Approaches to Solving Proxy Performance Problems for HTTP and SOCKS5 Protocols for the Case of Multi-Port Passwordless Access

Oleksandr Nonik[1], Nadiia Lobanchykova[1], Tetiana Vakaliuk[1,2,3], Viacheslav Osadchyi[2,4], and Oleksandr Farrakhov[5]

[1] *Zhytomyr Polytechnic State University, 103 Chudnivsyka str., Zhytomyr, 10005, Ukraine*
[2] *Institute for Digitalisation of Education of the NAES of Ukraine, 9 M. Berlynskoho str., Kyiv, 04060, Ukraine*
[3] *Kryvyi Rih State Pedagogical University, 54 Gagarin ave., Kryvyi Rih, 50086, Ukraine*
[4] *Borys Grinchenko Kyiv Metropolitan University, 18/2 Bulvarno-Kudriavska str, Kyiv, 04053, Ukraine*
[5] *Center for Information-analytical and Technical Support of Nuclear Power Facilities Monitoring of the National Academy of Sciences of Ukraine, 34a Palladin ave., Kyiv, 03142, Ukraine*

Abstract
The main problem is that most users use proxies for browsers that need more password authentication support. Most services solve this problem using IP address authentication and a "one port, one proxy" approach. This approach helps to solve the problem in principle, but a severe problem arises when scaling services vertically. A study showed that performance decreases in proportion to the number of ports listened to, and sometimes even 2x increases with further increases in the number of ports. The study also found that even three proxies can negatively affect server performance if they listen to thousands of ports. A way to solve this problem was by using efficient I/O (eBPF) in the Linux kernel and adding a universal proxy handler that can automatically detect whether it is a SOCKS, HTTP, or other type of Proxy. The purpose of the proxy reactor is to change incoming packet headers, and the whole scheme of the proxy reactor is based on loading a small, very lightweight piece of code into the Linux kernel to make dynamic changes. According to this concept, the default port can listen to an unlimited number of ports, more precisely, 65,000 ports and other numbers. Physically, only one socket in the system will serve this port, regardless of how operations are performed. The approach is aimed at optimizing and solving the above problems.

Keywords
SOCKS, SOCKS5, HTTP, HTTPS, Go, Linux, IP, TCP, UDP.

1. Introduction

Ensuring the high performance of proxy servers is one of the most critical problems in modern network infrastructure. Proxy servers are widely used for various purposes, including access to blocked resources, anonymous web browsing, and protection against malware. The growing demand for these services has led to an increase in the load on proxy servers [1]. Many existing proxy servers cannot perform well to meet the growing demand. This can lead to delays in accessing resources and other known problems. Many modern applications use multi-port network access [2]. This can lead to an additional load on proxy servers. Several approaches have been developed to solve this problem, such as parallel and asynchronous algorithms, caching, and traffic optimization. However, these approaches have limitations; in some cases, they cannot provide the required performance [3].

The problem of proxy server performance is especially relevant in the case of multi-port passwordless access. Many modern applications

CPITS-2024: Cybersecurity Providing in Information and Telecommunication Systems, February 28, 2024, Kyiv, Ukraine
EMAIL: nonikaleksandr@gmail.com (O. Nonik); lobanchikovanadia@gmail.com (N. Lobanchykova); tetianavakaliuk@gmail.com (T. Vakaliuk); v.osadchyi@kubg.edu.ua (V. Osadchui); farrakhov@ukr.net (O. Farrakhov)
ORCID: 0009-0008-5970-3583 (O. Nonik); 0009-0006-0856-6182 (N. Lobanchykova); 0000-0001-6825-4697 (T. Vakaliuk); 0000-0001-5659-4774 (V. Osadchui); 0000-0003-4988-126X (O. Farrakhov)

use multi-port access to the network. This can lead to an additional load on proxy servers, as each port requires separate request processing.

This paper aims to develop new approaches to solving the performance problems of proxy servers for HTTP and SOCKS5 protocols in the case of multi-port passwordless access.

2. Theoretical Background

2.1. Main Aspects of Proxy Servers for HTTP and SOCKS5 Protocol

Proxy servers are essential in today's online space, acting as a gateway between users and the Internet. These systems act as intermediaries, redirecting user requests to the appropriate servers and returning the received responses. This not only simplifies access to Internet resources but also provides additional security, as a proxy server can filter incoming and outgoing traffic and block access to malicious sites [4].

These servers also play a crucial role in ensuring anonymity and privacy on the Internet. They allow users to hide their real IP addresses, essential for protecting personal information and bypassing geographic restrictions on accessing content. This is especially useful for circumventing geo-restrictions or accessing content that may be blocked in the user's region [5]. Using different IP addresses on users' behalf helps hide their real addresses from web servers. This ensures anonymity and helps avoid potential cyberattacks, as the user's IP address remains unknown. Proxy servers can perform various functions, including redirecting requests from client applications to other servers [6].

Proxy servers simplify and optimize Internet access within local networks, such as office buildings or educational institutions [7, 8]. They allow centralized Internet traffic management, ensuring efficiency and security [9]. By acting as intermediaries between clients seeking resources and servers providing those resources, they can effectively distribute the load on network resources and optimize access to web content [10].

They perform actions on behalf of other servers, which is critical to improving performance and security on the Internet. They can be used to cache data, reduce server load, and improve the overall efficiency of network operations [11]. A sequence diagram illustrating the data transfer between the user and the end resource, both with and without a proxy server, is shown in Fig. 1.

These features make proxy servers indispensable in the modern digital world, where data security and privacy are essential. They help users to remain secure and anonymous while interacting with the Internet [12]. "Proxy" means "intermediary" in English. In the context of the Internet, a Proxy server is a program that acts as an intermediary between a client and a server. It receives requests from the client and forwards them to the server. The server responds to the request, and the Proxy returns the response to the client [13].

In the diagram shown in Fig. 1, you can see how user requests are redirected through the proxy server to the web server, and then the web server response is returned to the user through the proxy server. In the case of direct data transfer, the user's request is sent directly to the web server, and the response from the web server is sent directly to the user.

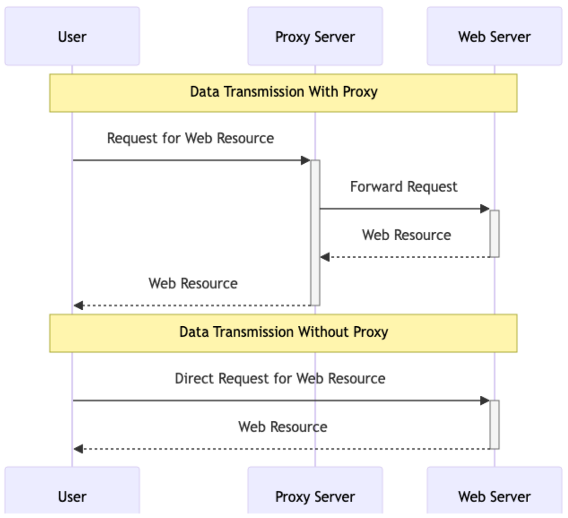

Figure 1: A sequence diagram illustrating the transfer of data between the user and the end resource

A detailed study of the proxy server functionality allowed us to identify several application tasks that they solve—for example, improving the efficiency of IPTV, where proxy servers are used to reduce the delay when watching IPTV, in particular when changing channels and playing Video on Demand (VoD). The use of P2P and proxy servers significantly reduces response times and improves the quality of services [4, 14]. Proxies are also used to develop "living" prototypes in intelligent cities—innovative prototypes in smart cities that allow for detailed data collection and real-time interaction with users, facilitating the development and design process [15, 16].

Web service caching: Proxy servers are used to cache web service data, providing high performance and efficiency. This is especially important for SOAP-based web services, where a properly configured cache can significantly improve performance [17]. Targeted subsidization of medical services: Proxies are used to identify target groups for subsidized healthcare services in countries with limited government capacity, such as Myanmar. This helps to ensure that subsidies are provided to those who need them most.

2.2. Types and Classification of Proxy Servers

Proxy servers can perform different functions so they can be divided into several types based on different criteria. One of the most important criteria is the protocol the proxy server uses. Let us analyse the protocols, in Fig. 2:

- An FTP proxy is used to upload data to an FTP server.
- CGI proxies (anonymizers) help to open any website in a browser by masking the user's IP address.
- SMTP, POP3, and IMAP proxies are used to send and receive email.
- HTTP and HTTPS proxies are used to browse the web. HTTP proxies are used for normal web browsing, and HTTPS proxies are used for encrypted web browsing.
- SOCKS proxies are the most anonymous type of proxy servers because they redirect all data to the end server as a client.

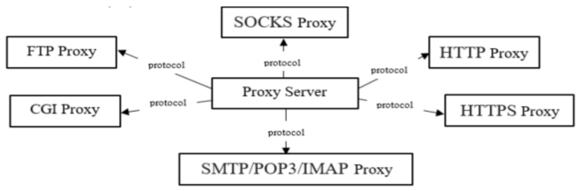

Figure 2: Diagram illustrating different types of proxy servers by protocol

HTTP, HTTPS, and SOCKS proxies are the most commonly used. Each type of proxy server plays a unique role in network communication and has specific functions.

According to the level of anonymity, proxy servers are divided into Fig. 3:

- Transparent proxy servers do not change the requested or received data, so anonymity is not assumed, and websites can read your IP address. They are used for caching content or for securing local networks.
- Anonymous proxies change your IP address to the address of a proxy server to hide your actual IP address. This can be useful for protecting your privacy or accessing websites blocked in certain regions.
- High-anonymity proxies use additional measures to protect your privacy, such as encrypting your data or changing your IP address periodically. They are the most secure option for anonymous browsing.

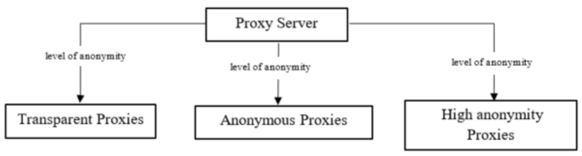

Figure 3: Diagram illustrating different levels of proxy anonymity

A diagram of proxy server classification by IP origin is shown in Fig. 4:

- Proxy data centers are used by companies specializing in proxy services. They have high bandwidth and can handle a large number of requests simultaneously.
- Residential proxy servers are provided by individuals who use their personal computers as proxy servers. They typically have lower bandwidth than proxy data centers, but they can be more secure because you know who you trust with your traffic.

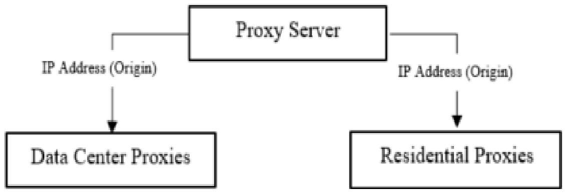

Figure 4: Diagram illustrating different types of proxy servers by IP address origin

A diagram illustrating the different types of proxy servers by the way they are used is shown in Fig. 5. This criterion distinguishes the following types of proxy servers:

- Public proxy servers are available for free and do not require registration. They are a good option for one-time use but can be slow and insecure.
- Companies or individuals provide private proxy servers. They are usually paid for but offer better performance and security than public proxy servers.

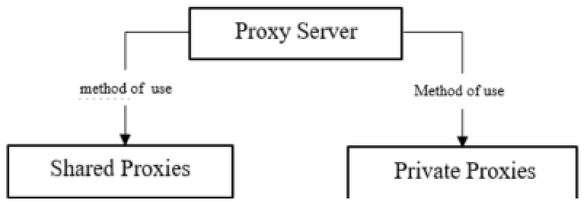

Figure 5: A graphical diagram illustrating the different types of proxy servers by how they are used

The following types of proxy servers are distinguished based on the duration of IP address use (Fig. 6):

- Non-reversible or static IP proxy servers always have the same IP address. This can be useful for applications that require constant access to a particular server.
- Revolving or variable IP proxy servers change their IP address periodically. This can be useful for protecting privacy or for bypassing blocking.

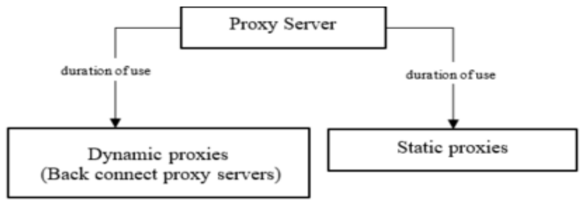

Figure 6: A graphical chart illustrating the different types of proxy servers based on the duration of IP address usage

A diagram illustrating different types of proxies based on data modification is shown in Fig. 7.

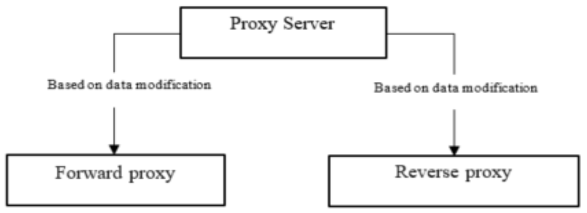

Figure 7: A graphical diagram illustrating different types of proxies based on data modification

The following types of proxy servers are distinguished:

- Direct proxies redirect requests from clients to destination servers. They do not make any changes to the data.

Reverse proxy servers can perform various functions, such as caching content, filtering requests, or redirecting traffic to different servers.

2.3. Analyzing the Advantages and Disadvantages of Using Proxy Servers

The advantages of proxy servers include:

1. Protecting the network and users: proxy servers protect the network and its users by offering a secure and fast Internet connection. They are instrumental in business and academic environments.
2. Improving the efficiency of VoD systems: Proxies are used to improve the efficiency of Video-On-Demand (VoD) systems by reducing latency and improving the quality of service.
3. Efficient proxy selection in cooperative caching: Proxies can select the server that efficiently offers the best response time to the client, reducing overall network traffic consumption and improving average response time.
4. Improving caching architecture: Integrating browser and proxy caches can reduce Internet traffic by integrating browser caches with proxy servers to improve cache management.
5. Ensuring access to digital resources: proxy servers are used by libraries to provide access to proprietary databases for off-campus users, as well as to restrict access to specific databases or classes of users within the library.

The disadvantages of proxy servers include:

1. Bandwidth and speed limitations: proxy servers can have bandwidth and speed limitations, especially in the case of a large number of simultaneous requests.
2. Potential security issues: Proxy servers can pose potential risks, especially if they are not correctly configured or outdated.
3. Difficulty in scaling: Proxy servers can be challenging, especially with many users or high data volumes. Proxies can lead to performance degradation by adding an extra step in the request processing process.

2.4. Overview of the Primary Proxy Server Protocols

Let us analyze the primary proxy server protocols. An HTTP proxy is the most common type of Proxy. Its primary purpose is to organize the work of browsers and other HTTP programs. How it works: a program or browser sends a request to a proxy server to open a specific URL resource, and the proxy server receives data from the requested resource and transmits it to the browser. With the help of an HTTP proxy, you can control such requests:

- Caching of data (images, pages, etc.). Provides a speed boost with static content and low bandwidth of the external communication channel. Restricting access to specific resources: creating a blacklist of prohibited sites or a white list of allowed sites.
- Substitution of a resource for a request other than the one requested by the user: for example, instead of banners with advertising, transparent images are displayed, significantly saving loading time and traffic.
- Uniform distribution of bandwidth between users of the local network: for example, you can limit the speed of file uploads to avoid server congestion.
- Logging: you can count traffic for each user and see a list of popular websites.
- Request routing: send some requests directly and redirect others through other proxies.

However, the lack of encryption support makes it impossible to work with HTTPS websites. Also, an HTTP proxy can transmit your IP address in the request headers, which excludes this type of Proxy from the category of anonymous proxies. An HTTPS proxy is an HTTP proxy that uses encryption (S-Secure). All traffic passing through the proxy server is encrypted with a tamper-resistant algorithm. With this approach, there is no way to find out what kind of information (picture, video, document) is being transmitted through the proxy server since the proxy server is not involved in encrypting and decrypting information. This allows you to use an HTTPS proxy to transfer almost any TCP protocol: POP3, SMTP, IMAP, NNTP, etc. HTTPS can still transmit your IP address in the request headers, which excludes this type of Proxy from the category of anonymous proxies.

Socks Proxy is the most advanced information transfer protocol. The SOCKS protocol was developed to allow applications that do not support using proxies to use network resources. The protocol is a translator, but the Socks client is located between the application and transport layers in the network compared to other proxies. The Socks server is located at the application layer, eliminating the need for high-level protocols. A Socks server does not transmit IP in the request header, being anonymous. Socks proxies support almost all protocols, including HTTP, HTTPS, FTP, SSH, Telnet, etc. This makes them a versatile tool for networking. It disguises actual IP addresses, hides physical locations, and prevents outsiders from tracking your online activities. Although a SOCKS5 proxy is not as reliable and versatile in terms of protection as a VPN, it offers several serious advantages, including anonymity: The IP address is hidden, and Internet traffic is routed through a proxy server, making the client less vulnerable to tracking; bypassing blockages: SOCKS5 allows you to bypass geographic restrictions and content blocking; speed: SOCKS5 offers high data transfer speeds.

The choice of proxy server depends on needs and requirements. If you need a proxy to work with a browser, then an HTTP proxy or HTTPS Proxy will be a good choice. If you need a proxy to work with applications that do not support proxies, then a Socks proxy is the best option. If you need anonymity, then a Socks proxy is the only option.

3. Discussion

3.1. Overview of Popular Proxy Server Solutions

Many software solutions can be used to raise a proxy server. Here are some of the most popular ones: Squid is a free and open-source proxy server that supports a wide range of protocols and functions; Apache is a web server that can also be used as a proxy; Nginx is another web server that can be used as a proxy; HAProxy is a highly available proxy server that can be used to balance the load between different servers; Varnish is a fast and efficient proxy server that can be used for caching content; Shadowsocks is a popular proxy server protocol that supports various encryption and anonymity methods; V2Ray is a new proxy server protocol that is faster and more efficient than Shadowsocks; Trojan is another proxy server protocol that is popular in China. The following solutions are also available: GoAgent, Clash, X-Tunnel, Psiphon, and Windscribe.

In addition, some other software solutions can be used to raise a proxy server:

- ProxyChains is a command-line tool that allows you to use proxy servers for various applications.
- Proxifier is a Windows tool that allows you to use proxy servers for various applications.
- FoxyProxy is a Firefox browser extension that allows you to use proxy servers for various websites.
- SwitchyOmega is a Chrome browser extension that allows you to use proxy servers for various websites.

The choice of a specific software solution depends on your needs and requirements. If you need a simple and reliable proxy server, Squid or Apache are good choices. If you need a highly available proxy server, HAProxy is a good choice. Varnish is a good choice if you need a fast and efficient proxy server for caching content.

3.2. Overview and Analysis of 3proxy

3proxy is a popular solution for creating and configuring proxy servers. It is known for its ease of configuration, support for various protocols and features, and high performance and stability. Here are some of its features:

- Supports numerous protocols, including HTTP, HTTPS, FTP, SOCKS, SMTP, POP3, IMAP, TELNET, and others.
- Provides various types of anonymity, from transparent to highly effective.
- Allows you to filter traffic based on IP addresses, users, domain names, protocols, and headers.
- Caches content to improve performance. Supports load balancing to distribute traffic across multiple servers.
- Has a built-in system to protect against attacks.

The advantages of 3proxy include easy setup and use, open source code available for free use and modification, support for a wide range of protocols and features, high performance and stability, and online resources for training and support.

Among the disadvantages is that it can be difficult for advanced users to configure; some features require additional modules or configuration and do not guarantee complete anonymity, especially with accessible settings.

3proxy provides many additional features that can be configured in the configuration file, including traffic filtering, data caching, load balancing, and access control.

3.3. Advanced Overview and Configuration of the Squid Proxy Server

Here is an advanced overview and configuration of the Squid proxy server. Squid is an accessible, open-source proxy server that improves network performance, security, and control. It was developed at the National Supercomputing Centre in the United States and was first released in 1996. Squid supports many protocols, including HTTP, HTTPS, FTP, SMTP, IMAP, and POP3. It can also be used for load balancing between multiple servers. One of the main features of Squid is caching data in the proxy server's memory. This allows you to reduce delays and increase the speed of loading websites and other resources. Squid can also be used for traffic filtering (blocking or allowing certain types of traffic). This can be useful for protecting your network from malware and other threats. Squid can be used for network access control (allowing or denying access to certain websites or resources). This can be useful for companies that want to protect employees from harmful content. Squid is a popular choice for web servers in corporate networks and other organizations that need to improve performance, security, and network control. The main benefits of using Squid include improved security, access control, load balancing, data caching, and traffic filtering.

4. Results

The proxy reactor operates based on the joint work of low-level mechanisms (iptables, ebpf) and high-level network traffic processing code (golang, gnet), Fig. 8.

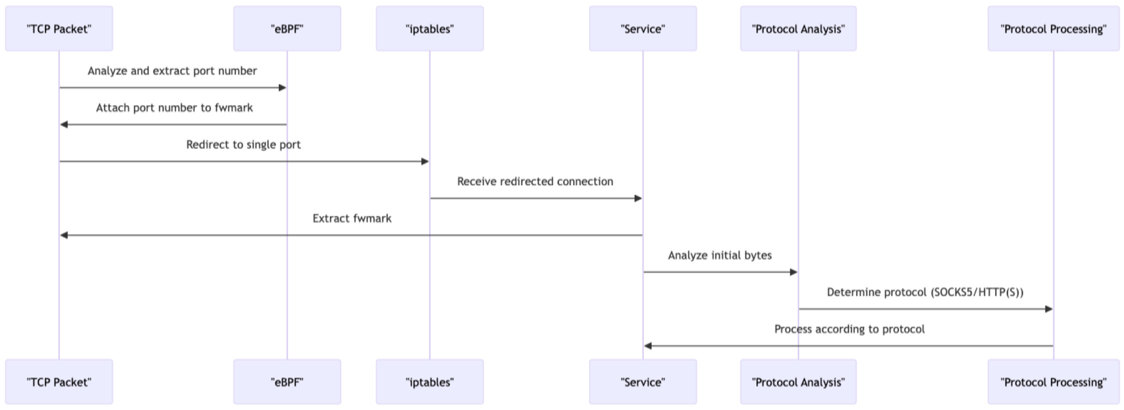
"TCP Packet"
"eBPF"
"iptables"
"Service"
"Protocol Analysis"
"Protocol Processing"
Analyze and extract port number
Attach port number to fwmark
Redirect to single port
Receive redirected connection
Extract fwmark
Analyze initial bytes
Determine protocol (SOCKS5/HTTP(S))
Process according to protocol

Figure 8: Sequence diagram illustrating the process of processing an incoming connection

The process of processing an incoming connection can be briefly described as follows. The incoming TCP packet is analyzed, and the port number is written to the packet's mark (fwmark) using eBPF. Iptables, according to the rule, redirects any port in the specified range to a single port. The service accepts the connection and extracts the marker attached in the first step. This way, it receives information about the original port to which the user sent the connection request. The initial bytes in the incoming packet are analyzed, and based on them, the protocol used by the user (SOCKS5 or HTTP(S)) is determined. Processing of the protocol that was determined in the previous step is started.

The algorithm for processing an incoming connection can be represented as follows:

Step 1: The incoming TCP packet is analyzed using eBPF. eBPF is a technology that allows developers to write code that executes directly in the Linux kernel. In this case, the eBPF code extracts the incoming port number from the packet. The port number is then written to the packet's fwmark.

Step 2: iptables is a Linux firewall that can filter network traffic. In this case, iptables is configured to redirect any port within a specified range to a single port. This allows the service to accept connections on any port and forward them to the appropriate destination.

Step 3: The service accepts the connection and extracts the token attached in step one. The token contains the original port the user sent the connection request.

Step 4: The initial bytes of the incoming packet are analyzed. The first few bytes of a SOCKS5 or HTTP(S) packet contain a unique identifier that can be used to determine the protocol the user is using.

Step 5: If the protocol is SOCKS5 or HTTP(S), the appropriate processing is started. For example, if the protocol is SOCKS5, the service will create a SOCKS5 proxy connection to the destination server.

A SOCKS5 proxy server performs the following actions: transmits data from the user to the destination server, receives data from the destination server, and transmits it to the user. A SOCKS5 proxy server can perform additional functions like traffic filtering, security settings, and logging.

In the case of HTTP(S), the proxy server performs the following actions: receives an HTTP request from the user, sends the HTTP request to the destination server, receives an HTTP response from the destination server, and transmits the HTTP response to the user.

An HTTP(S) proxy server can perform additional functions, such as data caching, accelerating the loading of web pages, security settings, and logging.

4.1. Development of High-Level Network Traffic Processing Code for Protocol Processing

One of the most essential parts of traffic processing is packet queuing. This is due to the asynchronous nature of the architecture, which is more critical due to the frequent imbalance between the channel speeds between the user and the proxy server and the proxy server and the server to which the traffic is proxied. One of the most common cases is when the client and proxy server have a higher ping and slow speed, and the proxy server has a wide channel and minimal ping, Fig. 9:

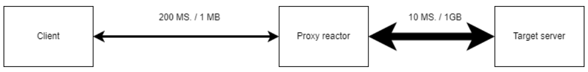

Figure 9: Demonstration of imbalance between client and proxy, proxy and server bandwidth

This feature makes it necessary to implement a queue of packets that are compiled in memory and sent whenever possible.

A more complex approach is used that does not require blocking the flow:

- Asynchronously receives a portion of traffic from the sender.
- Write the received traffic to the end of the queue.
- Get the first packet in the queue from the queue.
- Send the asynchronous record to the recipient.

This approach is more complicated, but it allows you to guarantee the sequence of sending the received traffic under conditions of different pings, bandwidths, and other network factors.

Instead of using byte array allocation to process and temporarily store packets, the entire project uses byte slice, a specialized sync version. Pool for byte arrays. This reduces the load on the garbage collector and generally optimizes memory usage by minimizing memory fragmentation.

At the heart of the processing of these flows is a state machine that handles the sequence greeting—authorization—connection—tunneling. In code, this describes the following states:

negotiateHandle—by the SOCKS5 protocol specification, the first packet sends the authorization method supported by the client. This method can go into the following states: *authStatus*—the client supports login and password authentication. You need to process the next packet in which the login and password will be sent: *connectStatus*—the client has sent that it does not support any authorization methods (passwordless authorization). If the client's IP address is on the list, it is possible to skip the authorization stage and go straight to the connect status status. In case of a protocol error or authorization failure, a SOCKS5 error is sent, and the connection is closed.

authStatus—the sent packet with the client's login and password is steamed and checked for correctness. It can switch to the connectStatus state or send a client authentication error and close the connection.

connectStatus—according to the SOCKS5 protocol specification, the client sends a command to connect to the target servers. At this point, all the blinklists (port, IP) are checked, and the connection to the server sent by the client is launched. It can switch to the tunnelStatus state or send a protocol error to the client and close the connection. The final status is when the handler pumps traffic between the client and the server using the packet queue.

This handler hides two protocols that are quite different in their logic. However, to optimize performance, the processing was combined into one handler because both protocols use the HTTP format for initial packets (and subsequent packets in the case of HTTP).

An example of a packet for proxying an HTTP request:

```
GET http://example.com/ HTTP/1.1
Host: example.com
Proxy-Authorization: Basic <base64 encoded user:password>
User-Agent: curl/8.4.0
Accept: */*
Proxy-Connection: Keep-Alive
```

An example of the first packet for proxying an HTTPS request:

```
CONNECT example.com:443 HTTP/1.1
Host: example.com:443
Proxy-Authorization: Basic <base64 encoded user:password>
User-Agent: curl/8.4.0
Proxy-Connection: Keep-Alive
```

Processing of both protocols begins with waiting for all request headers to arrive from the client:

```
func headersReady(c gnet.Conn) (bool, error) {
    buf, err := c.Peek(-1)
    if err != nil {
      return false, err
    }

    contains := bytes.Contains(buf, []byte("\r\n\r\n"))
    return contains nil
}
```

```
func (h *httpProxy) negotiateHandle(state *State, c gnet.Conn) ([]byte, error) {
  if ready, err := headersReady(c); !ready || err != nil {
    return nil, err
  }
```

Next, the headers are manually processed, and a "cleaned" packet is formed, which will be sent to the target server in the case of the HTTP protocol. We use the first header to get information about the server to which we want to connect. This approach is universal for HTTP and HTTPS.

If this is a Proxy-Authorisation header, we note that the user has sent a login and password for further authorization (passwordless authorization based on the user's IP is possible). "Dangerous" headers are removed if they are present (Proxy-Authenticate, Proxy-Authorisation, Proxy-Connection) to ensure user privacy in the case of the HTTP protocol, as sending these headers will reveal to the target server the fact that the user is using a proxy.

Further processing is divided into HTTPS and HTTP protocols. This protocol is very similar to the SOCKS5 approach. After authorization, the proxy server establishes a connection with the target server and then pumps traffic in both directions.

```
var (
  connectMethod    = []byte("CONNECT")
)


parser := wildcat.NewSizedHTTPParser(headerCount)
state.tunnel = bytes.EqualFold(parser.Method, connectMethod)


func (h *httpProxy) handleTraffic(state *State, c gnet.Conn) error {
    ...
If state. tunnel {
  _, err := state.queue.CopyAsync(c, state.downstream.Current)
  return err
}
    ...
}
```

The processing logic of this protocol is more complex than HTTPS because it is impossible to "transfer" traffic between the parties. However, it requires complete packet processing, header transformation, and analysis to determine the actual size of the request.

In the previous step, during the header transformation processing, a packet that is already safe to send to the target server was generated. However, for correct operation, it is necessary to additionally find out the size of the request body by extracting the number of bytes specified in the Content-Length header.

Based on this information, the following logic is run:

1. Connect to the target server.
2. Send the cleaned headers to the target server.
3. Write to the connection state in the byte from the Content-Length header that you want to transfer "as is".

The transfer of the specified number of bytes is in the httpProxy.handleTraffic method:

```
func (h *httpProxy) handleTraffic(state *State, c gnet.Conn) error {
  ...
  if state.tunnelSized > 0 {
    t, err := state.queue.CopyAsync(c, state.downstream.Current)
    if err != nil {
      state.tunnelSized = 0
      return err
    }

    atomic.AddInt64(&state.tunnelSized, -t)
    return nil
  }
  ...
}
```

After the specified number of bytes has been transferred, the system checks to see if more data is needed to process. If the incoming data stream is not exhausted, the processor continues to receive and analyze the following packets, ensuring continuous data transfer between the client and the server. This allows you to process multiple requests or long sessions without interruption.

The measurements will be made on a VPS hosted by DigitalOcean. Requirements: the system should be able to listen to 5000 HTTP(S)+SOCKS5 ports and support passwordless IP authentication.

Testing process: 100 threads are accessed from a server in the same data center to random ports in the range of 5000 thousand, and a 16 KB file is downloaded through a proxy.

3proxy—the software does not support one port = 2 protocols; to meet this limitation, you need to listen to 5000×2 ports.

The system idle state is shown in Fig. 10. Demonstrating the load of 3proxy on the VPS during downtime allows you to analyze and make management decisions.

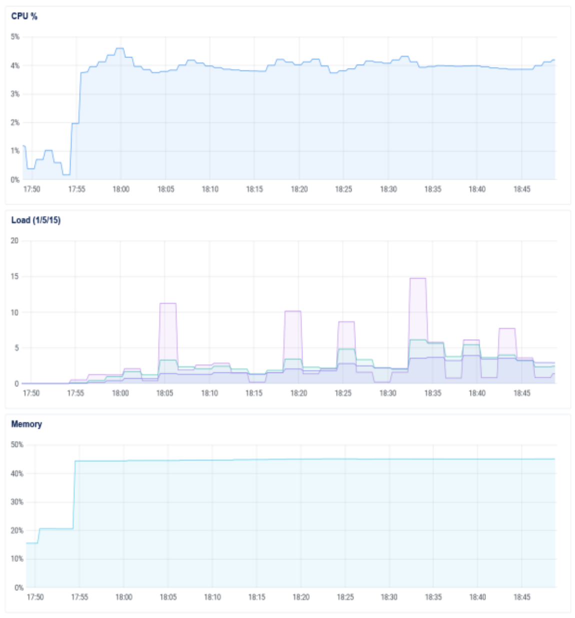

Figure 10: Demonstration of 3proxy load on VPS during idle time

A demonstration of 3proxy load on a VPS under load is shown in Fig. 11.

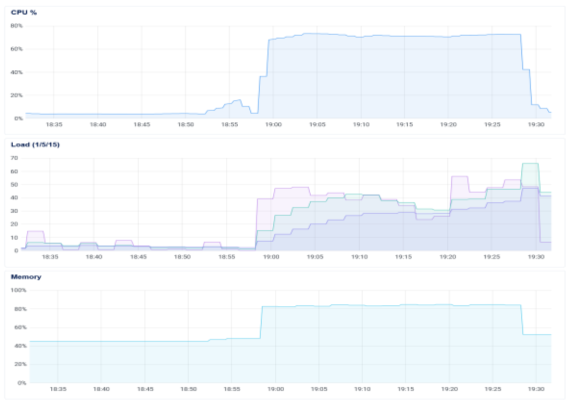

Figure 11: Demonstration of 3proxy load on VPS under load

A demonstration of the Proxy Reactor load on the VPS during downtime is shown in Fig. 12.

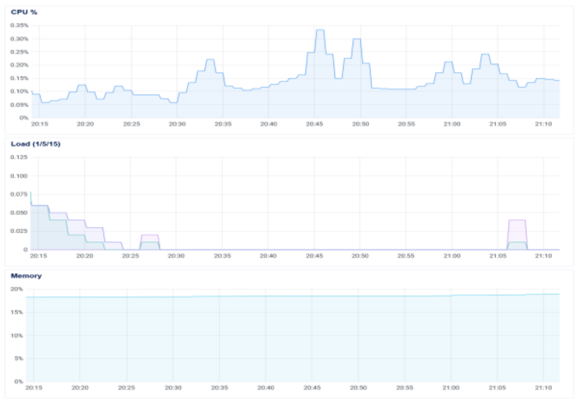

Figure 12: Demonstration of the Proxy Reactor load on VPS during idle time

A demonstration of the Proxy Reactor load on the VPS during load is shown in Fig. 13.

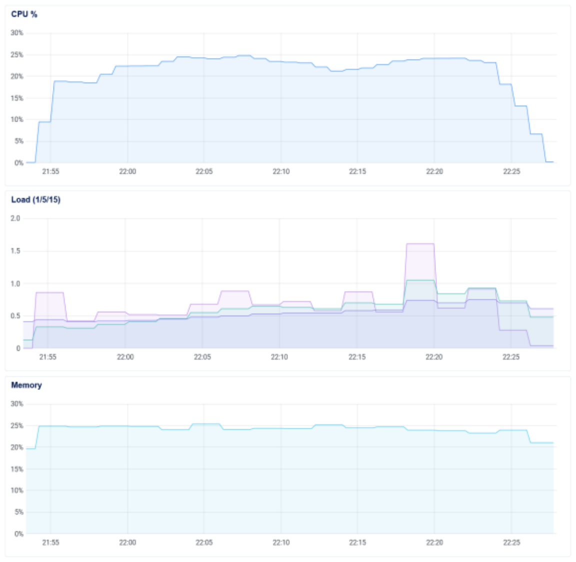

Figure 13: Demonstration of the Proxy Reactor load on VPS under load

Under load, the proxy reactor demonstrated significantly better performance and resource management efficiency than 3proxy, ensuring stability and high request processing speed even with many simultaneous connections.

5. Conclusions

This study is devoted to the issue of proxy server performance optimization, reflecting current challenges in the field of network technologies. The topic's relevance is justified by the constant growth of the load on proxy servers caused by the increasing demand for high-performance, reliable, and secure network services.

The paper focuses on analyzing existing performance problems, particularly pronounced in the context of multi-port access and passwordless authentication methods. The theoretical foundations of new optimization methods are developed, including parallel and asynchronous algorithms, dynamic load balancing between ports, and innovative solutions using eBPF and backtracking algorithms to determine the original port.

Experimental verification and testing of the proposed approaches have shown their effectiveness in practical conditions, significantly improving proxy servers' performance. The results confirmed the work's scientific novelty and opened the way for further development and improvement of network technologies.

Thus, this study contributes to the networking technology field by offering new solutions to proxy server performance problems that are of great practical importance for modern network infrastructure.

References

[1] Y. Sadykov, et al., Technology of Location Hiding by Spoofing the Mobile Operator IP Address, in: IEEE International Conference on Information and Telecommunication Technologies and Radio Electronics (2021) 22–25. doi: 10.1109/UkrMiCo52950.2021.9716700.

[2] A. Carlsson, et al., Sustainability Research of the Secure Wireless Communication System with Channel Reservation, in: 2020 IEEE 15th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (2020). doi:10.1109/tcset49122.2020.235583.

[3] V. Buriachok, V. Sokolov, P. Skladannyi, Security Rating Metrics for Distributed Wireless Systems, in: Workshop of the 8th International Conference on "Mathematics. Information Technologies. Education:" Modern Machine Learning Technologies and Data Science, vol. 2386 (2019) 222–233.

[4] S. Bhandari, G. Lee, N. Crespi, Peer to Peer Proxy Based IPTV Services, IEEE Globecom Workshops (2009) 1–6. doi: 10.1109/GLOCOMW.2009.5360711.

[5] M. Boonen, B. Lievens, The Use of Live-Prototypes as Proxy Technology in Smart City Living Lab Pilots, Distributed, Ambient and Pervasive Interactions: Understanding Humans, LNISA 10921 (2018) 203–213. doi: /10.1007/978-3-319-91125-0_17.

[6] P. Küngas, M. Dumas, Configurable SOAP Proxy Cache for Data Provisioning Web Services, SAC'11: ACM Symposium on Applied Computing (2011) 1614–1621. doi: 10.1145/1982185.1982523.

[7] R. Marusenko, V. Sokolov, P. Skladannyi, Social Engineering Penetration Testing in Higher Education Institutions, Advances in Computer Science for Engineering and Education VI, vol. 181 (2023) 1132–1147.

[8] R. Marusenko, V. Sokolov, V. Buriachok, Experimental Evaluation of Phishing Attack on High School Students, Advances in Computer Science for Engineering and Education III, vol. 1247 (2020) 668–680. doi:10.1007/978-3-030-55506-1_59.

[9] S. Htet, T. Ludwick, A. Mahal, Targeting Subsidised Inpatient Services to the Poor in a Setting with Limited State Capacity: Proxy Means Testing in Myanmar's Hospital Equity Fund Scheme, Trop. Med. Int. Health 24(9) (2019) 1042–1053. doi: 10.1111/tmi.13286.

[10] I. Khalil, G. PeiQi, Efficient Proxy Selection in Cooperative Web Caching, 15th IEEE International Conference on Networks (2007) 376–381. doi: 10.1109/ICON.2007.4444116.

[11] A. Imtiaz, M. Hossain, Distributed Cache Management Architecture: To Reduce the Internet Traffic by Integrating Browser and Proxy Caches, International Conference on Electrical Engineering and Information & Communication Technology (2014) 1–4. doi: 10.1109/ICEEICT.2014.6919088.

[12] J. Duke, X. Yu Authenticating Users Inside and Outside the Library, Internet Ref. Serv. Q. 4(3) (1999) 25–41. doi: 10.1300/J136v04n03_05.

[13] M. Sysel, O. Doležal, An Educational HTTP Proxy Server, Proced. Eng. 69 (2014) 128–132. doi: 10.1016/j.proeng.2014.02.212.

[14] S. Chen, et al., Pretty-Bad-Proxy: An Overlooked Adversary in Browsers' HTTPS Deployments, 30th IEEE

Symposium on Security and Privacy, (2009) 347–359. doi: 10.1109/SP.2009.12.

[15] O. Talaver, T. Vakaliuk, Reliable Distributed Systems: Review of Modern Approaches. J. Edge Comput. 2(1) (2023) 84–101. doi: 10.55056/jec.586.

[16] N. Lobanchykova, I. Pilkevych, O. Korchenko, Analysis and Protection of IoT Systems: Edge Computing and Decentralised Decision-Making, J. Edge Comput. 1(1) (2022) 55–67. doi: 10.55056/jec.573.

[17] A. Jony, A. Arnob, A Long Short-Term Memory Based Approach for Detecting Cyber Attacks in IoT Using CIC-IoT2023 Dataset. J. Edge Comput. (2024). doi: 10.55056/jec.648.