Description of the project

Our tool offer a robust and efficient way to perform lan speed tests. The testing tools have been designed to be user-friendly, providing clear metrics and data visualization to facilitate understanding of the network's performance under a variety of conditions.

The results obtained from our testing suite have significant implications. They not only demonstrate the current state of the network's performance but also help in predicting the network's behavior under different loads. This is vital for user to make informed decisions regarding upgrades, maintenance, and configurations tailored to their specific needs.

LAN Speed Test

Throughout the course of this project, we employed the Internet Control Message Protocol (ICMP) to gauge the speed and responsiveness of Local Area Networks (LANs). This method allowed us to measure the round-trip time for messages sent between the host and a target device on the same network, which is essential for understanding the network's latency characteristics. Later, we came up with a new way by using Transmission Control Protocol (TCP). This not only help with testing ping, we can also send large packet to test the maximum bandwidth.

Contribution of each member of the team

A Runyu Yue [Student No: 1007391298]

Define the protocols headers and the check sum. Responsible for the sending logic for the arp and tcp backend(implementation of socket communication without using the "high level socket" but raw socket). Responsible for sending arp request and wrap outside headers to tcp packet and send. Also responsible for writing user interface in python and the plotting of the results.

B Kaiwei Zhang [Student No: 1007073873]

Responsible for receiving raw data packets, handling ARP reply for address mapping, sending and track ICMP packets to check reply and error reporting, developing a helper function for printing packet headers.

C Chris (Shaopeng) Lin [Student No: 1006027452]

Responsible for TCP implementation. Develop the TCP front end and used Runyu and Kawei's IP stack to implement **Stop&Wait**, **Fixed Size Sliding Window**, and **Congestion Control Sliding Window**.

How to run our code

1. set up the environment

Mininet VM from class is required.
We need the mininet environment to contain python3
We need the mininet environment to enable x-forwarding
we need the mininet environment to have numpy and matplotlib
numpy: sudo pip3 install numpy
matplotlib: sudo pip3 install matplotlib
Open the terminal of a host in our topology
run make in the src folder to compile our code
run the firsttry.mn (http://firsttry.mn) in the root directory of image or create your own
open the topology with miniedit

2. how to run the ip-ping delay detection:

Go to the src folder
run

```
python3 input.py --mode IP –-ip <IP we want to send to> –-packet_size <packet size>
```

Example command:

```
python3 input.py –mode IP –-ip 10.1.1.2 –-packet_size 100 –-interval 1 –-num_packet
```

```
--packet_size:
    Specifies the number of data bytes to be sent in a packet. The default is
50.

--interval:
    Wait interval seconds between sending each packet. The default is 2.

--num_packets:
    Specifies the number of packet to be sent. The default is 10.
```

In this process the mode, ip field is required, others has default value as described in the ip ping section.

3. how to run the tcp delay detection:

Go to the src folder
open up a port in the destination ip to receive packets using

```
nc -v -n -l <destination port number>
```

example:

```
nc -v -n -l 101
```

you can also open a iperf server using

```
iperf -s -w <window size> p <destination port>
```

example:

```
iperf -s -w 2m p 103
```

run

```
python3 input.py --mode TCP --ip <destination ip> --srcport <source port> --destpor
```

Example command:

```
python3 input.py –-mode TCP –-ip 10.1.1.2 –-srcport 1234 –-destport 101 –-variant S
```

```
--srcport
    The host machine's port we would like to act on. This is required

--destport
    The destination machine's port for us to connect to. This is required

--mss
    Ignore this option. It is there for a placeholder option we once used.

--variant               choices=['SAW', 'SWF', 'SWCC'].
    Specifies three versions of TCP testing we can do. Default 'SAW'.

--packsize
    Specifies the packet size for each segment in integer bytes. Default 1460
bytes

--period
    Specifies the testing period in seconds for the variant. Unused for 'SWF'.
Default 10s.
```

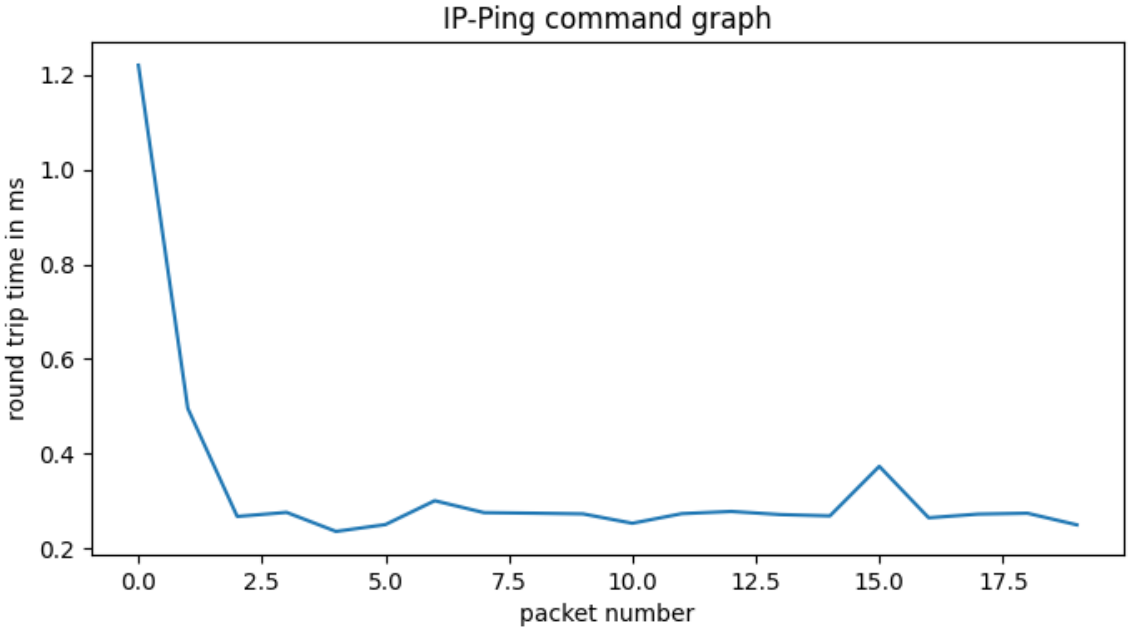
In this process the mode, ip field is required, source port is required, destination port is required, others has default value as stated in the tcp section

For more information, feel free to use the `-help` option to easily see available options.

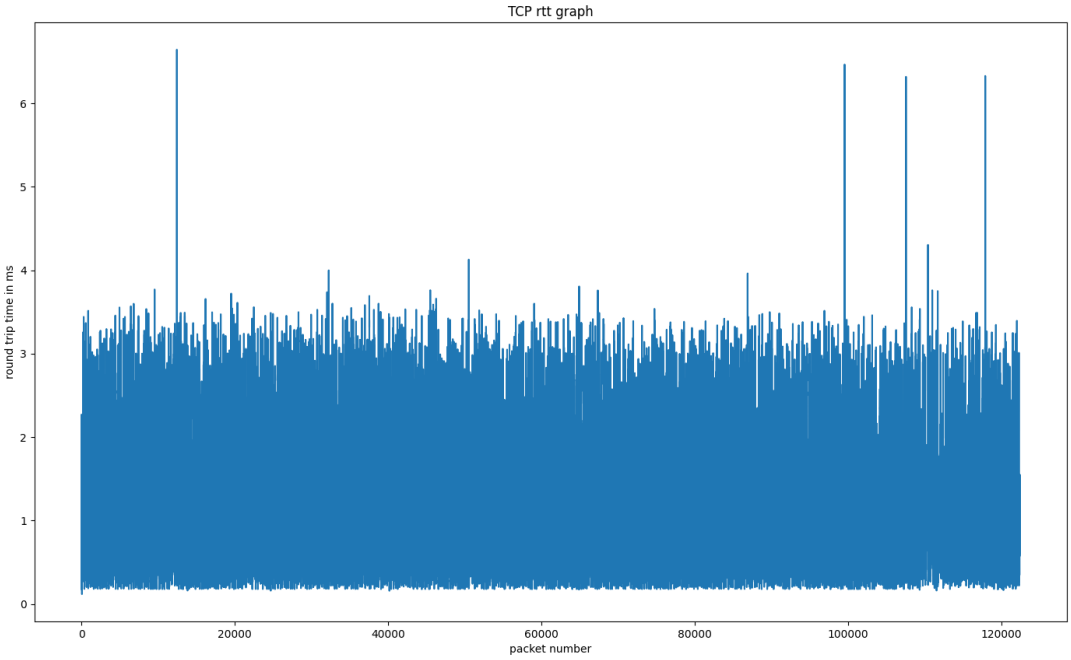
Results we will get:

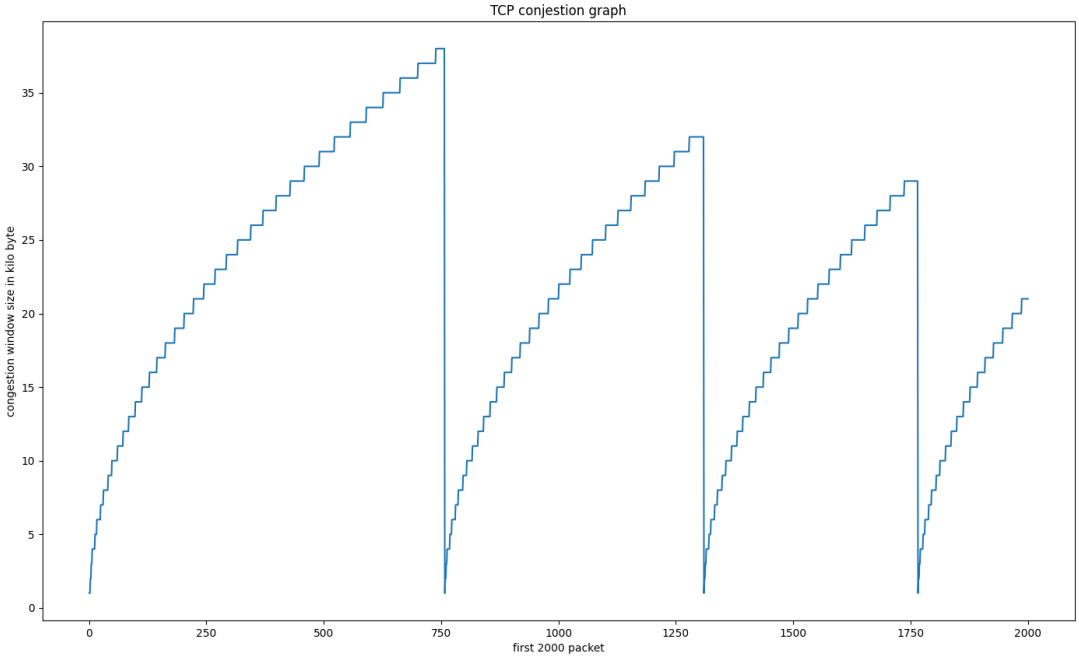
If the x forwarding is enabled, the picture will be immediately pops up containing the result of the execution

For IP mode, the result is store in result.png



For TCP mode
For SWCC mode,
the congestion window plot is in resultConjestionforTcp.png
the rtt plot is in resultRttforTcp.png
For the rest of modes,
the congestion window plot is in resultRttforTcp.png, ther is no conjestion window plot


TCP rtt graph
round trip time in ms
packet number
0
1
2
3
4
5
6
0
20000
40000
60000
80000
100000
120000


TCP conjestion graph
congestion window size in kilo byte
first 2000 packet
0
5
10
15
20
25
30
35
0
250
500
750
1000
1250
1500
1750
2000

Sample Stop&Wait:

```
***** Running Variant Stop & Wait...
***** SENT: 45.118164 KBytes, Bandwidth 369.608000 Kbits/s
***** SENT: 30.078125 KBytes, Bandwidth 246.400000 Kbits/s
***** SENT: 31.445312 KBytes, Bandwidth 257.600000 Kbits/s
***** SENT: 30.078125 KBytes, Bandwidth 246.400000 Kbits/s
***** SENT: 31.445312 KBytes, Bandwidth 257.600000 Kbits/s
***** SENT: 31.445312 KBytes, Bandwidth 257.600000 Kbits/s
***** SENT: 30.078125 KBytes, Bandwidth 246.400000 Kbits/s
***** SENT: 31.445312 KBytes, Bandwidth 257.600000 Kbits/s
***** SENT: 30.078125 KBytes, Bandwidth 246.400000 Kbits/s
***** SENT: 31.445312 KBytes, Bandwidth 257.600000 Kbits/s
***** SENT: 30.078125 KBytes, Bandwidth 246.400000 Kbits/s
***** SENT: 31.445312 KBytes, Bandwidth 257.600000 Kbits/s
*****
***** Average RTT: 41.613750 ms
***** Maximum RTT: 48.517512 ms
***** Minimum RTT: 0.003766 ms
***** Bandwidth Estimate: 262.267333 Kbits/s
***** Sliding Window Size Best Estimate Based on Bandwidth: 1364.240897 byte
*****
```

Sample Fixed Size:

```
***** INCREASING WND TO: 15400 byte
***** SENT: 4962.890625 KBytes, Bandwidth 40656.000000 Kbits/s
***** SENT: 4387.304688 KBytes, Bandwidth 35940.800000 Kbits/s
***** SENT: 4361.328125 KBytes, Bandwidth 35728.000000 Kbits/s

***** BW vs BEST Bandwidth: 37441.600000,88158.933333

***** Best Bandwidth: 88158.933333

***** RUNNING RESULT WITH WINDOW: 6
***** SENT: 26821.484375 KBytes, Bandwidth 219721.600000 Kbits/s
***** SENT: 29825.195312 KBytes, Bandwidth 244328.000000 Kbits/s
***** SENT: 28907.812500 KBytes, Bandwidth 236812.800000 Kbits/s
***** SENT: 38435.742188 KBytes, Bandwidth 314865.600000 Kbits/s
***** SENT: 30659.179688 KBytes, Bandwidth 251160.000000 Kbits/s
*****
***** Average RTT: 1.949649 ms
***** Maximum RTT: 11.437196 ms
***** Minimum RTT: 0.647240 ms
***** Bandwidth Estimate: 253377.600000 Kbits/s
***** Sliding Window Size Best Estimate Based on Bandwidth: 61749.687072 byte
*****
******************************
```

Sample Congetsion Controlled:

```
***** Running Variant Congestion Control Sliding Window...
***** SENT: 11072.852539 KBytes, Bandwidth 90708.808000 Kbits/s
***** SENT: 20408.007812 KBytes, Bandwidth 167182.400000 Kbits/s
***** SENT: 29134.765625 KBytes, Bandwidth 238672.000000 Kbits/s
***** SENT: 29364.453125 KBytes, Bandwidth 240553.600000 Kbits/s
***** SENT: 29755.468750 KBytes, Bandwidth 243756.800000 Kbits/s
***** SENT: 29971.484375 KBytes, Bandwidth 245526.400000 Kbits/s
***** SENT: 17624.414062 KBytes, Bandwidth 144379.200000 Kbits/s
***** SENT: 14056.054688 KBytes, Bandwidth 115147.200000 Kbits/s
***** SENT: 13995.898438 KBytes, Bandwidth 114654.400000 Kbits/s
***** SENT: 19183.007812 KBytes, Bandwidth 157147.200000 Kbits/s
***** SENT: 29648.828125 KBytes, Bandwidth 242883.200000 Kbits/s
***** SENT: 14277.539062 KBytes, Bandwidth 116961.600000 Kbits/s
*****
***** Average RTT: 2.029345 ms
***** Maximum RTT: 15.284267 ms
***** Minimum RTT: 0.062344 ms
***** Bandwidth Estimate: 176464.400667 Kbits/s
***** Sliding Window Size Best Estimate Based on Bandwidth: 44763.393474 byte
*****
*******************************
```

Implementation details & Documentation

RAW Socket implementation:

- int initTCPSocket ()
  This function runs on the main function which initialize the raw socket as global variable to send tcp packets
- char * find_active_interface ()
  This function returns a pointer to the string of name of the interface the code runs on
- int get_mac_ip (const char *iface, uint8_t *mac, uint32_t *ip)
  This function takes in an interface name, put the mac address of the interface to the mac pointer and put the ip of that inter face to the ip pointer
- int send_arp_packet (uint32_t targetIp)
  This function send an arp request packet to the targetIp
- int warpHeaderAndSendTcp (uint8_t *tcpbuff, int tcpTotalLen, uint32_t *dest_ip, uint8_t *dest_mac)
  This function takes in a buffer which contains tcp packet, the total length of the tcp header and data, the destination ip address, and the destination mac address. This function wraps the tcp packet with ip and ethernet header and send to the destination

Python scripts:

- input.py (http://input.py)
  This file transfers python command line argument and based on the command line argument, it creates different sub processes to run the corresponding mode and options of execution. After generation of data files by these c execution files, this file calles other python scripts which is responsible for data visulization
- plotPing.py (http://plotPing.py)
  This file plots the rtt of ip mode of the program by reading the generated data file “log”. It generates the picture directly in another window. The picture is also stored in the results.png file.
- plottcp.py (http://plottcp.py)
  This file plots the rtt and conjestion window of the tcp --variant SWCC mode of the program. It generates these 2 pictures directly in other windows by reading the “tcprtt.txt” and “tcpcong.txt”. The picture for the rtt plot is stored in resultRttforTcp.png, for the congestion window plot is stored in resultCongestionforTcp.png
- plotTcpWithoutCongestionWindow.py (http://plotTcpWithoutCongestionWindow.py)
  This file plots the rtt of the tcp mode where the variant is not SWCC. It genertes the picture resultRttforTcp.png by reading the data file named “tcprtt.txt” generated by the system

IP Implementation:

- void *packet_receiver(void *arg)
  After start running the program, main will first create a thread that keep reveiving packets.
- uint16_t send_ip_packet (struct arp_header *receive_arp_header, int size);
  This function will take received arp_header and generate a icmp echo packet to be send. If successfully sent, it returns identification of icmp packet.
- struct icmp_list
  This struct is a linked list that holds all sent icmp packets. It contains a icmp identification and time of packet being sent. After we send a icmp echo packet, we add the new packet to the begining.
- double handle_icmp(unsigned char * buffer, struct icmp_list *head);
  After we receive an icmp packet, we will call this function and it comares the receiving imcp identification with what we stored in the imcp_list. If we find corresponding record, we check the end time and return the time between.

TCP Implementation:

- `tcp_protocol.h` : **Defines the data structures, constants, and functions to help construct tcp packets**
  Notable Members:
  - `struct tcp_hdr` : defines the TCP packet header, but without options.
  - `struct tcp_pseudo_hdr` : defines the TCP pseudo-header used in checksum calculation.
  - `void tcp_gen_packet (tcp_hdr_t *header, ...)` : converts host values into the variable header, which can be passed to the network.

- `tcp_op.h` : **Defines the TCP procedures such as Handshake and Teardown**
  Notable Members:
  - `struct tcp_iq tcp_inq` : TAILQ HEAD representing the list of recieved packets
  - `struct tcp_cq tcp_ckq` : TAILQ HEAD representing the list of packets awaiting response
  - `struct tcp_sq tcp_sdq` : TAILQ HEAD representing the list of packets awaiting to be sent
  - `pthread_mutex_t inq_lock` : Syncrhonization between the ckq & inq.
  - `pthread_cond_t inq_cond;` : Used to wake up sending thread when a packet times out or new packets are arriving.
  - `void *tcp_check_timeout ()` : Timer simulation function to check timeouts or other time critical work for keeping track of statistics.
  - `void handle_tcp(...)` : Adds new packets to the tcp_inq list. Uses inq_lock.
  - `uint32_t tcp_handshake(...)` : Does three way handshake
  - `void tcp_stop_and_wait(...)` : Does Stop & Wait indefinitely until the testing period ends
  - `uint32_t tcp_send_sliding_window_fixed(...)` : Sends out fixed window size of packets until the testing period ends.
  - `void tcp_send_sliding_window_slowS_fastR (...)` : Sends out Congestion Controled window size of packets until the testing period ends.
  - `void tcp_teardown` End the connection.

- `tcp_helpers.h` : **Helps initializing Global values and defiens Miscelleneous subprocesses for TCP procedures**
  Notable Members:
  - `void initializeTCP (...)` : Initializes the required static time values such as lists, locks and user arguments to global.

- `tcp_hdr_t *tcp_wait_packet (...)` : Blocks and wait for a particular packet. Returns the matched packet.
- `void tcp_add_sw_packet (...)` : Unblocking call to add a particular packet into the checking phase.
- `handle_XX_XX_retransmit` : Handles different types of retransmission procedures.

- `tcp_stats.h` : **Defines statistical functions to record data such as RTT.**
  Notable Members:
  - `void initializeTCP (...)` : Initializes the required static time values such as lists, locks and user arguments to global.
  - `struct tcp_rtt_q tcp_rttQ` : TAILQ HEAD representing the list of RTT values for packets, arrived in the order of iteration.
  - `struct tcp_bw_q tcp_bwQ` : TAILQ HEAD representing the list of bandwidths recorded in the order of insertion.
  - `struct tcp_cong_q tcp_congQ` : TAILQ HEAD representing the list of congestion window recorded in the order of insertion.

Discussion on Project Result

Achievements

The project was able to reliabally test the speed in a network and in a low latency network, can achieve a bandwidth up to **400 MBits/s** within Ethernets(1460 bytes only compared to Iperf who sends a almost cheating amount of tens of thousands of bytes per packet). The program not only allows user to find their bandwidth, it also provides insight to optimal window sizes with respect to the bandwidth.
Moreover, testing LAN speed are able to receive the reply packet with similar result compare to real Ping test. It also create a statistic result with a graph of view.

Difficulties

1. Finding the active interface of the host and get the mac address and ip takes lots of time to investigate
2. handling of global variables is puzzling at first
3. Handling multithreading is difficult especially when the network packet also comes in asynchronously. Finding the correct way to handle in-flight packets and congestion window.

Drawbacks and Flaws

1. Time contraint forces us to discard Sequence Number Wrapping implementation and Checksum verification. The Checksum isn't too problematic as Mininet don't seem to have a way to corrupt packets? The Wrapping however, would mean if we are stuck with testing lower bandwidth.
2. The congestion control's fast retransmission is not working exactly as expected, potentially due to the network created in Mininet is much more likely to loss packet than recieve and respond.
3. Race condition sometime still persist in the TCP implementation and unfortunately it takes a great deal of work to debug them and we could not offord the time.
4. It is really hard to make the TCP client general purpose and we had to resort to coordinating with particular server implementations.
5. Retransmits indefinitly if the link is not available.