International Scientific Conference 20 UNITECH 26

INTERNATIONAL SCIENTIFIC CONFERENCE
19-21 November 2026
GABROVO

REMOTE-ACCESS IMPLEMENTATION FOR A NETWORK ANOMALY DETECTION SYSTEM

Viktoria Ivanova

Technical University of Gabrovo, Gabrovo, Bulgaria
Corresponding author: v.ivanova@tugab.bg

Abstract
Remote access and development of computer systems and servers, alongside their virtualized systems such as network anomaly detection systems (NADS), across several complex and independent network sites often brings up privacy concerns and risks of third-party data control involvement. This paper proposes and empirically evaluates a self-hosted and security-oriented implementation that brings together RustDesk for remote access of computer systems, servers, and a virtualized NADS across two geographically separated and operationally distinct networks and Tailscale for creating a software-defined mesh virtual private network (VPN), connecting devices from both networks. The main results demonstrate that direct intra-LAN connections with minimal latency of 1-3 ms were established, while inter-LAN links used a relay server resulting in higher latency range of 115-120 ms. The RustDesk service checks reported 100% reachability while the RustDesk server's usage was approximately 0.31% CPU and 15.40 MiB RAM during one active remote session. The study's value resides in the achieved objective which is defined as implementing and evaluating an open-source, operationally viable, scalable, and security-oriented remote-access solution under the tested cross-site network environment conditions and constraints, without intentionally opening device ports and risking public exposure of internal services.

Keywords: Network anomaly detection, Remote access, RustDesk, Tailscale VPN, open-source.

INTRODUCTION

Experience from developing and operating a network anomaly detection system (NADS) indicated that remote access is a viable approach for modern remote development, maintenance, operation, and monitoring of network traffic analysis and anomaly detection solutions. In the computer systems and technologies field, there is a plethora of commercial remote desktop solutions, such as AnyDesk, Splashtop, and TeamViewer [1].

Such solutions are proven to be highly accessible and quick to set up for personal use. However, from a research perspective, their subscription-based services create several limitations, such as per-user licenses, long-use detection flags, when using the free-tier licenses, limited number of connected devices, active session timeouts, private cloud server relays, and artificial connection constraints. In result leading to incompatibility, privacy concerns, and inefficiency when compared to open-source alternatives like RustDesk for academic research purposes (e.g., design and reproducibility of privacy-oriented solutions, real-time monitoring and training of modules, scalability and long-term sustainability of lab environments) [4].

AnyDesk claims over 1 billion downloads and 200,000 companies while TeamViewer reports over 2.5 billion device installations and more than 600,000 customers, which confirms their global influence on the market. However, even against these serious competitors, RustDesk illustrates that open-source remote access is no longer niche as the project reached more than 113,000 GitHub stars, 30 million client downloads, and 10 million Docker deployments. These numbers indicate that

This is an open access article licensed under Creative Commons Attribution 4.0 International
Print ISSN: 1313-230X, Online ISSN: 3033-1404

an open-source solution with self-hosting capabilities has achieved wide real-world adoption and credibility within the field. Furthermore, it is free of cost, and its features include P2P (Peer-to-Peer) connection with end-to-end encryption based on NaCl (Networking and Cryptography library), self-hosting a free RustDesk open-source server, a wide range of operation systems compatibility, and full data control. Ultimately, these factors support a shift of focus from commercial to open-source solutions [1] [3].

The core motivation for this work arises from the evaluation of several remote-access solutions, including AnyDesk, TeamViewer, and HelpWire, during the development stage of the open-source NADS which is described in prior work [6][7].

The integration of these existing tools ultimately failed, because of regular interruptions during crucial long-lasting real-time sessions in the controlled environment, such as machine learning model training, continuous experimentation and network anomaly detection, as well as system monitoring and optimizations.

The main objective of this paper is to design and evaluate a self-hosted implementation for remote access to address these challenges by combining the open-source software RustDesk with the secure networking solution Tailscale. This solution aims to achieve reproducibility, data privacy, and unified network visibility and management of the configured devices, while avoiding device port exposure to the Internet. The implementation is the operational remote-access layer, specifically designed for the open-source NADS [6].

Tailscale is a software-defined mesh virtual private network (VPN) that connects devices into a private mesh network called tailnet. It has been chosen to further ensure the security of the implementation by using its zero-trust architecture and unified P2P connection across firewalls and network address translation (NAT) boundaries with end-to-end encryption over the open-source WireGuard protocol [4][8][9].

The main contributions of this paper are as follows:

- Design and implementation of a security-oriented and reproducible remote-access architecture created to bypass NAT (Network Address Translation) constraints and public port exposure of internal services by using RustDesk and Tailscale;
- A detailed deployment exposition of the network topology, self-hosted virtualized RustDesk server relay, client connection configuration process, and the Tailscale's tailnet setup;
- Empirical evaluation measuring throughput and resource overhead.

The remainder of this paper is organized as follows: the exposition discusses the proposed implementation, including the network topology, RustDesk self-hosted setup, Tailscale tailnet integration, experimental evaluation and results. Finally, the conclusion summarizes the paper and outlines plans for future work.

EXPOSITION

A. Network Environment

The proposed remote-access implementation links two independent local area networks (LANs) functioning in separate geographical locations. The network topology of the suggested remote-access solution is shown in Fig. 1.

The first LAN is designed and created to be the access site and contains the following devices: two Windows 11-based computer systems named PC-1 and PC-2 that have crucial roles for the successful realization of the remote access. The main workstation from which the remote access is initiated to the laboratory site is PC-1. While the second system PC-2 by using the virtualization software VMWare Workstation Pro 17 hosts a virtual machine (VM) operating on Ubuntu Server 26.04 LTS. The main role of the created VM for this implementation is running and maintaining the RustDesk server.

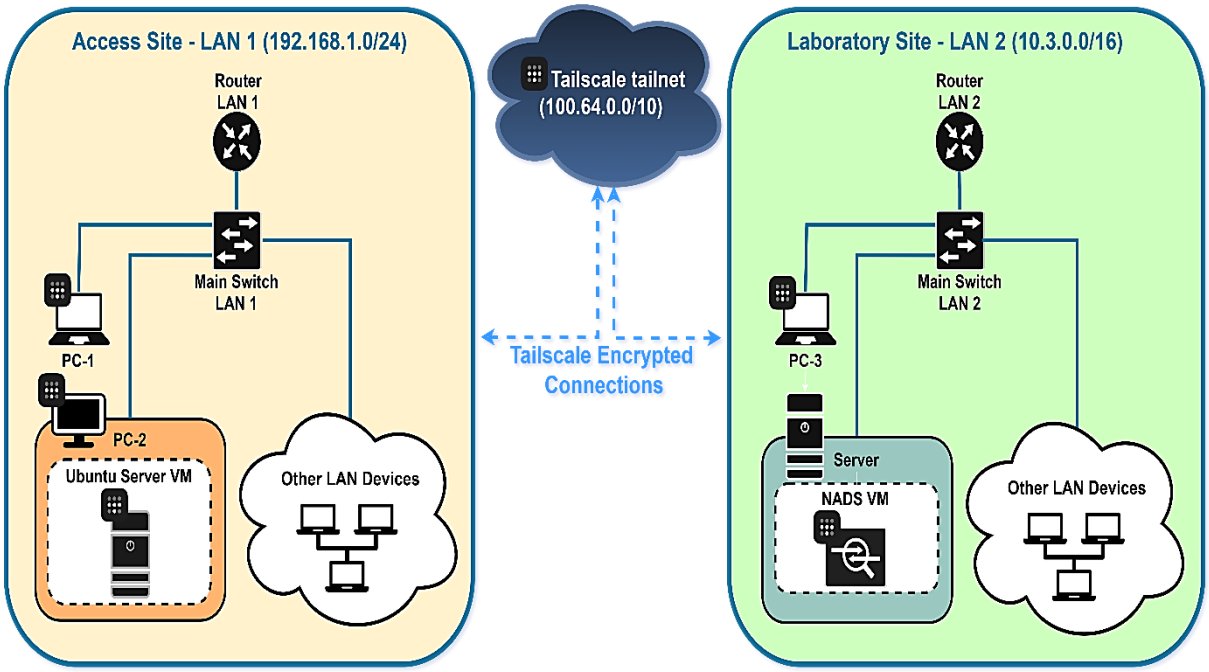


Fig. 1. *Logical network topology of the remote-access architecture*

On the other hand, the second LAN is the laboratory site which consists of a computer system PC-3, a physical server, and other local network devices. The Windows 11-based PC-3 system is the workstation that controls the laboratory site, specifically the NADS VM using VMware Workstation Pro 17. The Oracle Linux 9.7-based VM is hosted on the physical server with the aid of its virtualization software VMware ESXi 6.0.0 and runs the Security Onion security suite. The hardware details, detection logic, operation of the specified NADS architecture, as well as the physical server configuration are previously described and therefore omitted from this paper [6].

The implementation uses three IPv4 (Internet Protocol version 4) subnets:

- **192.168.1.0/24** – a subnet within the class C private range used by LAN 1, providing 256 IP (Internet Protocol) addresses from which 254 are assignable host addresses;
- **10.3.0.0/16** – a subnet within the class A private range used by LAN 2, providing 65,536 addresses from which 65,534 are assignable host addresses;
- **100.64.0.0/10** – a subnet within the CGNAT (Carrier-Grade Network Address Translation) private range used for the tailnet, providing 4,194,304 from which 4,194,302 are assignable host addresses;

The two LANs are private internal IPv4 address ranges reserved by the standard RFC 1918. In contrast, Tailscale uses the private virtual IP range. The three subnets within this range 100.100.0.0/24, 100.100.100.0/24, and 100.115.92.0/23 are reserved for internal system tasks [8][10][11].

The five endpoints PC-1, PC-2, PC-3, the Ubuntu server VM, and the NADS VM have the Tailscale client installed and form the Tailscale tailnet. The remaining routers, switches, and other local network devices in both geographically separated LANs, illustrated in Fig. 1, provide conventional LAN connectivity and are not part of the tailnet or the RustDesk setup. As a result, these devices are not the focus of this paper and are omitted.

To provide further visual clarity in Fig. 1 PC-3, NADS VM, and the other local network devices in LAN 2 are connected to a single main switch. However, in practice,

the physical deployment at the laboratory site has a complex multi-tier physical switching infrastructure. Its realization details are abstracted away to maintain focus on the core five endpoints connected in the tailnet.

B. Implementation of the Remote Access Solution

Before the planned Tailscale tailnet was realized as part of the proposed remote access implementation in this paper, the initial open-source solution WireGuard VPN was deployed in the Ubuntu server VM as a Docker container. The main goal was to connect the two geographically separated LANs. However, the solution was unsuccessful for our specific remote access scenario. During the troubleshooting phase the assigned WAN (Wide Area Network) address of the router in LAN 1 was from the 10.0.0.0/8 IP range. The results from executing route tracing on both PC-1 and the Ubuntu server VM revealed a hop using a private address within the same range that was not the WAN address of the router. These discoveries imply that LAN 1 has an additional NAT layer that is controlled by the ISP (Internet Service Provider). As a result, opening a port on the local router was ineffective because incoming traffic was being blocked long before reaching LAN 1. This failed WireGuard deployment and NAT constraints motivated our choice to adopt Tailscale as the VPN overlay for the proposed implementation in this paper [9].

The Tailscale setup is a straightforward process where all 5 endpoints after installing the client authorize their access to the same tailnet by logging in using a specific Tailscale account credentials. No further manual endpoint setup, firewall configurations, or local router port-forwarding rules were required. The endpoints get dynamically assigned IPv4 addresses within the 100.64.0.0/10 range. Tailscale expands upon WireGuard by adding NAT traversal, TCP (Transmission Control Protocol) transport capabilities, and access control policies. Finally, in this implementation Tailscale provides the encrypted connection between the endpoints and builds a mesh network topology, while RustDesk provides the remote-access service [8].

Within the Ubuntu server VM, the self-hosted RustDesk server was deployed using Docker Compose. Inside the docker configuration file with .yml file format the two containers hbbs and hbbr are defined, using the latest RustDesk server version image releases. The network mode is set to host and the restart policy to unless-stopped for both containers. This allows hbbs and hbbr to share the Ubuntu server VM's network stack and have automatic recovery of their services in situations where the containers are not intentionally stopped [12].

Additionally, the hbbs container starts up with the option -r followed by the assigned IPv4 address of the Ubuntu VM inside the tailnet. This command is executed in order to specify the exact IPv4 address of the RustDesk relay server that hhbs advertises to clients. The hbbr container, on the other hand, provides relay services in this implementation.

After installing the RustDesk client software on the four devices PC-1, PC-2, PC-3, and NADS VM the next phase performed manual network configuration from RustDesk settings on all clients. The tailnet IPv4 address of the Ubuntu server VM is used as the IP for both the RustDesk ID (Identification) server (hbbs), also called rendezvous or signaling server and relay server (hbbr) fields. However, to ensure session encryption the RustDesk 32-byte public key stored in the file id_ed25519.pub is also entered in the key field for a successful client-side identification. This key is automatically generated alongside the 64-bytes private key (id_ed25519), during the initial startup of the RustDesk server, building the unique server key pair. This configuration of the RustDesk server and clients direct all RustDesk traffic to the private Ubuntu server VM rather than public RustDesk servers. Without the self-hosted deployment and network configuration in the proposed implementation the client endpoints would connect through

RustDesk's public ID and relay servers, creating potential privacy risks and third-party dependence [4].

The primary remote-access workflow begins with PC-1 connecting to PC-3 for starting, configuring, updating, or shutting down the NADS VM by using the VMWare Workstation Pro 17 software. After the VM reaches operational state for analysis, monitoring, experimental, and other type of sessions, PC-1 closes the connection with PC-3 and switches to a direct connection with the VM. All established sessions using the implementation depend on the Tailscale clients running on the endpoints, the RustDesk clients to be configured to reach the RustDesk server, and the Ubuntu server VM to remain powered on.

C. Experimental Results

The summary of the evaluation of the proposed remote-access implementation is presented in Table 1.

Table 1. Performance evaluation of the remote-access implementation

Test ID	Test Category	Source → Target	Test Window	Primary Result	Supporting Result Details
1	WAN interface speed test	PC-1 → Ookla Server	5 runs	Download: 103.99 Mbps; Upload: 23.25 Mbps	Internet latency: 41.10 ms; Jitter: 10.84 ms; No packet loss reported.
2		PC-3 → Ookla Server	5 runs	Download: 94.45 Mbps; Upload: 94.46 Mbps	Internet latency: 9.98 ms; Jitter: 0.16 ms; No packet loss reported.
3-A	Inter-LAN path	PC-1 → PC-3	5 runs	Connection used the Frankfurt DERP relay server.	Direct P2P: Failed; Latency range: 89–348 ms; Median latency: ~115–120 ms.
3-B	Intra-LAN path	PC-1 → PC-2	5 runs	Direct P2P connection was established.	Locally via port: 41641 on PC-2; Median latency: 3 ms.
4-A	Inter-LAN path	PC-1 → NADS VM	5 runs	Connection used the Frankfurt DERP relay server.	Direct P2P: Failed; Latency range: 91–182 ms; Median latency: ~112–118 ms.
4-B	Intra-LAN path	PC-3 → NADS VM	5 runs	Direct P2P connection was established.	Locally via port: 41641 on NADS VM; Median latency: 1 ms.
5-A	Inter-LAN latency	PC-3 → Ubuntu VM	50 packets	No packets were lost.	Avg. latency: 117 ms; range: 90–222 ms.
5-B	Intra-LAN latency	PC-3 → NADS VM	50 packets	No packets were lost.	Avg. latency: 5 ms; main range 1–2 ms with one packet spiking at 199 ms.
6-A	Inter-LAN latency	PC-1 → NADS VM	50 packets	No packets were lost.	Avg. latency: 119 ms; range: 92–252 ms.
6-B	Intra-LAN latency	PC-1 → PC-2	50 packets	No packets were lost.	Avg. latency: 3 ms; range: 2–11 ms.
7	Tailscale relayed TCP throughput	PC-1 → PC-3	5 runs (30s)	Avg. throughput: 2.61 Mbps.	Sender CPU usage: 0.61%.
8		PC-3 → PC-1	5 runs (30s)	Avg. throughput: 6.13 Mbps.	Sender CPU usage: 1.12%.
9	RustDesk connectivity	PC-1 → Ubuntu VM	5 runs across three ports	100% Reachable (All 15 TCP checks succeeded).	Ports 21115, 21116 and 21117: 19.22, 16.35 and 17.75 ms median latency.
10	RustDesk response timing		10 runs across three ports	100% Reachable (All 30 TCP checks succeeded).	Ports 21115, 21116 and 21117: 17.94, 16.61 and 15.69 ms median latency.
11	RustDesk remote session stability	PC-1 → PC-3	60 probes (10 min)	98.33% Reachable (59/60 probes succeeded).	Median latency: 95.87 ms; Avg. latency: 110.87 ms; One brief timeout recovered automatically.
12	RustDesk service uptime and health check	Ubuntu VM → RustDesk Server (hbbs/hbbr)	Over 14 hours.	100% Uptime.	Both hbbs and hbbr containers recorded zero restarts.
13	RustDesk event log and exception audit		2-hour session observation	No errors or disconnections.	The logs did not contain fatal errors or daemon exceptions across both hbbs and hbbr containers.
14	RustDesk server resource utilization		12 idle and 12 active session samples	Minimal resource utilization.	Idle: ~0.04% CPU / ~15.0 MiB RAM; Active: ~0.31% CPU avg (1.02% peak) / ~15.4 MiB RAM.

The carried-out tests from Table 1 cover WAN (Wide Area Network) interface connection, Tailscale intra and inter-LAN path selection, latency and TCP (Transmission Control Protocol) throughput, as well as RustDesk's service (hbbr and hbbs) reachability and health, session stability and server resource utilization.

The Ookla WAN speed tests highlighted the difference in the network links of the two sites (Tests 1 and 2). PC-1 (LAN 1) achieved throughput of 103.99 Mb/s download and 23.25 Mb/s upload, 41.10 ms latency and 10.84 ms jitter. In contrast, PC-3 (LAN 2) achieved near-symmetrical speed of 94.45 Mb/s download and 94.46 Mb/s upload, noticeably lower latency of 9.98 ms and 0.16 ms jitter. As a result, the performance evaluations of any cross-site tests heavily depended on from which LAN the connection was initiated, with LAN 1 showing significantly lower upload speed and higher jitter than LAN 2.

Direct P2P connections with minimal latency of 1-3 ms were established with no packet loss by Tailscale via UDP (User Datagram Protocol) port 41641 of the target devices in Tests 3-B and 4-B. However, P2P inter-LAN routing paths from PC-1 to both PC-3 and the NADS VM (Tests 3-A and 4-A) failed due to the existence of NAT constraints, forcing Tailscale to fall back and use the closest DERP (Designated Encrypted Relay for Packets) relay server with the least amount of latency located in Frankfurt to establish encrypted connection. This automatic fallback mechanism influenced the latency of the inter-LAN paths, confirming the network constraints were overcome by performing NAT traversal [8].

Furthermore, Tests 5 and 6 which were carried out with the help of ICMP (Internet Control Message Protocol) via the ping command also highlight the occurrence of the high latency event. The results of Test 6-A show successful transmission of 50 packets with no loss, but a high avg. latency of 119ms. Test 5-B also reports no packet loss but a noticeably lower avg. latency of 5 ms. During the execution of this test an anomaly occurred where one of the packet's latency spiked to 199 ms while the rest 49 transmissions had latency ranging from 1 to 2 ms.

In addition, the relayed TCP throughput of Tests 7 and 8 averaged 2.61 Mb/s from LAN 1 to LAN 2 and avg. 6.13 Mb/s for the reverse connection, with zero retransmissions during both 30-second tests. The throughput difference between the results of both tests shows direct relation with the WAN upload speed capacities, specifically the first LAN's 23.25 Mb/s limit. While these tests confirm functional transfer between the networks, they do not directly address and measure the graphical screen (video) rendering of the remote sessions, its performance nor responsiveness.

The reachability of the RustDesk hbbs and hbbr services, was verified by initiating TCP checks on 3 specific ports 21115, 21116 and 21117 on the Ubuntu VM. All 45 probes across Tests 9 and 10 were successful with a median response time between 15.69 and 19.22 ms. The three ports have the important role of supporting RustDesk's server signaling and relay functionality [4].

In order to establish remote connections, the source client devices used the destination RustDesk client ID, an access one-time password, and the server's public key (id_ed25519.pub). After this initial setup the implementation is in a ready state for remote-access sessions. During the remote session in Test 11 the reachability probes reached 98.33% success rate with only a single timeout which automatically recovered. The avg and median latencies as expected from the previous tests were rather high (95.87 and 110.87 ms). The services remained operational for more than 14 hours without restarts of the hbbr or hbbs containers (Test 12). After reviewing logs from the 2-hour session in Test 13 no fatal errors, daemon exceptions, or disconnections were present. Finally, in Test 14 the resource usage of the RustDesk server remained low. The CPU utilization slightly increased from approximately 0.04% while

no remote-access sessions were running (idle state) to 0.31% during one active remote session where the hbbr service usage peaked at 1.02%. Additionally, the RAM memory utilization also increased from approximately 15.00 to 15.40 MiB.

The experimental results showcase the primary operational trade-off of this solution. Direct intra-LAN connections have low latencies of 1-3 ms, but the inter-LAN routing paths have to fall back to the Frankfurt DERP relay which significantly worsens the average latency of approximately 115-120 ms. Alongside this relay-based latency another key trade-off is the noticeable TCP throughput asymmetry in result of the different WAN interface upload speeds from the two networks.

The proposed implementation's ability to scale and deployment capability relies on several factors like the hardware resources allocated to the Ubuntu server VM, the available IP address range for host devices, the network bandwidth, and the number of simultaneous remote sessions.

Overall, Tailscale's DERP relay fallback and NAT traversal capabilities preserved inter-LAN encrypted connection despite the failed direct P2P routing paths, while RustDesk provided stable and secure remote sessions between the two sites without significant resource overhead of the current Ubuntu VM RAM and virtual CPU cores allocation.

The evaluation scope embraces network, server service and infrastructure behavior rather than the visual remote-desktop quality experienced by the users. The experimental results demonstrate the operational viability, network security, and connection stability of the proposed implementation under the tested NAT, ISP, router and firewall conditions of both LANs. However, they do not establish a universal low-latency or superior remote-access performance because video frame rates and user input (e.g., mouse and keyboard) responsiveness and latency were not measured as they are planned for future work alongside the file transfer capabilities of the implementation.

CONCLUSION

This paper presented and evaluated a self-hosted and security-oriented architecture for remote access that brings together Tailscale and RustDesk for managing a virtualized NADS across two geographically separated and operationally distinct networks. The main contribution of this study is the design and empirical evaluation of the layered remote-access implementation with encrypted connections and self-hosted service capabilities under diverse WAN links, ISP routing, NAT-based constraints, and cross-site network architecture conditions. By assigning the network overlay transport to Tailscale and the remote access to RustDesk, operating on the application layer of the OSI (Open Systems Interconnection) model, the implementation's results confirm that the proposed design maintains stable operation with administrative reachability without intentional device port exposure on both networks to achieve the remote connection services goal.

Despite the evident operational trade-offs of the implementation, the RustDesk service checks (Tests 9-14) recorded full reachability, minimal resource utilization of the Ubuntu server VM (0.31% CPU and 15.4 MiB RAM usage, during one active remote session) all of the tested and connected devices, part of the proposed implementation remained stable and operational. These findings indicate that the proposed architecture sets a practical baseline for deploying a remote-access solution in experimental or enterprise environments where secure access, service ownership and reliability of the encrypted connection across the theoretical and practical real-world boundaries take priority over achieving minimum overall latency.

Future work includes executing an experimental phase focusing on measuring remote user desktop responsiveness, frame rate stability, server and user performance during multiple simultaneous remote sessions, RustDesk client input latencies and file transfer performance, in order to define

the implementation's practical capacity. Further investigation of the observed NAT constraints causing the relay fallback and running in-depth port scans to further verify the security and public exposure assumptions is also planned. Finally, future study and evaluation may also include other open-source solutions such as Headscale, which is the self-hosted alternative of the Tailscale control server to achieve greater privacy and data control [13].

ACKNOWLEDGMENTS: *This research was funded by the University Center for Research and Technology at the Technical University of Gabrovo, project NIP2026-6 "Study and Development of a Computer Networks Anomaly Detection System".*

REFERENCE

[1] AnyDesk, The fast remote desktop application – AnyDesk available at: https://anydesk.com/en, 2026 (accessed September 26, 2026).

[2] Splashtop, Splashtop: Secure Remote Access & Remote Support Software available at: https://www.splashtop.com/, 2026 (accessed September 26, 2026).

[3] TeamViewer, TeamViewer – The Digital Workplace Platform available at: https://www.teamviewer.com/en/, 2026 (accessed September 26, 2026).

[4] RustDesk, RustDesk Documentation - Open Source Remote Desktop available at: https://rustdesk.com/docs/en/, 2026 (accessed 26 September 2026).

[5] Bernstein DJ, Lange T, Schwabe P. The security impact of a new cryptographic library. In: Progress in cryptology – LATINCRYPT 2012, Lecture Notes in Computer Science, vol. 7533, 2012, p. 159-176. available at: https://doi.org/10.1007/978-3-642-33481-8_9, 2026 (accessed 26 September 2026).

[6] Ivanova V, Genkov D. A machine learning-based network anomaly detection system prototype using Isolation Forest. In: Engineering Proceedings, vol. 154, 2026, art. 7. available at: https://doi.org/10.3390/engproc2026154007, 2026 (accessed 26 September 2026).

[7] HelpWire, HelpWire: Free and simple remote support available at: https://www.helpwire.app/, 2026 (accessed 26 September 2026).

[8] Tailscale, Tailscale Documentation available at: https://tailscale.com/docs/, 2026 (accessed 26 September 2026).

[9] Donenfeld JA. WireGuard: next generation kernel network tunnel. In: Proceedings of the Network and Distributed System Security Symposium (NDSS 2017), 2017 available at: https://doi.org/10.14722/ndss.2017.23160, 2026 (accessed 26 September 2026).

[10] Rekhter Y, Moskowitz B, Karrenberg D, de Groot GJ, Lear E. RFC 1918: BCP 5: Address Allocation for Private Internets, 1996. available at: https://www.rfc-editor.org/info/rfc1918/, 2026 (accessed 26 September 2026).

[11] Генков, Д., Основи на компютърните мрежи, ISBN 978-619-7071-61-0, Издателство „Медиатех" – Плевен, 2014 // Genkov, D., Osnovi na kompyutarnite mrezhi, ISBN 978-619-7071-61-0, Izdatelstvo „Mediateh" – Pleven, 2014 available at: http://delian.genkovi.com/netbook/netbook.php, 2026 (accessed 26 September 2026).

[12] Docker, Docker Documentation available at: https://docs.docker.com/, 2026 (accessed 26 September 2026).

[13] Headscale, Headscale available at: https://headscale.net/stable/, 2026 (accessed 26 September 2026).