

# Passthru: Protocol Omni-multiplexer

CSCI-ECEN 5273 Network Systems, 2022 Fall

Gaukas Wang, Parker Morr, Sharon Moses Jangam

University of Colorado Boulder

# Special Thanks

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

# Overview

- Background – Shared Infrastructures in Cloud Computing
- Original Problem – inefficiency in proxying TLS traffics
- A smarter way – Passthru
  - Design
  - Implementation
  - Application Scenario

# Background

- Infrastructure-as-a-Service (IaaS)
  - Provides physical Datacenter, Servers, Networking and Storage
  - The base of Cloud Computing
- Platform-as-a-Service (PaaS)
  - Above IaaS, providing Middleware, Operating Systems, DevTools, etc
  - Augments the user experience, optimizes the service quality
- International Private Leased Circuit (IPLC)
- International Ethernet Private Line (IEPL)

# Original Problem

- Transport Layer Security (TLS)
  - A popular cryptographic protocol based on certificates signed by CA
  - Communication with integrity, authenticity and privacy per se
  - The foundation of modern HTTP-over-TLS (<https://>)
  - Works (mostly) with TCP, with UDP adaptations DTLS and QUIC(HTTP3)
- TLS provides a strong guarantee for end-to-end encryption
  - Unauthorized Man-in-the-Middle (MITM) is trivially identifiable

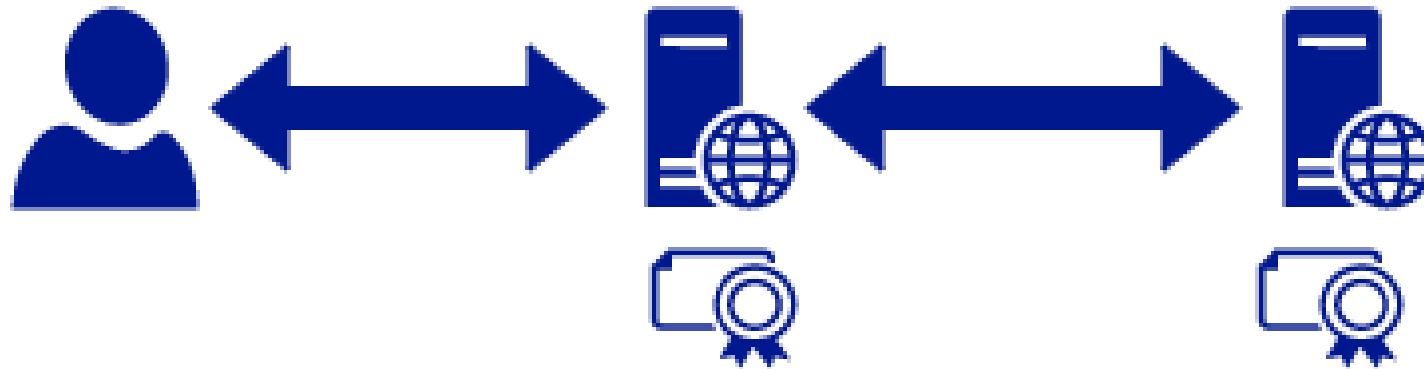
# Traditional ways to proxy TLS

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



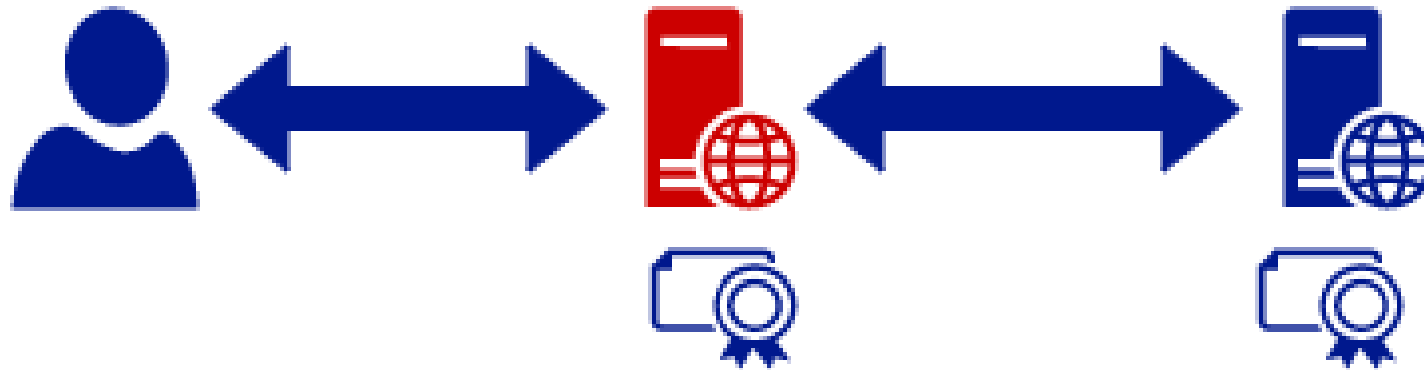
# Traditional ways to proxy TLS

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



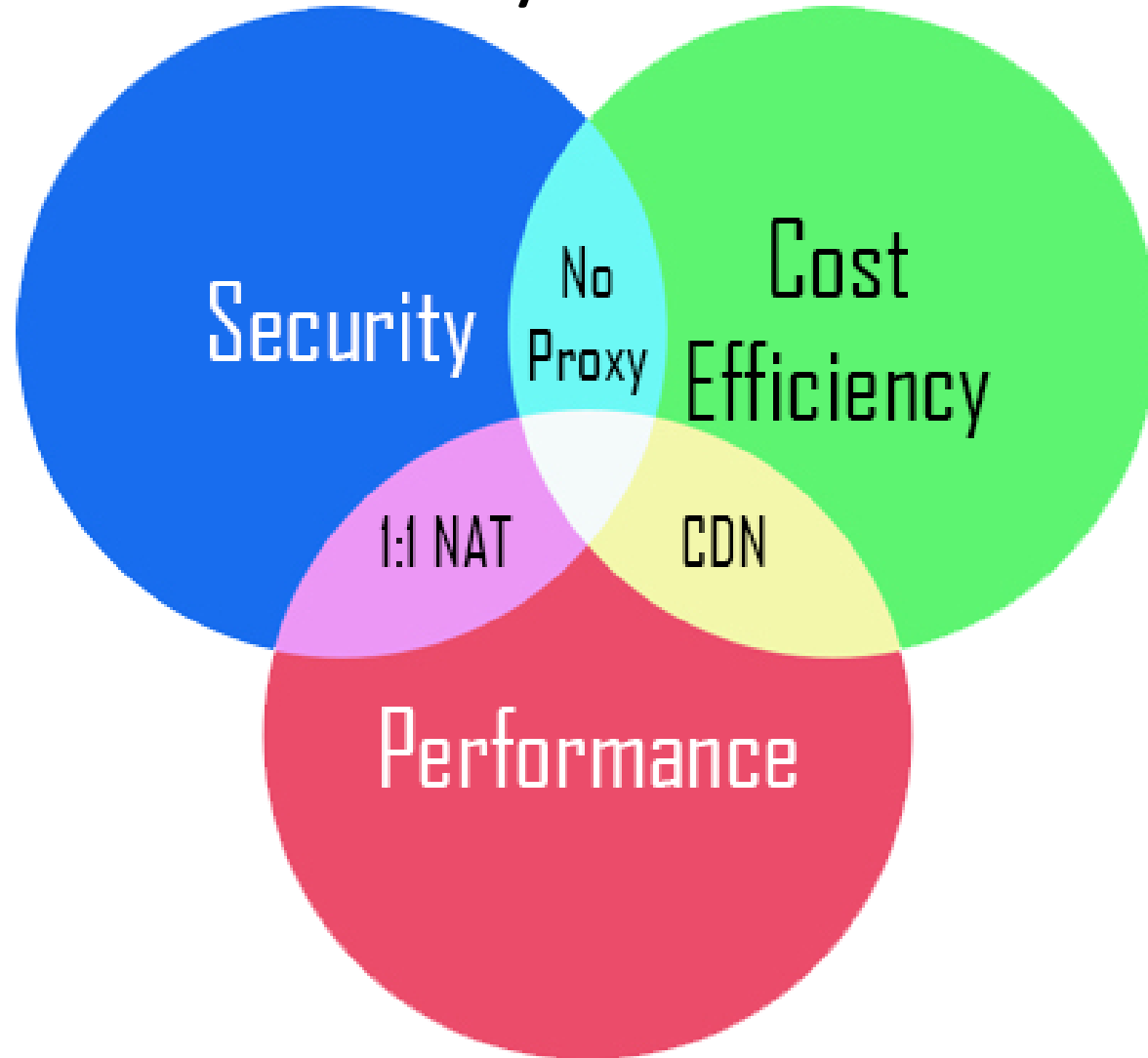
# Traditional ways to proxy TLS

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

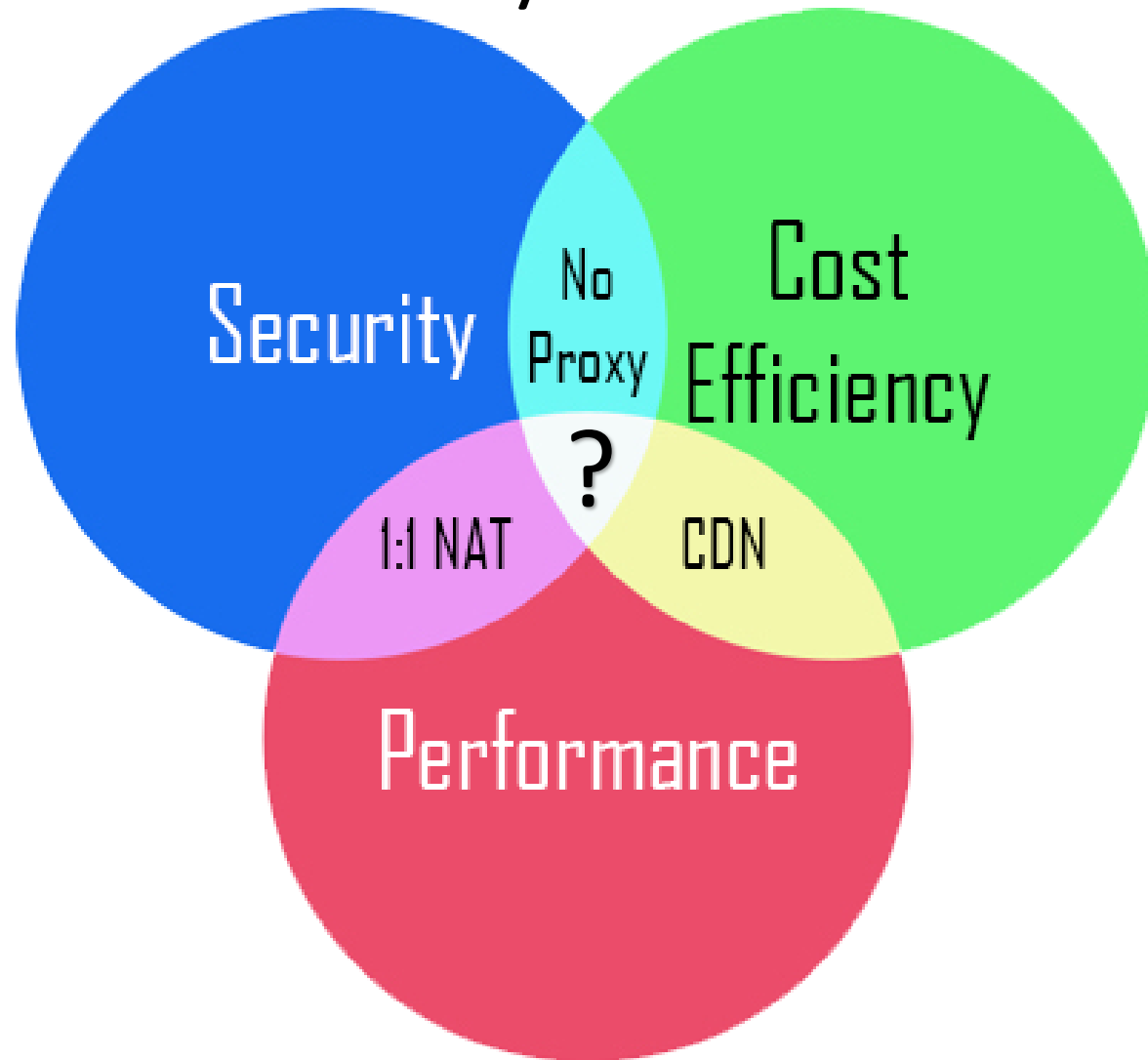




# Trilemma: Security vs. Cost vs. Performance



# 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 Identification MUST be deterministic and exclusive
  - Protocol identification SHOULD happen at line speed

# Passthru

- A DPI-based Protocol Omni-multiplexer
  - DPI: for Application Data Sniffing
  - Protocol: identified at Transport Layer or lower
  - Omni: Highly programmable/customizable
- Prototype
  - Unprivileged user space application at Transport Layer written in Go
  - TCP Protocols only
  - TLS Protocol Detection included in application, AND
  - Allows custom pluggable Detection Modules to be added by user

# Definition

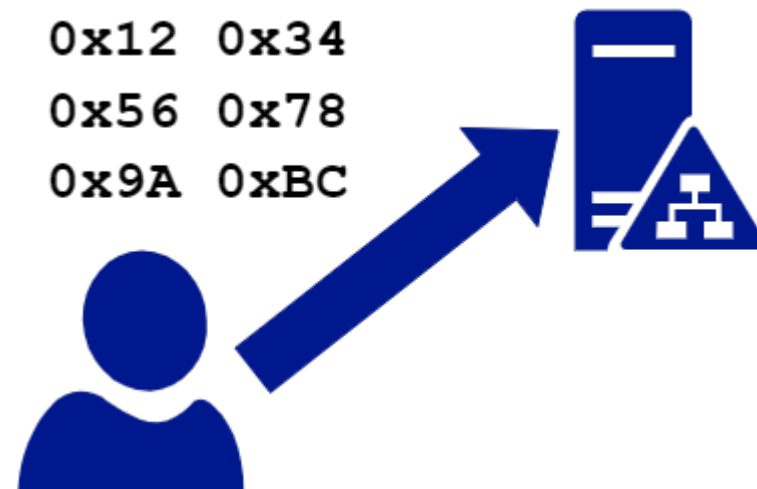
- `Server`
  - Accepts incoming connections
  - Dials outgoing connections
  - Relaying data between clients and destination hosts
- `Protocol`
  - Performs DPI on incoming bytes
  - Tells `Server` if matching



# Design



# Design



# Design



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



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



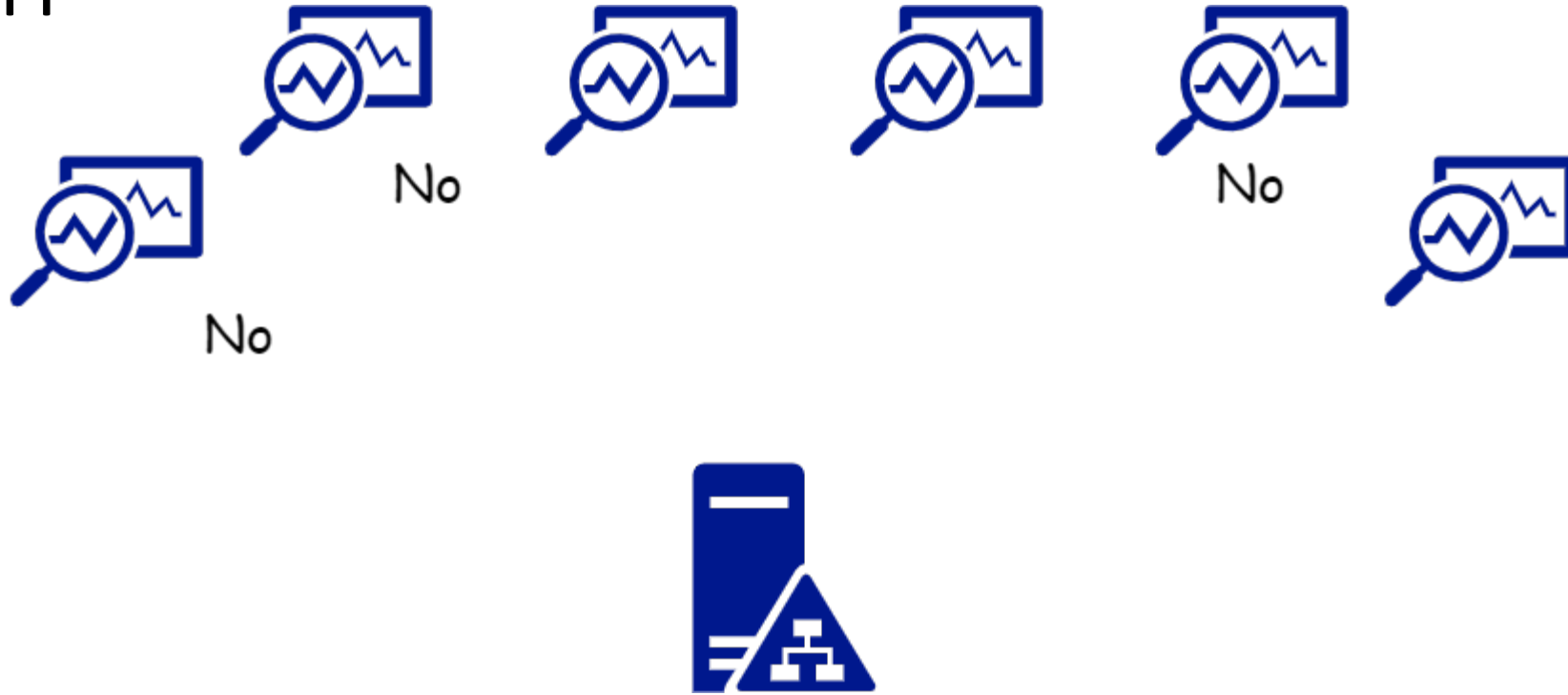
# Design



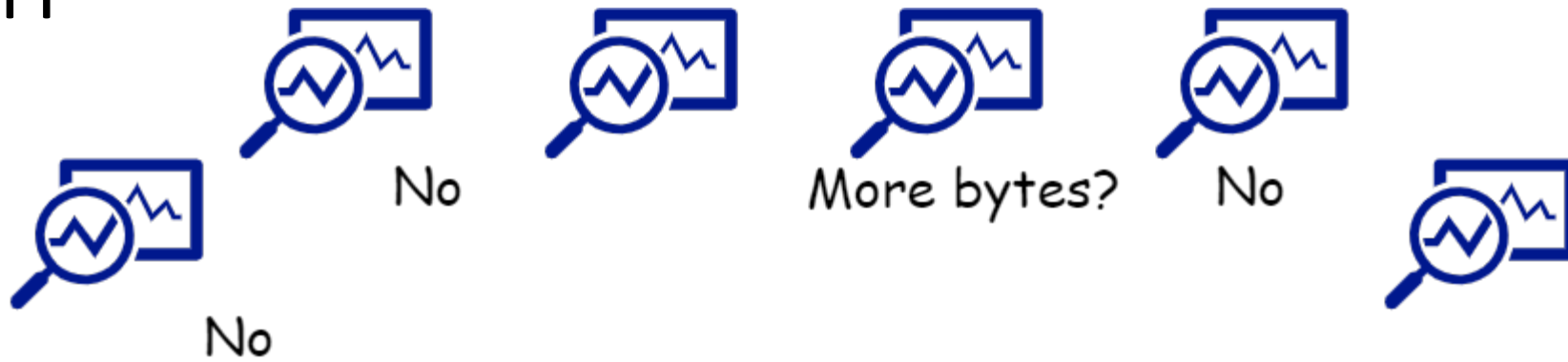
# Design



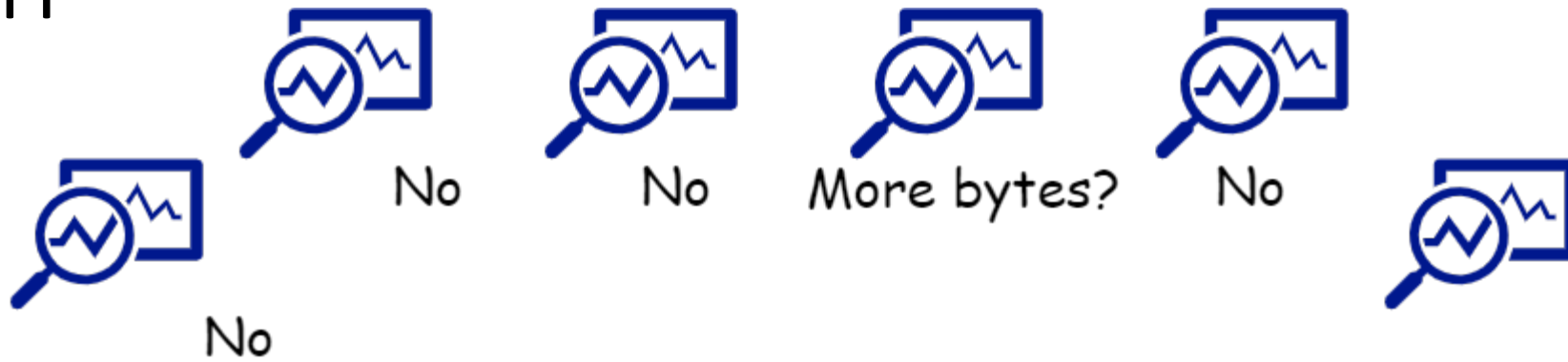
# Design



# Design



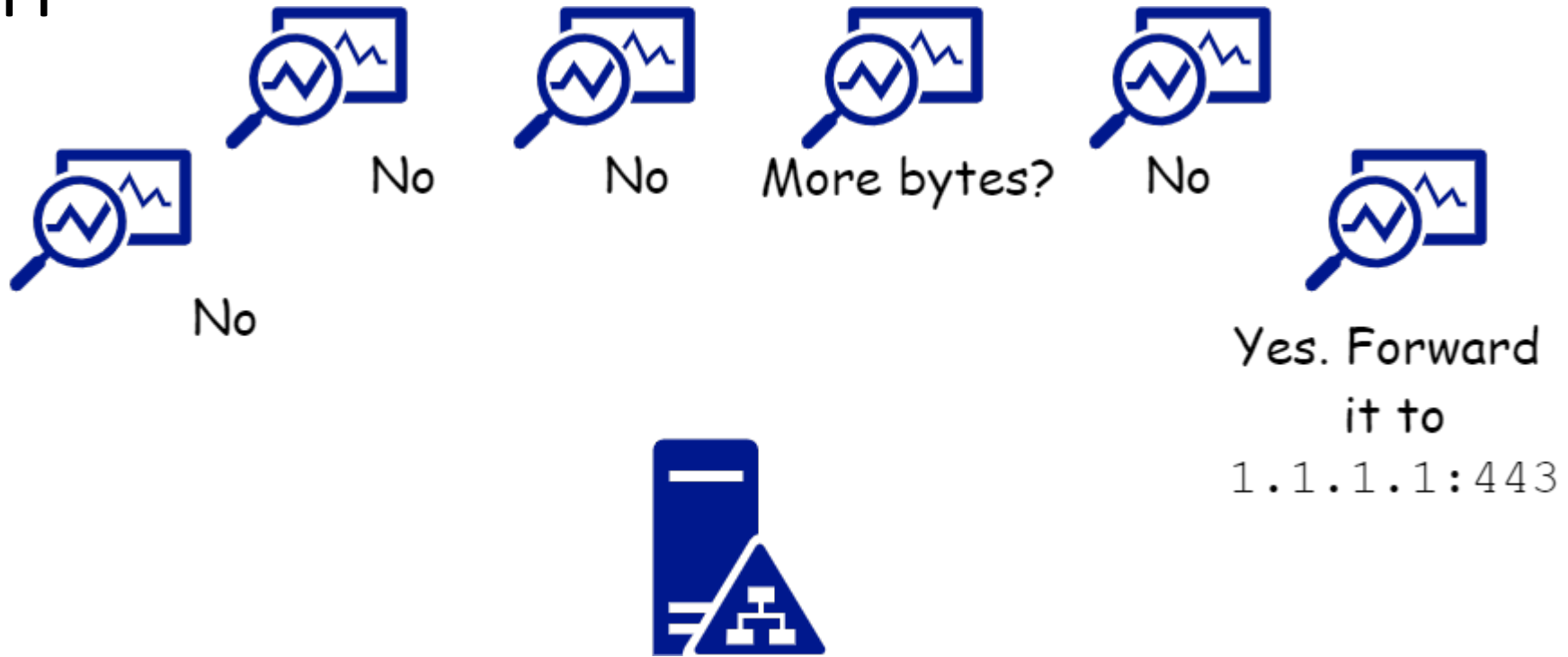
# Design



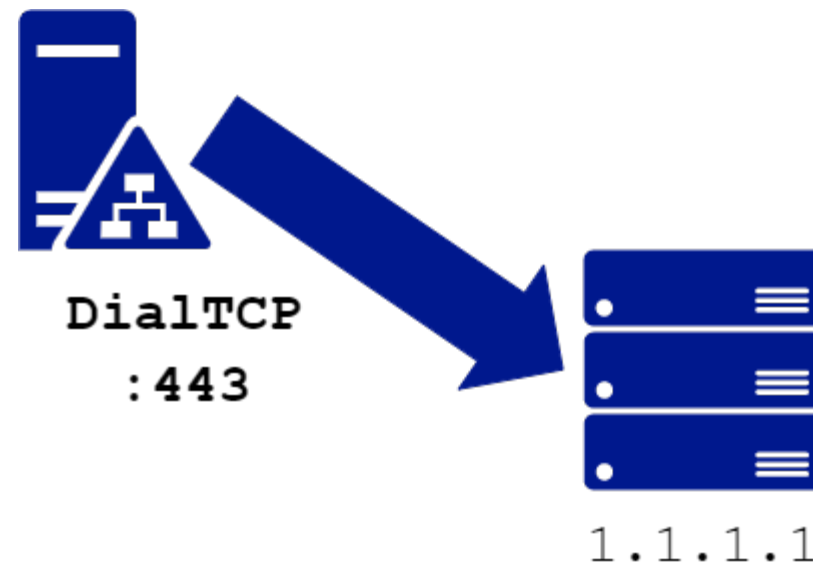
More bytes?



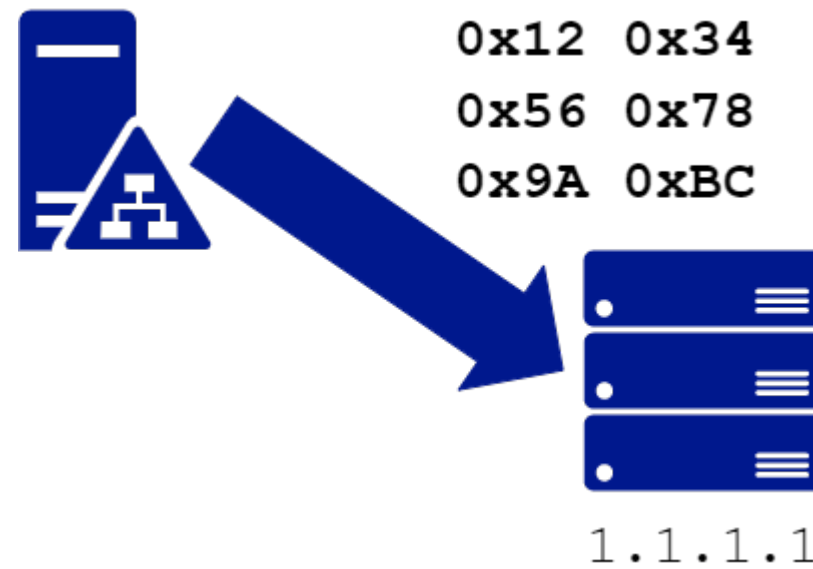
# Design



# Design

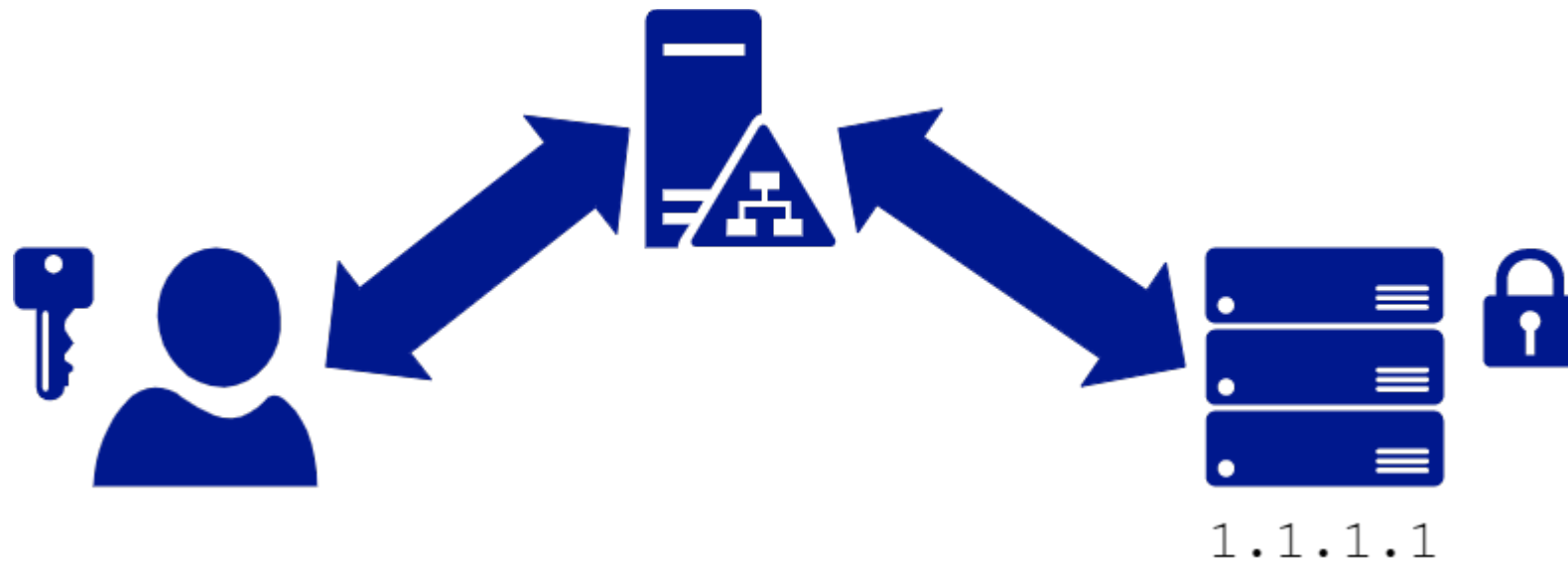


# Design





# Design



# Implementation

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

```
1  {
2      "version": "v0.2.0",
3      "servers": {
4          "127.0.0.1:443": {
5              "TLS": {
6                  "SNI_gaukas.wang": {
7                      "action": "FORWARD",
8                      "to_addr": "185.199.111.153:443"
9                  },
10                 "APLN h2": {
11                     "action": "FORWARD",
12                     "to_addr": "1.1.1.1:443"
13                 },
14                 "CATCHALL": {
15                     "action": "REJECT"
16                 }
17             },
18             "CATCHALL": {
19                 "CATCHALL": {
20                     "action": "FORWARD",
21                     "to_addr": "neverssl.com:443"
22                 }
23             }
24         }
25     }
26 }
```

# 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 IPLC/IEPL Integration
  - Shares IP addresses on premium infrastructure
- 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](https://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](https://github.com/gaukas/passthru))
- Application
- Questions?