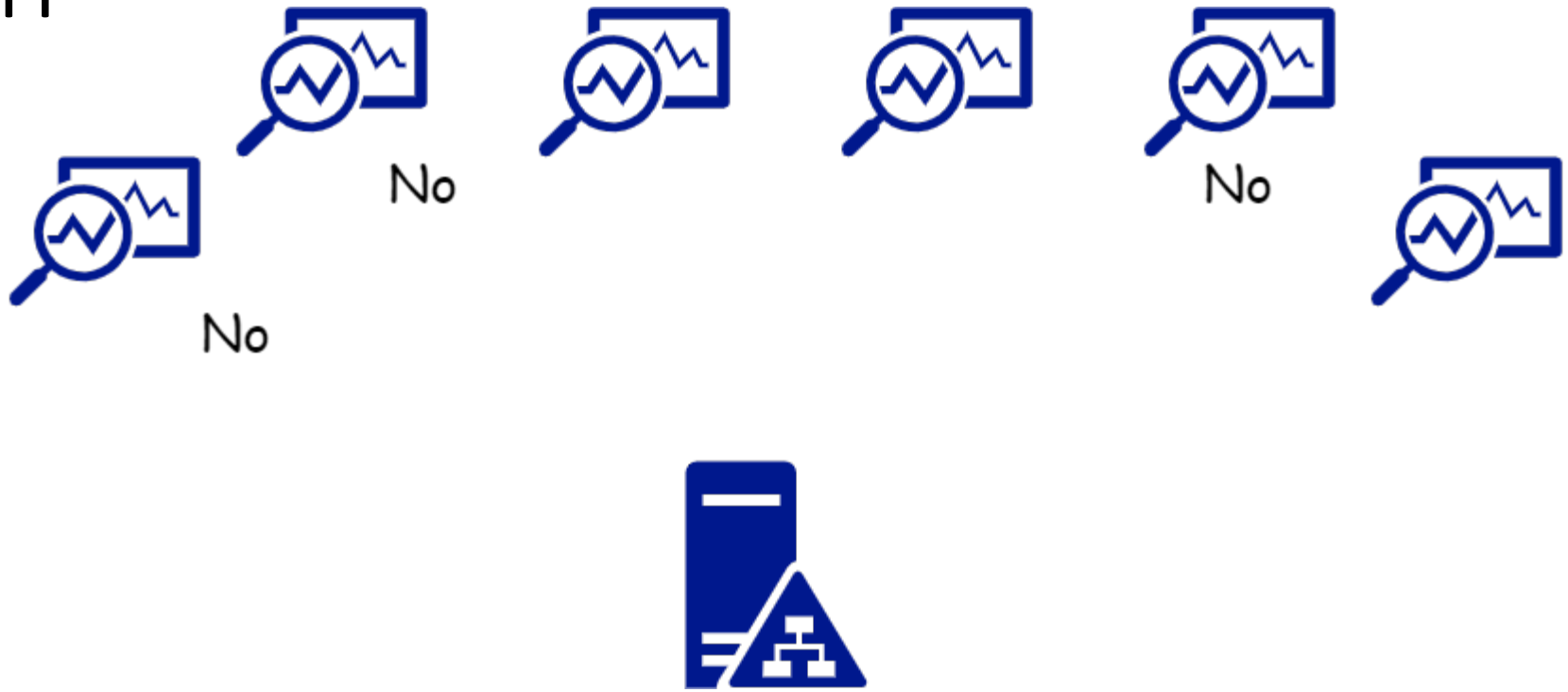
Anyone recognize this?
If so, tell me: what to do?

Design

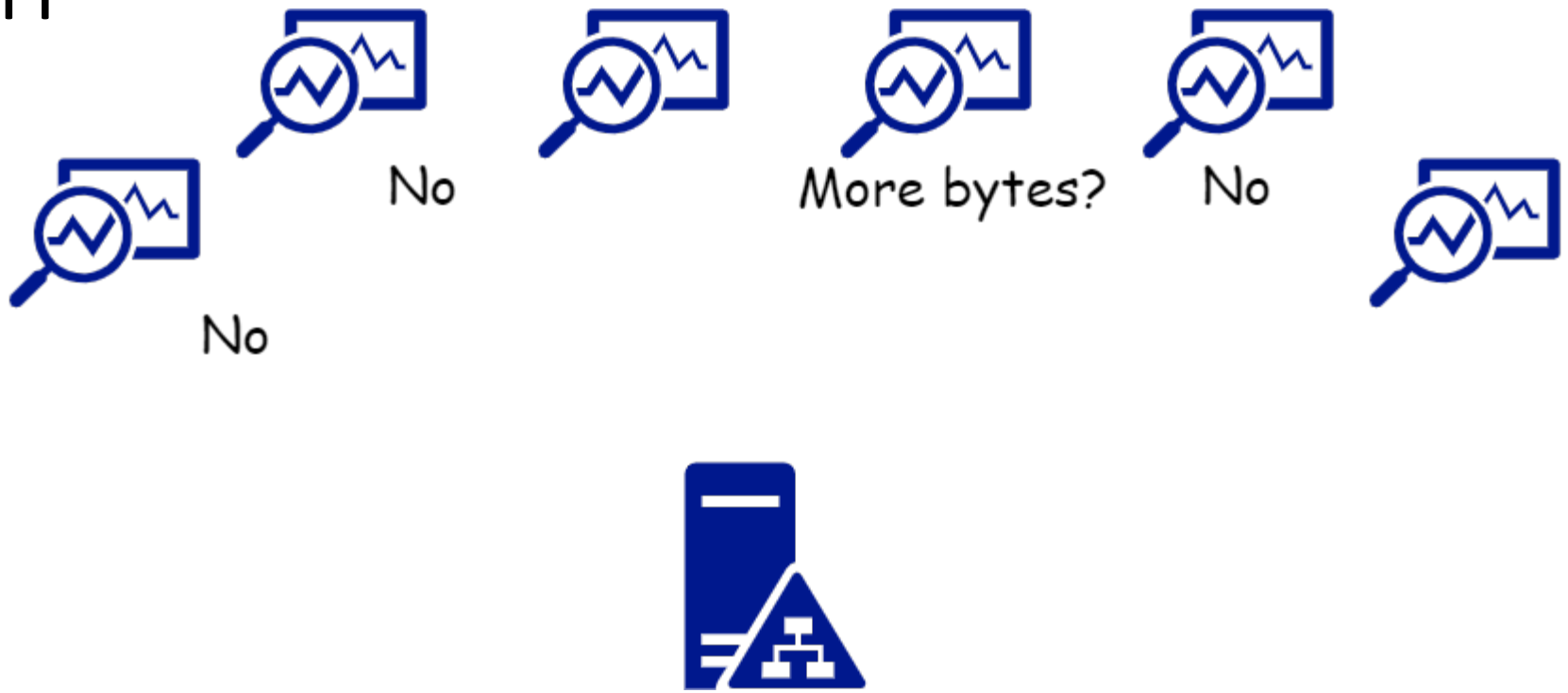
Design



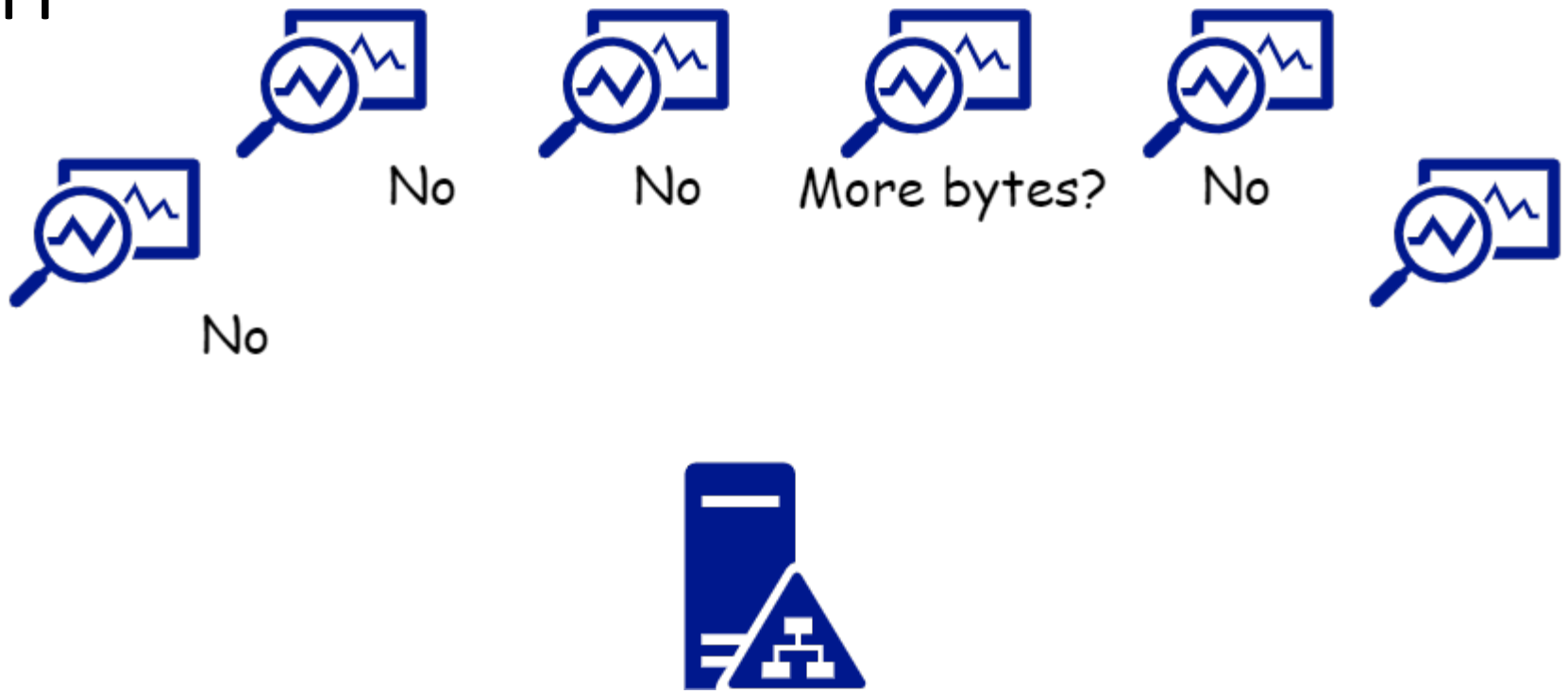
Design



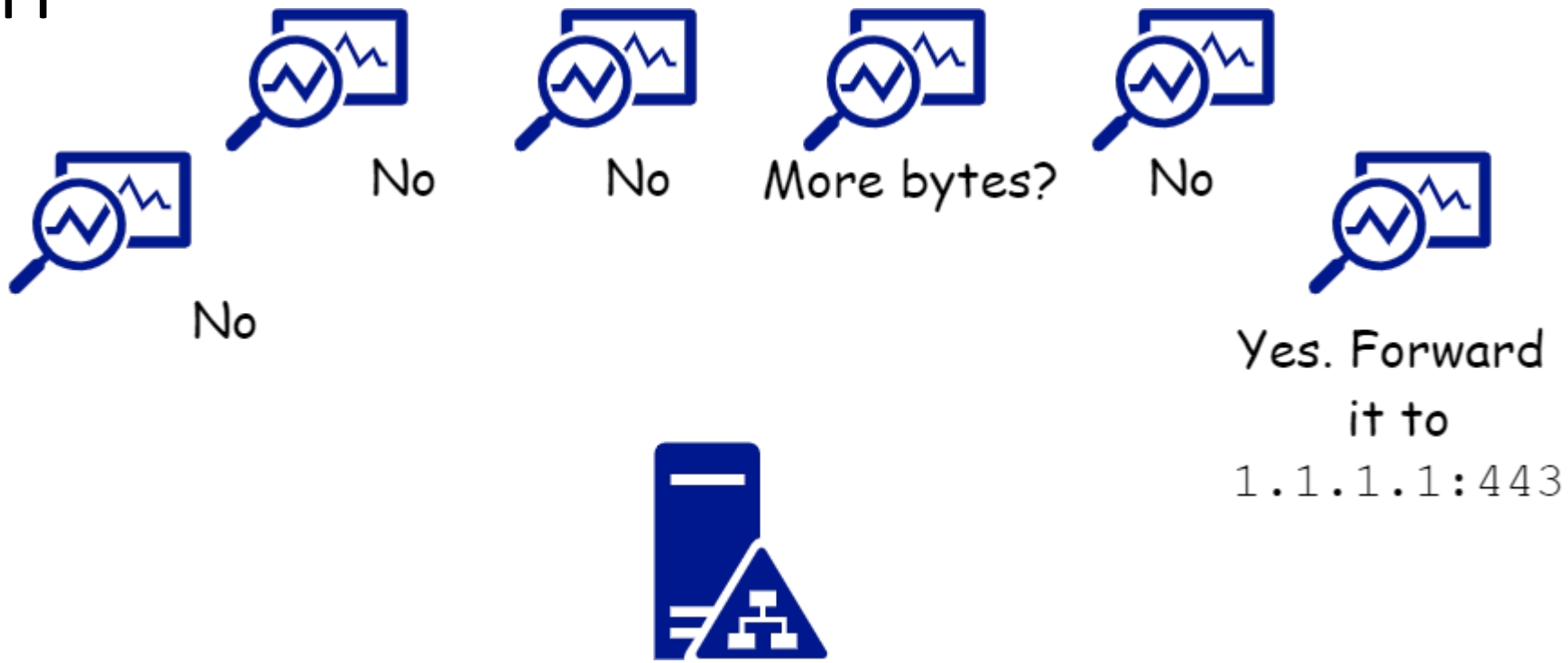
Design



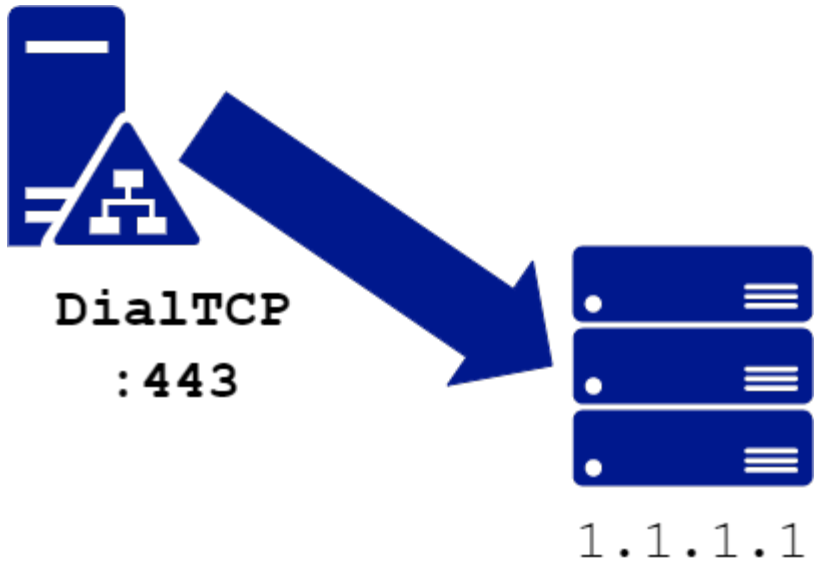
Design



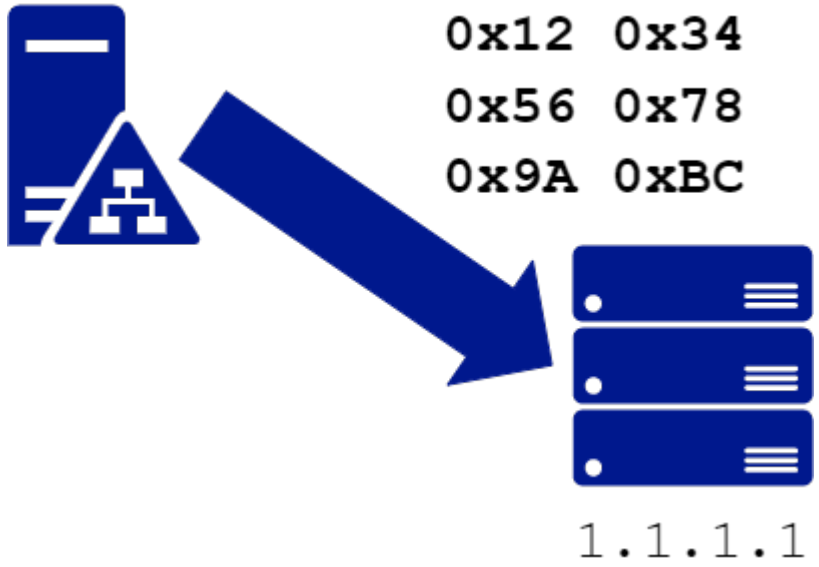
Design



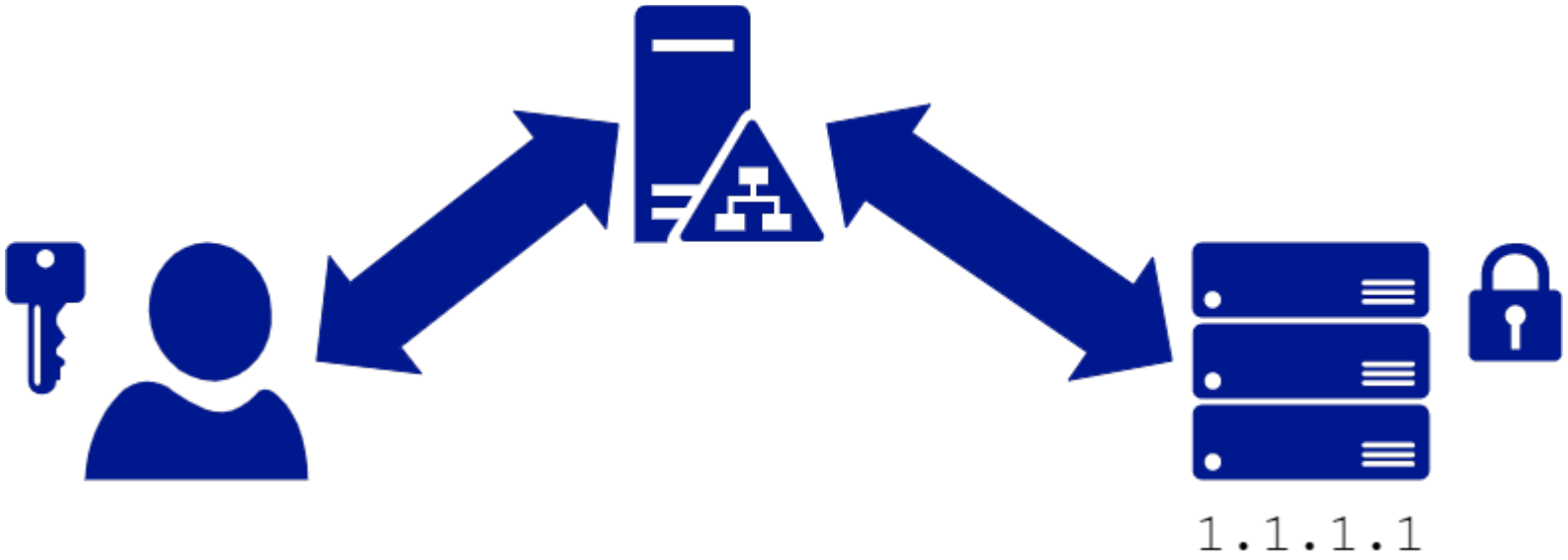
Design



Design

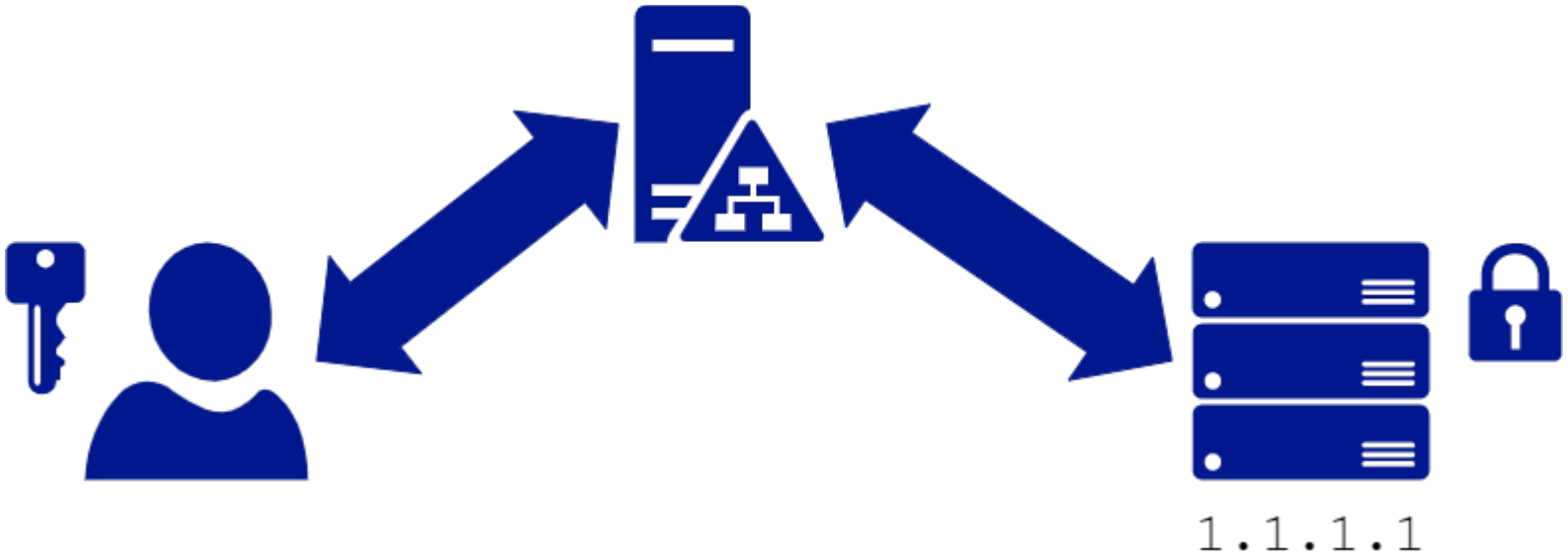


Design



Design

End-to-End Encryption Available!



Implementation

- Configuration File
  - Standard JSON Format
- Config File Structure
  - Version
  - Server Group
    - Server
      - Protocol
        - Rules
          - Action

```
{
    "version": "v0.2.0",
    "servers": {
        "127.0.0.1:443": {
            "TLS": {
                "SNI gaukas.wang": {
                    "action": "FORWARD",
                    "to_addr": "185.199.111.153:443"
                },
                "APLN h2": {
                    "action": "FORWARD",
                    "to_addr": "1.1.1.1:443"
                },
                "CATCHALL": {
                    "action": "REJECT"
                }
            },
            "CATCHALL": {
                "CATCHALL": {
                    "action": "FORWARD",
                    "to_addr": "neverssl.com:443"
                }
            }
        }
    }
}
```

Implementation

- Connection Handler
  - Creates `Server`
  - Creates a `Protocol Manager` for each `Server`
  - Asks `Protocol Manager` to match connection to `Action`
  - Applies matched `Action` to the connection
    - REJECT: close the connection
    - FORWARD: copy (or zero-copy?) all bytes from the client to a remote target
    - …more possible actions

Implementation

- Protocol Manager
  - Keeps the config including the mapping from `Rules` to `Actions`
  - Keeps a list of known `Protocols`, configure them with their `Rules`
  - When asked to match bytes to `Action`
    - Ask all known `Protocols`: Is this byte stream matching any of your known `Rules`? And what is that `Rule`?
    - As soon as a Protocol returns a `Rule`, look for the corresponding `Action`
    - Return the `Action` to caller

Application Scenario

- (Virtual) Gateway Integration
  - Enables IP-layer forwarding to Passthru to preserve Client IP
- Flexible Reverse Proxy/IPLC/IEPL Integration
  - Better utilizations of shared infrastructures
- Better Security
  - Hides services from probes and consequential attacks

Recap

- Passthru: a protocol omni-multiplexor
- Prototype
  - Go based, unprivileged user space application
  - Open-source under **GPL 3.0** on **GitHub** (github.com/gaukas/passthru)
- Application

Recap

- Passthru: a protocol omni-multiplexor
- Prototype
  - Go based, unprivileged user space application
  - Open-source under **GPL 3.0** on **GitHub** (github.com/gaukas/passthru)
- Application
- Questions?