IP/TCP Implementation with Traceroute and TCP Reno

Edward Yan, Jason Silva, Navaiya Williams, Nick DeMarinis

Abstract

Computer networks deliver and process information according to various protocols. Two important protocols are the Internet Protocol (IP) and the Transmission Control Protocol (TCP). IP allows individual nodes to forward packets toward a specified destination node, while TCP works on top of IP to synchronize continuous data streams between two remote nodes. We implement IP and TCP on virtual nodes, and we implement traceroute, a utility that attempts to detect the path to a specific node on the network, and TCP Reno, a congestion-control algorithm that probes network conditions to avoid overwhelming the network.

1 Overview of IP/TCP

IP [6] is a protocol for routing packets of data through nodes, which are each identified by an IP address. Packets include the IP address of a destination node, and each node forwards packets to a neighbor by looking up the destination in an internal routing table.

The IP header does not contain fields to disambiguate multiple connections on one node. Instead, the data inside an IP packet will typically contain a header for a higher-level protocol, such as the User Datagram Protocol (UDP) [7] or TCP, which contain disambiguating information.

TCP synchronizes continuous data streams between two parties [3]. Each party has two buffers: one to store data received, and one to store data to send. To synchronize both buffers, we note three fields in the TCP header:

1. Sequence number: the starting index of data being sent.
2. Acknowledgment number: the first empty position in the receive buffer. This acknowledges the reception of all prior data.

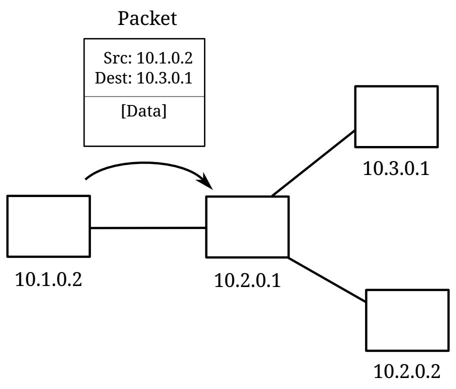


Figure 1: Multiple nodes are connected in a network. Each node has an IP address, which is used to route packets.

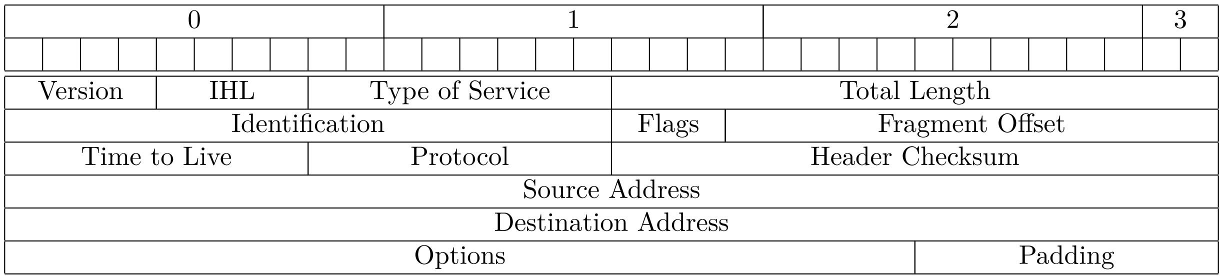


Figure 2: IPv4 header [6]. Each row represents 32 bits.

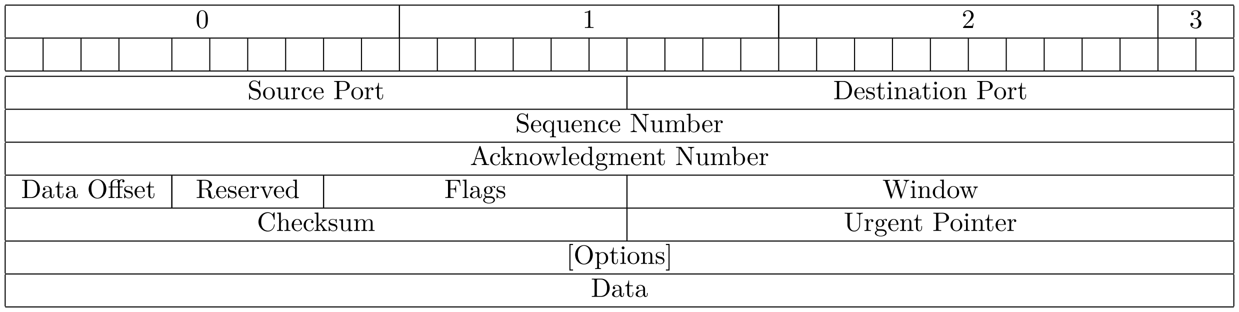


Figure 3: TCP Header [3]. Each row represents 32 bits.

3. Window: the amount of free space remaining in the receive buffer. The other party cannot send data beyond the available window.

Typically, the sender sends data in short segments, then waits for the receiver to acknowledge the reception of the data. If the sender doesn't receive an acknowledgment for some time, the sender will rettransmit the first unacknowledged segment.

2 Traceroute

Traceroute is a utility that "traces" the route to a specified destination node, which can be used to locate connection failures [2].

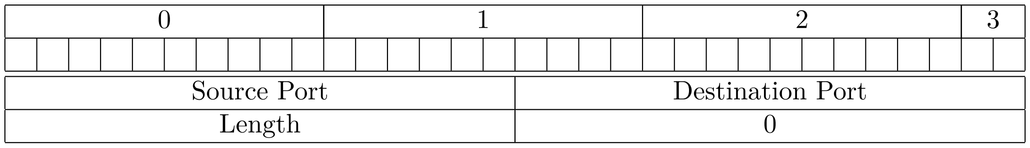
Traceroute may be implemented in many ways. Generally, the host sends probe packets with fixed time-to-live (TTL) values. Nodes along the network are meant to decrement the TTL when receiving the packet, and, if the TTL reaches 0, drop the packet and respond with a message using the Internet Control Message Protocol (ICMP). By sending a packet with a low TTL, one can identify a node at a specified distance along a path.

Our implementation of traceroute contains three parts.

1. An implementation of the ICMP messaging protocol. This allows nodes to return an error message when the TTL reaches 0.

2. An application-aware protocol on top of IP. The host uses this protocol to identify the packet that originated an ICMP error.

3. A traceroute binary. This traces the route by sending and receiving packets.



0	1	2	3

Source Port	Destination Port
Length	0

Figure 4: MUDP header. Each row represents 32 bits. The MUDP header is identical to the UDP header but zeros the checksum [7].

2.1 ICMP

ICMP is proposed in RFC 792 [5] to return informational messages to a packet's source. An ICMP packet includes a type, a subtype, the IP header of the packet that originated the message, and the first 64 bits of the IP packet's data. ICMP is designed to handle a variety of messages, but we only implemented the Destination Unreachable message and the Time Exceeded message.

We implemented ICMP as a separate module. On startup, the ICMP module registers error handlers with the core IP stack. These error handlers process internal errors by constructing and sending ICMP packets through the IP stack.

2.2 Linking ICMP messages to applications

IP is session-agnostic, so it cannot link ICMP responses to the session that sent them. To fix this, we implemented a basic session-aware layer on top of IP, consisting of InetSockets and a basic networking protocol named the Minimal User Datagram Protocol (MUDP).

InetSockets allow applications to open handles to the network, each identified by a port number. Protocols can read port numbers from packet data and deliver packets to the specified handle. In the special case that an ICMP packet is received, the ICMP module assumes that the first 32 bits contain a source and destination port, then triggers an error in the corresponding InetSocket.

MUDP is a basic session protocol for sending unordered packets. The MUDP header contains a source port, a destination port, and a length, which are used to trace errors to the originating InetSocket.

2.3 Traceroute binary

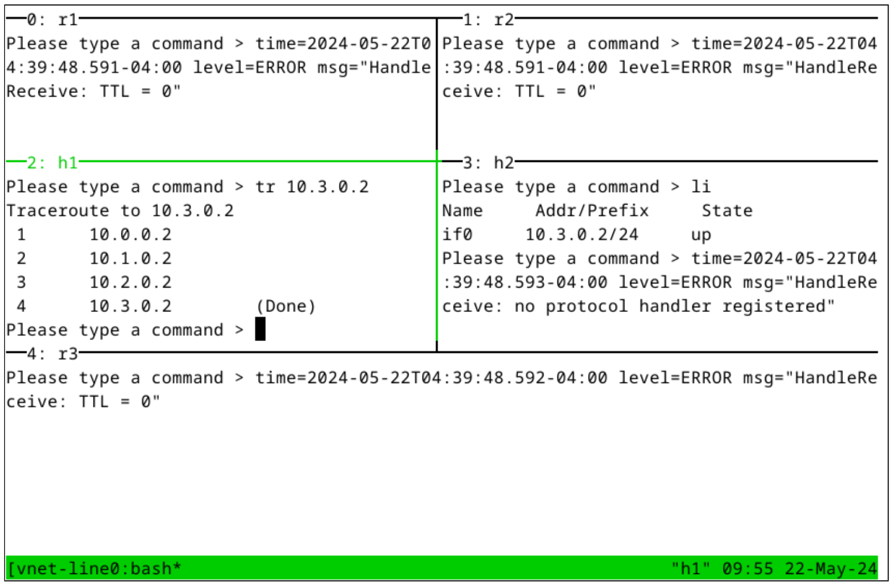
The traceroute application counts TTLs from 0 to 16, sending one MUDP packet for each TTL. Each packet will trigger either an ICMP error response or a response from the destination. An ICMP error indicates that the packet stopped at a node along the path, which is specified in the source field of the returning IP header. A response from the destination indicates that the path has ended, though an ICMP error may also indicate the same. To link returned ICMP errors to a TTL, each packet is sent along a separate socket.

Due to network errors, a response may not come. To avoid waiting forever, traceroute will wait at most 3 seconds for a response before returning an error.

3 Congestion Control: TCP Reno

Typically, a TCP sender attempts to send all segments that will fit in the receiver's window, all at once. However, this can overwhelm a network, causing intermediate nodes to drop packets that don't fit into their queue. This causes large retransmission delays, as the sender typically waits for some time before resending an unacknowledged segment.

To avoid this issue, TCP Reno [1] artificially reduces the sender's window, limiting the number of packets a node should send.



```
—0: r1———————————————————————————————————————┬—1: r2———————————————————————————————————————
Please type a command > time=2024-05-22T0│Please type a command > time=2024-05-22T04
4:39:48.591-04:00 level=ERROR msg="Handle│:39:48.591-04:00 level=ERROR msg="HandleRe
Receive: TTL = 0"                        │ceive: TTL = 0"

—2: h1———————————————————————————————————┼—3: h2———————————————————————————————————————
Please type a command > tr 10.3.0.2      │Please type a command > li
Traceroute to 10.3.0.2                   │Name     Addr/Prefix     State
 1      10.0.0.2                         │if0     10.3.0.2/24     up
 2      10.1.0.2                         │Please type a command > time=2024-05-22T04
 3      10.2.0.2                         │:39:48.593-04:00 level=ERROR msg="HandleRe
 4      10.3.0.2        (Done)           │ceive: no protocol handler registered"
Please type a command > █                │
—4: r3———————————————————————————————————┴—————————————————————————————————————————————
Please type a command > time=2024-05-22T04:39:48.592-04:00 level=ERROR msg="HandleRe
ceive: TTL = 0"






[vnet-line0:bash*                                                "h1" 09:55 22-May-24
```

Figure 5: Example usage of the traceroute command tr. Host h1 traces a route to host h2 through nodes 10.0.0.2, 10.1.0.2, and 10.2.0.2. The intermediate nodes display an error when they drop a packet with no TTL, and the destination node returns a different error when the the probe packet is not recognized.

3.1 Slow Start and Congestion Avoidance

The slow start and congestion avoidance algorithms expand the sender's window while no congestion is observed. When the window size is smaller than the slow-start threshold, slow start expands the window at the rate that data is acknowledged. When the window is larger than the slow-start threshold, congestion avoidance increments the window size by one segment whenever an entire window is acknowledged. Slow start expands the window exponentially, while congestion avoidance expands the window linearly in the absence of congestion.

If a segment is not acknowledged in time, the window size is reset to one segment, and the slow-start threshold is set to half of the current window size. This starts the slow start algorithm, exponentially expanding the window back to half of its original size.

3.2 Fast Retransmit and Fast Recovery

TCP Reno includes two more mechanisms, fast retransmit and fast recovery, which reduce the retransmission time when a segment is lost. Fast retransmission and fast recovery assume that, if a segment arrives out of order, the correct segment has been dropped from the network. When this happens, the receiver sends an immediate acknowledgment. When the sender receives three duplicate acknowledgments for the same segment, the sender retransmits the dropped segment and sets the slow-start threshold to half of the current window size. This bypasses the normal waiting period for the retransmissions and avoids further losses due to congestion.

Because acknowledgments signal that a segment was received, the sender artificially inflates their window when receiving duplicate acknowledgments. The window is restored to the slow-start threshold when a new acknowledgment is received.

3.3 Implementation

We implemented TCP Reno within our TCP stack and tested its performance by transferring *Les Misérables*, 3292 kB in size, [4]. Without congestion control, the transfer too 3500 seconds. Using TCP Reno, the transfer

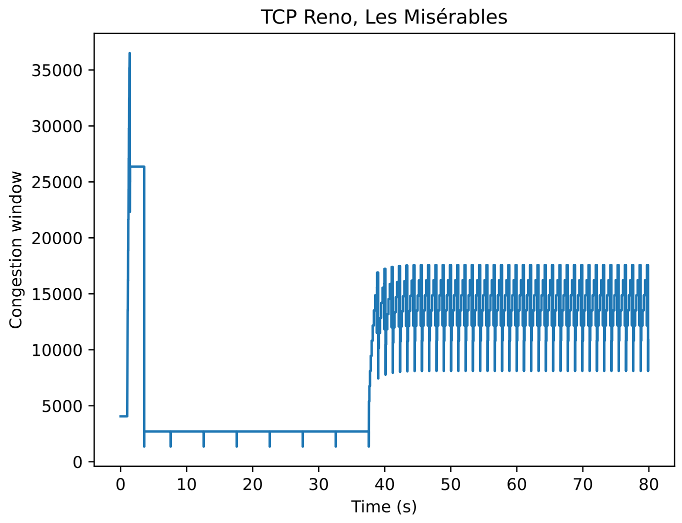
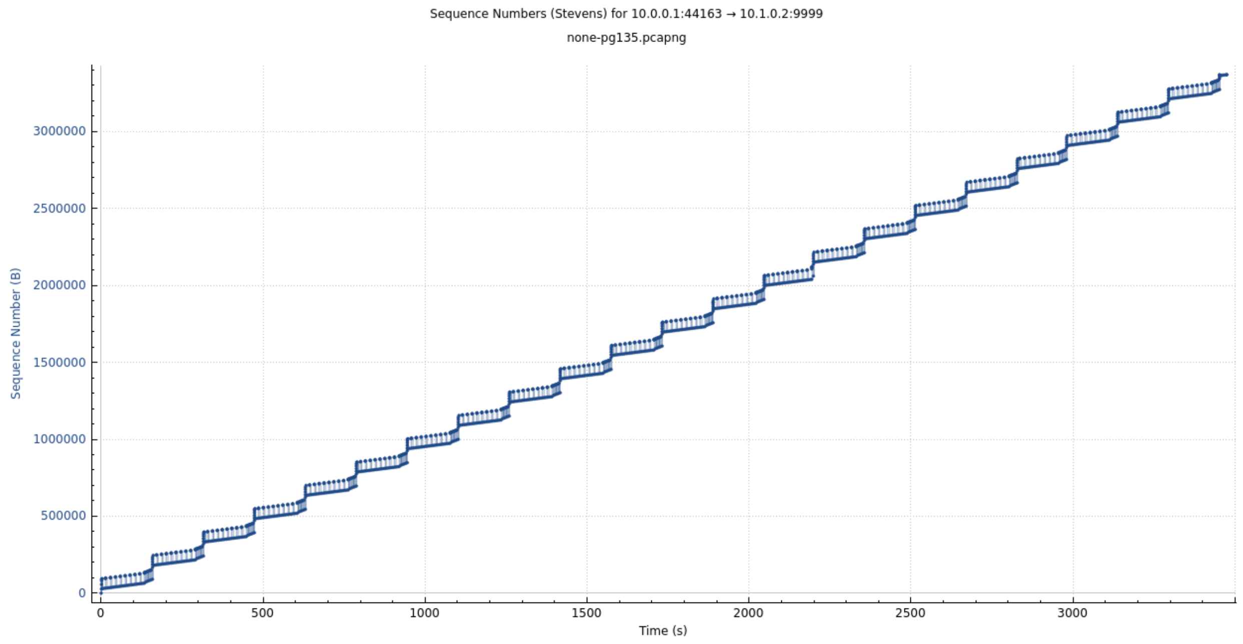


Figure 6: Window size using TCP Reno. Congestion avoidance causes a linear-increase and multiplicative-decrease pattern in the second half of transmission.

was completed in 80 seconds.



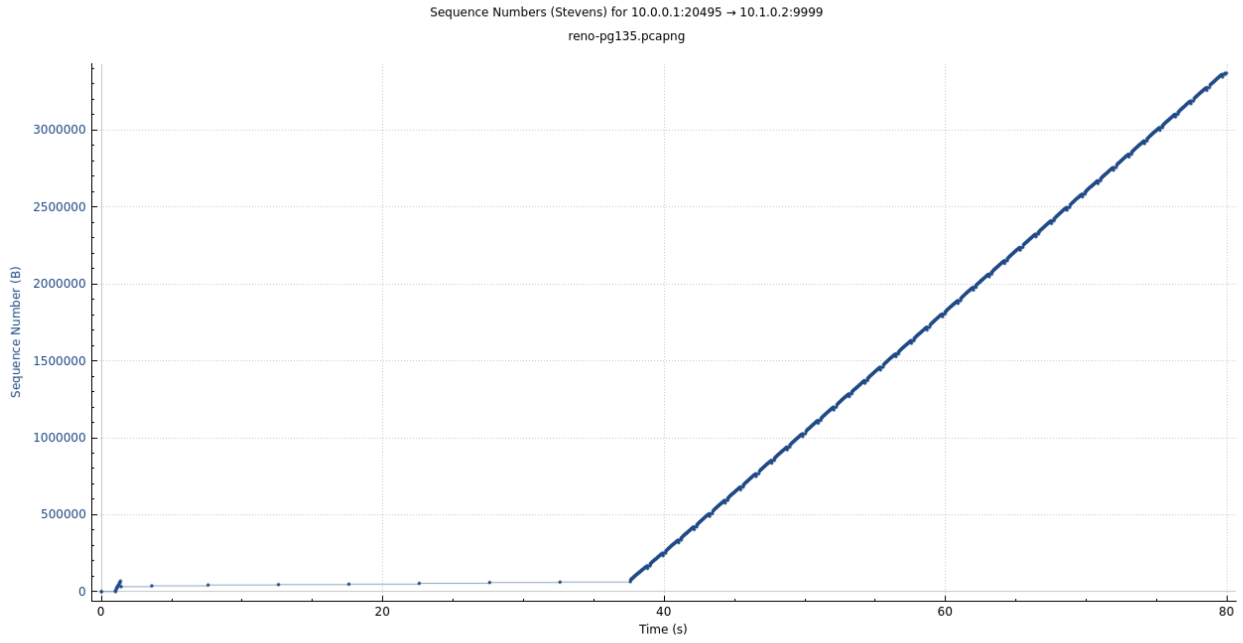


Figure 7: Top: sequence numbers without congestion control. Bottom: sequence numbers with TCP Reno. Without congestion control, data is transmitted in large bursts, which cause congestion and require slow retransmissions. With TCP Reno, there is an initial burst of packets, but slow start ensures that only the last half of data is ever lost. After retransmitting this data, the rest of the file is delivered steadily.

References

[1] Ethan Blanton, Dr. Vern Paxson, and Mark Allman. *TCP Congestion Control*. RFC 5681. Sept. 2009. DOI: `10.17487/RFC5681`. URL: `https://www.rfc-editor.org/info/rfc5681`.

[2] Dmitry Butskoy. *Traceroute*. Traceroute for Linux. Version 2.1.2. Oct. 11, 2016.

[3] Wesley Eddy. *Transmission Control Protocol (TCP)*. RFC 9293. Aug. 2022. DOI: `10.17487/RFC9293`. URL: `https://www.rfc-editor.org/info/rfc9293`.

[4] Victor Hugo. *Les Misérables*. Trans. by Isabel Florence Hapgood. Project Gutenberg, June 22, 2008. URL: `https://www.gutenberg.org/ebooks/135`.

[5] *Internet Control Message Protocol*. RFC 792. Sept. 1981. DOI: `10.17487/RFC0792`. URL: `https://www.rfc-editor.org/info/rfc792`.

[6] *Internet Protocol*. RFC 791. Sept. 1981. DOI: `10.17487/RFC0791`. URL: `https://www.rfc-editor.org/info/rfc791`.

[7] *User Datagram Protocol*. RFC 768. Aug. 1980. DOI: `10.17487/RFC0768`. URL: `https://www.rfc-editor.org/info/rfc768`.