Design Report: A Custom Tool for Simulating TCP SYN Flood Denial-of-Service Attacks with IP Source Spoofing

[Arnab Dey], [Mushfiqur Rahman]

July 23, 2025

Authored by

Name	**Student ID**
Arnab Dey	2005112
Mushfiqur Rahman	2005107

Contents

1 Introduction

This report presents the comprehensive design of a custom software tool developed for the purpose of simulating a TCP SYN Flood Denial-of-Service (DoS) attack. The tool is designed to be a practical instrument for academic study, enabling a deeper understanding of fundamental vulnerabilities within the Transmission Control Protocol (TCP). By generating a high-volume stream of spoofed connection requests, the tool demonstrates the principle of resource exhaustion, a common vector for DoS attacks. This document details the theoretical basis of the attack, the strategic design choices, the precise structure of the network packets to be crafted, and a robust justification for the design's anticipated effectiveness against systems not employing advanced mitigation techniques.

2 Definition and Topology of the TCP SYN Flood Attack

2.1 The TCP Three-Way Handshake

The Transmission Control Protocol (TCP) is the bedrock of reliable, connection-oriented communication across the internet, underpinning services from the World Wide Web (HTTP) to email (SMTP). Its reliability stems from a mandatory connection establishment process known as the three-way handshake. This process ensures both client and server are ready to communicate, synchronizing their initial sequence numbers to facilitate ordered, error-checked data transfer. The procedure unfolds in three distinct steps:

1. **SYN**:A client wishing to establish a connection sends a TCP segment to the server with the SYN (Synchronize) control flag set to 1. This initial packet contains no data payload but includes a crucial piece of information: a randomly generated 32-bit Initial Sequence Number (ISN), let's call it x. This ISN marks the starting point for the sequence of bytes the client will send.

2. **SYN-ACK**: Upon receiving the SYN segment, a listening server transitions the potential connection into a SYN-RECEIVED state. It allocates resources for this nascent connection within a special memory structure, the Transmission Control Block (TCB), which is placed in a backlog queue. The server then replies to the client with a TCP segment where both the SYN and ACK (Acknowledge) flags are set to 1. This SYN-ACK packet contains the server's own randomly generated ISN, let's call it y, and an acknowledgment number set to $x + 1$. The acknowledgment number confirms to the client that its initial SYN packet was successfully received.

3. **ACK**: A legitimate client, upon receiving the SYN-ACK, completes the handshake by sending a final ACK segment to the server. This packet has the ACK flag set to 1, its sequence number is now $x+1$, and its acknowledgment number is set to $y+1$, confirming receipt of the server's SYN-ACK. When the server receives this final ACK, it moves the connection's TCB from the backlog queue to the established connection table,

transitioning its state from SYN-RECEIVED to ESTABLISHED. The connection is now fully open, and data transfer can commence.

The vulnerability exploited by a SYN flood attack lies not in a flaw or bug in this process, but in the protocol's inherent, foundational assumptions. TCP was designed in an era where network actors were presumed to be cooperative. The protocol trusts that a client initiating a connection (Step 1) is genuine and will complete the handshake (Step 3). It therefore commits a finite resource—a slot in its connection backlog—based solely on this initial, unverified request.

2.2 Exploiting the Handshake: The SYN Flood "Half-Open" Attack Mechanism

A TCP SYN flood is a form of resource-exhaustion Denial-of-Service (DoS) attack that directly targets the finite capacity of a server's connection backlog queue. The attack is commonly referred to as a "half-open" attack because it systematically initiates the three-way handshake but intentionally prevents it from ever reaching completion, leaving a multitude of connections perpetually in the SYN-RECEIVED state.

The mechanism is deceptively simple. An attacker sends a massive volume of TCP SYN packets to a victim server's open port(s). The server, adhering strictly to the TCP protocol specification, responds to each SYN request with a corresponding SYN-ACK packet. Concurrently, for each of these requests, it allocates a TCB in its backlog queue and starts a timer, awaiting the final ACK from the client.

The attacker, however, is designed to ensure this final ACK never arrives. By withholding the third step of the handshake, the attacker forces the server to keep these half-open connections in its memory. Each entry consumes resources and occupies a slot in the connection table. The attack's success hinges on sending SYN packets at a rate that outpaces the server's ability to clear out these stale entries after their timeout period expires. Once the backlog queue is completely filled with these bogus half-open connections, the server has no resources left to process new, legitimate connection requests. Any subsequent SYN packet from a valid user is simply dropped, effectively denying them service and rendering the targeted application or server unavailable.

2.3 Attack Variants and Our Chosen Strategy: IP Source Spoofing

Research identifies three primary methods for executing a SYN flood, each with distinct characteristics and levels of sophistication :

- **Direct Attack**: The attacker uses their real, unaltered IP address to send the flood of SYN packets. While simple to implement, this method is also the easiest to defend against, as administrators can quickly identify the single source of the attack and block it with a simple firewall rule.
- **Distributed Attack (DDoS)**: A far more potent variant, this attack is launched in a coordinated fashion from a large network of compromised computers, known as

a botnet. The sheer volume and distributed nature of the attack make it extremely difficult to trace back to the perpetrator and mitigate effectively.

- **Spoofed Attack**:The attacker forges, or "spoofs," the source IP address in the header of each SYN packet they send. This makes the packets appear to originate from a different, often non-existent or unreachable, machine.

2.4 Attack Topology Diagram

The following diagram illustrates the network topology and packet flow for the designed spoofed SYN flood attack. It visually represents the one-way flood of malicious requests and the misdirection of the server's legitimate responses.

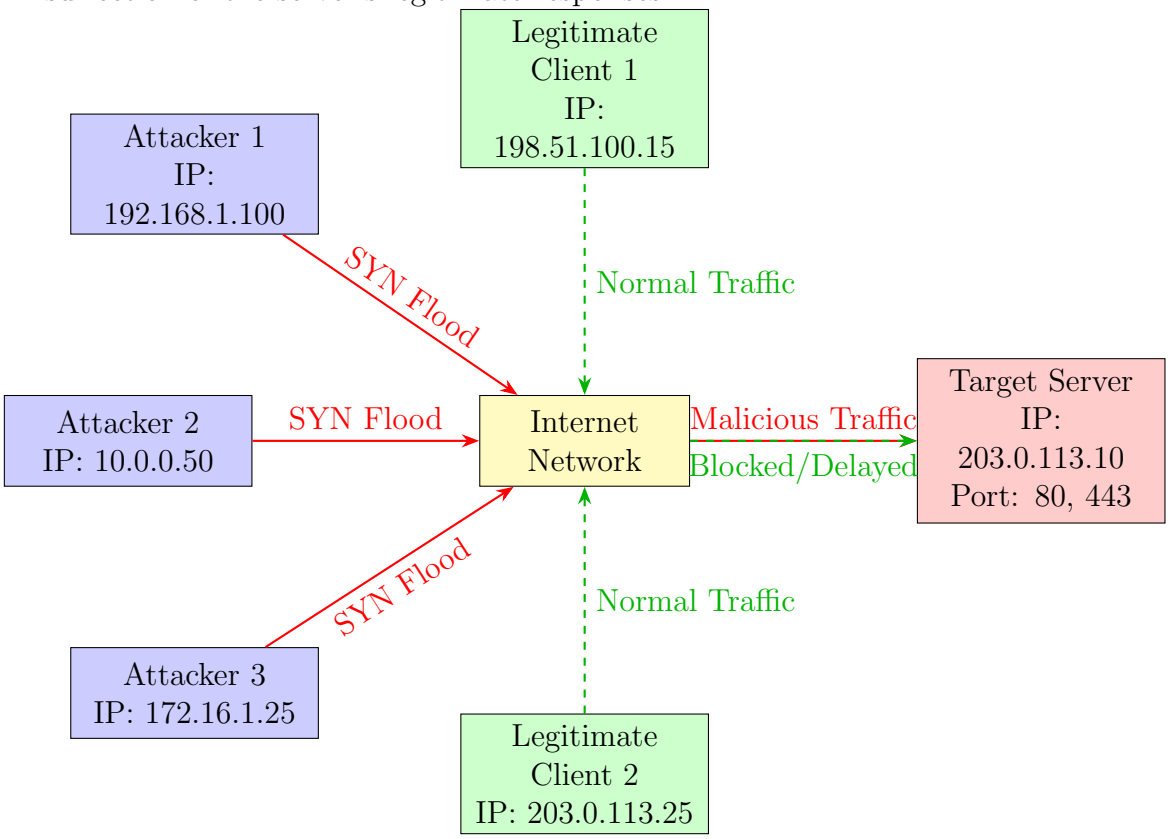


Diagram Explanation

1. **Attacker 1, Attacker 2, Attacker 3 (purple boxes):** These are the machines under the attacker's control, each with its own IP address. They each send a continuous flood of TCP SYN packets ("*SYN Flood*"), shown as solid red arrows toward the Internet node. By never completing the three-way handshake, they exhaust the server's connection backlog.

2. **Internet Network (yellow box):** This represents the routing infrastructure that forwards both malicious and legitimate traffic to the target server. Because the SYN

floods use spoofed source IPs, the Internet node cannot distinguish them from normal client requests.

3. **Legitimate Client 1 and Client 2 (green boxes):** These are genuine users with valid IPs. Their requests are shown as dashed green arrows labeled "*Normal Traffic*." During the flood, their SYNs are either delayed or dropped (dashed green arrow labeled "*Blocked/Delayed*") once the server's backlog is full.

4. **Target Server (red box):** At IP 203.0.113.10 (ports 80, 443), the server must allocate a Transmission Control Block for each incoming SYN.

   - *Malicious Traffic*: Shown as solid red arrows, representing the spoofed SYN flood.
   - *Blocked/Delayed Traffic*: Shown as dashed green arrows, representing legitimate client SYNs that are refused once the backlog is exhausted, resulting in a denial of service.

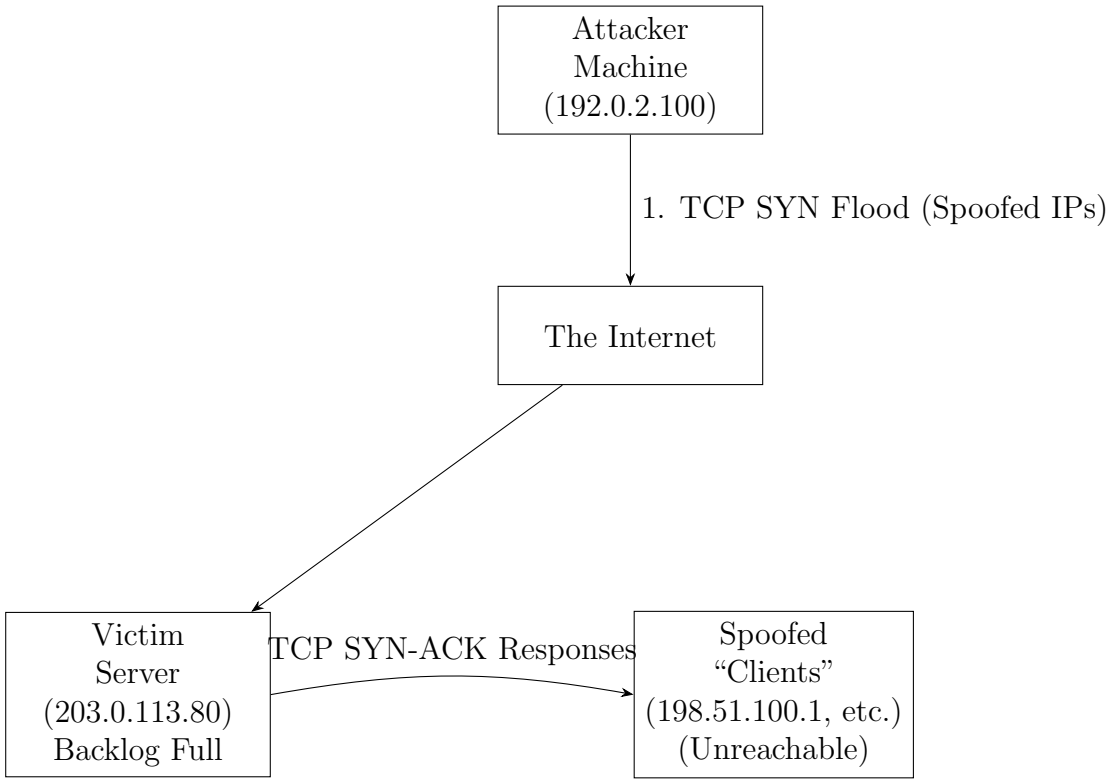


Figure 1: Topology of a TCP SYN Flood Attack with IP Source Spoofing

Diagram Explanation

1. **Attacker Machine:** This is the host running our custom-designed attack tool. It generates a high-volume stream of TCP SYN packets.

2. **Victim Server:** This is the target system. Its TCP port (e.g., port 80 for a web server) is the destination for the SYN flood.

3. **Spoofed "Clients":** These are not real machines but conceptual representations of the forged source IP addresses used by the attacker. These addresses are deliberately chosen to be unreachable.

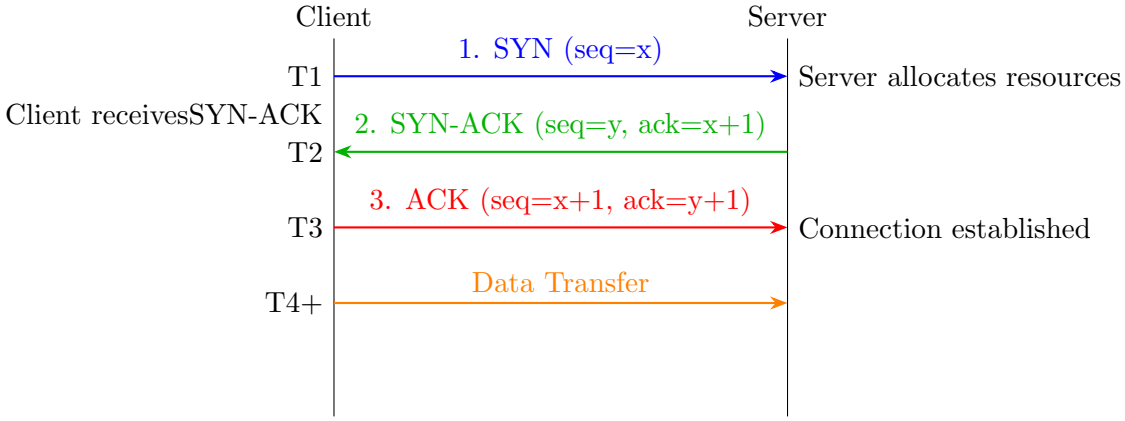
4. **Flow 1 (SYN Flood):** The attacker sends a continuous flood of TCP SYN packets towards the Victim Server. Crucially, each packet has its source IP address forged to appear as if it's coming from one of the non-existent Spoofed Clients.

5. **Flow 2 (Misdirected SYN-ACKs):** The Victim Server, following protocol, responds to each incoming SYN with a SYN-ACK. However, these responses are directed to the spoofed IP addresses. Since these "clients" do not exist or are unreachable, the SYN-ACKs are lost, and the server never receives the final ACK required to complete the handshake. This leads to the server's connection backlog becoming saturated with half-open connections, ultimately causing a denial of service.

3 Protocol and Attack Timing Diagrams

3.1 Timing Diagram of a Standard TCP Connection

To fully appreciate the attack's mechanics, it is essential to first visualize the timing and sequence of a legitimate TCP connection. The following sequence diagram illustrates the successful three-way handshake.



3.2 Timing Diagram of the Spoofed SYN Flood Attack

In stark contrast, the timing diagram for our designed attack shows a parallel, unrequited flood of requests that exploits the server's willingness to enter the SYN-RECEIVED state.

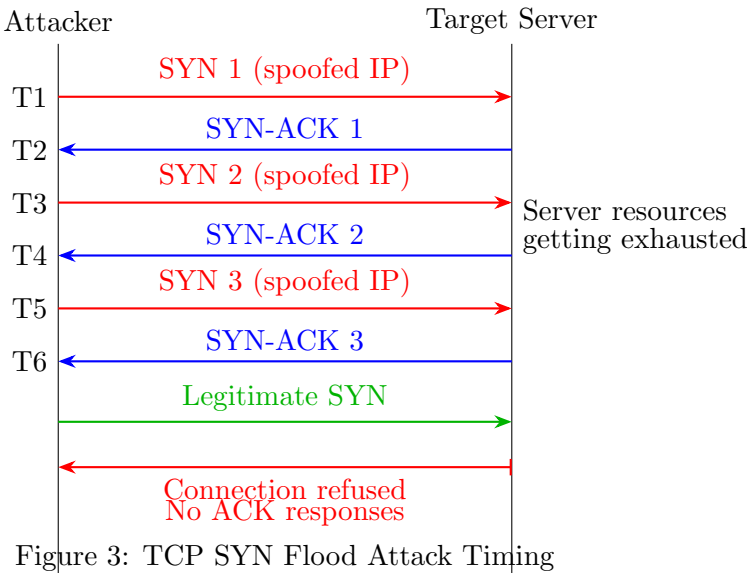


Figure 3: TCP SYN Flood Attack Timing

This diagram highlights the core of the attack: the Attacker sends a continuous stream of SYN packets, but the Victim Server's SYN-ACK responses are directed to unreachable entities (SC1, SC2, etc.). The absence of any returning ACK packets is the key element that leads to resource exhaustion. The attack's success is predicated on the asymmetry of work: the attacker performs a simple, stateless action (sending a small packet), while the server is forced into a complex, stateful reaction (allocating memory, generating a response, and managing a timer). This resource amplification means an attacker with relatively modest bandwidth can impose a significant load on a powerful server, targeting its state-table capacity rather than its network pipe.

3.3 Core Attack Strategies in Design

To effectively execute the attack illustrated above, the design of our tool incorporates several key strategies:

1. **IP Address Spoofing:** This is the central tactic. The tool will not use a single spoofed IP but will generate a new, random source IP address for each SYN packet sent. The primary purpose of this is not anonymity, but to guarantee that the victim server's SYN-ACK responses are undeliverable. If a SYN-ACK were to reach a real machine that did not initiate the connection, that machine's TCP stack would respond with a Reset (RST) packet, which would prematurely close the half-open connection on the victim server and defeat the purpose of the attack. By using random, likely non-existent IP addresses, we ensure the SYN-ACKs vanish, forcing the server to wait for the full timeout period.

2. **Source Port Randomization:** In addition to spoofing the source IP, the tool will randomize the TCP source port for each packet, typically selecting a value from the ephemeral port range (1024-65535). This adds a minor layer of complexity to the attack traffic, making it harder to filter based on simple IP:port pair rules and more closely mimicking legitimate, diverse network traffic.

3. **High-Rate Packet Transmission:** The effectiveness of a SYN flood is a function of rate. The tool must be designed to generate and transmit SYN packets at a rate sufficient to fill the target's backlog queue before existing entries time out. The critical relationship is:

$$\text{Attack Packet Rate} > \frac{\text{Backlog Size}}{\text{Connection Timeout}}$$

Therefore, the tool's implementation must utilize a high-speed, non-blocking sending loop to achieve the necessary packets-per-second.

4 Attack Packet and Frame Construction

4.1 Custom Tooling Philosophy and Implementation Choice

To comply with the strict project requirement of designing a fully custom attack tool without relying on prebuilt libraries (such as Scapy), our implementation leverages Python's `socket` module with the `SOCK_RAW` interface. This approach necessitates the complete manual construction of both IP and TCP headers, including manual computation of checksums, offering full control over every byte of the packet structure.

This design is highly instructive from a security engineering standpoint, as it exposes the intricacies of packet crafting and network stack interactions that higher-level tools abstract away.

4.2 Layer 3: IP Header Construction

The IPv4 header is manually assembled as a 20-byte binary structure following RFC 791. Each field is explicitly defined:

- **Version & IHL:** 0x45 indicating IPv4 with a 5-word (20-byte) header.
- **Type of Service:** 0x00 (default).
- **Total Length:** Set to 40 bytes (20 bytes IP + 20 bytes TCP).
- **Identification:** Randomized per packet for minor obfuscation.
- **Flags & Fragment Offset:** 0x4000 (Don't Fragment).
- **TTL:** 64, sufficient for typical routing.
- **Protocol:** 6 for TCP.
- **Header Checksum:** Calculated over the IP header.
- **Source Address:** A randomly generated spoofed IP address.
- **Destination Address:** The victim server's IP.

4.3 Layer 4: TCP Segment Construction

The TCP header is similarly built as a 20-byte binary sequence following RFC 793. Fields include:

- **Source Port:** Randomized ephemeral port (1024–65535).
- **Destination Port:** Typically 80 or 443 for web servers.
- **Sequence Number:** Randomized initial sequence.
- **Acknowledgment Number:** 0, since this is a SYN.
- **Data Offset:** 5 (no TCP options).
- **Flags:** Set to `0x02` (SYN).
- **Window Size:** 5840, typical default.
- **Checksum:** Computed manually over a pseudo-header + TCP header.
- **Urgent Pointer:** 0.

A key detail is the construction of the TCP checksum, which requires a pseudo-header incorporating fields from the IP layer (source and destination addresses, protocol, TCP length) concatenated with the actual TCP segment.

4.4 Packet Assembly and Transmission

The complete attack packet is constructed by concatenating the IP header and TCP header binary strings. Each packet is then sent via a raw socket:

```
s = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_RAW)
s.sendto(packet, (victim_ip, 0))
```

4.5 Summary of Frame Modifications

- **IP Header:** Spoofed source IP, correct checksum, manual assembly.
- **TCP Header:** Randomized source port, SYN flag set, checksum computed over pseudo-header.
- **Operating System Interaction:** Raw sockets require superuser privileges and typically disabling kernel-level RST responses via `iptables` to prevent interfering resets.

This approach guarantees that every byte of the packet is directly controlled by our implementation, ensuring compliance with the project's requirement to craft frames entirely from scratch.

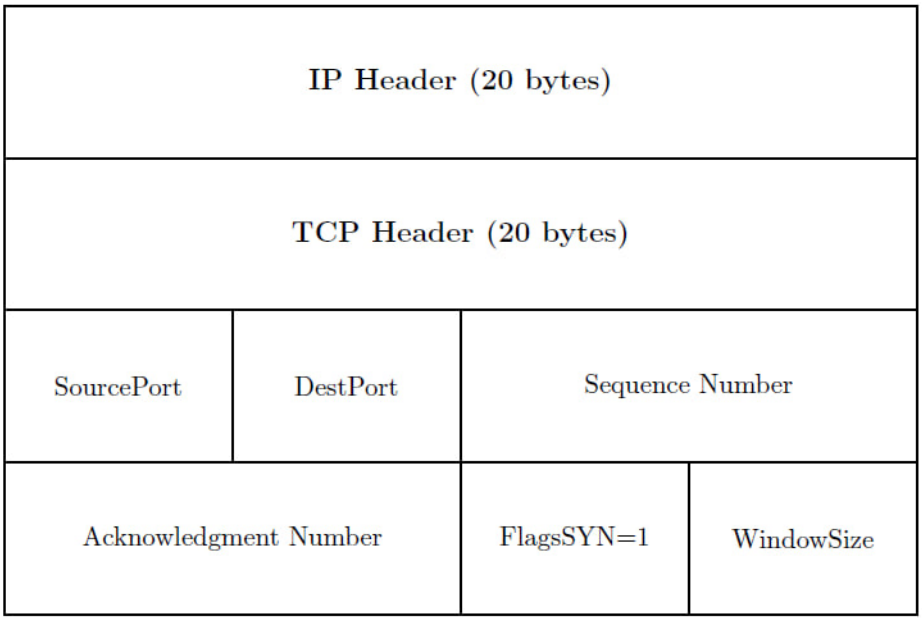


Figure 2: TCP Three-Way Handshake Sequence Diagram

4.6 Overcoming OS-Level Interference

A critical, practical detail in designing such a tool is accounting for the behavior of the attacker's own operating system kernel. If a spoofed source IP address happens to belong to another host on the same local network, that host will receive the victim's SYN-ACK. Not having initiated a connection, its TCP stack will correctly send a TCP Reset (RST) packet back to the victim. This RST would close the half-open connection, effectively neutralizing that specific attack packet.

To ensure the attack's efficacy in a lab environment, the tool's operational procedure must include a step to prevent the attacker's own OS from sending these RSTs. This is accomplished by adding a firewall rule on the attacker's machine *before* launching the script. For a Linux-based system, this is achieved with a simple `iptables` command:

```
# This command must be executed on the attacker's machine prior to running the tool.
# It creates a rule in the OUTPUT chain to DROP any outgoing TCP packets that
# have the RST flag set and are destined for the victim's IP address.
sudo iptables -A OUTPUT -p tcp --tcp-flags RST RST -d 203.0.113.80 -j DROP
```

Inclusion of this step is non-negotiable for a robust design and demonstrates a comprehensive understanding of the full network dynamics involved, including the potential for the attacker's own system to interfere with the attack.

5 Conclusion on Design Viability

The proposed design for a custom TCP SYN flood tool is highly likely to be effective under the specified laboratory conditions: a target system where advanced countermeasures, par-

ticularly SYN Cookies, have been intentionally disabled for academic study. The strategic use of high-rate packet transmission combined with IP source address spoofing directly and efficiently exploits the finite nature of the target's TCP connection backlog queue.

By leveraging the protocol's own rules regarding stateful connection establishment, the tool provides a powerful demonstration of a classic and foundational denial-of-service vulnerability. Its ultimate value extends beyond simply causing a DoS condition; it serves as a standardized, controllable instrument for security education and for testing and validating the very defenses that have been engineered in response to this persistent threat to network availability.

References

[1] Cloudflare, "What is a SYN flood DDoS attack?," *Cloudflare Learning Center*. [Online]. Available: https://www.cloudflare.com/learning/ddos/syn-flood-ddos-attack/

[2] Akamai, "What Are SYN Flood DDoS Attacks?," *Akamai Glossary*. [Online]. Available: https://www.akamai.com/glossary/what-are-syn-flood-ddos-attacks

[3] A. Alshammari and A. K. Singh, "Design of a New Dynamic TCB Timeout Algorithm for TCP SYN Flooding Attack Mitigation," *Sensors*, vol. 23, no. 8, p. 3817, Apr. 2023.

[4] S. Singh, S. K. Dhurandher, and I. Woungang, "A Survey on TCP SYN Flood Attack: Its Types, and Mitigation Techniques," *International Journal of Computer and Network Security*, vol. 5, no. 8, pp. 1-8, Aug. 2013.

[5] M. V. Mäffert, S. D. Mainka, and G. D. Ditty, "Investigating Mitigation Techniques for TCP SYN Floods," in *2019 IFIP Networking Conference (Networking)*, Warsaw, Poland, 2019, pp. 1-9.

[6] A. H. Lashkari, A. F. A. Kadir, and H. Gonzalez, "TCP SYN Flood Attack Detection and Prevention Using Adaptive Thresholding," in *ITM Web of Conferences*, vol. 40, p. 01016, 2021.

[7] A. S. Kapadia, "In-depth: SYN Flooding," *University of Toronto CSC427*, 2015. [Online]. Available: http://www.cs.toronto.edu/ arnold/427/15s/csc427/indepth/syn-flooding/index.html

[8] A. Kuzmanovic and E. W. Knightly, "Low-Rate TCP-Targeted Denial of Service Attacks," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '03)*, 2003, pp. 75–86.

[9] D. J. Bernstein, "SYN cookies," 1996. [Online]. Available: http://cr.yp.to/syncookies.html

[10] W. Eddy, "TCP SYN Flooding Attacks and Common Mitigations," *RFC 4987*, Internet Engineering Task Force (IETF), Aug. 2007. [Online]. Available: https://www.rfc-editor.org/info/rfc4987

[11] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow," in *2010 IEEE Globecom Workshops*, Miami, FL, USA, 2010, pp. 1-6.

[12] S. A. H. Mohsan, A. A. Khan, and M. A. Al-sharif, "A Machine Learning Based Approach to Detect TCP-SYN Flood Attack using Unsupervised Method," *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 8, 2020.

[13] H. Wang, D. Zhang, and K. G. Shin, "Detecting SYN flooding attacks," in *Proceedings of the IEEE INFOCOM 2002*, vol. 3, New York, NY, USA, 2002, pp. 1530-1539.

[14] M. Z. Abdullah, A. S. Al-Sumaidaee, and S. M. Said, "A Review of TCP SYN Flood DDoS Attack in IoT," in *2021 International Conference on Computer Science and Software Engineering (CSASE)*, Duhok, Iraq, 2021, pp. 13-18.

[15] J. Lemon, "Resisting SYN flood DoS attacks with a SYN cache," in *Proceedings of the USENIX BSDCon 2002*, San Francisco, CA, USA, 2002, pp. 89-97.

[16] P. Biondi, "Scapy: A Python-based Interactive Packet Manipulation Program," *Scapy Documentation*, 2020. [Online]. Available: https://scapy.net