OT/SCADA Penetration Testing Methodology

55 Minutes Remaining

Instructions Resources Help  100%

Exercise 1: ModBus Protocol Analysis - I

Objective

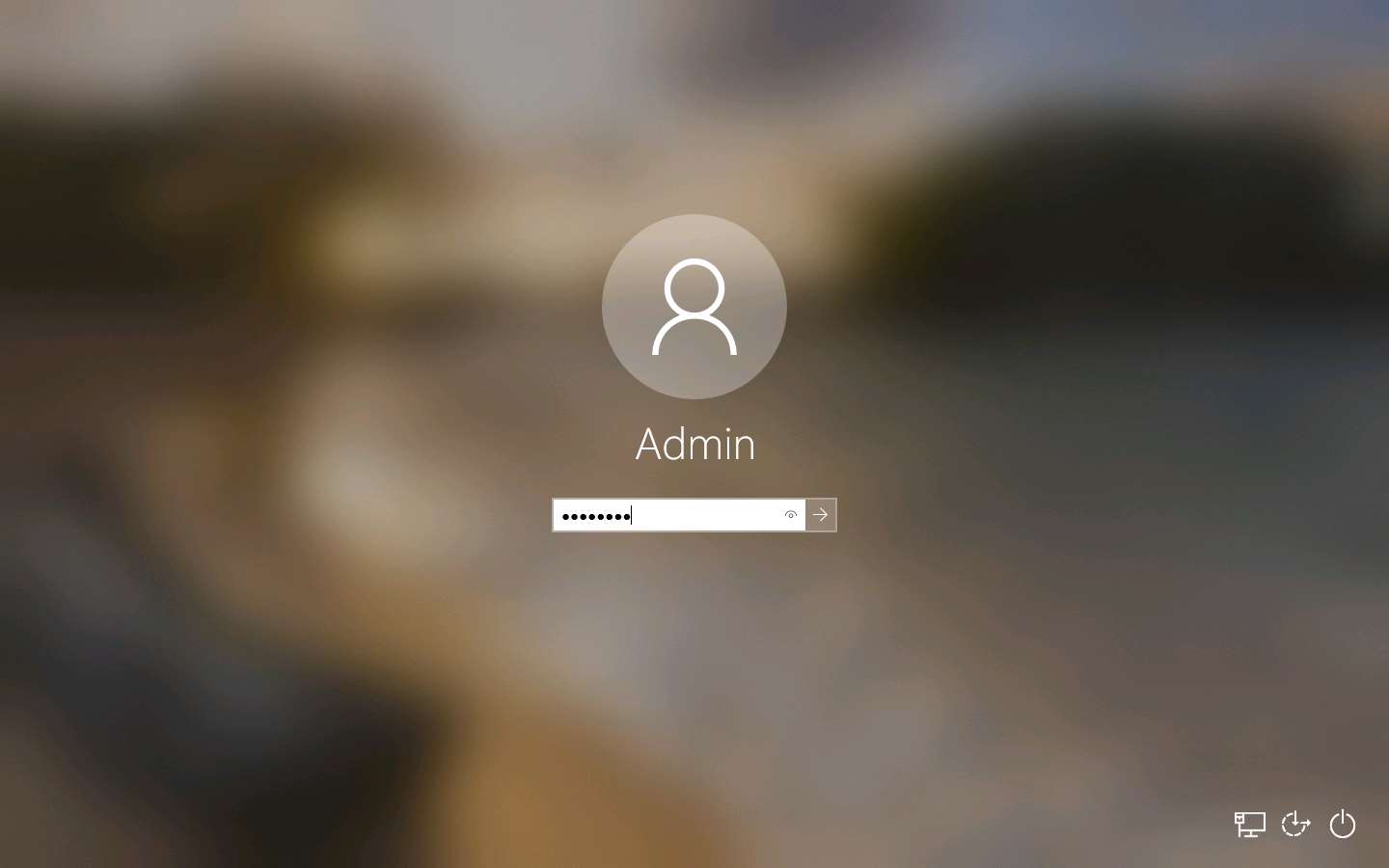
* Explore the ModBus protocol at the packet level

**Lab Duration**: **15** Minutes

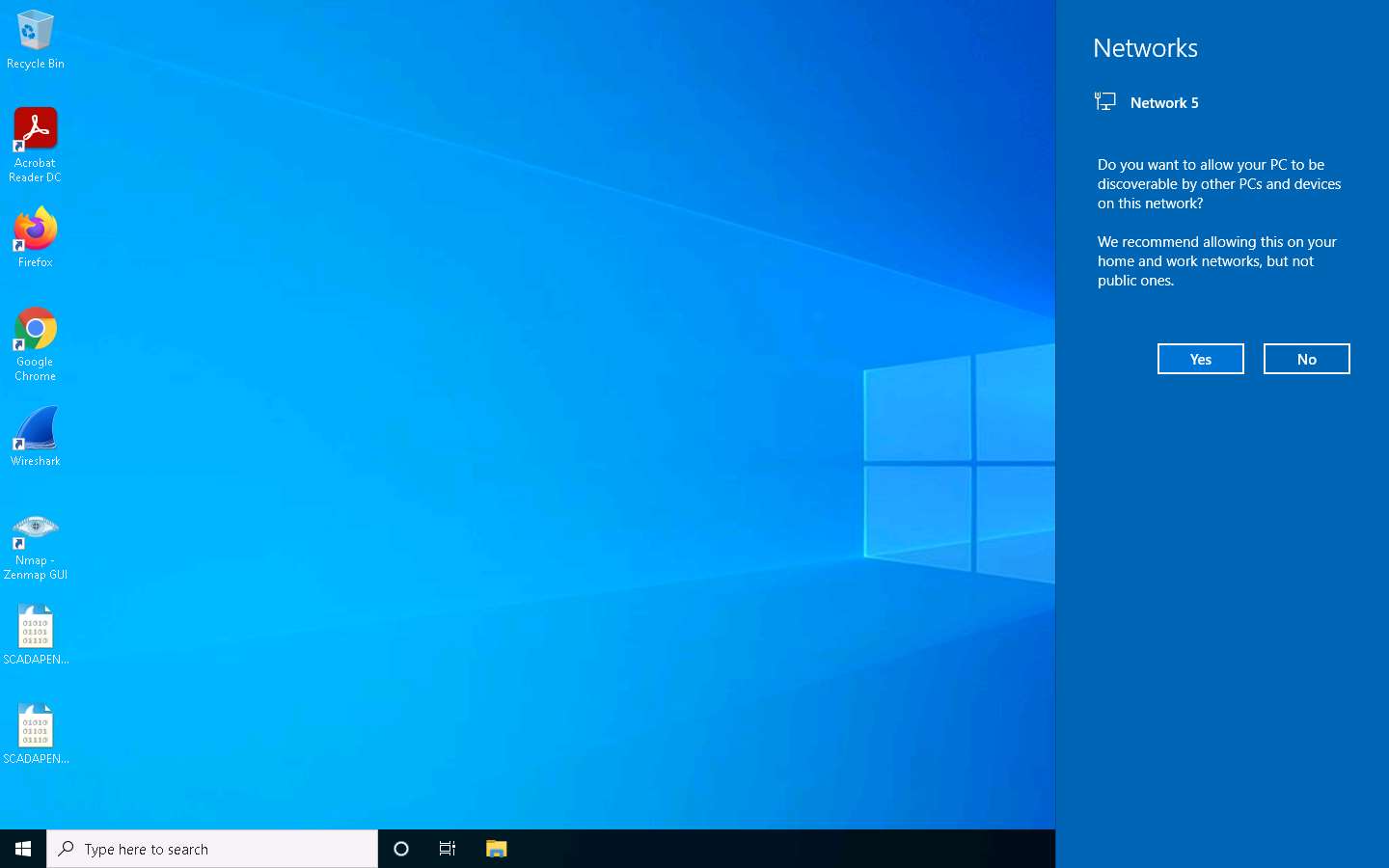
1. Click [SCADA Master](https://labclient.labondemand.com/Instructions/f213e2c3-c5a7-47dc-a7af-efba687edcf4?rc=10) and click [Ctrl+Alt+Delete](https://labclient.labondemand.com/Instructions/f213e2c3-c5a7-47dc-a7af-efba687edcf4?rc=10).



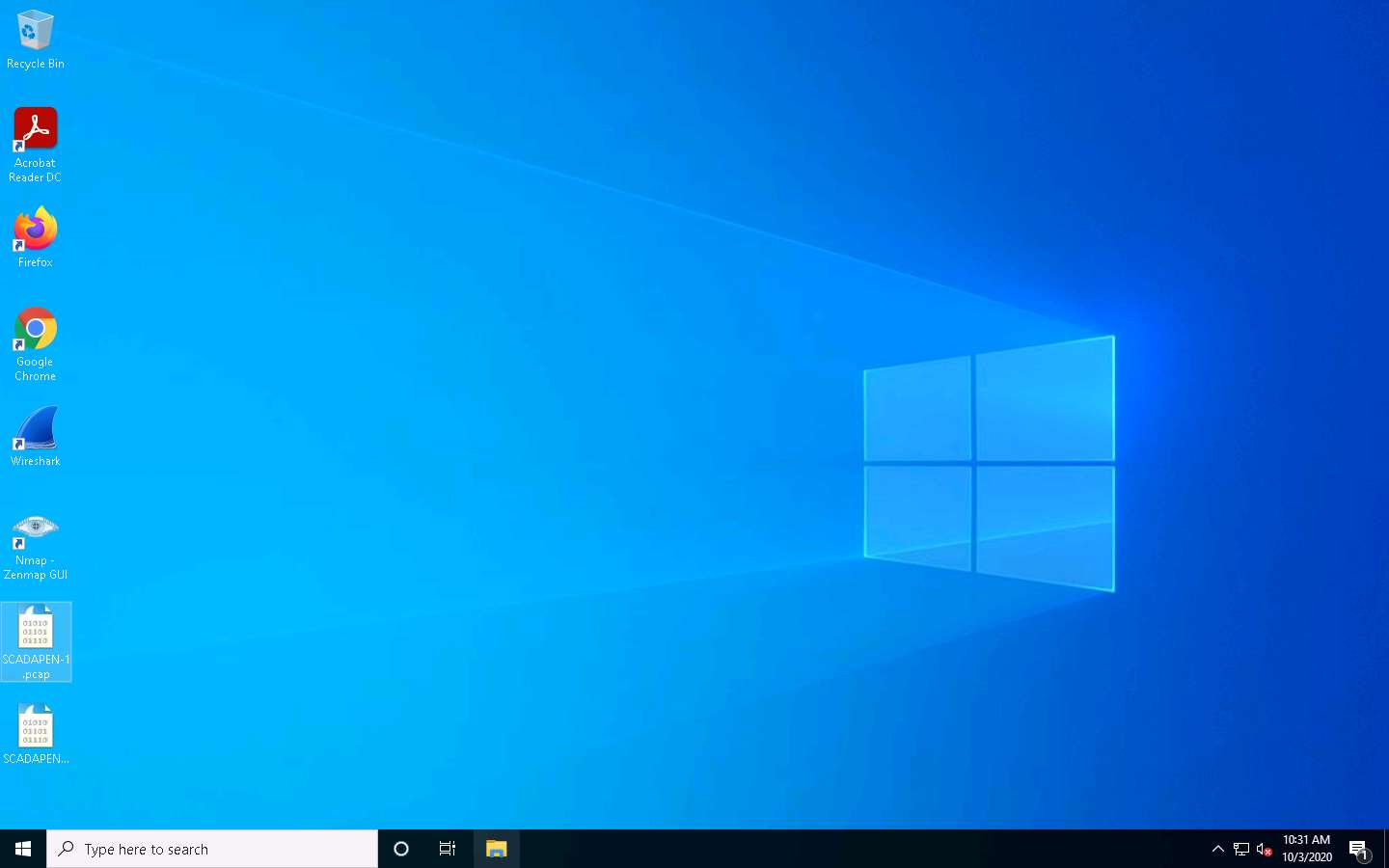
1. Click Pa$$w0rd or type **Pa$$w0rd** and press **Enter**.



1. Networks screen appears, click **Yes**.



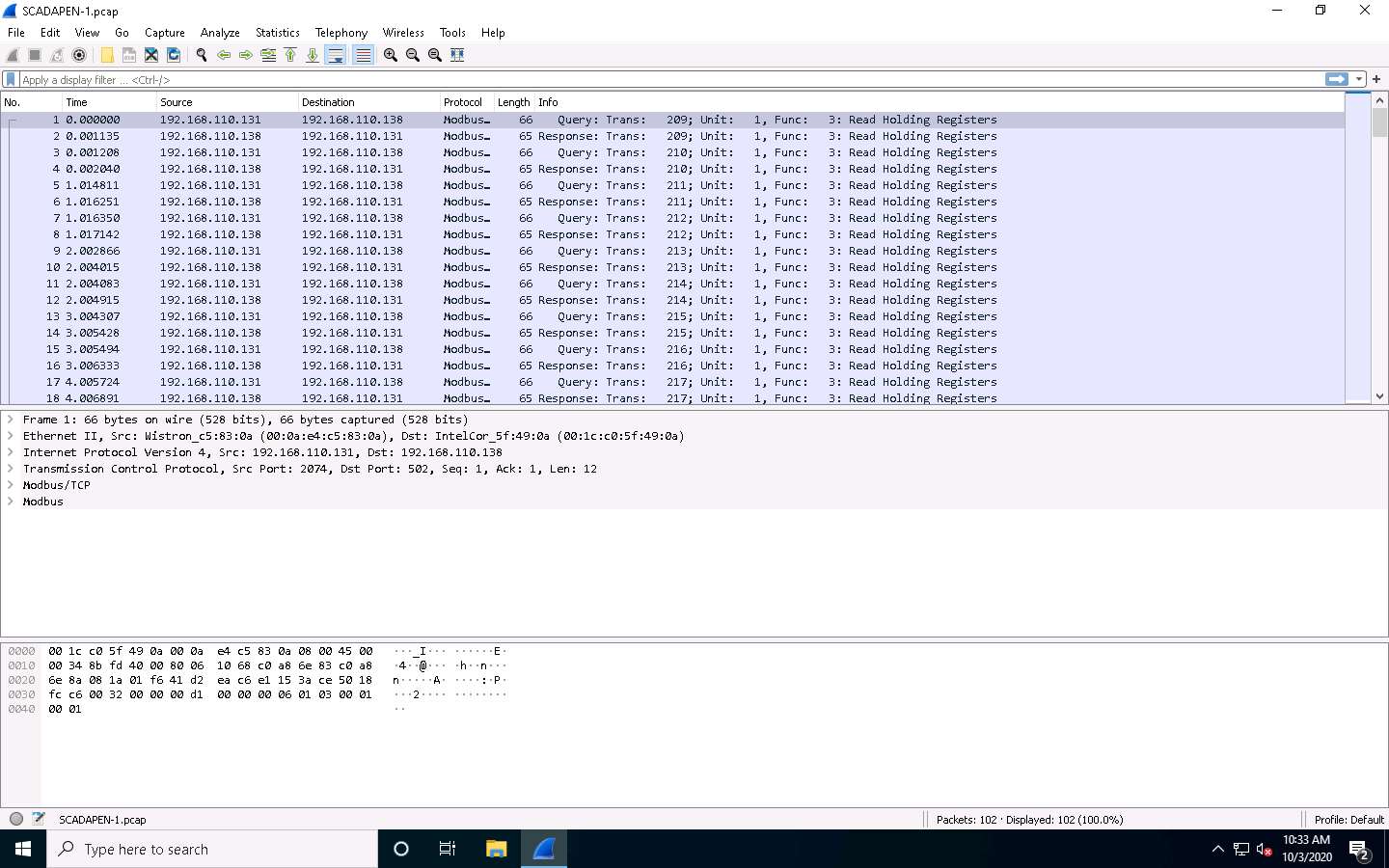
1. In this Exercise, you will review the network traffic of communication among devices on a network that uses the **ModBus** protocol.
2. Double-click **SCADAPEN-1.pcap** file on **Desktop**.



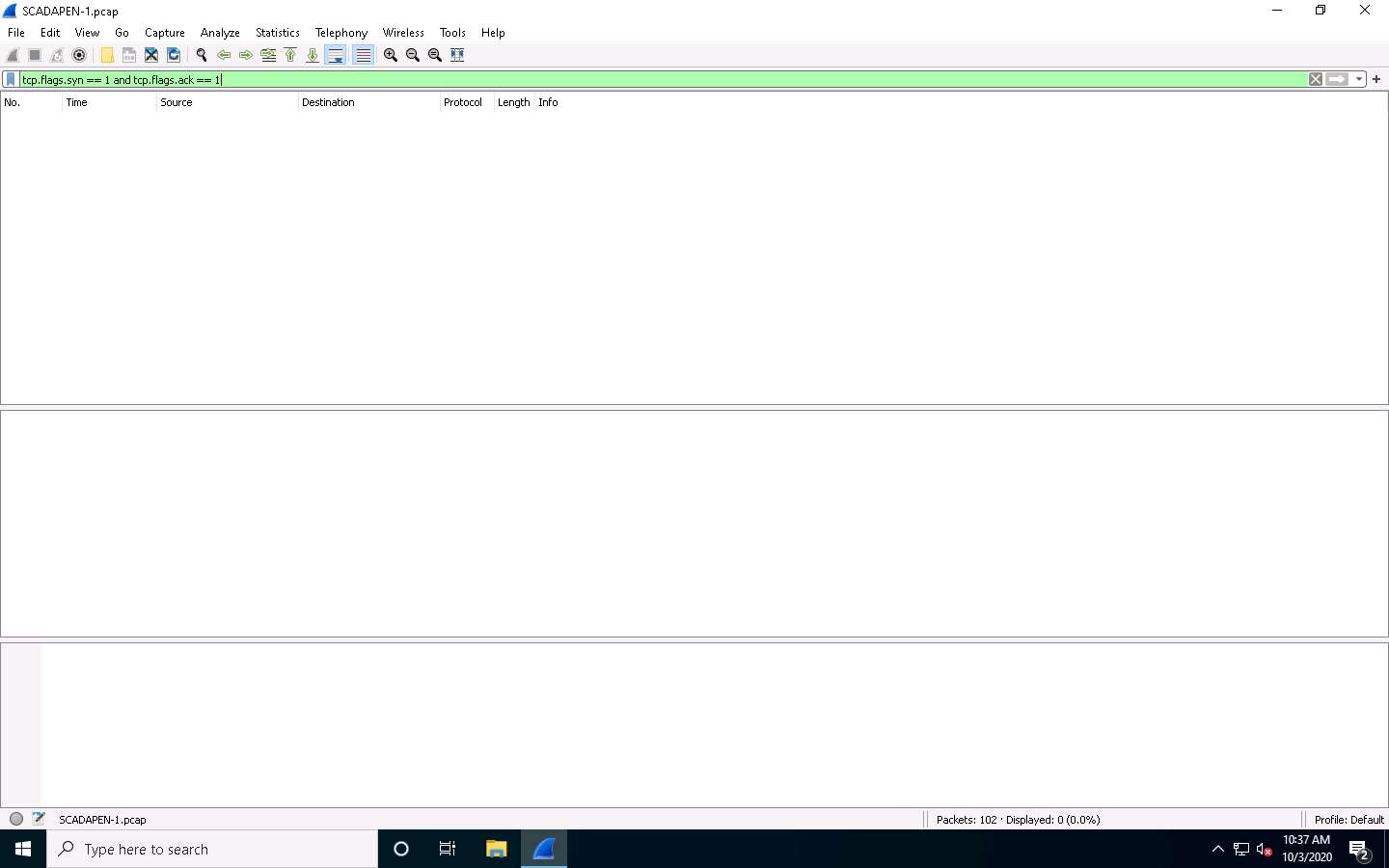
1. The **SCADAPEN-1.pcap** files opens up in the Wireshark window as shown in the screenshot. As a review, the protocol is mostly a **master**/**slave** type of communication. As noted in the lecture, the protocol is in clear text and does not require authentication. As a result, it is highly susceptible to a **man-in-the-middle** (MITM) attack.

If Software Update for Wireshark pop-up appears, click **Skip this version** or **Remind me later**.

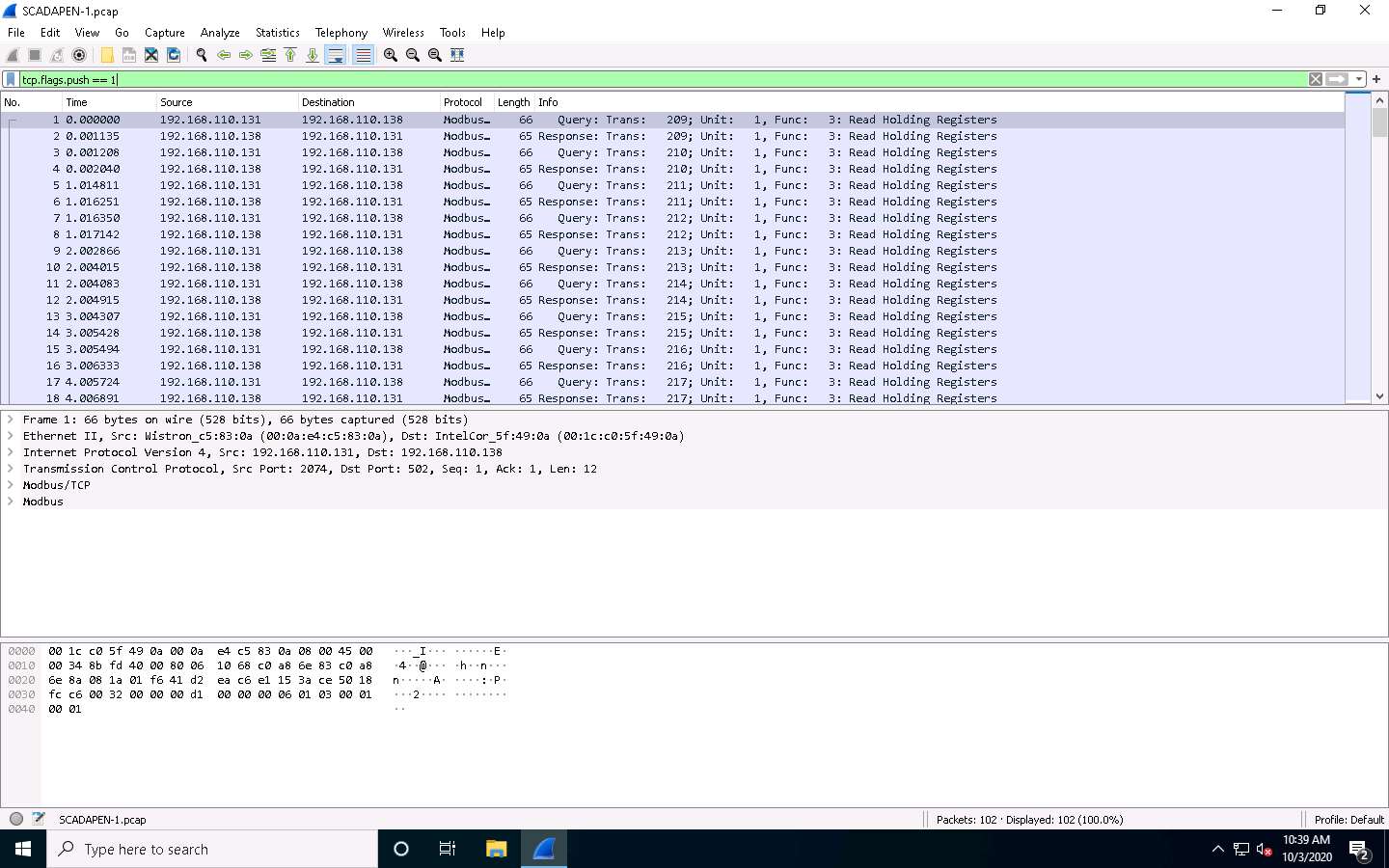
1. You can have only one **master** on a “**Modbus**” network and maximum **247** slaves, each with a **unique slave ID**. In the serial world, the devices must been connected in a **daisy-chain** manner, and not in a star topology.
2. In **Transmission Control Protocol** (TCP), we often refer to the master as the client and to the slave as the server. Thus, continue to use and reference the **ModBus** protocol running using TCP and not the serial version.



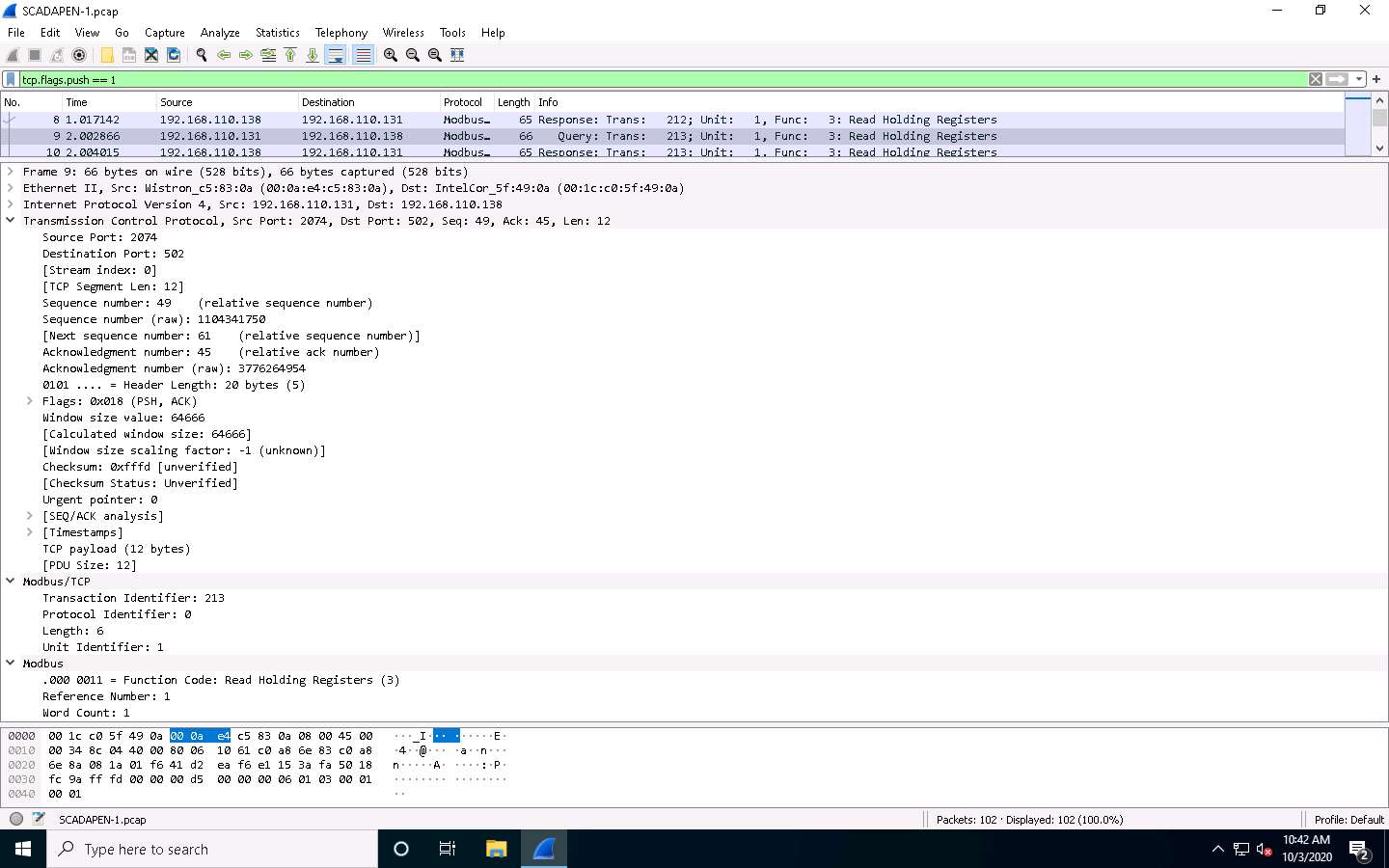
1. As shown in the above screenshot, there is communication between a **master** and a **slave** using the **ModBus** protocol; in this actual example, the communication is working with registers.
2. In protocol analysis, you will be taught the following process and methodology:
   * Suspicious
   * Open ports
   * Data
   * Review sessions
   * Look for signs of compromise
3. From the file capture, let us explore the second step. In the filter field, type **tcp.flags.syn == 1 and tcp.flags.ack == 1** and press **Enter**. This flag will show all open ports that are located in the capture file.
4. The results show that you are dealing with a capture file that has been stripped, or the data have been extracted, and none of the handshakes of steps of TCP are represented in the file. Moreover, the packet capture file could consist of the **User Datagram Protocol** (UDP) traffic as well.



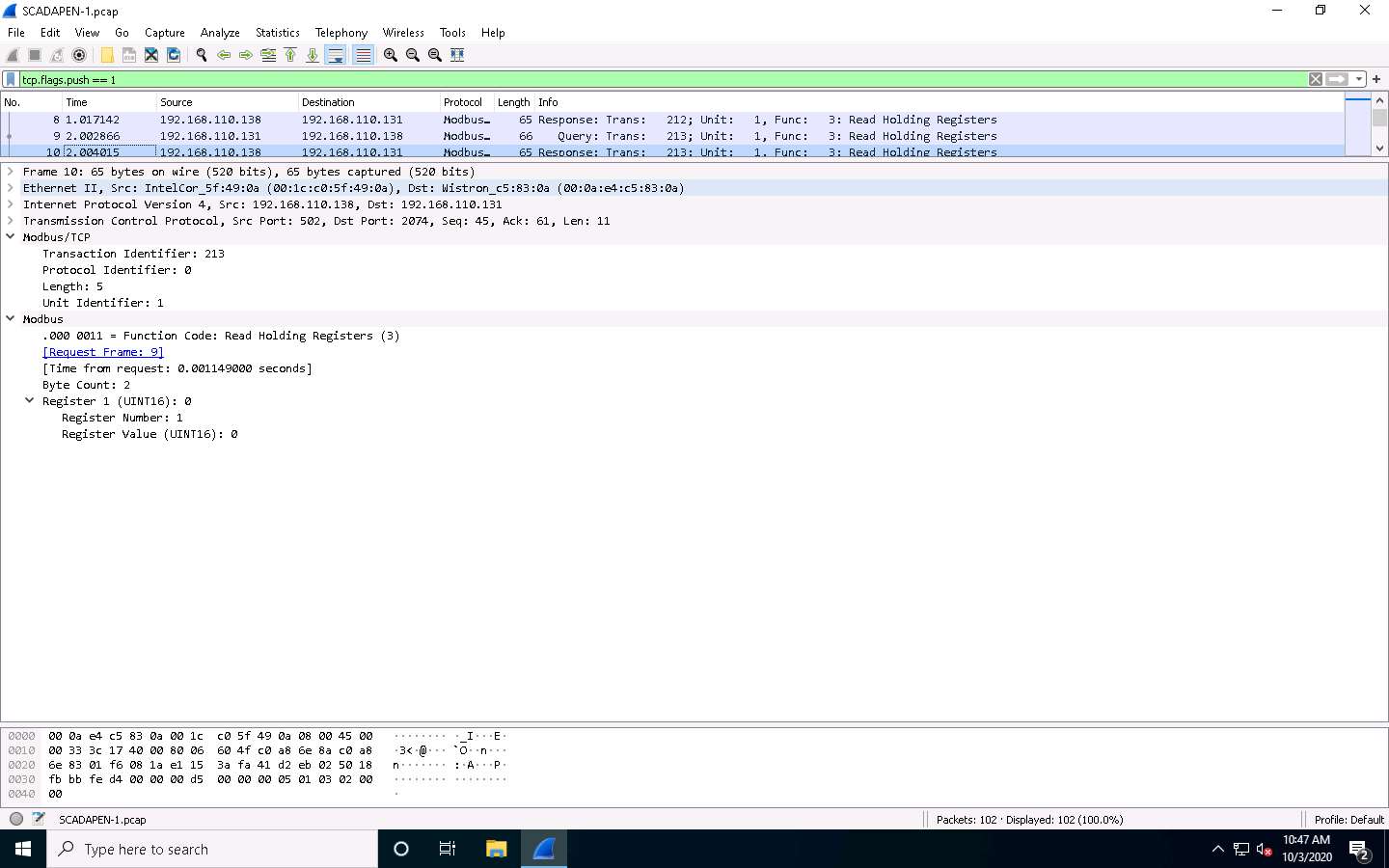
1. Let us look for the data. In the filter field, enter the filter for the data, and then review the results. In the filter field, clear the current filter and type **tcp.flags.push == 1** and press **Enter**. Any packet that has data in it will have the **Push flag set**.



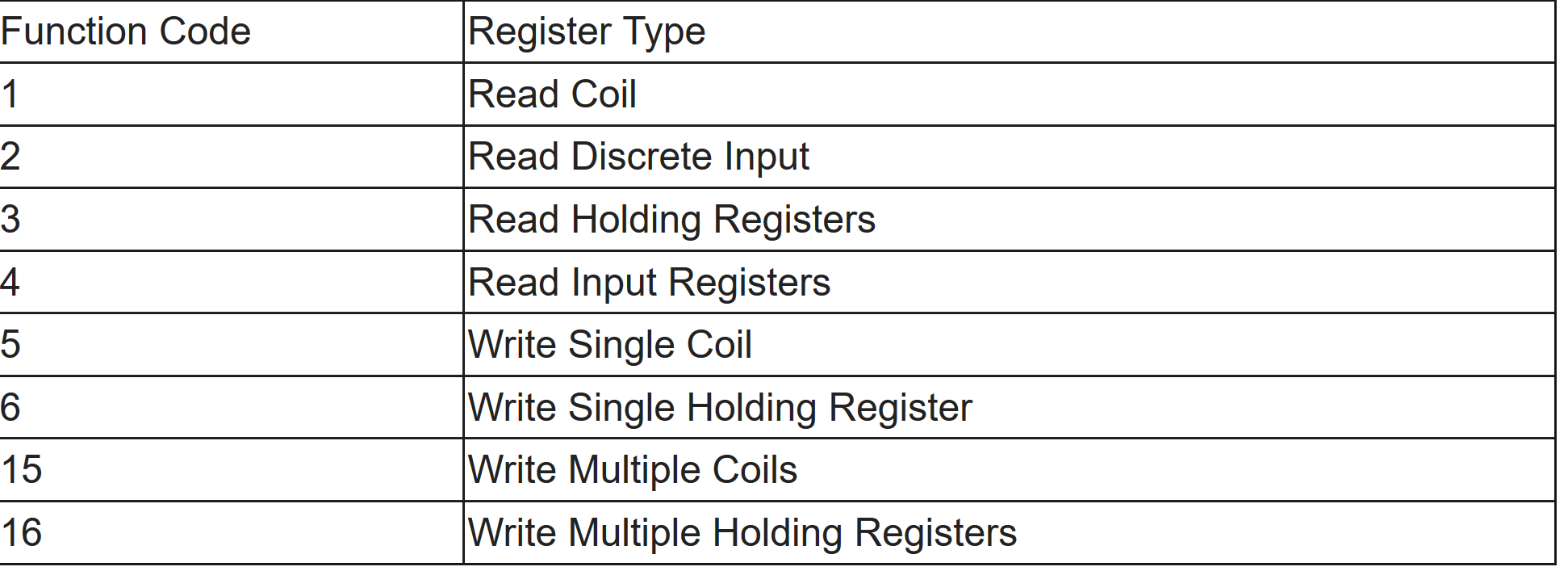
1. In the case of this packet capture file, you only have the data. Select packet number **9**. The example of the middle section of the packet is shown in the screenshot.
2. You now have the protocol Modbus located within the **TCP** packet. Review the section of the packet that represents the **Modbus/TCP**. Importantly, note the Unit Identifier, as you can use it to reference the data that are contained within the protocol and one of the areas that the attacker will target. Remember that this protocol was created when the network was considered trusted.
3. Regarding Modbus unit devices, in most cases you do not need a **unit ID**, because you have already addressed the correct unit via its IP address. In some cases, you will run into a situation wherein multiple devices are connected to one IP address (for example, “bridges”). In such a case, the unit ID is not required.
4. The unit ID of 0 can be seen as a broadcast address. Messages sent to 0 can be accepted by all slaves. If you setup a Modbus client, remember that it cannot have unit ID 0. It might have to be set to 255. 17. Next, explore the data for Modbus that are encapsulated within the Modbus/TCP packet.



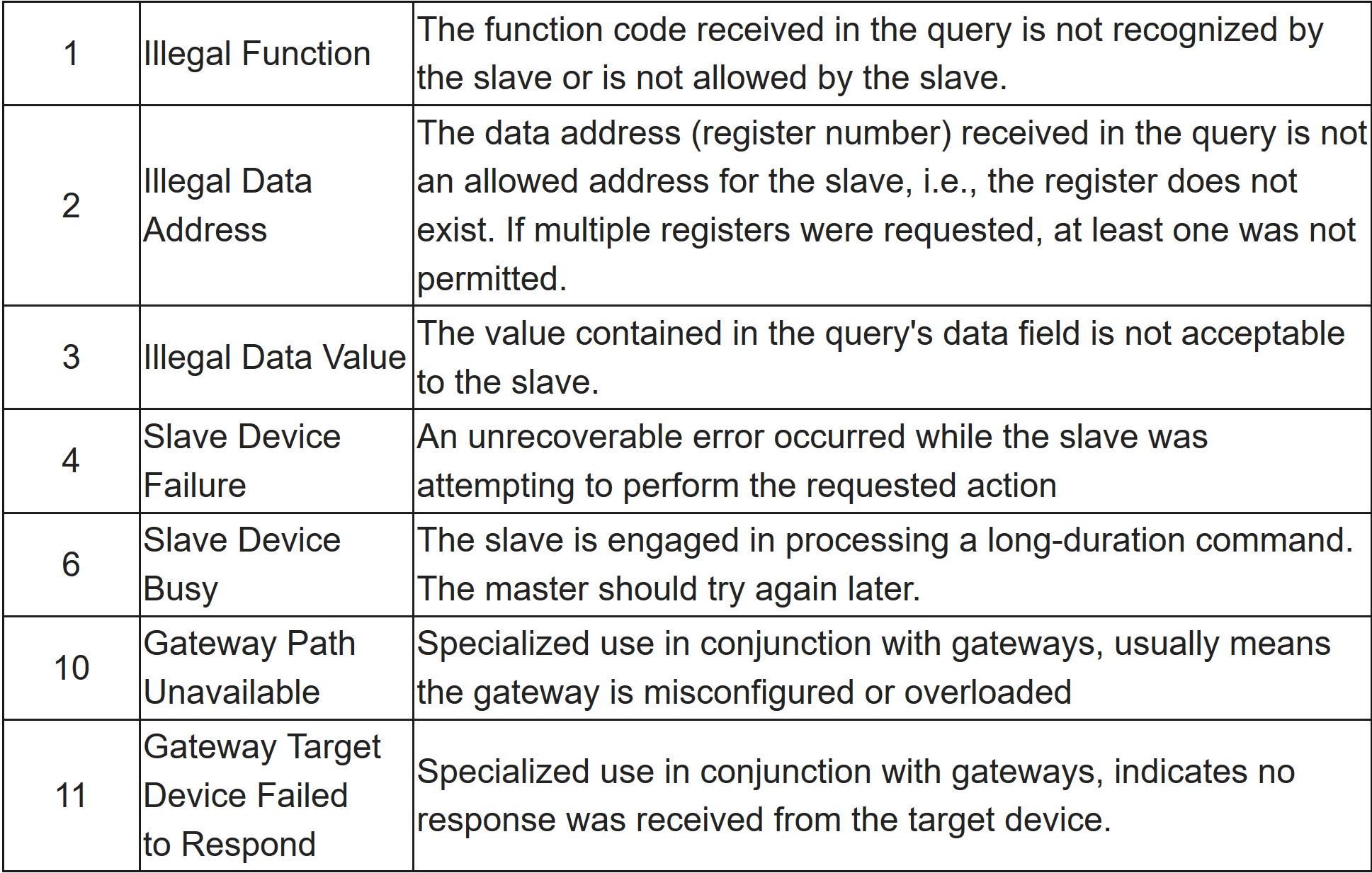
1. Now, select packet number **10** and observe the **Modbus** node in the middle section. Since the image is a response, the data include the information in the register. There are two types of places where information can be stored: coils and registers. Each of these datastore types has two different types of registers: a read/write and a read only. Each of these datastore types, in turn, is a reference to a memory address.



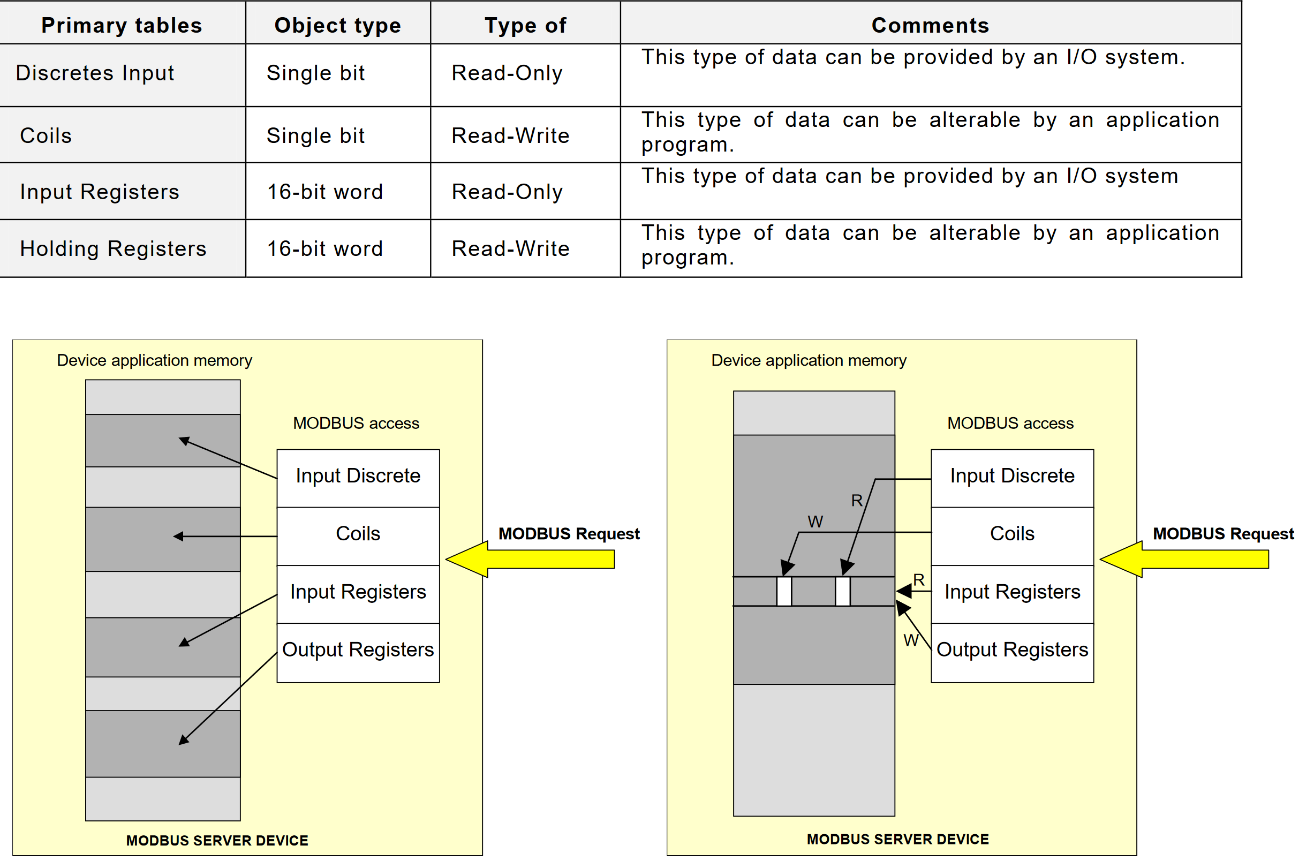
1. This is the process for reviewing the communications sequences of the Modbus protocol that, in this case, is interacting with the registers. Next, explore the traffic reading the coils.
2. Simply put, a coil is used for storing simple Booleans (1 bit). It is “read/write,” and starts from 00001 to 09999; a discrete input is a read only type for Booleans, starting from 10001 to 19999; an input register is a read only type for longer values (16 bits), starting from 30001 to 39999; and a holding register is a read/write type for longer values (16 bits), starting from 40001 to 49999.
3. Beware that, depending on the hardware implementation, sometimes the registers start at 0 and sometimes at 1.Before you continue, review more information about the protocol. The screenshot shows an example of the different function codes:



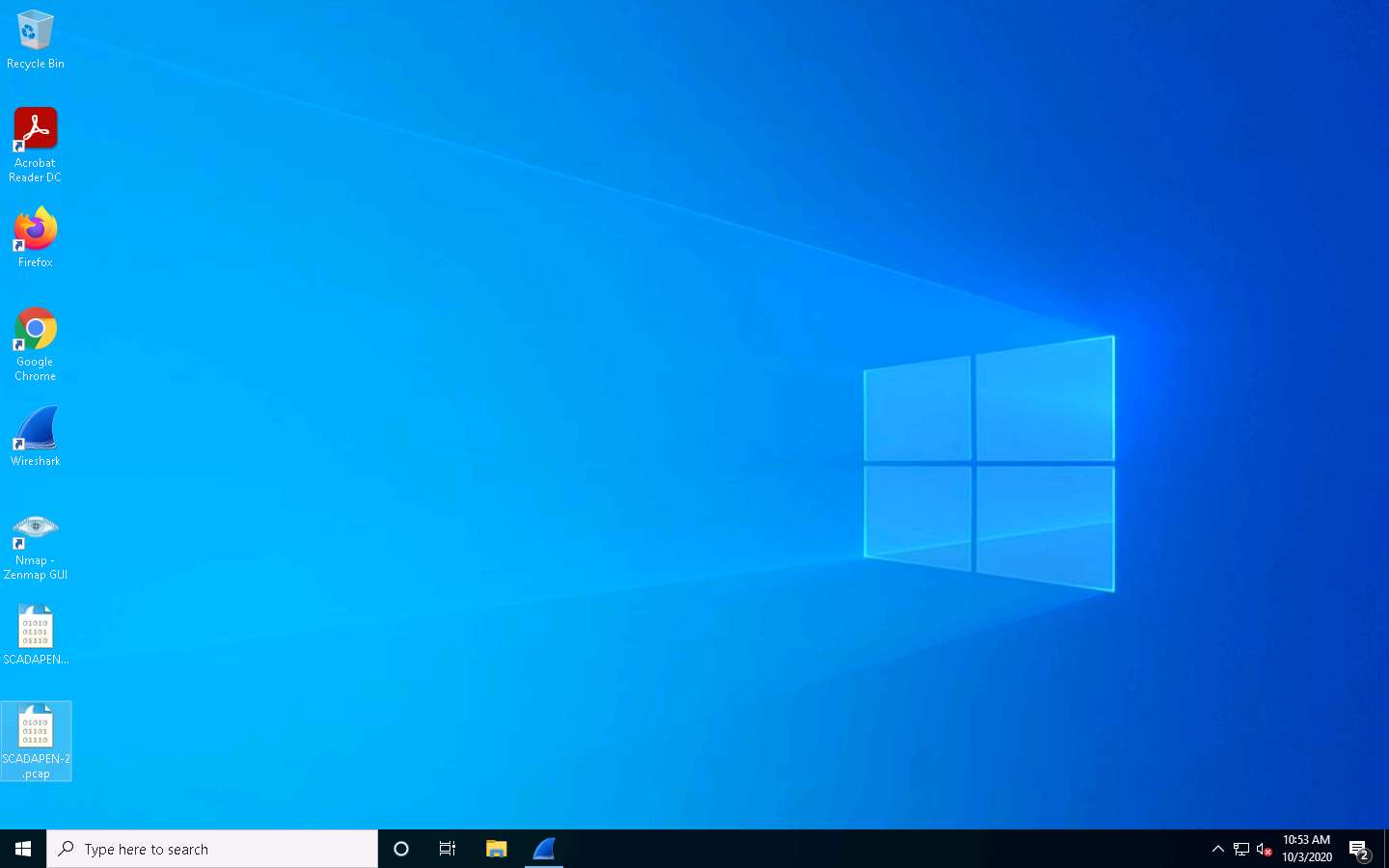
1. When a Modbus slave recognizes a packet, but determines that there is an error in the request, it will return an exception code reply instead of a data reply. The exception reply consists of the slave address or unit number, a copy of the function code with the high bit set, and an exception code. If the function code is 3, for example, the function code in the exception reply will be 0x83. The exception codes will be one of the following codes reflected in the screenshot.



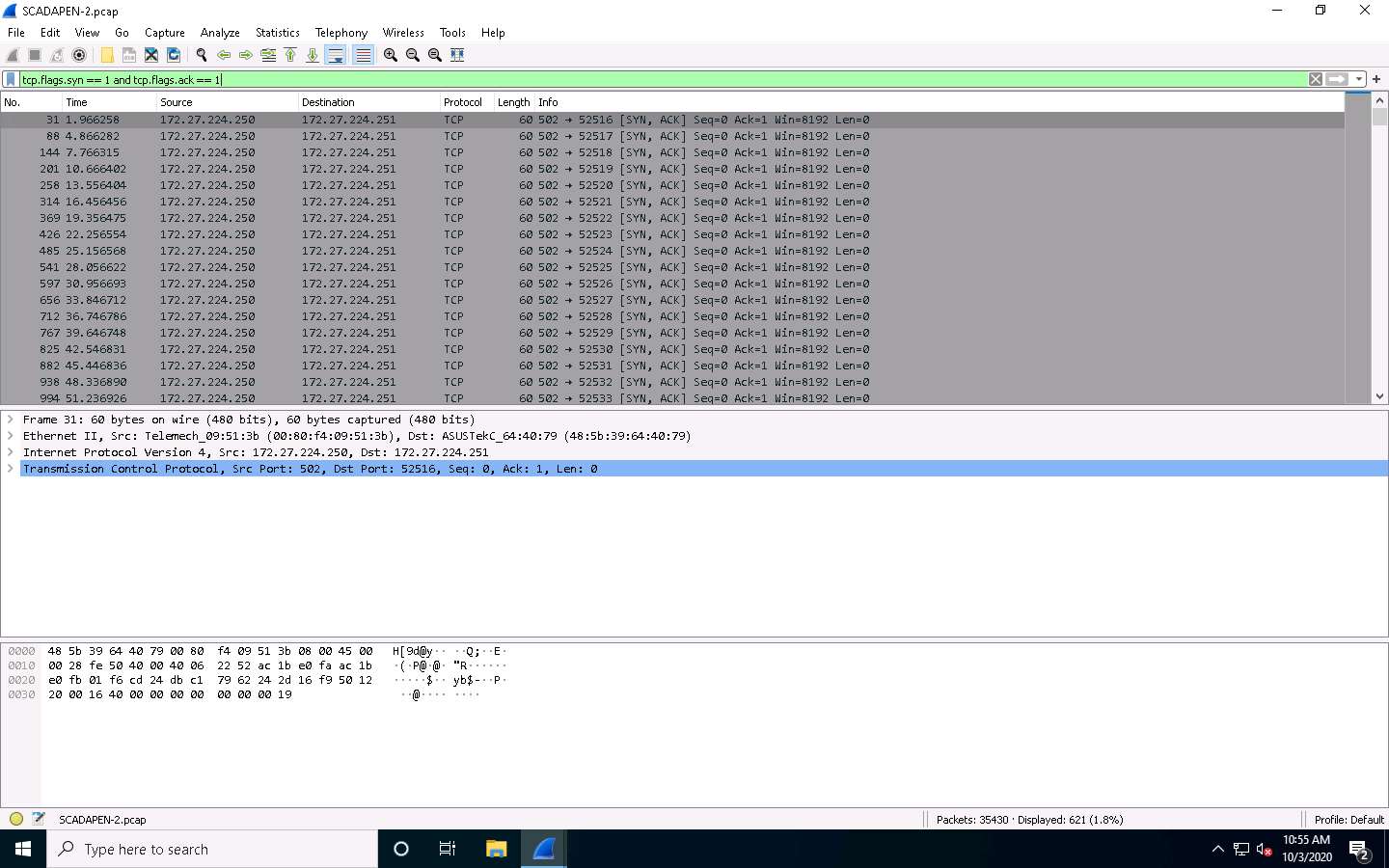
1. A ModBus backend interface can be provided, allowing indirect access to user application objects. Four areas comprise this interface: input discrete, output discrete (coils), input registers, and output registers. Perform a pre-mapping between this interface and the user application data (local issue).



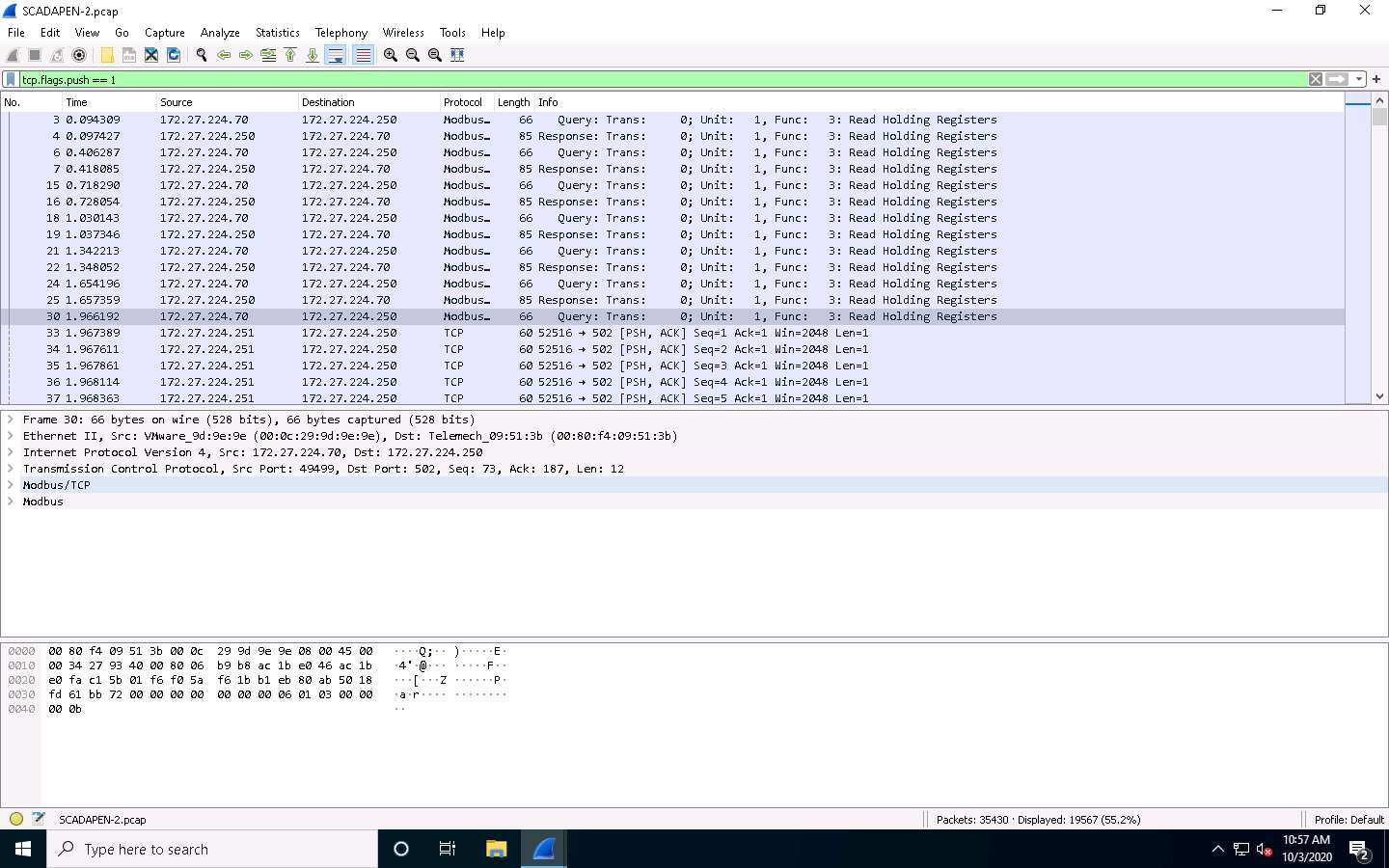
1. Now, navigate to **Desktop** and double-click **SCADAPEN-2.pcap** to open the file in Wireshark.



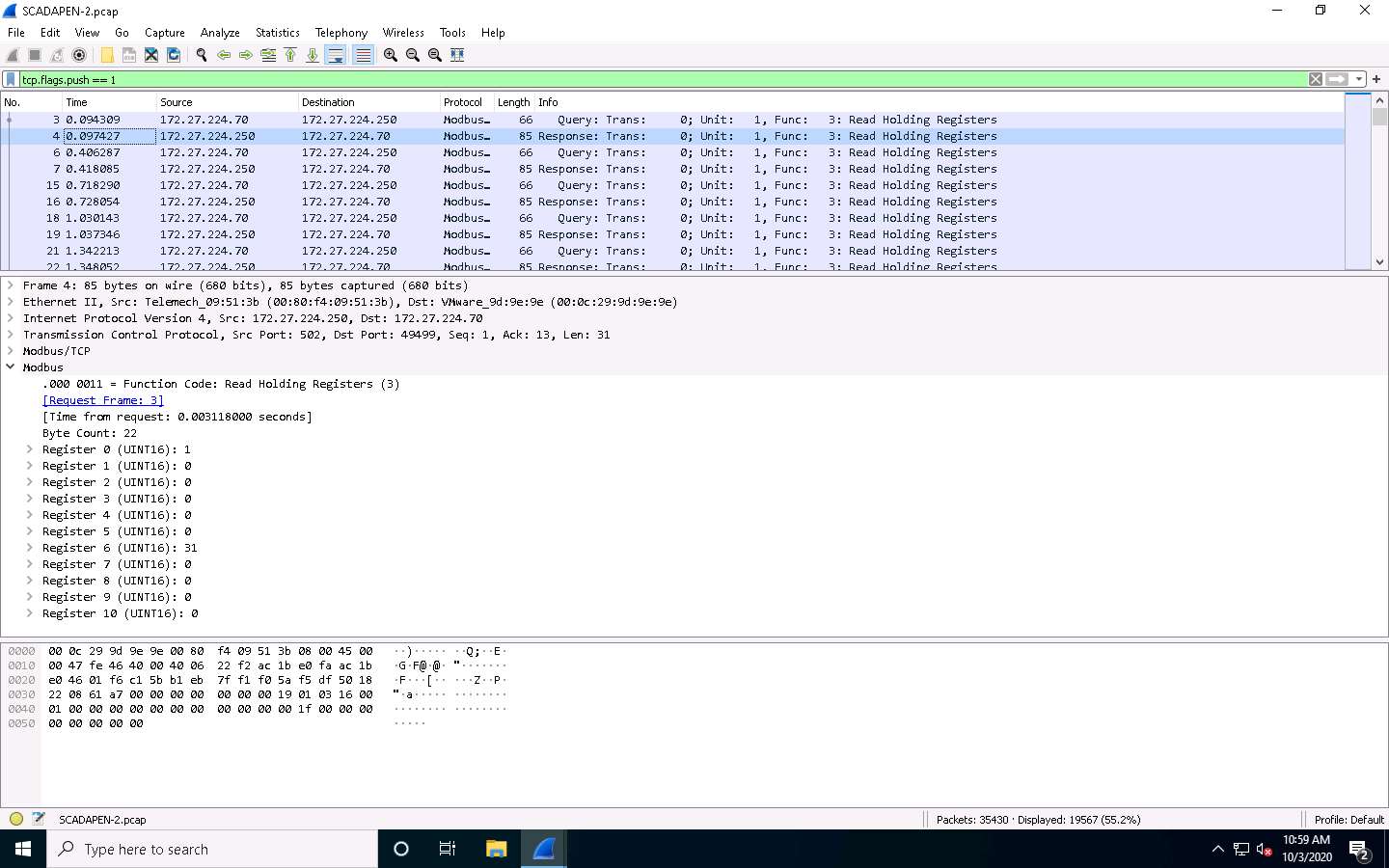
1. The **SCADAPEN-2.pcap** file opens up in the Wireshark window. There is now more **ModBus** traffic again. As done previously, run through the analysis methodology again. The first step of suspicious is not required, since this network capture file is of normal traffic. Move to the next step of open ports. In the filter field type **tcp.flags.syn == 1 and tcp.flags.ack == 1** and press **Enter**.



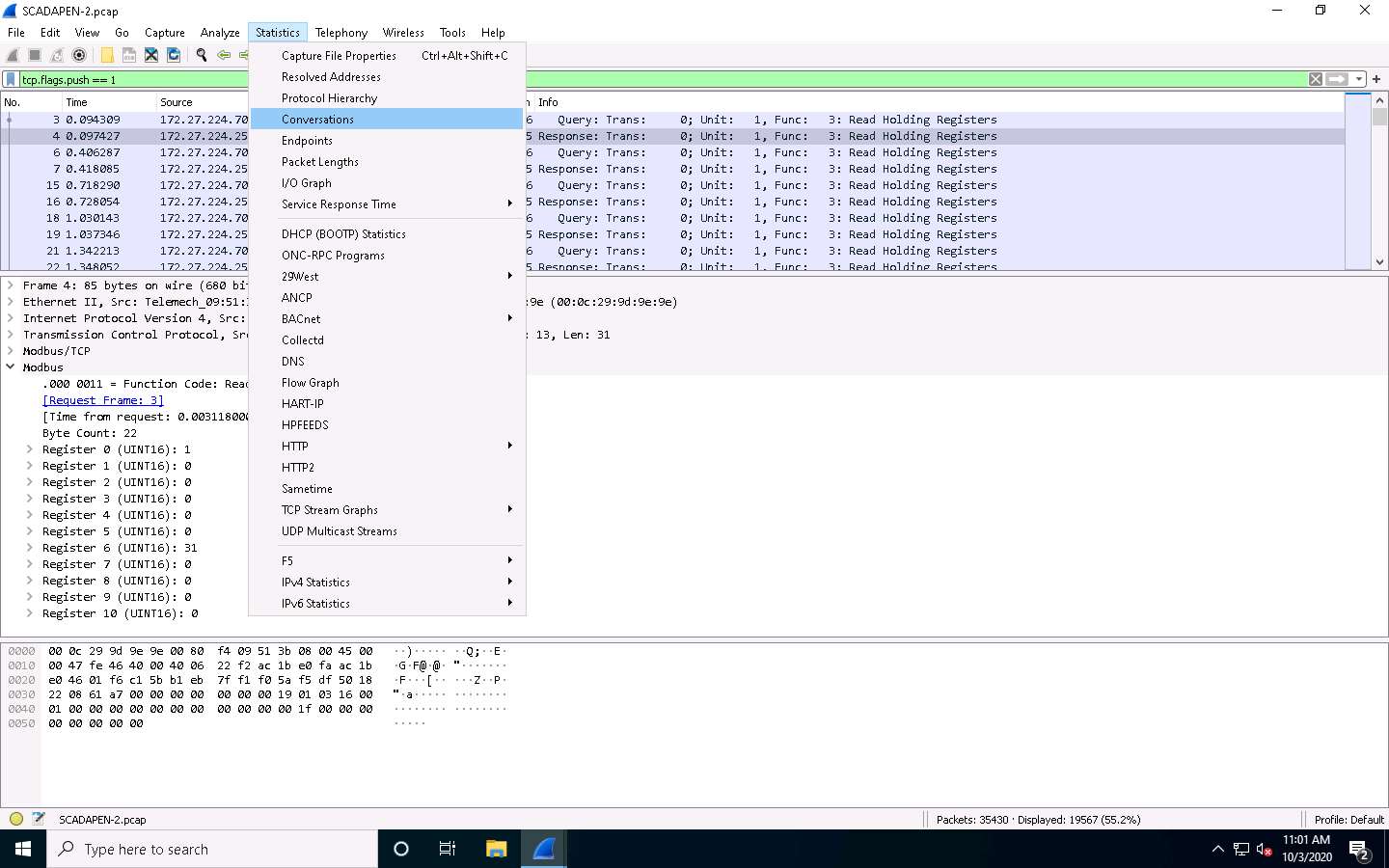
1. As the above screenshot shows, **port 502** is the **ModBus** port, so this is the only port that is in the trace of the **TCP** traffic. Next, explore the data. In the filter field, type **tcp.flags.push == 1** and press **Enter**.

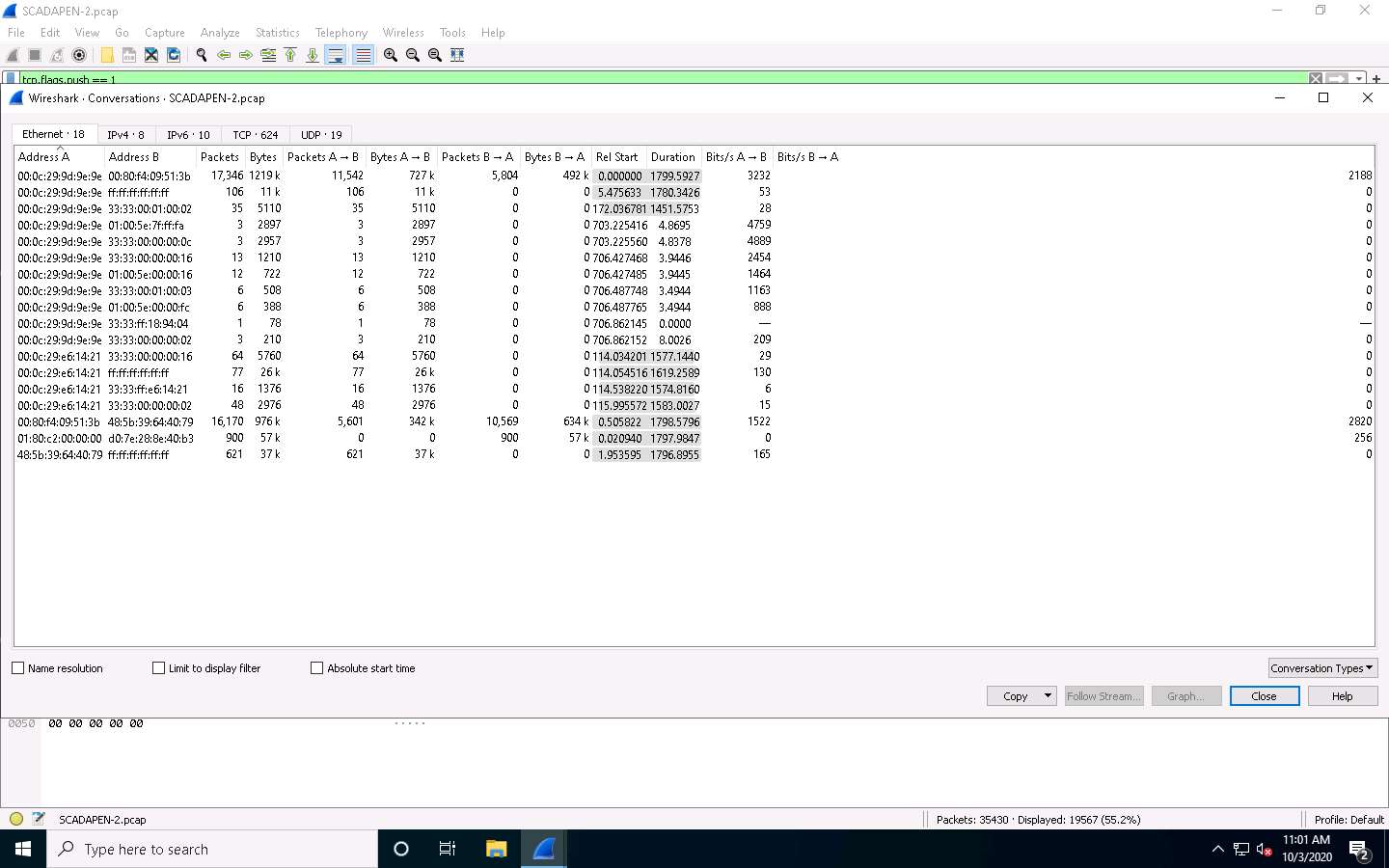


1. There are ample data. The next step is to review the sessions and step through the data. In this case, choose the **packet number 4**. The data are too big, so the better method is to expand the data located in the **middle** window. An example of the response of a read request is shown in the screenshot.
2. You have queried **11 registers** and extracted their data; if you select the register, then the data are revealed.

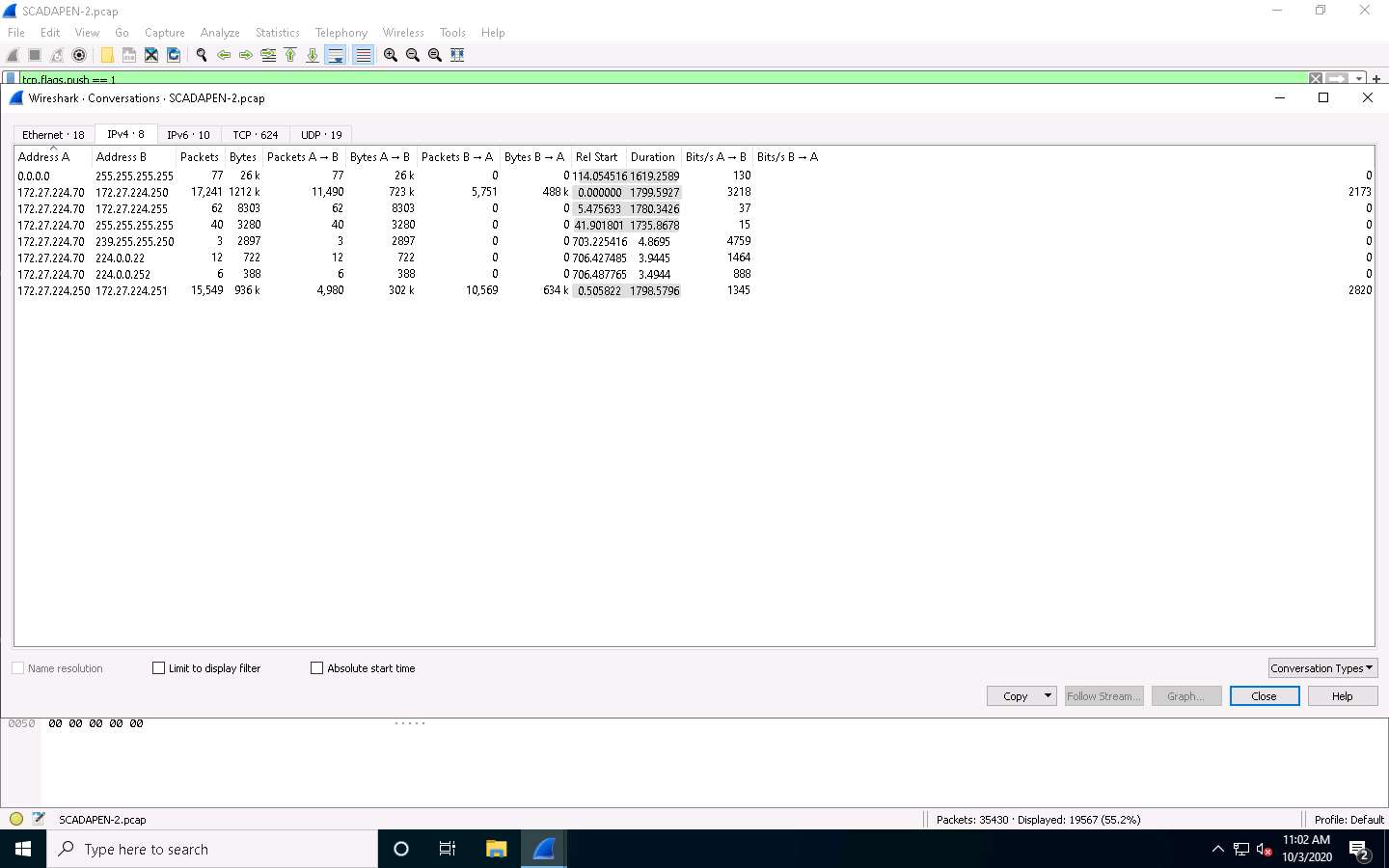


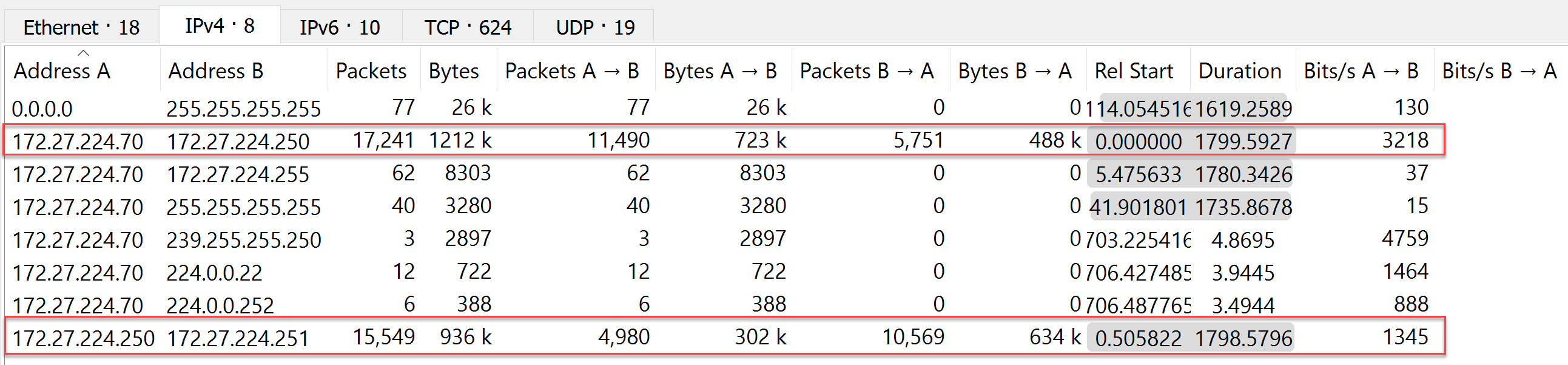
1. Since the capture file is of normal traffic, it is a good location to practice the analysis of the ModBus network traffic. Click on **Statistics** to review a variety of details. Select **Conversations**. The details of all conversations within the network traffic appear.

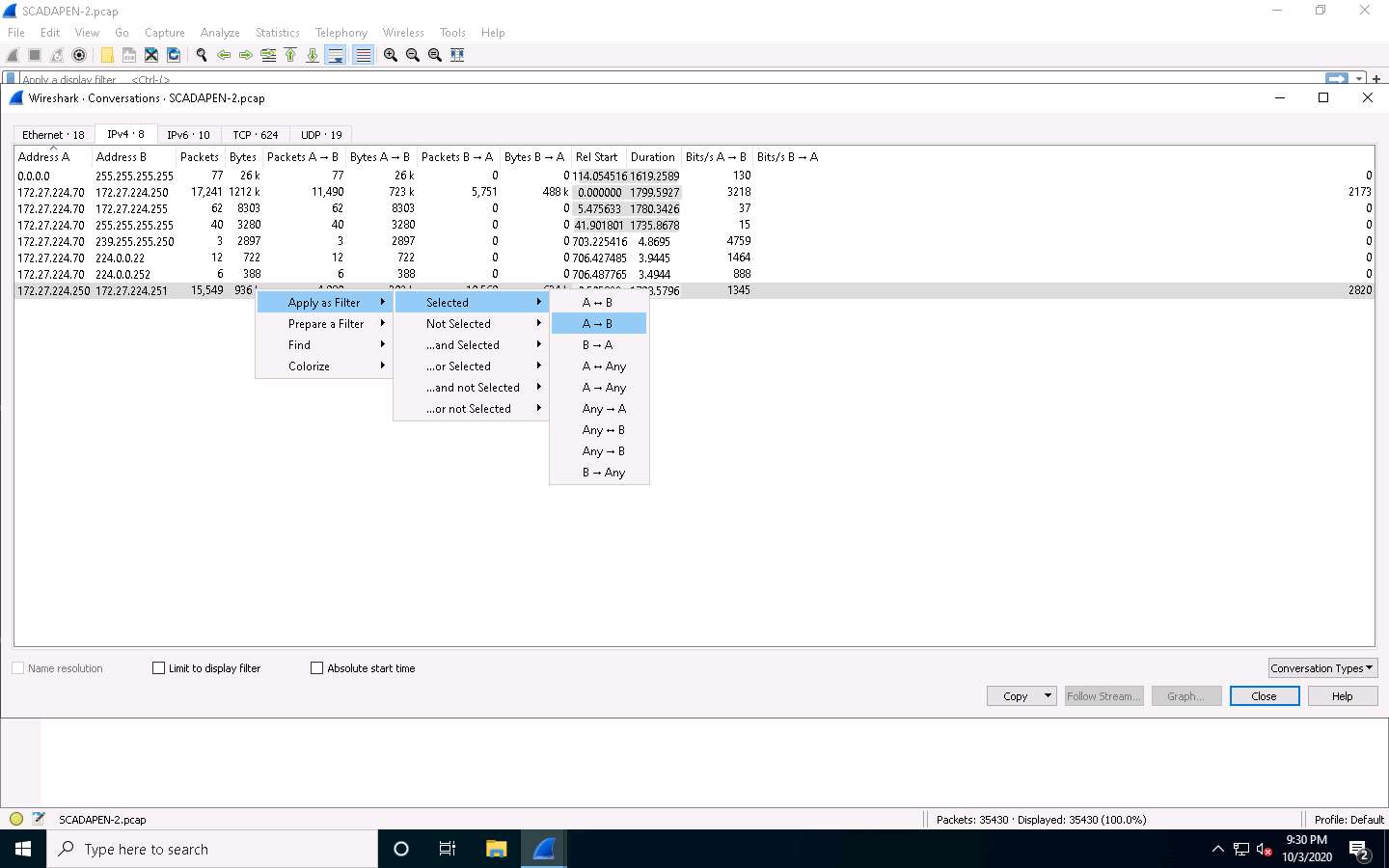




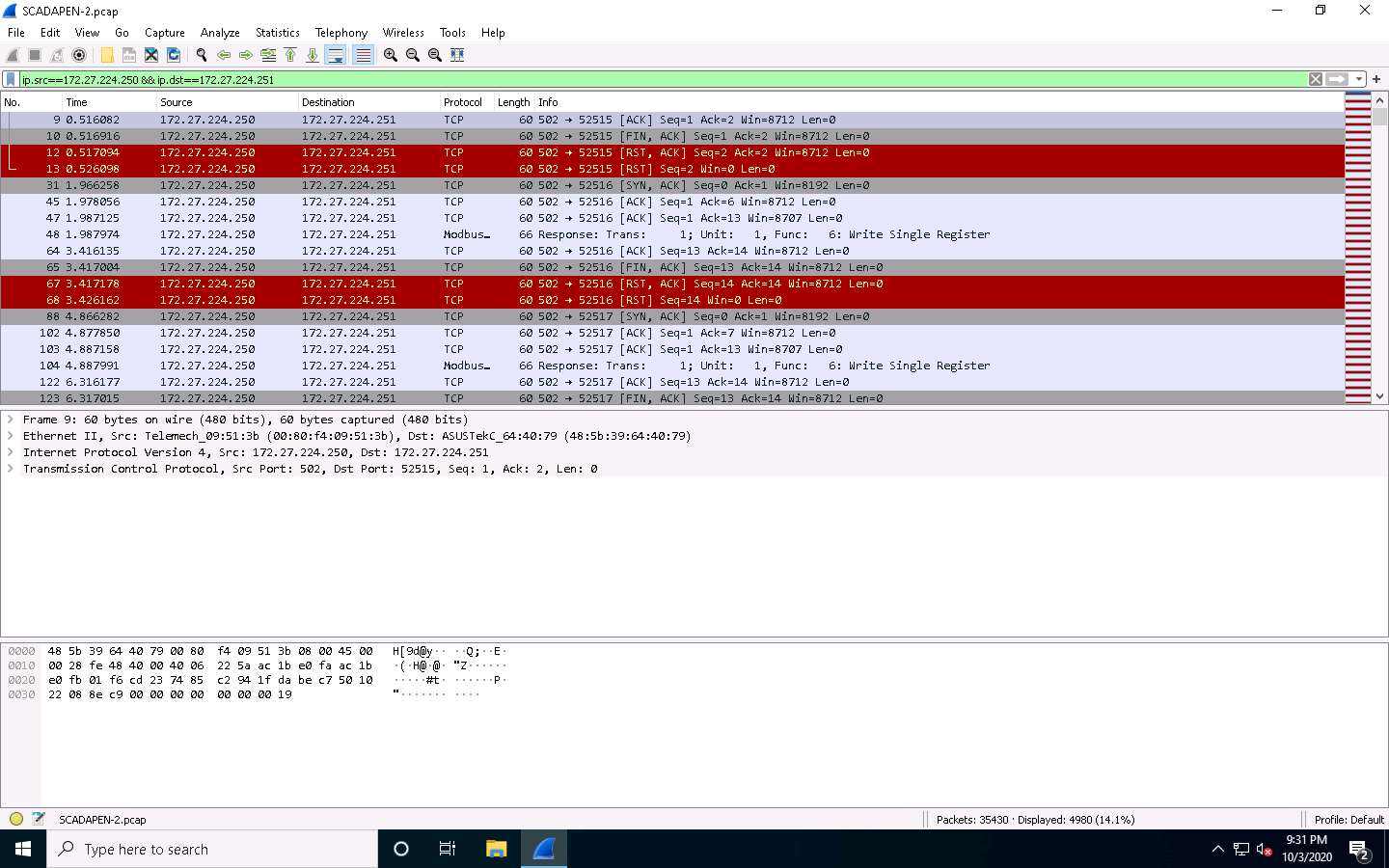
1. Next, explore the **IP details**; click the **IPv4** tab.
2. The two conversations that are shown in the red boxes represent the majority of the network traffic. The process would be to select them, and then right-click on the conversations and select the **Apply as filter | Selected | A -> B**. Next, click on **Close** to filter the traffic displayed. This is an excellent way to isolate specific traffic. Practice the same process you have been using on a smaller subset of data. You may record the data from the trace, and list the IP addresses and the ports that are open on these different IP addresses. This is, in effect, your process to create your target database, so you can identify and prioritize your targets.



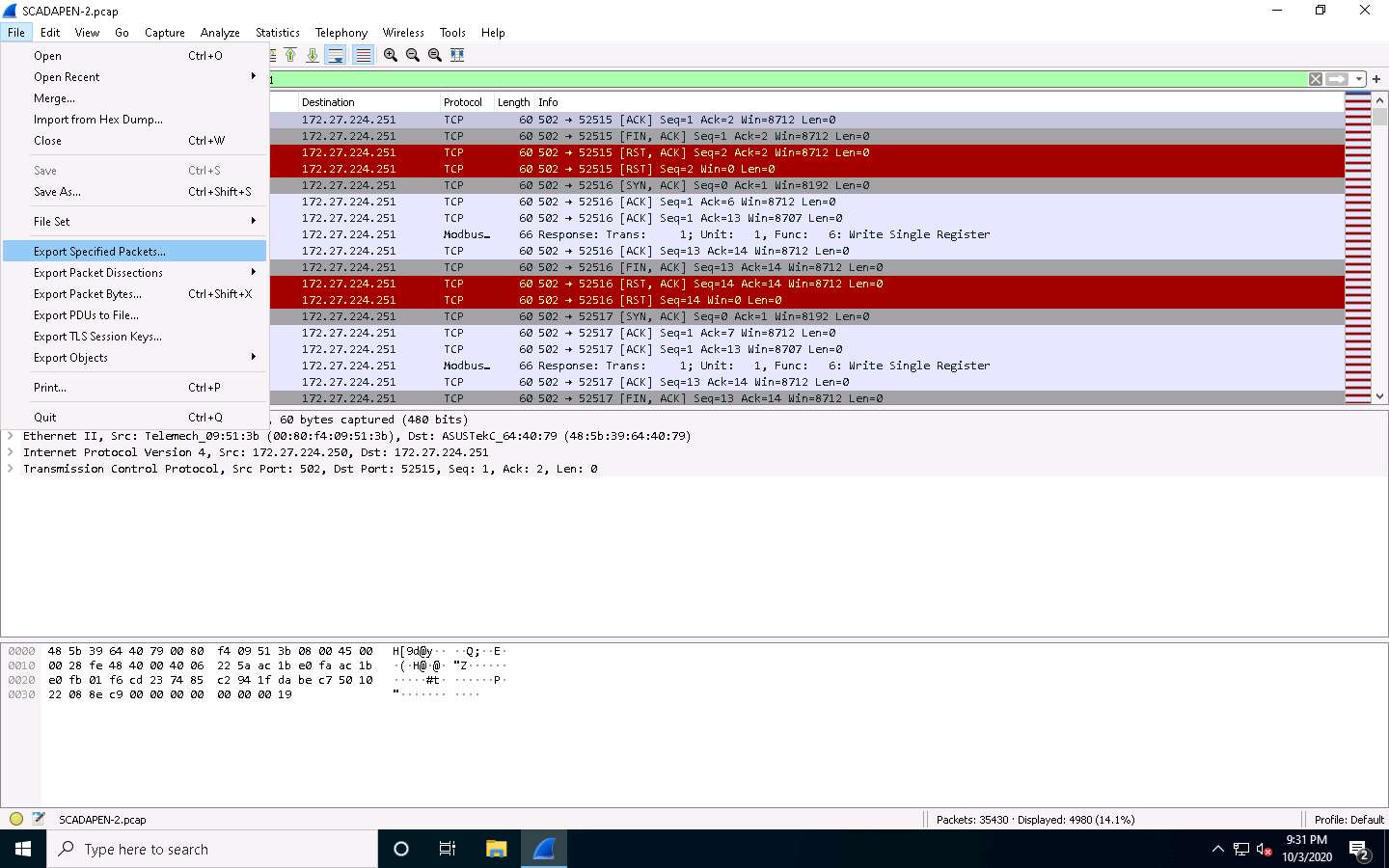




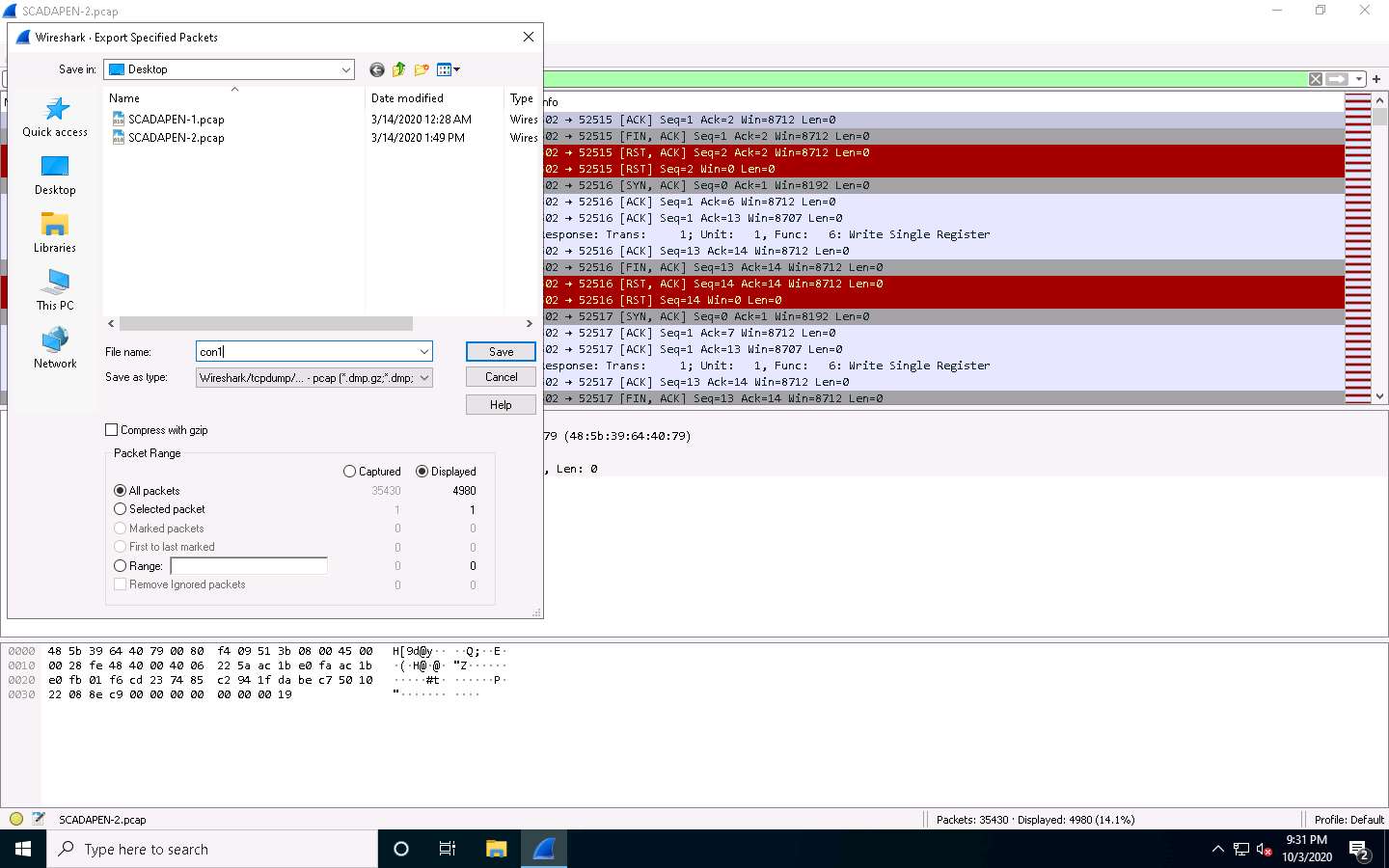
1. The process of applying this filter reduces the amount of packets to perform the process against. Observe at **lower right corner** of the window. The packets have been reduced from 35,430 to 11,490, which is an improvement. This way, you can make these conversations more digestible for analysis.



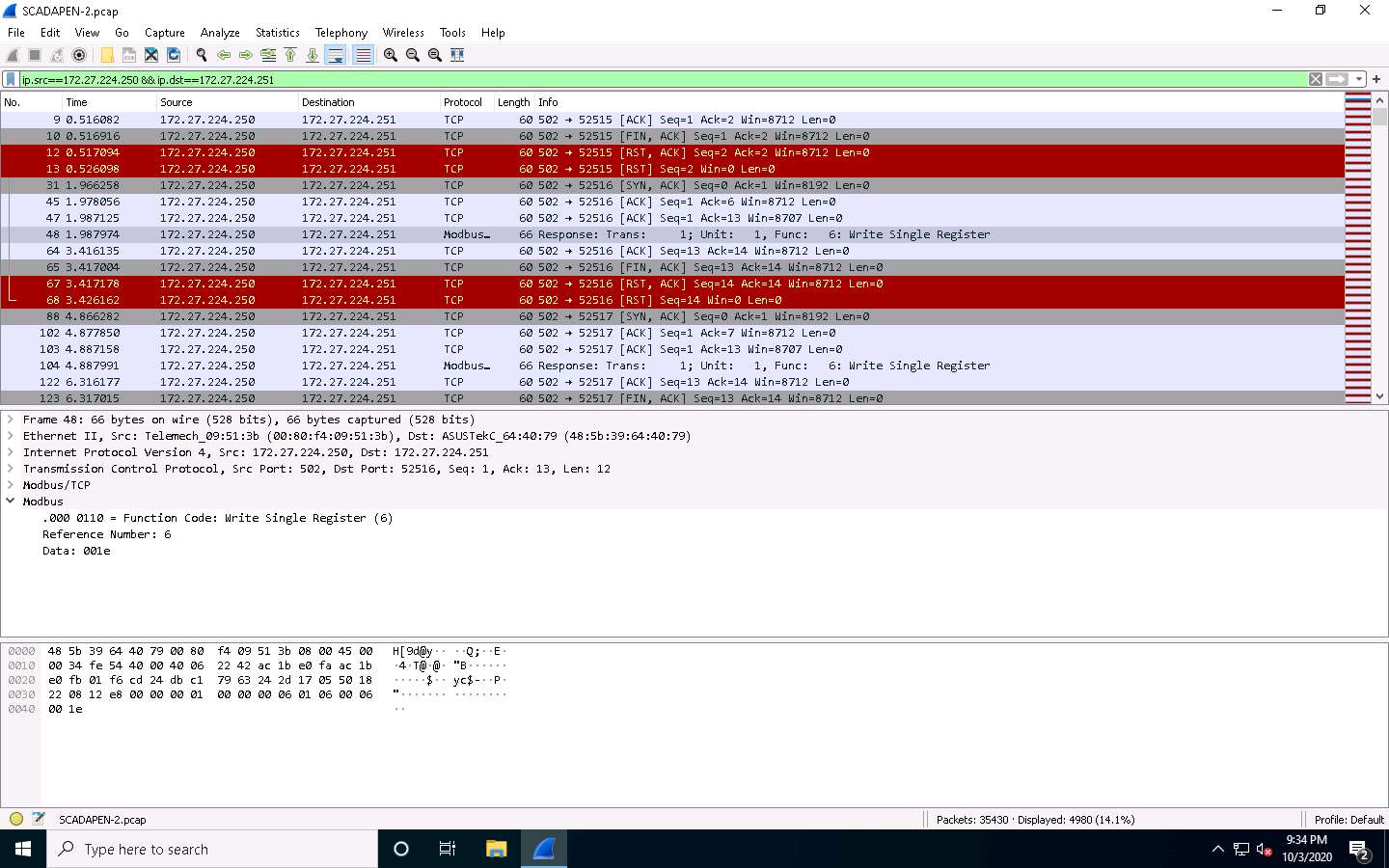
1. You have reduced the number of packets again; this represents the other conversation that contains the majority of the network traffic. Extract this session and save it in its own file. Click **File** | **Export Specified Packets**.



1. Enter the name **con1** for the file name, and then select the options for the output. You now have the specific packets located within the file. **Close** Wireshark and open the file again to verify this and clear the filter.



1. You have successfully extracted the specific packets of interest. Next, focus on specific conversations, and then send them to different teams. Select the packet number **48** and review the information there. This “**writes**” the register, as shown in the screenshot.
2. You have a request that is writing data **001e** to the register. You will not see the response, because, when you captured the specific packets, you only explored one direction. You can either show this by naming the file or capturing both sides of the conversation. When you return to the original file, you can review the conversation again. This time, select the filter for both directions and review the “**write**” statement to check if it was successful.



1. At packet number **29**, the three-way handshake for TCP to port **502** takes place, followed by the **push data**. These data are for “**writing**” the single register. In this case, it is being written to each of the **11** registers.





1. As the screenshot shows, you have the data value of 001e written to the registers. You have now examined the different processes and methods of analyzing ModBus traffic. Close all windows and clean the lab.