IP/TCP in Go - Adam Bredvik and Emily Ye

Abstract

This project creates a virtual networking stack, combining both Internet Protocol (IP) and Transmission Control Protocol (TCP) implementations to facilitate reliable communication between hosts over a virtual network. We begin by developing an IP layer in the Go programming language, and extend it with a detailed and RFC-compliant TCP implementation. The project is broken down into several critical components: IP packet handling, TCP state management, sliding window protocol, socket API, and congestion control mechanisms.

IP Layer Implementation

The IP layer is responsible for the fundamental task of routing and delivering virtual IP packets between nodes in the network. Key aspects of the IP implementation include:

- **IP Packet Handling**: Robust functionalities for parsing and handling IP headers, ensuring efficient data transmission across network nodes. This involves verifying packet integrity, identifying source and destination IP addresses, and updating TTL values.
- **Routing Information Protocol (RIP)**: Integration of RIP for dynamic routing table management. RIP enables routers to exchange routing information periodically, updating forwarding tables to adapt to changes in network topology. RIP implementations adhere to RFC2453, providing a standardized approach to routing in small networks.
- **Address Resolution Protocol (ARP)**: Incorporation of ARP for dynamic mapping of IP addresses to MAC addresses within local network segments. ARP resolves IP addresses to corresponding MAC addresses, facilitating efficient communication between devices on the same subnet.
- **Traceroute**: operates by sending packets with incrementally increasing TTL values, analyzing ICMP Time Exceeded responses from intermediate routers to map network paths.

TCP Layer Implementation

Building upon the IP layer, the TCP implementation introduces mechanisms for reliable data transmission. The major components of the TCP layer are:

- **Socket API and Abstraction** A custom socket API was developed to manage multiple simultaneous connections. This API provides essential functions such as `Listen`, `Connect`, `Read`, `Write`, and `Close`, mirroring the functionality of a real-world networking API. The socket abstraction allows virtual applications to interact seamlessly with the network stack, maintaining state and context for each connection.

- **TCP State Machine** The TCP state machine manages the lifecycle of each connection, implementing connection setup (three-way handshake), data transmission, and connection termination (four-way handshake). The state machine adheres to the RFC9293 specification, ensuring robust state transitions and handling edge cases, such as incorrect ACKs during the connection setup phase.
- **Sliding Window Protocol** The sliding window protocol governs the flow of data packets between hosts, ensuring reliable and ordered data transmission. Key features of the sliding window protocol include:
  - **Data Transmission**: Packets are sent and received within the specified window, maintaining the order and integrity of data.
  - **Retransmissions**: Lost or unacknowledged packets are retransmitted based on computed round-trip times (RTT), with an adaptive retransmission timeout (RTO) to optimize performance. The protocol ensures all data reaches its destination correctly and in sequence.
  - **Event-Driven Mechanism**: The protocol is event-driven, using concurrency primitives like channels and condition variables to avoid busy-waiting and improve efficiency. This design ensures the protocol responds promptly to network events and minimizes unnecessary CPU usage.
- **Congestion Control**: As part of the capstone component, a congestion control algorithm was implemented and evaluated. The chosen algorithm, TCP Tahoe, includes:
  - **Slow Start**: Gradually increasing the transmission rate to avoid initial network congestion.
  - **Congestion Avoidance**: Adjusting the transmission rate based on network feedback to maintain optimal throughput.
  - **Fast Retransmit**: Quickly retransmitting lost packets to maintain data flow and minimize the impact of packet loss.

Conclusion

This project successfully integrates a robust TCP protocol on top of an IP layer, providing a complete virtual networking stack capable of reliable data transmission. The implementation adheres to relevant RFC standards, ensuring compliance and functionality within a simulated network environment. The addition of a socket API and congestion control mechanisms further enhances the stack's capabilities, offering a comprehensive tool for understanding and exploring TCP/IP networking concepts. The project not only demonstrates the complexity and intricacies of TCP but also provides valuable insights into the challenges of building real-world network protocols.