

# PROTOCOL ANALYZER USER MANUAL

C++ WIN32 asynchronous TCP and UDP connections

*Purpose,  
Description, Design  
and Testing  
information for a  
TCP and UDP  
protocol analyzer  
tool.*

## Contents

Purpose .....	2
Usage.....	3
Design.....	4
Design Choice Notes .....	4
State Diagrams .....	4
WinMain.....	4
WndProc - Start.....	5
WndProc – UI Events.....	5
WndProc – Socket Events .....	6
Pseudocode.....	6
WinMain.....	6
WndProc.....	6
Testing Document.....	10
Summary .....	10
Test 1) Program Runs Without Crashing.....	12
Test 2) UI is functional .....	13
Test 3) Client/Server Mode Changes Modifies UI.....	14
Test 4) Sending a single TCP packet .....	15
Test 5) Sending multiple TCP packets .....	16
Test 6) Sending 100 packets size 1024 in TCP.....	17
Test 7) Large 32768 byte packets can be sent via TCP .....	18
Test 8) Sending a single UDP packet .....	19
Test 9) Sending multiple UDP packets .....	20
Test 10) Sending 100 packets size 1024 in UDP .....	21
Test 11) Large 32768 byte packets can be sent via UDP .....	22
Test 12) TCP can send 500 different packets size 8192 .....	23
Test 13) UDP can send 500 different packets size 8192 .....	23

## Purpose

To test and compare the difference in performance between TCP and UDP protocols asynchronously utilizing WIN32's Winsock and WSAAsyncSelect.

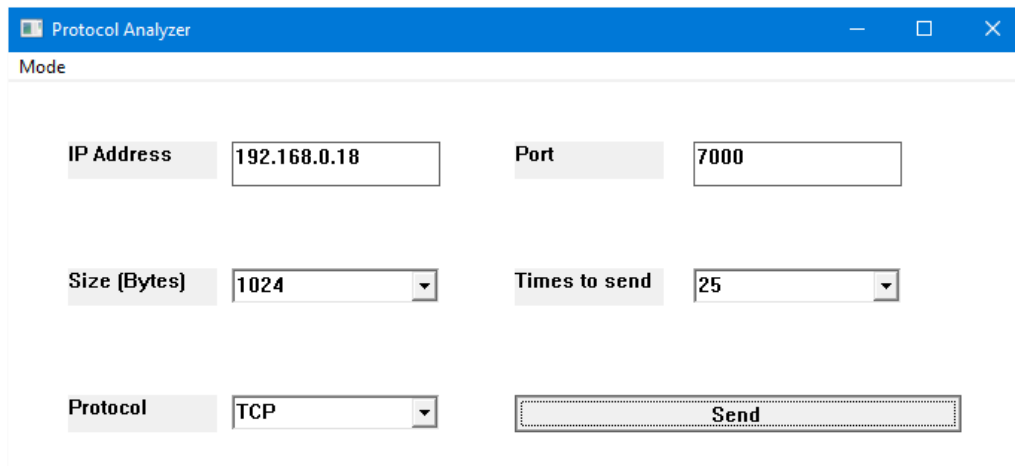
## Usage

Run the program by double clicking on the exe file.

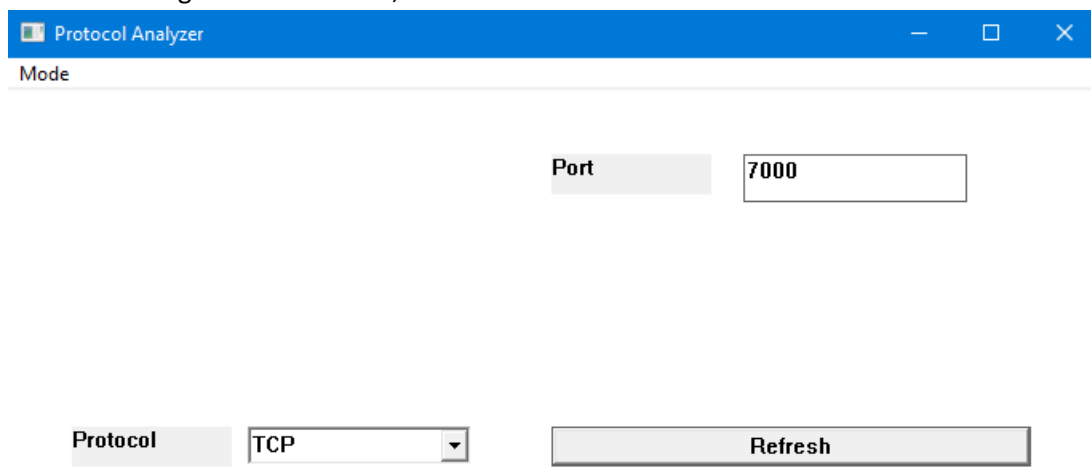
By default, the application runs in client mode. To switch to server mode, click on the word “Mode” at the top left and change to Server. Follow the same process to switch back to Client when needed.



As a client, specify the type of data to be generated and the location (IP and port) to send it to. The specific types of data include using the TCP or UDP, specifying the packet size and the amount of packets to send. Once you have entered the correct information, click the Send button.



As a server, select the port you wish to connect to and for either TCP or UDP connections. Once you make a change to these values, click the Refresh button to restart the server connection.



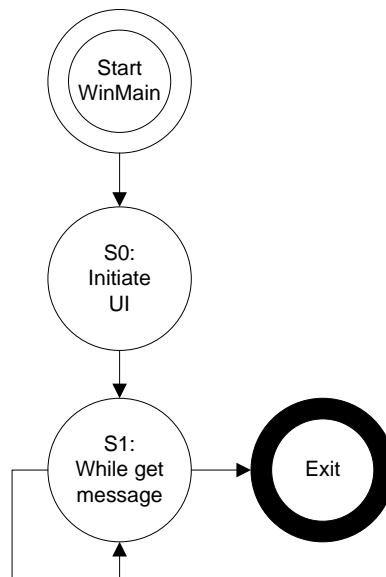
## Design

### Design Choice Notes

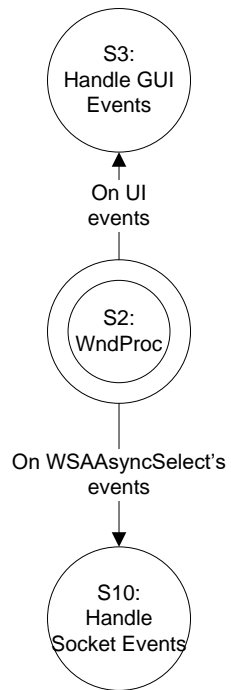
- The program is to be written in C++ using Win32 API
- Both TCP and UDP will be asynchronous
- They will both utilize Winsock2.h and WSAAsyncSelect calls to accomplish the above

### State Diagrams

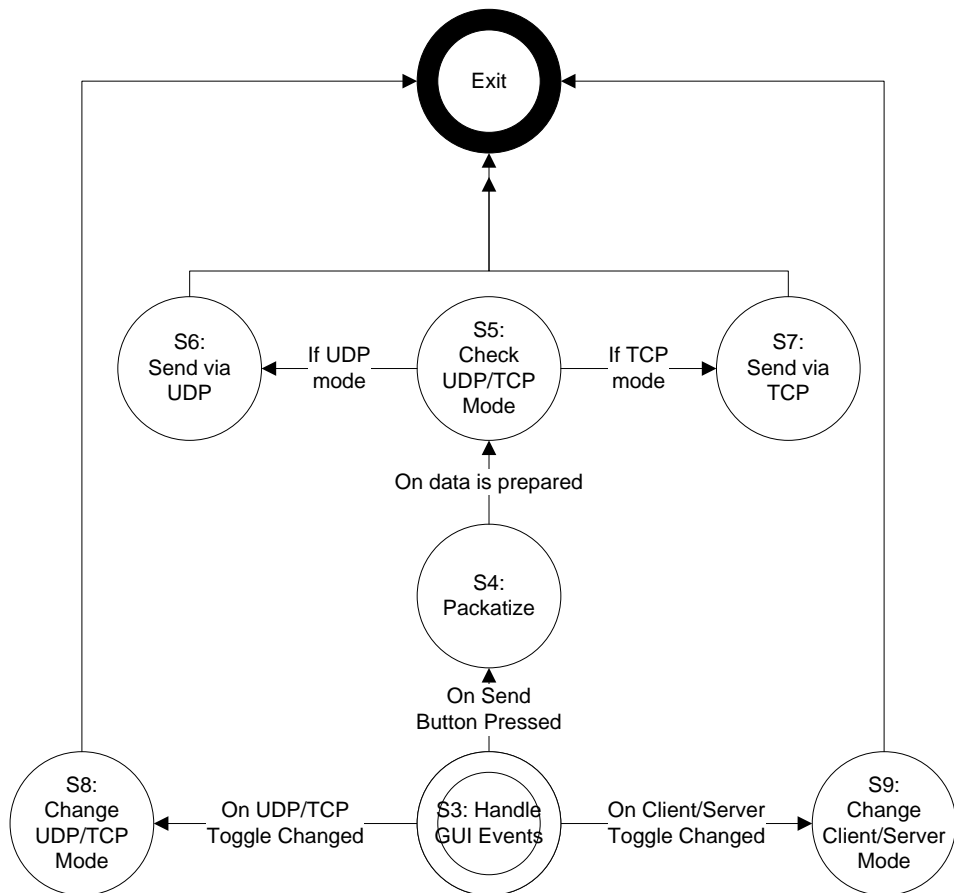
#### WinMain



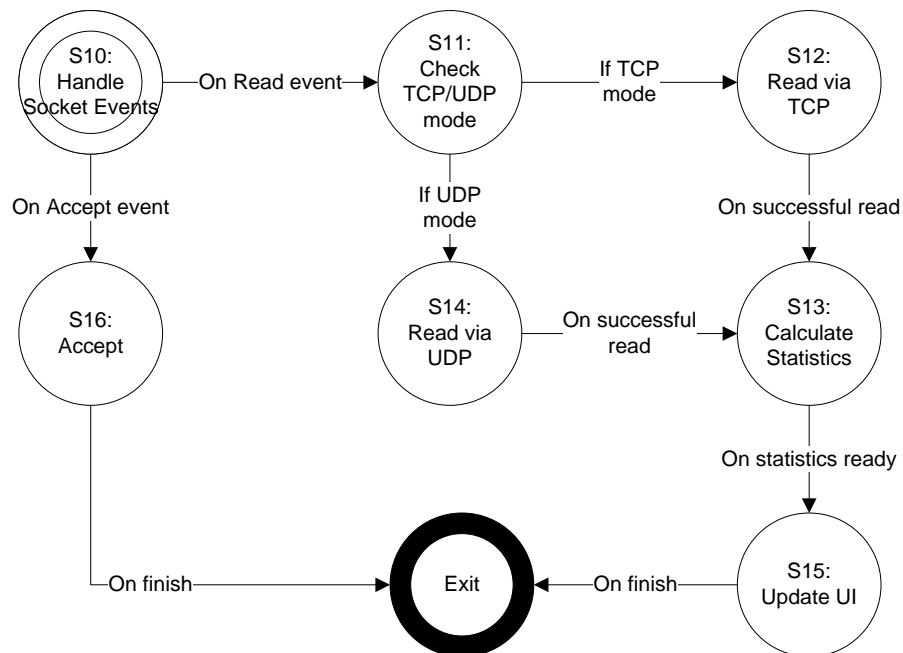
## WndProc - Start



## WndProc – UI Events



## WndProc – Socket Events



## Pseudocode

### WinMain

#### *S0: Instantiate UI*

Create Window and store its handle  
Get handles for all UI elements on Window  
Set default data for all UI elements  
GOTO S1:

#### *S1: While get message*

```
//Mandatory WinMain loop
while GetMessage() is true
    TranslateMessage()
    DispatchMessage()
Exit()
```

### WndProc

#### *S2: WndProc*

```
switch case event message
    case any UI event
        GOTO S3
    case WSAAsyncSelect's socket events
        GOTO S10
```

### ***S3: Handle GUI Events***

```
switch case event message
    case UDP/TCP toggle changed
        GOTO S8
    case Send button pressed
        GOTO S4
    case Client/Server toggle changed
        GOTO S9
```

### ***S4: Packatize***

```
Check if GUI is set to read from file
if yes
    packatize packets based on GUI settings from file
else
    packatize packets based on randomly generated data
GOTO S5
```

### ***S5: Check UDP/TCP Mode***

```
if mode == UDP
    GOTO S6
else
    GOTO S7
```

### ***S6: Send via UDP***

```
Call WSASStartup to setup the enviroment
Call WSASocket to create a datagram socket
foreach packet created
    WSASendTo with socket
Exit()
```

### ***S7: Send via TCP***

```
Call WSASStartup to setup the enviroment
Call WSASocket to create a stream socket
foreach packet created
    WSASend with socket
Exit()
```

### ***S8: Change UDP/TCP Mode***

```
if wndproc event invoked was to UDP
    ProtocolMode = UDP
else
    ProtocolMode = TCP
if ClientServerMode == Server
```



```

        if ProtocolMode == UDP
            WSAAsyncSelect to invoke WndProc under socket needing to be read
        else
            WSAAsyncSelect to invoke WndProc under socket needing to be read or
accepted
    Exit()

```

### ***S9: Change Client/Server Mode***

```

    if wndproc event invoked was Client
        Update UI for client-related elements to be displayed
        Close any open server sockets
    else
        Update UI for server-related elements to be displayed

```

### ***S10: Handle Socket Events***

```

    switch case socket event messages
        case Read
            GOTO S11
        case Accepting
            GOTO S16

```

### ***S11: Check TCP/UDP Mode***

```

    if ProtocolMode == TCP
        GOTO S12
    else
        GOTO S14

```

### ***S12: Read via TCP***

```

    start timer
    while there is more data on the socket
        read data
    end timer
    GOTO S13

```

### ***S13: Calculate Statistics***

```

    User timer from S12/S14 & data read to determine statistics
    Such as reading efficiency for each packet size & amount that came in.
    GOTO S15

```

### ***S14: Read via UDP***

```

    start timer
    while there is incoming data on the socket
        read data

```

end timer  
GOTO S13

### *S15: Update GUI*

Draw statistics onto GUI  
Exit()

# Testing Document

## Summary

Screenshots and more information on the tests can be found below.

Section #	Description	Test	Expected Output	Success
1	Program runs without crashing	Run the program	The program does not crash upon starting	Passed
2	UI are functional	Repeat Test #1 and type in a field or click a drop down combo box.	The UI is responsive	Passed
3	Client/Server mode switching changes the UI	Repeat Test #1 and click on the Mode button in the menu. Switch back and forth between Server and Client	The UI will update itself to signify the change in mode. Non-relevant fields in server side will be removed.	Passed
4	A packet can be sent between a client and server instance of this application via TCP	Run two instances of the program on separate computers.	The server UI will state a packet was received	Passed
5	Multiple packets can be sent between a client and a server instance of this application via TCP	Repeat Test #4, however raise the amount of packets to be sent to 10.	The server UI will state data was read equivalent to ten times the packet size.	Passed
6	100 packets can be sent between a client and a server at 1024 byte packets	Repeat Test #5, however raise the amount of packets to 10 and specify a size of 1024	The server UI will state 102400 bytes were read.	Passed
7	Large packets can be sent via TCP. 32768 bytes.	Repeat Test #4, however raise the size of the packet to 32768 bytes.	The server UI will state 32768 bytes were read.	Passed
8	A packet can be sent between a client and server instance of this application via UDP	Run two instances of the program on separate computers.	The server UI will state a packet was received	Passed
9	Multiple packets can be sent between a client and a server instance of this application via UDP	Repeat Test #8, however raise the amount of packets to be sent to 10.	The server UI will state data was read equivalent to ten times the packet size.	Passed
10	100 packets can be sent between a client and a server at 1024	Repeat Test #8, however raise the amount of packets to	The server UI will state 102400 bytes were read.	Passed

	byte packets	10 and specify a size of 1024		
<b>11</b>	Large packets can be sent via UDP. 32768 bytes.	Repeat Test #8, however raise the size of the packet to 32768 bytes.	The server UI will state 32768 bytes were read.	Passed
<b>12</b>	TCP can send 500 packets all size 8192	Repeat Test #5, but make it send 500 packets of size 8192	All data should be sent and the UI display that the data is received.	Passed

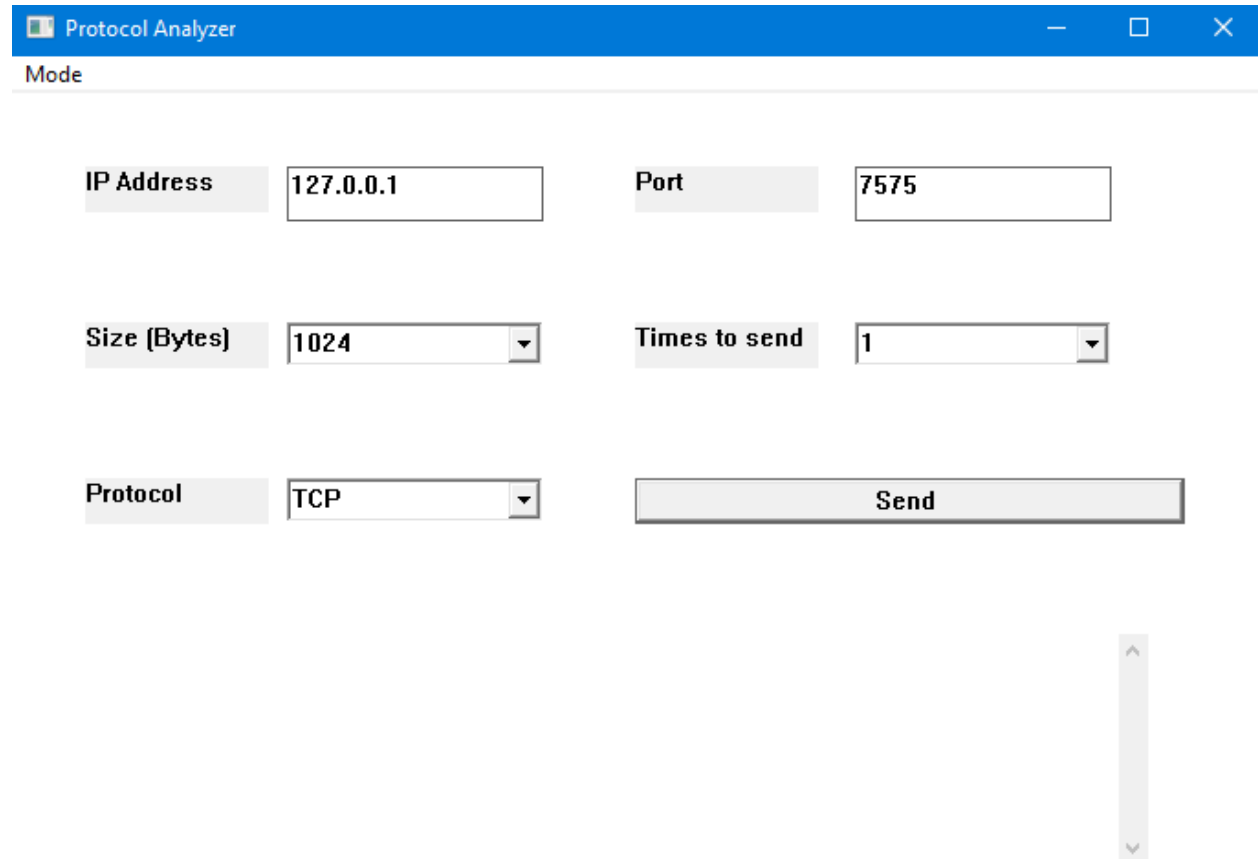
## Test 1) Program Runs Without Crashing

**Description:** The program runs without crashing.

**Result:** The program will display the UI.

**Result:** Passed

*Figure 1) UI loaded, showing the program does not crash.*



The screenshot shows a window titled "Protocol Analyzer" with a blue header bar. Below the header, the word "Mode" is displayed. The main area contains several input fields and a button:

- IP Address:** A text box containing "127.0.0.1".
- Port:** A text box containing "7575".
- Size (Bytes):** A dropdown menu showing "1024".
- Times to send:** A dropdown menu showing "1".
- Protocol:** A dropdown menu showing "TCP".
- Send:** A large button to the right of the Protocol dropdown.

A vertical scrollbar is visible on the right side of the window.

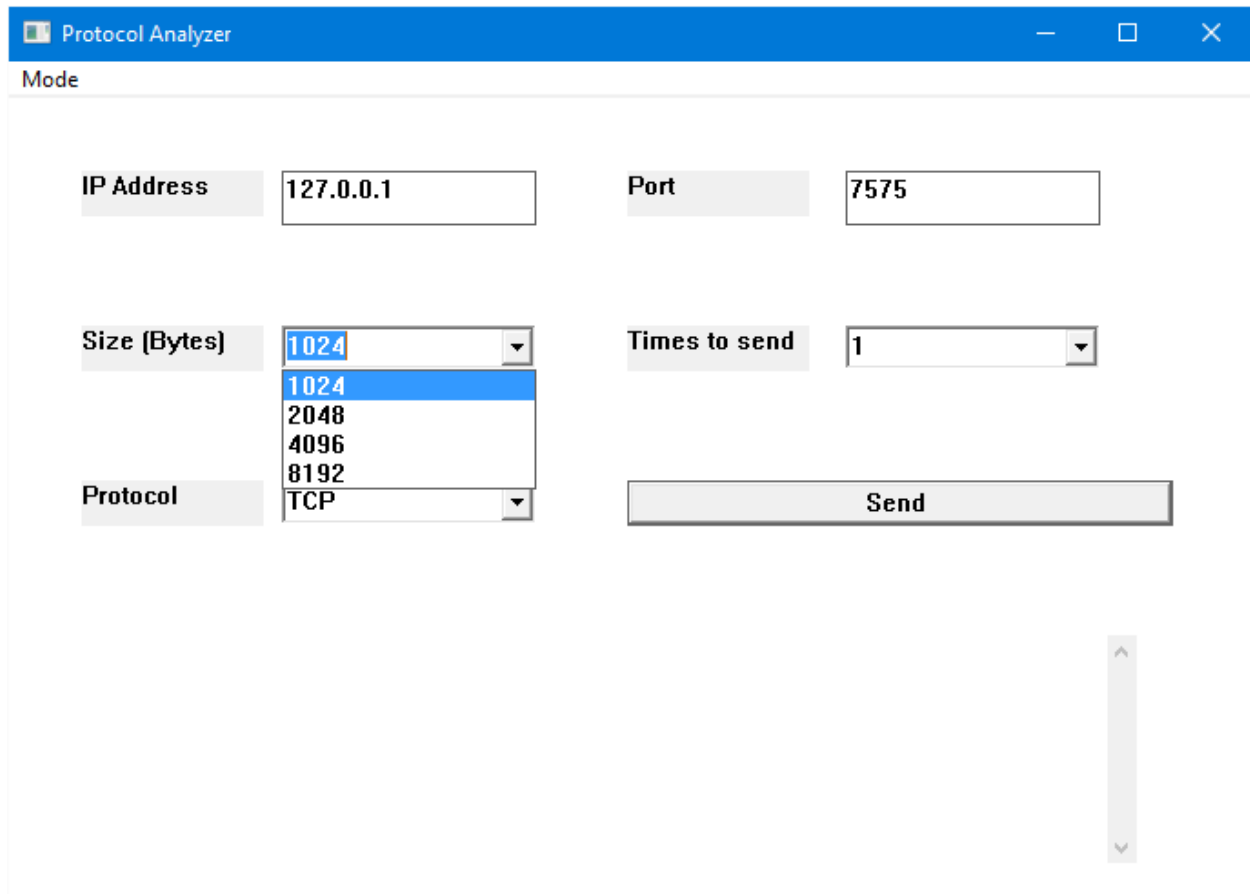
## Test 2) UI is functional

**Description:** The UI is fully functional and can be edited without error.

**Result:** The UI is functional

**Result:** Passed

*Figure 2) UI loaded, showing the program does not crash.*



The screenshot shows a window titled "Protocol Analyzer" with standard Windows window controls (minimize, maximize, close). Below the title bar is a tab labeled "Mode". The main area contains several input fields and a button:

- IP Address:** A text box containing "127.0.0.1".
- Port:** A text box containing "7575".
- Size (Bytes):** A dropdown menu with "1024" selected. The dropdown list is open, showing options: 1024, 2048, 4096, and 8192.
- Times to send:** A dropdown menu with "1" selected.
- Protocol:** A dropdown menu with "TCP" selected.
- Send:** A large button to the right of the dropdowns.

A vertical scrollbar is visible on the right side of the window.

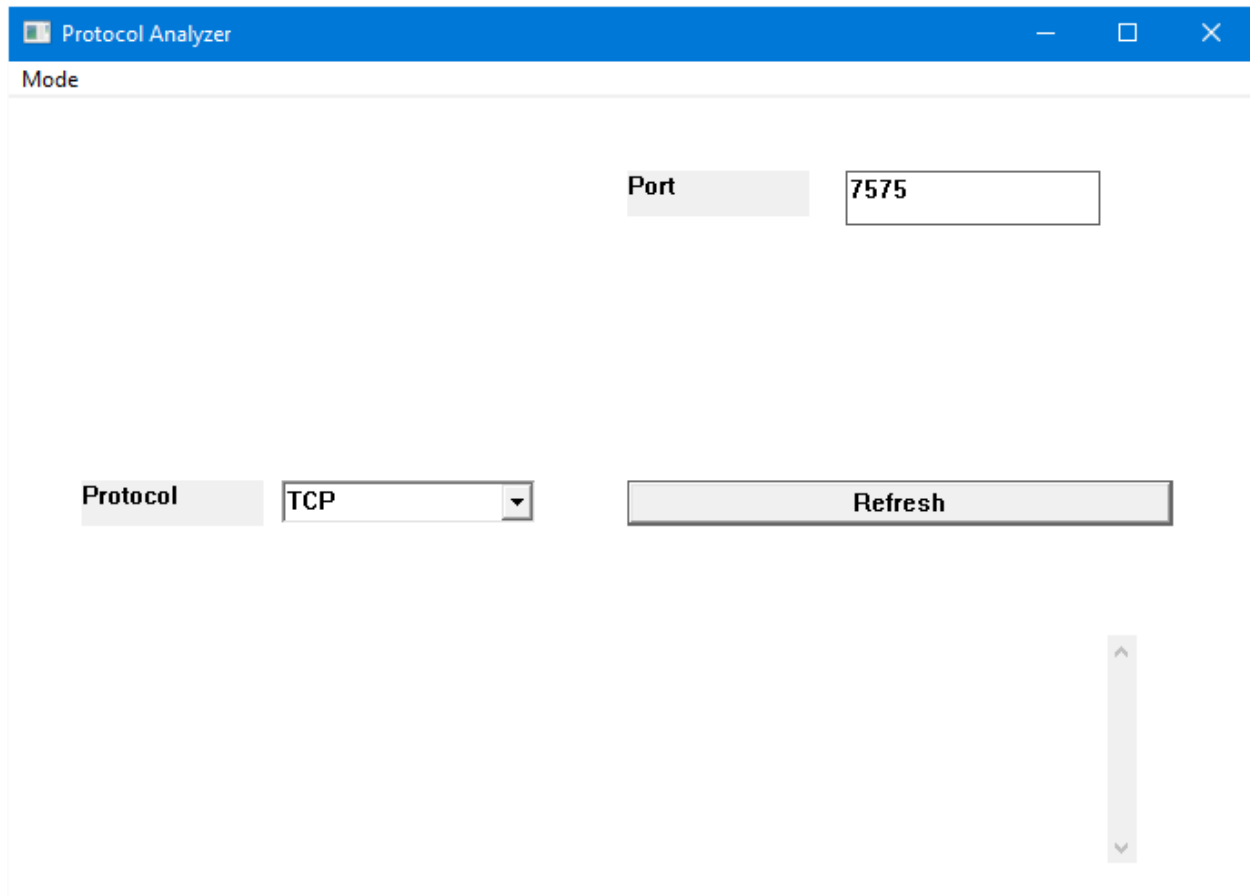
### Test 3) Client/Server Mode Changes Modifies UI

**Description:** When the user changes between client and server mode, the UI will be changed. We do this by clicking the Mode option at the top right and changing over to server or back to client.

**Result:** The UI will have elements that are irrelevant disappear on server mode, and reappear on client mode.

**Result:** Passed

*Figure 3) UI in server mode.*



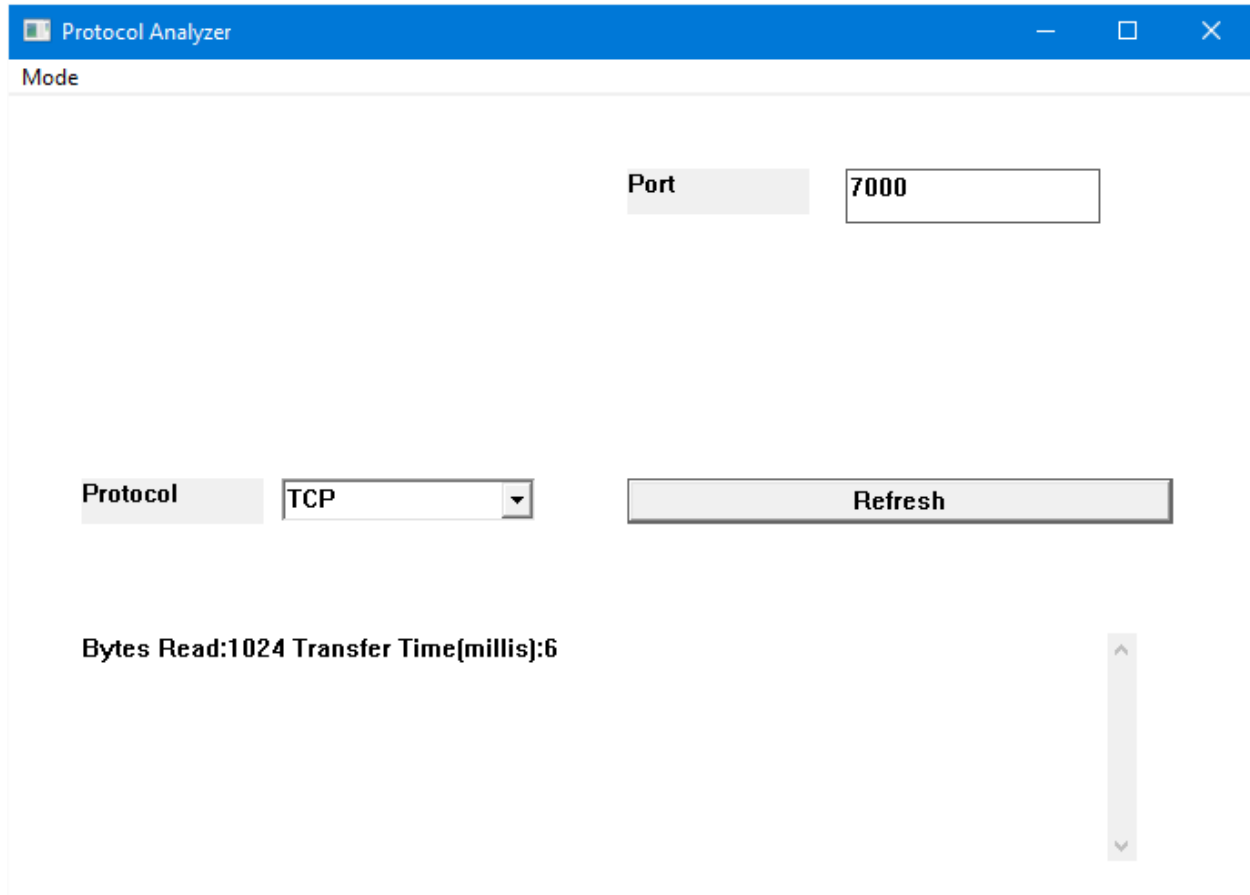
## Test 4) Sending a single TCP packet

**Description:** The TCP client can send a server application a single packet.

**Result:** The server displays the data for a single packet.

**Result:** Passed.

*Figure 4) Server UI*



The screenshot shows a web application titled "Protocol Analyzer" with a blue header bar. Below the header, the word "Mode" is displayed in a light gray box. The main content area is white and contains several interactive elements: a "Port" label next to a text input field containing "7000"; a "Protocol" label next to a dropdown menu currently showing "TCP"; and a "Refresh" button. At the bottom, a status line reads "Bytes Read:1024 Transfer Time(millis):6". A vertical scrollbar is visible on the right side of the main content area.



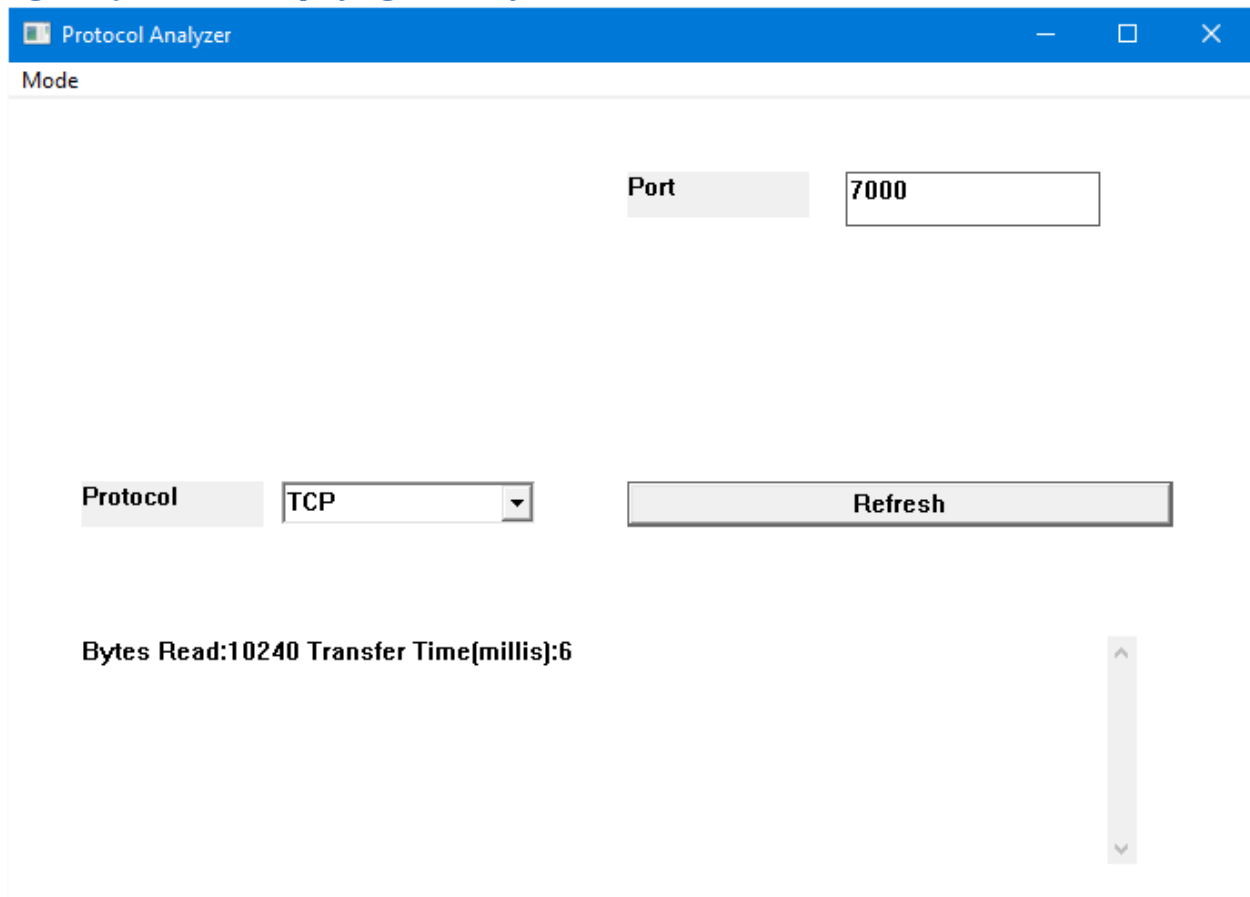
## Test 5) Sending multiple TCP packets

**Description:** The client can send multiple TCP packets to a server instance and the server instance acknowledges them. Send 10 different 1024 packets to test.

**Result:** The server instance displays 10240 bytes read, AKA 10 different 1024 packets were received.

**Result:** Passed

*Figure 5) Server UI displaying 10240 bytes read*



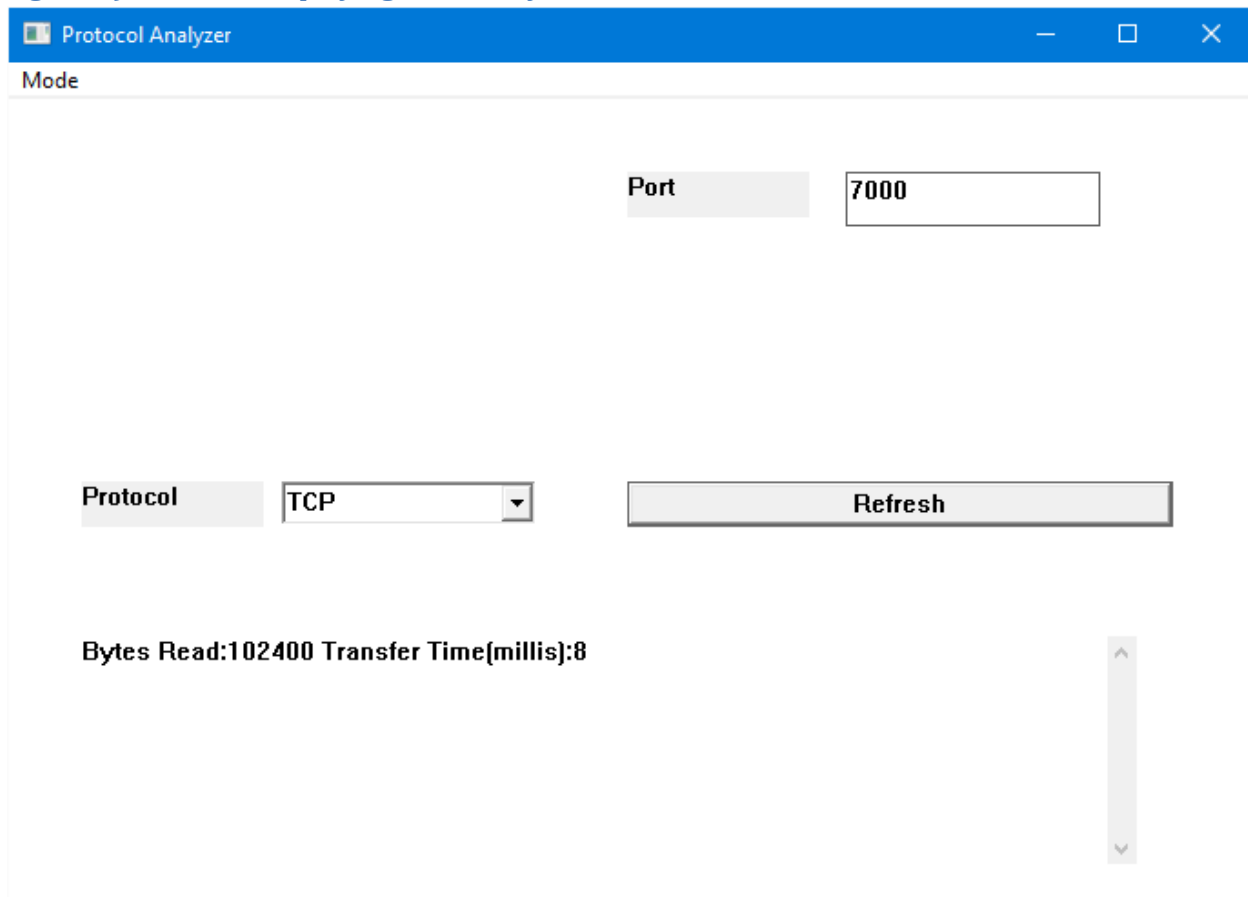
## Test 6) Sending 100 packets size 1024 in TCP

**Description:** A client TCP application can send 100 different packets of size 1024 to the server application.

**Result:** The server displays 102400 bytes read, AKA 100 different 1024 byte packets read.

**Result:** Passed

*Figure 6) Server UI displaying 102400 bytes read*



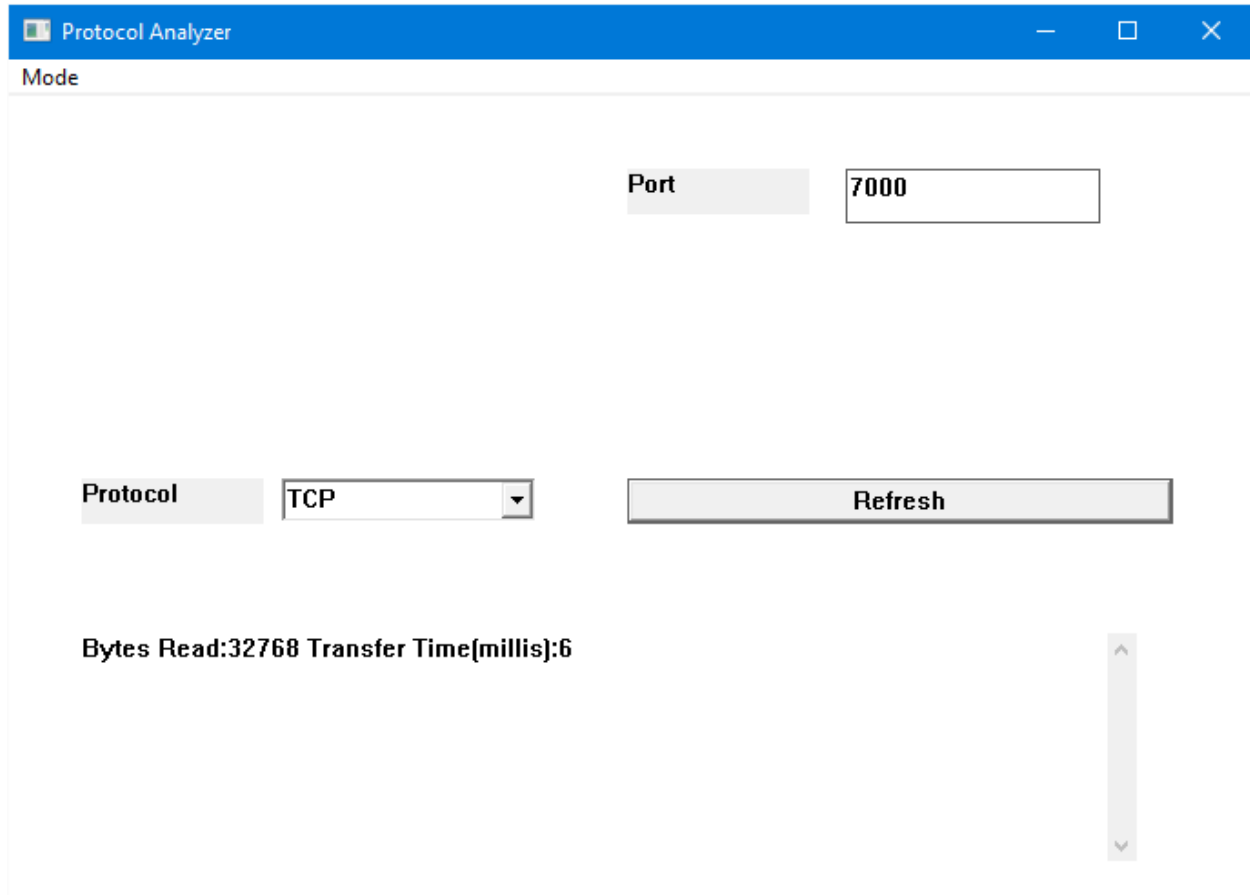
## Test 7) Large 32768 byte packets can be sent via TCP

**Description:** Client can send a large packet in TCP to server instance. Size for test is 32768.

**Result:** 32768 bytes read by server displayed on screen.

**Result:** Passed

*Figure 7) Server UI*



The screenshot shows a web application titled "Protocol Analyzer" with a blue header bar. Below the header, the word "Mode" is displayed in a light gray box. The main content area is white and contains several controls:

- A "Port" label in a gray box next to a text input field containing the value "7000".
- A "Protocol" label in a gray box next to a dropdown menu showing "TCP".
- A "Refresh" button in a gray box.
- A status line displaying "Bytes Read:32768 Transfer Time(millis):6".
- A vertical scrollbar on the right side of the main content area.

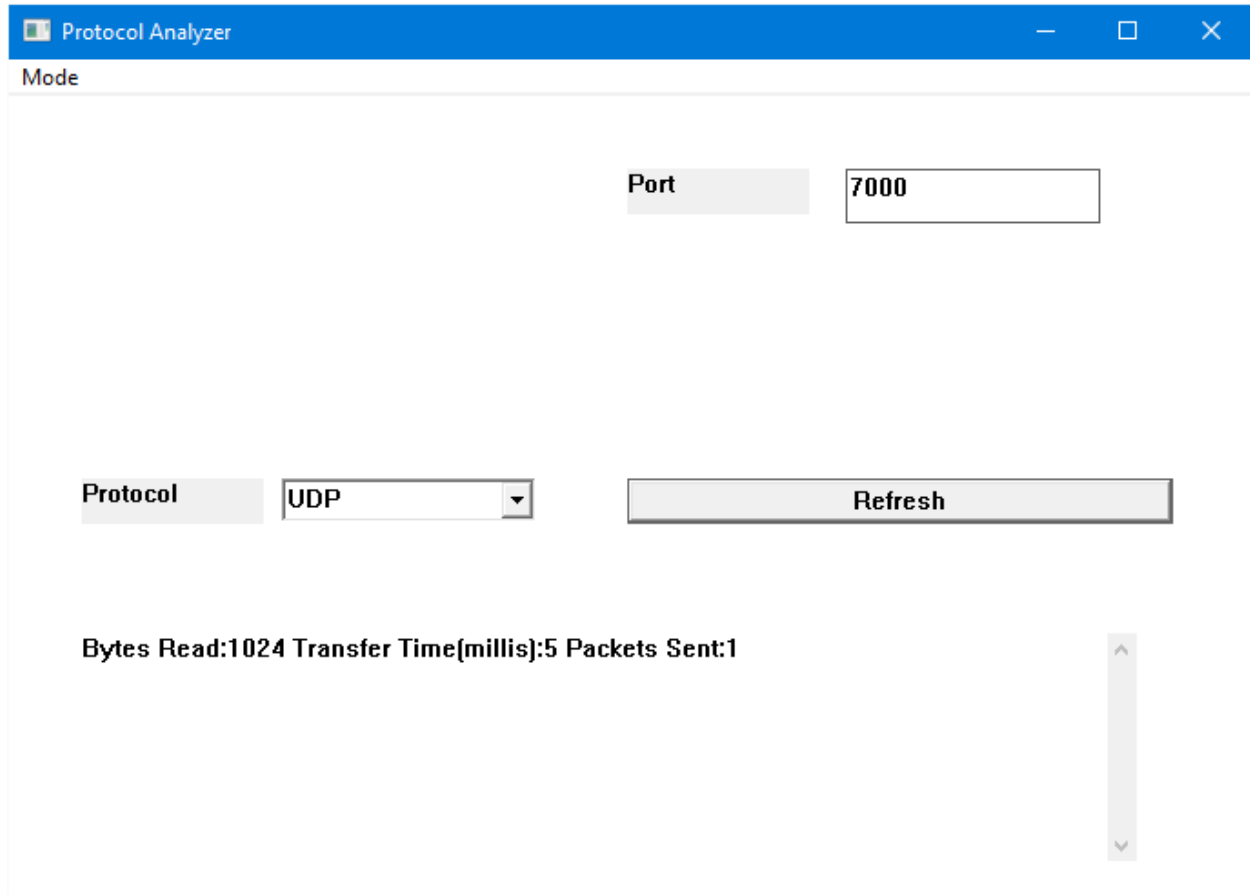
## Test 8) Sending a single UDP packet

**Description:** The UDP client can send a server application a single packet.

**Result:** The server displays the data for a single packet.

**Result:** Passed.

*Figure 8) Server UI*



The screenshot shows a web application window titled "Protocol Analyzer". The interface includes a "Mode" label, a "Port" input field with the value "7000", a "Protocol" dropdown menu set to "UDP", and a "Refresh" button. At the bottom, it displays statistics: "Bytes Read:1024 Transfer Time(millis):5 Packets Sent:1". A vertical scrollbar is visible on the right side of the main content area.

Protocol Analyzer

Mode

Port 7000

Protocol UDP Refresh

Bytes Read:1024 Transfer Time(millis):5 Packets Sent:1

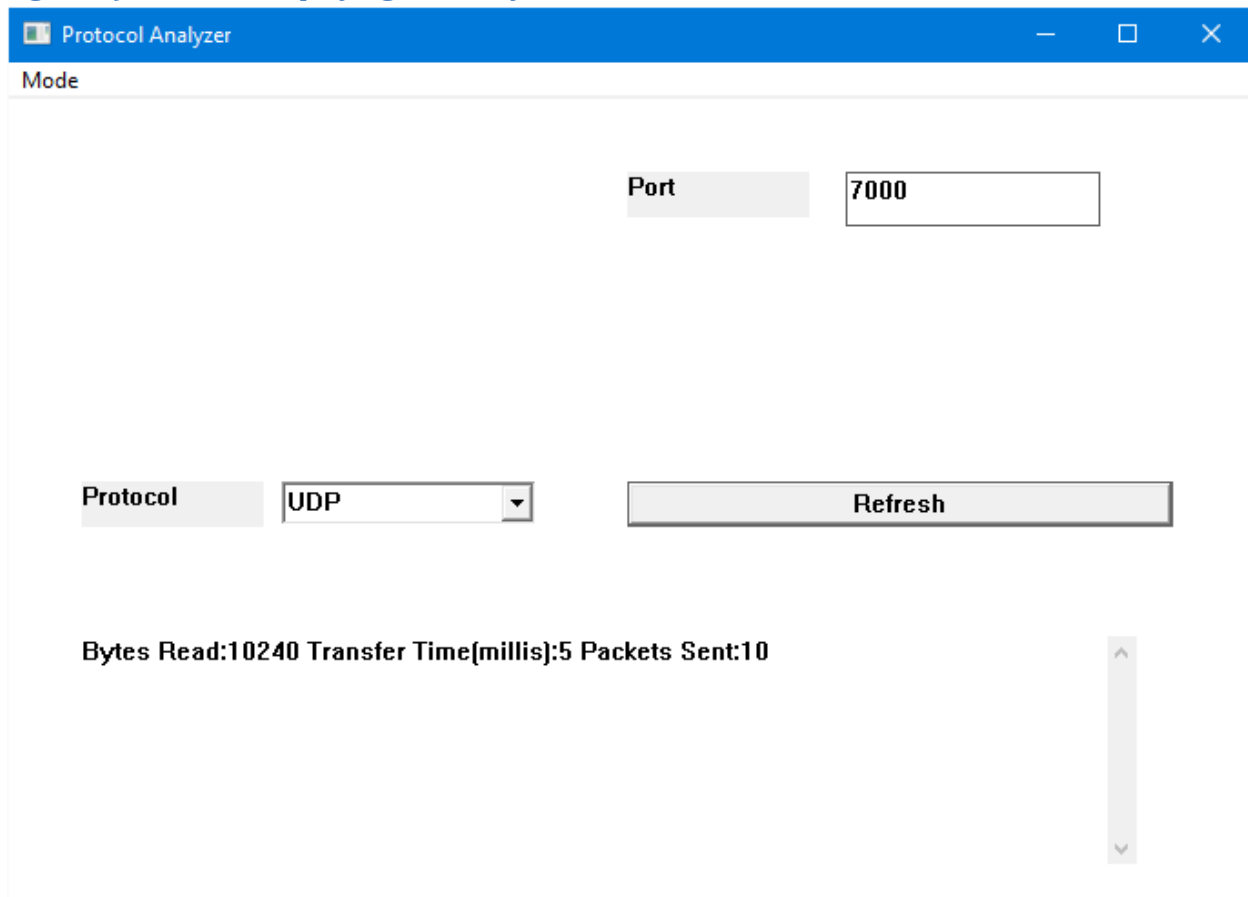
## Test 9) Sending multiple UDP packets

**Description:** The client can send multiple UDP packets to a server instance and the server instance acknowledges them. Send 10 different 1024 packets to test.

**Result:** The server instance displays “Packets Sent: 10”

**Result:** Passed

*Figure 9) Server UI displaying 10240 bytes read*



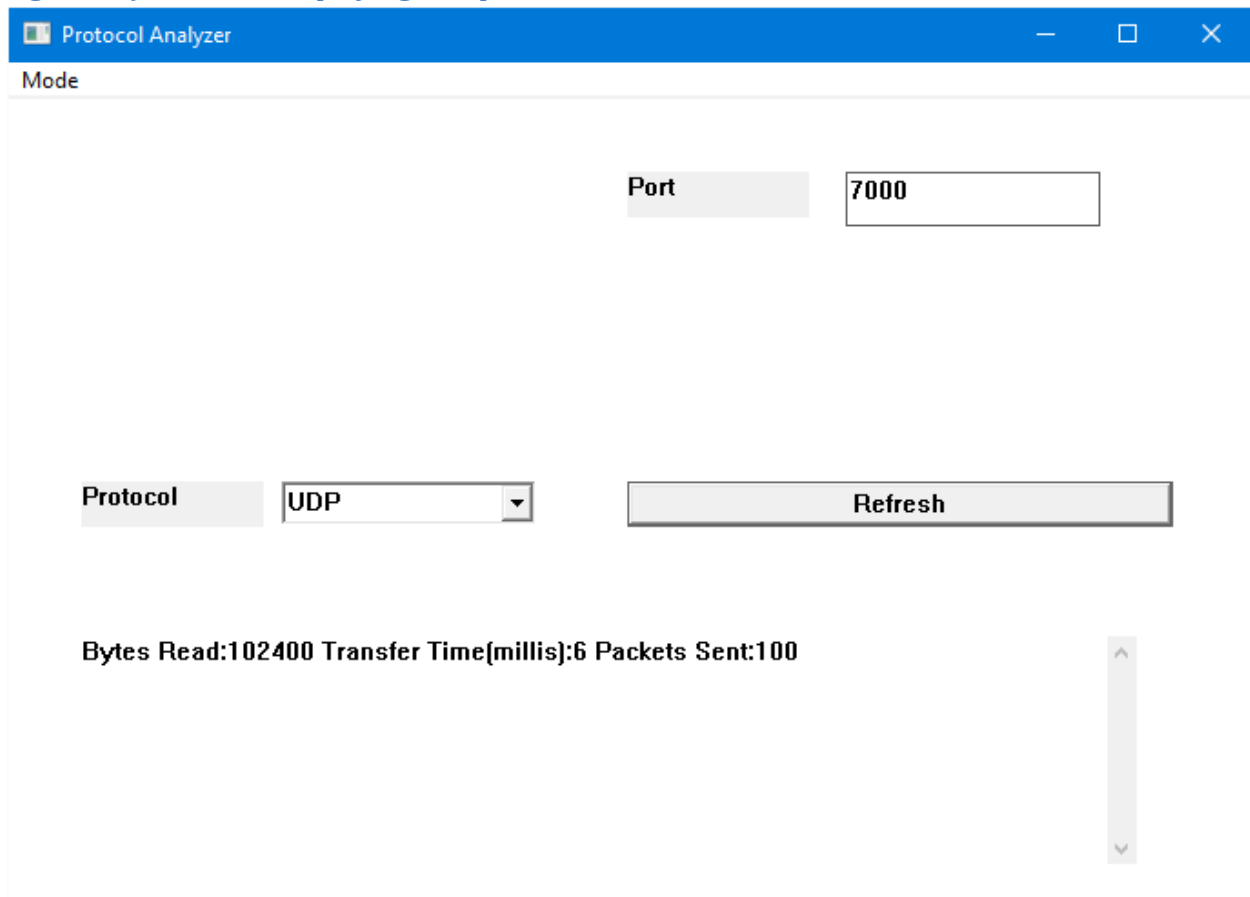
## Test 10) Sending 100 packets size 1024 in UDP

**Description:** A client UDP application can send 100 different packets of size 1024 to the server application.

**Result:** The server displays “Packets Sent: 100”

**Result:** Passed

*Figure 10) Server UI displaying 100 packets read*



The screenshot shows a window titled "Protocol Analyzer" with a "Mode" tab. The interface includes a "Port" label next to a text box containing "7000". Below this, there is a "Protocol" label next to a dropdown menu showing "UDP", and a "Refresh" button. At the bottom, the status text reads "Bytes Read:102400 Transfer Time(millis):6 Packets Sent:100". A vertical scrollbar is visible on the right side of the status area.

Port
7000

Protocol
UDP

Refresh

Bytes Read:102400 Transfer Time(millis):6 Packets Sent:100

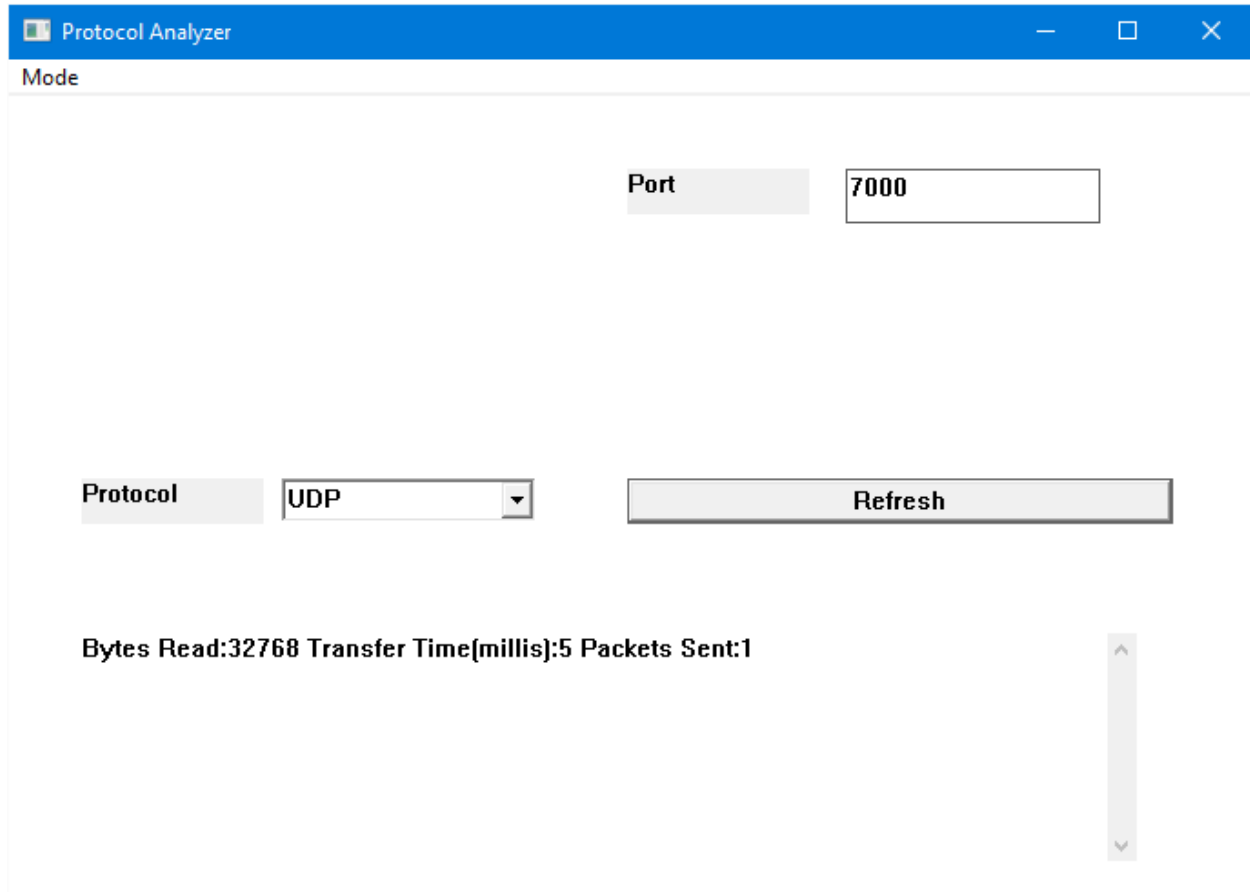
## Test 11) Large 32768 byte packets can be sent via UDP

**Description:** Client can send a large packet in UDP to server instance. Size for test is 32768.

**Result:** 32768 bytes read by server displayed on screen.

**Result:** Passed

*Figure 11) Server UI*



The screenshot shows a web application titled "Protocol Analyzer" with a "Mode" header. It features a "Port" input field set to "7000", a "Protocol" dropdown menu set to "UDP", and a "Refresh" button. Below these controls, the test results are displayed: "Bytes Read:32768 Transfer Time(millis):5 Packets Sent:1". A vertical scrollbar is visible on the right side of the results area.

Port
7000

Protocol
UDP

Refresh
Refresh

Bytes Read:32768 Transfer Time(millis):5 Packets Sent:1

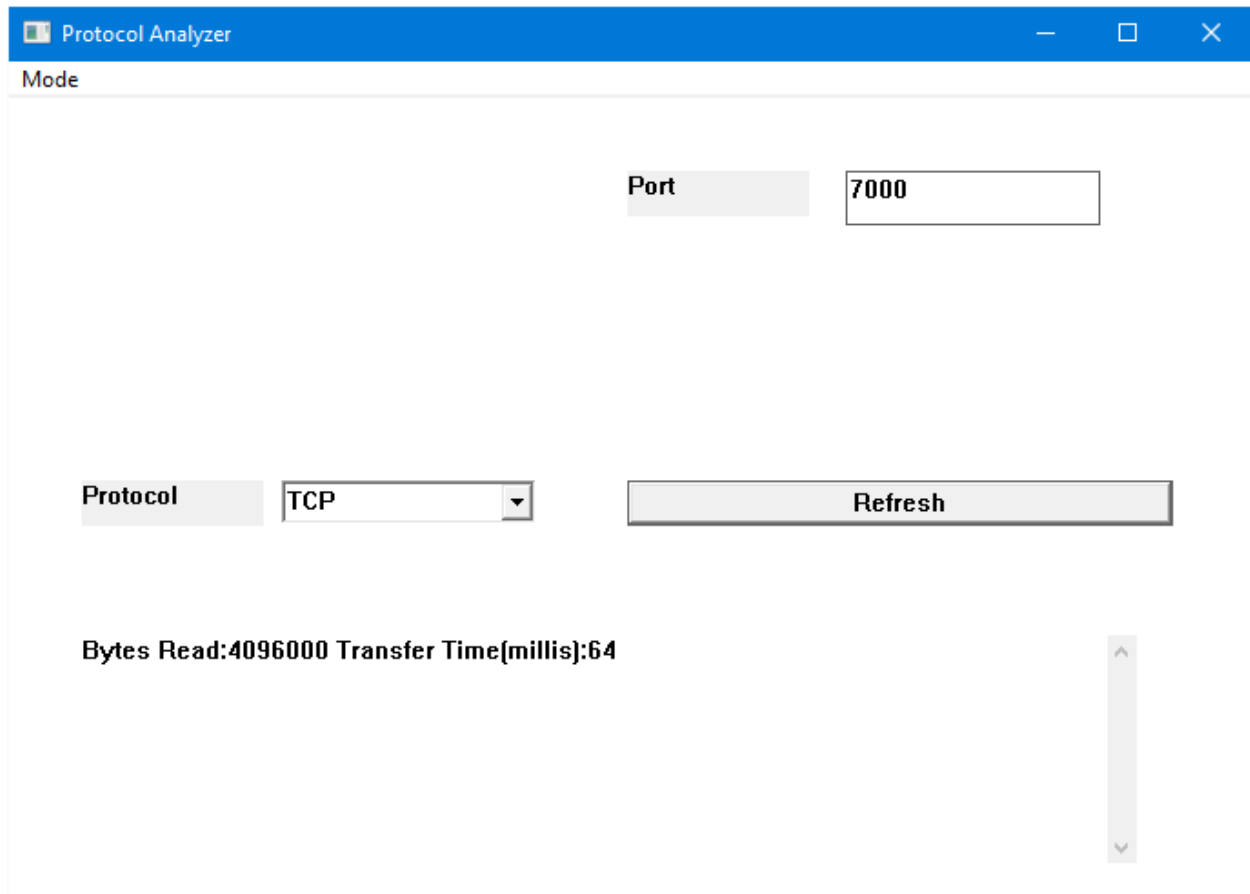
## Test 12) TCP can send 500 different packets size 8192

**Description:** TCP client can send 500 packets all size 8192 to TCP server

**Result:** 4096000 (500 packets times 8192 bytes per) bytes displayed on screen under Bytes Received

**Result:** Passed

*Figure 12) Server UI*



The screenshot shows a web application titled "Protocol Analyzer" with a blue header bar. Below the header, the word "Mode" is displayed in a light gray box. The main content area is white and contains several controls: a "Port" label next to a text input field containing "7000"; a "Protocol" label next to a dropdown menu showing "TCP"; and a "Refresh" button. At the bottom, the text "Bytes Read:4096000 Transfer Time(millis):64" is displayed. A vertical scrollbar is visible on the right side of the main content area.

## Test 13) UDP can send 500 different packets size 8192

**Description:** TCP client can send 500 packets all size 8192 to UDP server



**Result:** Server displays 500 packets read

**Result:** Passed

*Figure 13) Server UI*

Protocol Analyzer

Mode

Port 7000

Protocol UDP Refresh

Bytes Read:4096000 Transfer Time(millis):62|Packets Sent:500