VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ISA 2024 Project - Network statistics monitoring

AUTOR PRÁCE: **DANIEL JACOBS**
AUTHOR

BRNO 2024

Table of Contents

1. Introduction
This project is a simple program to monitor network communication statistics. The goal is to create a reliable monitoring tool that is capable of handling both IPv4 and IPv6 communication and supports TCP, UDP, ICMP and ICMPv6 protocols.

2. Theoretical background
The program operates across multiple layers of the OSI (Open Systems Interconnection) model, primarily focusing on the Network Layer (Layer 3) and Transport Layer (Layer 4). I would like to describe some basic theoretical concepts used in this project.

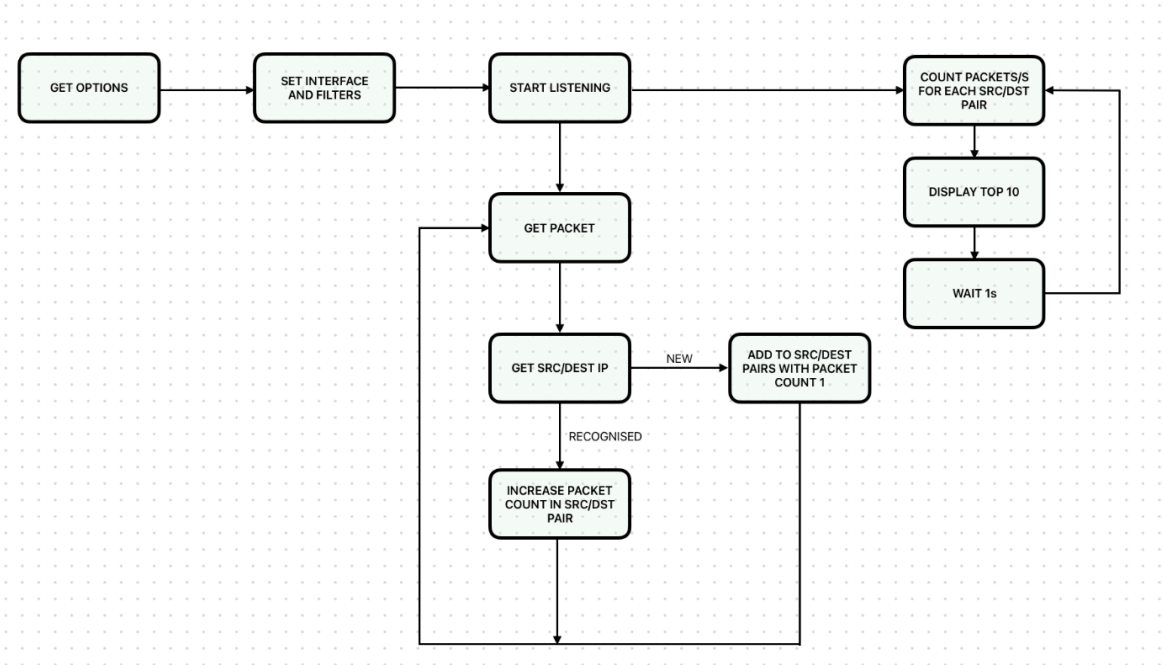
TCP (Transmission Control Protocol)
Operating at Layer 4, TCP provides connection-oriented, reliable data transmission. The program tracks TCP connections by monitoring:
• Source and destination ports
• Packet flow and byte counts

UDP (User Datagram Protocol)
Also at Layer 4, UDP offers connectionless, lightweight communication. The program monitors:
• Source and destination ports
• Packet flow and byte counts
• Broadcast and multicast traffic

ICMP (Internet Control Message Protocol)
Operating alongside IPv4 at Layer 3, ICMP handles network control messages and error reporting. Since it is a layer 3 protocol, it does not use ports, like TCP and UDP.

ICMPv6
The IPv6 equivalent of ICMP, operating at Layer 3.

3. Design
After processing user-specified options and filters, it begins capturing packets on the selected interface. Each network connection is stored in an unordered map, using source/destination IPs, ports, and protocol as keys, with transferred bytes and packets as values. For existing connections, it updates the byte and packet counts; for new connections, it creates a new entry. These statistics are then regularly processed and displayed.



3.1 PACKET CAPTURING
The packet capturing system consists of two main components. listener.cpp uses the pcap library to capture network packets and forwards them to the packet handler. handler.cpp processes these packets by extracting relevant information and storing it in an unordered map. The handler supports both IPv4 and IPv6 traffic, along with TCP, UDP, ICMP, and ICMPv6 protocols. Any other protocols are marked as “unknown”.

3.2 STATISTICS
The statistics are updated once every second. Only the top 10 connections should be displayed. The connections are ordered by their total packets or bytes transmitted in the last 5 seconds. To prevent inactive connections clogging up the display and overshadowing any active connection, a timeout is implemented. When a connection remains inactive for a set amount of time, the connection will be removed, ensuring that new and currently active connections remain visible and properly monitored.

4. Implementation
The main modules of this program are the packet listener, handler, statistics calculator and display. The file structure is as follows:

4.1 FILES
listener.cpp - handles capturing packets using the pcap library. These packets are then fed into the packet handler.

handler.cpp - extracts information from the packets and adds them to the unordered map. The handler currently supports ipv4 and ipv6. Supported protocols are tcp, udp, icmp and icmpv6. Any other protocol is displayed as unknown.

stats.cpp - periodically, once every second, processes all captured packets. The unordered map into which all captured packet are loaded is locked, a copy of all newly captured packets is created, and the lock is then lifted. Copied packets are then added to an unordered map containing all established connections. After that all expired connections are removed. The remaining connections are ordered either by total packets or total bytes transmitted, specified by the user at program launch. Inactive connections, eg. no new packets arrived for this connections for 5 seconds, are removed.

display.cpp - contains functions for formatting and displaying data fed from the statistics module.

main.cpp - initialises the display, parses program arguments, creates two threads, one for the listener and one for the statistics.

4.2 THREADS
This program works with threads. The functionality is split into two threads: listener thread and statistics thread. This is to ensure that operations responsible for manipulating and calculating data for the network statistics output minimally interfere with the packet capturing process.

4.3 ERROR HANDLING
Errors and fault states are handled with try-catch blocks and throwing runtime errors. This choice was made for the ease of propagating error states from the thread to the main function, which then can properly clean up and shut down gracefully.

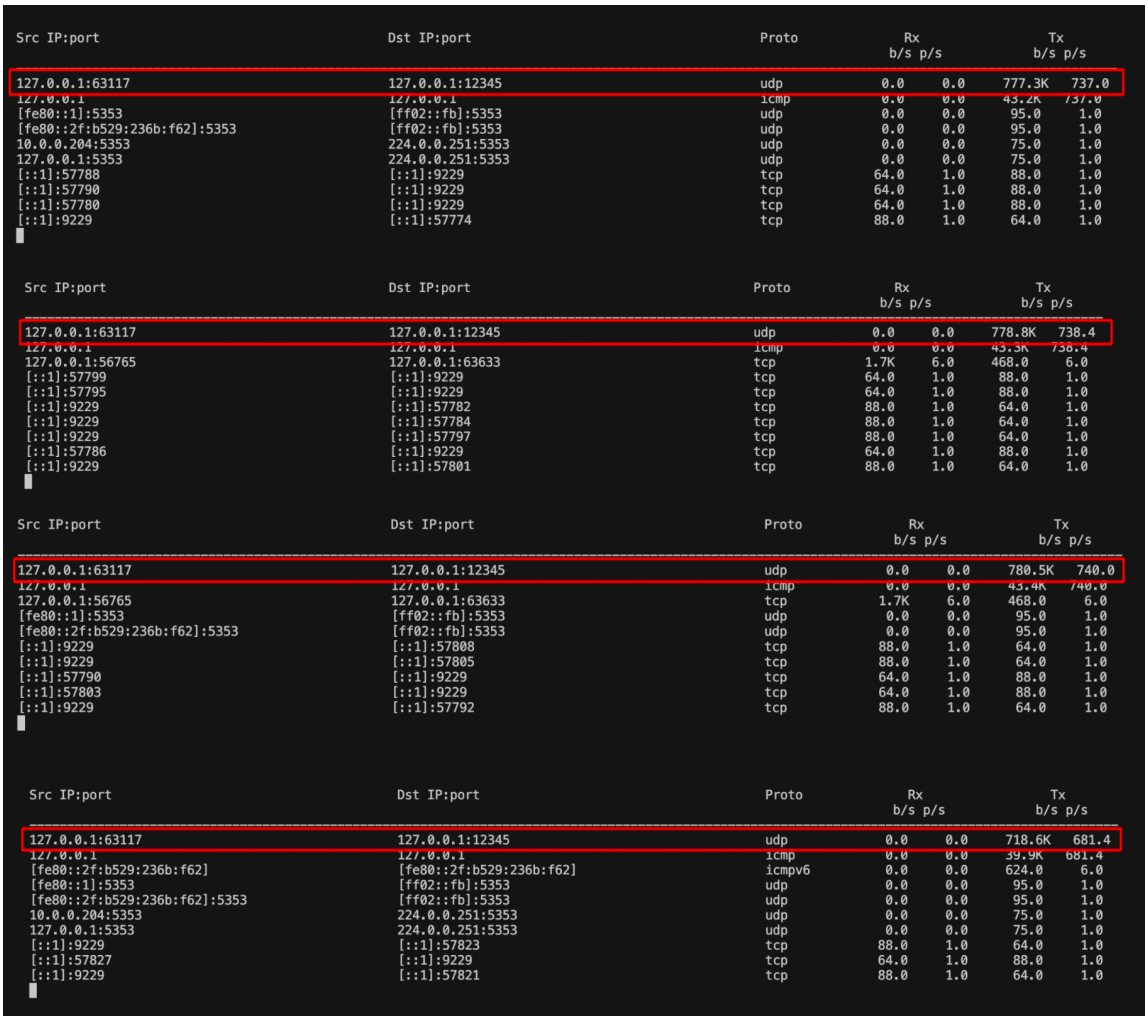
5. Testing

5.1 TESTING PACKET AND BYTE RATES

Testing of this programs packet and byte rates per second was done using a python script that sent roughly 10 000 UDP packets over a 10 second period at a target rate of 1 MB/S. Due to performance limitations, the actual rate was usually about 0.75 MB/S. During the test, the state of the network statistic program output was captured four times, to get an approximation of the average packet and byte count per second, because the actual captured rates will be slightly different in every packet processing cycle.

Testing conditions: Test interface: lo0 (MacOS loopback interface)
Test host: 127.0.0.1 (localhost)
Test port: 12345

The four captured states are shown in the image below:



Src IP:port	Dst IP:port	Proto	Rx b/s	Rx p/s	Tx b/s	Tx p/s
127.0.0.1:63117	127.0.0.1:12345	udp	0.0	0.0	777.3K	737.0
127.0.0.1	127.0.0.1	icmp	0.0	0.0	43.2K	737.0
[fe80::1]:5353	[ff02::fb]:5353	udp	0.0	0.0	95.0	1.0
[fe80::2f:b529:236b:f62]:5353	[ff02::fb]:5353	udp	0.0	0.0	95.0	1.0
10.0.0.204:5353	224.0.0.251:5353	udp	0.0	0.0	75.0	1.0
127.0.0.1:5353	224.0.0.251:5353	udp	0.0	0.0	75.0	1.0
[::1]:57788	[::1]:9229	tcp	64.0	1.0	88.0	1.0
[::1]:57790	[::1]:9229	tcp	64.0	1.0	88.0	1.0
[::1]:57780	[::1]:9229	tcp	64.0	1.0	88.0	1.0
[::1]:9229	[::1]:57774	tcp	88.0	1.0	64.0	1.0

Src IP:port	Dst IP:port	Proto	Rx b/s	Rx p/s	Tx b/s	Tx p/s
127.0.0.1:63117	127.0.0.1:12345	udp	0.0	0.0	778.8K	738.4
127.0.0.1	127.0.0.1	icmp	0.0	0.0	43.3K	738.4
127.0.0.1:56765	127.0.0.1:63633	tcp	1.7K	6.0	468.0	6.0
[::1]:57799	[::1]:9229	tcp	64.0	1.0	88.0	1.0
[::1]:57795	[::1]:9229	tcp	64.0	1.0	88.0	1.0
[::1]:9229	[::1]:57782	tcp	88.0	1.0	64.0	1.0
[::1]:9229	[::1]:57784	tcp	88.0	1.0	64.0	1.0
[::1]:9229	[::1]:57797	tcp	88.0	1.0	64.0	1.0
[::1]:57786	[::1]:9229	tcp	64.0	1.0	88.0	1.0
[::1]:9229	[::1]:57801	tcp	88.0	1.0	64.0	1.0

Src IP:port	Dst IP:port	Proto	Rx b/s	Rx p/s	Tx b/s	Tx p/s
127.0.0.1:63117	127.0.0.1:12345	udp	0.0	0.0	780.5K	740.0
127.0.0.1	127.0.0.1	icmp	0.0	0.0	43.4K	740.0
127.0.0.1:56765	127.0.0.1:63633	tcp	1.7K	6.0	468.0	6.0
[fe80::1]:5353	[ff02::fb]:5353	udp	0.0	0.0	95.0	1.0
[fe80::2f:b529:236b:f62]:5353	[ff02::fb]:5353	udp	0.0	0.0	95.0	1.0
[::1]:9229	[::1]:57808	tcp	88.0	1.0	64.0	1.0
[::1]:9229	[::1]:57805	tcp	88.0	1.0	64.0	1.0
[::1]:57790	[::1]:9229	tcp	64.0	1.0	88.0	1.0
[::1]:57803	[::1]:9229	tcp	64.0	1.0	88.0	1.0
[::1]:9229	[::1]:57792	tcp	88.0	1.0	64.0	1.0

Src IP:port	Dst IP:port	Proto	Rx b/s	Rx p/s	Tx b/s	Tx p/s
127.0.0.1:63117	127.0.0.1:12345	udp	0.0	0.0	718.6K	681.4
127.0.0.1	127.0.0.1	icmp	0.0	0.0	39.9K	681.4
[fe80::2f:b529:236b:f62]	[fe80::2f:b529:236b:f62]	icmpv6	0.0	0.0	624.0	6.0
[fe80::1]:5353	[ff02::fb]:5353	udp	0.0	0.0	95.0	1.0
[fe80::2f:b529:236b:f62]:5353	[ff02::fb]:5353	udp	0.0	0.0	95.0	1.0
10.0.0.204:5353	224.0.0.251:5353	udp	0.0	0.0	75.0	1.0
127.0.0.1:5353	224.0.0.251:5353	udp	0.0	0.0	75.0	1.0
[::1]:9229	[::1]:57823	tcp	88.0	1.0	64.0	1.0
[::1]:57827	[::1]:9229	tcp	64.0	1.0	88.0	1.0
[::1]:9229	[::1]:57821	tcp	88.0	1.0	64.0	1.0

Next step is to compare this output with the rate the python script claims it sends packets:

```
Starting packet generation on loopback interface (lo0)
Interface details:
- IP: 127.0.0.1
- UDP Port: 12345

Sending 10.0 MB over 10 seconds...
Packet size: 1048 bytes
Rate: 1.0 MB/s

Finished sending packets:
Total bytes sent: 7.5 MB
Actual duration: 10.0 seconds
Actual rate: 0.7 MB/s
Socket closed
```

Lastly, compare this information with analysis tools like Wireshark conversations:

Address A	Port A	Address B	Port B	Packets	Bytes	Stream ID	Total Packets	Percent Filtered	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration
127.0.0.1	63117	127.0.0.1	12345	7 495	8 MB	10	7 495	100.00%	7 495	8 MB	0	0 bytes	106.177421	9.9996

and Wireshark packet lengths:

Topic / Item	Count	Average	Min Val	Max Val	Rate (ms)	Percent	Burst Rate	Burst Start
Packet Lengths	19188	460.76	44	1380	0.0456	100%	1.6400	106.285
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	10387	59.93	44	75	0.0247	54.13%	0.8400	106.250
80-159	1300	91.86	88	108	0.0031	6.78%	0.0800	169.535
160-319	0	-	-	-	0.0000	0.00%	-	-
320-639	4	462.50	349	576	0.0000	0.02%	0.0400	175.553
640-1279	7495	1080.00	1080	1080	0.0178	39.06%	0.7900	107.210
1280-2559	2	1380.00	1380	1380	0.0000	0.01%	0.0100	111.164
2560-5119	0	-	-	-	0.0000	0.00%	-	-
5120 and greater	0	-	-	-	0.0000	0.00%	-	-

5.2 TESTING PROGRAM ARGUMENTS

Testing of valid and invalid program arguments is simple. The following cases have been tested:
Launching the program with an interface that does not exist:

```
[redacted] ISA % ./isa-top -i invalid
Listener thread exception caught
Error: Failed to activate: No such device exists
```

Launching the program with an interface that does exist, but the user does not have sufficient permissions to use it:

```
xjacob00@merlin: ~/ISA/netstat$ ./exec -i eth0
eth0
    Listener exception caught
                        Cleaning up
Error: Invalid network interface: eth0: You don't have permission to perform this capture on that device (socket: Operation not permitted)
```

Launching the program with an invalid order:

```
[redacted] ISA % ./isa-top -i en0 -s k
invalid order, use b (bytes/s) or p (packets/s)
```

5.3 TESTING RESULTS

The test results show that our program's measurements closely match both the transmitted data and independent Wireshark analysis. Minor variations in measurements are expected due to the real-time nature of network monitoring and the sampling intervals used for statistics collection.

6. How to use

6.1 COMPILING AND REMOVING THE PROGRAM

Compile the program using the *make* command.
You can remove the compiled executable using the *make clean* command.

6.2 RUNNING THE PROGRAM

You can run the program using the following command:

./isa-top -i <interface> -s <b|p>

where interface is a mandatory argument which specifies what interface the program is going to monitor, and an option -s with the options either b (sort by bytes transmitted, in descending order) or p (sort by packets, in descending order).

For example:

./isa-top -i en0 -s p

This will capture packets on the interface en0 and sort them by packets captured.

7. Conclusion

The implementation meets its primary objectives of supporting the required protocols (TCP, UDP, ICMP, ICMPv6), providing real-time statistics monitoring and handling both IPv4 and IPv6 traffic.

Bibliography

1. “C++ Reference: std::unordered_map.” cppreference.com. Retrieved from https://en.cppreference.com/w/cpp/container/unordered_map
2. “C++ Reference: std::lock_guard.” cppreference.com. Retrieved from https://en.cppreference.com/w/cpp/thread/lock_guard
3. “C++ Reference: std::this_thread::sleep_for.” cppreference.com. Retrieved from https://en.cppreference.com/w/cpp/thread/sleep_for
4. “Programming with pcap.” tcpdump.org. Retrieved from https://www.tcpdump.org/pcap.html
5. “TCP (Transmission Control Protocol) - What is it, and how does it work?” ClouDNS Blog. Retrieved from https://www.cloudns.net/blog/tcp-transmission-control-protocol-what-is-it-and-how-does-it-work/
6. “UDP (User Datagram Protocol) explained in details.” ClouDNS Blog. Retrieved from https://www.cloudns.net/blog/udp-user-datagram-protocol-explained-in-details/
7. “ICMPv6.” Network Insight. Retrieved from https://network-insight.net/2015/04/08/icmpv6/
8. “Using Insertion Operators and Controlling Format.” Microsoft Documentation. Retrieved from https://learn.microsoft.com/en-us/cpp/standard-library/using-insertion-operators-and-controlling-format
9. “LINK-LAYER HEADER TYPES.” tcpdump.org. Retrieved from https://www.tcpdump.org/linktypes.html
10. “unordered_map find in C++ STL.” GeeksforGeeks. Retrieved from https://www.geeksforgeeks.org/unordered_map-find-in-c-stl/