Understanding Distributed Denial of Service (DDoS) Attacks

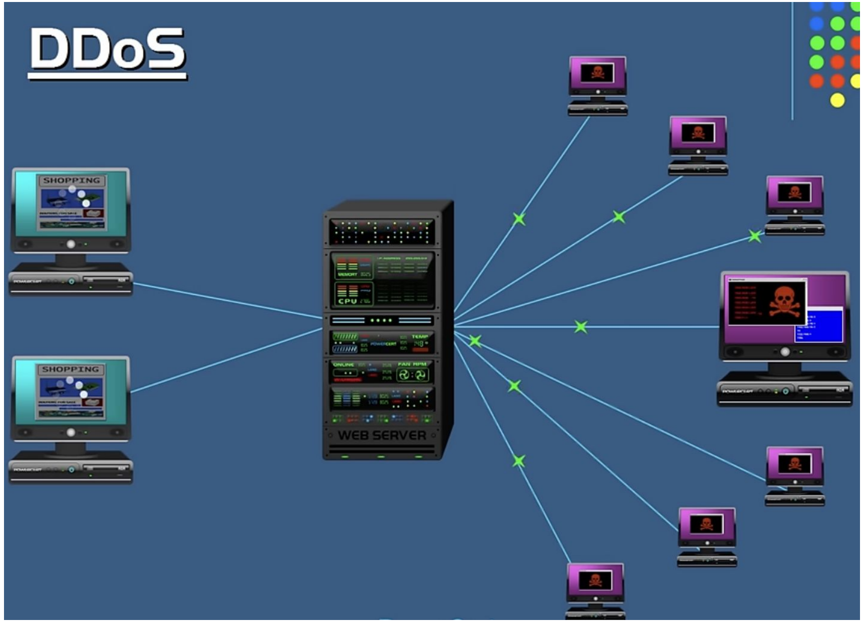
Attack Types, Historical Background, Case Studies, and Future Trends

Presented by: Mohammed, Kiet, Vivek, Sai, Zhengguan, Ziyue

Research

What is a DDoS Attack?

- DDoS = Distributed Denial of Service
- Attempts to make an online service unavailable by flooding it with traffic
- Uses multiple sources (botnets) to overwhelm the target
- Hard to distinguish from legitimate traffic because mimics user behavior



Types of DDoS Attacks

Volumetric Attacks - **Focus:** Bandwidth
Goal: Saturate the internet connection of the target by flooding it with high-volume traffic
How: Uses amplification (DNS/NTP) or floods (UDP, ICMP) to create a huge amount of traffic

Protocol Attacks - **Focus:** Network infrastructure and server protocols
Goal: Exploit weakness in Layer 3 and 4 protocols (IP, TCP) to consume server resources like connection tables or firewalls
How: Send malformed, partial or spoofed requests (SYN Flood, Ping of Death)

Application Layer Attacks (Layer 7) - **Focus:** Web applications and servers (Layer 7)
Goal: Overload app (website or API) by mimicking legitimate user actions
How: Send HTTP requests or stimulate user behavior to exhaust CPU, memory, or database connections by mimicking legitimate user behavior (HTTP Flood, Slowloris)

Historical Background of DDoS Attacks

- **First Recorded DDoS Attack (1999):**.
  - University of Minnesota targeted via *Trinoo*, a tool using hijacked servers to flood the network — marking the first recorded DDoS incident.
- **Early Notable DDoS Attacks (2000):**
  - Michael Calce ("Mafiaboy"), age 15, disrupted Yahoo!, eBay, CNN, Amazon, and Dell using a botnet. Sites were offline for hours, exposing major vulnerabilities.
- **Common Early Methods:**
  - *Trinoo, TFN, Stacheldraht*: Used to launch coordinated attacks
  - *SYN Floods*: Incomplete TCP handshakes overwhelmed servers
  - *UDP Floods*: Excessive UDP packets overloaded systems
  - *ICMP Floods*: Ping storms exhausted network capacity

Case Study – Dyn Attack (2016)

Company: Dyn (DNS Provider)

Attack Method: Mirai botnet using 100,000 IoT devices

Impact: Disrupted sites: Amazon, Twitter, Netflix, PayPal. Major North America/Europe outage.

Estimated Cost: $110M U.S. economy loss

Case Study – Spamhaus Attack (2013)

Company: Spamhaus (Anti-spam org)

Attack Method: DNS Amplification

Peak Traffic: 300 Gbps

Impact: Slowed down parts of the internet

2024 DDoS Trends and Emerging Threats

- **Increased Attack Volume and Frequency**
  - Web-based (Layer 7) DDoS attacks have skyrocketed by 550%
  - Network-layer DDoS traffic has expanded by 120%
  - On average, attack durations have increased by 37%
- **Geopolitical Impact**
  - Hacktivist-led attacks surged, especially in conflict zones.
  - **Key targets:** Ukraine, Israel, and the United States
  - The EMEA region: 78% of all Layer 7 attacks
- **Targeted/Most Affected Sectors**
  - **Finance:** Up by 393% in attack volume
  - **Transportation:** Up by 375%
  - **E-commerce:** Increasing by 238%
- **Emerging Threats on the Horizon**
  - AI-driven attack automation is enabling larger, faster, and more adaptive campaigns
  - DNS-based DDoS attacks have seen an 87% spike
  - Attacks on web applications and APIs have risen by 41%
  - Over one-third of malicious traffic exploits known vulnerabilities
- **Security Implications**
  - Static defenses are no longer enough
  - Shift toward adaptive, AI-driven security solutions is essential

Project

DDoS Attack Simulation and Analysis

Understanding Vulnerabilities and Evaluating Mitigation Strategies

Project Goals

- Understand DDoS behaviors and attack patterns
- Simulate real-world DDoS scenarios
- Collect and analyze performance data
- Test and evaluate mitigation strategies
- Deliver a comprehensive report with recommendations

Simulation Environment

Host: Win11

Hypervisor: Oracle Virtualbox

Guest: Kali linux, with 36 CPU cores and 8GB memory

Implementation - network topology

30 attackers and 1 normal user are connected to one server via one switch

```
class DDOSTopo(Topo):
    def build(self):
        self.server = self.addHost('server', ip='10.0.0.1')
        self.normal_user = self.addHost('normal', ip='10.0.0.2')
        self.switch = self.addSwitch('s1')
        self.addLink(self.server, self.switch, cls=TCLink, bw=1)
        self.addLink(self.normal_user, self.switch)

        self.attackers = []
        for i in range(1, 32):
            attacker = self.addHost(f'attacker{i}', ip=f'10.0.0.{i+2}')
            self.addLink(attacker, self.switch)
            self.attackers.append(attacker)
```

Implementation - attacker

Use hping3 to attack the server.

```
def start_attack(host, target_ip):
    attack_cmd = f"hping3 -S -p 80 -d 1200  --flood {target_ip} &"
    host.cmd(attack_cmd)
```

Implementation - server

Open an http service, and collect data using multiple packages

```
def start_server(host):
    host.cmd('python3 -u -c "from http.server import ThreadingHTTPServer, SimpleHTTPRequestHandler; '
             'ThreadingHTTPServer((\\"10.0.0.1\\", 80), SimpleHTTPRequestHandler).serve_forever()" '
             '> /tmp/http_server.log 2>&1 &')
    host.cmd('tcpdump -i server-eth0 tcp -w /tmp/server_traffic.pcap &')
    host.cmd("tcpdump -tt -n -i server-eth0 tcp | awk '{print int($1)}' | uniq -c | "
             "awk '{print strftime(\"%Y-%m-%d %H:%M:%S\", $2) \" packets: \" $1}' > /tmp/packet_count_per_sec.log &")
    host.cmd("tcpdump -nn -i server-eth0 tcp and port 80 -l | awk '{print $3}' | cut -d. -f1-4 | "
             "sort | uniq -c | sort -nr > /tmp/ip_access_count.log &")
    host.cmd("pgrep -f http.server > /tmp/server_pid.txt")
    host.cmd("mpstat -P ALL 1 > /tmp/cpu_memo_usage.log &")
    host.cmd("bwm-ng -t 1000 -o plain -u bytes -I server-eth0 > /tmp/traffic_per_sec.log &")
```

Implementation - normal user

Ping the server every 2 second and check if the pinging is success or not.

Save the log of pinging result

```python
def normal_user_behavior(host, target_ip):
    host.cmd('echo "ICMP ping results with latency in ms" > /tmp/normal_user_latency.log; '
             'while true; do '
             'ping_result=$(ping -c 1 -W 1 ' + target_ip + '); '
             'timestamp=$(date +\"%Y-%m-%d %H:%M:%S\"); '
             'if echo \"$ping_result\" | grep -q \"time=\"; then '
             'latency=$(echo \"$ping_result\" | grep \"time=\" | sed -n \"s/.*time=\\([0-9.]*\\).*/\\1/p\"); '
             'echo \"$timestamp latency: ${latency} ms\" >> /tmp/normal_user_latency.log; '
             'else '
             'echo \"$timestamp latency: timeout\" >> /tmp/normal_user_latency.log; '
             'fi; '
             'sleep 1; '
             'done &')
```

Implementation - whitelist defense

Change the topology, add one switch for whitelist user, limit the rate for the public switch

```
class DDOSTopo(Topo):
    def build(self):
        self.server = self.addHost('server', ip='10.0.0.1')
        self.normal_user = self.addHost('normal', ip='10.0.0.2')
        self.s1 = self.addSwitch('s1')  # whitelist path
        self.s2 = self.addSwitch('s2')  # public path

        # whitelist path
        self.addLink(self.server, self.s1, cls=TCLink, bw=1000)
        self.addLink(self.normal_user, self.s1)

        # attacker path (server second interface)
        self.addLink(self.server, self.s2, cls=TCLink, bw=1000)

        self.attackers = []
        for i in range(1, 32):
            attacker = self.addHost(f'attacker{i}', ip=f'10.0.0.{i+2}')
            self.addLink(attacker, self.s2)
            self.attackers.append(attacker)
```

Implementation - ratelimit defense

Limit the bandwidth consumption of each ip

Ban the ip if the received package is too large

```
class RateLimitHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)
        self.end_headers()
        self.wfile.write(b"GET OK\n")
        log(f"GET from {self.client_address[0]}")

    def do_POST(self):
        length = int(self.headers.get('Content-Length', 0))
        if length > MAX_REQUEST_SIZE:
            self.send_response(413)
            self.end_headers()
            self.wfile.write(b"Payload Too Large\n")
            log(f"BLOCKED POST from {self.client_address[0]}: {length} bytes")
            return

        data = self.rfile.read(length)
        self.send_response(200)
        self.end_headers()
        self.wfile.write(b"POST OK\n")
        log(f"POST from {self.client_address[0]}: {length} bytes")

    def log_message(self, format, *args):
        return
```

Example of raw data

CPU usage

```
03:09:46 AM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
03:09:47 AM  all   23.65    0.00   65.77    0.00    0.00    4.43    0.00    0.00    0.00    6.15
...............................................................................................
03:09:47 AM   30   33.33    0.00   32.10    0.00    0.00    2.47    0.00    0.00    0.00   32.10
03:09:47 AM   31   13.33    0.00   27.78    0.00    0.00   15.56    0.00    0.00    0.00   43.33
03:09:47 AM   32    8.24    0.00    9.41    0.00    0.00    0.00    0.00    0.00    0.00   82.35
03:09:47 AM   33   36.96    0.00   30.43    0.00    0.00    5.43    0.00    0.00    0.00   27.17
03:09:47 AM   34   28.92    0.00   37.35    0.00    0.00    2.41    0.00    0.00    0.00   31.33
03:09:47 AM   35   11.54    0.00   26.92    0.00    0.00   26.92    0.00    0.00    0.00   34.62
```

Example of raw data

IP access counter

```
3646 10.0.0.27
1790 10.0.0.3
1022 10.0.0.1
 190 10.0.0.18
 105 10.0.0.29
  45 10.0.0.9
  28 10.0.0.21
  27 10.0.0.8
  23 10.0.0.24
  20 10.0.0.25
  15 10.0.0.6
  12 10.0.0.16
   7 10.0.0.17
   6 10.0.0.26
   6 10.0.0.23
   6 10.0.0.13
   5 10.0.0.28
   1 10.0.0.32
   1 10.0.0.22
   1 10.0.0.15
   1 10.0.0.14
   1 10.0.0.12
```

Example of raw data

User pinging latency

```
ICMP ping results with latency in ms
2025-03-26 03:09:40 latency: 2.56 ms
2025-03-26 03:09:41 latency: 0.487 ms
2025-03-26 03:09:42 latency: 0.037 ms
2025-03-26 03:09:43 latency: 0.038 ms
2025-03-26 03:09:44 latency: 0.035 ms
2025-03-26 03:09:45 latency: 0.027 ms
2025-03-26 03:09:47 latency: timeout
2025-03-26 03:09:49 latency: timeout
2025-03-26 03:09:51 latency: timeout
2025-03-26 03:09:53 latency: timeout
2025-03-26 03:09:55 latency: timeout
2025-03-26 03:09:57 latency: timeout
2025-03-26 03:09:59 latency: timeout
2025-03-26 03:10:01 latency: timeout
2025-03-26 03:10:03 latency: timeout
2025-03-26 03:10:05 latency: timeout
```

Example of raw data

Packet counter

```
2025-03-26 03:09:45 packets: 46
2025-03-26 03:09:46 packets: 200
2025-03-26 03:09:47 packets: 203
2025-03-26 03:09:48 packets: 196
2025-03-26 03:09:49 packets: 205
2025-03-26 03:09:50 packets: 200
2025-03-26 03:09:51 packets: 198
2025-03-26 03:09:52 packets: 200
2025-03-26 03:09:53 packets: 202
2025-03-26 03:09:54 packets: 201
2025-03-26 03:09:55 packets: 110
```

Example of raw data

Server traffic

```
================================================================================
    server-eth0:           74.72 KB/s             3.46 KB/s            78.18 KB/s
/         iface                    Rx                    Tx                 Total
================================================================================
    server-eth0:          122.58 KB/s             5.69 KB/s           128.27 KB/s
-         iface                    Rx                    Tx                 Total
================================================================================
    server-eth0:          121.73 KB/s             5.99 KB/s           127.72 KB/s
\         iface                    Rx                    Tx                 Total
================================================================================
    server-eth0:          121.09 KB/s             5.68 KB/s           126.77 KB/s
|         iface                    Rx                    Tx                 Total
================================================================================
```

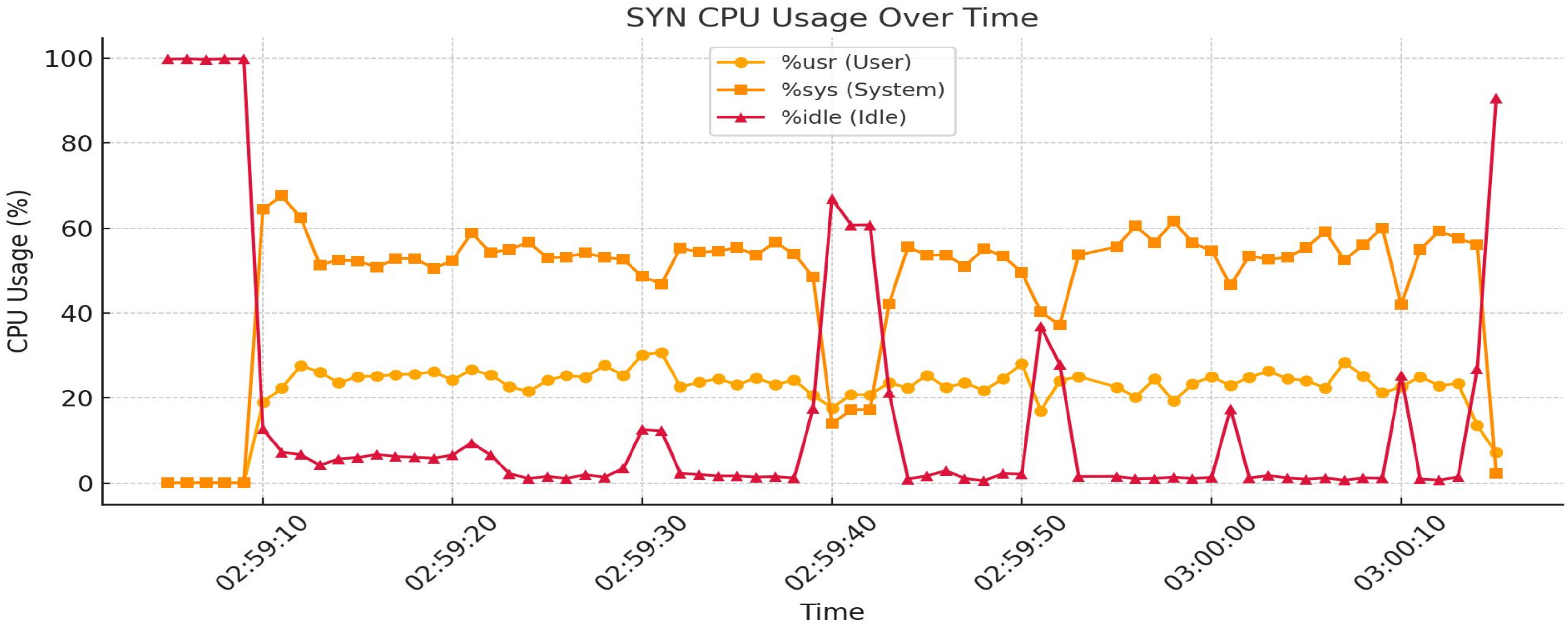
Example of raw data

Server access data

```
02:59:09.983022 IP 10.0.0.7.2588 > 10.0.0.1.80: Flags [S], seq 2102619075:2102620275, win 512, length 1200: HTTP
02:59:09.983042 IP 10.0.0.1.80 > 10.0.0.7.2588: Flags [R.], seq 0, ack 2102620276, win 0, length 0
02:59:09.983088 IP 10.0.0.7.2589 > 10.0.0.1.80: Flags [S], seq 1794435557:1794436757, win 512, length 1200: HTTP
02:59:09.983091 IP 10.0.0.1.80 > 10.0.0.7.2589: Flags [R.], seq 0, ack 1794436758, win 0, length 0
02:59:09.994730 IP 10.0.0.7.2590 > 10.0.0.1.80: Flags [S], seq 424284683:424285883, win 512, length 1200: HTTP
02:59:09.996669 IP 10.0.0.1.80 > 10.0.0.7.2590: Flags [R.], seq 0, ack 424285884, win 0, length 0
02:59:10.000951 IP 10.0.0.7.2591 > 10.0.0.1.80: Flags [S], seq 478805148:478806348, win 512, length 1200: HTTP
02:59:10.000965 IP 10.0.0.1.80 > 10.0.0.7.2591: Flags [R.], seq 0, ack 478806349, win 0, length 0
02:59:10.016776 IP 10.0.0.7.2592 > 10.0.0.1.80: Flags [S], seq 1354710134:1354711334, win 512, length 1200: HTTP
02:59:10.016843 IP 10.0.0.1.80 > 10.0.0.7.2592: Flags [R.], seq 0, ack 1354711335, win 0, length 0
```

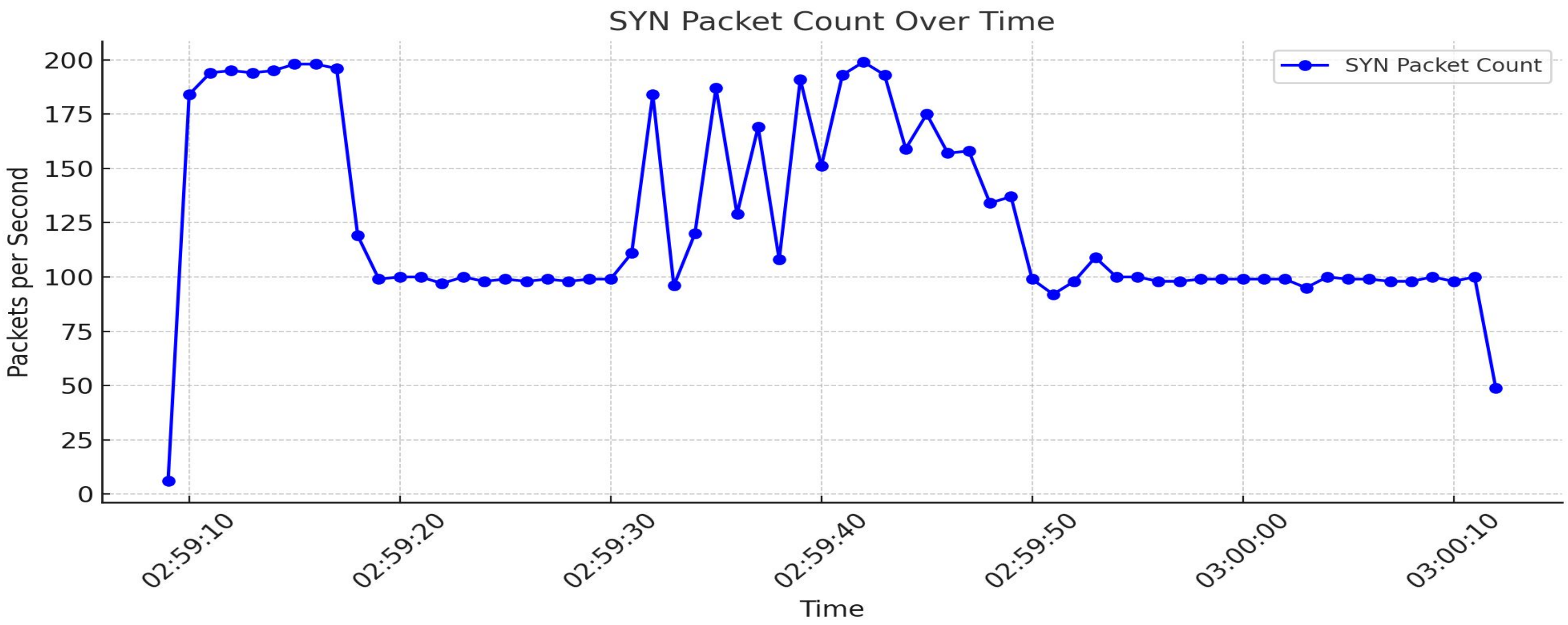
Data Analysis - CPU, Traffic, and Packet Count

Data Analysis - SYN CPU Usage

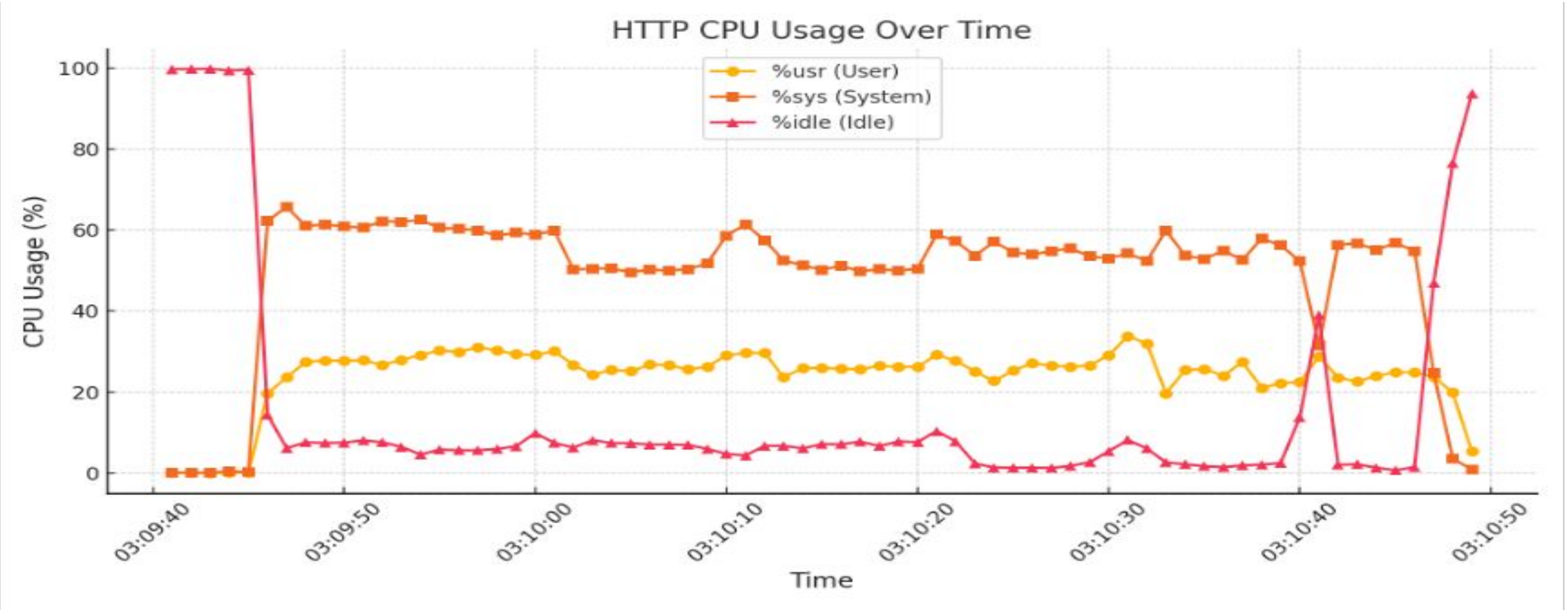


Data Analysis - SYN Traffic

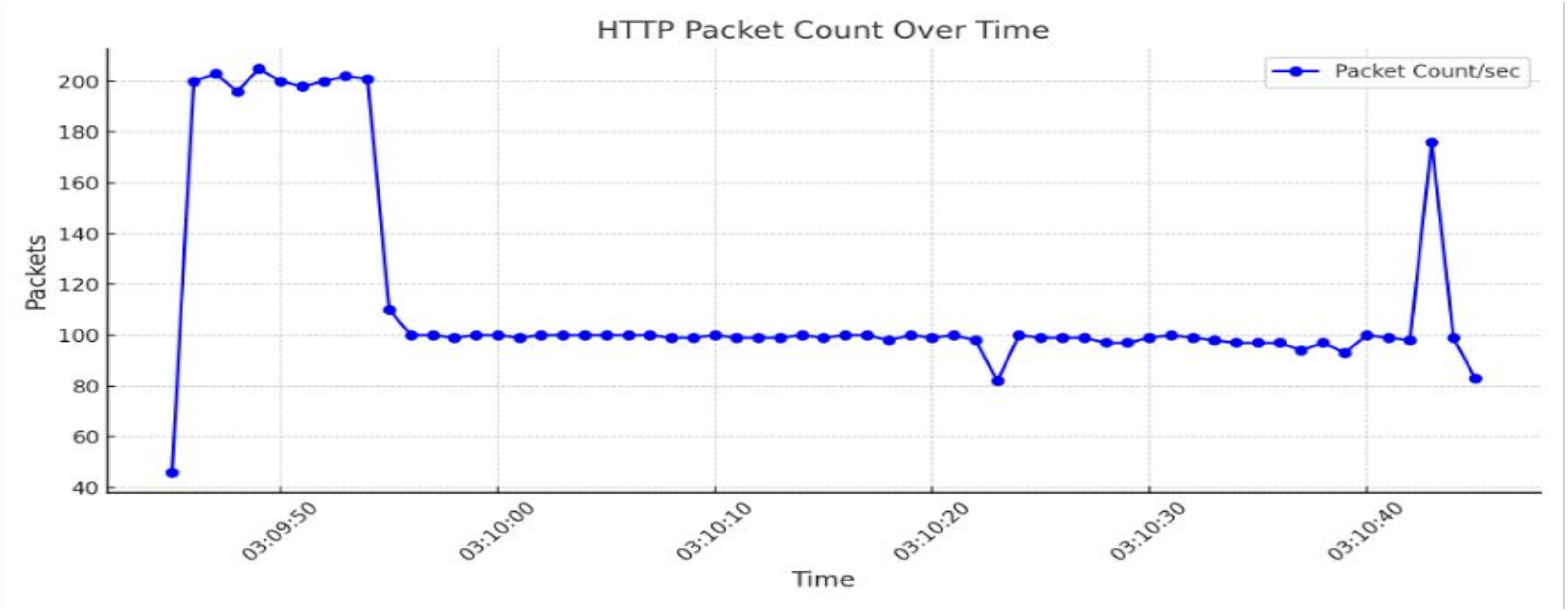
Data Analysis - SYN Packet Count



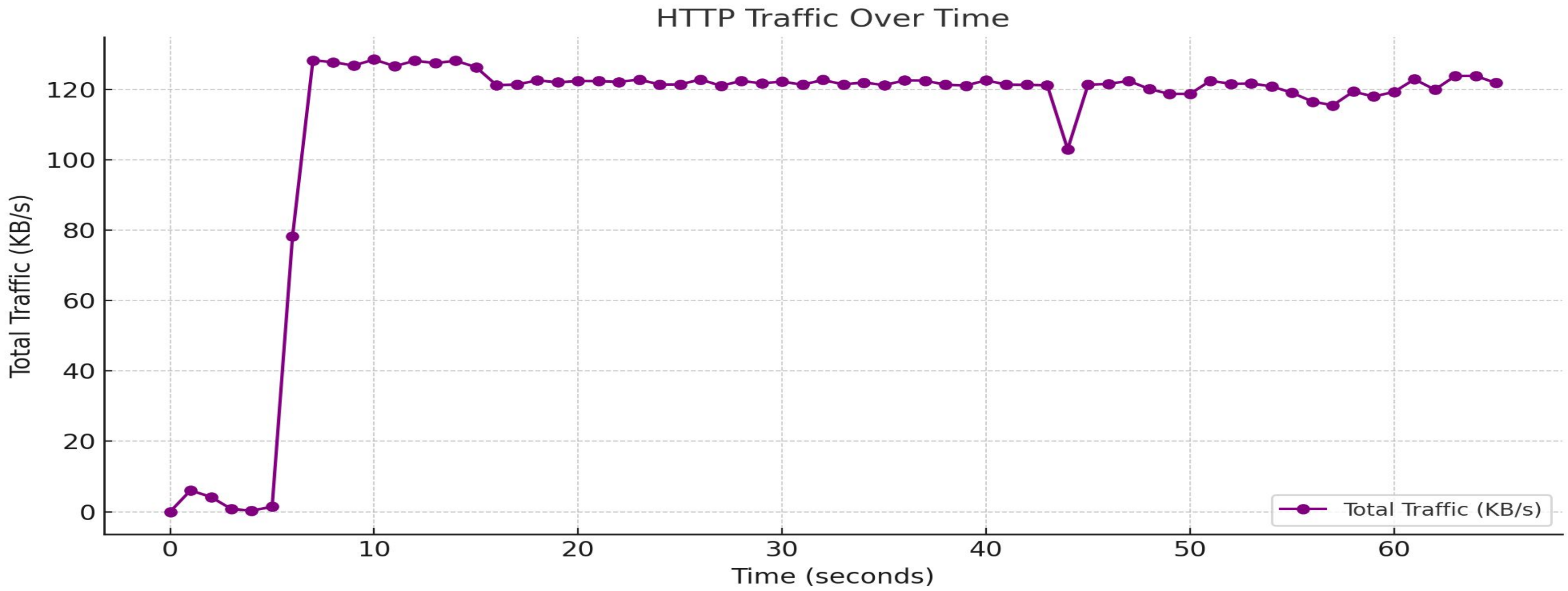
Data Analysis - HTTP CPU Usage



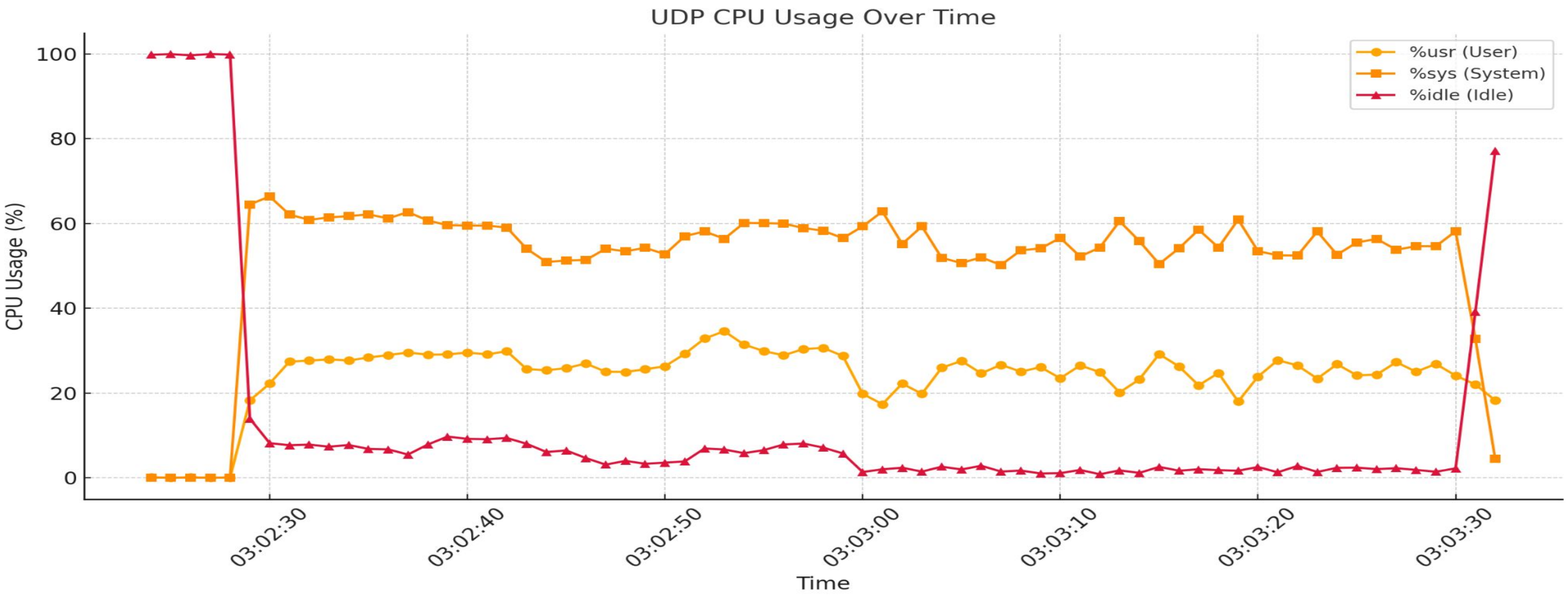
Data Analysis - HTTP Packet Count



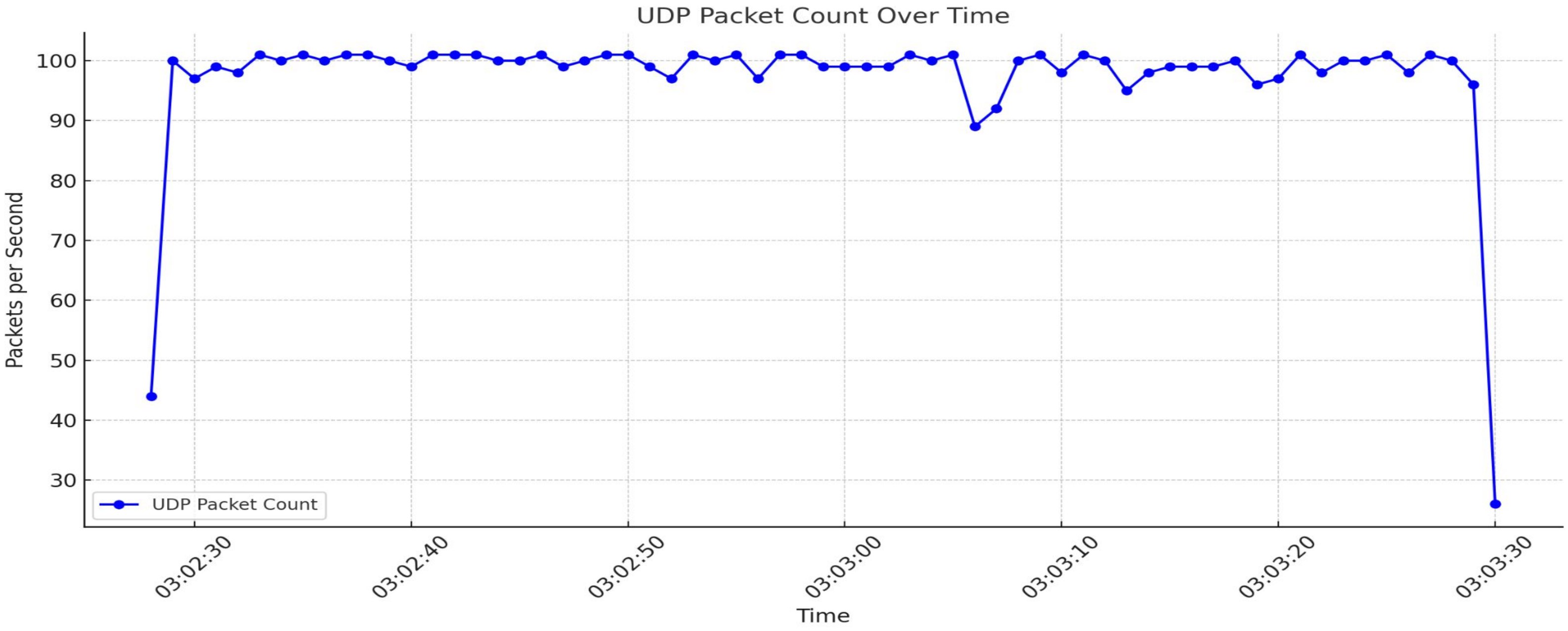
Data Analysis - HTTP Traffic



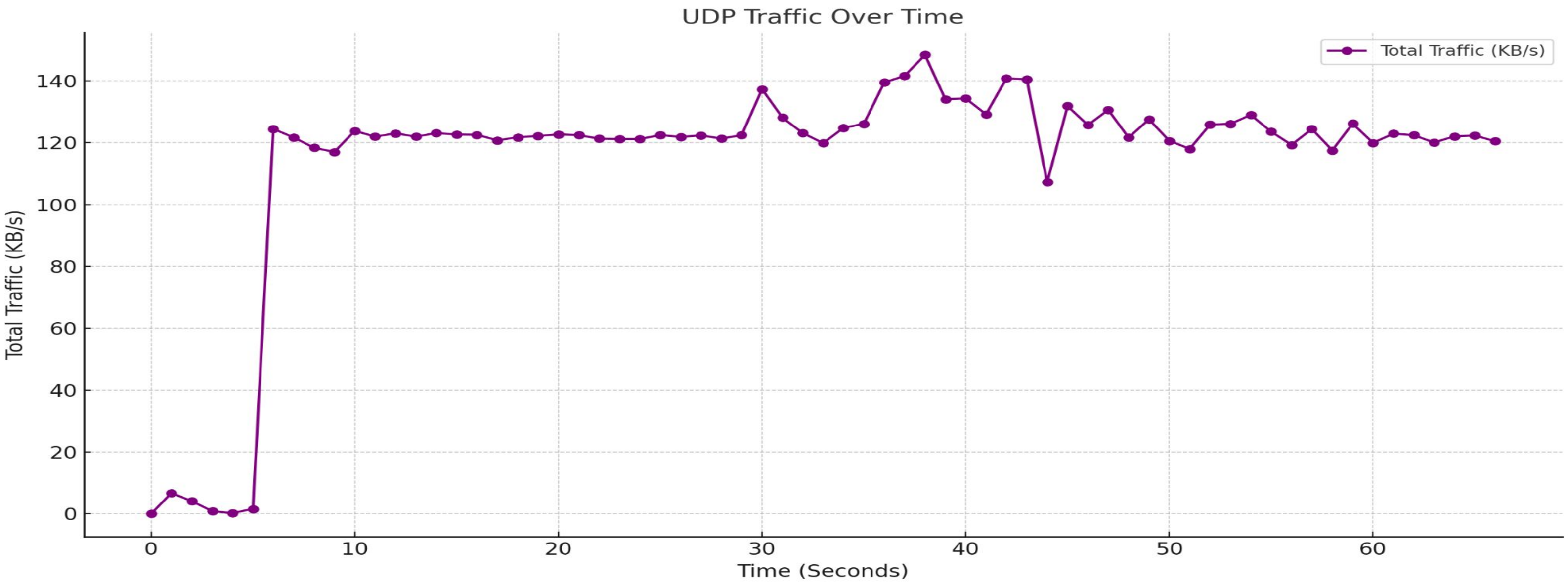
Data Analysis - UDP CPU Usage



Data Analysis - UDP Packet Count

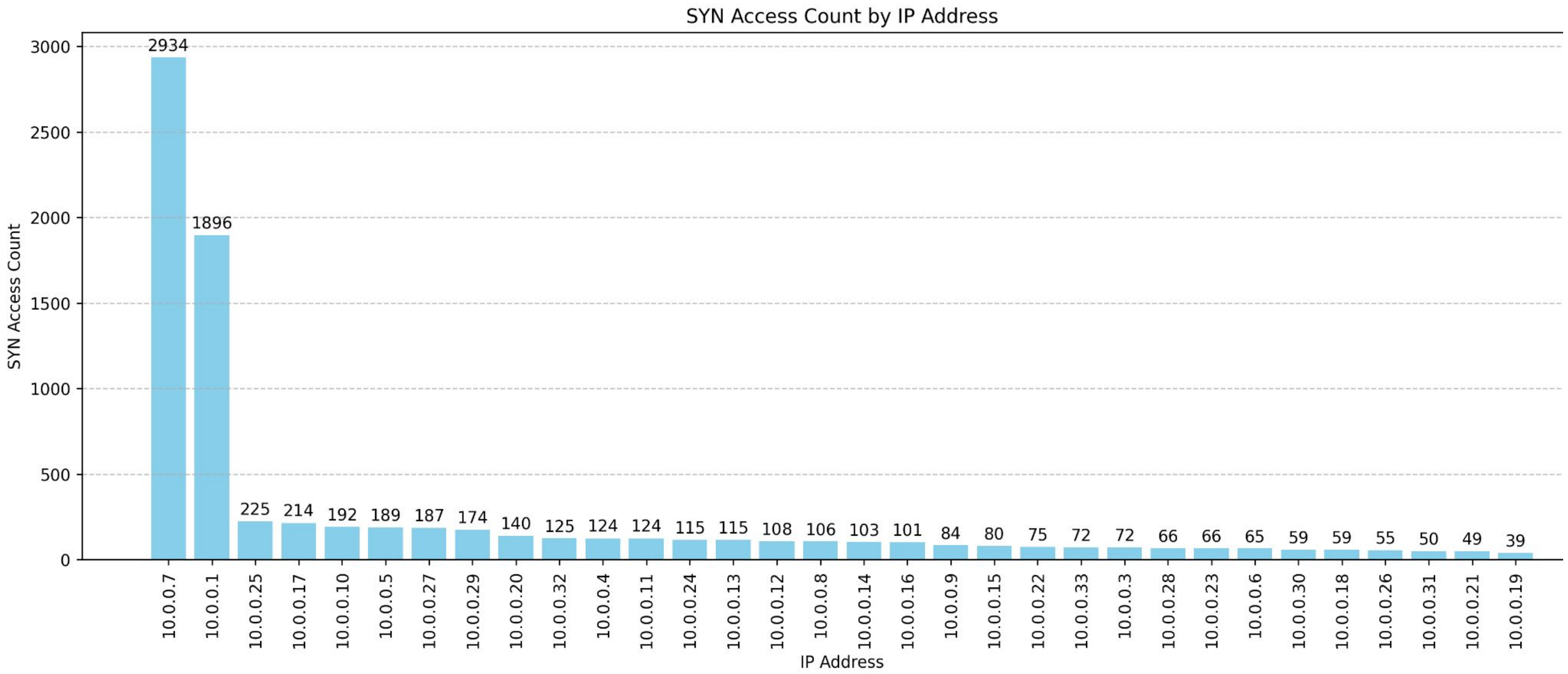


Data Analysis - UDP Traffic

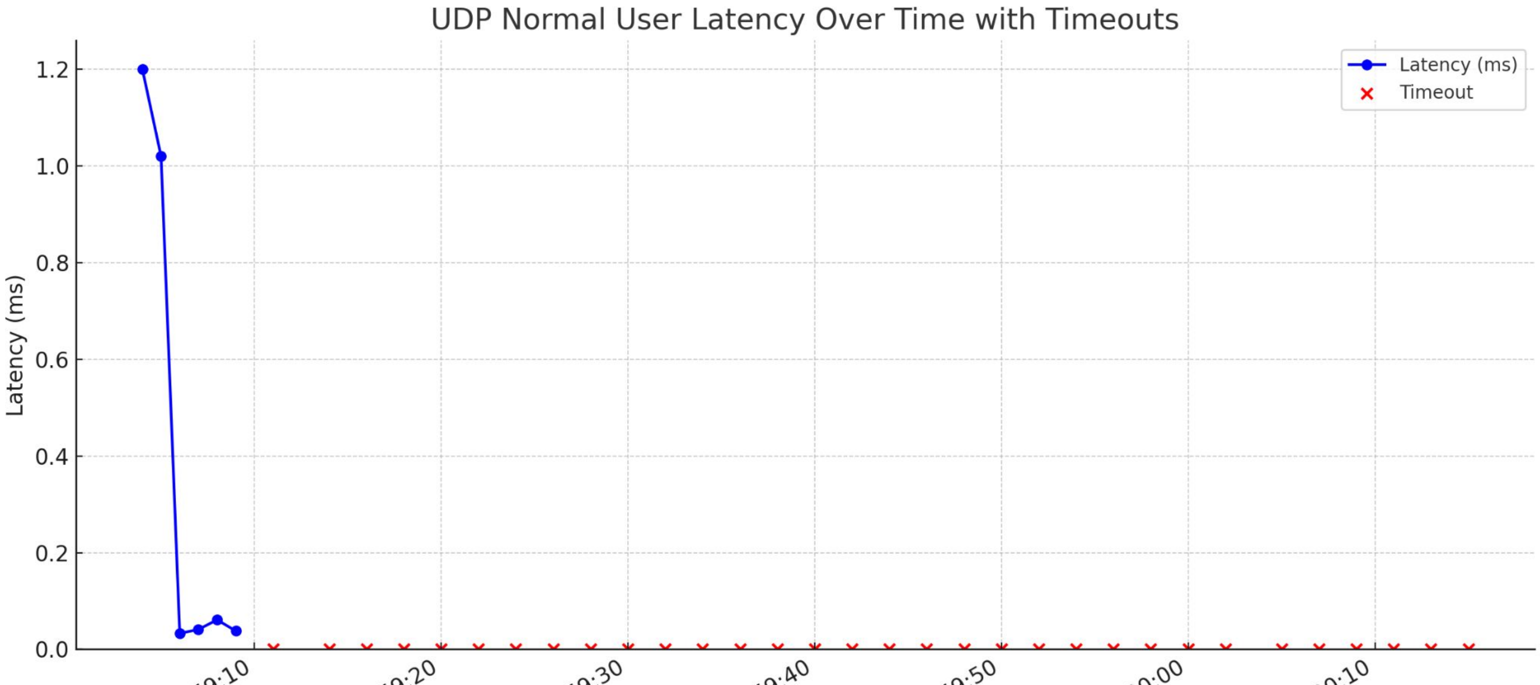


IP Access and User latency

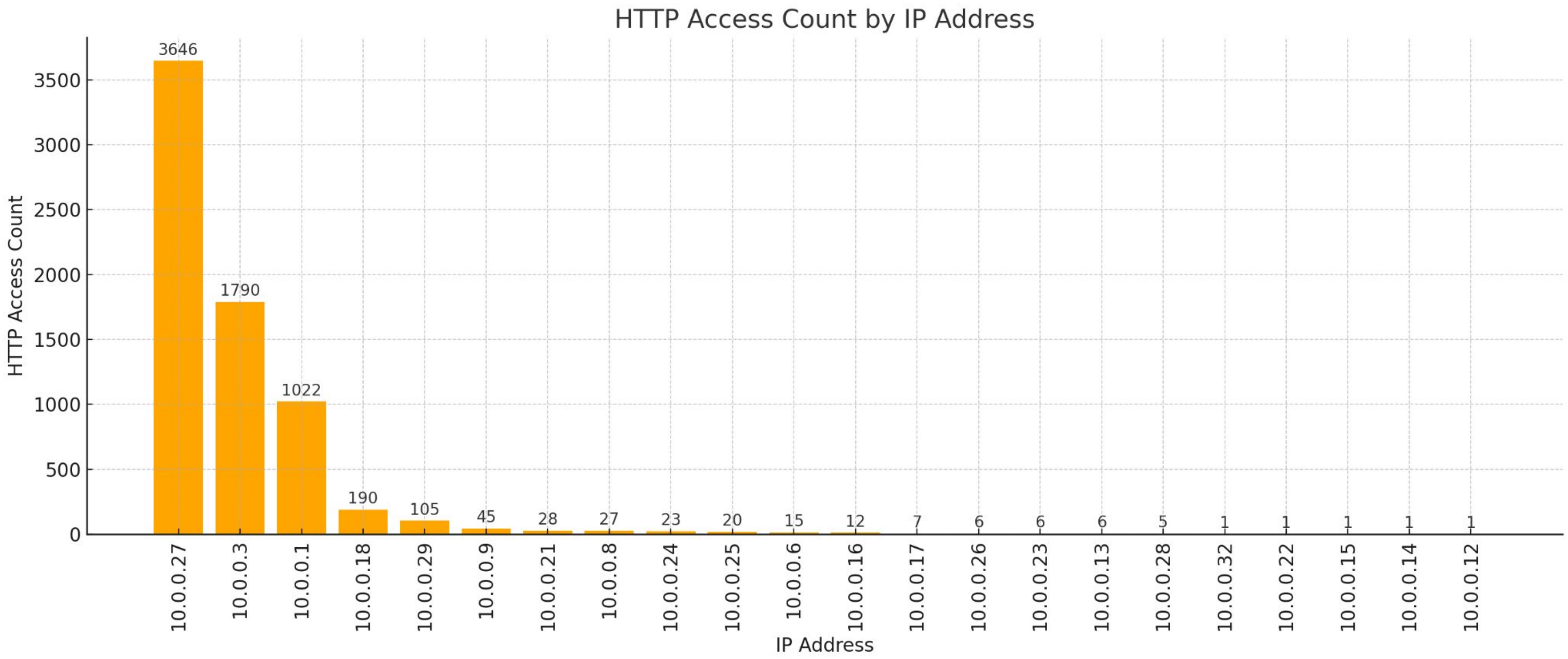
Data Analysis - SYN IP Access



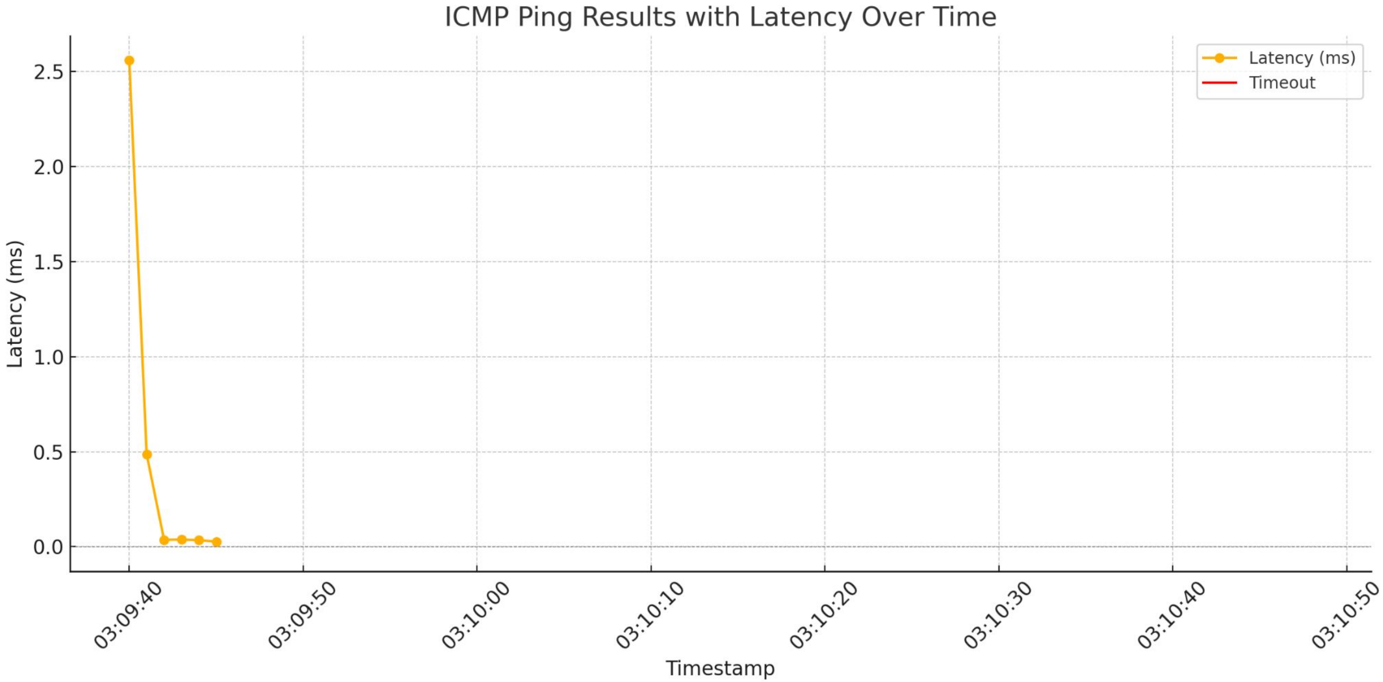
Data Analysis - SYN Latency



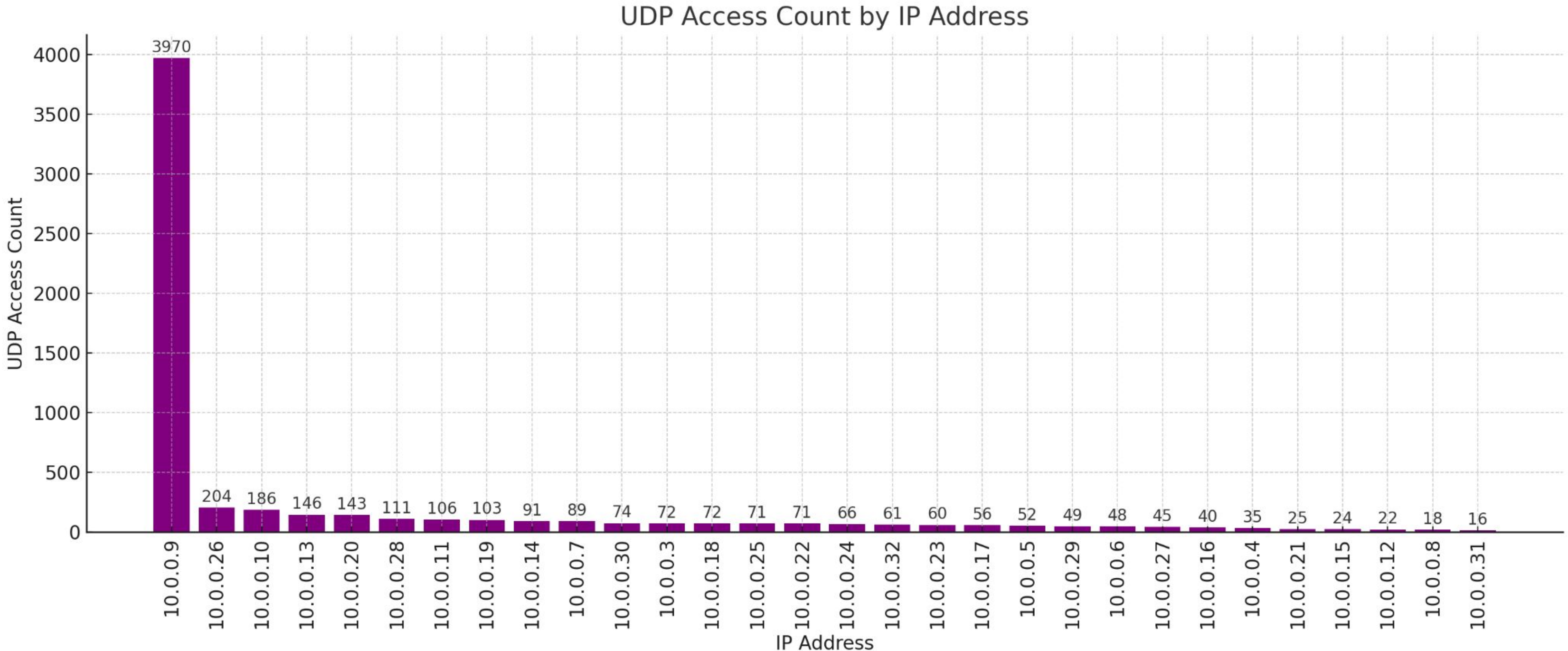
Data Analysis - HTTP IP Access



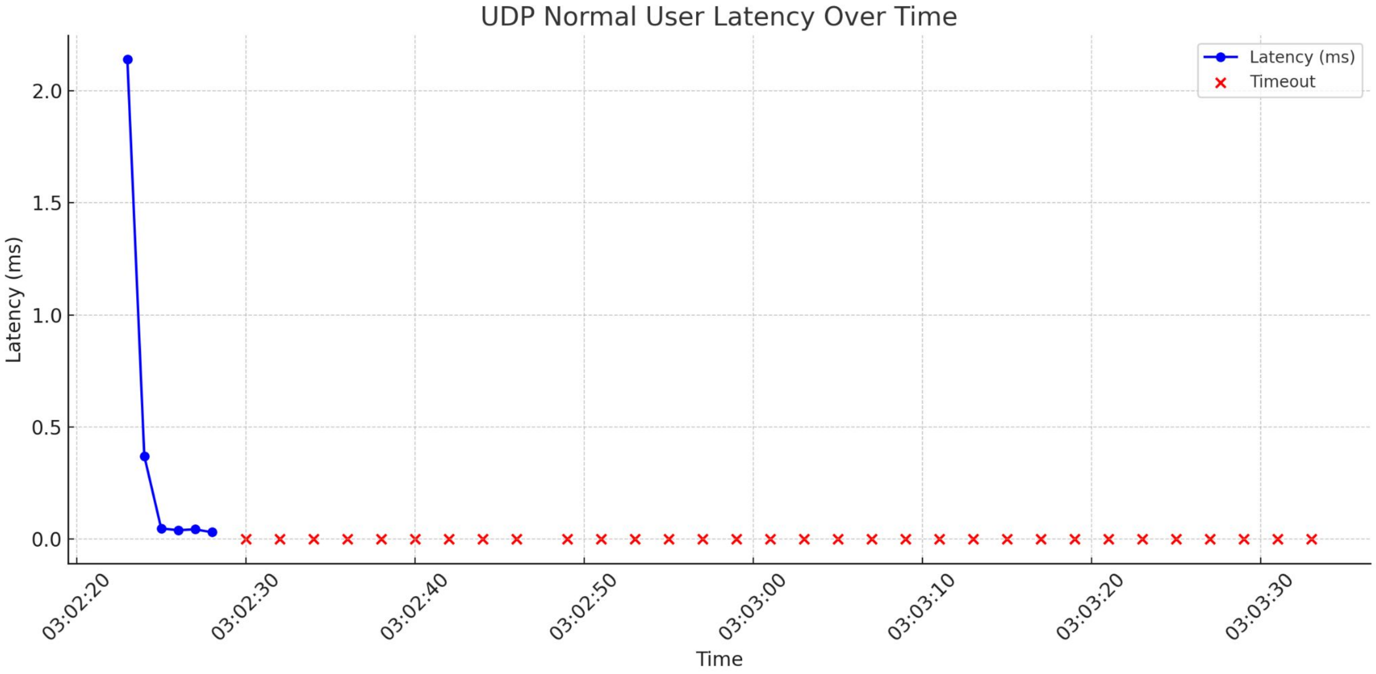
Data Analysis - HTTP Latency



Data Analysis - UDP IP Access



Data Analysis - UDP Latency



Mitigation Strategies - SYN Flood

Description: Attacker sends a flood of TCP SYN packets to exhaust server resources (half-open connections).

Mitigation Techniques:

- **SYN Cookies**: A server-side technique that defers resource allocation until a completed handshake is received.
- **TCP Intercept (Firewall Feature)**: The firewall completes the TCP handshake and only forwards legitimate connections.
- **Rate Limiting**: Restrict the number of SYN packets accepted per second from a single IP or subnet.
- **Connection Timeout Tweaks**: Lower the timeout for half-open connections (e.g., `tcp_syncookies` in Linux).
- **Firewall/IDS Rules**: Use tools like iptables or Snort to detect and drop malicious SYN floods.
- **Reverse Proxies/WAFs**: Offload TCP handshakes to a proxy that can better absorb large connection attempts.

Mitigation Strategies - HTTP Flood

Description: Attacker sends legitimate-looking HTTP GET/POST requests to overload the web server.

Mitigation Techniques:

- **Rate Limiting**: Limit number of requests per second/minute per IP.
- **CAPTCHA/JavaScript Challenges**: Used to verify that a client is human.
- **Bot Detection**: Identify and block headless browsers, suspicious user agents, or IPs with bad reputation.
- **Web Application Firewall (WAF)**: Filters out bad HTTP requests using rules and anomaly scoring.
- **Caching**: Offload frequently accessed content using reverse proxies (e.g., Cloudflare, NGINX).
- **Load Balancing**: Distribute traffic to multiple backend servers to prevent overload.

Mitigation Strategies - UDP

Description: Attacker sends massive amounts of UDP packets to random ports, causing the target to send many ICMP unreachable responses.

Mitigation Techniques:

- **Rate Limiting**: Control the rate of UDP packets per source/destination.
- **Ingress/Egress Filtering**: Block spoofed IP addresses using BCP38 on routers.
- **Blocking Unused Ports**: Drop UDP packets at the firewall for ports not used by legitimate services.
- **Deep Packet Inspection (DPI)**: Detect abnormal UDP packet patterns or payloads.
- **Blackhole Routing**: Drop all traffic destined to the target IP (last resort).

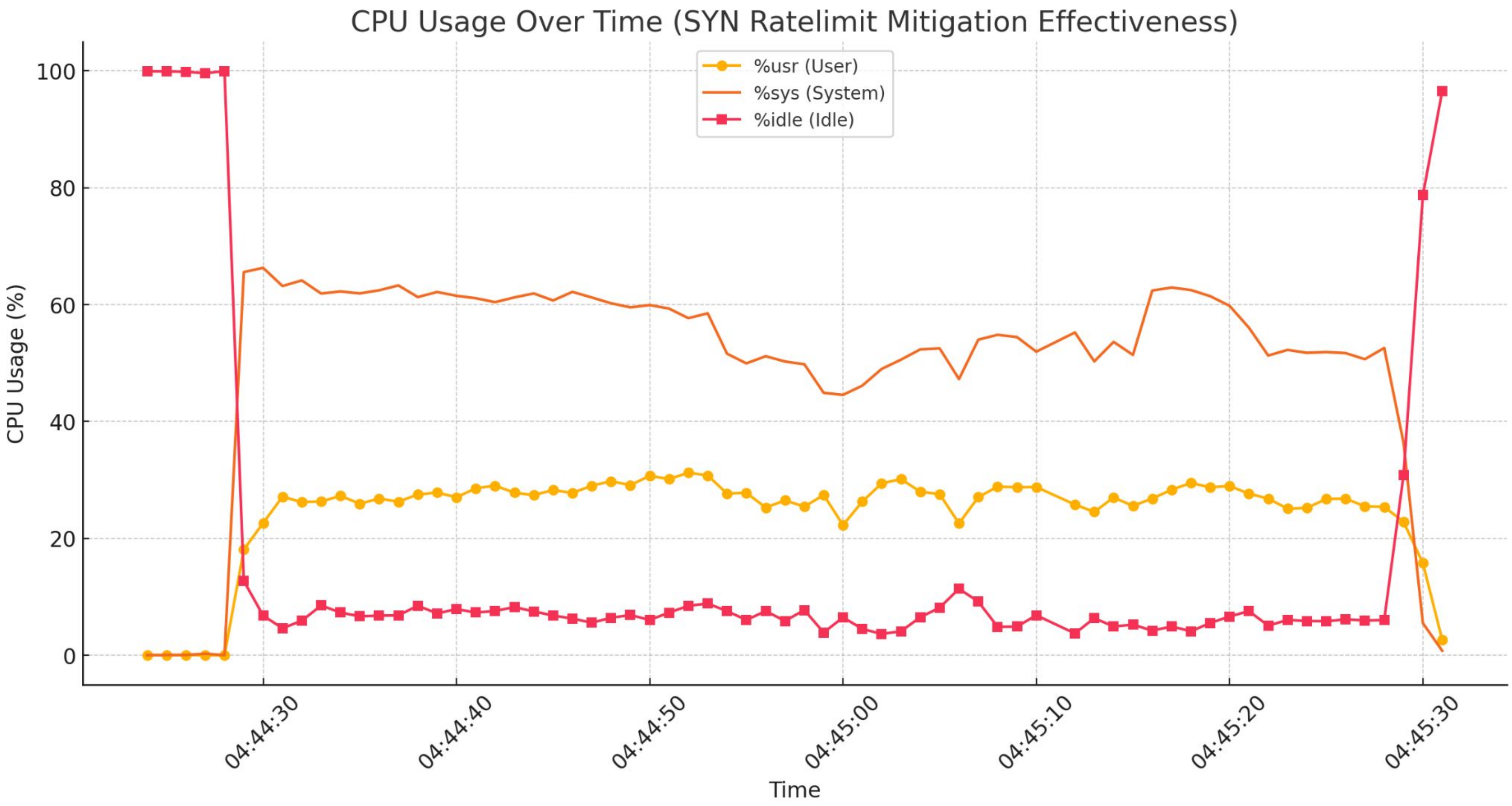
Mitigation Strategies - SYN Rate Limit

Description: A variation or response to SYN floods — attacks try to bypass SYN rate-limiting settings or trigger unintended consequences.

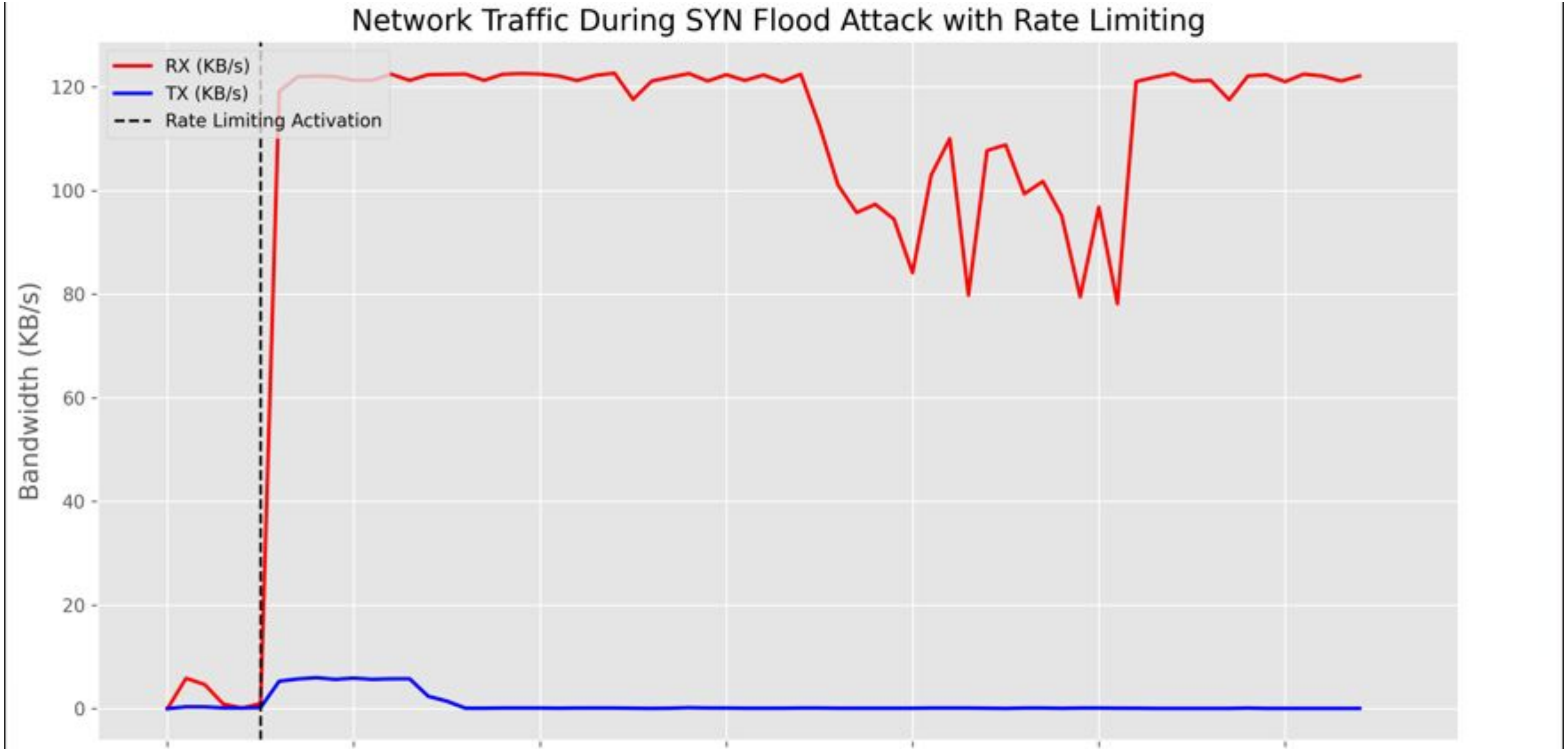
Mitigation Techniques:

- **Fine-Tuned Rate Limits**: Adjust thresholds to allow legitimate traffic while still throttling abusive traffic.
- **Adaptive Algorithms**: Use algorithms like token buckets that dynamically adjust based on traffic behavior.
- **Application-Layer Gateways**: Inspect traffic contextually to allow bursts from known-good clients (e.g., mobile app users).
- **Alerting & Monitoring**: Detect and respond to suspicious traffic patterns around the SYN rate limit.

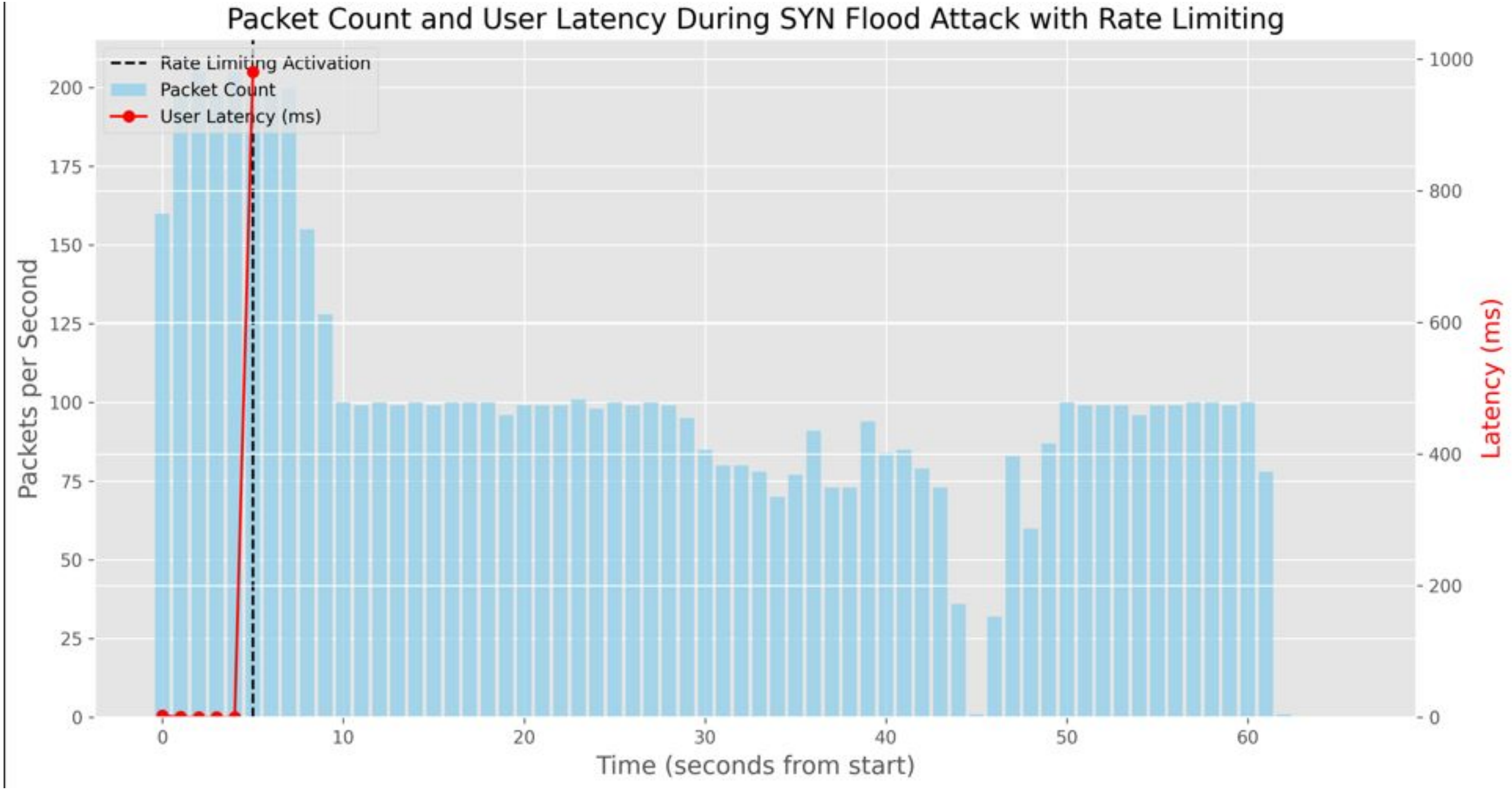
SYN Rate Limit Data - CPU Usage



SYN Rate Limit Data - Network Traffic



SYN Rate Limit Data - Packet Count and Latency



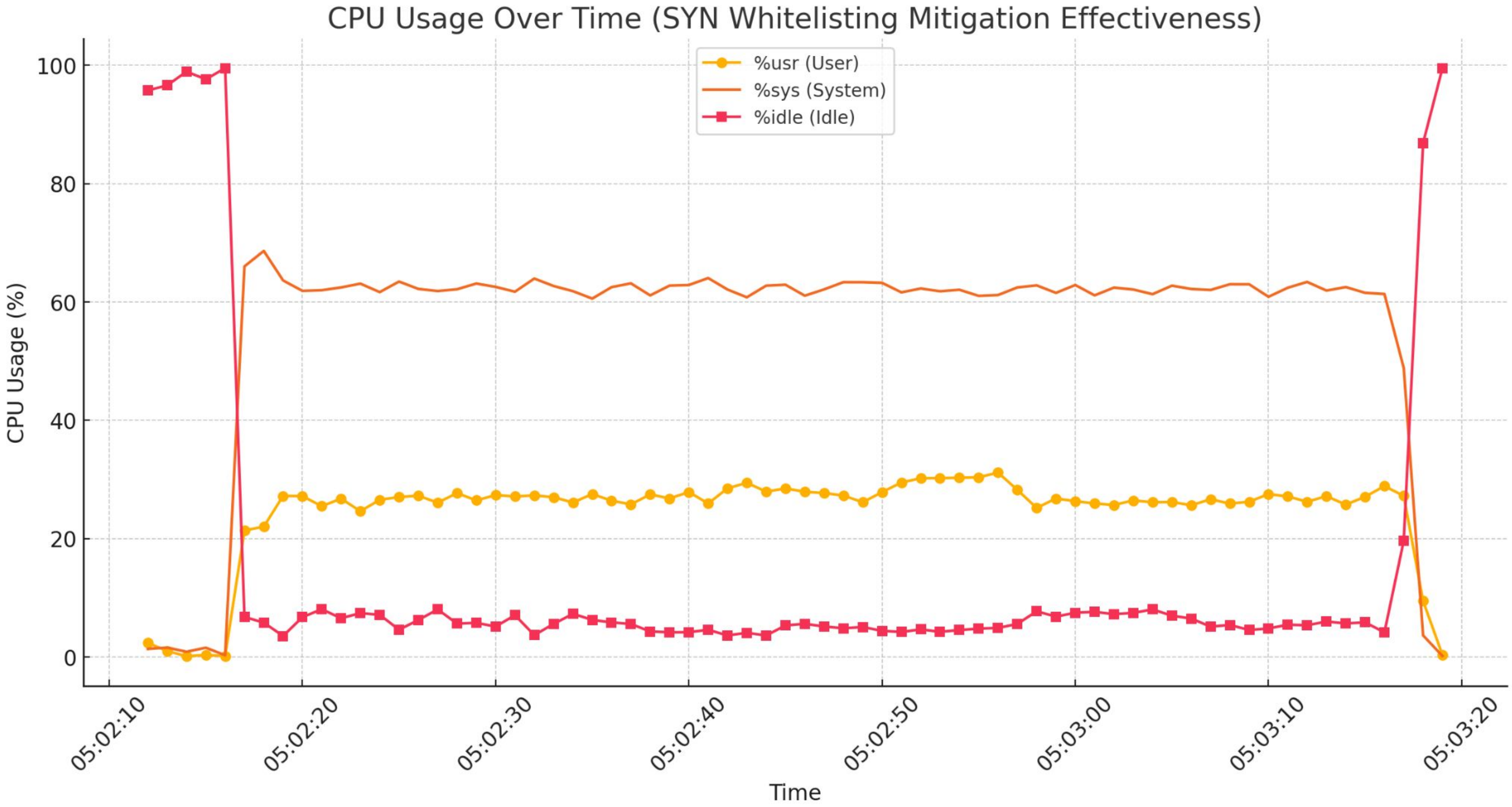
Mitigation Strategies - SYN Whitelist Bypass

Description: Attackers spoof or mimic IPs on a SYN whitelist to get past protections.

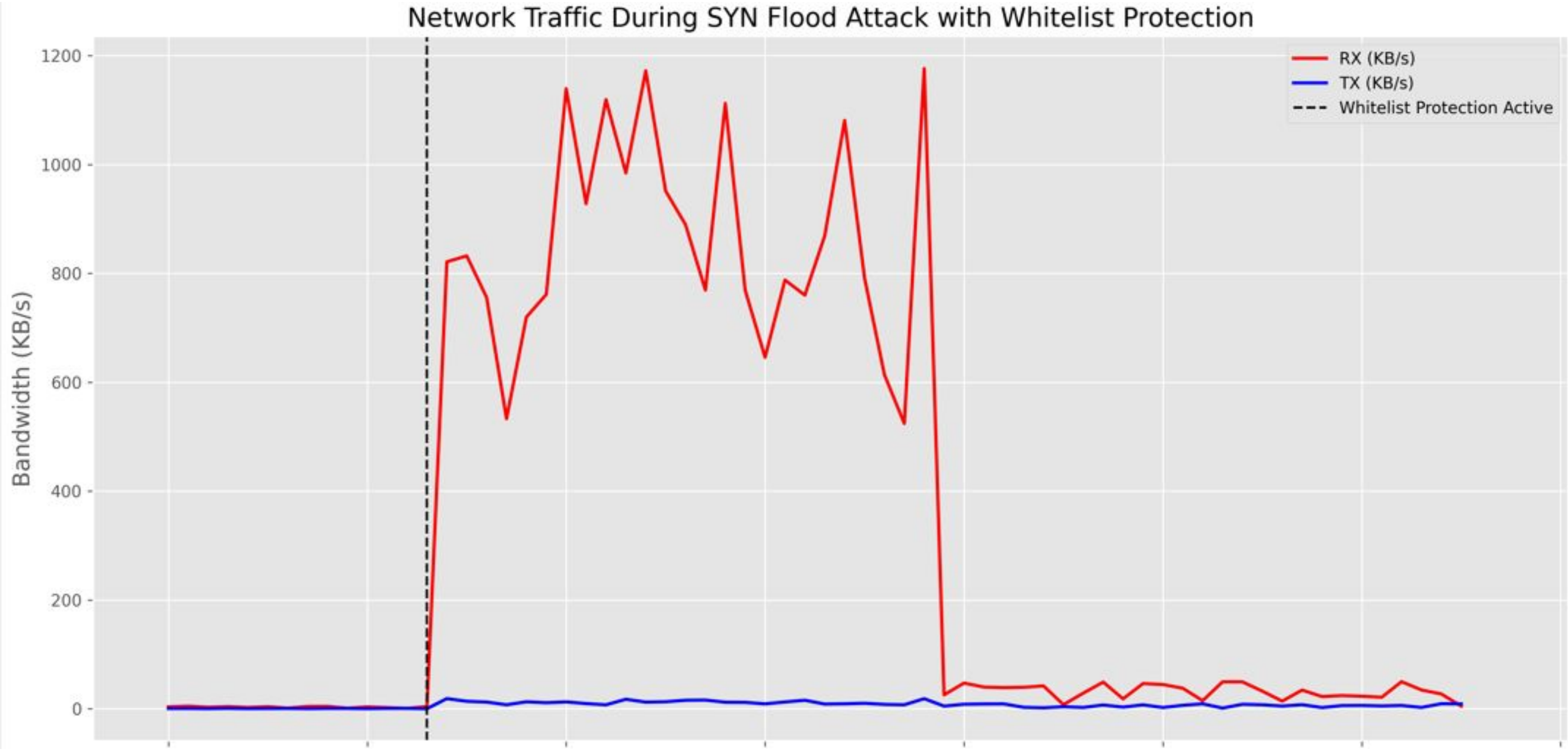
Mitigation Techniques:

- **Challenge-Response for Whitelisted IPs**: Use cryptographic or behavioral challenges even for whitelisted IPs.
- **Strict IP Verification**: Implement Layer 3-4 spoofing prevention (e.g., ingress filtering, IPsec, or reverse path filtering).
- **Logging & Anomaly Detection**: Monitor whitelisted IPs for abnormal behavior or traffic volume.
- **Behavior-Based Access Control**: Rather than static whitelisting, use reputation or behavioral baselining.

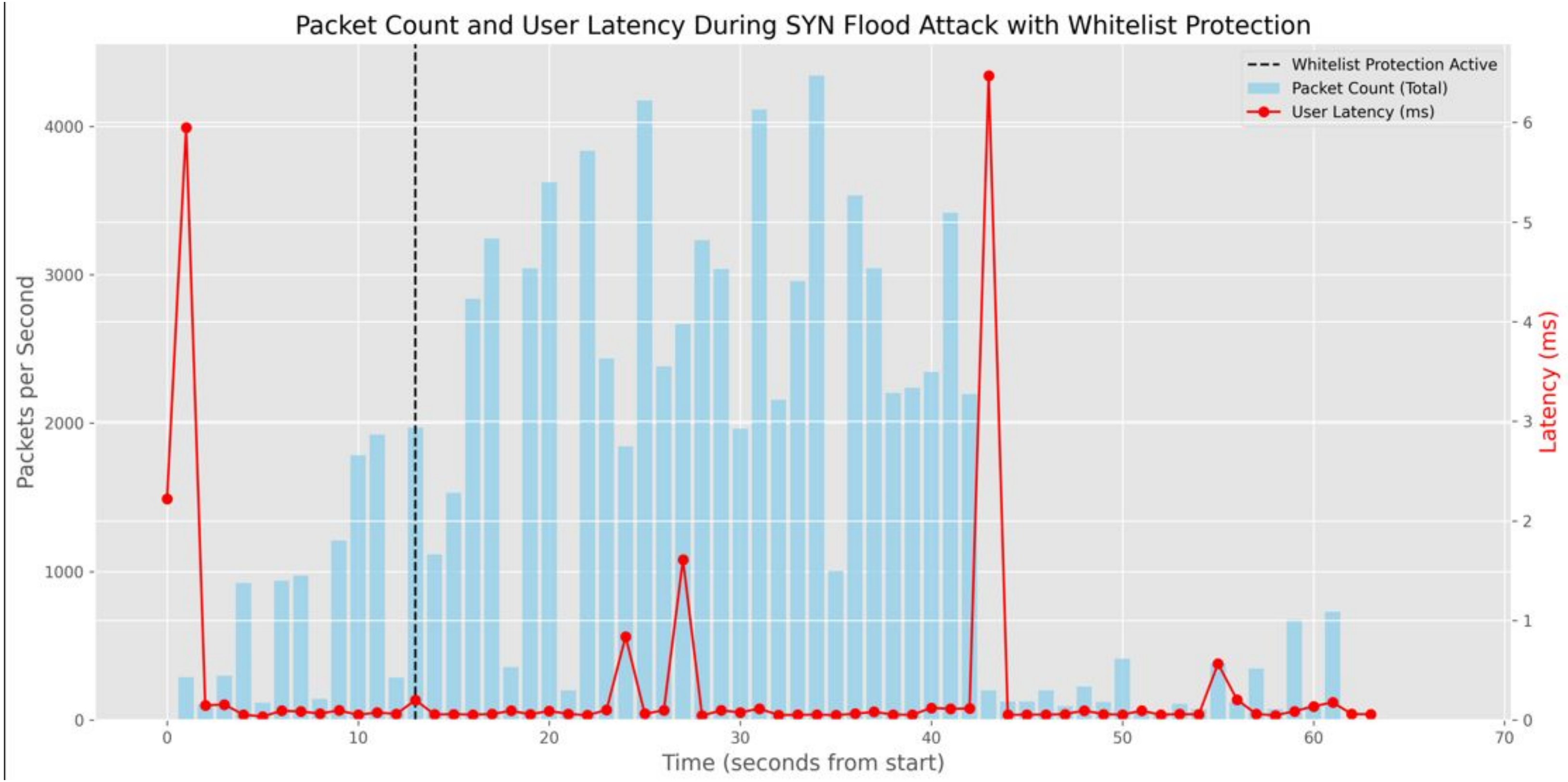
SYN White Listing Data - CPU Usage



SYN White Listing Data - Traffic



SYN White Listing Data - Packet Count and Latency



References

Maayan, G. D. (2025, April 7). *DDOS Trends & Predictions for 2025*. Cyber Security Intelligence. https://www.cybersecurityintelligence.com/blog/ddos-trends-and-predictions-for-2025-8350.html

Arazi, E. (2024, April 24). *The 3 Trends Reshaping the DDoS Threat Landscape in 2023*. radware. https://www.radware.com/blog/ddos-protection/the-3-trends-reshaping-the-ddos-threat-landscape-in-2023/

Ben-Yehuda, S. (2025, February 27). *Cyber threats escalate in 2024 as AI and geopolitics fuel ddos surge | security solutions media*. Security Solutions. https://www.securitysolutionsmedia.com/2025/02/27/cyber-threats-escalate-in-2024-as-ai-and-geopolitics-fuel-ddos-surge/

Kaiyue. (2024, November 10). *What's the future impact of quantum computing on ddos defense - kaiyue April 22, 2025 accelerate, Secure & compute your global business streaming content data file downloads take your edge, to the next level*. Edgenext. https://www.edgenext.com/whats-the-future-impact-of-quantum-computing-on-ddos-defense/

Q & A