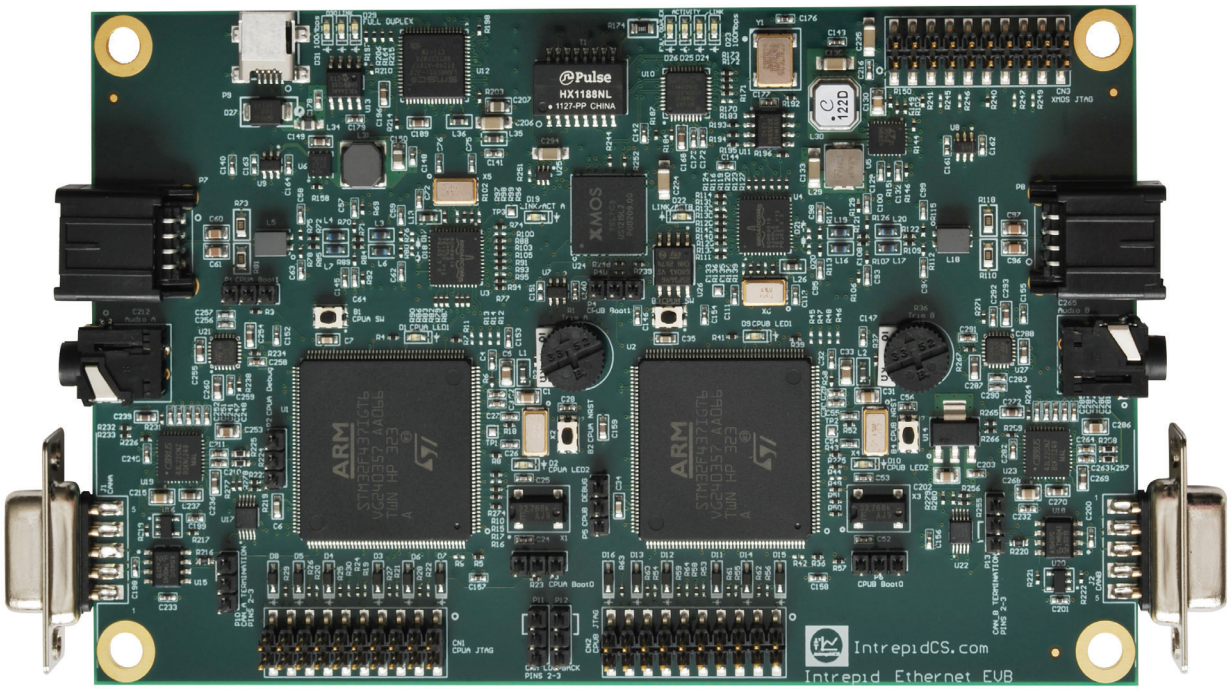
Ethernet EVB

Ethernet Experimentation & Evaluation Board



Lab Manual

Version 1.0 - August 3, 2015

Intrepid Control Systems, Inc.
31601 Research Park Drive Madison Heights, MI 48071 USA
(ph) +1-586-731-7950 (fax) +1-586-731-2274
www.intrepidcs.com

automotive engineering tool alliance
www.aeta-rice.com

Table of Contents

Introduction to the Intrepid Ethernet EVB Lab Manual

Welcome to the Lab Manual for the Intrepid Control Systems (ICS) *Ethernet Experimentation and Evaluation Board*, which for convenience is abbreviated as the *Ethernet EVB* or just *EEVB*. This document contains about two dozen hands-on, detailed tutorials that demonstrate the operation of both the EEVB hardware and Vehicle Spy 3 (*VSpy*) software. Following the step-by-step instructions in these experiments, which we call *labs*, will help you learn about the operation of Automotive Ethernet and the TCP/IP protocols that run on it.

About the Ethernet EVB Lab Manual and User's Guide

As you may already know, the EEVB is supported by a pair of documents rather than just one. The EEVB User's Guide describes the EEVB in detail, outlining its components and features, and providing instructions for installing and setting up the board and Vehicle Spy 3. It also contains troubleshooting information to help you deal with problems that may arise when dealing with the product. The Lab Manual, as mentioned above, contains detailed experiment tutorials that let you really dig into using the board.

To help you make the most of your time, the Lab Manual was written assuming that you have already read through the User's Guide. If for some reason you have come here first, we recommend referencing that document before proceeding. It's not necessary to read everything cover to cover, but please be sure to at least follow the instructions necessary to get your hardware and software installed and working correctly before you try these labs.

There is also a special demo in the User's Guide that helps verify correct operation of your EEVB setup, and also serves as a nice preview of the procedures you'll find in the Lab Manual. As part of this demo you will create a logon name in Vehicle Spy 3, which must be done in order to set up a data directory. The Lab Manual assumes that you have already followed the instructions for this demo and run it, and so that this data directory ("EEVB") already exists. One of the later labs also builds upon this demo.

Organization of the Lab Manual

The introduction you are reading right now describes the Lab Manual itself and includes a few important topics you'll want to understand before proceeding with the experiments themselves. The labs are broken into four *sections*, each with several numbered labs. Here's a summary of the lab sections:

1. **Basic Ethernet Traffic Analysis and Frame Transmission:** Provides an introduction to Vehicle Spy 3 and the Ethernet EVB. You'll learn how to capture and analyze Ethernet traffic, define custom Ethernet messages, and create simple scripts to allow message exchanges between the two EEVB nodes.
2. **Experiments with the TCP/IP Address Resolution Protocol (ARP) Over Ethernet:** A set of labs oriented around ARP, which, due to its simplicity, is an ideal place to begin exploring TCP/IP over Automotive Ethernet. You'll set up an ARP request/reply exchange, and learn how to send and receive messages between the PC and EEVB.

3. **Simulations Using TCP/IP Internet Protocol (IP) and Internet Control Message Protocol (ICMP) Messages:** In this section we move into more advanced experiments that illustrate the operation of IP and ICMP, two essential TCP/IP protocols. This includes simulations of the popular *ping* utility and a message routing scenario.

4. **TCP/IP User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) Data Exchanges (Coming Soon):** Here we explore TCP/IP's two main Transport Layer protocols, looking at data transfer over UDP and connection establishment with TCP.

The Lab Manual contains over 100 figures to help you understand exactly what is going on as you proceed. *Action items*, meaning steps where you need to actively do something, are set apart from the rest of the text for greater visibility using a right-facing pointer character ("►").

How to Get the Most of the Lab Manual

We recommend doing the labs in the order they are listed in the manual. One reason is that to avoid repetition, the fundamentals of using the hardware and software are covered in more detail earlier in the Lab Manual, and we assume you have these basics down by the time you get further into the document. In addition, the lab sections build upon each other logically; for example, UDP and TCP (section 4) both use IP (section 3).

The EEVB also comes with a complimentary copy of ICS's book *Automotive Ethernet - The Definitive Guide* (Figure 1). This 1,100+ page reference provides a thorough description of Automotive Ethernet, as well as a comprehensive description of TCP/IP protocols. Each section of the Lab Manual begins with an overview of the protocols introduced and used within that section. However, of necessity, these are quite brief. If you are new to TCP/IP or want to know more about protocols such as ARP, IP, ICMP, UDP and TCP as you make use of them in this manual, be sure to use the book as a resource!

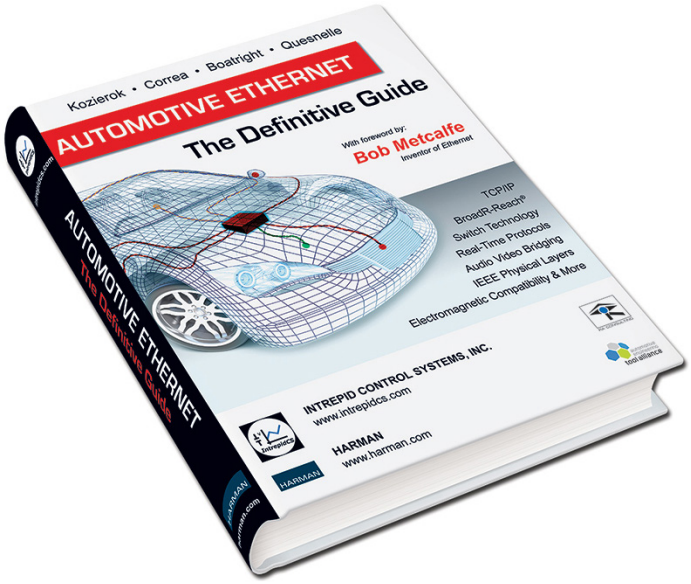


Figure 1: Automotive Ethernet - The Definitive Guide. Intrepid's industry-leading book will help you understand the technologies used in Automotive Ethernet.

Optional RAD-Moon Labs

Some of the experiments in the Lab Manual make use of the ICS RAD-Moon media converter (Figure 2), which connects conventional and BroadR-Reach Ethernet networks. Please try these labs if you have purchased the bundle containing the EEVB and RAD-Moon, or purchased the converter separately; they will show you some of what you can do with the combination of these two Automotive Ethernet devices. If you don't have a RAD-Moon, these optional labs can be readily skipped, as other labs are designed to be independent of them.

Figure 2: Intrepid RAD-Moon. The RAD-Moon is an inexpensive media converter that allows conventional Ethernet and Automotive Ethernet devices and networks to be connected directly to each other.

Online Setup Files

Many of the labs make use of Vehicle Spy 3 setup files (also known as *.VS3 files*) that have been specially prepared in advance for you to save time. We have collected all of these setup files into a single zip file that you can download and unzip into your data directory, so they will be available when needed by specific labs as you progress through the manual. These files should be placed into the data directory corresponding to the logon name you created when doing the demo in the EEVB User's Guide. By default, the logon name should be **EEVB**, and the full path to the data directory **C:\IntrepidCS\Vehicle Spy 3\Data Directory\EEVB**.

Please follow these steps to download the setup files and prepare them for use:

- **Download Setup Zip File:** If you are reading this in electronic form, you can simply click this Web link to download the EEVB zip file to your PC: http://www.intrepidcs.com/

ae/eevb/EEVB_Setup_Files.zip. Otherwise, please enter the address into your browser. Save the file to the desktop or another convenient location.

- **Unzip the File to Your VS3 Data Directory:** Open the folder where you saved the zip file, right-click it, and select Extract All... (Figure 3). Enter the correct data directory folder location, as seen in Figure 4, which assumes the default mentioned above. Then either press Enter or click Extract.

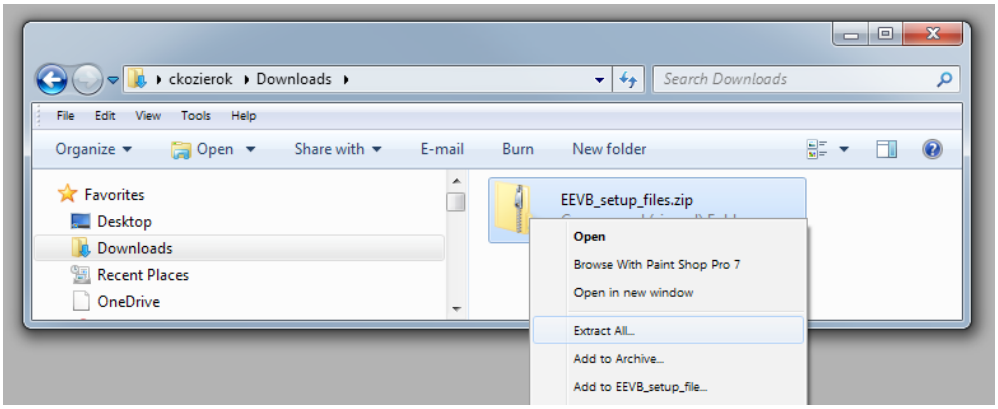


Figure 3: Extracting the EEVB Setup Files. Right-click the downloaded zip file to begin the extraction process.

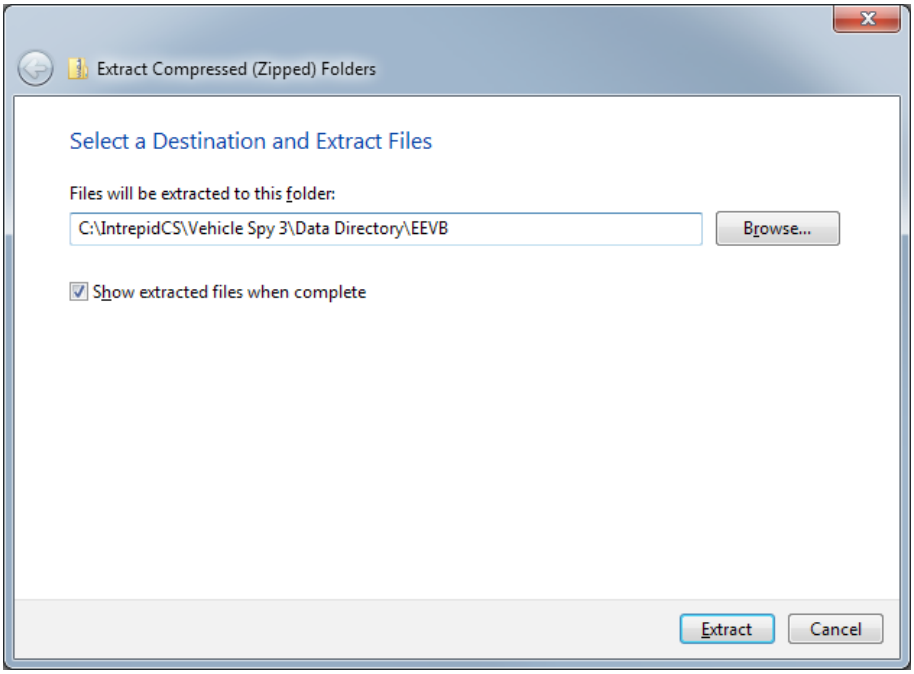


Figure 4: Selecting a Data Directory Target Folder. Select the folder for your data directory, which by default will be *C:\IntrepidCS\Vehicle Spy 3\Data Directory\EEVB*.

Your setup files should now be ready to use.

Online Resources

Intrepid maintains a special area of its website dedicated to the Ethernet EVB, located at http:www.intrepidcs.com/ae/eevb. In addition to the setup files we just discussed, you can also find additional information about the EEVB there, including the latest versions of the EEVB User's Guide and Lab Manual.

Need Help?

If you are experiencing difficulty with your EEVB or Vehicle Spy 3 setup, please first try some of the relevant suggestions in the Troubleshooting section of the User's Guide. If this does not resolve your problem, or if you need help with any of the directions in this Lab Manual, please feel free to contact ICS by phone or email using the following contact information:

- **Phone:** (800) 859-6265 or (586) 731-7950, extension 1.
- **Fax:** (586) 731-2274.
- **Email:** icssupport@intrepidcs.com

Intrepid's normal support hours are from 8 am to 5 pm, Monday to Friday, United States Eastern time. If you contact us outside standard business hours, a member of our support team will get back to you as soon as possible.

Feedback Welcomed!

Comments? Complaints? Compliments? Whatever feedback you may have about the Ethernet EVB Lab Manual, we want to hear it! Feel free to e-mail us any time using the special address eevb@intrepidcs.com, and you'll receive a personalized response, usually within one business day.

Section 1 Basic Ethernet Traffic Analysis and Frame Transmission

With the preliminaries out of the way, we're ready to get down to business! In this first section of the Lab Manual we'll start using Vehicle Spy 3 to work with both conventional Ethernet traffic and messages sent by the Ethernet EVB.

In this section you will do the following:

- Learn how to use Vehicle Spy 3 to monitor regular Ethernet traffic.
- Gain a better understanding of how Ethernet messages are displayed and formatted in VSpy's *Messages View*.
- Examine the layer details in a TCP/IP message.
- Explore some of VSpy's more advanced analysis features.
- Set up transmit and receive messages in Vehicle Spy 3.
- Create function block scripts to be converted to CoreMinis for the EEVB.
- Demonstrate both basic transmission and a transmit/receive exchange using "raw" Ethernet frames.
- Discover how to modify existing setups to enhance their capabilities.
- Learn how to have manual input from the EEVB control script operation.

As mentioned earlier, since this is the first section, more detail will be provided in the steps for each lab here. Some of the specifics, such as how to go online or send a CoreMini to the Ethernet EVB, will be skipped in subsequent sections since by then you will already know how to do them.

> **Note:** This section provides basic instructions on the Vehicle Spy 3 features needed to go through the Lab Manual. For a full description of this powerful and complex software, please refer to the separate Vehicle Spy 3 documentation. Also remember that VSpy has a handy help system you can access at any time by pressing the *F1* key. There are also tutorials you can access directly from the *Logon Screen*.

Lab 1.1 Analyzing Ethernet Traffic Using Vehicle Spy 3

It may seem a bit surprising that we will begin the Ethernet EVB Lab Manual with a tutorial that doesn't actually use the EEVB. There are two reasons why we designed the manual this way. The first is that looking at regular Ethernet traffic is simpler than using the EEVB, and simpler is always better when starting out. The second is that we want to underscore for you the most important features of Vehicle Spy 3 for Ethernet message capture and analysis, which can be used with or without the EEVB.

For this lab, please ensure that the PC you are using is connected to either the Internet or an internal network so that there will be Ethernet traffic to monitor and examine.

Part 1.1A Working with the Ethernet Interfaces List in VSpy

Let's start by getting Vehicle Spy 3 up and running.

- **Start Vehicle Spy 3:** Select Vehicle Spy 3 from the Windows Start Menu, or click the icon on your desktop. The program will start within a couple of seconds, beginning at the *Logon Screen* (Figure 5).

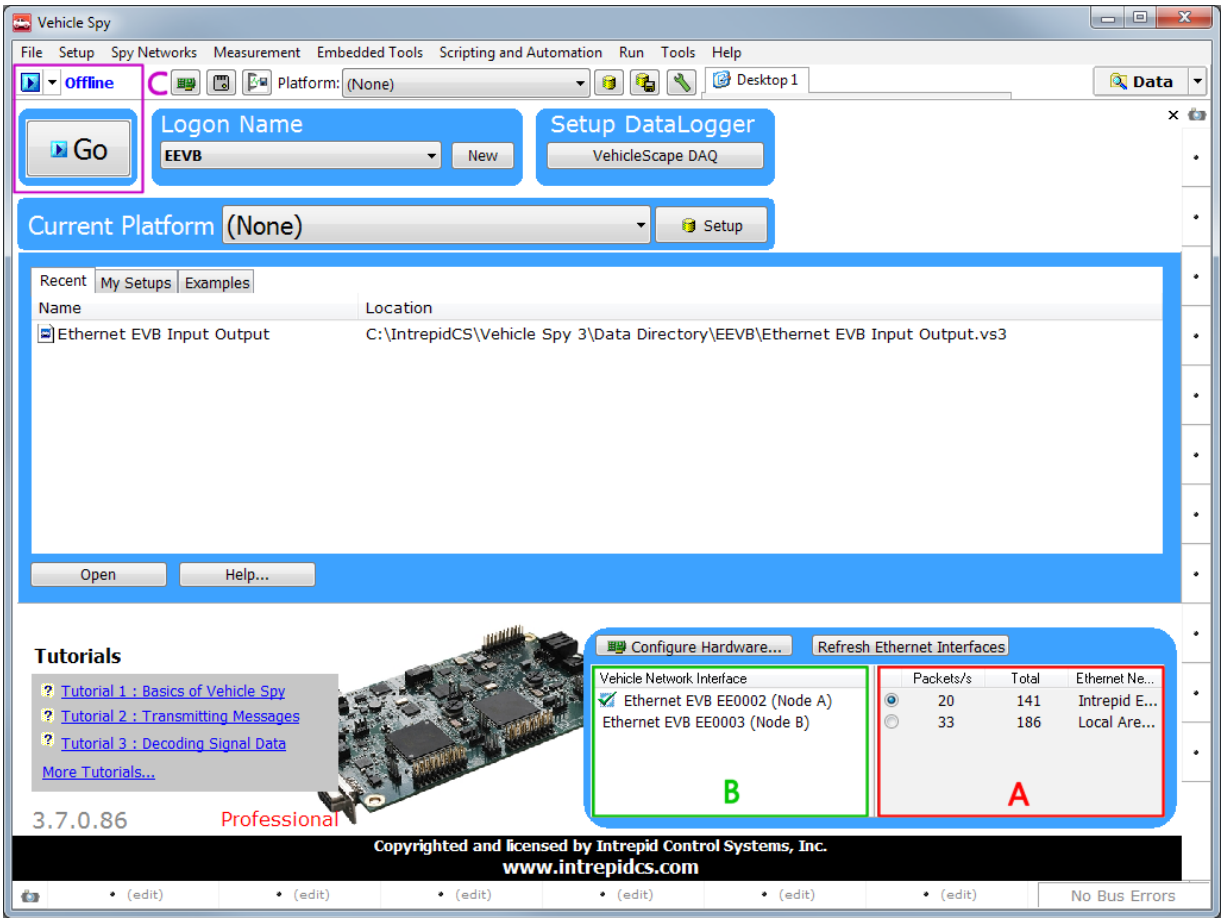


Figure 5: Vehicle Spy 3 Logon Screen. This is a typical view of VSpy after first starting the program. [A: Ethernet interfaces list. B: EEVB hardware nodes. C: Online/offline control buttons.]

On the bottom right of the display you should see a listing of the network connections on the PC upon which VSpy is running (Figure 5, label A; from here out we will simplify such references using the form “Figure 5:A”). Note that although these are called “Ethernet interfaces”, all wired and wireless network connections on the computer will be shown, including Wi-Fi (IEEE 802.11) and Bluetooth as well. Vehicle Spy 3 can examine traffic from all of these sources, treating them the same as it would conventional wired Ethernet links.

> **Note:** If you do not see any network interfaces listed on the logon screen, you probably do not have Ethernet support turned on in Vehicle Spy 3. Please refer to the User’s Guide for instructions on how to enable it.

In the bottom center of the *Logon Screen* you will see a list of Intrepid hardware devices that have been recognized by VSpy (area B in Figure 5, or Figure 5:B). You should have your EEVB connected and powered on, and so should see an image of the board here. You will also see listed two *Ethernet EVB* entries, one per EEVB node, followed by the two nodes’ serial numbers (which begin with “EE”, as seen in Figure 5) and their node designations (“Node A” and “Node B”).

A vertical divider separates the list of Intrepid hardware devices and the list of Ethernet interfaces. If you are using a relatively small display, you may find that the Ethernet interface descriptions are truncated to very short strings, while there is some extra space in the list of EEVB nodes. You can make the interface descriptions a bit easier to read by changing the position of the divider.

- **Resize the Ethernet Network Interface Display:** Hover the mouse over the vertical divider until the resize cursor (↔) appears. Click the divider, then drag it left and release to make the Ethernet interfaces list wider (Figure 6). Of course, you can also slide it to the right if you prefer.

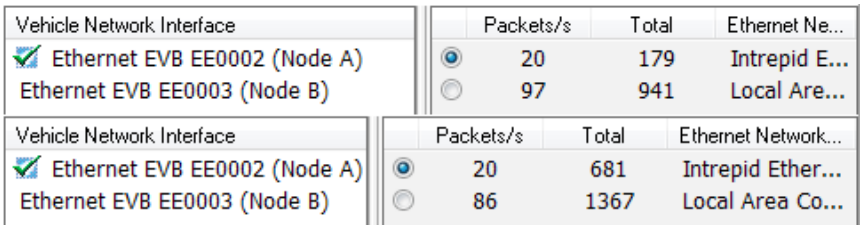


Figure 6: Resizing the Ethernet Interfaces List. This is what the list looks like before (above) and after (below) moving the vertical divider.

Notice the two columns on the left side of the Ethernet interface list. The *Packets/s* column shows approximately how many messages are being received on the interface each second, while *Total* counts all messages seen by Vehicle Spy 3, either since it was started, or since the list was refreshed. The Refresh Ethernet Interfaces button can be used to tell Vehicle Spy 3 to

look for new network connections that have been added since the program ran, or to reset the packet count.

► **Refresh the Ethernet Interfaces List:** Click Refresh Ethernet Interfaces.

Notice that the packet counts of all interfaces are reset to zero, and then they begin counting up again.

Part 1.1B Selecting an Ethernet Interface to Monitor

It is now time to select an Ethernet interface to look at in Vehicle Spy 3. This is done simply by clicking the radio button ( ) to the left of the desired interface. To ensure that we are able to look at lots of frames, we want to pick the most active interface on the list. Note that for the purposes of this exercise we do not want to select the Ethernet EVB itself—even if it is in fact the most active—but rather a regular network connection. This will usually be the one over which the PC connects to the Internet, and many PCs will have only one interface showing traffic at any given time.

► **Select the Most Active Regular Ethernet Interface:** Examine the Ethernet interfaces list and select the (non-EEVB) one with the most packet traffic by clicking its corresponding radio button (Figure 7).

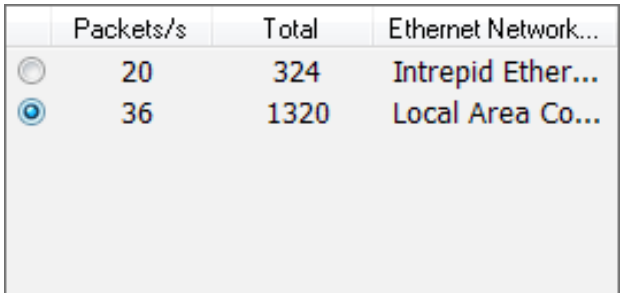


Figure 7: Selecting the Active Standard Ethernet Interface. At the time this screenshot was taken, the standard Ethernet port on this computer was receiving 36 Ethernet packets per second. (Note that the EEVB interface shows 20 packets per second because it is running the Ethernet EVB Input Output demo from the User's Guide, which tells each node to transmit a data message every 100 milliseconds.)

Part 1.1C Going Online with Vehicle Spy 3

It's now time to tell Vehicle Spy 3 to start showing us the traffic on the Ethernet interface we selected.

► **Go Online:** Press the button located near the top left of the *Logon screen*. You can also press the Go button found just below it (Figure 5:C).

Vehicle Spy 3 will automatically switch to *Messages View* and begin displaying the Ethernet traffic found on the selected interface. (If for some reason the view does not change, simply select *Messages* from the *Spy Networks* menu.) You should see a display similar to Figure 8. Notice that the button has changed to the (Stop) button.

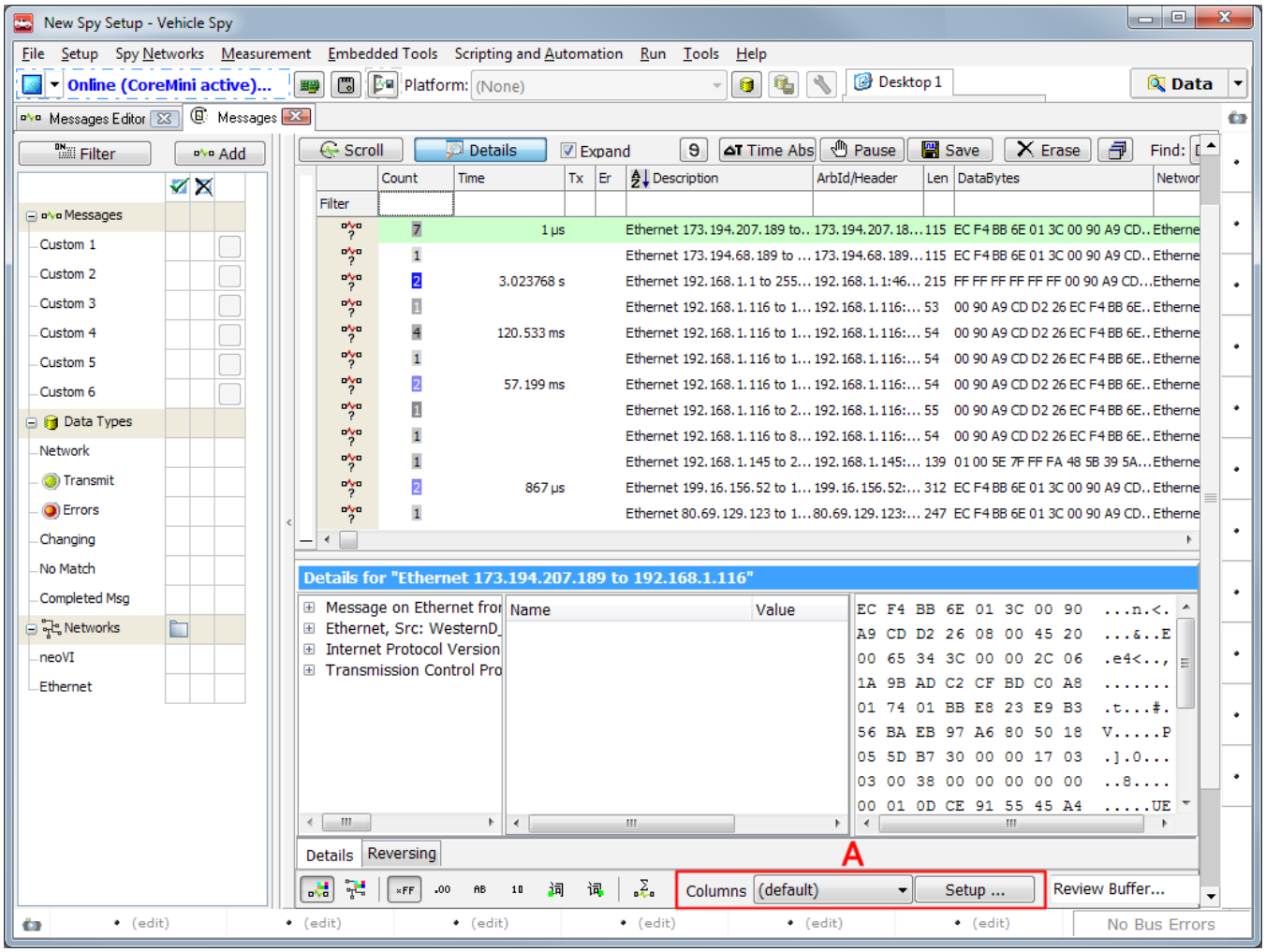


Figure 8: Initial Messages View Display. After going online, you should see a *Messages View* screen similar to this one. [A: Column display selection drop-down box and columns/messages setup button.]

> **Note:** If Vehicle Spy 3 responds with an error such as *Could not find hardware*, this means that the EEVB is not properly connected to the computer. Please connect the board, return to the *Logon Screen* and press Refresh Ethernet Interfaces. If this does not resolve the issue, there may be a configuration problem; please consult the EEVB User's Guide for troubleshooting or support information.

Part 1.1D Switching to Ethernet Column Display and Examining Message View Columns

By default, Vehicle Spy 3 begins with the *Messages View* in its generic display format, which is designed to support many types of vehicle networks. Since we are specifically interested in

Ethernet we will change the display to one that provides more information relevant to Ethernet messages.

- **Change the Messages View Columns Display to Ethernet:** Near the bottom of the *Messages View* (Figure 8:A) you should see Columns, and to the right of it, a drop-down box in which **(default)** is currently selected. Click the box, scroll down, and change the setting to **Ethernet** (Figure 9).

The information in the *Messages View* will immediately change to more Ethernet-specific data.

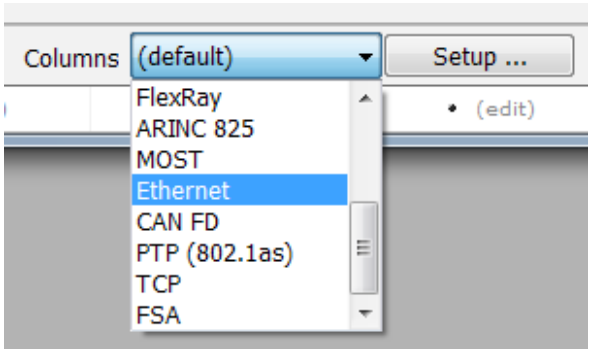


Figure 9: Selecting Ethernet Column Display. This setting will cause *Messages View* to display information pertinent to work with Automotive Ethernet.

Part 1.1E Adjusting the Vehicle Spy 3 Window Size and Column Widths

Depending on the size of your display, you may or may not see all of the columns available; you may also find that some are too narrow to see all of their contents. This is the case with Figure 10, which shows how Figure 8 looks immediately after changing the column display.

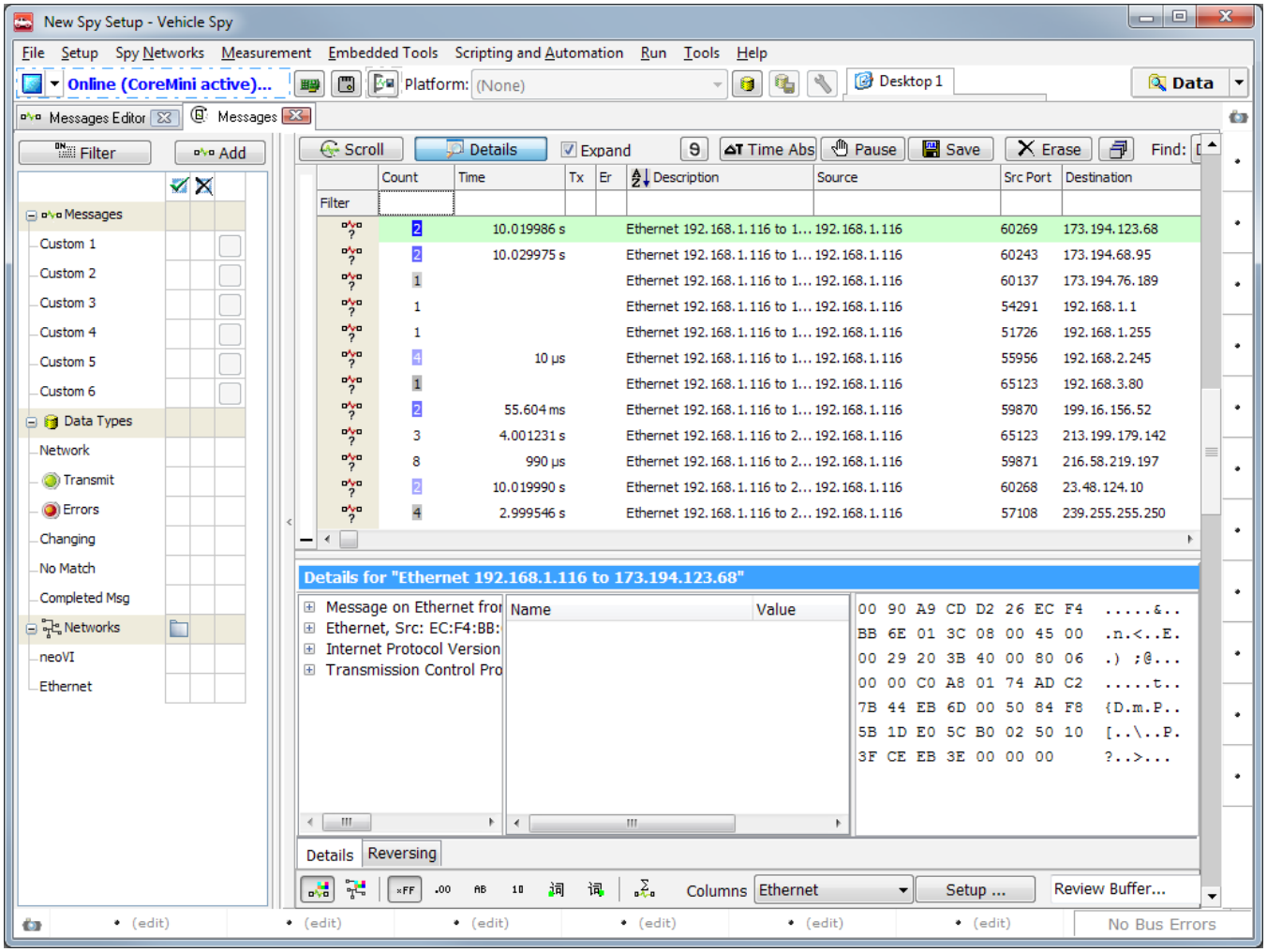


Figure 10: Messages View After Selecting Ethernet Columns. After changing the columns setting we can see several new columns specific to Ethernet messages, including *Source*, *Src Port* and *Destination*. However, this window is not wide enough to show all of the columns we want to see.

If your PC's display is large enough, one obvious option here is to widen the window. Like many complex programs, Vehicle Spy 3 does work best with more graphical "real estate".

- **Enlarge the Vehicle Spy 3 Window:** Drag the right edge of the VSpy program window to the right until it is just large enough to include the columns up to Len (short for "Length").

Figure 11 shows what Figure 10 looks like after this has been done.

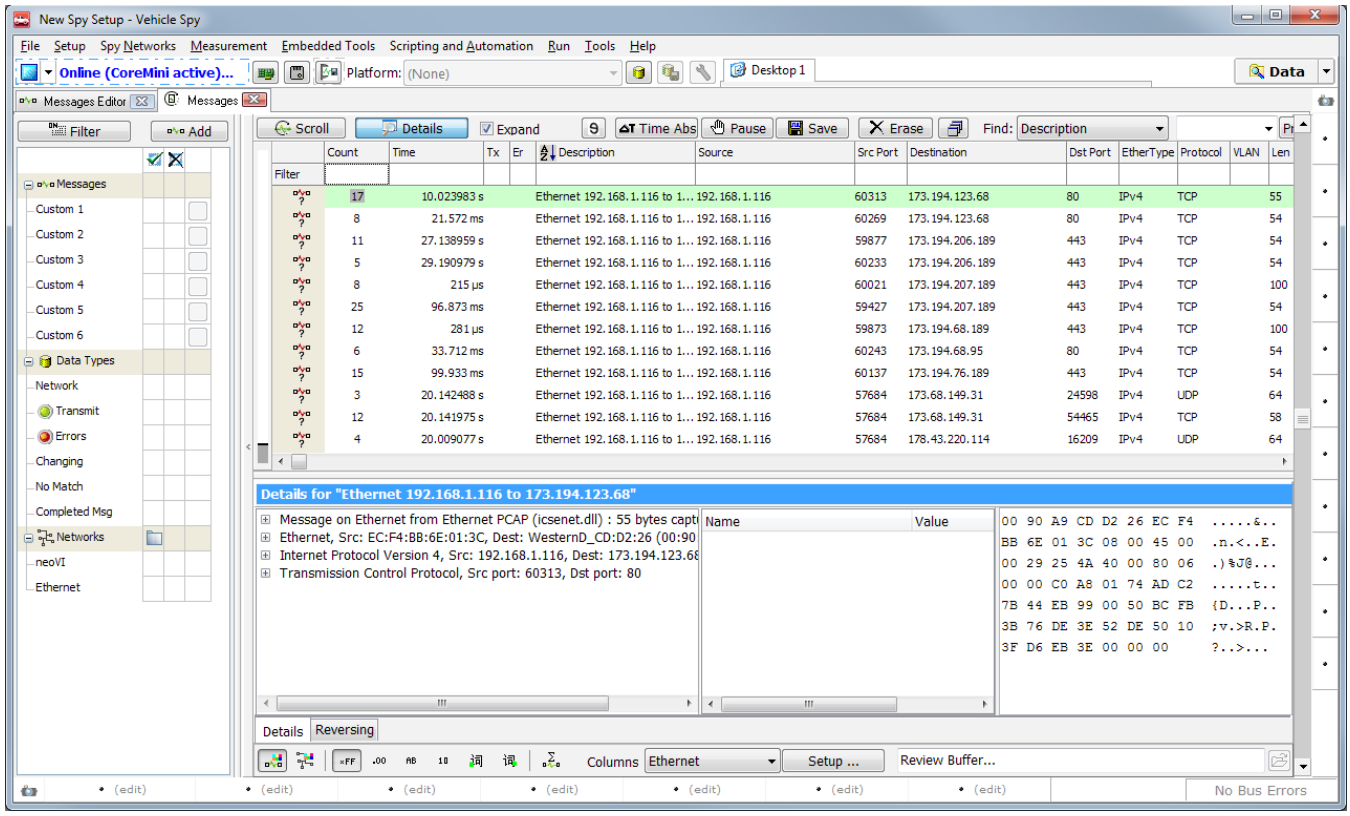


Figure 11: Widened Messages View. After widening the Vehicle Spy 3 window, we can now also see the *Dst Port*, *EtherType*, *Protocol*, *VLAN* and *Len* columns.

Because of the wealth of information provided in the Messages View, it is indeed advantageous to have a large screen when running Vehicle Spy 3. However, this isn't always possible; for example, you may need to use VSpy on a small laptop. In the case of this Lab Manual, we have deliberately kept the window size to 1024x768 both because it is a "least common denominator" and also in order to keep screenshots legible.

Another option in these cases is to resize the columns. This can be done in the same way as we did with the Ethernet interfaces list divider on the *Logon Screen*. As an example, let's tinker with the width of one column.

- **Make the Count Column Narrower:** Hover over the divider in the header row to the right of Count and drag to the left until the divider overlaps the letter "t".

The name will change to *Co...*, with the periods indicating that the column is now too narrow to display its full name.

- **Make the Count Column Wider:** Now drag the divider back to the right again, and Count will reappear.

Of course, you can also do a little of both, making the VSpy window larger and also adjusting the column widths until you find what is most practical and appealing to you. Notice that the default widths of many columns are greater than they need to be for the data they contain, so they can be easily narrowed.

In Figure 12 we have shrunk the Vehicle Spy 3 window back to the original size we used in Figure 10, but reduced the widths of most of the columns so that we can still view most of the same data we had in the more expansive Figure 11. In this manner we can fit all of the most important Ethernet fields in the view.

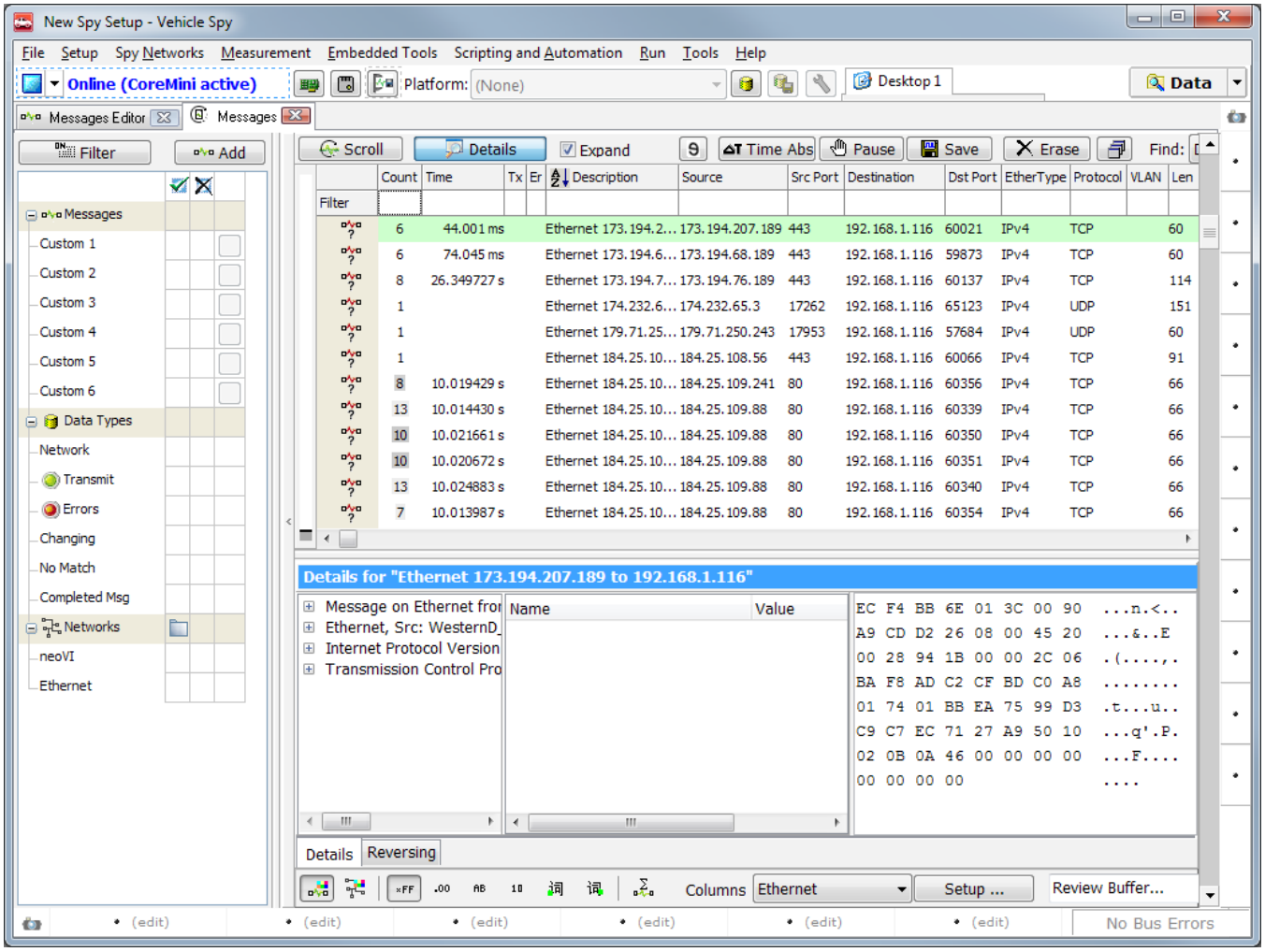


Figure 12: Messages View with Resized Columns. Here we have restored the Vehicle Spy 3 window to its original size, but changed the column widths to still show all of the key Ethernet data.

Part 1.1F Saving the Custom Column Setup

We have now tailored the *Messages View* to suit our preferences. However, if we close the program and then restart it from scratch, it will reset all of the columns to their defaults. To preserve our changes, we need to save them in a new setup file. But first we need to go offline, as setups can't be loaded or saved while online.

- **Go Offline:** Press the button to go offline.
- **Create a New Setup File:** Select Save As from the *File* menu. When the dialog box appears, enter 1.1 Custom Column Setup.

That's it. But now, let's test that this does in fact work.

- **Close Vehicle Spy 3:** Close Vehicle Spy 3 by selecting *Exit* from the *File* menu. You can also press the *Alt+F4* key combination.
- **Start Vehicle Spy 3:** Select Vehicle Spy 3 from the Windows Start Menu, or click the icon on your desktop, as before.
- **Select the Most Active Regular Ethernet Interface:** Select the same interface that you used earlier.
- **Go Online:** Press .

You will be back where you were at the start of Part 1.1D, with the generic *Messages View* display. Now let's load our setup and see what changes.

For convenience, Vehicle Spy 3 remembers the last several setup files you have used and lists them under the *Recent* tab on the *Logon Screen*. You should see *1.1 Custom Column Setup* here since you just created it.

- **Go Offline:** Press .
- **Return to the Logon Screen:** Select *Logon* from the *File* menu.
- **Load the Custom Column Setup:** Double-click 1.1 Custom Column Setup, which should be on the Recent tab. If for whatever reason you don't see it there, look under the My Setups tab.

Vehicle Spy 3 will automatically switch you to the *Messages View*. You should see that your adjusted columns have been restored.

- **Go Online:** Press .

Messages should begin appearing in a display similar to that of Figure 12.

- **Go Offline:** Press .

Part 1.1G Changing the Column Setup

Squishing the column widths allowed us to squeeze all of the columns into a relatively small program window. That said, you can see that the display is pretty crowded. In particular, we've really cut the size of the *Description* field down to the point where it's heavily truncated, and not that useful for longer message summaries.

There's another option available to us for adjusting the *Messages View* to meet our needs: customizing the column setup. The columns you see when you first run Vehicle Spy 3 and select the Ethernet column set are simply defaults. There may be some columns in there that you don't care about, and so you can tell VSpy not to show them, providing more space for the data you do want to see.

We can make the changes we are after using a special setup dialog box accessible from the *Messages View*. The same setup area is used both to change the columns in the message display (what we are doing now) and to adjust the size of the message history buffer (which we will do in Lab 1.2).

- **Enter Scrolling Message History Setup:** Press the Setup ... button, which can be found at the bottom of the display near where you changed the column view (Figure 8:A).

Since we told VSpy to show the Ethernet column setup in *Messages View*, the dialog box shows the column setup for Ethernet. On the left is a list of all available columns, and on the right, the ones presently being shown for Ethernet messages.

Let's say we know we will never be using virtual LANs (VLANs) on our network, so we want to remove that column from the display to save space. We also don't need to know how long messages are, don't care about elapsed time between messages, and don't need the *Network* column either—it will always be **Ethernet**.

- **Delete VLAN Column:** Click VLAN to highlight the VLAN column, and then press − (found below the list) to remove it from the column setup (Figure 13). The Messages View will be updated immediately.
- **Delete Time, Len and Network Columns:** Repeat the process above for the Time, Len and Network entries.
- **Save Changes:** Click Close to close the dialog box.

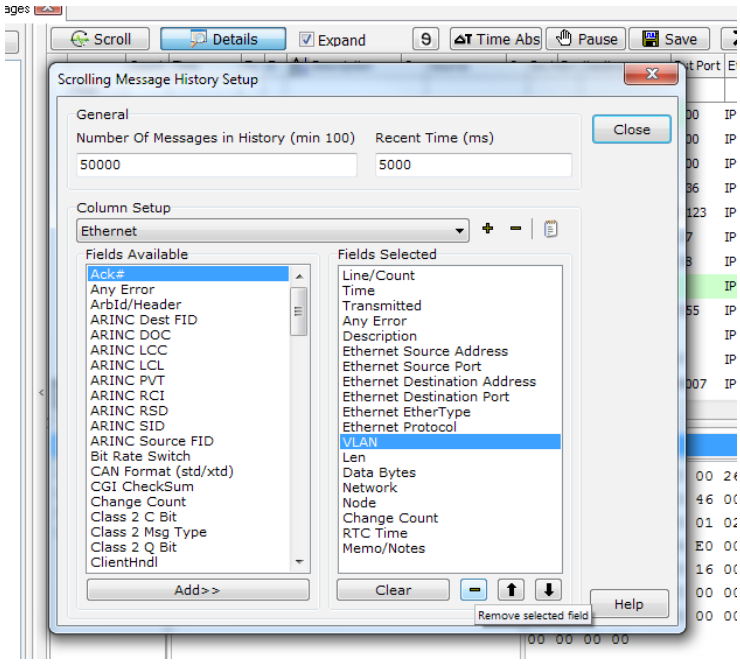


Figure 13: Deleting a Column from the Messages View Display.

We now have more room for the remaining columns, and can increase the size of the *Description* field. The result can be seen in Figure 14.

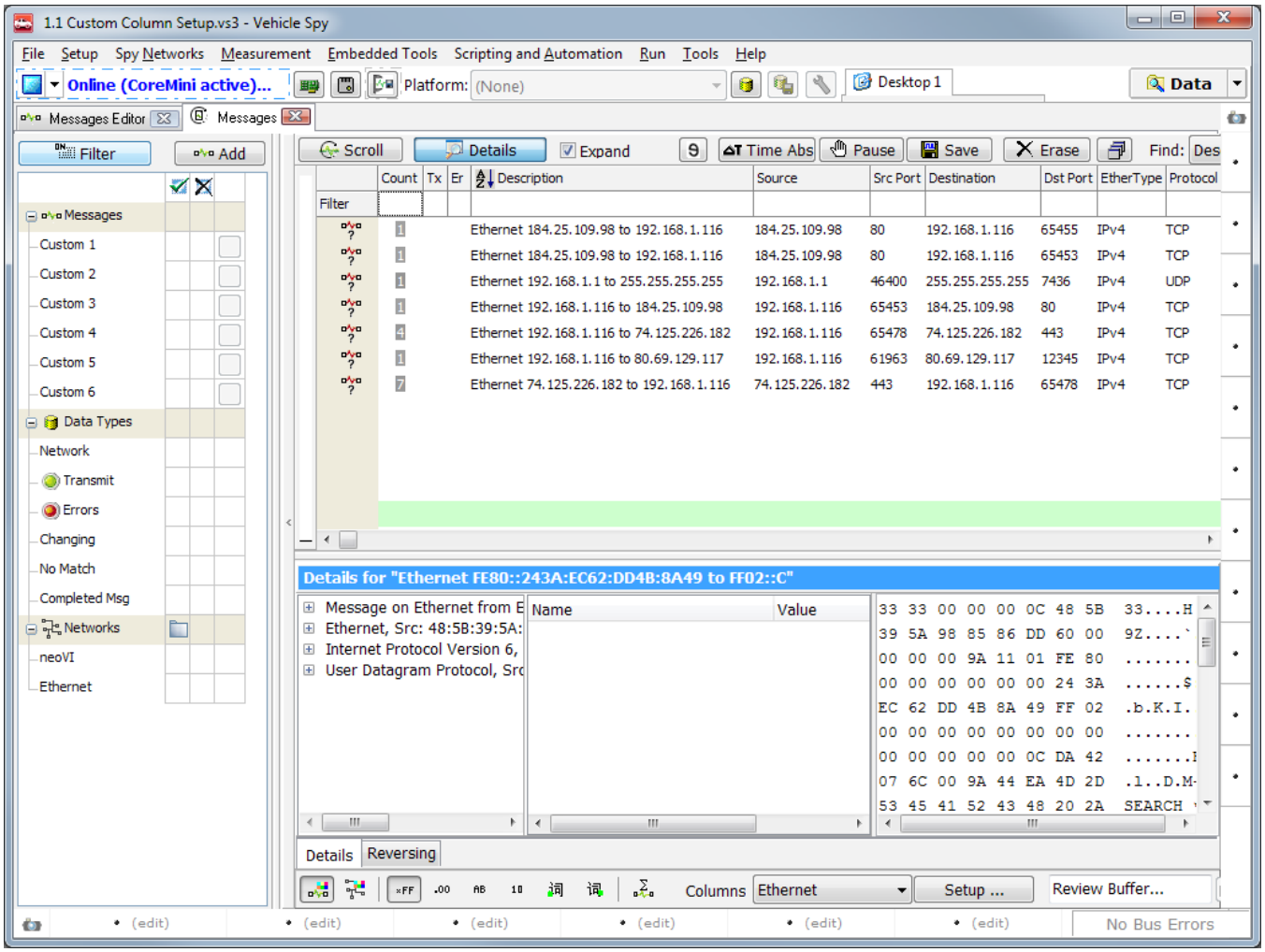


Figure 14: Messages View with Customized Columns. Now we have removed some of the columns, allowing us to make more room for the remaining columns we want without needing to enlarge the Vehicle Spy 3 window.

That looks much better. There's only one problem: we actually *do* want those columns that we just removed! Unfortunately, this tug of war between window size, number of columns and column width is just one of those trade-offs that exists in the world of computing. In this case we want to keep the columns, but also leave the screen relatively narrow so screenshots are readable, so we'll have to live with a skinny Description column.

The easiest way to get back the columns we deleted is simply to restore the setup we saved just before we changed the column setup. (A good illustration of why saving frequently is a good idea!)

- **Return to the Logon Screen:** Select Logon from the *File* menu.
- **Load the Custom Column Setup:** Double-click 1.1 Custom Column Setup, which should be on the Recent tab.

Naturally, on your own computer you should feel free to adjust the window size, column selections and column widths to suit your own needs, then save the setup if you wish.

Part 1.1H Understanding Ethernet-Specific Columns in the Messages View

There are quite a few columns in the *Messages View*. This can seem a bit daunting at first, but remember that these fields represent a substantial amount of information that you will eventually appreciate as you work with Ethernet frames. To keep things simple for now, let's start by looking at just a few of the more interesting columns, especially when working with Ethernet:

- **Count:** Vehicle Spy 3 aggregates messages of the same type so you can more easily analyze the nature of traffic on the network. This column shows the number of each message type that has been received since going online.
- **Time:** The approximate elapsed time between receipt of the most recent message of this type and the one before it.
- **Description:** A summary of the message type. For Ethernet messages these of course begin with **Ethernet** and then generally contain the source and destination addresses. These will be either MAC addresses for plain Ethernet or AVB frames, or IP addresses for TCP/IP datagrams.
- **Source and Destination:** The sender and receiver of the message, which again will be either MAC addresses or IP addresses.
- **Src Port and Dst Port:** The source and destination port numbers for UDP and TCP messages.
- **EtherType:** The interpreted value of the two-byte *EtherType* field in the Ethernet header, indicating the type of data being carried in the frame. Typical values are **IPv4**, **IPv6** or **ARP**.
- **Protocol:** The interpreted value of the IPv4 *Protocol* field or IPv6 *Next Header* field, specifying the higher-level protocol message being carried in an IPv4/IPv6 message. This will normally be **UDP** or **TCP**.

> **Note:** Concepts such as IP addresses and UDP/TCP ports are explained in later sections where we use these protocols. You can also find more information on them in your Automotive Ethernet book.

Lab 1.2 Making Use of Advanced Vehicle Spy 3 Analysis Functionality

We now know how to go online and view Ethernet traffic, but have only really scratched the surface of Vehicle Spy 3's powerful *Messages View*. In this lab we will further explore this essential element of the VSpy software, looking at some of the functions and tools that will let you dig into and understand Ethernet messages.

You should still have *1.1 Custom Column Setup* loaded from having done Lab 1.1. If not, follow the directions in Lab 1.1 to load that setup.

Please start by taking a brief look at Figure 15. This image is a capture of the VSpy *Messages View* similar to the ones we saw in the previous lab, but various elements of the user interface have been highlighted and labeled. We will refer back to this figure throughout the pages that follow.

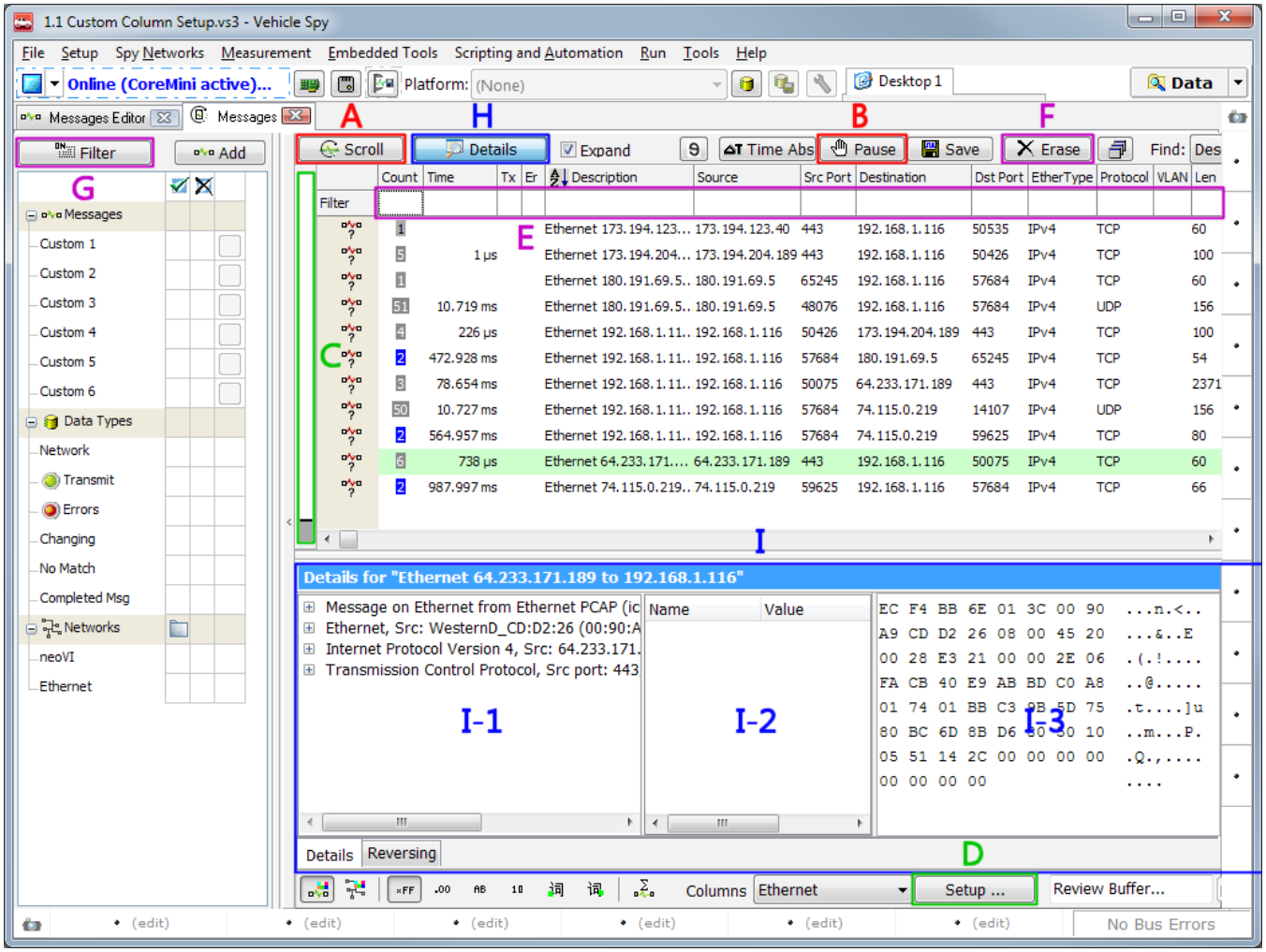


Figure 15: Annotated Messages View. Another screenshot of the Vehicle Spy 3 *Messages View*, with various features highlighted for easy reference as you read this lab. [A: Scroll button. B: Pause button. C: Circular message buffer display. D: Column and buffer setup button. E: Filter row. F: Erase button. G: Master filter on/off button. H: *Details View* button. I: *Details View* window; I-1: Information pane; I-2: Signal Name/Value pairs; I-3: Message byte/character display.]

Part 1.2A Exploring Scroll Mode

As we mentioned in Lab 1.1, Vehicle Spy 3 defaults to aggregating similar messages for easier tracking and analysis. This is called the *static mode* of the *Messages View*. However, sometimes we want to see messages sequentially rather than having them grouped in this manner. Naturally, we can easily change between the two modes.

- **Go Online:** Press .
- **Enable Scroll Mode:** Press the Scroll button, located near the top left of the *Messages View* (Figure 15:A).

You will immediately see the change in the message display, as it (appropriately enough) begins scrolling, each new message appearing in a separate line rather than being grouped. Notice also that the *Count* column has now been replaced by a *Line* column, the number of which increases sequentially. The Scroll button has also changed to Scroll, indicating that scroll mode is currently on.

If you have a decent amount of traffic on your network connection, you may find that the *Messages View* scrolls rather quickly—perhaps even too quickly to get much of a chance to look at individual messages. One way of dealing with this is simply to pause the screen.

- **Pause Message Scrolling:** Go to the top middle of the *Messages View* and press the Pause button (Figure 15:B).

Obediently, Vehicle Spy 3 pauses the message display, though of course the program will continue capturing messages as they come in. Notice that the Pause button has also turned blue to indicate that message display has been temporarily halted. Let's resume message display now.

- **Resume Message Scrolling:** Press the Pause button again.

You'll notice that the line number "jumps" as Vehicle Spy 3 resumes showing you the most recent messages, and the blue highlighting on the button disappears. The messages with numbers that were not displayed are held in the buffer so you can pause again and scroll back to find them if desired.

Now let's go back to static mode.

- **Enable Static Mode:** Press Scroll again to turn off scroll mode.

Vehicle Spy 3 resumes showing you messages statically aggregated by type with counts.

Part 1.2B Changing Buffer Size and Clearing the Buffer

Vehicle Spy 3 stores messages in a circular buffer, which is represented by a slowly rising gray vertical bar to the left of the *Count* column (Figure 15:C). When the bar gets all the way to the top, the buffer is full, and VSpy will then start overwriting the oldest data with newer data.

By default Vehicle Spy 3 will store 50,000 messages, but this can be changed. Let's give it a go.

- **Go Offline:** We can't change the buffer size while data is coming in, so press the (Stop) button to go offline.
- **Enter Message Setup:** Press the Setup ... button (Figure 15:D). You may recall that we used this button to customize the column setup in Lab 1.1.

A dialog box appears with the first edit box containing the default value of 50000.

- **Change Buffer Size:** Enter 100000 in the box below the Number Of Messages in History (min 100) label. Then press the Close button to close the dialog box.
- **Go Online:** Let's go back online again now, using the button.

VSpy will now collect 100,000 messages before overwriting old data.

Part 1.2C Sorting Columns

Vehicle Spy 3 allows you to sort the messages in the *Messages View* so you can see the data in whatever form works best for you. To do this, simply click on a column header until it contains the symbol (for ascending order) or (for descending order). You may notice that VSpy actually starts up with *Messages View* sorted by *Description* in ascending order.

> **Note:** If you've made the *Count* column very narrow, you may want to widen it somewhat for this part of the lab, so you can see the changes to the header.

- **Go Offline:** Press to go offline.
- **Sort by Count in Ascending Order:** With scroll mode off, press the Count column header once.

The messages are automatically sorted with the lowest count at the top and the highest at the bottom (Figure 16).

- **Sort by Count in Descending Order:** Press the Count column header.

The order is now reversed.

- **Disable Sorting:** Press Count.

Sorting is disabled and the standard view restored.

If you wish, try clicking some of the other columns to experiment with the sorting feature.

Count	Count	Count
3	1	4
4	1	3
3	2	3
2	2	3
2	2	2
2	3	2
1	3	2
1	3	1
3	4	1

Figure 16: Sorting in the Messages View. Left, message type counts as they appear by default in the *Messages View* of VSpy; middle, the same messages sorted in ascending order by count; right, sorted in descending order.

Part 1.2D Filtering

Another way of managing the large amounts of data generated by Ethernet networks is to use *filters* so you see only what interests you at a particular time. At the top of the *Messages View*, just below the column headers, is a row that says *Filter* on the far left, and has an empty box for each column (Figure 15:E). To create a filter, simply enter something in the appropriate box.

- **Filter by Protocol:** In static mode, enter UDP in the box below the Protocol header.

Vehicle Spy 3 will update the *Messages View* so only UDP messages are shown, while others are suppressed (Figure 17).

	Count	Time	Tx	Er	Description	Source	Src Port	Destination	Dst Port	EtherType	Protocol	VLAN	Len
Filter													
	2	10.029000 s			Ethernet 192.16...	192.168.1.116	58740	64.233.171.95	80	IPv4	TCP		55
	1				Ethernet 192.16...	192.168.1.116	65123	65.55.223.41	40016	IPv4	UDP		193
	8	780.000 ms			Ethernet 192.16...	192.168.1.116	65123	68.98.143.123	65428	IPv4	TCP		86
	1				Ethernet 192.16...	192.168.1.116	65123	68.98.143.123	25616	IPv4	UDP		64
	3	6.008000 s			Ethernet 192.16...	192.168.1.116	58762	69.167.144.15	443	IPv4	TCP		62
	1				Ethernet 192.16...	192.168.1.116	65123	71.163.153.73	12679	IPv4	UDP		64
	8	420.000 ms			Ethernet 192.16...	192.168.1.116	65123	71.163.153.73	50816	IPv4	TCP		58
	1				Ethernet 192.16...	192.168.1.116	58761	74.125.22.189	443	IPv4	TCP		54
	2	10.009000 s			Ethernet 192.16...	192.168.1.116	58751	74.125.226.185	80	IPv4	TCP		55
	2	9.228000 s			Ethernet 192.16...	192.168.1.116	58630	74.125.226.86	443	IPv4	TCP		54
	2	10.026000 s			Ethernet 192.16...	192.168.1.116	58756	74.125.228.5	80	IPv4	TCP		55
	6	2.1000000 s			Ethernet 192.16...	192.168.1.145	62348	239.255.255.250	1900	IPv4	UDP		139

	Count	Time	Tx	Er	Description	Source	Src Port	Destination	Dst Port	EtherType	Protocol	VLAN	Len
Filter											UDP		
	2	0 µs			Ethernet 192.16...	192.168.1.116	65123	157.55.56.168	40017	IPv4	UDP		195
	1				Ethernet 192.16...	192.168.1.116	52034	192.168.1.1	53	IPv4	UDP		83
	1				Ethernet 192.16...	192.168.1.116	62858	192.168.1.1	53	IPv4	UDP		78
	3	760.000 ms			Ethernet 192.16...	192.168.1.116	137	192.168.1.255	137	IPv4	UDP		92
	4	0 µs			Ethernet 192.16...	192.168.1.116	52608	192.168.2.245	161	IPv4	UDP		120
	1				Ethernet 192.16...	192.168.1.116	65123	213.199.179.157	40018	IPv4	UDP		80
	1				Ethernet 192.16...	192.168.1.116	65123	50.77.232.178	22687	IPv4	UDP		73
	1				Ethernet 192.16...	192.168.1.116	65123	50.77.232.178	9825	IPv4	UDP		73
	1				Ethernet 192.16...	192.168.1.116	65123	65.55.223.41	40016	IPv4	UDP		193
	1				Ethernet 192.16...	192.168.1.116	65123	68.98.143.123	25616	IPv4	UDP		64
	1				Ethernet 192.16...	192.168.1.116	65123	71.163.153.73	12679	IPv4	UDP		64
	6	2.1000000 s			Ethernet 192.16...	192.168.1.145	62348	239.255.255.250	1900	IPv4	UDP		139

Figure 17: Messages View Filtering. The message list area of the Vehicle Spy 3 Messages View, seen before (above) and after (below) adding a *Protocol* field filter to select only UDP messages.

- **Clear the Filter:** Return to the same box, delete the letters UDP and press *Enter*.

The *Messages View* is now restored to its default state.

Next, let's try a more complex filter.

- **Filter by Protocol and Count:** Enter TCP in the Protocol column filter box, and 1 in the Count filter box.

You will now see only TCP messages that were received exactly once since Vehicle Spy 3 went online. While online, Vehicle Spy 3 will even add and remove entries from the display dynamically as their counts change.

You can also use a comma to separate two different filter criteria.

- **Filter on Two IP Counts:** Change the Count filter to 1,2.

Now all messages that have been received either one or two times will be displayed.

Part 1.2E Filtering in Scroll Mode

When scroll mode is on, adding a column filter will only suppress the display of newly-arriving messages that don't match the filter; existing messages will be preserved. Let's see how this works.

- **Clear Filters:** Delete all filters from the filter row.
- **Enter Scroll Mode:** Press Scroll.
- **Go Online:** Press .
- **Enter a Protocol Filter:** Enter ARP in the EtherType column filter box.

You will probably now see the scrolling messages appear to stop, as if nothing is happening. This is because ARP messages are usually received infrequently on real Ethernet networks. Vehicle Spy 3 is still running, and is still showing you the messages that were received before you entered the filter. However, it is suppressing the display of all non-ARP frames, and these usually arrive only infrequently.

If you wait for a minute or so, you should see some ARP messages start to show up. However, the old messages will also still hang around. It would be easier to see what we're doing without this clutter of old messages, which don't match what we're looking for anyway. Let's remove them from the display.

- **Erase All Messages from Messages View:** Press the Erase button, near the top center of the *Messages View* display (Figure 15:F).

All current messages in the display are removed. Now only new ARP messages that arrive will be shown (Figure 18).

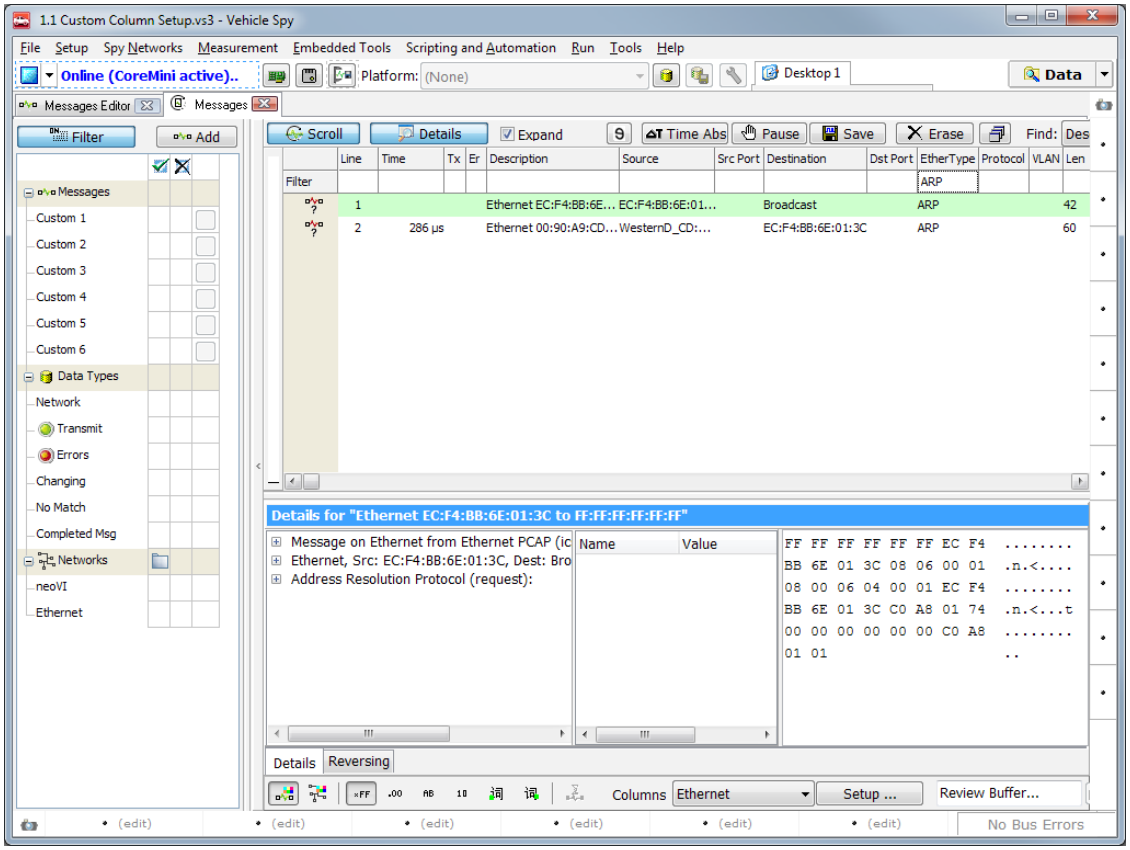


Figure 18: Messages View with ARP Filter in Scroll Mode. In scroll mode, applying a filter will not cause older messages to be cleared. After hitting the *Erase* button, only messages matching the filter will be shown, as displayed here.

Leave this filter active for the next part of the lab, but exit scroll mode.

► **Press** Scroll**.**

Vehicle Spy 3 now will resume collecting together ARP messages of the same type.

Part 1.2F Master Filtering Control

You can disable all filtering at once by using the Filter button located far left (Figure 15:G). This will cause Vehicle Spy 3 to ignore any filters entered into the filter row, which will be retained, but grayed out.

► **Disable Filtering:** Press Filter to turn filtering off.

Notice that all messages are displayed again as filtering is suppressed.

► **Enable Filtering:** Press Filter to resume filtering.

And now the filter is applied again.

Part 1.2G Using Details View

This extra area in the *Messages View* allows you to dig into the fields in any message to let you see what's really going on in your network. It's an especially powerful tool for Automotive Ethernet because you can use it to examine each of the layers of headers and data found in complex encapsulated messages such as those used in TCP/IP.

You should see *Details View* in a window near the bottom of *Messages View*; the title will begin with Details for (Figure 15:I). The Details button, located to the right of the Scroll button at the top of the message display (Figure 15:H), can be used to turn *Details View* off and on. As with the other buttons, it is highlighted in blue when enabled. *Details View* is normally enabled by default, as you may recall from the demo we did in the User's Guide, where we temporarily disabled it and then turned it on again.

- **Enable or Disable Details View:** Press Details or Details to toggle the *Details View* off and on. Stop with the view enabled.

The *Details View* window contains three panes. The information pane on the left (Figure 15:I-1) shows a list of messages and decoded information about them. On the right is a byte/character display of the selected message (Figure 15:I-3). In the center, you'll see an area with *Name* and *Value* columns (Figure 15:I-2) that are used to display the values of decoded messages. The contents of all of these panes will change depending on the type of message is selected in the *Messages View*. The relative sizes of the three panes can be changed by dragging the vertical dividers between them, just as we did with the Ethernet interfaces list.

Let's start by looking at an ARP message. Conveniently enough we still have an ARP filter applied in Messages View, so that should not be difficult to do!

- **Select an ARP Request Message:** Find an ARP message in *Messages View* that has a Destination of Broadcast and click it.

Vehicle Spy 3 will highlight the chosen ARP message and display its information in the *Details View*. You will see that this message contains an Ethernet header and an Address Resolution Protocol header for the *ARP Request* message type. An example *Details View* display for an *ARP Request* message can be seen in Figure 19.

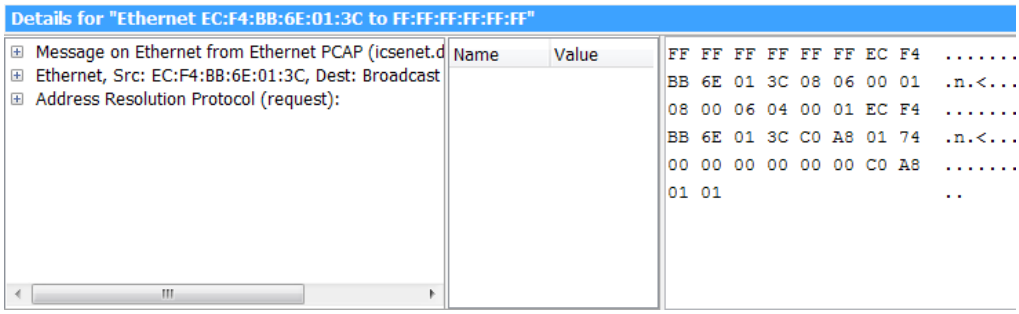


Figure 19: Initial Details View Display for an ARP Request Message.

Now let's try a different message type.

► **Filter for UDP Messages:** Enter IPv4 in the EtherType filter field and UDP in the Protocol field.

One of the UDP messages is displayed, which shows three header types: *Ethernet*, *Internet Protocol* and *User Datagram Protocol*. If you select any of these, the corresponding header bytes in the message will be highlighted in the data area on the right. Let's try it.

► **Select the Internet Protocol Header:** Click the Internet Protocol Version 4 header in the information pane.

You should see 20 bytes highlighted in gray in the byte area, corresponding to the 20 bytes in a standard IPv4 header (Figure 20).

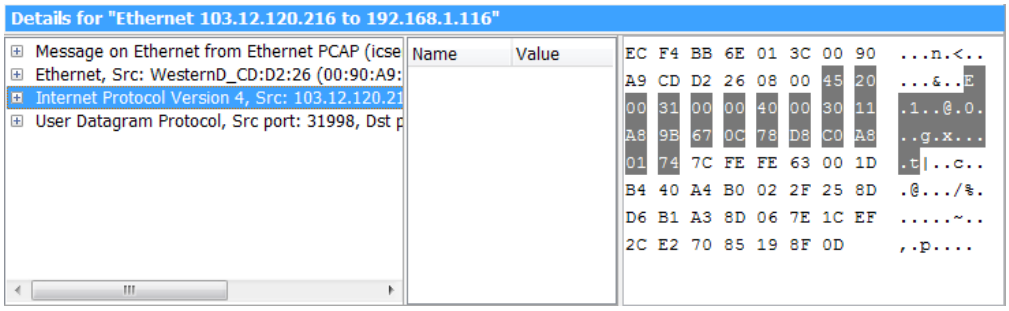


Figure 20: Details View Display for a UDP Message with IP Header Highlighted. Here we have the *Details View* display for a typical UDP message. When the IPv4 header is selected in the information pane, Vehicle Spy 3 highlights its 20 bytes in the byte/character display area.

Next, let's try drilling down into the headers to look at the fields they contain.

► **Expand the Ethernet Header:** Click the Ethernet header in the *Details View* information pane, and then click the ⊞ button to the left of it.

Vehicle Spy 3 shows you the *Destination*, *Source* and *Type* fields in the Ethernet header (Figure 21).

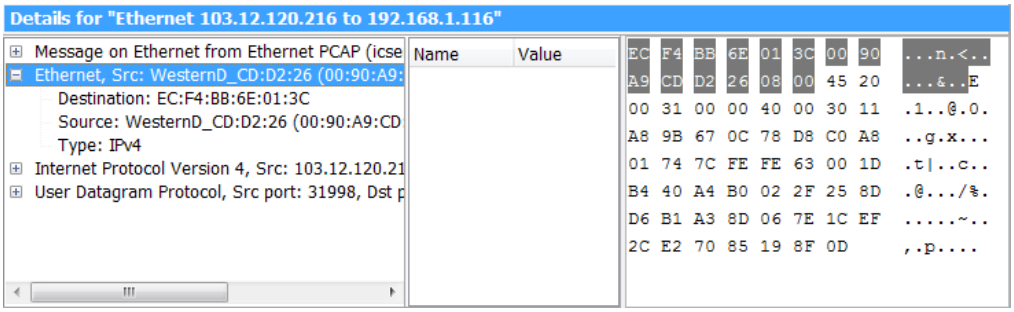


Figure 21: UDP Message Details View with Expanded Ethernet Header. The same UDP message shown in Figure 20, but with the Ethernet header selected and expanded to show its three fields. Notice the change to the highlighting in the byte/character display area.

Next, let's try the IP and UDP headers.

► **Expand the Other Headers:** Click the ⊞ buttons to the left of the Internet Protocol Version 4 and User Datagram Protocol labels.

Notice how Vehicle Spy 3 decodes all of the headers for you, describing what each field is, and where relevant, what the field values represent. You may need to scroll down to see all the fields due to the large number of fields in the IPv4 header.

You can even pinpoint the exact location of specific bytes in a header field.

► **View the Data Bytes for the Time to live field:** Under the Internet Protocol Version 4 header, click the Time to live field header.

You will see a single byte on the right side of the display now highlighted in gray. The hex value shown here should match the decoded decimal value in the information pane (Figure 22).

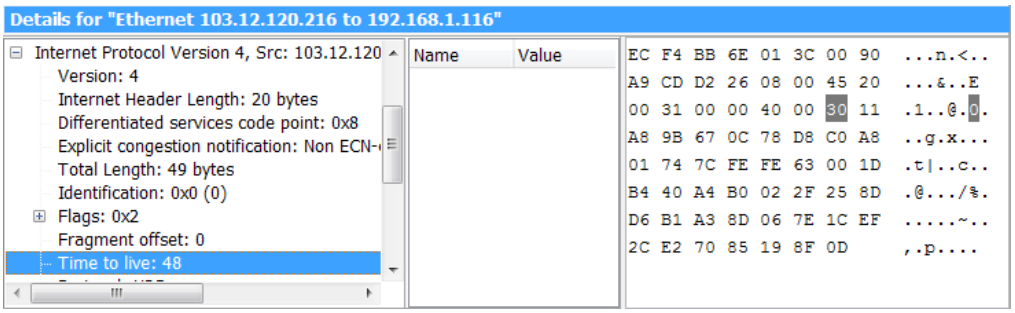


Figure 22: UDP Message Details View with IPv4 Header Expanded and TTL Selected. The same UDP message again, this time with the IPv4 header expanded and the *Time to live* field selected in the information pane. Note that the value of the field is 48 (decimal) corresponding to the byte value 30 (hexadecimal) in the byte/character display.

We're done for now.

► **Go Offline:** Press to go offline.

Lab 1.3 Using the Messages Editor and a Function Block Script to Transmit Raw Ethernet Frames

With some Vehicle Spy 3 experience under our belts, we are ready to begin actually using the EEVB to simulate Ethernet node functionality. We will start with a very simple demonstration that programs one of the two EEVB nodes to regularly transmit a basic "raw" Ethernet message. And even though this is our first EEVB demo, to help you get familiar with how to use Vehicle Spy 3's *Messages Editor* and *Function Block* features, we're going to dive in head first by having you create the demo yourself rather than using a pre-made setup file. Don't worry, though—it's actually quite easy, we will guide you through the process step by step, and you'll learn a lot this way!

> **Note:** A "raw" Ethernet message is one that only has Ethernet headers and not those from any encapsulated higher-level protocols such as IP or ARP. This is the simplest Automotive Ethernet message type and forms the basis for the more complex messages we will work with later in the Lab Manual. You can find much more information about Ethernet message formats in your copy of *Automotive Ethernet - The Definitive Guide*.

Part 1.3A Restart Vehicle Spy 3 and Load Custom Column Setup

Since we're about to do something new, let's restart Vehicle Spy 3 to ensure that we are beginning with a fresh slate.

- **Close Vehicle Spy 3:** Select *Exit* from the *File* menu or press *Alt+F4*. Do not save changes if you are asked.
- **Start Vehicle Spy 3:** Select Vehicle Spy 3 from the Windows Start Menu or click .

As we first saw in Lab 1.1, Vehicle Spy 3 remembers the last several setup files you have used, listing them under the Recent tab on the *Logon Screen*. You should see *1.1 Custom Column Setup* here since we created it earlier in the Lab Manual and have used it since.

- **Select Active Ethernet Interface:** On the *Logon Screen*, click the radio button next to the most active Ethernet interface.
- **Load the 1.1 Custom Column Setup:** Double-click 1.1 Custom Column Setup, which should be on the Recent tab. If you don't see it there, look under My Setups.

We are now ready for business.

Part 1.3B Create a Custom Raw Ethernet Transmit Message

Since this is a transmission demo, our first task is to create the message we will transmit using Vehicle Spy 3's *Messages Editor*.

► **Open the Messages Editor:** Select Messages Editor from the *Spy Networks* menu.

The *Messages Editor* is used to define both receive and transmit messages. You can switch between the *receive side* and the *transmit side* by clicking the Receive and Transmit buttons near the top left of the screen. (Don't worry about the Database button.) The currently selected "side" is indicated by which of these buttons is highlighted in blue.

► **Switch to Transmit Messages:** If the Transmit button near the top left of the screen is not already highlighted, click it.

Let's make sure that Vehicle Spy 3 knows we want to work with Ethernet messages.

► **Select the Ethernet Network:** Click the on Network drop-down box, scroll down through the list, and select Ethernet (Figure 23).

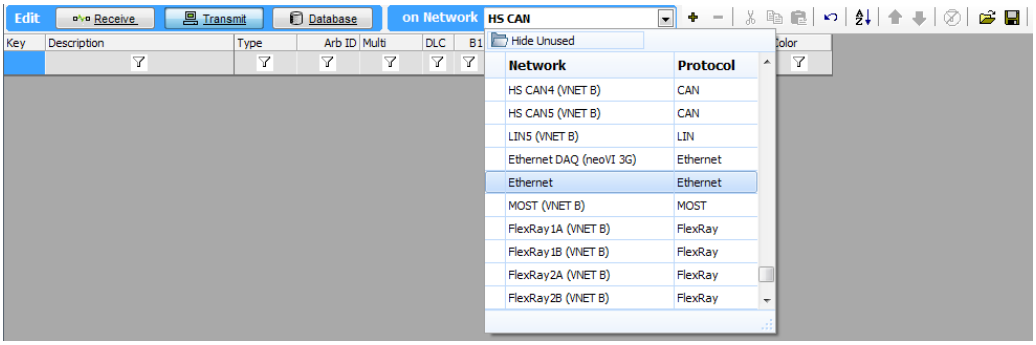


Figure 23: Selecting Ethernet in the Messages Editor.

Vehicle Spy 3 will remember this setting until you restart the program. The setting is also saved in the current setup file. Come to think of it, we are always going to want this set to Ethernet, so let's update our column setup file now; this way, we won't have to do it again.

► **Save Setup File:** Select Save from the *File* menu.

Okay, now we are ready to add the new message.

► **Add a New Ethernet Message:** Click the ✚ button to the right of the on Network drop-down box.

Vehicle Spy 3 will create a new message called *Tx Message Ethernet 1* and open a window pane where signals in the message can be viewed and edited (Figure 24).

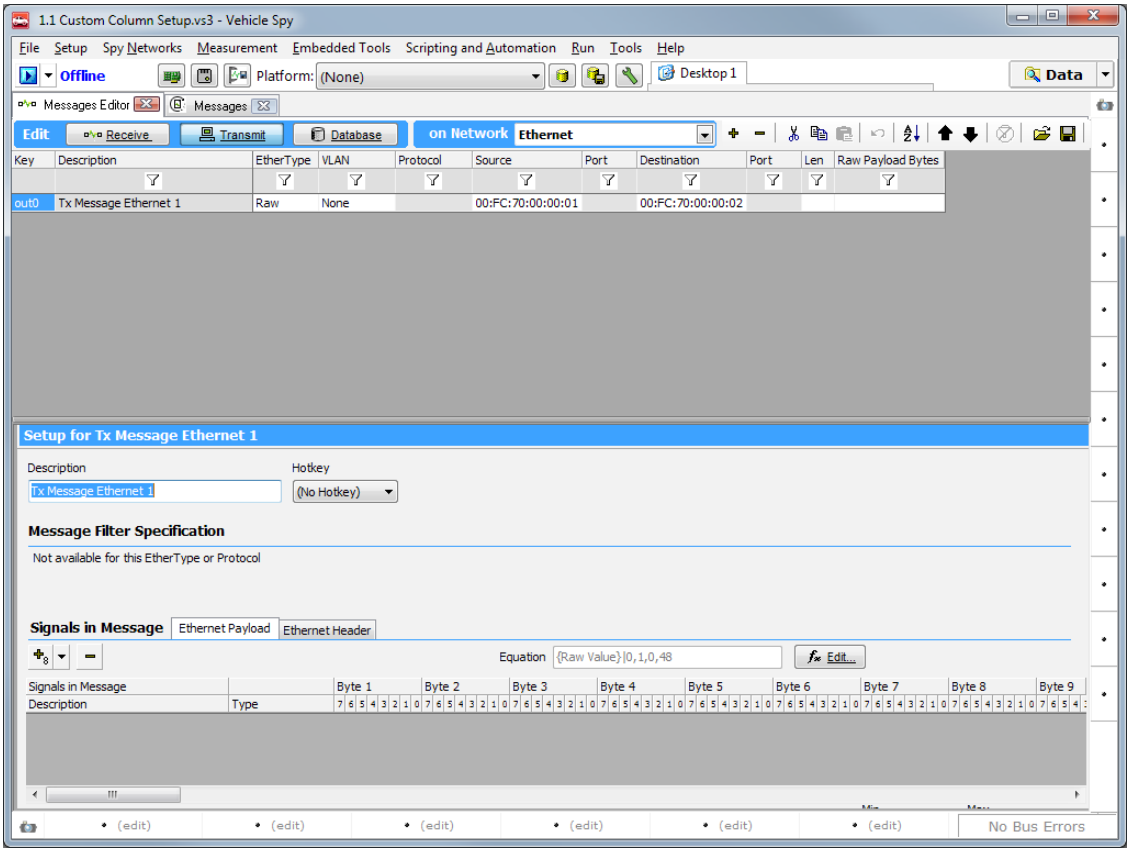


Figure 24: New Transmit Message Added in the Messages Editor.

Note the *Key* entry for this message, which will begin with **out** (and will normally be **out0**). The key is the actual means by which the message is referenced internally within Vehicle Spy 3, which allows us to change message names without affecting functionality. Let's do that now, since the default name is somewhat generic.

► **Change Message Title:** Double-click Tx Message Ethernet 1 under Description in the transmit messages list. Enter the name Ethernet Lab 1.3 Frame instead (Figure 25).

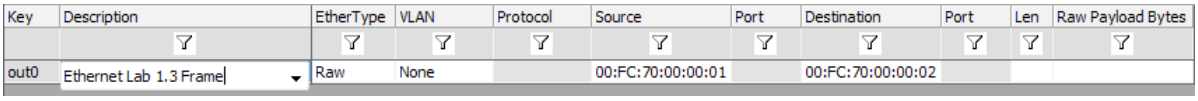


Figure 25: Changing the Transmit Message Description in the Messages Editor.

Notice that the *EtherType* for this message is **Raw**, indicating a simple frame with only Ethernet headers; as we'll see later on, Vehicle Spy 3 will also let you directly set up more complex messages. Let's take a look at the header fields.

> **Note:** In addition to a header, Ethernet frames also have a *footer* transmitted after the frame's data that contains a single 4-byte field called the *Frame Check Sequence* (*FCS*). This field carries a cyclical redundancy check (CRC) code to allow the recipient of a frame to detect many types of transmission errors. The FCS is computed automatically by Vehicle Spy 3 and by the EEVB, and is not needed by VSpy users, so it is not shown in message formats.

- **Select Ethernet Header Fields:** In the window pane near the center of the screen, which should now be titled **Setup for Ethernet Lab 1.3 Frame**, find the Signals in Message area, and click the tab labeled Ethernet Header.

You can see that Vehicle Spy 3 has automatically set up the standard three Ethernet frame header fields, which are called *signals* in VSpy for consistency with the terminology used in other automotive networks (Figure 26). The fields will be transmitted in the order shown, and have sizes indicated by the byte display to their right: 6 bytes for each of *Destination MAC Address* and *Source MAC Address*, and 2 bytes for *EtherType or Length*. (You won't be able to see the full display of all three fields unless your screen is very wide or you use the scroll bar on the bottom.)

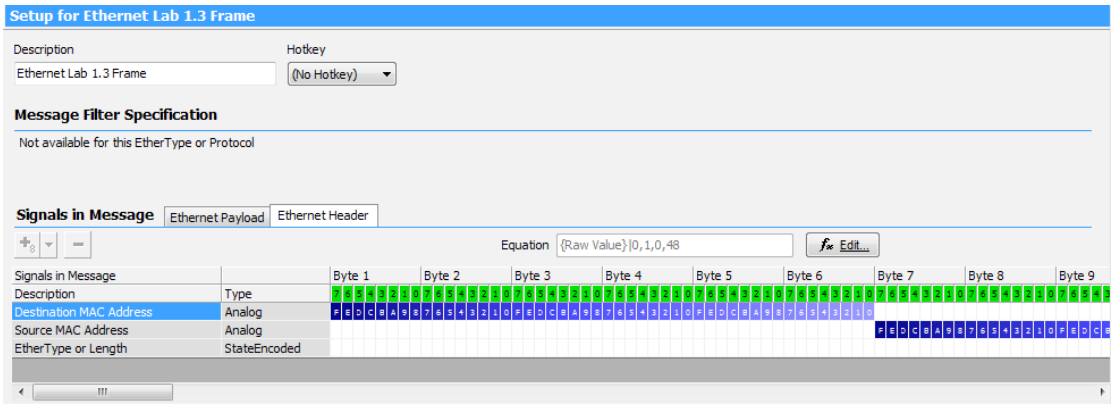


Figure 26: Default Ethernet Headers for Transmit Message in the Messages Editor. Note the scroll bar on the bottom, used to allow you to view fields off to the right when they won't all fit; exactly what you see here will depend on the size of your Vehicle Spy 3 window.

We have header fields but no actual data fields in our message yet; let's rectify that.

- **Select Ethernet Payload Fields:** Click the Ethernet Payload tab in the *Messages Editor*.
- **Add an 8-Bit Signal:** Click the button, located just below the label Signals in Message.

An 8-bit signal will appear called *Signal 3*. If you scroll to the right, you'll notice that its value is in byte 15 of the message, after the standard 14-byte Ethernet header. Let's rename this field to something less generic as well.

> **Note:** Why *Signal 3*? While it may not be obvious at first, this is the fourth signal in this message, since it comes after the three Ethernet header fields. VSpy numbers messages starting from 0.

► **Change Payload Signal Name:** Double-click Signal 3 and change the name to Lab 1.3 Data (Figure 27).

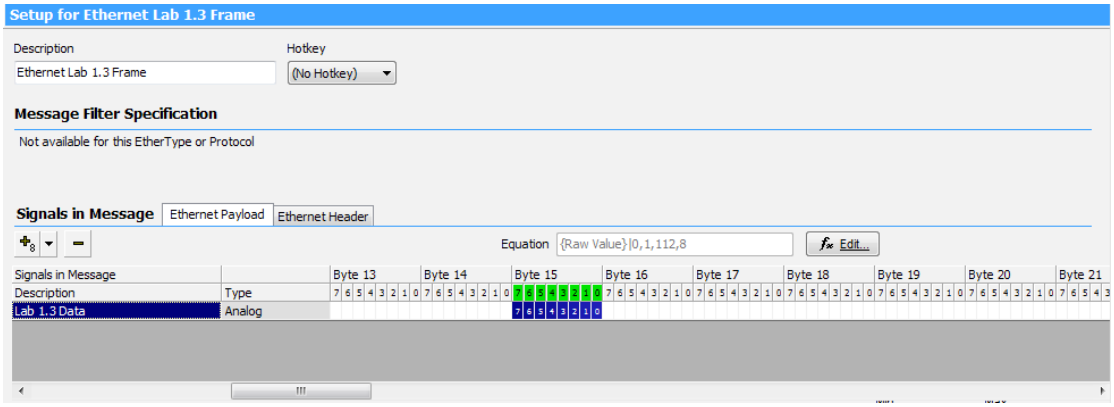


Figure 27: Ethernet Payload Signal for Transmit Message. We have created an 8-bit field (signal) to be carried in our transmitted Ethernet message. It is located in byte 15, after the 14-byte Ethernet header, and was shown here by using the scroll bar.

Part 1.3C Examine and Change Transmit Message Field Values

Our basic transmit message is now complete, but while we've defined the message's fields, we haven't set their values yet. There are generally two ways to do this: manually editing them, or assigning them values in a function block program. We'll once again start with the simpler method, manual assignment; this is accomplished via the Vehicle Spy 3 *Tx Panel*.

► **Open the Tx Panel:** Select Tx Panel from the *Spy Networks* menu.

The Tx Panel is split into two halves; on the left is a list of all transmit messages, along with some controls, while on the right is an area where signals within a message are shown with boxes for setting their values.

► **Select the Ethernet Lab 1.3 Frame Message:** On the left side of the screen, click Ethernet Lab 1.3 Frame.

On the right side you will now see the four fields (signals) in our message: the three Ethernet headers, and the *Lab 1.3 Data* field.

Depending on your window size, you may find that at first too much of the left-hand pane is showing, while the values on the right are cut off after only a few characters. This is easily corrected.

► **Move Tx Panel Divider Bar:** Find the divider that separates the messages list from the message details and drag it to the left until the Value column is fully displayed (Figure 28). (The Raw Value column is not important at the moment.)

Figure 28: Tx Panel Display of Ethernet Transmit Message. Selecting the *Ethernet Lab 1.3 Frame* message on the left displays its four signals and their current values in editable boxes on the right.

But wait... some of these fields already have values, which we never entered. Not to worry, this is just another way that Vehicle Spy 3 makes life easier for you, in this case by filling in default figures for message headers automatically. Here only three values were set, which would have been easy enough to do ourselves. However, we'll soon see how this feature becomes increasingly valuable as we start working with longer message types, some of which can have a dozen headers or more (including many TCP/IP messages). In the next lab we will see where these defaults come from, and how to change them.

Let's now put a specific value into our data field, which is currently blank.

► **Set the Value of the Lab 1.3 Data Field:** Double-click the empty field under the Value column for Lab 1.3 Data, and enter 78 (Figure 29).

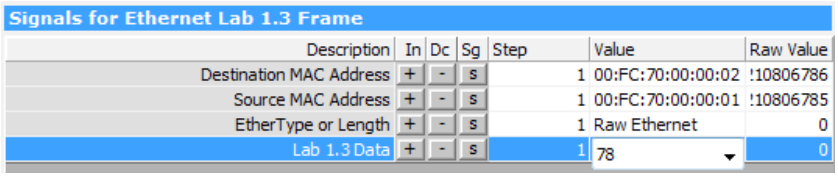


Figure 29: Setting Payload Signal Value for Transmit Message.

Each transmission of the *Ethernet Lab 1.3 Frame* message will now send the value 78 in its Ethernet frame payload.

Part 1.3D Create a Function Block Script to Tell the Ethernet EVB to Transmit the Message

Our message is now ready for transmission, so it's time to write a (very) short program to tell the Ethernet EVB to send it. We'll instruct one node to simply send the same message out on a regular interval, let's say every 2 seconds.

Let's go to the *Function Blocks* area and create a new function block.

► **Open the Function Blocks Window:** Under the *Scripting and Automation* menu, select Function Blocks.

- **Create a New Function Block Script:** Click the ✚ button at the top left of the screen, and then select Script from the pop-up that appears (Figure 30).

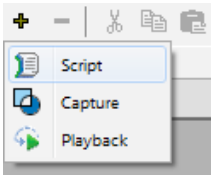


Figure 30: Adding a Function Block Script.

In the upper part of the screen you will now see a new function block, creatively titled *Function Block 1* and with a *Key* value of tst0. A lower pane shows that this script currently has 10 empty program steps. Since we will have only one function block, its name is unimportant, so we can leave it as the default.

We only need our program to do two things: transmit our message and then wait 2 seconds before doing it again. The transmit step comes first.

- **Create a Transmit Step:** Double-click the Description field in *Step 1* and select Transmit. Then hit *Enter*. Ignore the message that appears, as it will go away as soon as we select a message. Notice that the default comment moves to the Comment column.
- **Select Ethernet Lab 1.3 Frame to Transmit:** Double-click the field that currently contains Select Message in the Value column for *Step 1*. A menu appears listing the transmit messages currently defined in VSpy; of course, there is only one right now, so click Ethernet Lab 1.3 Frame, which has been conveniently pre-selected for you.
- **Enter Comment for Transmit Step:** Double-click the Comment field and replace the default value (// TODO: Add step commands here) with Send Lab 1.3 raw Ethernet frame.. Vehicle Spy 3 will add the double-slash ("//") comment header for you.

While comments do not affect the operation of a script, they are still very important, because they serve as a function block's documentation. Explaining what each step does in a program may seem like extra work, but is essential when longer or more complex function blocks need to be maintained or modified weeks or months down the road.

At this point your Vehicle Spy 3 window should look something like Figure 31.

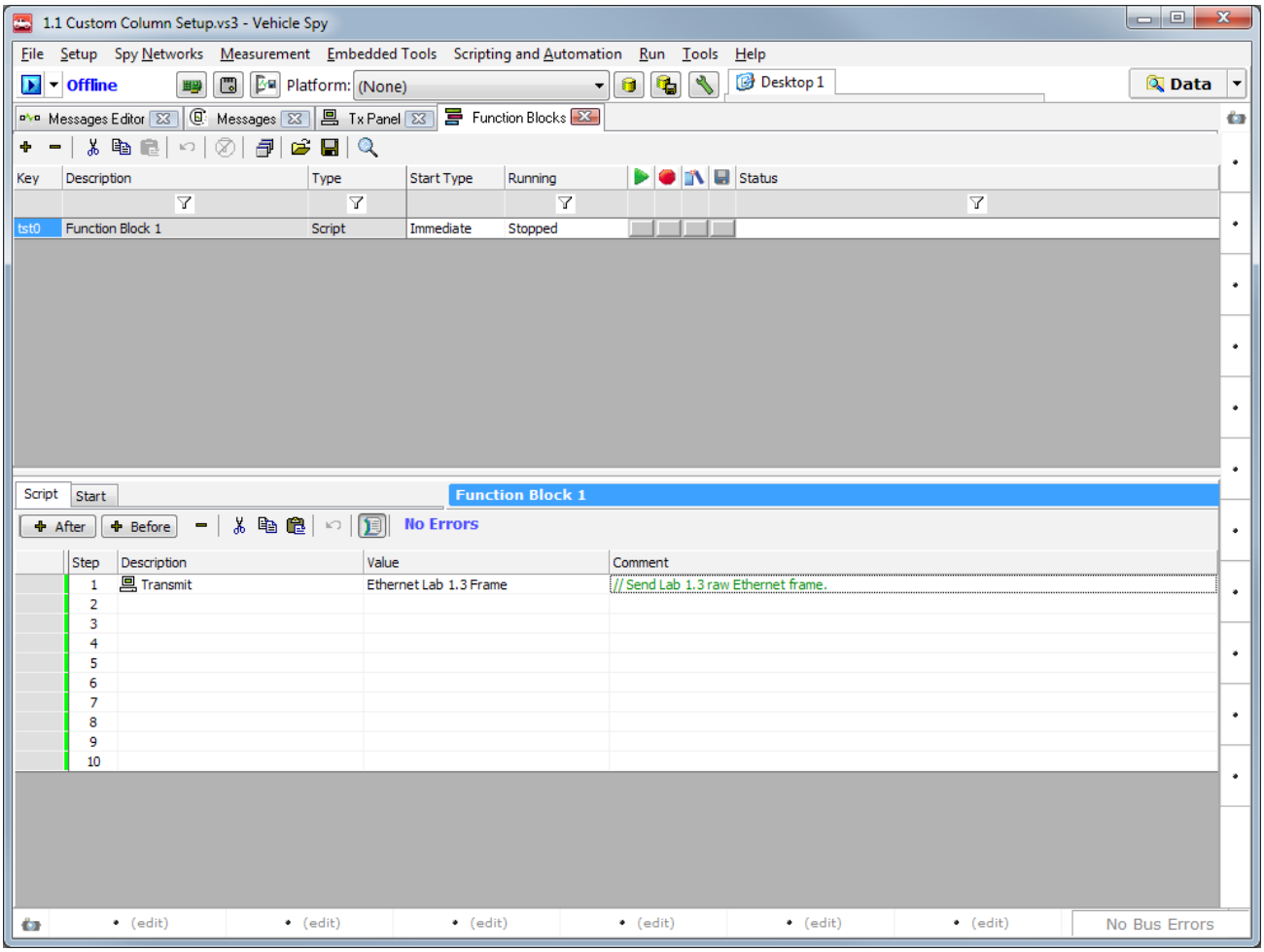


Figure 31: Function Block Script After Adding Transmit Step.

Let's now add the delay step.

- **Create a Wait For Step:** Double-click the Description field for *Step 2* of the script and select Wait For.
- **Change the Wait Time:** Double-click the default entry in the Value field (0.001000 sec) and then press the = key to bring up the *Expression Editor*. Enter 2000 in the Expression field (Figure 32) and either click OK or press the *Enter* key.
- **Enter Comment for Wait Step:** Enter the following comment for this step: Wait for 2 seconds before transmitting again..

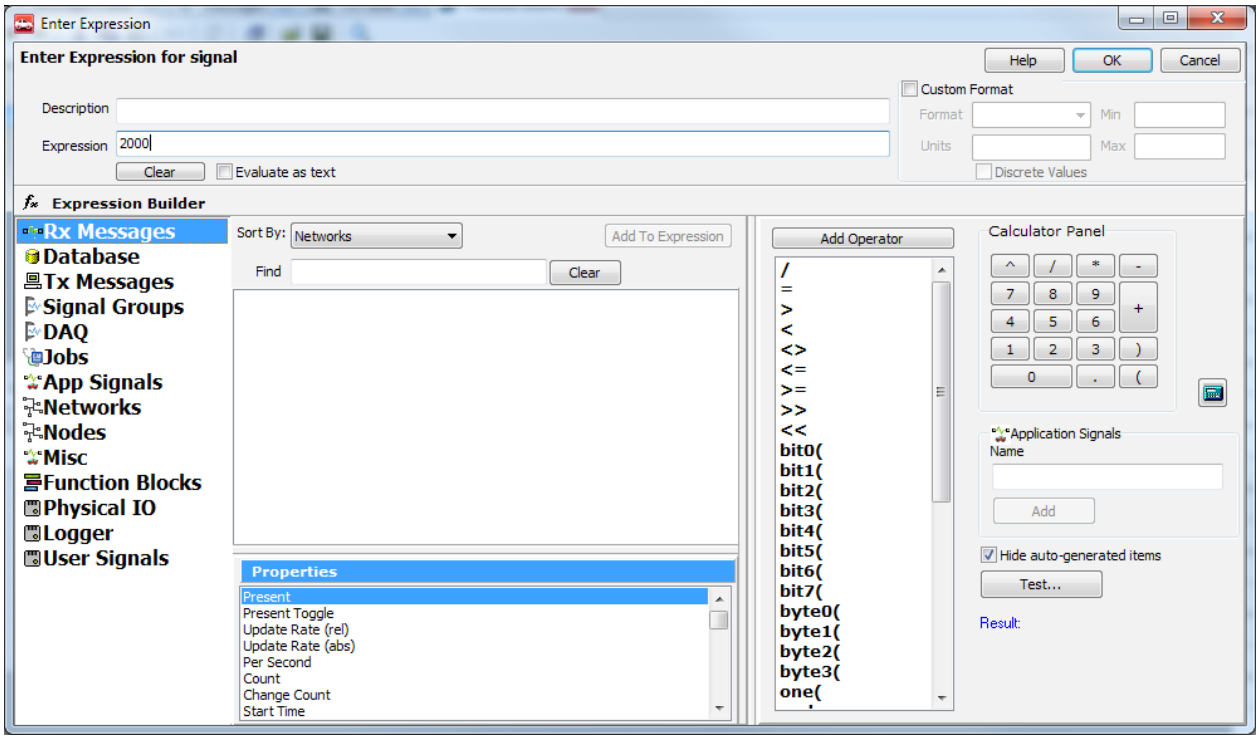


Figure 32: Setting a Numeric Value in the Expression Editor. This is the simplest use of the *Expression Editor*; later in the Lab Manual we'll see that we can do much more here.

Vehicle Spy 3 will ignore the 8 remaining blank steps, but let's delete them just for cleanliness (and so we can learn how to delete steps, of course!)

- **Delete Blank Steps:** Click on *Step 3*, then hold down the *Shift* key and click on *Step 10* to highlight the 8 blank entries. Then either right-click and select Delete from the context menu (Figure 33), or press the − button just above the script steps.

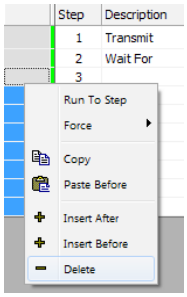


Figure 33: Function Blocks Right-Click Menu. Here we have highlighted eight blank steps that we are preparing to delete.

Finally, we must tell Vehicle Spy 3 that we want this program to run on the EEVB as soon as it is loaded, but not to run on the PC itself. We do this by changing the *Start* parameter for the block.

- **Change the Start Type:** Click the Start tab. Then click the drop-down box that currently contains Start Immediately and change this to Start Immediately Embedded Only (Figure 34).

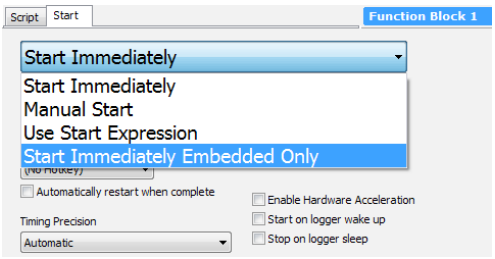


Figure 34: Function Blocks Start Tab. Changing the start type from **Start Immediately** to **Start Immediately Embedded Only** will ensure that our script runs on the EEVB without a duplicate copy also running within Vehicle Spy 3.

In the function block summary at the top of the screen, you should now see **Immediate Embedded** in the line for *Function Block 1* under *Start Type* (though you may need to widen the column to see the full text).

Part 1.3E Define a Receive Message to Match Our Transmit Message

In a moment we will send our program and custom Ethernet message to the EEVB. Before we do, however, we need to be sure to define a receive message that matches our transmit message. This will let Vehicle Spy 3 recognize and decode our special message so we can see its fields properly in *Messages View*. (As we'll see later on, this is only needed for messages we are defining in EEVB scripts, not for those that run natively within VSpy.)

Vehicle Spy 3 keeps the most recent windows you have used in tabs along the top of the display, so you can switch between views easily. Let's use this feature to switch back to the *Messages Editor*.

- **Return to the Messages Editor:** Click on the Messages Editor tab, located in the row just below the / button. If the tab is not present (perhaps because you closed the *Messages Editor*) then just select Messages Editor from the *Spy Networks* menu.

Next, we create a receive message simply by duplicating our transmit message.

- **Copy the Transmit Message to the Receive Side of the Messages Editor:** Click on Ethernet Lab 1.3 Frame, right-click it, select Copy To from the pop-up menu, and then choose Receive (Figure 35).

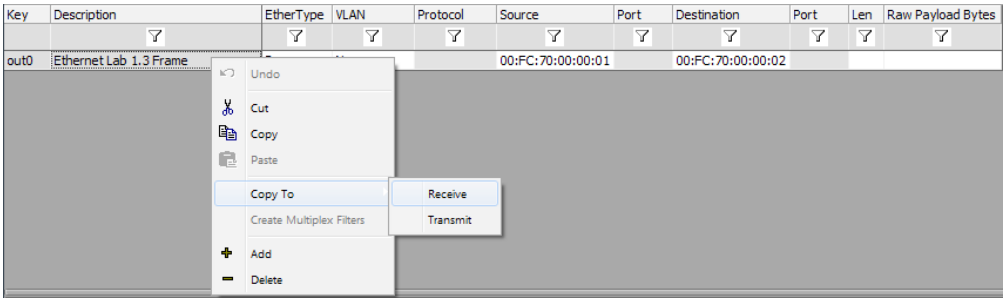


Figure 35: Copying a Transmit Message to the Receive Side.

- **Go to the Receive Messages Area:** Click Receive to switch to receive messages.

You should now see a receive message with the same name as the transmit message. Notice that the *Key* value here begins with **in** rather than **out**, since a receive message serves as input to Vehicle Spy 3, while a transmit message is output.

Part 1.3F Save Setup File

We're done creating our demo. Before proceeding let's save it so we can easily retrieve our program and message definitions. This will save time if we want to do this experiment again, and also lets us to make new demos by changing this one rather than starting from scratch.

- **Save the Setup File:** Select *Save As* from the *File* menu. When the dialog box appears, enter 1.3 Raw Ethernet Transmit.

Part 1.3G Send CoreMini to EEVB Node A

We will now send this setup to the EEVB so it can begin transmitting our message. This is accomplished via the same process that we followed for the initial demo in the User's Guide.

- **Enter CoreMini Console:** Click the *Tools* menu, hover over *Utilities* and then select *CoreMini Console...* from the popup.

The *CoreMini Executable Generator* dialog box appears. Vehicle Spy 3 automatically compiles your program into a CoreMini and connects to one of the nodes of the EEVB attached to your PC. Near the bottom of the window you'll see a drop-down box next to the word *Device*, which should by default be displaying the serial number of your EEVB's Node A or Node B in the form *EExxxx*, with the node label also present for clarity (Figure 36).

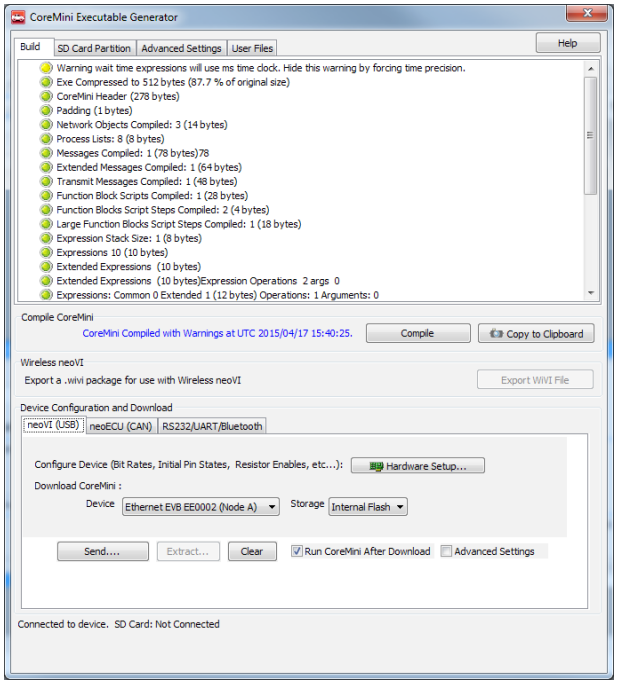


Figure 36: CoreMini Console. The CoreMini Console is used to send compiled message definitions and scripts to the Ethernet EVB. Here the *Device* drop-down box shows that Node A is selected.

- **Select Node A:** If Node A is not shown in the drop-down box, click the box and select it.
- **Download CoreMini:** Press the Send.... button below the device drop-down box. The message Sending CoreMini. appears in green text, followed a few seconds later by neoVI updated (Time nnnn ms) - Success, where nnnn is the number of milliseconds the task required.

That's it! Your EEVB's node is now running your program.

If you ran the demo in the User's Guide, your board will still have a copy of the CoreMini used for that setup running in Node B. Let's clear that out now.

- **Select Node B:** Click on Node B in the drop-down box.
- **Clear CoreMini:** Press the Clear button. After a few seconds the message Cleared CoreMini will appear.
- **Close CoreMini Console:** Close the console by pressing *Esc* or clicking the button top right.

Part 1.3H Go Online to View Transmitted Messages

Now let's go online to see the messages that we told the EEVB to transmit.

- **Select EEVB Ethernet Interface:** Return to the *Logon Screen*. Click the radio button for the Ethernet EVB in the Ethernet Interfaces list.
- **Go Online:** Press to go online.

Now let's switch to *Messages View*, and set a filter so we only see the message we told the EEVB to transmit.

- **Switch to Messages View.**
- **Enter Message Type Filter:** Enter Lab 1.3 in the Filter row for the Description column.

Now you should see just *Ethernet Lab 1.3 Frame* messages in *Messages View* (Figure 37).

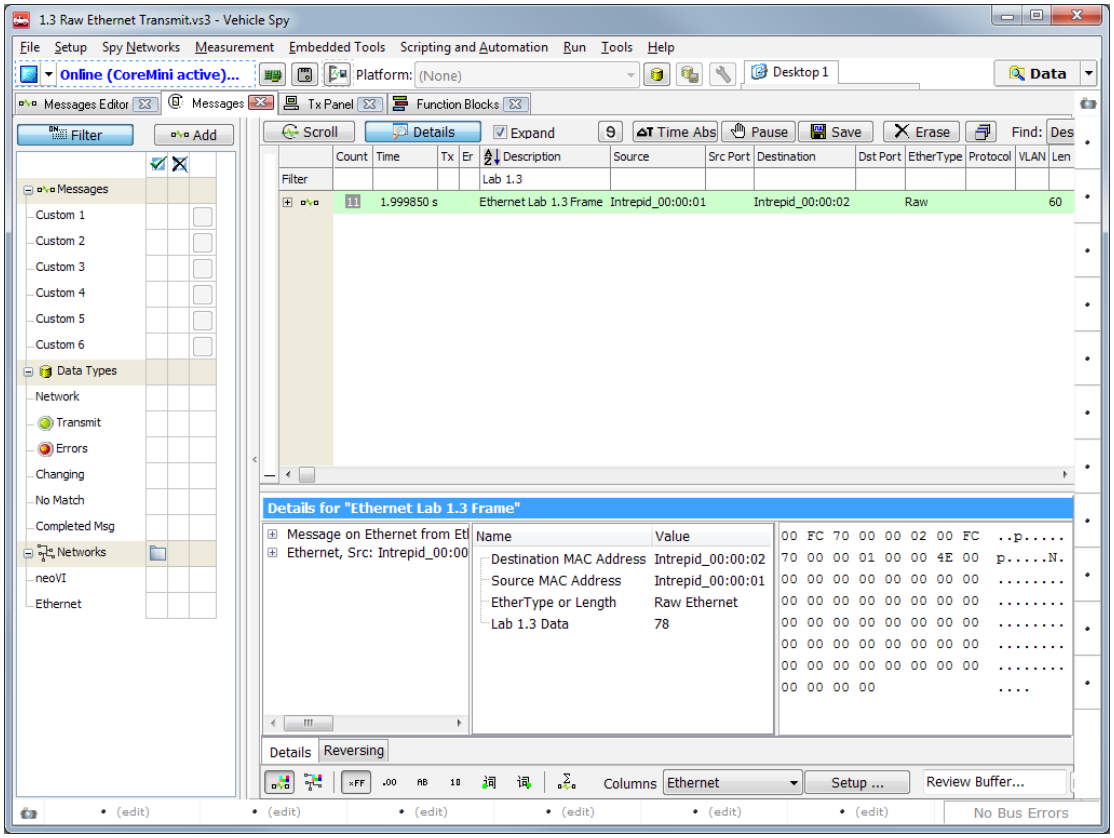


Figure 37: First Ethernet Frames Received from Ethernet EVB. You should see something like this when you go online for the first time to see your EEVB in action.

Notice that the *Time* column shows that a new message arrives approximately every 2 seconds (though the number is not exact because of natural variations). Since this is a raw Ethernet frame, the *Source* and *Destination* columns show MAC addresses. These begin with **Intrepid** because the first three bytes of every MAC address are the organizationally unique identifier (OUI) of the device's hardware manufacturer; in this case the device is the Ethernet EVB, so the manufacturer is Intrepid Control Systems. Vehicle Spy 3 recognizes and automatically decodes many common OUIs to enhance readability.

Let's take a closer look at the message.

- **Display Message Signals:** Find the ⊞ button on the left of Ethernet Lab 1.3 Frame in the upper window pane and click it.

You will now see the three Ethernet header fields we looked at before, and also our custom data field, *Lab 1.3 Data*. The data field has the value **78**. Let's expand the message within the *Details View*.

- **Expand Ethernet Message Details:** Open the *Details View*, if necessary, by clicking the Details button. Then press the ⊞ button to the left of the Ethernet entry in the *Details View* pane.

You will see duplicated here the same Ethernet header information from the message summary above, along with the values in those headers. Notice also that in the center of the *Details View* we can now see *Name/Value* pairs for each of the fields in the message.

► **Select the Data Field:** In the middle part of the *Details View*, click on Lab 1.3 Data.

Notice that the byte 4E is highlighted in gray; this is the 15th byte of the message, corresponding to the (decimal) value of 78 we put in the *Lab 1.3 Data* field.

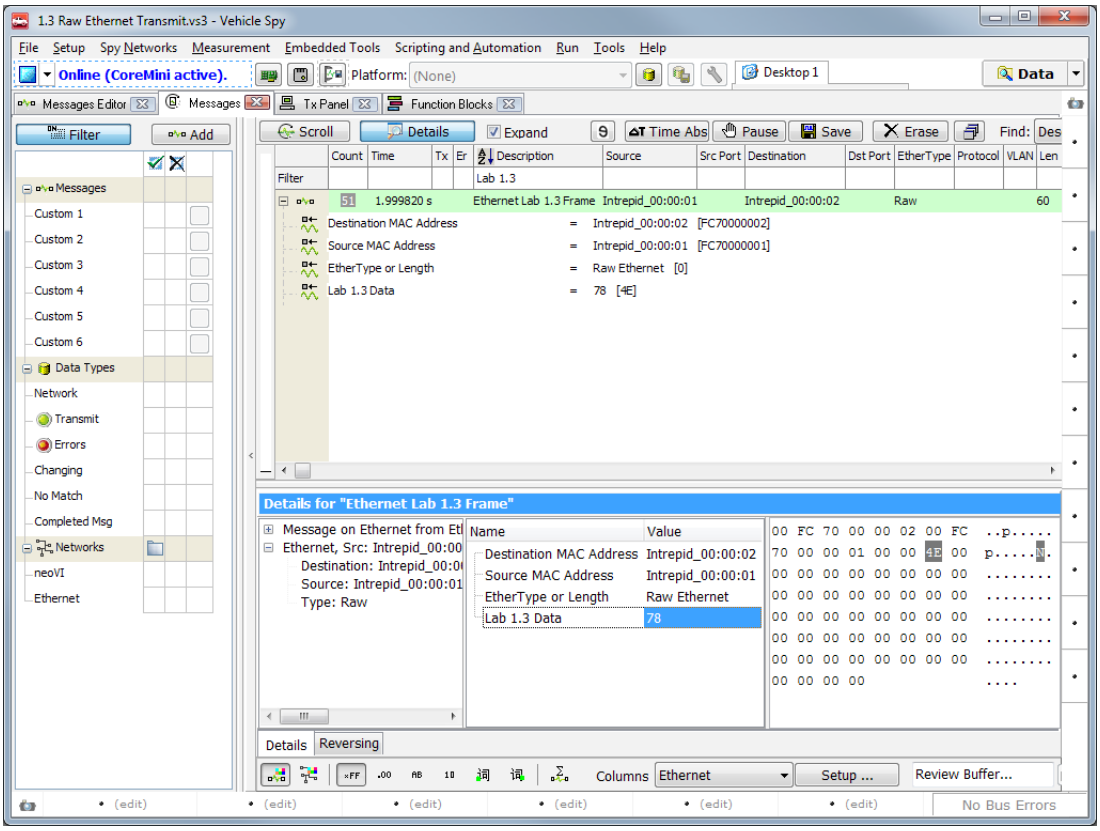


Figure 38: Ethernet Transmission Details. Here we have expanded the *Ethernet Lab 1.3 Frame* message and highlighted the *Lab 1.3 Data* field, showing its value of 78 (decimal) or 4E (hexadecimal).

Feel free to look around some more at the message details before proceeding to the next lab.

Lab 1.4 Reviewing and Modifying Ethernet Templates and Setup Files

We saw in the prior lab that Vehicle Spy 3 will conveniently provide default values for header fields in Ethernet messages, and that we can change these values if desired in the *Tx Panel*. However, there may be situations where you always want to use a particular set of field values, and needing to edit the fields in the *Tx Panel* each time would be inconvenient. In this lab we'll learn how to change the default values themselves by editing Vehicle Spy 3's Ethernet templates. As part of this process, we'll also learn how to modify VSpy setup files, using the one we created in the previous lab as a starting point.

An example of a global change that one might wish to make to default values would be customizing the default MAC addresses used in Ethernet frames. Vehicle Spy 3 comes set to use addresses beginning with the three-byte OUI assigned to Intrepid Control Systems, which is *00:FC:70*. In this lab, we'll modify the Intrepid value to the OUI *12:34:56* belonging to a fictional organization. We'll then adjust the previous demo to use the new value in source and destination addresses.

Part 1.4A Restart Vehicle Spy 3 and Load the 1.3 Raw Ethernet Transmit Setup File

Let's again restart Vehicle Spy 3 to ensure that we are beginning with a fresh slate, and then load the setup file we created in Lab 1.3.

- **Close Vehicle Spy 3:** Select *Exit* from the *File* menu or press *Alt+F4*.
- **Start Vehicle Spy 3:** Select Vehicle Spy 3 from the Windows Start Menu or click .

As we first saw in Lab 1.1, Vehicle Spy 3 remembers the last several setup files you have used, and lists them under the Recent tab on the *Logon Screen*. You should see *1.3 Raw Ethernet Transmit* here since you created it in the preceding lab.

- **Load the 1.3 Raw Ethernet Transmit Setup:** On the Recent tab, double-click the entry 1.3 Raw Ethernet Transmit to load the setup file from the previous lab.

Vehicle Spy 3 will automatically load the view you were on when you saved the file, which should be the Messages Editor if you followed the steps in Lab 1.3 exactly, but may be something different. Note the tabs along the top of the screen for the various windows used in making that setup file (Messages, Messages Editor, Tx Panel and Function Blocks).

Part 1.4B Verify Correct Operation of the Setup

We have now restored the previous operating environment, and EEVB Node A should still be running the same CoreMini from before as well. To confirm this, let's go online, switch into the *Messages View*, and take a look around.

- **Go Online:** Press to go online.
- **Switch to Messages View:** Click the Messages tab or select Messages from the *Spy Networks* menu.

► **Enter Message Type Filter:** Enter Lab 1.3 in the Filter row for the Description column.

You should once again see the *Ethernet Lab 1.3 Frame* message every 2 seconds. If you wish, examine its details to verify that it is the same as before.

Part 1.4C Examine and Modify Ethernet Template Values for Raw Ethernet Frames

Now let's take a look at where the default values in that message came from, and change them to use the new *12:34:56* OUI for MAC addresses in raw Ethernet frames.

► **Switch to Messages Editor:** Click the Messages Editor tab or select *Messages Editor* from the *Spy Networks* menu.

► **Open the Ethernet Packet Template Editor:** Click on the button (Figure 39:A). It can be found at the far right end of the row that contains the Receive and Transmit buttons.

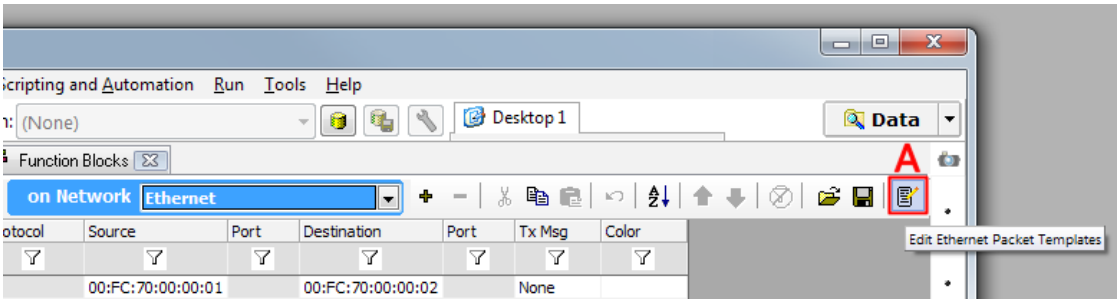


Figure 39: Ethernet Packet Template Editor Button. This image shows the right-hand part of the VSpy *Messages Editor* window. The Ethernet Packet Template Editor button is highlighted, along with the balloon that appears when you hover over it.

> **Note:** If you are using a small Vehicle Spy 3 window, you may need to widen it in order to see the Ethernet Packet Template Editor button.

After pressing the button, the *Ethernet Packet Template Editor* window appears. There are templates for dozens of message types and variations here; fortunately, the first in the list, *Raw Ethernet Frame Template*, happens to be the one we want. You can see here the standard three Ethernet frames and the values we saw previously in our custom message (Figure 40).

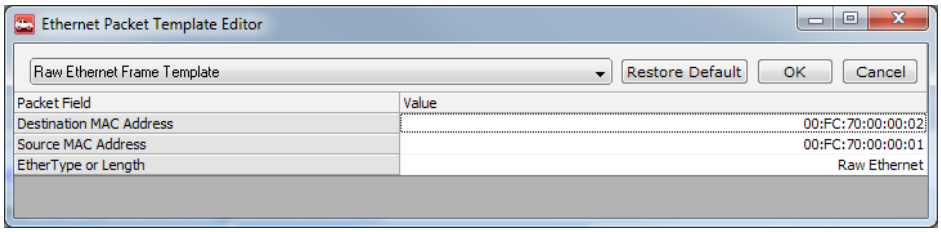


Figure 40: Ethernet Packet Template Editor. By default the editor starts with the raw Ethernet template selected, which conveniently is the one we want to work with. (Note that the editor window was reduced in height for this screenshot; it will normally open much larger to accommodate templates that have many more fields than the one for raw Ethernet.)

Let's change the values now.

- **Change the Default Destination MAC Address:** Double-click the *Value* field for the Destination MAC address; the current value will be highlighted. Press the *Home* key, then type 123456 and press *Enter*. Vehicle Spy 3 will automatically add the colons for you, resulting in the current value of 00:FC:70 being replaced by 12:34:56. The rest of the MAC address (00:00:02) will be retained without change.
- **Change the Default Source MAC Address:** Repeat this process for the source MAC address default; the end value should be 12:34:56:00:00:01.
- **Save Changes:** Click OK to save the changes for this template.

The field defaults have now been changed. If you like, you can enter the *Ethernet Packet Template Editor* again to verify that the new values have "stuck".

Part 1.4D Examine the Transmit Message

Let's go back and take a look at the transmit message we've already defined in this setup.

- **Switch to the Tx Panel:** Click the Tx Panel tab.
- **Select the Ethernet Lab 1.3 Frame Message:** Click on Ethernet Lab 1.3 Frame, just as you did in Part 1.3C.

Notice that the *Destination MAC Address* and *Source MAC Address* are the same as they were before. This is because we only changed the *defaults* for these values, so they will only appear in *new* raw Ethernet messages.

Part 1.4E Create New Transmit and Receive Messages Using New Default Values

Let's create a replacement for our current message that uses the new defaults. This process is similar to the steps we followed near the start of the last lab, but since you are now used to doing this, we won't spell everything out in quite as much detail. We'll save the setup under a new name so we preserve our old one.

- **Go Offline:** Press .
- **Switch to the Messages Editor, Transmit Side.**

- **Create a New Transmit Message:** Press the ✚ button to create a new Ethernet transmit message, and name it Ethernet Lab 1.4 Frame.

Notice in the summary line for the message that the *Source* and *Destination* columns contain our new defaults beginning with 12:34:56.

Key	Description	EtherType	VLAN	Protocol	Source	Port	Destination	Port	Len	Raw Payload Bytes
out0	Ethernet Lab 1.3 Frame	Raw	None		00:FC:70:00:00:01		00:FC:70:00:00:02			
out1	Ethernet Lab 1.4 Frame	Raw	None		12:34:56:00:00:01		12:34:56:00:00:02			

Figure 41: Ethernet Transmit Messages with Old and New Defaults. Above, the *Ethernet Lab 1.3 Frame* message containing the original Vehicle Spy 3 defaults; below, the new *Ethernet Lab 1.4 Frame* message that has been automatically set to the new defaults we set using the *Ethernet Packet Template Editor*.

- **Add a Data Field:** Click the button to add an 8-bit data field; name it Lab 1.4 Data.
- **Switch to the Tx Panel.**
- **Review and Edit Field Values:** Click on Ethernet Lab 1.4 Frame. You should see here as well that the MAC addresses begin with *12:34:56*. Enter the value 88 for the *Lab 1.4 Data* Value field.
- **Switch to the Messages Editor.**
- **Copy Transmit Message to Receive Side:** Click on the Ethernet Lab 1.4 Frame message, then right-click, choose Copy To, and then Receive. Click Receive to verify that the new message is present.

Part 1.4F Complete Modifications and Save New Setup File

We have a new message using our new default values, but we aren't done yet. For one thing, our function block script still refers to the old message; we also should get rid of the old transmit and receive messages, which we don't need any more. Finally, we should save this new setup so we can restore it later if needed.

- **Remove Old Messages:** Click and then right-click and delete both the transmit and receive versions of *Ethernet Lab 1.3 Frame*.
- **Switch to Function Blocks.**
- **Edit Function Block Script Transmit Step:** Double-click the Value field for step 1. Change the entry from Ethernet Lab 1.3 Frame to Ethernet Lab 1.4 Frame.
- **Update Transmit Step Comment:** For consistency, change 1.3 to 1.4 in the comment.
- **Save Setup File:** Select Save As from the *File* menu. When the dialog box appears, enter 1.4 Raw Ethernet Transmit 123456.

Part 1.4G Send New CoreMini to EEVB Node A and Go Online

All we have to do now is download our updated setup and go online to see the results.

- **Download CoreMini:** Enter the *CoreMini Console* and download the new CoreMini to Node A of your EEVB using the same process described in Part 1.3G.
- **Go Online:** Ensure that the EEVB is the selected Ethernet interface and press .
- **Switch to Messages View.**
- **Update Message Filter:** Change the value in the Filter row for the Description column from Lab 1.3 to Lab 1.4.

You should now see *Ethernet Lab 1.4 Frame* being received by Vehicle Spy 3 in the *Messages View* every two seconds (Figure 42). Examine the details to verify the new MAC addresses and that the data field is now *Lab 1.4 Data* with a value of 88 (or 58 in hexadecimal).

- **Go Offline.**

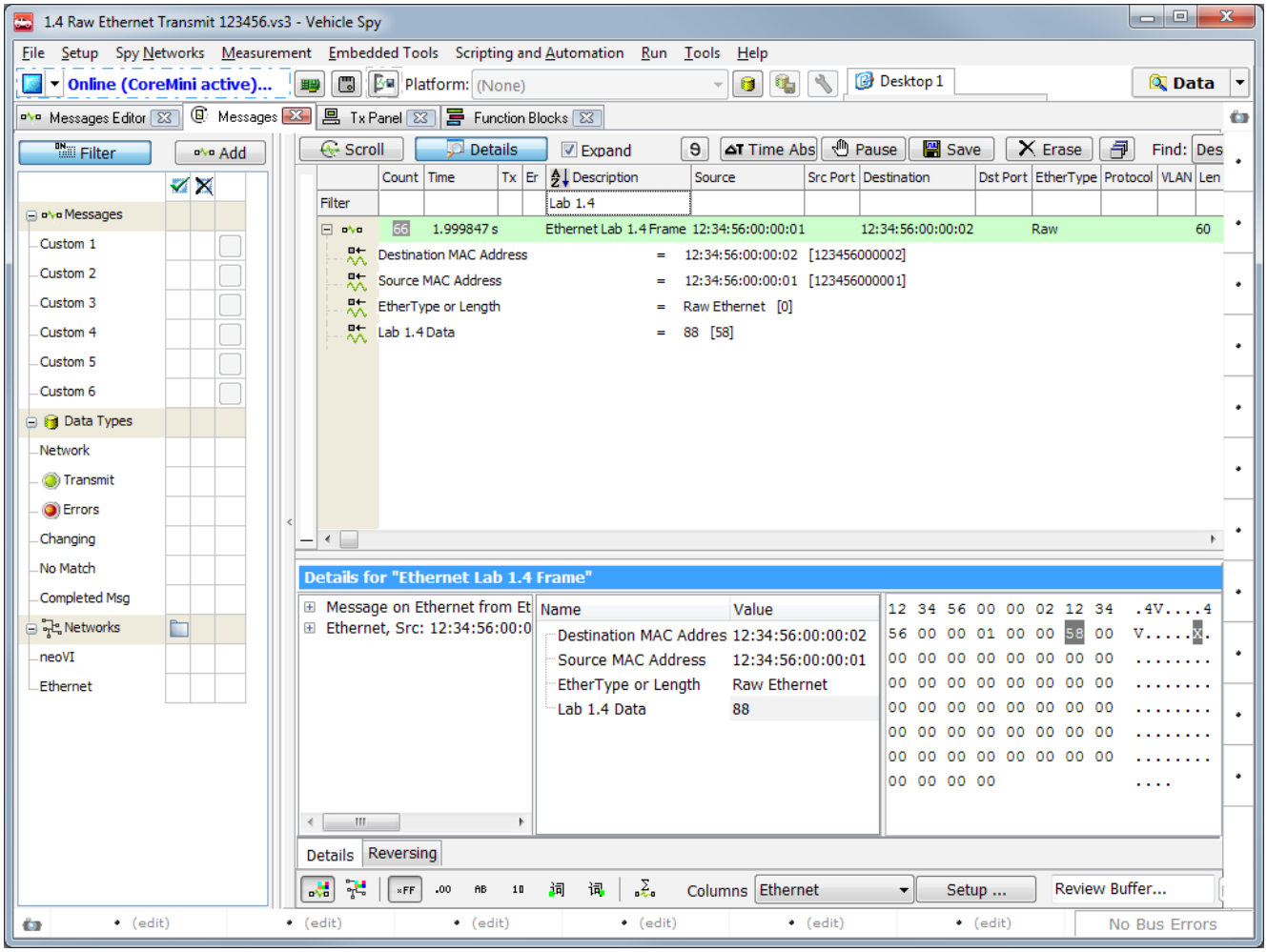


Figure 42: Modified Raw Ethernet Frames Received from EEVB. This image shows our modified raw Ethernet frames, containing MAC addresses beginning with 12:34:56, being displayed in Vehicle Spy 3 as they are received from the EEVB.

Part 1.4H Restore Default Ethernet Template Values for Raw Ethernet Frames

Changes made to Ethernet templates persist even when new setups are made or Vehicle Spy 3 is restarted. Since we're done with our lab now, and the rest of the Lab Manual assumes that we are using the preset values programmed into Vehicle Spy 3, we should change back the values we modified in Part 1.4C. Fortunately, this is easy to do, as VSpy includes a way to reset a template back to its original value—the "default default", if you will.

- **Open the Ethernet Packet Template Editor:** Click the Messages Editor tab or select it from the *Spy Networks* menu. Then click on the button.
- **Reset Raw Ethernet Packet Template:** Click the Restore Default button, and then click OK.

The original values beginning with 00:FC:70 will now be restored for the raw Ethernet template.

Lab 1.5 Setting Up a Transmission and Response Exchange Using Ethernet Frames

In Lab 1.3 we created a simple Vehicle Spy 3 setup file that tells one node of the EEVB to regularly transmit a single raw Ethernet message. We'll now build on that experiment by having one node transmit and the other respond to it. This represents a simplified version of the request/reply mechanism used by many networking protocols, which we'll explore further when we tackle ARP in Section 2 and other protocols later in the Lab Manual.

We will make use of both the VSpy setup file we created in Lab 1.3 and an additional pre-made one as well. One node will regularly transmit a raw Ethernet message containing a single byte of payload data; the other node will look for this message, read the data, compute double its value, and then transmit this as a reply. We'll then modify this functionality to make it a bit smarter, learning a little more about function block scripts in the process.

Part 1.5A Reset EEVB Node A to Raw Ethernet Transmit CoreMini from Lab 1.3

Assuming you followed the steps in Lab 1.4, Node A of your EEVB is now running a modified setup that uses the changed MAC addresses we dealt with in that lab. Let's restore the node back to running the original version from Lab 1.3.

- **Load Raw Ethernet Transmit Setup File:** Start Vehicle Spy 3 if it is not already running. Then from the *Logon Screen*, select *1.3 Raw Ethernet Transmit* from either the Recent or My Setups tabs.
- **Download CoreMini to Node A:** Enter the *CoreMini Console* and send the CoreMini to Node A of your EEVB. Please refer to the detailed procedure in Part 1.3G if you need a refresher on the exact steps required.

Part 1.5B Load Raw Ethernet Response Setup File and Send to EEVB Node B

Now let's load another setup file, this one containing messages and a function block script that we will send to EEVB Node B.

- **Load Raw Ethernet Response Setup File:** Go back to the *Logon Screen*. From the My Setups tab, double-click *1.5 Raw Ethernet Response*. Choose to discard changes if prompted.
- **Download CoreMini to Node B:** Go to the *CoreMini Console*. Click the drop-down box and select the entry corresponding to your EEVB's Node B (Figure 43). Then download the CoreMini.
- **Exit the CoreMini Console.**

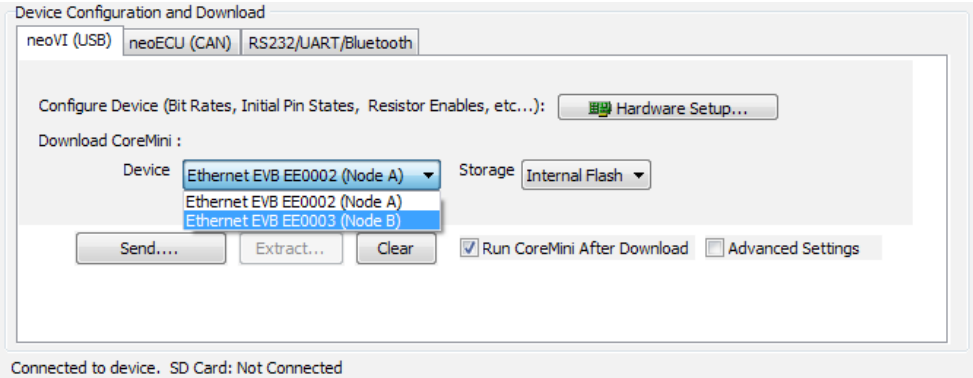


Figure 43: Selecting EEVB Node B in the CoreMini Console.

Part 1.5C Examine the Raw Ethernet Response Setup File

Before we go online, let's take a quick "guided tour" of this setup file to see how it works. Loading *1.5 Raw Ethernet Response* should have put you in the *Messages Editor*, on the receive side; if you are not there, please go there now.

You will see two messages here (Figure 44). The first, *Ethernet Lab 1.3 Frame*, is identical to the one we used in Lab 1.3, containing the same *Lab 1.3 Data* field, and is defined here to tell EEVB Node B what message to look for. The second is the response we will transmit, which contains the field *Lab 1.5 Response Data*; it is here on the receive side so that the messages we transmit will be properly decoded for display in the *Messages View*.

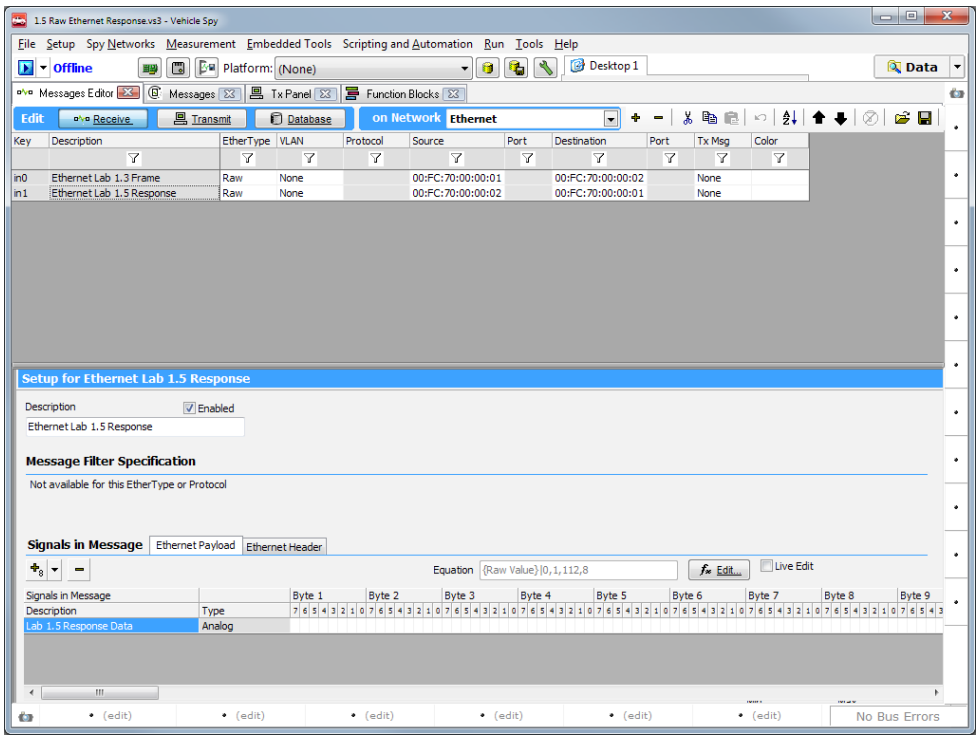


Figure 44: Receive Messages for Ethernet Lab 1.5 Response Setup. The response setup has two messages defined in the receive area: the message we are looking for from Node A, and the message we are sending from Node B.

Notice that *Ethernet Lab 1.3 Frame* has a *Source* of 00:FC:70:00:00:01 and a *Destination* of 00:FC:70:00:00:02, while *Ethernet Lab 1.5 Response* has the opposite MAC addresses.

► **Switch to the Transmit Side.**

Here we just see our response message, identical to the version on the receive side.

► **Review Function Block Script:** Click the Function Blocks tab or select Function Blocks from the *Scripting and Automation* menu.

The script is slightly more complex than the one we used in Lab 1.3, but still quite straightforward (Figure 45). Basically, all it does is wait until an *Ethernet Lab 1.3 Frame* message is received from Node A, and then transmit the response message. Take a look at the comments to see what happens in each step.

Step	Description	Value	Comment
1	Wait Until	{Ethernet Lab 1.3 Frame (Present) :in0-0}	// Don't do anything until we see an Ethernet Lab 1.3 Frame transmission from Node A.
2	Set Value	{Ethernet Lab 1.3 Frame (Present) :in0-0} = 0	// When we do see a message, set the "Present" flag to 0 so this script is not triggered again until the next instance of the message.
3	Set Value	{Lab 1.5 Response Data (Value) :out0-sig3-0} = {Lab 1.3 Data (Value) :in0-sig3-0}*2	// Compute the value of the Lab 1.5 Reply Data field in the Response message as double the value of the Lab 1.3 Data field in the message from Node A.
4	Transmit	Ethernet Lab 1.5 Response	// Transmit the response.

Figure 45: Function Block Steps for Ethernet Lab 1.5 Response Setup. The script waits for a message from Node A, fills in the data field with double the value received, and then transmits.

Part 1.5D Go Online to View Transmission and Response Messages

Now let's see our two nodes in action!

► **Switch to Messages View.**

► **Enter Message Filter:** Enter Lab in the Filter row for the Description column.

► **Go Online:** Press .

You should see the *Ethernet Lab 1.3 Frame* and *Ethernet Lab 1.5 Response* messages, with the count increasing every 2 seconds.

> **Note:** If you see only *Ethernet Lab 1.3 Frame* and not *Ethernet Lab 1.5 Response*, this may be because you forgot to download the CoreMini to Node B. Also, be sure that the BroadR-Reach cable is connecting the two nodes, as this is the mechanism by which Node A's message is received by Node B. Finally, try pressing and releasing the CPU reset button for Node B (component B13 in Figure 2 of the EEVB User's Guide).

Let's expand the messages for a closer look.

► **Expand the Two Messages:** Click the ⊞ button to the left of the *Ethernet Lab 1.3 Frame* and *Ethernet Lab 1.5 Response* screen entries in the main *Messages View*.

You should see a display similar to Figure 46. Notice that the value of *Lab 1.3 Data* is 78, while that of *Lab 1.5 Response Data* is 156.

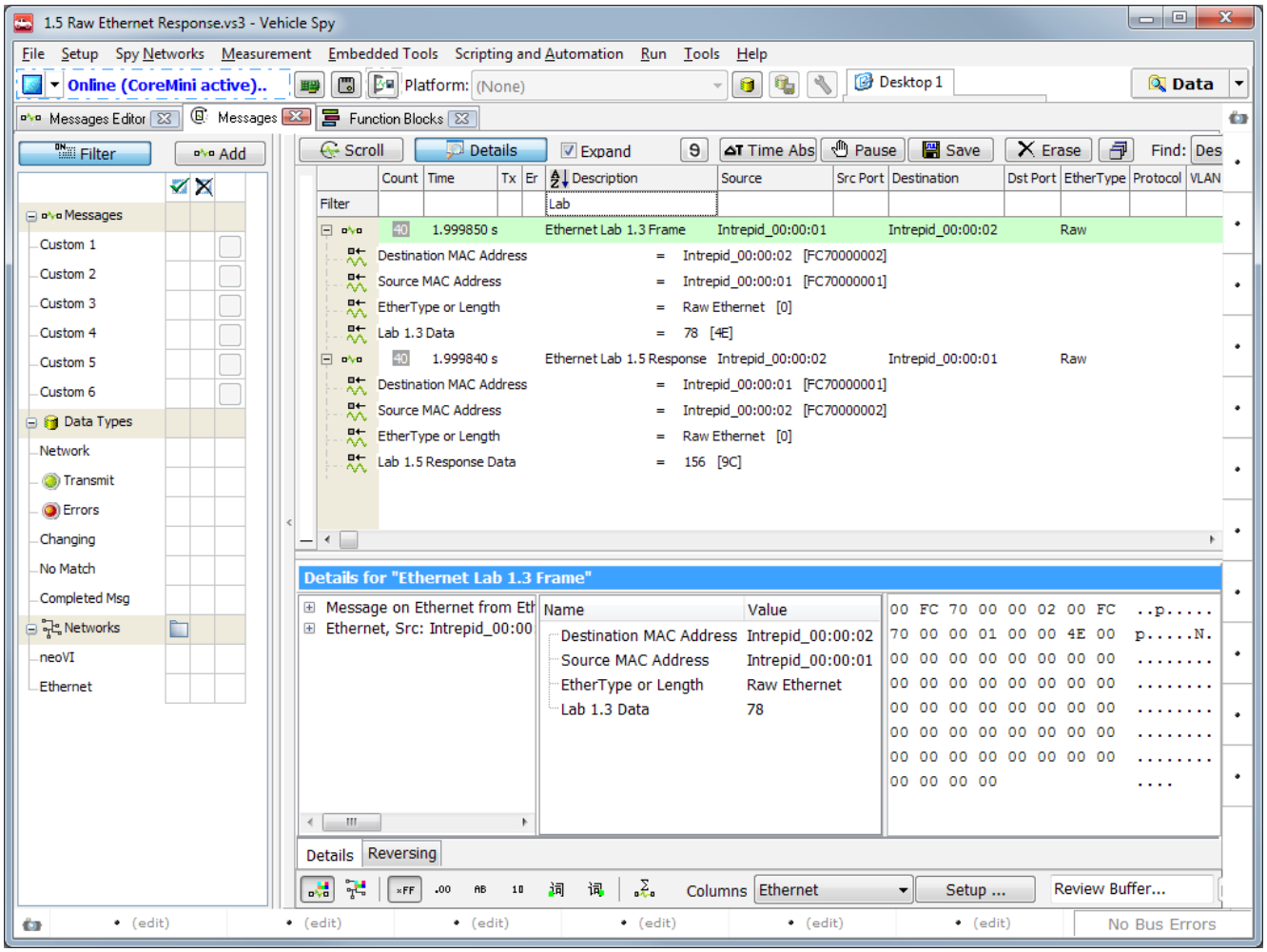


Figure 46: Message Details for Ethernet Lab 1.5 Response. Vehicle Spy 3 sees the original messages sent by EEVB Node A and the responses sent by Node B.

Now let's look at the messages in sequential order so we can see the request/reply nature of the exchange more clearly.

► **Collapse the Two Messages:** Click the ⊟ buttons next to each message to hide their signals.

► **Enter Scroll Mode:** Click the Scroll button to enter scroll mode.

You can now see that the *Ethernet Lab 1.3 Frame* message is sent about every two seconds, and the *Ethernet Lab 1.5 Response* message follows in just a few hundreds of microseconds after it (Figure 47).

	Line	Time	Tx	Er	Description	Source	Src Port	Destination	Dst Port	EtherType
Filter					Lab					
	263	2.001679 s			Ethernet Lab 1.3 Frame	00:FC:70:00:00:01		00:FC:70:00:00:02		Raw
	264	174 µs			Ethernet Lab 1.5 Response	00:FC:70:00:00:02		00:FC:70:00:00:01		Raw
	265	2.001686 s			Ethernet Lab 1.3 Frame	00:FC:70:00:00:01		00:FC:70:00:00:02		Raw
	266	183 µs			Ethernet Lab 1.5 Response	00:FC:70:00:00:02		00:FC:70:00:00:01		Raw
	267	2.001699 s			Ethernet Lab 1.3 Frame	00:FC:70:00:00:01		00:FC:70:00:00:02		Raw
	268	182 µs			Ethernet Lab 1.5 Response	00:FC:70:00:00:02		00:FC:70:00:00:01		Raw
	269	2.001669 s			Ethernet Lab 1.3 Frame	00:FC:70:00:00:01		00:FC:70:00:00:02		Raw
	270	198 µs			Ethernet Lab 1.5 Response	00:FC:70:00:00:02		00:FC:70:00:00:01		Raw
	271	2.001676 s			Ethernet Lab 1.3 Frame	00:FC:70:00:00:01		00:FC:70:00:00:02		Raw
	272	186 µs			Ethernet Lab 1.5 Response	00:FC:70:00:00:02		00:FC:70:00:00:01		Raw
	273	2.001679 s			Ethernet Lab 1.3 Frame	00:FC:70:00:00:01		00:FC:70:00:00:02		Raw
	274	229 µs			Ethernet Lab 1.5 Response	00:FC:70:00:00:02		00:FC:70:00:00:01		Raw

Figure 47: Ethernet Lab 1.5 Response Message Timing. Node A sends a frame about every two seconds, while Node B sends its response very quickly after each Node A transmission is received.

- **Enter Static Mode:** Click Scroll again to switch from scroll mode to static mode.

Part 1.5E Investigate the Impact of Changes to Receive Message Parameters

The transmission of a response message is triggered by Node B recognizing the messages sent by Node A. What happens if we change the definition of the receive message?

- **Go Offline.**
- **Switch to Messages Editor, Receive Side.**
- **Edit the Source MAC Address Field of the Ethernet Lab 1.3 Frame Receive Message:** Double-click the *Source* entry for *Ethernet Lab 1.3 Frame*, and then change the last digit from 1 to 3.
- **Download CoreMini:** Send the CoreMini to Node B.
- **Go Online.**
- **Switch to Messages View.**

Now instead of both messages, we have neither... or do we?

- **Change Message Filter:** Change Lab in the Filter row to 00:FC:70.

You'll now see that we are still getting a frame every 2 seconds, which is in fact *Ethernet Lab 1.3 Frame*. It isn't being recognized as such by Vehicle Spy 3 because we changed the definition of that message to only match on a source address of 00:FC:70:00:00:03. There is also only one message because Node B isn't recognizing the message as an *Ethernet Lab 1.3 Frame*. This means the function block is never triggered, and so the *Ethernet Lab 1.5 Response* message is never sent.

Let's undo these changes so we are back where we started.

- **Go Offline.**

- **Edit the Ethernet Lab 1.3 Frame Receive Message Again:** Go back to the *Messages Editor* and change the *Source* entry for *Ethernet Lab 1.3 Frame* back to 00:FC:70:00:00:01.
- **Send CoreMini to Node B.**
- **Switch to Messages View.**
- **Change Message Filter:** Change the Description column filter back to Lab.
- **Go Online.**

Verify that the original behavior of the two nodes has been restored.

- **Go Offline.**

Lab 1.6 Adding Intelligence and Control to Ethernet Transmission and Response Exchanges

So far we have only looked at function block scripts that are entirely linear in nature: they do the same thing every time. However, Vehicle Spy 3 actually supports a powerful array of instructions to let you build rather complex and intelligent programs. One of the most important sets of commands is the "*If .. Else .. End If*" construct, which allows you to add conditional decision-making to your scripts. We also have input devices on the EEVB that we can exploit to allow us to have direct control over the exchange process. Let's put these features to work.

Part 1.6A Load Raw Ethernet Response Setup File, If Necessary

If you are doing this lab right after Lab 1.5, your EEVB is likely already set up correctly and you can skip this part. If you aren't continuing from the previous lab, or if you want to be absolutely sure that the EEVB is properly configured for this one, follow these steps.

- **Load Raw Ethernet Transmit Setup File:** From the *Logon Screen*, select 1.3 Raw Ethernet Transmit from either the Recent *or* My Setups tabs.
- **Download CoreMini to Node A:** Go to the *CoreMini Console*. Select Node A and download the CoreMini to that node.
- **Load Raw Ethernet Response Setup File:** Go back to the *Logon Screen* and load 1.5 Raw Ethernet Response.
- **Download CoreMini to Node B.**
- **Switch to Messages View.**
- **Enter Message Filter:** Set the Description column filter to Lab.
- **Go Online.**

Verify that the two nodes are operating as they did in Lab 1.5.

- **Go Offline.**

Part 1.6B Add a Conditional Statement to the Response Script

Let's change how we determine the value of the *Lab 1.5 Response Data* field. Instead of always setting it to double the value of the incoming *Lab 1.3 Data* field, we will double the value if it is greater than 127, and halve it otherwise.

- **Switch to Function Blocks:** Click the Function Blocks tab.
- **Add If Statement:** Click *Step 3*. Press the Before button (just above the script steps) to add a new step. Then select If under the Description field. Ignore the error that appears.

- **Enter If Statement Condition:** Double-click on the Value field, which currently contains Equation Not Set. The *Expression Editor* appears. Select Rx Messages from the menu on the left side if it is not already highlighted. Then, under Ethernet Lab 1.3 Frame, double-click Lab 1.3 Data; this will cause values to appear in the Description and Expression fields above. Click at the end of the Expression field, after the curly brace. Add < 128 to the end of the field, then press *Enter* or click OK.
- **Enter If Statement Comment:** Add this comment to the new command's Comment field: If incoming Lab 1.3 Data field is less than 128, send back double the value..
- **Add Else Step:** Click *Step 5*, currently the last command in the script. Press the Before button and select Else in the Description field.
- **Enter Else Statement Comment:** Add this comment to the new statement: If incoming data field is greater than or equal to 128, send back half the value..
- **Add End If Statement:** Click on *Step 6*, the last command again, and press Before once more. Choose End If in the Description field.

Any errors should disappear once the *End If* statement is added. However, we are not quite done yet: we already have the command to compute double the incoming value when it is below 128, but not the one that halves it when it is 128 or higher. We could write this step, but why bother entering a new step from scratch when it is much easier to just duplicate the existing command and make a small change?

- **Copy and Paste Step 4:** Click *Step 4* and press the button. Then click *Step 6* and press .

You will now have a duplicate of the *Set Value* command between the *Else* and *End If*.

- **Edit Second Set Value Statement:** Double-click the Value field for *Step 6*. In the Expression field of the *Expression Editor*, change the * (for multiplication) to / (for division) and save the changes. In the Comment field, change double to half.

Your script should now appear similar to Figure 48 (though there may be superficial differences depending on screen size and column widths).

Step	Description	Value	Comment
1	Wait Until	{Ethernet Lab 1.3 Frame (Present) :in0-0}	// Don't do anything until we see an Ethernet Lab 1.3 Frame transmission from Node 0.
2	Set Value	{Ethernet Lab 1.3 Frame (Present) :in0-0} = 0	// When we do see a message, set the "Present" flag to 0 so this script is not triggered again until the
3	If	{Lab 1.3 Data (Value) :in0-sig3-0} < 128	// If incoming Lab 1.3 Data field is less than 128, send back double the value.
4	Set Value	{Lab 1.5 Response Data (Value) :out0-sig3-0} = {Lab 1.3 Data (Value) :in0-sig3-0}*2	// Compute the value of the Lab 1.5 Reply Data field in the Response message as double the value of the Lab 1.3 Data field in the message from Node 0.
5	Else		// If incoming data field is greater than or equal to 128, send back half the value.
6	Set Value	{Lab 1.5 Response Data (Value) :out0-sig3-0} = {Lab 1.3 Data (Value) :in0-sig3-0}/2	// Compute the value of the Lab 1.5 Reply Data field in the Response message as half the value of the Lab 1.3 Data field in the message from Node 0.
7	End If		
8	Transmit	Ethernet Lab 1.5 Response	// Transmit the response.

Figure 48: Function Block Steps for Ethernet Lab 1.5 Conditional Setup. Four additional commands introduce some "smarts" into our response script.

Let's save this so we can get back here quickly when needed.

- **Save Setup File:** Choose *Save As* from the *File* menu and enter the name 1.6 Raw Ethernet Response Conditional.

Part 1.6C Download New Response Script to EEVB and Change Transmit Script to Test Operation

Okay, let's get this new script into the EEVB and see how it works.

- **Download CoreMini to Node B.**
- **Switch to Messages View.**
- **Enter Static Mode (If Necessary).**
- **Go Online.**
- **Expand the Ethernet Lab 1.3 Frame and Ethernet Lab 1.5 Response Messages.**

Let's take a look... hey, nothing changed: the Lab 1.3 Data value is still 78 and Lab 1.5 Response Data is still 156. Well, that's not a big surprise: 78 is less than 128, so it is still being doubled like it was before. Let's change that 78 to a larger number to see what happens.

- **Go Offline.**
- **Load Raw Ethernet Transmit Setup File:** Load the *1.3 Raw Ethernet Transmit* setup file from the *Logon Screen*.
- **Switch to Tx Panel.**
- **Change Lab 1.3 Data Field Value:** Click Ethernet Lab 1.3 Frame on the left. Then click on the value 78 and change the number to 178.
- **Download CoreMini to Node A:** Be sure to remember to change the node, since we most recently downloaded to Node B.

We've now changed our transmitted message. Let's load the response setup file again (since it has all the message definitions we need) and then go online.

- **Reload Conditional Raw Ethernet Response Setup File:** Go back to the *Logon Screen* and reload *1.6 Raw Ethernet Response Conditional*, the file we saved earlier with our changed script. You can discard changes when prompted.
- **Switch to Messages View.**
- **Enter Message Filter:** Set the Description column filter to Lab.
- **Go Online.**

Check the value of *Lab 1.5 Response Data* and you will see that it is 89, of course half of 178; the Details View for the *Ethernet Lab 1.5 Response* message can be seen in Figure 49.

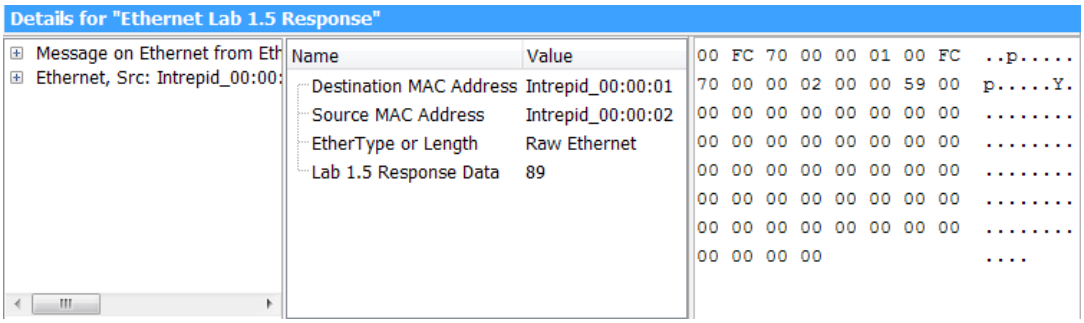


Figure 49: Details View for Conditional Ethernet Lab 1.5 Response Message.

Part 1.6D Add Pushbutton Control to the Response Script

As we saw in the User's Guide demo, each node on the EEVB has a pushbutton that can be used to provide input to Vehicle Spy 3. Let's change our program so that a response is only sent when the pushbutton on Node B is held down.

We're assuming you are continuing from the preceding example, so you should already have the *1.5 Raw Ethernet Response Conditional* setup file in Vehicle Spy 3. If not, load it now.

To implement this new functionality we only need to add a single "*If* .. *End If*" pair around the command that transmits the response message. Let's do it!

- **Go Offline:** Always a good idea as some areas of VSpy can't be changed while online.
- **Switch to Function Blocks.**
- **Add If Statement:** Click *Step 8* of the script, which should currently be the *Transmit* command. Click Before and then select If under the Description field.

The values of the EEVB pushbuttons are accessible within Vehicle Spy 3 as *Physical IO* inputs, and are considered *switches*.

- **Enter If Statement Condition:** Double-click on the Value field, and the *Expression Editor* appears. Select Physical IO from the left-hand menu, scroll down to Switches and expand it using the ⊞ button. Double-click Switch 1; the Expression field's value will become **{Switch 1 (Value) :neo0-sw0-0-index(0)}**. Press *Enter* or click OK.
- **Enter If Statement Comment:** Add this comment: Only transmit if the node's pushbutton is down..

> **Note:** A conditional expression normally requires a value to compare against, but this is a special case. An EEVB pushbutton's value is 0 normally, and 1 when it is held down. We want to transmit when it is down, so we would expect to need "= 1" in our condition. However, Vehicle Spy 3 evaluates conditions as true when 1 and false when 0, so the "= 1" would be redundant.

► **Add End If Statement:** Click the *Transmit* step again (now *Step 9*) and this time press After. Select End If for this program step.

The revised script should have 10 steps and look something like Figure 50.

Step	Description	Value	Comment
1	Wait Until	{Ethernet Lab 1.3 Frame (Present) :in0-0}	// Don't do anything until we see an Ethernet Lab 1.3 Frame transmission from Node 0.
2	Set Value	{Ethernet Lab 1.3 Frame (Present) :in0-0} = 0	// When we do see a message, set the "Present" flag to 0 so this script is not triggered again until the
3	If	{Lab 1.3 Data (Value) :in0-sig3-0} < 128	// If incoming Lab 1.3 Data field is less than 128, send back double the value.
4	Set Value	{Lab 1.5 Response Data (Value) :out0-sig3-0} = {Lab 1.3 Data (Value) :in0-sig3-0}*2	// Compute the value of the Lab 1.5 Reply Data field in the Response message as double the value of the Lab 1.3 Data field in the message from Node 0.
5	Else		// If incoming data field is greater than or equal to 128, send back half the value.
6	Set Value	{Lab 1.5 Response Data (Value) :out0-sig3-0} = {Lab 1.3 Data (Value) :in0-sig3-0}/2	// Compute the value of the Lab 1.5 Reply Data field in the Response message as half the value of the Lab 1.3 Data field in the message from Node 0.
7	End If		
8	If	{Switch 1 (Value) :neo0-sw0-0-index(0)}	// Only transmit if the node's pushbutton is down.
9	Transmit	Ethernet Lab 1.5 Response	// Transmit the response.
10	End If		

Figure 50: Function Block Steps for Ethernet Lab 1.5 Conditional Setup with Pushbutton Control.

Now let's save our changes. We'll use a new filename again—it's good practice to preserve older files in case you want to undo your changes.

► **Save Setup File:** Select Save As from the *File* menu. When prompted, enter the name 1.6 Raw Ethernet Response Conditional Pushbutton.

Part 1.6E Send Response Script to EEVB and Check Operation

Let's send our revised script to Node B of the EEVB.

► **Download CoreMini:** Download the CoreMini to Node B. The CoreMini Console is likely set to Node A from earlier lab steps, so check it and change it if necessary.

► **Switch to Messages View.**

► **Go Online.**

You should see *Ethernet Lab 1.3 Frame* messages but no responses.

► **Hold Down the Pushbutton for Node B:** This is component B09 in Figure 2 of the EEVB User's Guide.

Now *Ethernet Lab 1.5 Response* messages should start appearing, and will continue until you release the pushbutton.

Congratulations, you've completed Section 1 of the Intrepid Ethernet EVB Lab Manual!

Section 2 Experiments with the TCP/IP Address Resolution Protocol (ARP) Over Ethernet

Now that we've got basic Ethernet message traffic analysis and generation under our collective belts, it's time to start our extensive look at how the TCP/IP family of protocols work in an Automotive Ethernet environment. And we'll begin this beginning with the *Address Resolution Protocol* (*ARP*), which is used on Ethernet networks to allow devices to discover their neighbors' MAC addresses to enable message transmission.

We start with ARP rather than a more "defining" protocol such as IP for two reasons. First, ARP is one of the simplest to understand of the TCP/IP protocols, making it easy for us to understand real ARP messages and to simulate the operation of the protocol within Vehicle Spy 3 and the Ethernet EVB. Second, despite this simplicity—and despite now being over 30 years old—ARP is still a widely-used protocol on TCP/IP+Ethernet networks, making learning about it relevant to any student of Automotive Ethernet.

In this section you will accomplish these goals:

- Learn how ARP works on modern networks and observe it in action.
- Load, examine and run a script on an EEVB node to periodically generate *ARP Request* messages.
- Set up the other EEVB node to listen for and respond to *ARP Request* messages with *ARP Reply* messages when appropriate.
- Learn more about using conditional expressions and values within function blocks.
- Use application signals, custom variables that provide extra flexibility to function block scripts.
- Understand better how Vehicle Spy 3 interprets and recognizes various messages.
- Dock VSpy windows so you can work with more than one view at a time.
- See how to have the Ethernet EVB send messages to the PC instead of to the other EEVB node.
- Work with two Vehicle Spy 3 instances at the same time.
- Use the Ethernet EVB, Vehicle Spy 3 and a RAD-Moon media converter to perform bidirectional message exchange. (Requires optional hardware.)

This section builds upon what you learned in Section 1 of the Lab Manual. Going forward we assume you already know how to switch among common Vehicle Spy 3 views and windows, including the *Messages View*, *Messages Editor*, *Tx Panel* and *Function Blocks* area, so these tasks aren't spelled out. You should also be familiar with how to start and stop Vehicle Spy 3, go online and offline, load and save VSpy setup files, and download a CoreMini to your EEVB. If you need a refresher on any of these, please review the appropriate parts of Section 1.

An Overview of the Address Resolution Protocol

The following brief description of ARP is intended to provide some basic background for those new to TCP/IP. For a more thorough explanation of the protocol, please refer to Chapter 14 of *Automotive Ethernet - The Definitive Guide*.

Communication among network devices is accomplished using addresses that indicate where they are located, much like street addresses. However, most networks use two address types: one at a lower level linked to actual hardware, and one at a higher, more abstract level. In the case of Automotive Ethernet, these are usually *MAC addresses* and *IP addresses* respectively.

IP addresses are used for logical message exchanges between devices on internetworks such as the Internet. For example, a Web browser running on a tablet in Germany can make a request to a server in Brazil using the server's IP address; the server will use the tablet's IP address to send back the requested Web page. While virtual in concept, this communication is implemented as many hardware-level transactions, since that's where all data transfers occur. Thus, each step requires a hardware-level address. On a TCP/IP+Ethernet network there is a constant need to translate between IP addresses and MAC addresses, and this is ARP's job.

Suppose Device A has a TCP/IP message for Device B, a neighbor on its AE network. It must look up Device B's IP address in a special table to find its MAC address, and if it is not there, then attempt to determine the address using the following simplified exchange (Figure 51):

- **ARP Request:** Device A broadcasts an *ARP Request* message to all other devices on the local network that says "What is the MAC address of Device B?"
- **ARP Reply:** All devices receive the broadcast and check the IP address in the request against their own IP addresses. Device B will find a match and then send an *ARP Reply* back directly to Device A saying "That is my IP address and here is my MAC address.".

Now Device A can deliver the message using Ethernet. It will do so, and update its local table (called an *ARP cache*) so that the next time it needs to send to Device B, it will have that device's MAC address already. And that's ARP in a nutshell!

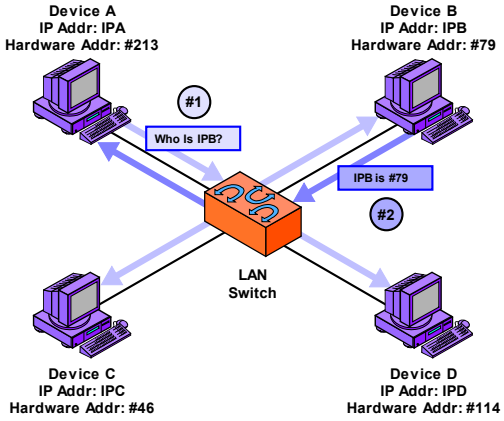


Figure 51: Dynamic Address Resolution Using ARP. Device A needs to send data to Device B but knows only its IP address (IPB) and not its hardware address (#79). A broadcasts an *ARP Request* asking to be sent the hardware address of the device using the IP address IPB. B responds back to A directly with an *ARP Reply* containing its hardware address.

Lab 2.1 Observing ARP in Action

We'll get the ball rolling in Section 2 in much the same way we did in Section 1: not using the Ethernet EVB yet, but first looking at "real world" ARP messages. This will help us build familiarity with how the protocol works before we start using it in active scenarios.

As was the case with Lab 1.1, be sure you are connected to a network of some sort before proceeding or you won't see any ARP messages.

Part 2.1A Go Online and Filter to See ARP Messages

Let's begin with a fresh new instance of Vehicle Spy 3, to clear out any settings from previous labs.

- **Close Vehicle Spy 3.**
- **Start Vehicle Spy 3.**

Now let's go online and start collecting Ethernet traffic. For convenience, let's load the custom column setup file we created in Lab 1.1 so that the *Messages View* is ready to show Ethernet traffic.

- **Select Active Ethernet Interface:** On the *Logon Screen*, click the radio button next to an active (non-EEVB) Ethernet interface, such as the device the PC uses to connect to the Internet.
- **Load Custom Column Setup File:** Load *1.1 Custom Column Setup* from the Recent tab, or if it is not there, under My Setups.
- **Go Online.**

You should automatically be switched to *Messages View*. As we did before in Section 1, let's tailor the view so we can more easily spot the messages we are after while avoiding those we do not care about.

- **Set ARP EtherType Filter:** Enter ARP in the EtherType column filter box.

Vehicle Spy 3 will now only show us ARP messages, with all others hidden. You should see a few of them, with the number increasing slowly over time. Since ARP is a request/reply protocol, however, the best way to observe it is to see the messages in sequential order—a job tailor-made for the *Messages View* scroll mode:

- **Enable Scroll Mode:** Press the Scroll button.

Vehicle Spy 3 will show only new ARP messages that arrive, but older non-ARP messages will still be present. Let's do some clutter reduction:

- **Clear Messages View:** Press Erase to clear the *Messages View*.

You will now see only ARP messages on the screen (Figure 52). Notice that you will generally see the ARP messages arrive in pairs, exactly as expected, one request and one reply each. However, this will not always be the case, as sometimes a request may not receive a reply.

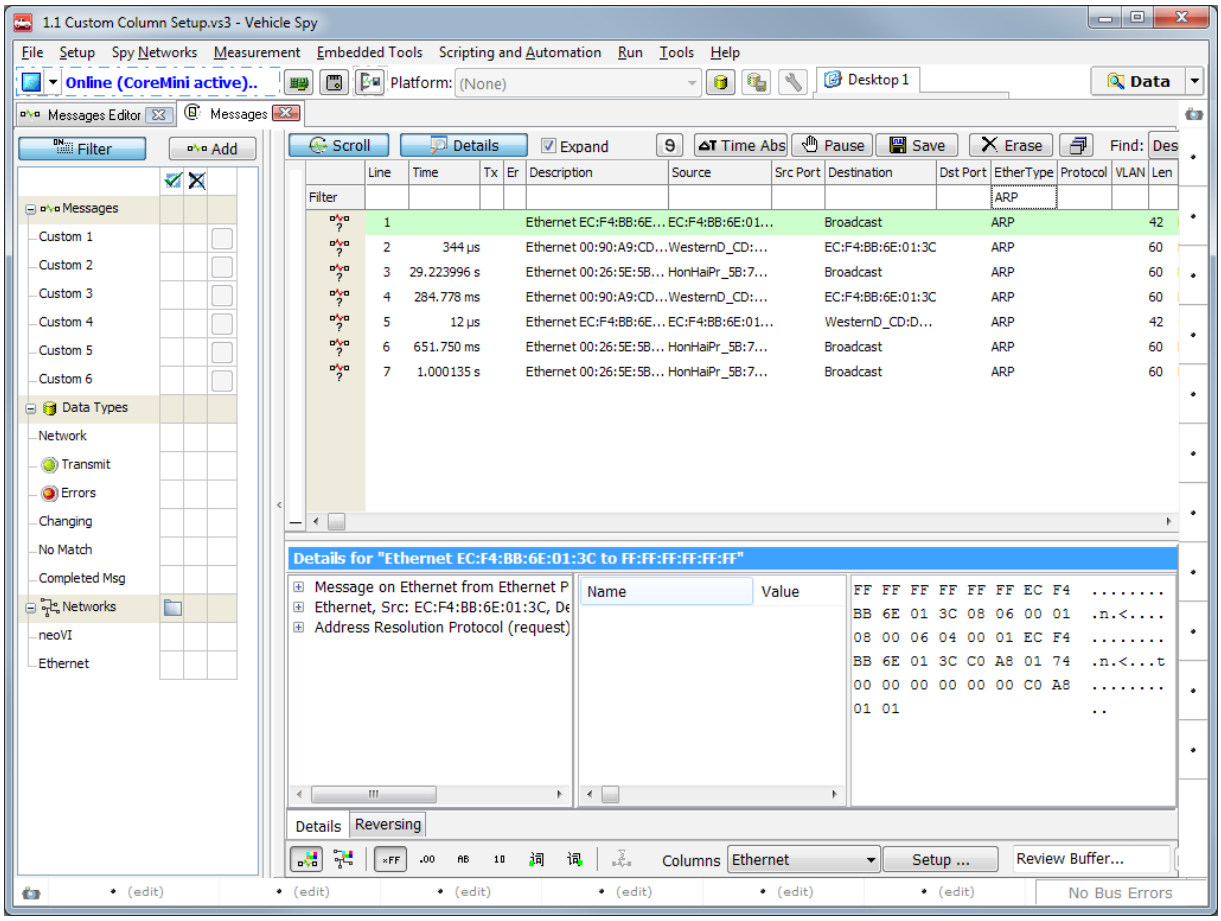


Figure 52: "Real World" ARP Message Traffic. This image shows seven ARP messages received on a standard Ethernet interface connected to the Internet. Notice that the last message is a duplicate of the one before it; the second was likely sent because the transmitting device did not receive an *ARP Reply* within a full second after sending the first *ARP Request*.

Part 2.1B Manually Generate ARP Traffic on an Internet Connection

ARP messages are required for the proper operation of an Ethernet network. However, since they only carry control information and not data, they represent *overhead* in the network. This is why it takes several seconds to witness even a few ARP messages on a real network—ARP caches are used so that ARP messages need only be sent when absolutely necessary.

The operation of the ARP cache on a Windows PC can be managed using a built-in program called, unsurprisingly, *arp*. This utility allows the ARP cache on a machine to be examined and manipulated. We can even delete the entire ARP cache with a command. If we do so, the empty cache will cause the very next attempt to send data from the computer to generate an *ARP Request* message, prompting an *ARP Reply* in return. Thus, using the arp tool, we can conveniently generate an ARP message exchange any time we wish.

- **Open Command Prompt:** In the Windows Start Menu, enter cmd in the box labeled *Search programs and files*. When the program *cmd.exe* appears, right-click it and choose Run as administrator (Figure 53).

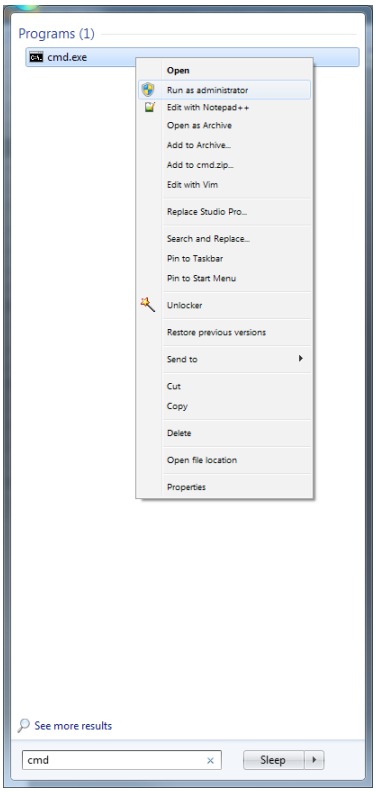


Figure 53: Opening a Windows Command Prompt.

- **Clear Messages View:** With Vehicle Spy 3 still online and in scroll mode, press Erase so we can more easily see the ARP messages we are about to generate.
- **Delete ARP Cache:** In the command prompt, enter arp -d * and then press the *Enter* key.

You should immediately see *ARP Request* and *ARP Reply* messages show up in Vehicle Spy 3, as the PC attempts to find the MAC address of the router or other device on the network through which it connects to the Internet (Figure 54).

- **Go Offline.**
- **Close Command Prompt:** Enter exit in the command prompt and press *Enter* to close the box.

We'll use the ARP messages we generated here in the next part of the lab. If you need to generate another exchange, you can just repeat the steps above.

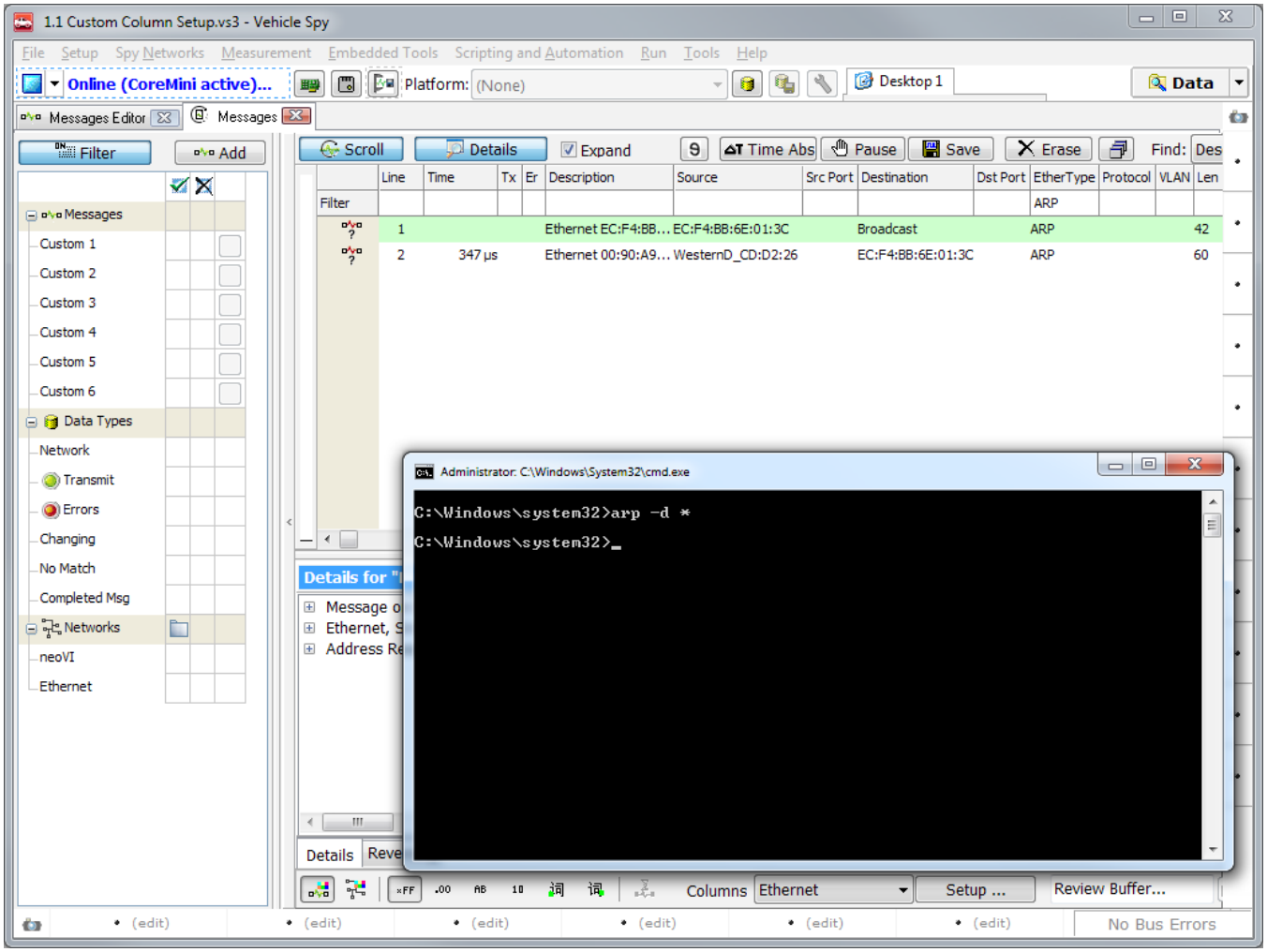


Figure 54: Deleting the ARP Cache to Force an ARP Request/Reply Exchange. Immediately after clearing the PC's ARP cache using the arp command shown, Vehicle Spy 3 records the computer sending an *ARP Request* and receiving an *ARP Reply*.

Part 2.1C Analyze an ARP Request / ARP Reply Message Exchange

We'll now take a closer look at the details of the *ARP Request* and *ARP Reply* messages we just generated, and which are displayed in Figure 54. This will help us better understand how these messages work in later labs, where we program the EEVB to simulate ARP behavior.

Let's start by taking a look at the *Source* and *Destination* fields for the two messages. As mentioned earlier, a standard *ARP Request* message is sent to the special MAC address reserved for broadcasts, which is **FF:FF:FF:FF:FF:FF**. (This is shown by Vehicle Spy 3 as **Broadcast** for greater readability.) The *Source* field of this message contains the MAC address of the device sending the request, which in this example is **EC:F4:BB:6E:01:3C**.

The *ARP Reply* message is sent back from the device whose IP address matches the one in the request. In this case (and many others) the responder is the local router, a device made by Western Digital Corporation; its MAC address is decoded by Vehicle Spy 3 as **WesternD_**

CD:D2:26, which is the *Source* of the *ARP Reply*. The *Destination* of this message is, of course, the *Source* of the ARP Request, EC:F4:BB:6E:01:3C.

Now let's examine the individual fields in these messages.

- **Enable Details View, If Necessary:** If *Details View* is not currently enabled, click Details to turn it on.
- **Select the ARP Request Message:** Click the *ARP Request* message in the *Messages View*.

There should be three lines in the *Details View*, the third of which will be *Address Resolution Protocol (request):*. Let's expand this to look at its fields.

- **Expand ARP Request Message:** Click the ⊞ button to the left of Address Resolution Protocol (request): in the *Details View*.

You should now see the individual fields (signals) of the *ARP Request* message in the Details View information pane; the display for the example we have been using is shown in Figure 55.

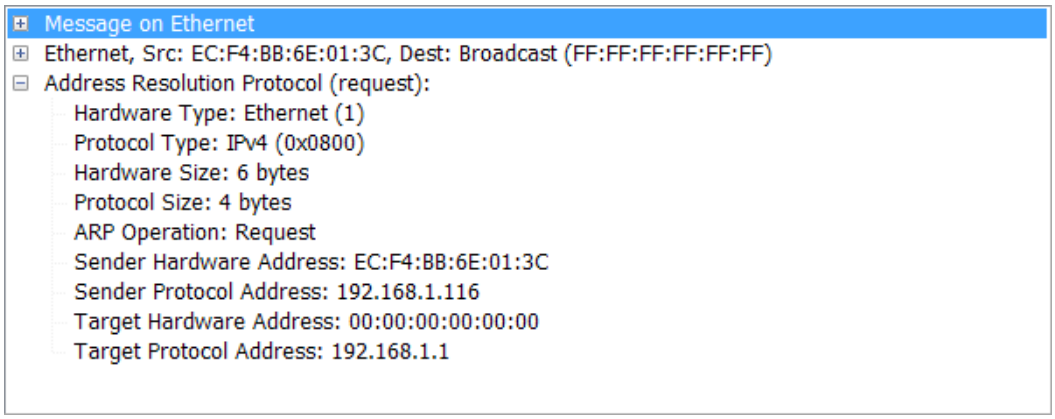


Figure 55: ARP Request Message Details.

Here's a quick description of the fields in the *ARP Request* and what they contain:

- **Hardware Type and Protocol Type:** Codes that indicate the hardware-level and higher-level address types being used. These are usually 1 and 0x800 for Ethernet and IPv4 respectively, as you see here.
- **Hardware Size and Protocol Size:** The lengths of the hardware-level and higher-level addresses; MAC addresses are 6 bytes long and IPv4 addresses are 4 bytes.
- **ARP Operation:** A value of 1 for an *ARP Request* or 2 for an *ARP Reply*, which VSpy decodes for you into their descriptive equivalents.
- **Sender Hardware Address and Sender Protocol Address:** The MAC and IP addresses of the device sending the *ARP Request*.

- **Target Hardware Address and Target Protocol Address:** The MAC and IP address of the device that is the subject of the *ARP Request*; the former is ignored in an *ARP Request*, since it is what we are trying to find out, and therefore is usually set to 0.

Now let's take a look at the matching *ARP Reply* in our example.

► **Click the ARP Reply Message:** Click the *ARP Reply* message in the *Messages View*.

The *Details View* information pane will now show the *ARP Reply* message fields and field values; the one for our example can be found in Figure 56.

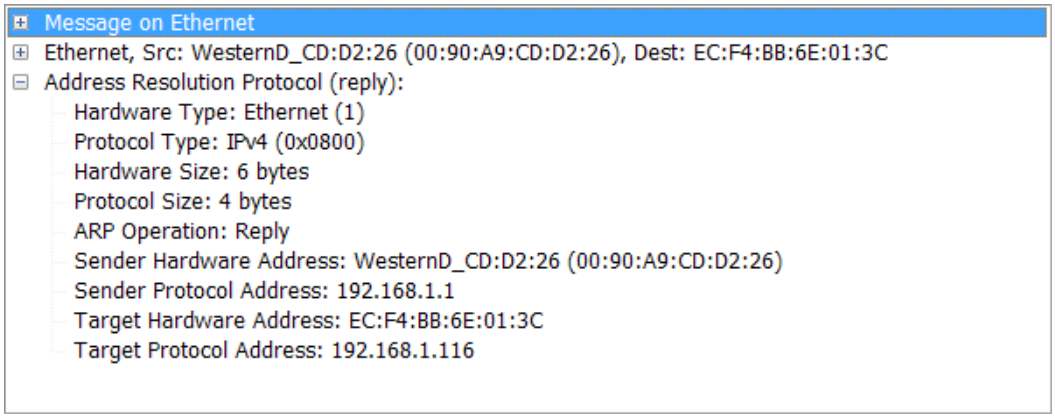


Figure 56: ARP Reply Message Details.

If you click back and forth between the *ARP Request* and *ARP Reply*, you can see clearly how the two messages work together. (Notice that you don't have to expand the message fields again.) The *ARP Reply* has *Target Hardware Address* and *Target Protocol Address* fields equal to the *Sender* equivalents in the *ARP Request*. When the *ARP Reply* is received, the value of its *Sender Hardware Address* field contains the MAC address that the original device requested.

We're done now, so let's go back to static mode and go offline.

► **Enter Static Mode:** Press Scroll.

Lab 2.2 Sending Periodic ARP Requests from EEVB Node A

Let's now turn our attention back to the star of our show, the Ethernet EVB. We will apply our newly-gained expertise in the operation of the Address Resolution Protocol by programming the two nodes of the EEVB to simulate ARP exchanges on an Ethernet+TCP/IP network. Along the way we'll also learn how to use Vehicle Spy 3's docking feature to work with more than one view at a time.

Part 2.2A Load ARP Request Setup File and Examine Messages and Function Block Script

Even though we are working with ARP here rather than raw Ethernet, the basic structure of the setup file for this lab is quite similar to the one we used in Lab 1.3. Not much would be gained by having you create the setup manually, so to save time we will begin with a downloaded scenario file.

► **Load the ARP Request Setup:** Go to the *Logon Screen*, click My Setups, and then double-click 2.2 ARP Request.

Vehicle Spy 3 will load the setup for this lab. Notice that again, along the top, tabs for several windows are present for convenient switching. This is a handy tip to remember: any view tabs open when you save a setup file are preserved when that file is loaded again later.

Let's now take a look at this setup to see what it does.

► **Switch to the Messages Editor:** Click the Messages Editor tab.

The view will show the only message defined in VSpy for this setup, unsurprisingly called *ARP Request*. It is identical on both the transmit and receive sides. If you look at the signals area you will see two tabs, *ARP Header* and *Ethernet Header*, which contain the field values for the ARP message and the Ethernet frame that carries it, respectively (Figure 57). The names here should be familiar from looking at the ARP fields in Lab 2.1.

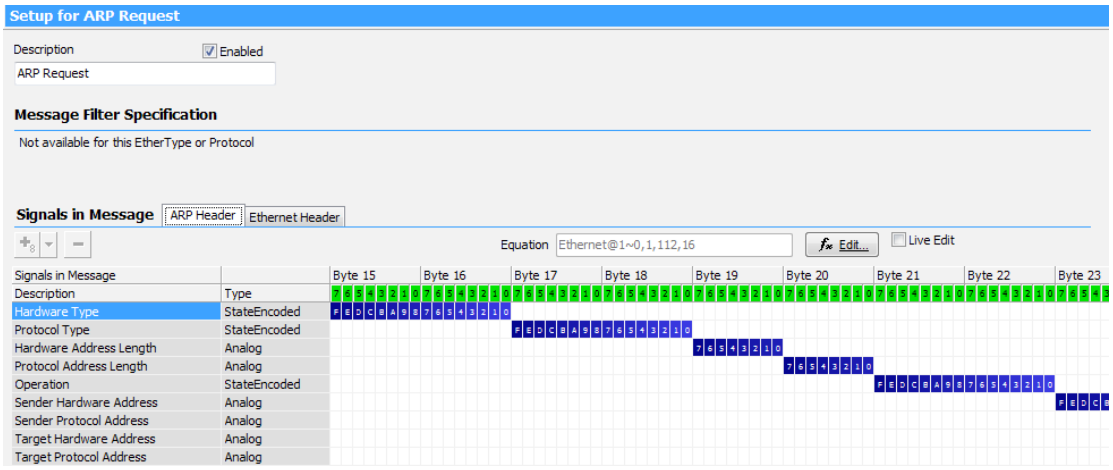


Figure 57: ARP Header Signals for ARP Request Message. Note that in this image we have enlarged the signal area and scrolled the byte display so that the start of the ARP signals in byte 15 is shown.

Now let's check out the function block script.

- **Switch to Function Blocks:** Click the Function Blocks tab.

There is a single function block script called *ARP Request Transmission*. You can see that it is indeed simple, with just two steps that cause it to transmit the *ARP Request* message once every three seconds (Figure 58).

Step	Description	Value	Comment
1	Transmit	ARP Request	// Send ARP Request message using values set in the Tx Panel.
2	Wait For	= 3000 ms	// Wait 3 seconds before sending again.

Figure 58: ARP Request Function Block Script.

Part 2.2B Compare ARP Request Message Values to Defaults Using Window Docking

Let's take a look at the values of the fields defined in the Messages Editor. As before, these are set in the Tx Panel.

- **Switch to Tx Panel:** Click the Tx Panel tab.
- **Select the ARP Request Message:** Click the ARP Request message. You may need to move the central vertical divider to see the *ARP Request* fields.

The values here are originally filled in by VSpy using the templates that we looked at in Section 1, and can then later be changed. It might be instructive to compare these values to the defaults in question; however, defaults are accessed from the *Messages Editor* view. No problem, because VSpy includes the ability to work with multiple views at the same time. We'll start by first docking the Tx Panel to the bottom of the screen.

- **Undock the Tx Panel from the Top of the Screen:** Click and drag the Tx Panel tab until a blue rectangle appears just below the mouse pointer.

Dragging the tab undocks the view from the top of Vehicle Spy 3; the blue rectangle is a preview of the window (Figure 59). If you release the mouse button when the rectangle is in the middle of the screen, the Tx Panel becomes a floating window.

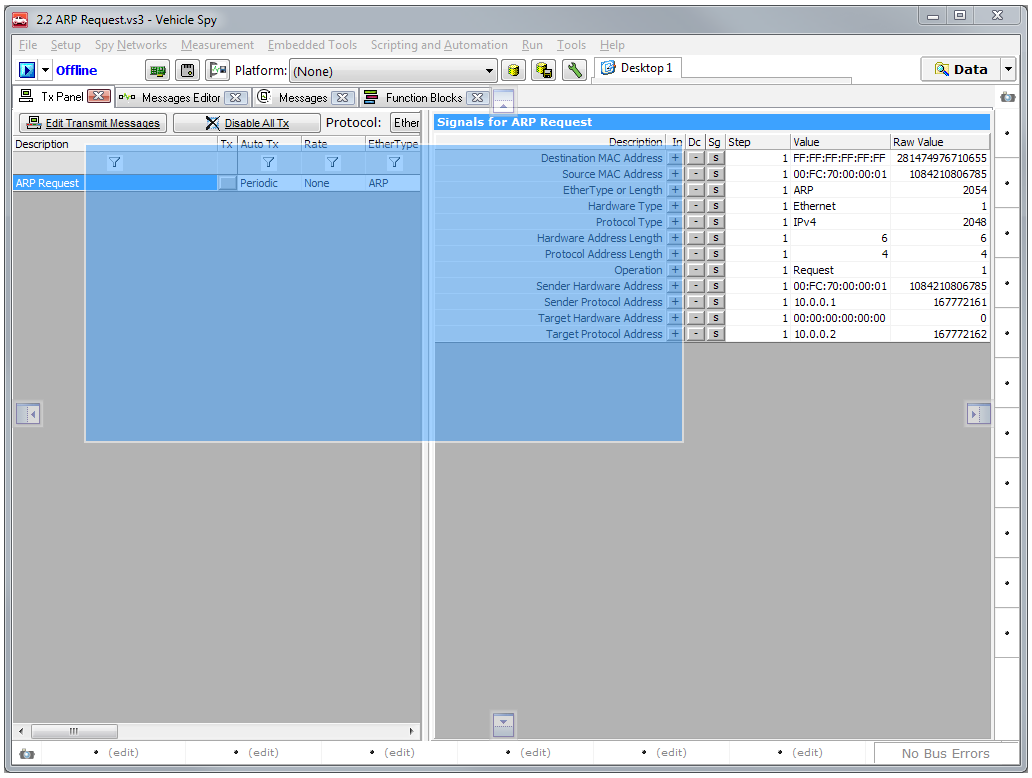


Figure 59: Dragging a Vehicle Spy 3 View Tab. When you click and drag the Tx Panel, at first you will see a blue rectangle representing the view, and releasing it will leave the view in a floating window. Notice the four semi-transparent docking icons at the four cardinal points, representing potential docking locations.

- **Drag the Tx Panel to the Bottom of the Screen:** Continue to drag towards the bottom of the screen until the pointer is over the  symbol there.

When you move the pointer over the “dock at bottom” icon, the blue rectangle fills the bottom half of the screen to show you that the Tx Panel will fill that area if you let go of the mouse there.

- **Dock the Tx Panel at the Bottom of the Screen:** Release the mouse.

The Tx Panel is now docked at the bottom of Vehicle Spy 3, occupying approximately the lower half of the screen. The top of the screen should now be filled by the *Messages Editor* (Figure 60), but you may see a different window, such as *Function Blocks*.

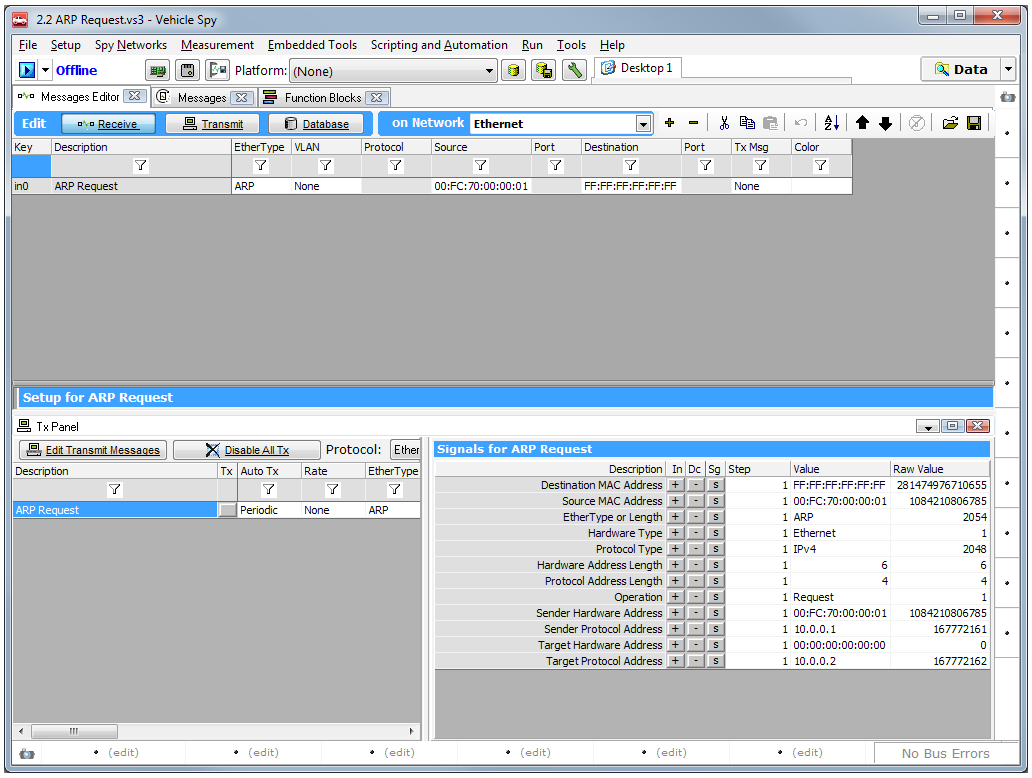


Figure 60: Tx Panel Docked at Bottom. Releasing the mouse button over the "dock at bottom" button causes the *Tx Panel* to be docked in the bottom half of the screen. This allows the view in the next adjacent tab to be shown, which should be the *Messages Editor* as seen here.

Now let's go check out those defaults.

- **Open the Messages Editor:** If necessary, click the Messages Editor tab.
- **Open Ethernet Packet Template Editor:** Click the button. (You may need to widen the Vehicle Spy 3 window to see it).
- **Select ARP Template:** Select ARP Packet Template from the drop-down box.

You should be able to resize and reposition the *Ethernet Packet Template Editor* dialog box so that it is easy to compare the default values to those in the Tx Panel (Figure 61). When you do so, you'll notice that *all* of the values in our message are the same as those in the template. This is a good example of how these default values save time!

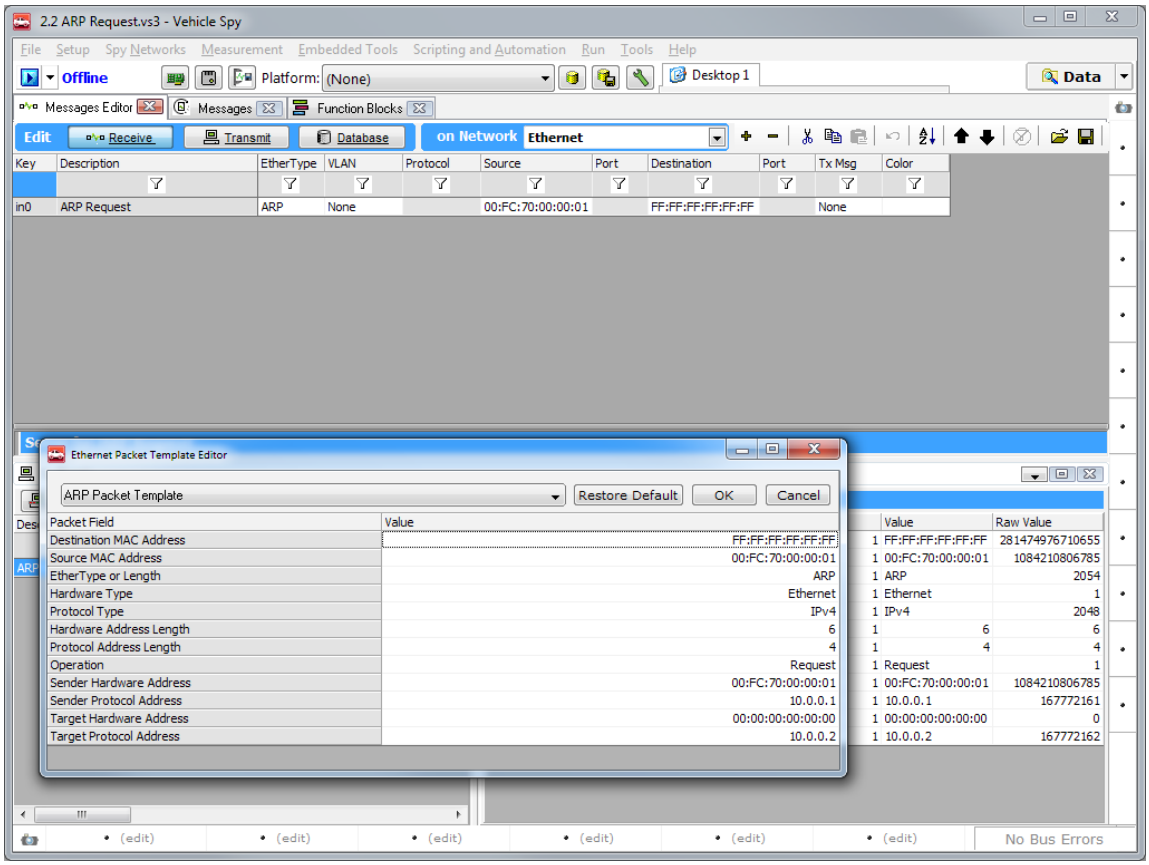


Figure 61: Comparing Ethernet Packet Template Editor Defaults to Current Tx Panel Signal Values. Bringing up the Ethernet Packet Template Editor allows us to easily compare its default values to the current values in the setup file, so we can see what, if anything has been changed (in this case, nothing).

We're done now, so let's reset the VSpy user interface to where it was before we began this slight diversion.

- **Close Ethernet Packet Template Editor:** Click Cancel to close the dialog box without any changes.
- **Dock the Tx Panel at the Top of the Screen:** Click and drag the *Tx Panel* window header until the blue rectangle appears. Keep dragging until the mouse is over the row of window view tabs, the Tx Panel tab reappears, and the blue rectangle fills the VSpy window. Release the mouse button.

And just like that, the *Tx Panel* is restored to its original position.

Part 2.2C Download ARP Request CoreMini to Node A and Clear Old CoreMini from Node B

CoreMinis are stored in flash memory so that they persist even if the EEVB is powered off. However, this also means that whenever you start a new lab with your board, there is the possibility that an old CoreMini may still be running from a previous experiment. For this reason, it is a good practice when starting new work with the Ethernet EVB to clear the

CoreMini from any nodes where you do not plan to download a new setup. This will help avoid unexpected behavior on the network.

- **Download CoreMini to Node A:** Enter the *CoreMini Console* and send the current CoreMini to Node A of your EEVB.
- **Clear CoreMini from Node B:** Select Node B for your EEVB from the drop-down box. Then press the Clear button.

Part 2.2D Go Online and Observe ARP Requests

Now let's go online and see the *ARP Request* messages coming from the EEVB. Since we used our normal Ethernet connection in Lab 2.1, we'll first need to select the board in the Ethernet Interfaces list.

- **Select EEVB Ethernet Interface:** Go to the *Logon Screen* and click the radio button for the EEVB in the Ethernet Interfaces list.
- **Switch to Messages View:** Select *Messages* from the *Spy Networks* menu.
- **Filter for ARP Messages:** Enter ARP in the EtherType filter field.
- **Go Online.**

You will now see the *ARP Request* message, with a new copy arriving approximately every three seconds, right on schedule.

- **Expand ARP Request Message Fields:** Click the ⊞ button next to the *ARP Request* message.

The fields will all contain the same values we saw in the Tx Panel.

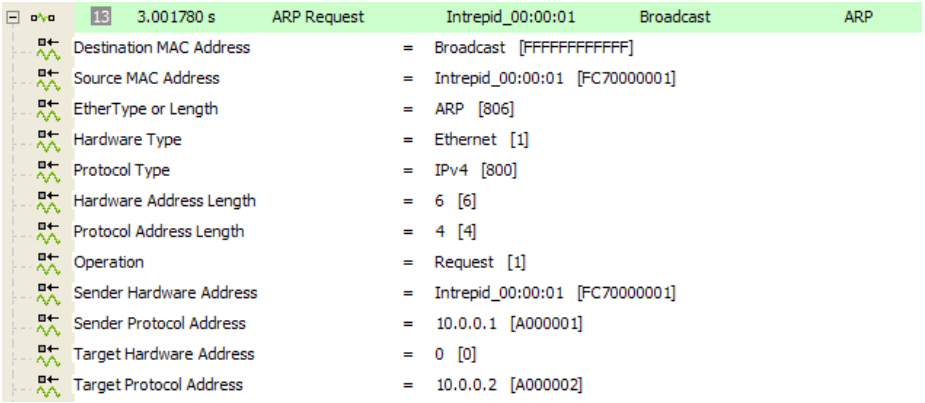


Figure 62: ARP Request Signals in Messages View.

- **Go Offline.**

Lab 2.3 Using Application Signals to Set up an Intelligent ARP Request/ Reply Exchange

In Lab 2.2 we told Node A of the EEVB to transmit an *ARP Request* message every three seconds. Naturally, there was no response to any of these messages, because we never programmed anything to reply to them. We'll now rectify that situation by setting up Node B to respond to *ARP Request* messages from Node A with *ARP Reply* messages. We'll introduce the use of Vehicle Spy 3 *application signals*, one of which we'll use to ensure that replies are only sent when they should be. And we'll also learn a bit more about how Vehicle Spy 3 matches and decodes messages.

Part 2.3A Load and Examine ARP Reply Setup File

We'll again start with a pre-made setup file for efficiency.

► **Load the ARP Reply Setup:** From the *Logon Screen*, select 2.3 ARP Reply from the My Setups tab.

As we usually do, let's start by looking at the messages defined in this setup, starting with the receive side. The setup file should have been saved with the *Messages Editor* as the default view, so you should already be where you need to be. If not, well, you know how to get there on your own at this point!

In the *Messages Editor* (receive side) you will see the *ARP Request* message defined, since this is the message that we are looking for in order to trigger our response (Figure 63). The *Source* field for the message is set to **XX:XX:XX:XX:XX:XX**, which means "any" to Vehicle Spy 3—it tells VSpy to consider an *ARP Request* coming from any device to be a match. The *Destination* field is **FF:FF:FF:FF:FF:FF**, because *ARP Requests* are always broadcast.

Key	Description	EtherType	VLAN	Protocol	Source	Port	Destination	Port	Tx Msg	Color
in0	ARP Request	ARP	None		XX:XX:XX:XX:XX:XX		FF:FF:FF:FF:FF:FF		None	

Figure 63: ARP Request Message Summary for ARP Reply Setup.

► **Switch to the Transmit Side.**

On the transmit side we have our *ARP Reply* message. The *Source* field has been set to 00:FC:70:00:00:02, the MAC address we will typically use for EEVB Node B in the Lab Manual. The *Destination* field is zero because we don't know where the *ARP Reply* will be sent until an *ARP Request* is received. Let's look at the rest of the values in this message.

Key	Description	EtherType	VLAN	Protocol	Source	Port	Destination	Port	Len	Raw Payload Bytes
out0	ARP Reply	ARP	None		00:FC:70:00:00:02		00:00:00:00:00:00			

Figure 64: ARP Reply Message Summary for ARP Reply Setup.

► **Switch to the Tx Panel.**

► **Click the ARP Reply Message.**

The field values here should be familiar. Note that the *Operation* field is here set to **Reply**, the *Sender* and *Source* addresses are the default values for Node B, and the *Target* and *Destination* addresses are zero since they are not known ahead of time.

► **Switch to Function Blocks.**

There is a single function block script here (Figure 65) and you'll notice immediately that it is quite a bit more complex than the one we sent to Node A. The reason is that we only want *ARP Reply* messages sent under the correct conditions. First, we want to generate one only when an *ARP Request* is received—other *ARP Reply* messages should be ignored. Second, we should only respond when the request contains our IP address, meaning we were the intended recipient of the request.

> **Note:** Since we wrote the program that sends the *ARP Request* messages, we already know they will always be sent to us. Furthermore, there are only two Ethernet nodes on the small EEVB network. However, the point here is to simulate correct operation of ARP, and so we have put in the logic that a real device would need when implementing this protocol.

Step	Description	Value	Comment
1	Wait Until	{ARP Request (Present) :in0-0}	// Trigger function block script when a message is received matching our ARP Request definition.
2	Set Value	{ARP Request (Present) :in0-0} = 0	// Clear Present flag to avoid duplication.
3	If	{Operation (Value) :in0-sig7-0} = 1	// Check the Operation field in the ARP message; only proceed if this is an ARP Request.
4	If	{Target Protocol Address (Value) :in0-sig11-0} = {My IP Address :sig0-index(0)}	// Compare the Target Protocol Address in the ARP Request to our own IP address so we only generate an ARP Reply if the original message was intended for us.
5	Set Value	{Target Hardware Address (Value) :out0-sig10-0} = {Source MAC Address (Value) :in0-sig1-0}	// Set the Target Hardware Address in the ARP Reply to be equal to the Source MAC Address in the ARP Request (which is our MAC address since we received that request).
6	Set Value	{Target Protocol Address (Value) :out0-sig11-0} = {Sender Protocol Address (Value) :in0-sig9-0}	// Set the Target Protocol Address in the ARP Reply to the Sender Protocol Address of the ARP Request.
7	Set Value	{Destination MAC Address (Value) :out0-sig0-0} = {Source MAC Address (Value) :in0-sig1-0}	// Set the Destination MAC Address in the Ethernet header to be the MAC address of the device that sent the ARP Request.
8	Transmit	ARP Reply	// Send the completed ARP Reply.
9	End If		
10	End If		

Figure 65: ARP Reply Function Block Steps.

The script begins by waiting for an *ARP Request* message to be received and recognized by Vehicle Spy 3, and ensuring that such a message is only seen once (*Steps 1* and *2*). It then checks the *Operation* field within the ARP header to ensure that it is a request and not a reply (*Step 3*). This is necessary because VSpy will match the *ARP Request* based only on the Ethernet field values: *Source MAC Address*, *Destination MAC Address* and *EtherType*. Since we want to trigger based on an ARP header field, not an Ethernet header field, we must program this explicitly.

If an *ARP Request* is received, we then check the *Target Protocol Address* field, which contains the IP address that the *ARP Request* is attempting to resolve (*Step 4*). If it matches our IP address, then this is a request for which we need to send a reply. We fill in the *Target Hardware Address* and *Target Protocol Address* using the *Source Hardware Address* and *Source Protocol Address* values from the *ARP Request* (*Steps 5* and *6*). We also set the *Destination MAC Address* to the *Source MAC Address* of the message we received so we respond back to the originator of the request (*Step 7*). Finally, we send the *ARP Reply* (*Step 8*).

But wait: how do we even know what "our IP address" is? Essentially, we decide ahead of time what value we want to use for our IP address and ensure that it matches the value that the *ARP Request* message is trying to resolve. Then we put it in a special variable that we can use in our function block script. These variables are called *application signals*, and are defined using the feature of the same name within Vehicle Spy 3.

► **Switch to Application Signals:** Click the Application Signals tab, or select Application Signals from the *Scripting and Automation* menu.

In this (still pretty simple) example we have only one application signal defined, which is the IP address we used for the check in *Step 4* of the function block script (Figure 66).

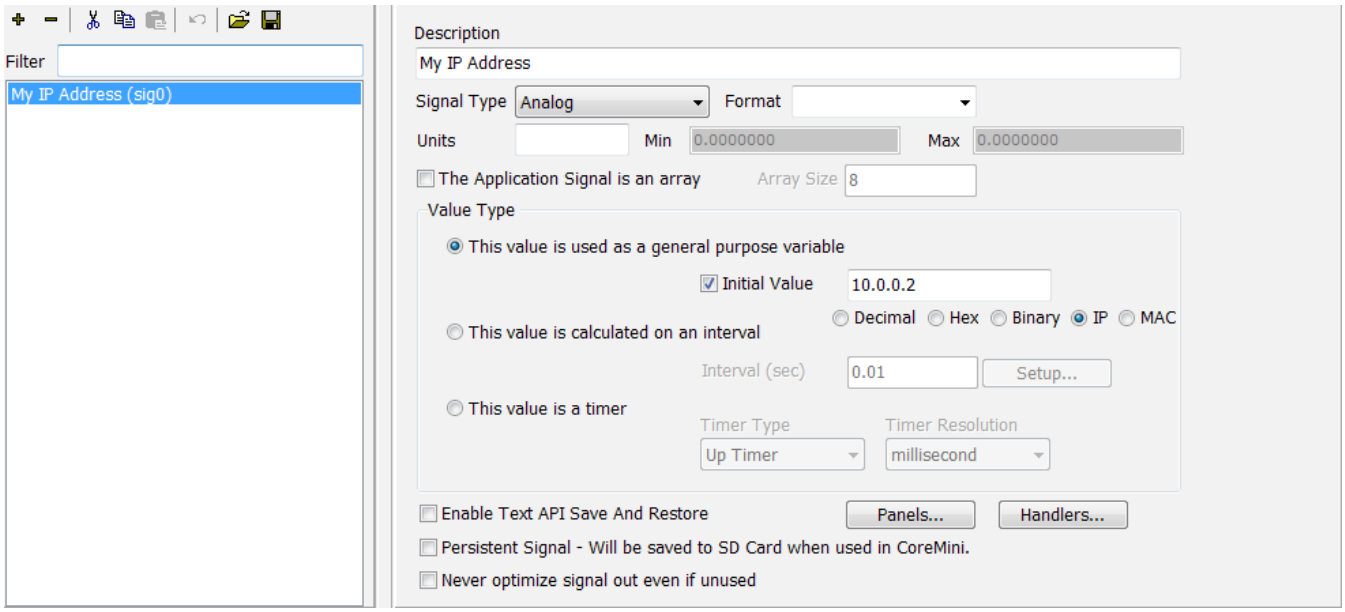


Figure 66: Application Signal Defining IP Address of ARP Reply Node. Vehicle Spy 3's application signals are variables that can be used in function block scripts. In this setup we use an application signal to define the IP address of the node so we can compare it to the target address in incoming *ARP Request* messages.

Part 2.3B Download ARP Reply CoreMini to Node B and Go Online to See ARP Request/Reply Exchanges

We're now ready to set up Node B of the EEVB to send *ARP Reply* messages when it receives *ARP Request* messages from Node A.

► **Download CoreMini to Node B:** Select Node B in the *CoreMini Console* and send to it.

- **Switch to Messages View:** Click the Messages tab.
- **Filter for ARP Messages:** Enter ARP in the EtherType filter field.
- **Go Online.**
- **Turn on Scroll Mode:** Press Scroll to enable scroll mode if it is currently off.

You should now see matched pairs of messages, with an *ARP Request* being received every three seconds followed a few milliseconds later by... wait, shouldn't those be labeled as *ARP Reply* messages?

Part 2.3C Figure Out Why the ARP Reply Messages are Not Decoding Properly

Our program script is working correctly, and Node B is indeed responding to Node A's *ARP Request* messages with *ARP Reply* messages. But Vehicle Spy 3 isn't showing them correctly in *Messages View*. Why? Well, remember back in Lab 1.3 that we defined a transmit message but also copied it to the receive side? We never did that here. We defined the *ARP Request* message on the receive side because it was the message we are receiving, and the *ARP Reply* on the transmit side since it's what we are transmitting. But we never put a copy of the *ARP Reply* on the receive side. So let's do that now.

- **Go Offline:** Always a good idea when making changes.
- **Switch to the Messages Editor, Transmit Side.**
- **Copy ARP Reply Message to Receive Side:** Click ARP Reply, then right-click it, select Copy To and then Receive.
- **Switch to Messages View.**
- **Go Online.**

And... it still doesn't work. Can you figure out why? Here's a hint: take a look at the values in the *Source* and *Destination* columns on the receive side for this message, and think about what those values mean.

Still stumped? Our original definition of the *ARP Reply* message (on the transmit side) had zero for the *Destination MAC Address* field. We did this because until an *ARP Request* arrives, we don't know where the *ARP Reply* will be transmitted; the field is filled in with the correct value by the function block script. But on the receive side these values are used for message *matching*. Vehicle Spy 3 therefore will only recognize an *ARP Reply* message if the *Destination MAC Address* field is all zeroes. Of course this never will happen (because we replace that zero value with a real address) so the messages are never decoded.

The fix is simple: change the field value from zero to "any".

- **Go Offline.**
- **Switch to the Messages Editor.**

- **Change Receive ARP Reply Destination MAC Address to "Any":** Change the receive-side *ARP Reply* message's *Destination MAC Address* from 00:00:00:00:00:00 to XX:XX:XX:XX:XX:XX. (You can just type "X" a bunch of times to make this faster; VSpy will fill in the colons.)
- **Go Online.**
- **Switch to Messages View.**

There we go. Now Vehicle Spy 3 will decode all *ARP Reply* messages, and as you should be able to see, the request/reply exchange between the nodes is working correctly (Figure 67).

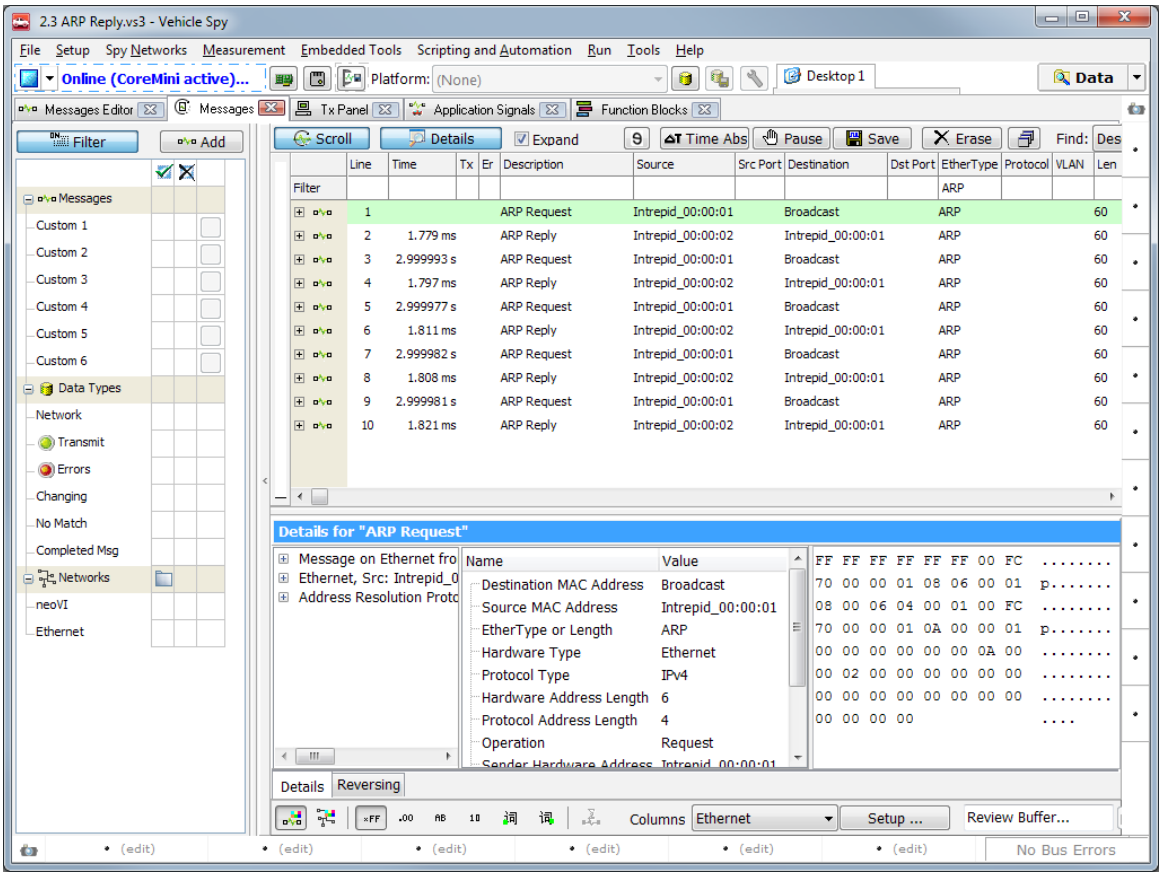


Figure 67: EEVB ARP Request / ARP Reply Exchange. With the message definitions fixed in the *Messages Editor*, Vehicle Spy 3 now correctly shows the *ARP Request* messages coming from Node A every three seconds and the *ARP Reply* messages sent in response to each by Node B.

Let's save our changes so that if we want to look at the exchange again, our messages will decode properly.

- **Go Offline.**
- **Save the Setup File:** Save the setup file as 2.3 ARP Reply Decoded.

Lab 2.4 Controlling ARP Request and Reply Operation Using EEVB Inputs

In the final lab of Section 1, Lab 1.6, we saw for the first time in this Lab Manual how to use a pushbutton input on the EEVB to influence the behavior of a CoreMini running on one of its nodes. We'll now build upon that experience by changing our basic ARP exchange simulation so that both *ARP Request* and *ARP Reply* messages are controlled using EEVB inputs. More specifically, Node A will be modified so the rate at which it sends requests depends on the setting of the node's potentiometer input, while Node B will only send a single response when its pushbutton is held down.

Part 2.4A Load and Modify ARP Request Setup File

Naturally, we'll begin with the *ARP Request* side of the equation. First, we load the original setup that we used in Lab 2.2.

- **Load 2.2 ARP Request:** It should be on the Recent tab since we just used it; if not, find it in My Setups.

Making the *ARP Request* sending rate vary with the EEVB potentiometer turns out to be surprisingly simple. The potentiometer is seen as an *analog input* by Vehicle Spy 3, producing an integer value ranging from 0 to 4,095. The *Wait For* step in our function block script has a wait time measured in milliseconds, so we simply replace the static 3000 value that is there now with the value from this input.

- **Switch to Function Blocks.**
- **Change Wait For Value:** Double-click = 3000 ms in *Step 2*, then press the = key. When the *Expression Editor* appears, delete the 3000 from the Expression box, leaving the cursor there. Click Physical IO on the left-hand menu, click the + button next to Analog Inputs, then double-click Analog Input 1. The value {Analog Input 1 (Value) :neo0-ai0-0-index(0)} appears in the Expression box. Press *Enter* or click OK.
- **Change Wait For Comment:** Change the comment to Wait a number of milliseconds determined by the potentiometer position before sending again..

That's it! Pretty simple, huh?

Let's save this so it's there if we need it later.

- **Save the Setup File:** Save the current setup as 2.4 ARP Request Potentiometer.

Part 2.4B Download and Test ARP Request Setup File Changes

Okay, let's put this new setup file to work. For starters we'll just do Node A, clearing from Node B the CoreMini left over from any previous steps.

- **Send CoreMini to Node A.**
- **Clear CoreMini from Node B.**

Now, let's set the potentiometer to maximum as a starting point for our test.

- **Set Node A Potentiometer to Maximum:** The potentiometer for Node A is the round black dial just left of the very center of the board. Gently turn it clockwise until it stops.

We'll now reload the setup from Lab 2.3, after our decoding modifications. This file has the definition of the *ARP Reply* message, without which that message won't be properly interpreted by Vehicle Spy 3. Besides, we're going to need it for the next part anyway.

- **Load the ARP Reply Setup File:** Select 2.3 ARP Reply Decoded from the Recent tab on the *Logon Screen*. Discard changes if prompted.
- **Filter for ARP Messages in Messages View:** Enter ARP in the EtherType filter field.
- **Go Online.**
- **Turn on Scroll Mode (If Necessary).**

You should now see *ARP Request* messages just as you did in Lab 2.2. However, the time between messages should now be a little over four seconds, instead of three seconds as it was before. This corresponds to the approximate maximum potentiometer value of 4,095.

- **Change Potentiometer Setting to Minimum:** Slowly turn the potentiometer counter-clockwise until it stops.

As you change the potentiometer setting, observe the impact on the *ARP Request* send rate (Figure 68). With the dial all the way to the left, messages will be coming at a pretty fast rate. Note, however, that there is a minimum of roughly 5 to 10 milliseconds between messages, which is largely due to the time required to generate them.

- **Go Offline.**

	Line	Time	Tx	Er	Description	Source	Src Port	Destination	Dst Port	EtherType
Filter										ARP
	1				ARP Request	Intrepid_00:00:01		Broadcast		ARP
	2	4.096698 s			ARP Request	Intrepid_00:00:01		Broadcast		ARP
	3	4.096706 s			ARP Request	Intrepid_00:00:01		Broadcast		ARP
	4	4.095692 s			ARP Request	Intrepid_00:00:01		Broadcast		ARP
	5	4.096722 s			ARP Request	Intrepid_00:00:01		Broadcast		ARP
	6	4.092715 s			ARP Request	Intrepid_00:00:01		Broadcast		ARP
	7	3.420752 s			ARP Request	Intrepid_00:00:01		Broadcast		ARP
	8	2.483821 s			ARP Request	Intrepid_00:00:01		Broadcast		ARP
	9	1.887863 s			ARP Request	Intrepid_00:00:01		Broadcast		ARP
	10	1.299912 s			ARP Request	Intrepid_00:00:01		Broadcast		ARP
	11	955.915 ms			ARP Request	Intrepid_00:00:01		Broadcast		ARP
	12	648.969 ms			ARP Request	Intrepid_00:00:01		Broadcast		ARP
	13	382.978 ms			ARP Request	Intrepid_00:00:01		Broadcast		ARP
	14	376.184 ms			ARP Request	Intrepid_00:00:01		Broadcast		ARP
	15	285.755 ms			ARP Request	Intrepid_00:00:01		Broadcast		ARP
	16	183.989 ms			ARP Request	Intrepid_00:00:01		Broadcast		ARP
	17	124.993 ms			ARP Request	Intrepid_00:00:01		Broadcast		ARP
	18	104.969 ms			ARP Request	Intrepid_00:00:01		Broadcast		ARP
	19	100.019 ms			ARP Request	Intrepid_00:00:01		Broadcast		ARP

Figure 68: Variable Delay Between ARP Request Transmissions. With the potentiometer at its farthest clockwise position, the *ARP Request* messages are sent by Node A about every 4.095 seconds, corresponding to its sampled value of 4,095. As the potentiometer is turned counter-clockwise, the value decreases, as does the time between transmissions.

Part 2.4C Modify ARP Reply Setup File to Trigger on Pushbutton Input

Now we'll work on the other half of the equation, the *ARP Reply* setup, which we already loaded in the previous step. Making these messages be sent only when a pushbutton is held down is similar to what we did in Lab 1.6. However, this time instead of having the script run continuously as long as the button is depressed, we will design it so only one reply is sent each time the button is pushed. This requires a bit more smarts in our program, and so this part will involve more extensive "renovations" than we performed on the *ARP Request* setup file.

We're going to do this in two steps. First we'll add a condition to the script so it runs only when the pushbutton is currently down. Then we'll add some logic to avoid having the script be triggered more than once for any pushbutton press.

- **Switch to Function Blocks.**
- **Enlarge Script Window Pane:** You'll want to move the horizontal divider up so you have more room to edit the script steps.
- **Add If Statement:** Click *Step 5* of the function block, then press Before. Select If under the Description field.
- **Enter If Statement Condition:** Double-click on the Value field, and in the *Expression Editor* select Physical IO from the menu on the left side, scroll down to find the Switches section and expand it using the + button, then double-click Switch 1. Press OK.
- **Enter If Statement Comment:** Add this comment to the command: Only send ARP Reply if pushbutton is currently down..
- **Add End If Statement:** Click *Step 10*, then click Before. Select End If in the Description field.

The *ARP Reply* message will now be sent only when the pushbutton is held down. However, it will send the messages continuously, rather than only one per button press. To accomplish what we want, we must keep track of whether or not we've already sent an *ARP Reply* for each pressing of the button. We need a variable to hold this transmission status, so we will create an application signal for this purpose. We will use a *digital* signal, which is the term Vehicle Spy 3 uses for a variable that holds a single binary value.

- **Go to Application Signals:** Click the Application Signals tab.
- **Add Application Signal:** Press + to add a new application signal, which by default is called App Signal 2. On the right-hand pane, change that name to ARP Reply Sent. The name will change in the left pane as you type it in.
- **Change Signal Type:** Click the drop-down box next to Signal Type and select Digital.
- **Set Initial Value:** Tick the checkbox next to Initial Value and confirm that 0 is in the adjacent box, or enter it if need be.

The application signal *ARP Reply Sent* is now ready for use (Figure 69). We will only send the *ARP Reply* when the pushbutton is held down *and ARP Reply Sent* is 0. When we do transmit it, we will set *ARP Reply Sent* to 1 so the next time, it won't be sent again.

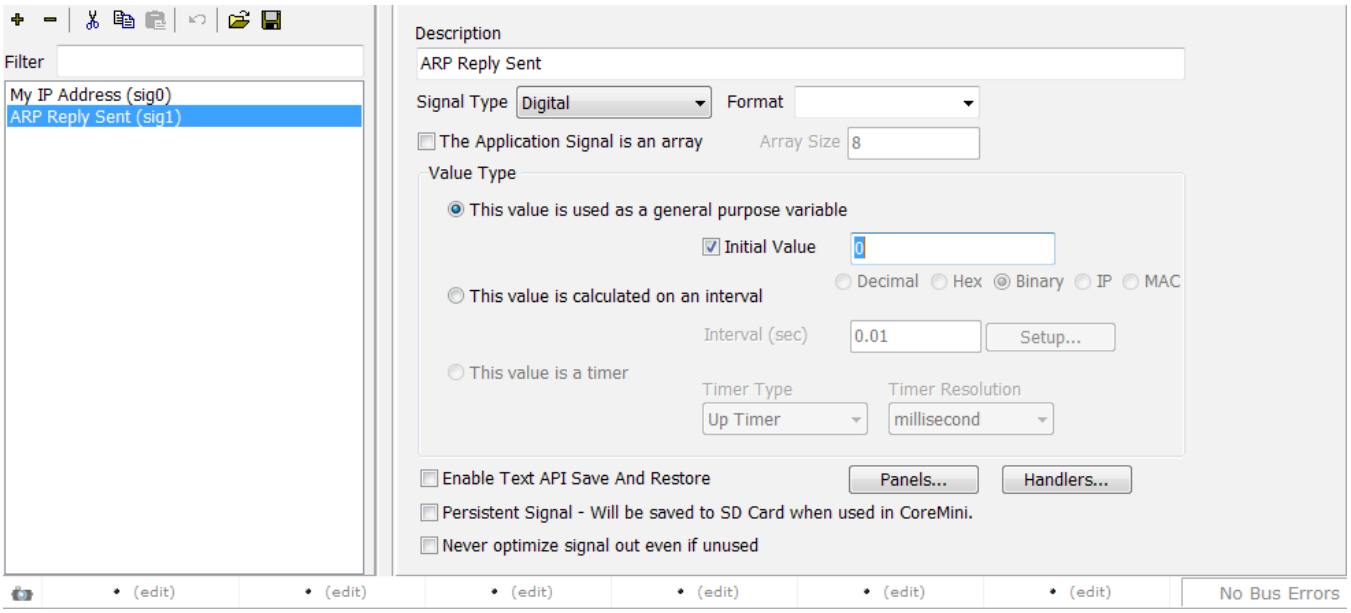


Figure 69: ARP Reply Sent Application Signal.

Let's return to our script and make the remaining changes.

- **Switch to Function Blocks.**
- **Add If Statement:** Click Step 6, then press Before and choose an If statement.
- **Enter If Statement Condition:** Double-click the Value field for this statement, then in the *Expression Editor* select App Signals and double-click ARP Reply Sent. Vehicle Spy 3 will put {ARP Reply Sent :sig1-index(0)} in the Expression field; edit the field, adding =0 to the end, and then click OK.
- **Enter If Statement Comment:** Add this comment for the step: Only trigger sending an ARP Reply message if we have not already sent one for this button press..
- **Add End If Statement:** Insert the End If statement to match the new If statement just above the current Step 11. (You should be able to do this without having each click spelled out at this point—give it a shot.)

Finally, we need to add statements to set our variable to the correct values at the right times. We want to set it to 1 whenever we transmit an *ARP Reply*, to ensure that we only trigger once. We also must set it back to 0 whenever we find that the pushbutton is currently not held down, so the next button press is seen. For the latter part of this we'll need an *Else* statement for the *If* statement that tests the pushbutton's status.

First, the command to set the variable to 1 on a transmit.

- **Add Set Value Statement:** Add a new function block step before *Step 7*. Select a Set Value command.
- **Enter Set Value Expression:** Double-click Value, then in the *Expression Editor* select App Signals (if necessary) and double-click ARP Reply Sent, just as you did earlier. This time Vehicle Spy 3 will put {ARP Reply Sent :sig1-index(0)} in the field called Value To Set. Click in the Expression box, enter the value 1, and click OK.
- **Enter Set Value Comment:** Enter this comment: Since pushbutton is down, set ARP Reply Sent to 1 so we don't trigger again until the button is released..

And now we'll add the two statements we need to reset the variable to 0 when the button is released.

- **Add Else Statement:** Add an Else command before the current Step 13. After adding this line, you should see one End If above it and three below.
- **Duplicate Set Value Statement:** Highlight the Set Value command you created just above, which should now be *Step 7*. Click the button to copy it, then click *Step 14* and press to insert a copy of the step below the Else command.
- **Modify Set Value Statement:** Double-click the Value field and in the *Expression Editor* change the 1 in the Expression box to 0. Change the comment to: Pushbutton is currently up, so clear ARP Reply Sent to 0 to prepare for next button push..

And that should do it! The final script should look something like Figure 70. I'm sure you'd rather not do that again, so let's save our changes in a new file.

- **Save Setup File:** Save the setup as 2.4 ARP Reply Pushbutton.

Step	Description	Value	Comment
1	Wait Until	{ARP Request (Present) :in0-0}	// Trigger function block script when a message is received matching our ARP Request definition.
2	Set Value	{ARP Request (Present) :in0-0} = 0	// Clear Present flag to avoid duplication.
3	If	{Operation (Value) :in0-sig7-0} = 1	// Check the Operation field in the ARP message; only proceed if this is an ARP Request.
4	If	{Target Protocol Address (Value) :in0-sig11-0} = {My IP Address :sig0-index(0)}	// Compare the Target Protocol Address in the ARP Request to our own IP address so we only generate an ARP Reply if the original message was intended for us.
5	If	{Switch 1 (Value) :neo0-sw0-0-index(0)}	// Only send ARP Reply if pushbutton is currently down.
6	If	{ARP Reply Sent :sig1-index(0)}=0	// Only trigger sending an ARP Reply message if we have not already sent one for this button press.
7	Set Value	{ARP Reply Sent :sig1-index(0)} = 1	// Since pushbutton is down, set ARP Reply Sent to 1 so we don't trigger again until the button is
8	Set Value	{Target Hardware Address (Value) :out0-sig10-0} = {Source MAC Address (Value) :in0-sig1-0}	// Set the Target Hardware Address in the ARP Reply to be equal to the Source MAC Address in the ARP Request (which is our MAC address since we received that request).
9	Set Value	{Target Protocol Address (Value) :out0-sig11-0} = {Sender Protocol Address (Value) :in0-sig9-0}	// Set the Target Protocol Address in the ARP Reply to the Sender Protocol Address of the ARP Request.
10	Set Value	{Destination MAC Address (Value) :out0-sig0-0} = {Source MAC Address (Value) :in0-sig1-0}	// Set the Destination MAC Address in the Ethernet header to be the MAC address of the device that sent the ARP Request.
11	Transmit	ARP Reply	// Send the completed ARP Reply.
12	End If		
13	Else		
14	Set Value	{ARP Reply Sent :sig1-index(0)} = 0	// Pushbutton is currently up, so clear ARP Reply Sent to 0 to prepare for next button push.
15	End If		
16	End If		
17	End If		

Figure 70: Function Block Steps for Pushbutton Control of ARP Reply Server.

Part 2.4D Download and Test ARP Reply Setup File Changes

Let's send this new script to the EEVB, go online, and see how well we did.

- ► **Download CoreMini to Node B.**
- ► **Switch to Messages View.**
- ► **Filter for ARP Messages:** Enter ARP as an EtherType filter if it is not already there.
- ► **Go Online.**

You should now see a stream of *ARP Request* messages, but no *ARP Reply* messages yet.

- ► **Press and Hold the Pushbutton on Node B.**

A single *ARP Reply* message should appear.

- ► **Release and then Again Press and Hold Node B Pushbutton.**

Vehicle Spy 3 will now display a second *ARP Reply* message.

Now let's try something.

- ► **Set Node A Potentiometer to Maximum:** Turn the Node A potentiometer clockwise until it stops.
- ► **Press and then Immediately Release the Node B Pushbutton.**

You likely won't see an *ARP Reply* message this time. Why? Our function block script triggers only when an *ARP Request* is seen by Vehicle Spy 3. With the potentiometer at maximum, one is only sent about every four seconds, so unless you are lucky with the button, the press will never register. We could fix this, of course, by making the script even smarter. But you get the idea, so we'll stop there.

- ► **Go Offline.**

Lab 2.5 Setting Up an ARP Request/Reply Exchange Between the EEVB and PC

All of the message examples in the Lab Manual so far have used both of the nodes on the Ethernet EVB. This makes sense, since of course a primary goal of this document is to illustrate how Ethernet works using the EEVB board. It's also possible, however, to use the PC as one of the participants in a request/reply exchange. This is in fact quite easy to do: we just have one of the scripts run on Vehicle Spy 3 within the PC rather than downloading it to a node.

In this brief lab we'll modify the previous experiment by having the PC respond to *ARP Request* messages sent by EEVB Node A instead of having this done by Node B.

Part 2.5A Reset EEVB Nodes

In the preceding lab we made some customizations to the CoreMinis running on both EEVB nodes. We will now reset the nodes so they are back to running the "standard" *ARP Request* / *ARP Reply* exchange from back in Lab 2.3.

- **Load the Original ARP Request Setup:** Load the `2.2 ARP Request` setup from the Recent tab on the *Logon Screen*.
- **Send CoreMini to Node A.**
- **Load the Decoded ARP Reply Setup:** Load `2.3 ARP Reply Decoded`.
- **Send CoreMini to Node B.**
- **Switch to Messages View.**
- **Filter for ARP Messages:** Enter ARP as an EtherType filter.
- **Enter Scroll Mode.**
- **Go Online.**

Verify that the operation of the nodes has reverted to what it was in Lab 2.3. Leave Vehicle Spy 3 online as you proceed to the next part of the lab.

Part 2.5B Take Node B Out of the Picture

Assuming you have been following the Lab Manual sequentially—as we recommend—you now have Node B looking for *ARP Request* messages and sending *ARP Reply* messages in response. We don't want to have both the EEVB and the PC responding to the same messages, so we need to stop Node B from reacting to them. One easy way to do this would simply be to clear the CoreMini from the node, but then if we want to have the node resume that role we'd have to download it again. There's an easier way: isolate Node B physically.

- **Disconnect the BroadR-Reach Cable:** Carefully detach the cable linking the two EEVB nodes by pressing down the release lock on either connector and then pulling it out of the jack.

You should now see that the *ARP Request* messages continue to arrive, while new *ARP Reply* messages are no longer received.

As explained in the EEVB User's Guide, only Node A is connected internally to the USB interface that runs to the PC; Node B is also not connected to Node A using the board's circuitry. The BroadR-Reach cable is the means by which Node B is able to communicate with Node A and the PC, so detaching it effectively silences that node while still allowing Node A to function normally.

Part 2.5C Change ARP Server Setup to Run on PC

Now all we need to do is set up the same functionality we put in Node B on Vehicle Spy 3 running on the PC. To accomplish this, we just change the setup so the function block script runs on the PC rather than the EEVB.

- **Go Offline.**
- **Switch to Function Blocks.**
- **Change the Start Type:** Click the Start tab. Then change the current value of Start Immediately Embedded Only in the drop-down box to Start Immediately.
- **Switch to Messages View.**
- **Go Online.**

Now you will see *ARP Reply* messages generated once again. Notice the symbol in the *Tx* column; this indicates that Vehicle Spy 3 recognizes that it sent the *ARP Reply* messages, as opposed to having seen them as they were transmitted by another device on the network (such as the EEVB).

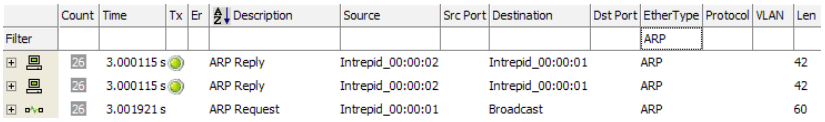
What if we wanted to do this the other way: have the PC generate ARP Request messages and then have the EEVB respond? Unfortunately this is not possible using just Vehicle Spy 3 and the EEVB. We can certainly set up the PC and an EEVB node in this manner just by swapping which setups we run on each. However, the EEVB is designed only to send Ethernet traffic to the PC over its USB link, not to receive Ethernet traffic from it over that connection. However, in Lab 2.6 we'll see that there is a way around this limitation if we add another hardware device to the mix.

Part 2.5D Adjust Message Definitions for PC-Based Operation

There's one small issue with the adaptation we have made here, but if you followed the directions exactly, you may not see it. That's because it is only visible when Vehicle Spy 3 is in static mode. Let's disable scrolling so we can see what changes.

► **Enter Static Mode.**

You will now see the standard static mode *Messages View* display, but with one curiosity: two *ARP Reply* messages for every *ARP Request* (Figure 71). We aren't actually sending twice as many messages, of course; the script didn't change just because we turned off scrolling. So what's going on?



	Count	Time	Tx	Er	Description	Source	Src Port	Destination	Dst Port	EtherType	Protocol	VLAN	Len
Filter										ARP			
⊞	26	3.000115 s			ARP Reply	Intrepid_00:00:02		Intrepid_00:00:01		ARP			42
⊞	26	3.000115 s			ARP Reply	Intrepid_00:00:02		Intrepid_00:00:01		ARP			42
⊞	26	3.001921 s			ARP Request	Intrepid_00:00:01		Broadcast		ARP			60

Figure 71: Dual ARP Reply Messages in Static Mode.

Remember how until now we have always copied all transmit messages over to the receive side so that VSpy would recognize and decode them? This was required because all of the messages were being sent by the EEVB, which is a device external to the PC. However, the program already knows about messages that it itself is sending. In scroll mode, Vehicle Spy 3 only displays one line per message it sees on the network, even if multiple messages match in the *Messages Editor*. This is why we only saw an *ARP Reply* once per transmission. In static mode, however, each message definition that matches receives a separate entry; this is intentional, to support cases where one message may match multiple definitions depending on the circumstances. Here we have an identical transmit and receive copy of the same definition, so we always see it twice.

The solution is pretty simple, of course: remove the extra message definition.

► **Go Offline.**

► **Switch to Messages Editor, Receive Side.**

► **Delete ARP Reply Message:** Click ARP Reply and then press − to delete the message.

► **Switch to Messages View.**

► **Go Online.**

You will now see only one *ARP Reply* line and one *ARP Message* line in both scroll and static modes.

Part 2.5E Return Back to EEVB Operation

We'll now revert our changes and resume operation using only the EEVB. We don't actually have to change anything on the EEVB itself, since we didn't change it to do this lab in the first place—all we need to do is reconnect the two nodes and load a fresh copy of the original *ARP Reply* setup file to erase the changes we made above.

► **Go Offline.**

- **Load the Decoded ARP Reply Setup:** Load *2.3 ARP Reply Decoded* (and discard changes when prompted).
- **Reconnect the BroadR-Reach Cable:** Attach the cable connector that you previously disconnected.
- **Switch to Messages View (If Necessary).**
- **Filter for ARP Messages:** Enter ARP in the EtherType filter field.
- **Go Online.**

You should now see *ARP Request* and *ARP Reply* messages in Vehicle Spy 3, as before.

- **Go Offline.**

Lab 2.6 Manual ARP Request from PC to EEVB Using RAD-Moon (Optional)

In Lab 2.5 we used the PC as one of the participants in an ARP exchange, having it generate *ARP Reply* messages in response to *ARP Request* messages received from an EEVB node. As mentioned near the end of that lab, though, the EEVB is designed for only one-way communication—it can send messages to the PC, but not receive them from it. Thus, with only the EEVB, one could not have the opposite setup, with the PC generating the *ARP Request* and the EEVB sending back the *ARP Reply*.

Enter the RAD-Moon, another member of Intrepid's full line of Automotive Ethernet tools. The RAD-Moon is a *media converter*, a simple device that translates between two physical layer implementations of the same network type. In this case the device converts between standard Ethernet (used by computers) and BroadR-Reach (used in automotive applications).

The RAD-Moon receives frames from both its Ethernet and BroadR-Reach connections, converts them, and then retransmits them on the other link, passing traffic in both directions simultaneously. This makes the RAD-Moon an essential tool for working with all sorts of AE applications. In this optional lab we'll see one such use, enabling a PC to communicate directly with an Ethernet EVB node. We'll set up an ARP exchange where the PC sends *ARP Request* messages and the EEVB sends *ARP Reply* messages back. This is comparable to how one might use Vehicle Spy 3 and the RAD-Moon to test an Automotive Ethernet ECU.

You will need a PC with a standard wired Ethernet connection for this lab to work properly. While not mandatory, access to an Ethernet hub or switch will make the hardware setup easier for this lab.

> **Note:** The RAD-Moon can be obtained bundled with the EEVB or purchased separately; please contact the sales department at Intrepid Control Systems for more details. If you don't have a RAD-Moon, feel free to skip this lab and continue with Section 3.

Part 2.6A Set Up the Ethernet EVB and RAD-Moon for Bidirectional Communication

Since we are using extra hardware, we'll need to start by creating a new physical configuration. This is pretty simple to do, and as usual, we'll guide you through the process step by step. As you follow along, you may find it useful to refer to the special hookup diagram for this lab, which can be found in Figure 72.

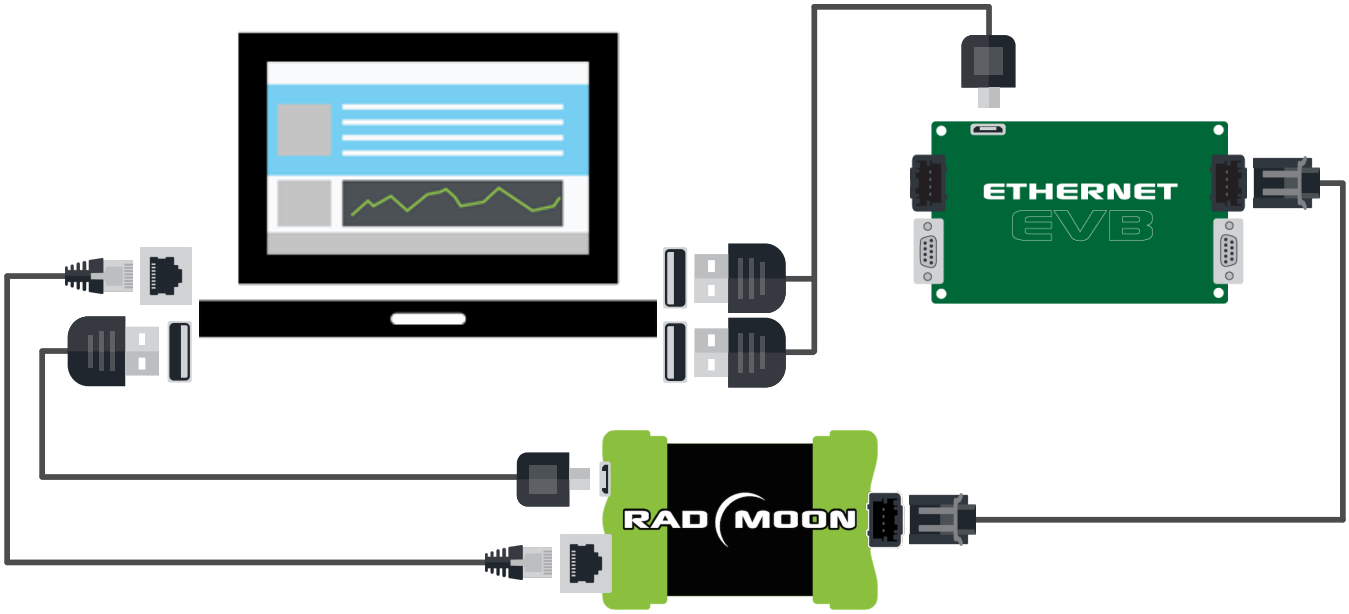


Figure 72: EEVB / RAD-Moon Hookup Diagram. The basic EEVB setup remains the same as before, except that its BroadR-Reach cable is detached from Node A and Node B is connected to the RAD-Moon's Mini50 connector instead. The RAD-Moon receives power over the connected USB cable from the PC, and is connected to the PC using a standard Ethernet cable. A USB hub and/or Ethernet switch can be used if necessary.

First, we connect the USB cable that provides power to the RAD-Moon.

- **Connect USB Cable:** Attach the full-sized USB connector to an open USB port on your PC, and the other end to the jack on the RAD-Moon box.

Next, we detach the BroadR-Reach cable linking Node A and Node B, and instead link Node B to the RAD-Moon.

- **Disconnect BroadR-Reach Cable from EEVB Node A:** Carefully press down the release lock on the BroadR-Reach cable connector attached to Node A of the EEVB and pull it from the jack.
- **Connect BroadR-Reach Cable to RAD-Moon:** Plug the connector you just detached into the BroadR-Reach jack of the RAD-Moon.

Finally, we connect the RAD-Moon to the PC over standard Ethernet. This can be done directly into a PC Ethernet port, but this will mean (temporarily) detaching any cable that may already be using that port. Alternately, you can use a hub or switch, plugging the PC into one of its ports and the RAD-Moon into another.

- **Connect Ethernet Cable:** Connect one end of a standard Ethernet cable to the Ethernet port on the RAD-Moon, and the other end to the PC Ethernet jack or a jack on a hub or switch to which the PC has already been connected.

Part 2.6B Set Up EEVB Node B to Send ARP Reply Messages

For this experiment Node B will be programmed to listen for *ARP Request* messages sent by the PC and to respond to them with *ARP Reply* messages. We have already created the necessary script, which we first employed in Lab 2.3, and can reuse here.

- **Load the Decoded ARP Reply Setup:** Load *2.3 ARP Reply Decoded* on the *Logon Screen*.
- **Download CoreMini to Node B.**

We aren't using Node A in this lab, but while we're here, let's just ensure that it is inactive.

- **Clear CoreMini from Node A.**

Part 2.6C Set Up Vehicle Spy 3 to Send ARP Request Messages

Next, we configure Vehicle Spy 3 to enable us to manually transmit *ARP Request* messages to the EEVB. We'll begin with the same *ARP Request* setup we used in Lab 2.2, and then make a couple of necessary modifications.

We're going to learn a new trick to help us here: working with multiple Vehicle Spy 3 instances. For clarity, we'll call the original instance that currently has *2.3 ARP Reply* loaded *Instance 2.3*, and the new copy of the program *Instance 2.2* (for reasons that will soon become obvious).

- **Start a New Vehicle Spy 3 Instance:** While leaving the current Vehicle Spy 3 window active, start the program again in the same way as usual. You should now see two Vehicle Spy 3 entries in the Windows task bar.
- **Load the ARP Request Setup:** Load *2.2 ARP Request* on the *Logon Screen* of Instance #2.

Why use two instances? We are going to be sending *ARP Requests* from Vehicle Spy 3 and so need that setup loaded into the program. However, we also need a receive message set up for the *ARP Reply* messages that come from the EEVB. This is already defined in the *2.3 ARP Reply* setup, so the easiest way to get it is to load both setups in different instances and then copy the *ARP Reply* message definition from Instance 2.3 to Instance 2.2.

Let's give it a shot.

- **Switch to Messages Editor in Instance 2.2, Receive Side.**
- **Copy ARP Request Message:** Click on ARP Request and then click  to copy the message.

Wait—we're copying the *ARP Reply* message *to* Instance 2.2, so why start in that instance by copying the *ARP Request* message definition? Well, we need to do so in order to enable the paste capability in this instance. It's just one of Vehicle Spy 3's charming little quirks. :)

- **Switch to Messages Editor in Instance 2.3, Receive Side.**

- **Copy ARP Reply Message:** Click on ARP Reply and then click .
- **Switch to Instance 2.2.**
- **Paste ARP Reply Message:** Click to paste the message into the Messages Editor on Instance 2.2.

We now have the definition we need, so there's no longer any need for Instance 2.3. There will also now be no more need to refer to instance numbers, so we'll drop that as well.

- **Close Instance 2.3.**

Remember that in Lab 2.5 we discussed that when transmitting from the EEVB we need each transmit message to also be defined on the receive side, but we don't need that when transmitting from within Vehicle Spy 3. Well, the *2.2 ARP Request* setup was originally created for sending from the EEVB. Now that we're going to send from the PC, we don't need the *ARP Request* definition on the receive side any more.

- **Delete ARP Request Message from Receive Side.**

Our message definitions are now how we want them, but we aren't quite done yet. We will want to use the *Tx Panel* to manually send *ARP Request* messages from within Vehicle Spy 3, while looking at the *Messages View* to observe the message exchanges. Since we want two views open, you guessed it, this is a job for the docking feature!

- **Dock Tx Panel at Bottom of Screen:** Drag the Tx Panel tab until the blue rectangle appears, continue dragging until the pointer is over the symbol, then release. (If you need more detailed instructions, please refer back to Part 2.2B.)
- **Switch to Messages View:** Click the Messages tab in the upper half of the window.
- **Reduce Size of Docked Tx Panel:** Drag the horizontal divider downward so that most of the screen is the *Messages View*, but the ARP Request message is still visible on the left side of the *Tx Panel*.
- **Close Details View:** Click Details to disable *Details View*. (It takes up a lot of space and we don't need it right now.)

The result of all of these changes should be a Vehicle Spy 3 window that looks something like Figure 73.

As usual, let's save this so we don't have to do all of that again!

- **Save the Setup File:** Save the current setup as 2.6 ARP Request from PC.

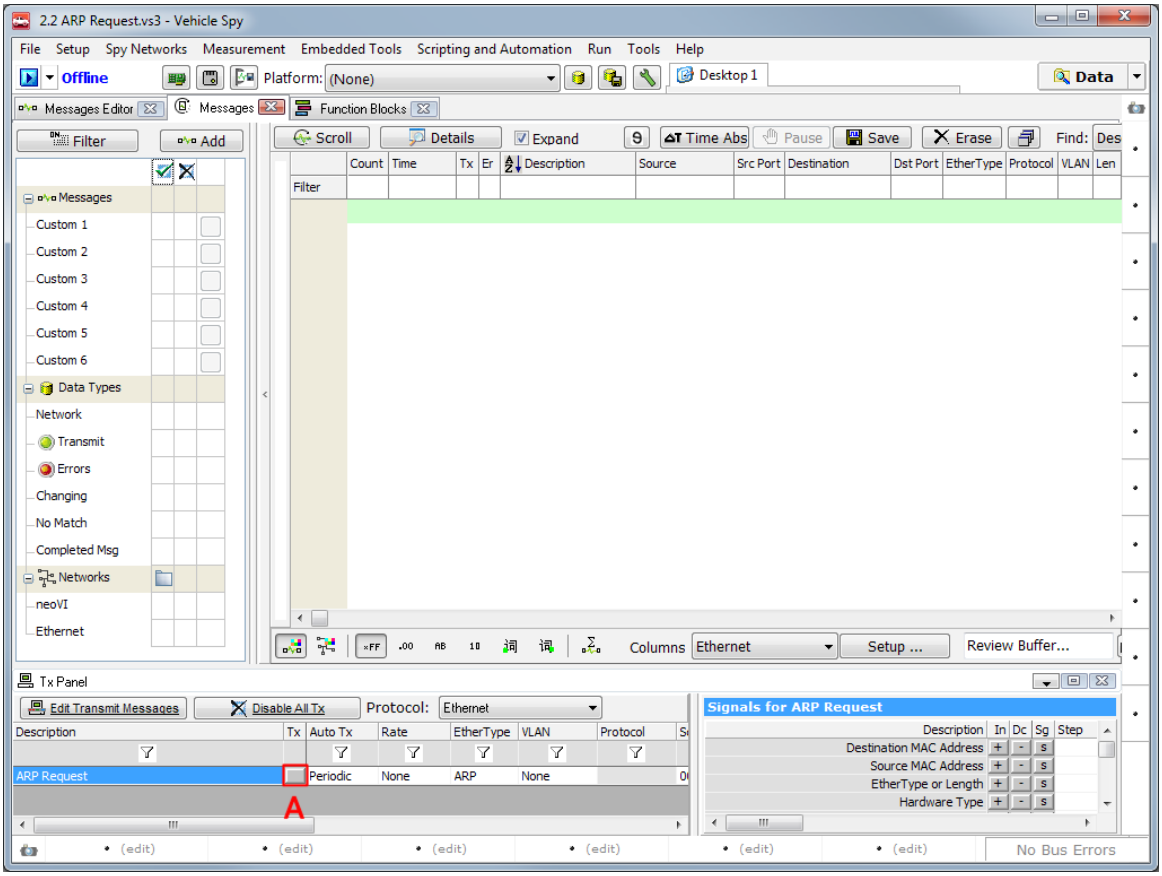


Figure 73: Messages View with Docked TX Panel. The Messages Editor takes up most of the screen, but with the Tx Panel occupying the bottom portion so we can access the manual transmit button (highlighted as item A).

Part 2.6D Manually Transmit ARP Request Messages from Vehicle Spy 3 and Observe ARP Reply Messages from EEVB

Let's now put our setup to work. We'll use the manual transmit button for the *ARP Request* message (Figure 73:A) to send that message, which will go through the RAD-Moon and be received by Node B of the EEVB. It will then respond with an *ARP Reply* message that will travel back through the RAD-Moon and be received and displayed by Vehicle Spy 3.

Since in this experiment we are transmitting over the PC's regular Ethernet connection and not the USB link to the EEVB, we must remember to change our Ethernet interface selection. (If you encounter any issues in this lab, there's a good chance you forgot to do this!) Also bear in mind that since you usually want the EEVB selected, that's the default when VSpy starts up; if you restart Vehicle Spy 3 you'll need to manually choose the non-EEVB interface again.

- **Change Ethernet Interface Selection:** Return to the *Logon Screen* and select the Ethernet interface corresponding to the PC's standard Ethernet port.

Now let's return to the *Messages Editor* (with docked *Tx Panel*) and get some traffic going.

- **Switch to Messages View.**

- **Filter to Show Only ARP Messages:** Enter ARP in the EtherType filter field.
- **Enable Scroll Mode.**
- **Go Online.**

You will now see displayed in the Messages View... absolutely nothing. The EEVB is waiting for an *ARP Request* message and we haven't sent any yet.

- **Transmit ARP Request Message:** Press the transmit button for the *ARP Request* message in the *Tx Panel*.

You should see *ARP Request* and *ARP Reply* messages appear. Of course they aren't sent at the exact same time, but it takes only a few milliseconds for the request to be recognized and the reply generated, so they seem almost simultaneous to the human eye. Note that the *ARP Request* message has the transmit symbol ( ) in the *Tx* column, indicating a transmitted message.

- **Transmit ARP Request Message Two More Times:** Press the transmit button twice more.

Now you will see three ARP exchanges in *Messages View* (Figure 74).

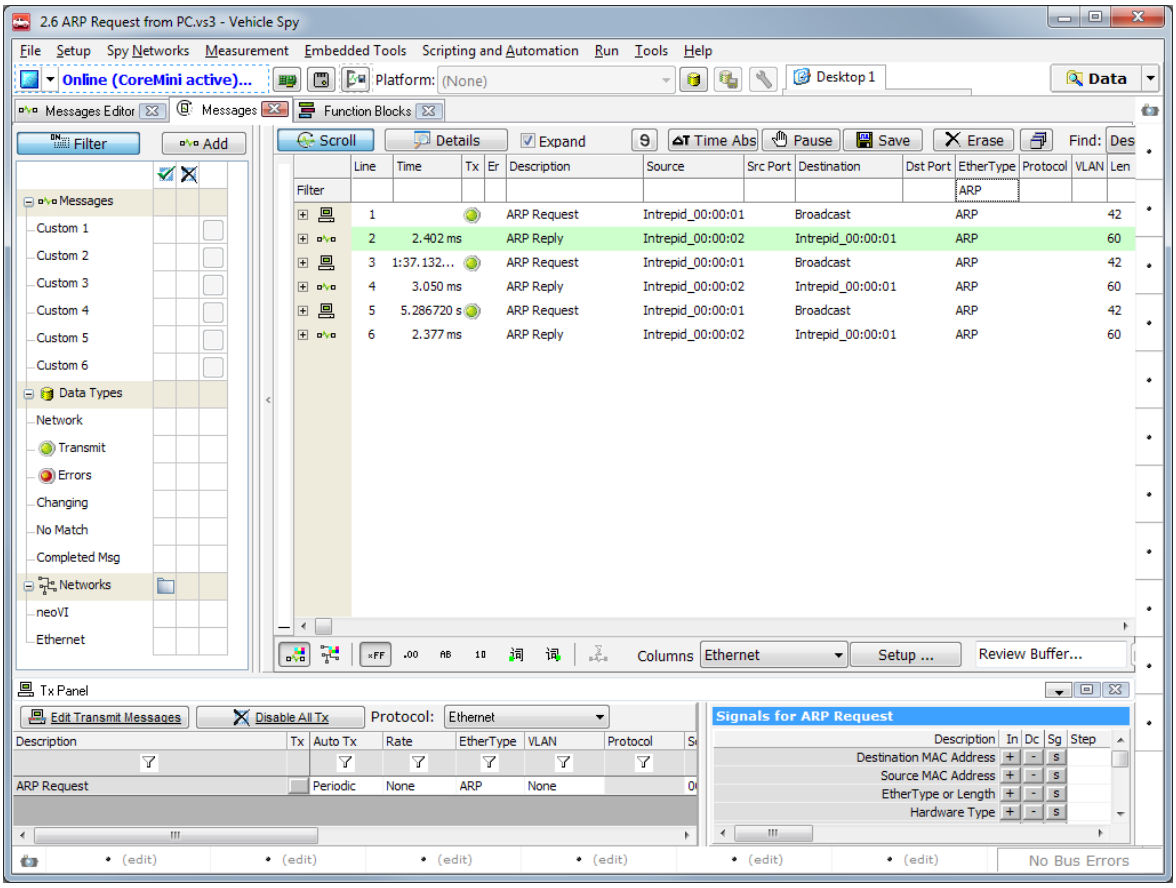


Figure 74: Messages View and TX Panel Showing ARP Exchange. We pressed the manual transmit button three times to send three *ARP Request* messages through the RAD-Moon to the EEVB, which responded with three *ARP Reply* messages..

Part 2.6E Set Up Vehicle Spy 3 to Regularly Transmit ARP Reply Messages—Two Ways

Being able to manually transmit messages by pressing the button in the *Tx Panel* for each one can be pretty useful. However, there are cases where we may need to keep our attention elsewhere, such as when diagnosing a problem. Here we are better off just having Vehicle Spy 3 send the *ARP Request* message periodically, just as we had EEVB Node A doing in earlier labs.

One way to do this is simply to use a function block script. Fortuitously, we already have the script we need in the current setup, because it was defined in *2.2 ARP Request* so we could send it to the EEVB. All we need to do is change the startup condition, just as we did in Lab 2.5.

- **Go Offline.**
- **Switch to Function Blocks.**
- **Change the Start Type:** Click the Start tab and change Start Immediately Embedded Only to Start Immediately.

And now let's go online and see what happens.

- **Switch to Messages View.**
- **Go Online.**

As you might expect, we now have an *ARP Request* message being sent about every three seconds, followed immediately by an *ARP Reply*—just as when the EEVB nodes were talking to each other.

Now let's try another method, which uses the *Tx Panel* in a different way. If you look under the *Auto Tx* column for the *ARP Request* message, you'll see that it is currently set by default to Periodic, which means that Vehicle Spy 3 will transmit the message regularly at a preset frequency. However, the entry under the *Rate* column is None, so automatic transmission is effectively disabled. Let's change that.

- **Go Offline.**
- **Change ARP Request Auto Tx Rate to Two Seconds:** Double-click the box for the *ARP Request* message under Rate and then click the arrow that appears on the right side of the box. Scroll down and select 2.000 from the drop-down box menu. Press *Enter*.

You will now see 2.000000 in the *Rate* column.

Let's disable the function block we were using earlier—we no longer need it.

- **Switch to Function Blocks.**

- **Change the Start Type:** Click the Start tab and change Start Immediately to Manual Start.

We also don't need the *Tx Panel* permanently docked any more, so let's free up some screen space.

- **Dock the Tx Panel at the Top:** Drag the Tx Panel window pane header to the top and release when the Tx Panel tab reappears.

And finally, let's see the results of our efforts.

- **Switch to Messages View.**
- **Go Online.**

Once again you'll see *ARP Request* and *ARP Reply* messages appear regularly, this time separately by about 2 seconds per pair.

Part 2.6F Restore EEVB-Only Configuration

We're done with the RAD-Moon, at least for now, so let's undo the changes we made and resume operation using only the EEVB.

- **Go Offline.**
- **Disconnect USB Cable from RAD-Moon.**
- **Disconnect BroadR-Reach Cable from RAD-Moon.**
- **Connect BroadR-Reach Cable to EEVB Node A.**
- **Disconnect Ethernet Cable from RAD-Moon and PC or Hub.**
- **Reconnect Ethernet Cable to PC Ethernet Port:** If you disconnected your main network connection for this lab, be sure to restore it.

Congratulations, you've completed Section 2 of the Intrepid Ethernet EVB Lab Manual!

Section 3 Simulations Using TCP/IP Internet Protocol (IP) and Internet Control Message Protocol (ICMP) Messages

In Section 1 of the Lab Manual we dealt with Ethernet messages, which are implemented at layers 1 and 2 of the OSI Reference Model; in Section 2, we experimented in detail with ARP, which interfaces between layers 2 and 3. As we move logically up the layer stack, the next step is layer 3 itself, called both the *Network Layer* and the *Internet Layer*. Here we find what is arguably the most important protocol in all of networking, the *Internet Protocol* (*IP*), and its adjunct, the *Internet Control Message Protocol* (*ICMP*).

In this section you'll learn about these two essential components of TCP/IP and work with them using the Ethernet EVB and Vehicle Spy 3. You'll achieve these objectives:

- Learn about the Internet Protocol and its two versions, IPv4 and IPv6.
- Capture IP packets and analyze their headers to see how IP works in the "real world".
- Generate and then analyze different types of ICMP messages.
- Create and transmit IP messages using the EEVB.
- Implement different functionality on each of the EEVB nodes without needing two separate setup files.
- Use a Vehicle Spy 3 signal list to more easily see signal values.
- Set up a signal plot to display signal values graphically.
- Run two CoreMini scripts in parallel on the EEVB.
- Make an EEVB simulation of the essential *ping* utility.
- Set up a request/reply ICMP exchange between the PC and EEVB, optionally including the RAD-Moon media converter.
- Learn how to use graphical panels to display message information and control the operation of Vehicle Spy 3.
- Simulate the generation of an ICMP error message.

Once again, we assume here that you have already completed earlier sections of the Lab Manual. Also, we will omit even more of the detailed, step-by-step instructions for common tasks, such as switching between views or downloading CoreMinis, since you should be quite comfortable by now with most of them.

Before beginning, we present a brief summary of IP and ICMP functions and operation for those who are new to these protocols. As always, a full description is well beyond the scope of the Lab Manual, but these technologies are covered in great detail in Chapters 15 to 25 of *Automotive Ethernet - The Definitive Guide*, included in your Ethernet EVB package.

An Overview of the Internet Protocol (IP)

Even though its name comes second, the Internet Protocol is actually the heart of the TCP/IP protocol suite. Layer 3 of the OSI model is primarily concerned with the delivery of data across an arbitrary internetwork (or *internet*) of networked devices, and in TCP/IP the Internet Protocol is what facilitates this essential function. As the name also suggests, the Internet Protocol is also the basis of operation of the global Internet.

IP has the following core functions:

- **Addressing:** IP addresses are familiar to everyone in the world of technology; we've already used them in the Lab Manual. They provide the means for uniquely identifying devices across an internetwork of potentially billions of devices.
- **Data Encapsulation and Message Formatting:** Virtually all higher-level protocol messages are encapsulated within IP messages (also called *packets* or *datagrams*).
- **Routing and Datagram Delivery:** IP plays a key role in ensuring the accurate and efficient routing of data between different network types in diverse geographic locations.

IP is designed to be a relatively simple protocol so that it can operate at a high speed. It is an *independent* protocol because it seamlessly carries data from thousands of different higher-level protocols. It is also called *unreliable* because it doesn't guarantee delivery of data, nor provide acknowledgments when data is received. This was a deliberate design choice, allowing reliability to be added in higher layers when needed, while allowing more-efficient basic data transport when it was not required. Finally, IP is said to be *universal* because nearly every modern computing device can understand the protocol.

There are two versions of IP, which are known as *IPv4* and *IPv6* (versions 1, 2, 3 and 5 don't exist for various reasons). IPv4 is our focus in the Lab Manual, because it is simpler to understand and still by far the more commonly used of the two. IPv6 was developed to address IPv4's shortcomings in areas such as address space size and routing efficiency, but is only very slowly supplanting its predecessor.

Incidentally, you may notice that despite underscoring the importance of IP, we don't seem to have many labs devoted to using it. This is a case where appearances can be deceiving, though. Only one lab features IP *by itself*, because that's not typically how IP is used—as mentioned above, it's the common data encapsulation of other TCP/IP protocols. All of the labs in this section that use ICMP also use IP, and the same is true of all the labs in Section 4.

An Overview of the Internet Control Message Protocol (ICMP)

Because IP is used in every transaction in TCP/IP, it was important to keep it as simple and streamlined as possible. For this reason, many of the administrative tasks required to keep an internetwork running smoothly were offloaded to a secondary protocol called the *Internet Control Message Protocol*, or *ICMP*. By way of analogy, IP could be thought of as a busy executive, and ICMP as the skilled assistant that allows that executive to get things done quickly and efficiently without worrying about small details.

Because of the special relationship between these two protocols, ICMP occupies a rather unique place in TCP/IP. ICMP messages are carried within IP packets, much like those of higher-level protocols such as the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). However, ICMP is charged with housekeeping tasks that support IP rather than with data transport. For this reason, ICMP is generally considered to reside at layer 3 of the OSI model, rather than layer 4 as is the case with TCP and UDP.

ICMP defines a set of messages that can be conveyed between internetworked devices to communicate administrative data. These are generally broken into two categories:

- **Error Messages:** These messages provide feedback to a source device of a problem condition, such as the inability to deliver a message or an invalid value in a message field (Figure 75).
- **Informational Messages:** These are used to query and provide information, perform testing or implement special features.

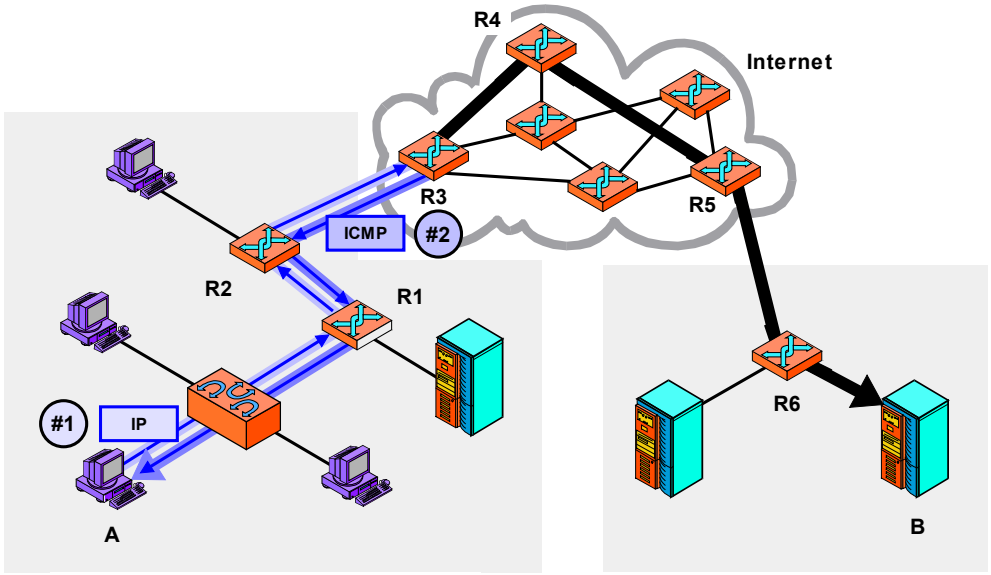


Figure 75: ICMP General Operation. A typical use of ICMP is to provide a feedback mechanism when a problem occurs with an IP message. In this example, device A is trying to send an IP datagram to device B (thin, light arrow). However, when it gets to router R3, an issue of some sort is detected that causes the datagram to be dropped. R3 sends an ICMP message (thick, dark arrow) back to A to tell it that something happened, hopefully with enough information to let A correct the problem if possible. Note that R3 can only send the ICMP message back to A, not to R2 or R1.

Like a good administrative assistant, ICMP does a lot of work in the background that isn't noticed by others. In addition to helping resolve various issues in a network, its informational messages are used to implement several essential networking tools that are used to test and manage network configurations. We will use one of these, ping, extensively in this section of the lab manual.

Just as there are two versions of IP, there are also two versions of ICMP. These are called ICMPv4 and ICMPv6 to make it easy to match them to their IP counterparts.

Lab 3.1 Examining IP and ICMP Messages and Some Common Network Utilities

As we have done in the last two sections, we'll ease into our exploration of IP and ICMP messages by capturing and analyzing some created by actual TCP/IP network devices before we start creating our own. As part of this process we will also play with a couple of common network utilities that employ ICMP to implement their functionality, making it easy for us to observe that protocol in use.

Again, as with earlier "real world" labs, you will need to ensure that the PC running Vehicle Spy 3 is connected to an actual network in order to see the messages. An Internet connection—either direct or through a router—is particularly well-suited to this lab.

Part 3.1A Go Online and Explore IP Filtering Variations

Since this is a new section of the lab manual, we'll begin with a new instance of Vehicle Spy 3 to ensure that any settings from previous labs have been cleared.

- ► **Close Vehicle Spy 3.**
- ► **Start Vehicle Spy 3.**

Now we'll once again select our regular (non-EEVB) Ethernet interface, so we can look at real-world traffic. To save time, we'll also load the custom column setup we made in Lab 1.1 so that the *Messages View* is set to show Ethernet traffic.

- ► **Select Active Regular Ethernet Interface.**
- ► **Load 1.1 Custom Column Setup.**

You will probably be automatically switched to *Messages View*, but if not, do so manually.

Recall that back in Lab 2.1 we entered a filter in the *EtherType* column so that we'd only see ARP messages and not the others that appear on the network. We needed that because ARP messages make up only a small percentage of the traffic on a typical TCP/IP network. That's definitely not the case with IP, which we can easily prove by *not* entering a filter and just going online. Let's do this in scroll mode so we can see the messages as they arrive.

- ► **Enter Scroll Mode.**
- ► **Go Online.**

You will now likely see a steady stream of TCP/IP traffic, with nearly all of it having either **IPv4** or **IPv6** in the *EtherType* column.

If we do want to make sure we only see IP messages, this is of course simple.

- ► **Set IP EtherType Filter:** Enter **IP** in the EtherType column filter box.

Non-IP messages are now suppressed.

Of the messages being displayed, the lion's share will be for IPv4 specifically; to see only those that are IPv6, we just need a more specific filter.

► **Set IPv6 EtherType Filter:** Change the EtherType column filter to IPv6.

The stream of displayed traffic will now slow down dramatically, since most networks don't use IPv6 a great deal yet. Of course we can also filter just for IPv4.

► **Set IPv4 EtherType Filter:** Change the EtherType filter to IPv4.

New IPv6 messages are now hidden and IPv4 messages shown.

Part 3.1B Analyze an IPv4 Message

Much as we did when looking at ARP messages in Part 2.1C, we will now explore the IPv4 message format more deeply, examining the individual header fields in a sample packet. One difference is that you may recall from our look at ARP that we used a utility to force the computer to generate ARP messages, since it can take some time for them to appear on their own. But as you saw above, that's certainly not an issue with IPv4 messages, many of which will appear on a typical Internet connection every second!

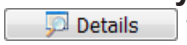
In fact, there are probably so many messages coming in right now that the VSpy display scrolls constantly. We only need a sample so we can just go offline for the moment.

► **Go Offline.**

Next, simply select a sample message you want to use for this analysis; they are all IPv4 because of the filtering we did earlier.

► **Select an IPv4 Message:** Click on any IPv4 message in the *Messages View*.

We'll look at individual fields in *Details View*. You may or may not have this enabled at present, depending on which parts of Section 2 you completed.

► **If Necessary, Enable Details View:** If *Details View* is not currently enabled, click 

to turn it on.

There will probably be four entries in the left pane of the *Details View*. The first will be *Message on Ethernet*, which is the general entry that Vehicle Spy 3 makes to denote an Ethernet message as opposed to one from a different automotive network. The second will be *Ethernet*, containing the details of the Ethernet header; you may recall that we examined these way back in Part 1.2G when we first learned about the *Details View*. The third entry is *Internet Protocol Version 4*, which is what we want, while the fourth will be for the higher-layer protocol carried in the IPv4 message, most likely either UDP or TCP.

► **Expand the IPv4 Message Header:** Click the ⊞ button to the left of Internet Protocol Version 4 in the *Details View*.

You will now see some of the individual fields of the IPv4 message you chose. Since there are many of them, you most likely will only see a few at first and will need to scroll down to view

the ones later on in the header. They will also likely be cut off on the side. It's probably worth increasing the size of the Details View for this exercise.

Before we do, notice that one of the fields has a ⊞ button of its own, which is hiding several subfields. Let's expand that so we can see everything.

- **Expand the IPv4 Flags Field:** Click the ⊞ button to the left of Flags in the *Details View* area for the already-selected message.

Now, let's make it easier to see what we're dealing with here.

- **Increase Size of Details View:** Drag the horizontal bar separating the *Details View* from the main *Messages View* window until all of the IPv4 header fields are visible. Also drag either or both of the vertical bars to the right, since we don't need to see the *Name/Value* area or the byte fields right now.

Ah, much better (Figure 76).

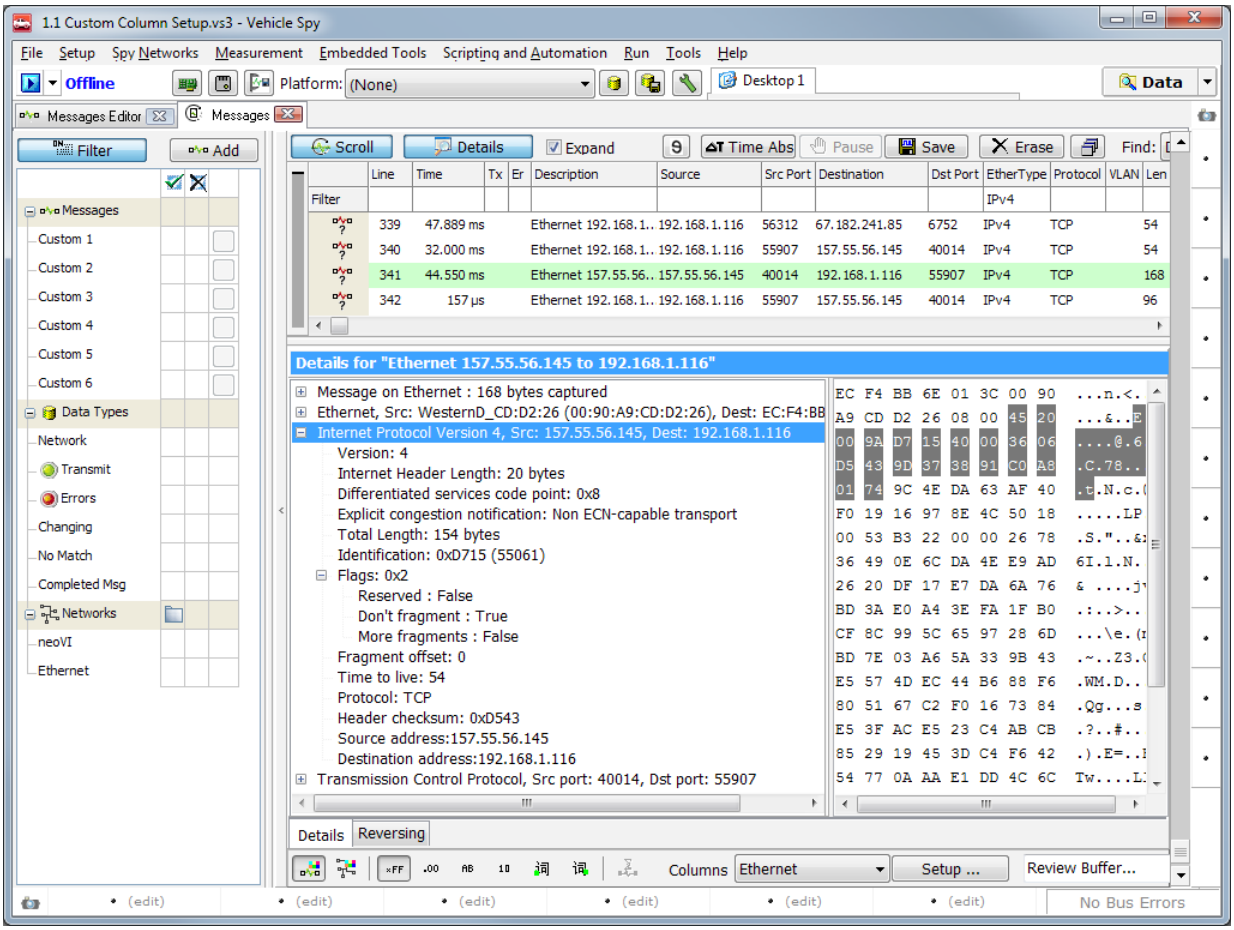


Figure 76: IPv4 Message Headers in Details View. For this exercise we selected an IPv4 message at random from our Internet connection. We expanded the IPv4 entry, as well as the *Flags* subsection, to show all the fields in the header. We then expanded the size of the information window so we could see all of the IPv4 header fields without needing to scroll the window.

Here's a short rundown of IPv4 header fields and what values you will typically find in each:

- **Version:** Always 4, since we are looking at IPv4 messages.
- **Internet Header Length:** The length of the IPv4 header, measured in 32-bit words. In most cases the header is 20 bytes long, so the field will contain the literal value 5; Vehicle Spy 3 shows the equivalent byte length for convenience.
- **Differentiated Services Code Point and Explicit Congestion Notification:** These fields are both used as part of advanced, optional features in IP, which we'll ignore at present.
- **Total Length:** The overall length of the IP packet. This will normally be equal to 14 less than the length of the entire message as indicated in the *Len* column in the *Messages View*, as is the case in Figure 76. (Do you know why the length difference is 14? Think about what carries the IPv4 message, and how many bytes its header requires.)
- **Identification:** Used to tag fragments that are part of the same original message so they can be reassembled correctly.
- **Flags:** A three-bit field containing two information flags and one reserved bit:
  - **Don't Fragment (DF):** Set to a 1 to force devices on the network to keep the current message whole.
  - **More Fragments (MF):** This is 0 for an unfragmented message. When a message is fragmented, it is set to 1 in every fragment except the last one to tell the recipient to look for more fragments before beginning reassembly of the original packet.
- **Fragment Offset:** Indicates which portion of the original datagram a particular fragment represents; this is used in message reassembly.
- **Time to Live:** The number of *hops*, or inter-device links, a packet may traverse before being discarded. This is used to prevent network errors, such as routing loops, from causing a packet to circulate on an internetwork forever. As we'll see later in the lab, it can also be used for certain network analysis functions.
- **Protocol:** The reserved number of the next-higher-layer protocol that the IPv4 message is carrying. Vehicle Spy 3 decodes this into a protocol name for you, which will match the *Protocol* column in the Messages View for the message.
- **Header Checksum:** This is computed by the transmitter of the packet and inserted in this field. The recipient then repeats the checksum calculation; if there's a mismatch, this indicates a transmission error and the packet is discarded.
- **Source Address and Destination Address:** The IP addresses of the transmitter and intended recipient of the message; these match the *Source* and *Destination* columns in the *Messages View*.

Note that, as with ARP, IPv4 messages are often used in request/reply pairs. IP is itself only a "common carrier" of messages in TCP/IP; it doesn't define specific message types such as the *ARP Request* and *ARP Reply* used in ARP. Instead, these are implemented by the higher-layer protocols that use IP. There will typically be many data exchanges taking place simultaneously via IP packets, however, which can make matching up requests and replies for analysis more difficult than was the case with ARP. However, in the next part of the lab we'll see that we can isolate some of the IP traffic to more easily see specific transactions.

We'll need more real estate for the main *Messages View* soon, so let's trim the *Details View* back down to size and collapse the IPv4 message headers.

- **Reduce Size of Details View:** Move the vertical and horizontal dividers to return the *Details View* to approximately its normal dimensions.
- **Collapse the IPv4 Message:** Click the ⊟ button next to the IPv4 message to hide the individual fields.

Part 3.1C Generate and Analyze ICMP Echo Request and Echo Reply Messages Using Ping

Now that we've looked at the overall IPv4 message format, we will shift our attention to its "helper" protocol, ICMP. To see these messages, we simply add a more specific filter to the one we used before so that only ICMP messages encapsulated within IPv4 are displayed. Since we are using version 4 of IP, we'll only to look at ICMPv4 and not ICMPv6.

- **Set Protocol Filter to ICMPv4:** Enter ICMPv4 as a Protocol column filter.
- **Enter Scroll Mode:** You probably already are in scroll mode, but if not, enable it.
- **Go Online.**

In theory, we should now see some ICMP messages appear. In practice, you may find that none show up at all, even after a full minute.

Why? Remember back in Section 2 that we said ARP messages represent overhead and so devices on the network try to keep them to a minimum. Well, the same is true of ICMP, only even more-so. These messages are only usually used to convey errors or to implement network testing functions. Even though IP is considered "unreliable", in real-world networks errors with IP packets are rare, so you won't often see ICMP errors.

Fortunately, we can use network utilities to generate ICMP messages for analysis purposes. The easiest and most common way to do this is with the *ping* utility. This tool, which is found in every modern operating system, is named after the same term used in sonar, referring to generating a sound pulse and then waiting for it to bounce off objects to determine their distance. In a network, you "ping" another device on the network by sending ICMP *Echo Request* messages to it, and waiting for it to respond back with *Echo Reply* messages. This enables you to verify that the other device is operating and that you are able to reach it over

the network. The ping utility also measures the time it takes for the message exchange, allowing “distance” to be determined in terms of network latency.

Let’s give it a whirl.

- **Open Command Prompt:** In the Windows Start Menu, enter cmd in the box labeled *Search programs and files*. When the program *cmd.exe* appears, right-click it and choose Run as administrator.
- **Clear Messages View:** Press Erase to clear any ICMP messages that may have appeared while you were reading.
- **Ping Google:** In the command prompt, enter ping google.com and press *Enter*.
- **Go Offline.**

You should see output showing the results of ping’s attempt to send four *Echo Request* to Google’s web site. (The IP address may differ from the one in the figure, as Google has many of them.) All four *Echo Request* messages should receive *Echo Reply* messages back, and the time required for each response will be indicated (Figure 77).

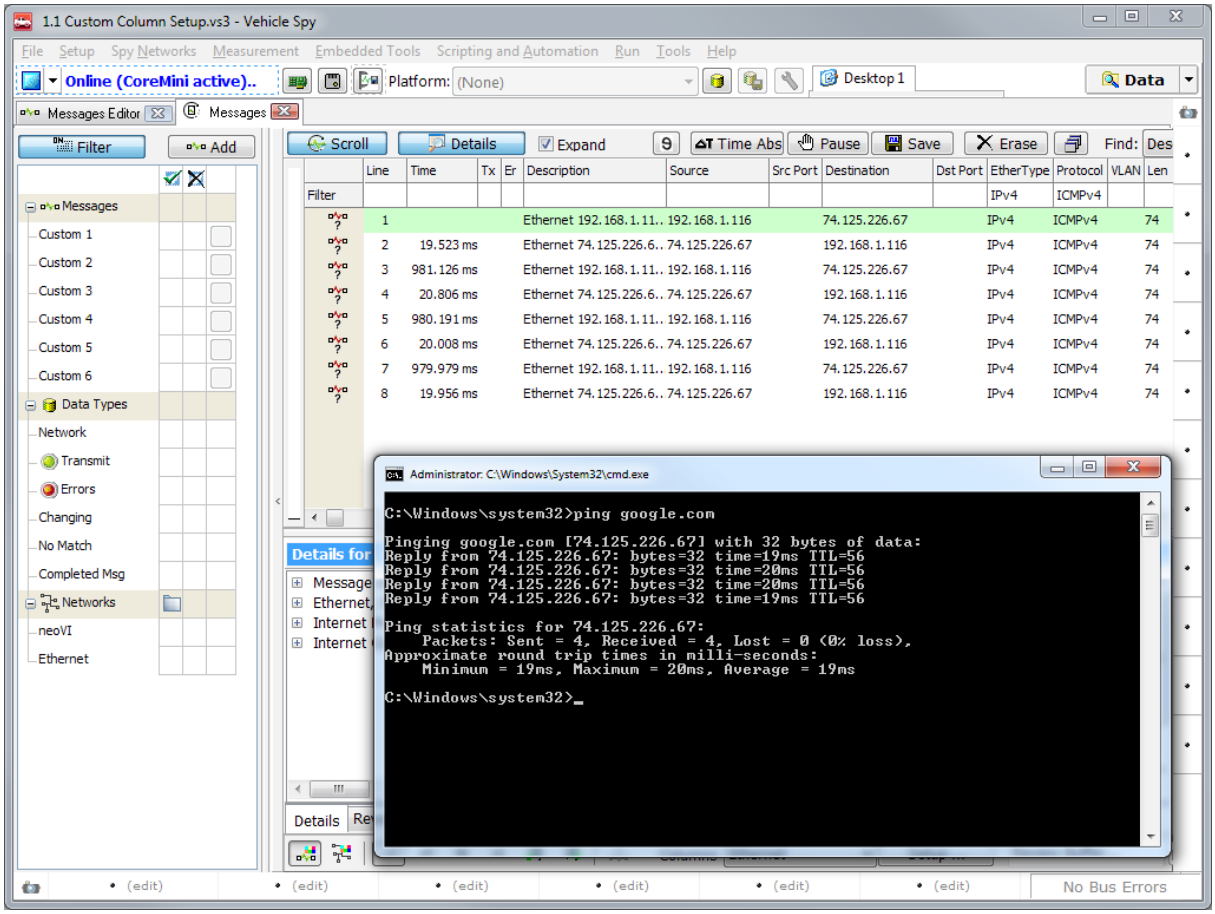


Figure 77: Pinging Google to Generate ICMP Messages. By default, the ping utility sends four *Echo Request* messages to a host, one every second. It reports back how much time elapsed between sending the *Echo Request* and getting a matching *Echo Reply*, or indicates if no response was received.

In Vehicle Spy 3 you will see eight messages, corresponding to the four *Echo Request* / *Echo Reply* exchanges that the ping utility generated. Take a look at the values in the *Time* column and you'll notice that the *Echo Request* transmissions are separated from each other by about one second, while the times for the *Echo Reply* messages closely match the time figures reported by ping. (Note, however, that the Windows version of ping seems to round down all values to whole milliseconds).

Let's take a look at an *Echo Request* message.

- **Select the First ICMP Message:** Assuming that scroll mode is on, the first message should be an *Echo Request*.

If you look in the *Details View*, you'll see that Vehicle Spy 3 has decoded the ICMPv4 message type for you right in the summary line, based on the values of the *Type* and *Code* fields, which are also shown. Let's see what the header fields look like.

- **Expand the Message :** Click the ⊞ button in the Details View to the left of Internet Control Message Protocol.

As you can see in Figure 78, there's not a whole lot here. ICMP is a relatively simple protocol, and *Echo* messages among its simplest variants. The *Type* and *Code* fields have already been discussed. The *Identifier* and *Sequence Number* fields are made available by the protocol definition to optionally help devices match *Echo Request* messages with their corresponding *Echo Reply* messages. In the case of this ping implementation, we can see that the *Sequence Number* field is used while the *Identifier* field is not (it's always 1).

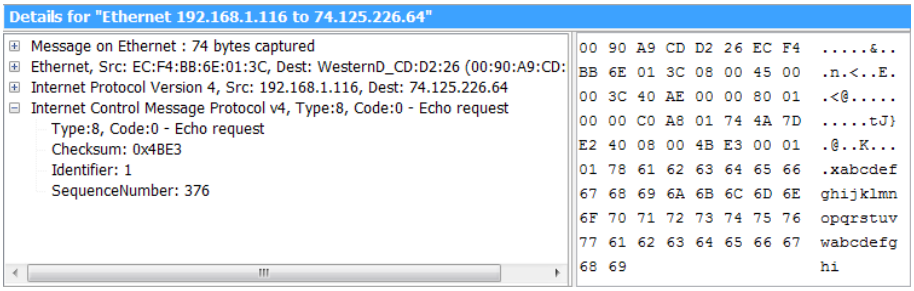


Figure 78: ICMPv4 Echo Request Details.

Make a note of the value of the *Sequence* field in the first (*Echo Request*) message, and then...

- **Select the Second ICMP Message:** Click on the second ICMP message, which should be an *Echo Reply*.

You will notice that the *Sequence* field is the same as for the *Echo Reply*.

- **Select the Third ICMP Message.**

This will be the second *Echo Request*; its *Sequence Number* field number should be one larger than in the first *Echo Request*. The ping program will keep incrementing this value to try to ensure unique message exchanges, even across multiple invocations of the utility.

Beyond the ICMP header there are 32 bytes of data, which if you look in the byte display area in Figure 78, you will see are just sequential lower-case letters (though oddly, “xyz” are omitted). An *Echo Request* can optionally contain any data here, which the recipient must “echo” back in its *Echo Reply*.

Leave the command prompt open for the next part of the lab.

Part 3.1D Generate ICMP Error Messages Using Traceroute

In our earlier look at IPv4 header fields, we discussed the *Time To Live* (*TTL*) field, which is used to help avoid problems that might cause a packet to bounce around an internetwork forever. When a packet is first transmitted, the sender chooses a value for this field that can range from 1 to 255. Each time the packet is handed from one device to the next, the recipient decrements the value in the *Time To Live* field. If it ever reaches zero, the packet is considered to have “expired” and is discarded, and the device sends an ICMP *Time Exceeded* error message back to the originating device.

Sometimes it is useful to be able to discover the specific route that a packet will take across an internet (or The Internet) between individual devices. However, there is no specific mechanism built into TCP/IP to implement this function. A few years after IPv4 and ICMP were first defined, a computer scientist devised a clever way of using the IP *Time To Live* field to determine the route to a device. The utility that implements this method, called *traceroute*, uses the following algorithm:

1. **Send Probe with TTL Value of 1:** A device sends a packet, usually either UDP or ICMP, to the host whose route it wishes to trace. It sets the *Time To Live* value of the packet to 1. The first router that receives the packet will decrement this value, and since it will now be 0, will discard the packet and send an ICMP *Time Exceeded* message back to the originator. That message will have the router’s IP address, so this will tell the originator the address of the first hop in the transmission.
2. **Send Probe with TTL Value of 2:** The device next sends a packet with a *TTL* value of 2. This time the first router will decrement the field, yielding a valid value of 1, so the message will be passed to the next router. This time *that* router will be the one that decrements *TTL* to 0, discarding it and sending a *Time Exceeded* message.
3. **Send Probes with TTL Values of 3+:** This process continues, with the originator increasing the *Time To Live* value by 1 each time until a packet gets all the way to the host being tested.

The Windows version of traceroute—called *tracert* because of the historical Microsoft limitation of 8 characters for filenames—uses ICMP to generate its probes, and of course receives back ICMP *Time Exceeded* messages, making it perfect for illustrating ICMP. One slight drawback is that instead of sending only one probe for each *TTL* value, it sends three, so there are three times as many messages to sift through. But that’s not a big deal.

We’ll now do a traceroute and take a look at the ICMP messages that are exchanged. The instructions below assume that you still have the command prompt open from the last part of

the lab. We also will assume you still have Vehicle Spy 3 running, in scroll mode, and set to filter only for ICMPv4 messages.

- ► **Clear Messages View.**
- ► **Go Online.**
- ► **Trace Part of the Route to Google:** In the command prompt, enter tracert -d -h 3 google.com and then press the *Enter* key.
- ► **Go Offline.**

You should receive a total of 18 messages, which will be received in 3 groups of 6 with a few seconds separating each set. The -h 3 parameter tells tracert to stop after the first three routers are found, since that makes the messages easier to analyze, while -d makes the process a bit quicker. There are 18 messages because three probes are sent for *TTL* values of 1, 2 and 3, yielding a total of 9, and each receives an error message in reply (Figure 79).

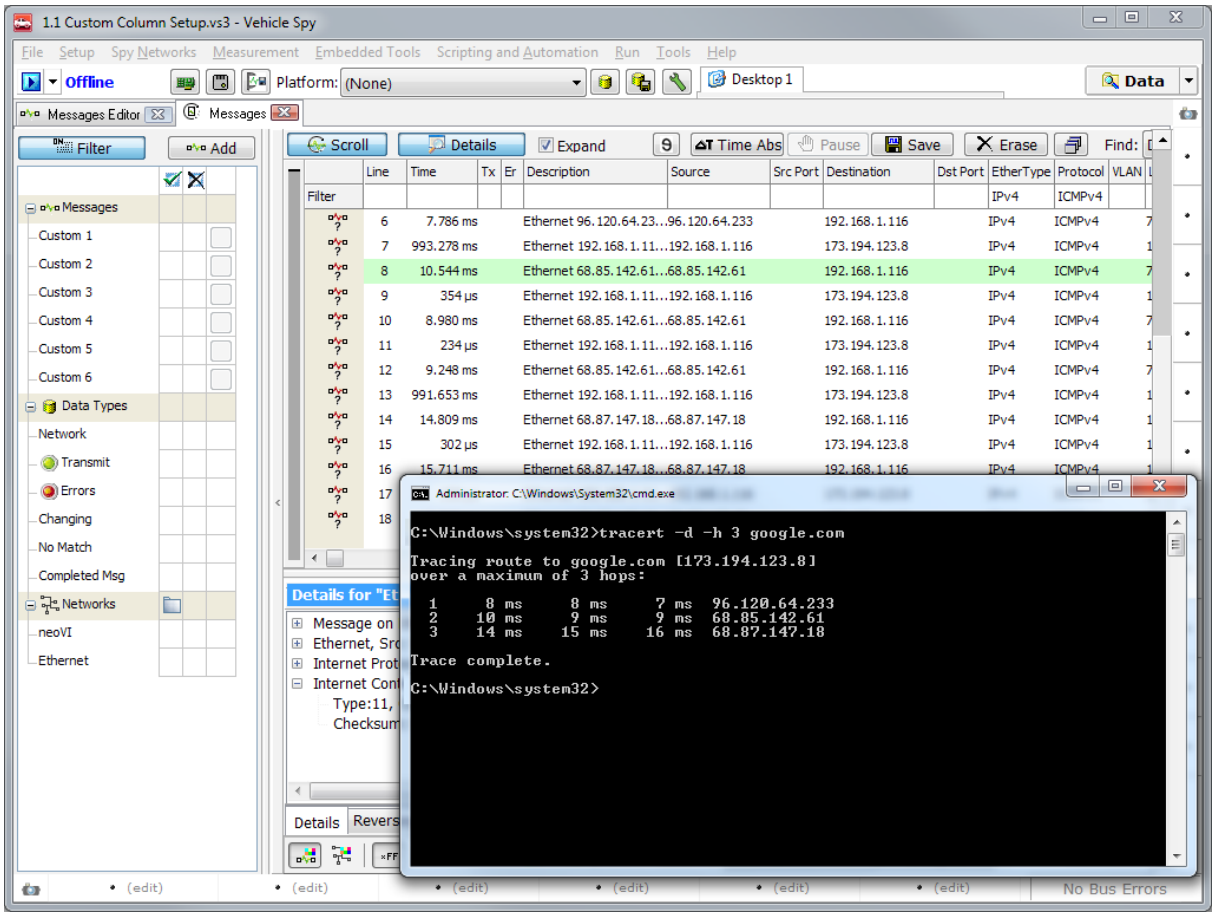


Figure 79: Using Traceroute to Generate ICMP Echo Request and Time Exceeded Messages. In this example we asked tracert to attempt to find the IP addresses of the first three hosts between us and Google. This results in the generation of 3 sets of 3 ICMP *Echo Request* messages, each of which has its IPv4 *Time To Live* field set to generate an ICMPv4 *Time Exceeded* message, resulting in a total of 18.

- ► **Select ICMP Message #1.**

Notice in the *Details View* that this is an ICMP *Echo Request* message.

► **Expand the IPv4 Message Header in the Details View.**

Here you will see that the *Time To Live* value for this message is 1, as expected.

► **Select ICMP Message #7.**

Notice that the *Time To Live* field value is now 2.

► **Select ICMP Message #13.**

And now it's 3, again as anticipated.

► **Select Any Even-Numbered Message.**

These should all be ICMP messages with a *Type* of 11 and *Code* of 0, which are *Time Exceeded* messages corresponding to the *Time To Live* field expiring in transit (Figure 80). Note that there is another type of *Time Exceeded* message as well, which is generated if it takes too long to reassemble a fragmented packet; this has a *Code* value of 1.

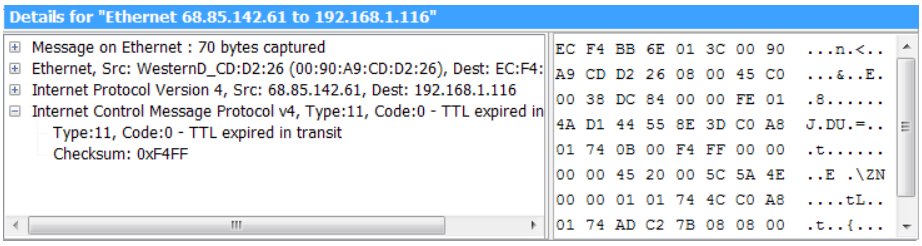


Figure 80: ICMP Time Exceeded Message Details.

And that will conclude our little journey into the magical world of ICMP messages. Let's tidy up...

► **Close Command Prompt:** Enter **exit** in the command prompt and press *Enter* to close the box.

► **Enter Static Mode.**

Lab 3.2 Creating and Transmitting Custom IP Datagrams

As we mentioned in the introduction, we aren't going to do a lot of work with IP messages by themselves, because real TCP/IP traffic generally consists of data from higher-layer protocols carried within IP packets. That said, it makes sense to start with just IP before complicating matters with additional protocols. Our work here will help us gain familiarity with IP, setting the stage for working with messages that are carried within it, including ICMP, UDP and TCP.

In this lab we'll program both of the EEVB nodes to send a simple message that has just Ethernet and IP headers, with a custom data field carrying a sequential counter. We'll also continue learning more about how to do things with CoreMinis, this time by creating a script that allows Node A and Node B to behave differently despite running the same setup.

Part 3.2A Reset Vehicle Spy 3 to Known Working State

To ensure that we have no leftover settings from earlier labs, and to accommodate those who may be starting this section from scratch, let's restart Vehicle Spy 3. We'll also load the custom column setup file we created in Lab 1.1 so that the *Messages View* is set to show Ethernet traffic.

- **Close Vehicle Spy 3.**
- **Start Vehicle Spy 3.**
- **Load Custom Column Setup File:** Double-click `1.1 Custom Column Setup` under either the Recent or My Setups tab.

Part 3.2B Create and Customize IPv4 Transmit and Receive Messages

We will create two IPv4 transmit messages, one for each of the two nodes to send on a regular basis. First, the message that Node A will send.

- **Switch to Messages Editor, Transmit Side.**
- **Create a New Transmit Message:** Click the ✚ button to create a new transmit message.
- **Rename Transmit Message:** Rename the message from Tx Message Ethernet 1 to Node A IPv4.
- **Change EtherType to IPv4.**

After this is done, tabs will appear in the signals area: *Ethernet Header*, *IPv4 Header* and *IPv4 Data*.

- **Add Data Signal:** Click the button in the signals area.
- **Rename Data Signal:** Change the name of the signal from Signal 18 to Node A Counter.

As always, default values for all the fields in this message will be supplied by Vehicle Spy 3 using its built-in templates. You can customize these using the *Ethernet Packet Template Editor*, just as we did in Lab 1.4 for raw Ethernet frames, but we won't do so now. As we have also done before, we can make individual modifications to field values using the *Tx Panel*, but we don't actually need to change anything for this message.

Now let's make a message for Node B, following the same steps with minor variations.

- **Create a New Transmit Message.**
- **Rename Transmit Message:** Rename the message from Tx Message Ethernet 2 to Node B IPv4.
- **Change EtherType to IPv4.**
- **Add Data Signal.**
- **Rename Data Signal:** Of course, this time we'll call it Node B Counter.

This message will have the same default values as the message for Node A. Since this message is for Node B, that's not what we want, so this time we do need to use the *Tx Panel*.

- **Switch to Tx Panel.**
- **Select Node B IPv4 Message.**
- **Swap Destination MAC Address and Source MAC Address Values:** Change the *Destination MAC Address* value to 00:FC:70:00:00:01 and the *Source* to 00:FC:70:00:00:02, the opposite of what we had for Node A IPv4.
- **Swap Source IP Address and Destination IP Address:** Repeat the swap for the IP addresses in the IPv4 header.

Finally, we need to ensure we have receive messages to match these transmit messages so that they are properly recognized by Vehicle Spy 3.

- **Switch to Messages Editor.**
- **Copy Node A IPv4 Message to Receive Side.**
- **Copy Node B IPv4 Message to Receive Side.**

Our messages are now done. Let's save this file to ensure we keep these changes (and so we don't overwrite *1.1 Custom Column Setup*).

- **Save the Setup File:** Save the setup file as 3.2 IPv4 Transmit.

Part 3.2C Create Function Block to Transmit IPv4 Messages

As mentioned at the start of this lab, unlike in previous examples, we are not making different setups for the two EEVB nodes—we will have only one function block script for both nodes. In

order to have this script behave correctly, it must be able to determine dynamically which node it is running on. Fortunately, there's a simple trick that makes this easy.

Vehicle Spy 3 allows function block scripts to access a number of special internal values that correspond to hardware device characteristics. We used these earlier in the Lab Manual when we had scripts take actions based on EEVB pushbutton presses or the value of potentiometer dials.

Another special value present in Vehicle Spy 3 is the serial number of the device running the script. The EEVB is designed so that Node A always has an even-numbered serial number, while Node B's serial number is that of Node A plus 1. We can thus simply use the modulo (division remainder) function to determine which node the script is running on.

- **Switch to Function Blocks.**
- **Add a New Function Block Script.**
- **Add If Statement in Step 1.**
- **Enter If Statement Condition:** Double-click on the Value field, and in the *Expression Editor* select Physical IO from the menu on the left. Expand the General neoVI Hardware section using the + button, and double-click Device Serial Number. Then click in the Expression field after the closing curly brace and add mod 2 = 0. Press *Enter* or click OK.
- **Enter If Statement Comment:** Use this comment: Modulo 2 returns 0 for an even number, representing Node A, or 1 for an odd number, which is Node B..
- **Add Set Value Statement.**
- **Enter Set Value Condition:** Open the *Expression Editor* for *Step 2*, select Tx Messages from the left menu and under Node A IPv4 double-click Node A Counter. The value {Node A Counter (Value) :out0-sig18-0} appears in the Value To Set field. Now press the Add To Expression button to add this same entry to the Expression field. Click at the end of it and add +1. Press *Enter* or click OK.
- **Add Set Value Comment:** Increment counter each time Node A message is sent..
- **Add Transmit Statement:** Create a *Transmit* statement in *Step 3* to send *Node A IPv4*.
- **Add Else Statement:** This separates the code that runs on Node A and that used by Node B.
- **Add Set Value Statement.**
- **Enter Set Value Condition:** This will be the same as the previous Set Value condition, with two changes. First, select Node B Counter rather than Node A Counter. Second, decrement the counter rather than incrementing it (so put -1 at the end of the Expression field).
- **Enter Set Value Comment:** Decrement counter each time Node B message is sent..

- **Add Transmit Statement to Send Node B IPv4.**
- **Add End If Statement.**
- **Add Wait For Statement:** After adding the statement, double-click on the Value field and then press = to open the *Expression Editor*. Then enter a value of 100 in the Expression field. This is to slow down the EEVB so it's not sending hundreds of messages per second.

You can leave the two extra blank steps. The result should appear similar to Figure 81.

Step	Description	Value	Comment
1	If	{Device Serial Num} mod 2 = 0	// Modulo 2 returns 0 for an even number, representing Node A, or 1 for an odd number, which is Node B.
2	Set Value	{Node A Counter (Value) :out0-sig18-0} = {Node A Counter (Value) :out0-sig18-0}+1	// Increment counter each time Node A message is sent.
3	Transmit	Node A IPv4	
4	Else		
5	Set Value	{Node B Counter (Value) :out1-sig18-0} = {Node B Counter (Value) :out1-sig18-0}-1	// Decrement counter each time Node B message is sent.
6	Transmit	Node B IPv4	
7	End If		
8	Wait For	= 100	
9			

Figure 81: Function Block Steps for IPv4 Message Transmit.

Note: Technically it is poor programming practice to increment or decrement a variable without first initializing it to a known value. However, Vehicle Spy 3 automatically sets all variables to 0 for you if they are not initialized to another value, a provision we are exploiting here for simplicity.

The script is now done except for changing it to only run on the EEVB.

- **Change Start Type to Start Immediately Embedded Only.**

And now let's be sure to save our work.

- **Save the Setup Under the Current File Name:** Select Save from the *File* menu.

Part 3.2D Download and Test IPv4 Transmit Script

We will now send this script to both EEVB nodes and go online to see how well it works.

- **Send CoreMini to Node A.**
- **Send CoreMini to Node B.**
- **Switch to Messages View.**

- **Filter for Custom IPv4 Messages:** Enter IPv4 in the Description field as a filter, so we only see our messages.
- **Go Online.**

You should see *Node A IPv4* and *Node B IPv4* messages arriving approximately every 100 ms, as expected. Let's take a look at the *Node A Counter* and *Node B Counter* fields; since IPv4 messages have a lot of fields, it's easier to do this with *Details View* off, especially on smaller screens.

- **Turn Off Details View.**
- **Show Signals in Node A IPv4:** Click the ⊞ next to Node A IPv4 in *Messages View*.

You should see a display similar to Figure 82. Notice that *Node A Counter* increases in value with every message receipt, rolling over back to 0 after it hits 255.

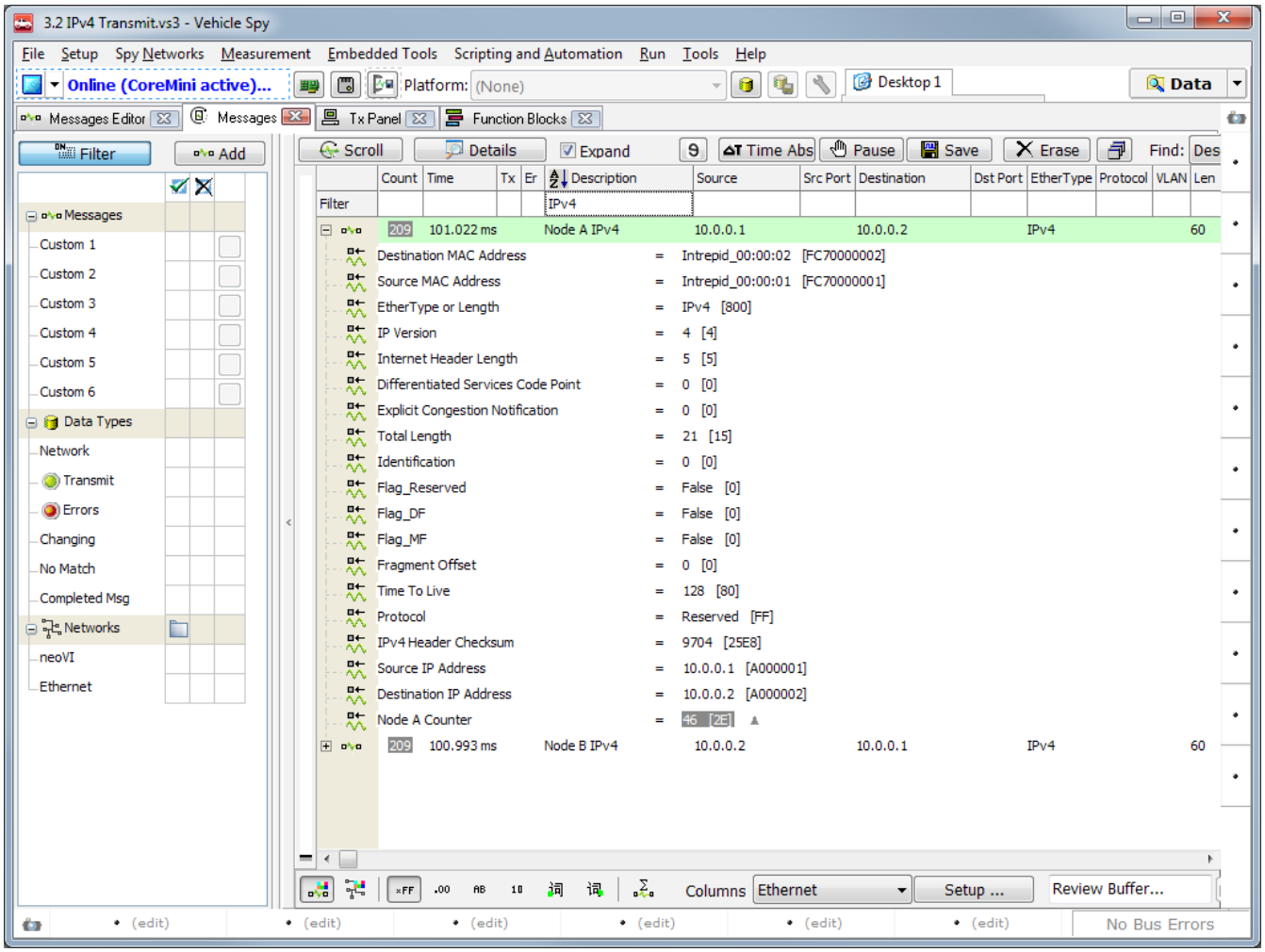


Figure 82: Node IPv4 Message Signals.

- **Collapse Signals in Node A IPv4.**

► **Show Signals in Node B IPv4.**

Conversely, *Node B Counter* goes down until it hits 0, and then rolls over to 255 and begins again.

► **Collapse Signals in Node B IPv4.**

Leave Vehicle Spy 3 active and online for the next lab.

Lab 3.3 Using Signal Lists and Plots to Display Data and Adding a Second Simultaneous CoreMini Script for Node Synchronization

We're now going to tinker with the script we wrote in the previous lab to tell the EEVB's nodes to send IPv4 messages. First, we'll create a *signal list* to allow us to isolate the counter fields we are interested in monitoring. Second, we'll graphically display those counters using a *signal plot*, and tailor the window setup so we can see the plot and incoming messages at the same time. Finally, we will write a second script to address a minor issue with the original one, and have both nodes run both scripts at the same time.

This lab assumes that the EEVB and Vehicle Spy 3 have been left in the state they were in at the conclusion of Lab 3.2. If this is not the case, you can get back to the right place by reloading the *3.2 IPv4 Transmit* setup file you created in that lab, and then following the steps in Part 3.2D.

Part 3.3A Create a Signal List to Isolate Specific Message Fields

In the previous lab we were able to look at the increasing *Node A Counter* field in the messages coming from Node A, and its decreasing counterpart, *Node B Counter*, in those originating with Node B. However, it was kind of a pain to do, because these fields were buried deep within messages with many fields. What if we had 10 or 20 such fields to examine rather than two? It would be very hard to keep track of them.

This is why Vehicle Spy 3 provides a perfect tool to make it easier to hand-pick specific signals and monitor their values.

- **Switch to Signal Lists:** Select *Signal List* from the *Measurement* menu.

You will see a large blank area with three columns: *Signal*, *Value* and *Update*. Above these headers is a drop-down box currently with the value **Default** in it. Vehicle Spy 3 allows you to keep track of many signals, and in some applications it is useful to be able to organize these into *signal groups* that you can switch among. In our case we are only interested in two signals, so we'll just use the default signal group.

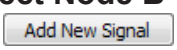
- **Select Signals:** Click the 

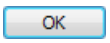
button.

A dialog box appears that looks very much like the *Expression Editor* we have seen a number of times before. The main difference is that instead of text entry boxes at the top left, there's a large box where signals are displayed when selected.

- **Select Node A Counter:** Select Rx Messages from the menu on the left, then find Node A Counter under the Node A IPv4 message and double-click it.

The entry **Node A Counter (Value)** appears in the signal list at the top left.

- **Select Node B Counter:** Repeat the process for Node B Counter. (You can also press the 

button instead of double-clicking.)
- **Finish Signal Selection:** Click 

to save the signal selections you have made.

You will be returned to the signal list screen, which will now display the two counters we selected. The *Value* column will display their value in real time, while the *Update* column contains a “spinner” to show that the value is changing—a useful feature when you have large numbers of signals to monitor (Figure 83).

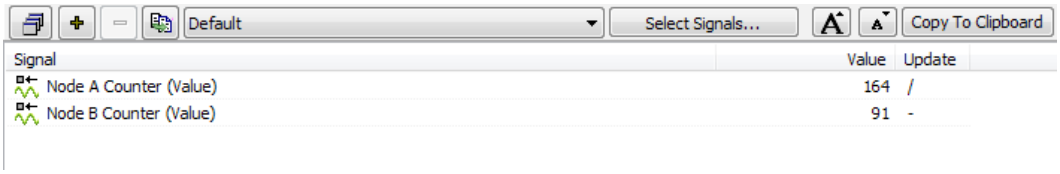


Figure 83: Signal List Showing Node A Counter and Node B Counter.

> **Note:** If the signals sit at 0 without updating, check to make sure you selected *Node A Counter* and *Node B Counter* under *Rx Messages* rather than under *Tx Messages*. The fields in the transmit messages don’t change because they are being incremented in the EEVB nodes, not within Vehicle Spy 3 on the PC.

Part 3.3B Create a Signal Plot to Display Counter Data Graphically

Signal lists allow us to focus on only the data we want to examine, while leaving uninteresting fields or signals behind. However, they still suffer from the problem of just being text displays of numeric (or alphanumeric) values. Suppose that instead of a very regular increasing or decreasing value, we had a signal that varied with some degree of randomness, and we needed to watch for rare but important spikes or dips in the value. In some cases the fluctuation we are looking for might last only a fraction of a second. This would be very hard to detect with a text display, but immediately obvious in a graphical presentation.

As they say, a picture is worth a thousand words; in Vehicle Spy 3 “pictures” are created using *signal plots*.

- **Switch to Signal Plots:** Select Signal Plot from the *Measurement* menu.

Boom! That was easy (Figure 84). The signal plots for our two messages are immediately displayed because the *Signal Plot* and *Signal List* features share the same signal definitions. Using this graphical display we can easily see our increasing and decreasing counters.

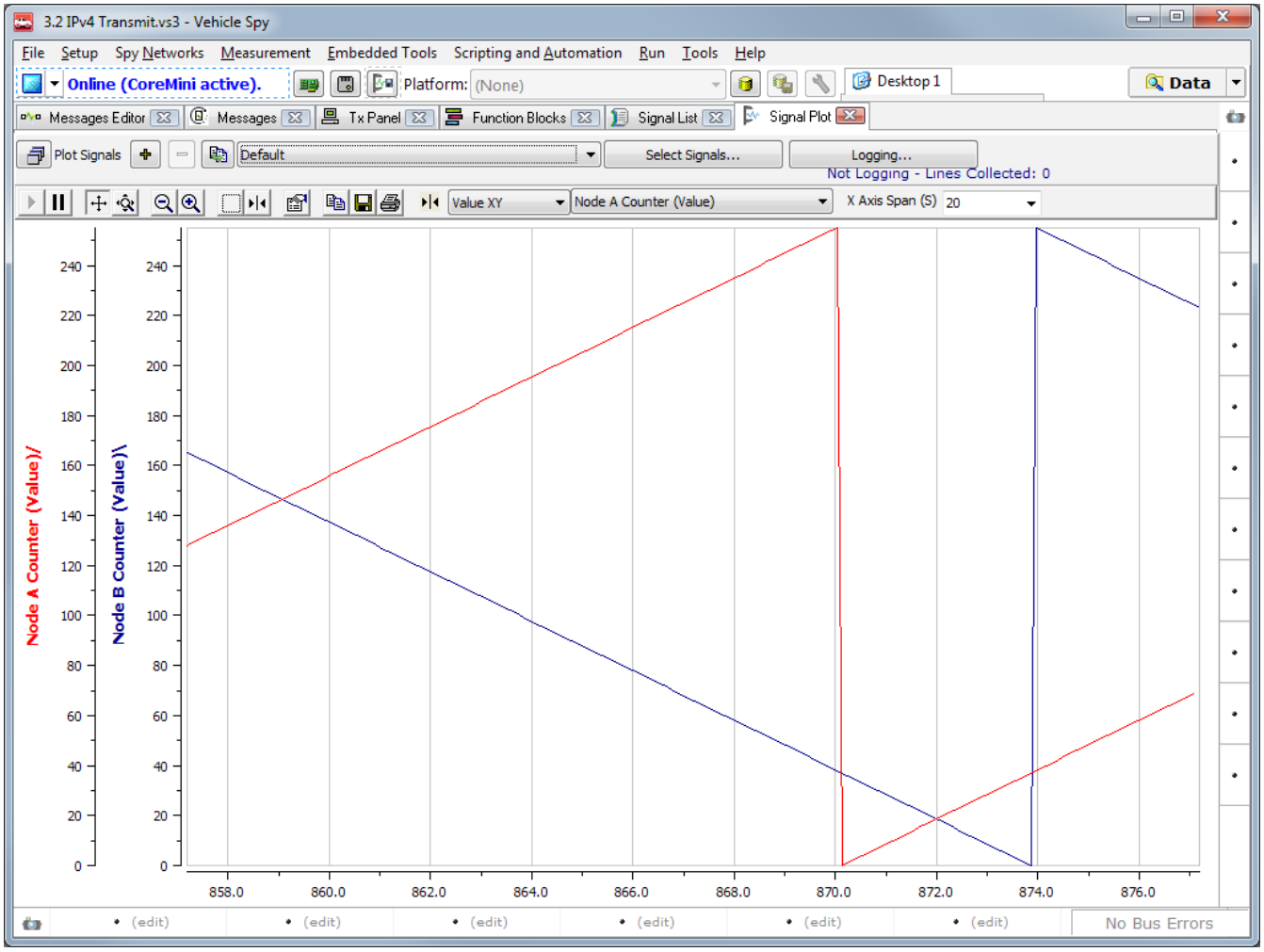


Figure 84: Signal Plot of Node A Counter and Node B Counter.

Of course we can easily adjust the plot to suit our needs. For example, we can change the amount of time for which the signals are displayed, allowing us to examine more cycles of the counter.

- **Change the X-Axis Span Value:** On the right side just above the signal display is a text box next to the label X Axis Span (S). Click the downward arrow and select 100 (you can also type this value in the box).

The display will clear and begin drawing the signal plots again; this time they will be more compressed, allowing 100 seconds of data to be shown.

By default, Vehicle Spy 3 will automatically scale the Y axes to fit the data values it sees in each signal, but we can change these as well if we wish.

- **Enter Signal Plot Properties:** Press the button.
- **Change the Node B Counter Y Axis:** Click Node B Counter (Value), then click the Axes tab. Change the 255 value in the Span box to 500. Click OK.

The signals will resume plotting again, but this time Node B Counter's value will only peak at about half the vertical height of the display. This looks weird, so let's change it back.

- **Revert the Node B Counter Y Axis Change:** Go back where you were and change the 500 value back to 255.

Finally, what if we want to look at signal plots while still being able to see messages coming into Vehicle Spy 3 in the Messages View? You probably already know the answer: we just dock the *Signal Plot* window on the screen and then bring up *Messages View* next to it. This time, let's try docking to the top of the display rather than the bottom.

- **Dock the Signal Plot Window to the Top:** Click and drag the Signal Plot tab until the blue rectangle appears. Drag it to the docking symbol on the top of the window, then release.

Vehicle Spy 3 "squishes" the plot to fit into the top half of the screen. Now we just select the *Messages View* to display on the bottom.

- **Switch to Messages View:** Click the Messages tab, which should now be located near the middle of the Vehicle Spy 3 screen.

There, we can now see our signal data graphically while viewing our incoming messages at the same time (Figure 85).

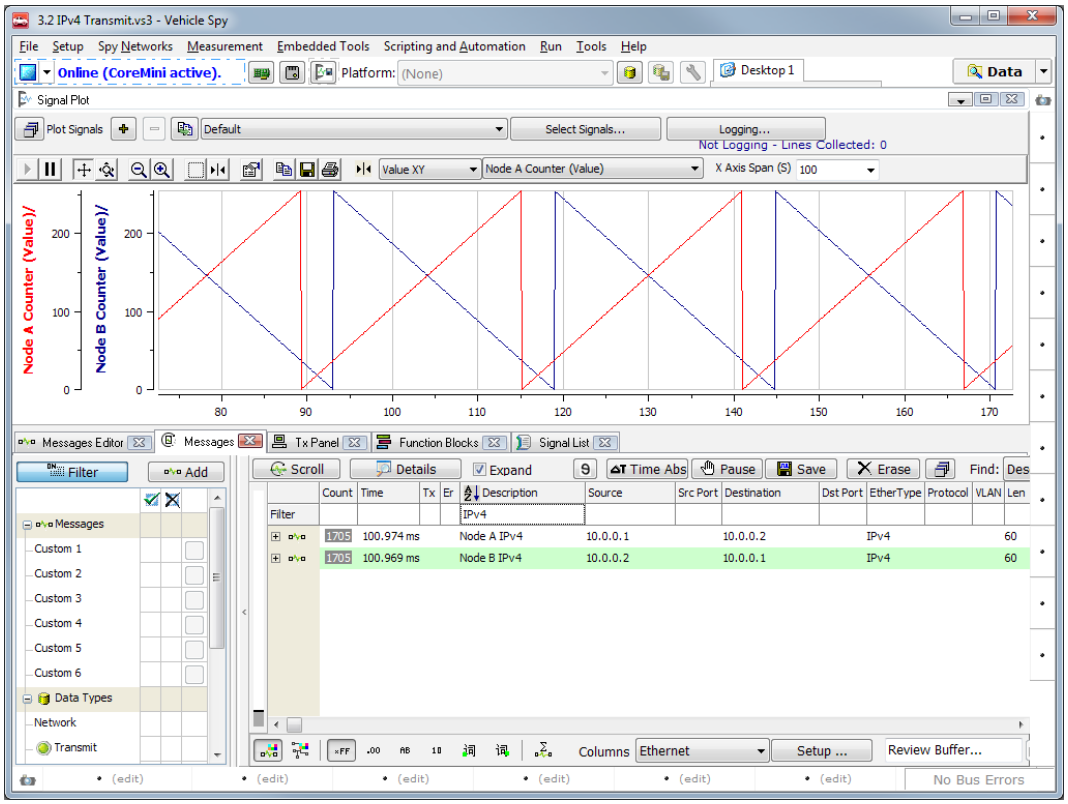


Figure 85: Signal Plot and Messages View of Node A Counter and Node B Counter.

When you're ready, let's go back to a single view.

- **Undock Signal Plot Window:** Drag the 

header until the blue rectangle appears, continue over to the window tabs area, and release.

Part 3.3C Synchronize the EEVB Nodes Using a Second Script

If you examine the signal plot we just created, you may notice something curious. Both nodes start with their counters at 0, and one increments it each time while the other decrements it. This means that the counters should go 1, 2, 3... and 255, 254, 253... so that their sum is always 256 (or 0). It also should mean that both signals "wrap around" at about the same time, which would be visible on the plot as both curves "snapping" up or down nearly simultaneously. Yet you will probably see a gap between the vertical lines due to one "wrapping around" well before the other, as was the case in both Figure 84 and Figure 85.

Think for a moment about why this might be.

Not sure? Here's an exercise that may make it more clear.

- **Go Offline.**
- **Send the CoreMini to Node A.**
- **Immediately send the CoreMini to Node B.**
- **Go Online.**

Observe the approximate delay between when the red curve hits the top and the blue curve hits the bottom, which can be "eyeballed" as the distance between the red and blue vertical lines when the respective curves wrap around. Then do the following.

- **Go Offline.**
- **Send the CoreMini to Node A.**
- **Wait Approximately 10 Seconds.**
- **Send the CoreMini to Node B.**
- **Go Online.**

You'll now see that the distance between the red and blue lines is much larger (Figure 86). That's right—the reason the two counters don't wrap around at the same time is because of the time delay between when we send the CoreMini to Node A and when we send it to Node B. Even though the programs seem to be synchronized, they start at different times, because the CoreMini begins running as soon as it is downloaded on each node.

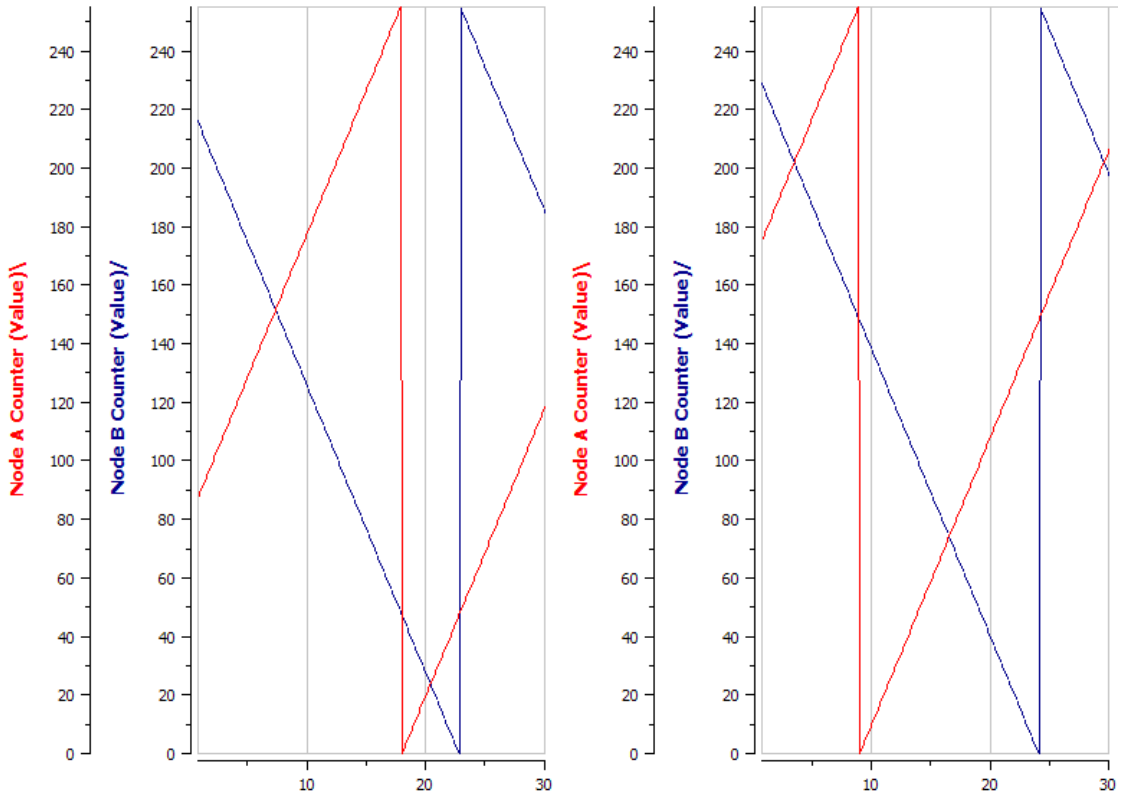


Figure 86: Signal Delay Comparison. On the left, the signal plot when the CoreMini is sent to Node A and then immediately to Node B; the red line hits 255 and wraps around about 5 seconds before the blue line hits 0. On the right, we wait about 10 seconds between sending to the two nodes, causing this delay to increase to around 15 seconds.

We can correct this in a number of ways. For the purposes of illustration, what we will do is create a second script that allows us to manually synchronize the nodes using the EEVB's pushbutton inputs. This illustrates a cool feature of the EEVB and CoreMinis in general: you can have more than one at a time and they will all run in parallel at the same time, automatically, when downloaded.

- **Go Offline.**
- **Switch to Function Blocks.**

The structure of this script will be similar to the one we already have, since it also has to check which node it is running on. To save time, why not duplicate the existing script and then modify it?

- **Duplicate Function Block 1:** Click the existing function block, Function Block 1. Click the button and then the button to create a duplicate called Function Block 2, and then click that entry to select it.
- **Switch to the Script Tab (If Necessary).**

One thing we don't need to be different on each node is the pushbutton check; each node automatically only references its own inputs. So we just need a single *If* statement surrounding

the whole script. It will reference *Switch 1* in the *Physical IO* area, which should ring a bell from earlier labs.

- **Add If Statement Before Step 1.**
- **Enter If Statement Condition:** Double-click on the Value field, choose Physical IO, scroll down to Switches and double-click on Switch 1. Press *Enter* or click OK.
- **Enter If Statement Comment:** Reset counter to 0 when node pushbutton is held down..
- **Add End If Statement At End of Script:** This should be Step 10 now.

Now we change our two Set Value statements so they set the counters to 0 rather than incrementing or decrementing them.

- **Edit Step 3 Value Entry:** Double-click the Value field for the first *Set Value* command, change the Expression field to 0, and press *Enter*.
- **Edit Step 3 Comment:** Change the comment to Reset Node A Counter to 0..
- **Repeat Above Two Steps for Step 6:** Of course, the comment this time will reference *Node B Counter*.

We aren't transmitting anything, of course, so let's get rid of those commands. We also don't need the Wait For statement.

- **Delete Both Transmit Statements.**
- **Delete Wait For Statement.**

And that's it, we're done; the resulting script should appear as in Figure 87.

Step	Description	Value	Comment
1	If	{Switch 1 (Value) :neo0-sw0-0-index(0)}	// Reset counter to 0 when node pushbutton is held down..
2	If	{Device Serial Num} mod 2 = 0	// Modulo 2 returns 0 for an even number, representing Node A, or 1 for an odd number, which is Node B.
3	Set Value	{Node A Counter (Value) :out0-sig18-0} = 0	// Reset Node A Counter to 0.
4	Else		
5	Set Value	{Node B Counter (Value) :out1-sig18-0} = 0	// Reset Node B Counter to 0.
6	End If		
7	End If		

Figure 87: Function Block Steps for IPv4 Message Transmit Synchronization.

Let's save our work.

- **Save the Setup File:** Save the setup file as 3.3 IPv4 Transmit Synchronized.

Part 3.3D Download and Test Synchronization Script

As always, we now send the script to both EEVB nodes to test it out.

- **Send CoreMini to Node A.**
- **Send CoreMini to Node B.**

► **Switch to Signal Plot.**

► **Go Online.**

At first nothing will appear to have changed.

► **Hold Down the Node A Pushbutton.**

► **Wait 5 Seconds.**

► **Release the Node A Pushbutton.**

As long as the button is down, the red curve will be roughly flat and near the bottom of the display. This makes sense since the pushbutton causes the counter to continually be reset to 0.

► **Repeat the Above with the Node B Pushbutton.**

Same deal, but naturally this time the value is constant at 255.

► **Hold Down Both Pushbuttons.**

► **Release Both Pushbuttons Simultaneously.**

Letting go of both buttons at the same time synchronizes the two nodes. They will now both hit 0 simultaneously, causing one to snap down at the same time the other snaps up (Figure 88).

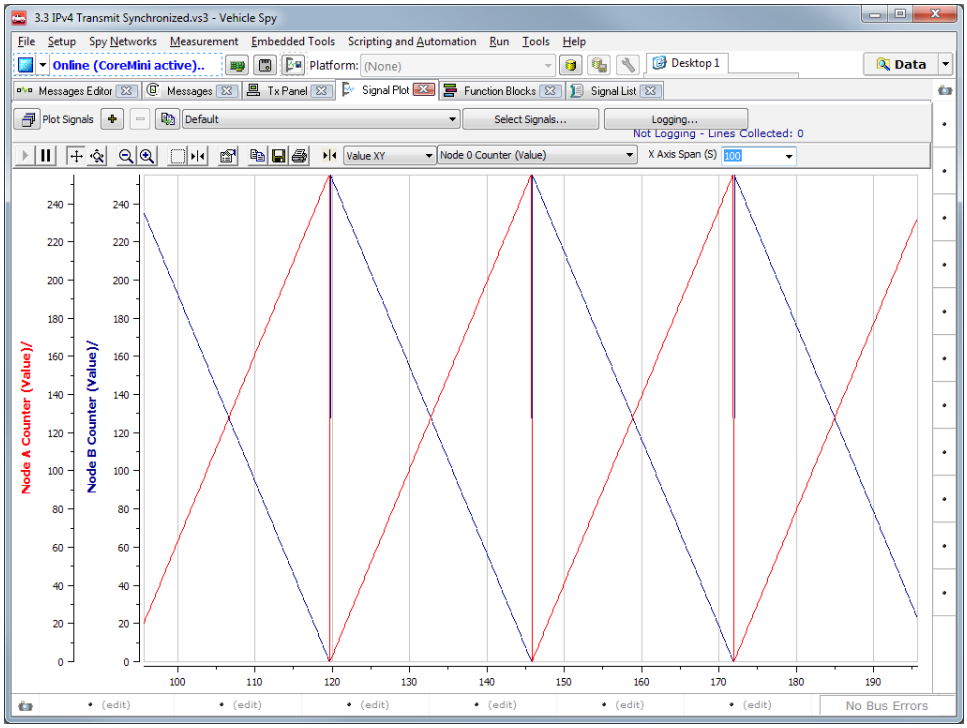


Figure 88: Signal Plot of Node A Counter and Node B Counter After Synchronization.

► **Go Offline.**

Lab 3.4 Simulating the Ping Utility and Monitoring Ping Exchanges Using a Graphical Panel

In Lab 3.1 we introduced the *ping* utility, which uses ICMP messages to test connectivity and time latency between two hosts on a network or internetwork. In this lab, we will implement our own simulation of this important yet simple tool, by setting up one node of the EEVB to initiate pings while the other replies to it. On the PC, we will introduce Vehicle Spy 3's powerful *graphical panel* feature, using it to monitor the exchanges of *Echo Request* and *Echo Reply* messages that our simulated ping client and server generate.

To save time, we will use pre-made setup files in this lab, and for the first time we'll have three of them: one for each EEVB node and one for the PC. Naturally, we will walk you through each setup to ensure that you understand what they are doing.

Part 3.4A Set Up EEVB Node A to Send Echo Request Messages

Node A will be the initiator of our simulated pings, sending ICMP *Echo Request* messages to Node B. We will use the node's pushbutton to control this activity, sending a message once per second while the button is held down.

► **Load 3.4 Ping Request.**

Let's take a quick look at this setup, which is pretty straightforward.

► **Switch to Messages Editor.**

On the transmit side you'll find a single message definition, which is of course the *Echo Request* we are sending. This is configured as an IPv4+ICMPv4 message. Note the two extra fields (signals) we have defined, *Identifier* and *Sequence Number*, which as we saw earlier are standard parts of *Echo Request* messages. (We decided to leave out the optional *Data* field for simplicity.) No messages are defined on the receive side, since even though Node B sends *Echo Reply* messages back to Node A, the latter node doesn't do anything with them.

► **Switch to Tx Panel.**

► **Click the Echo Request Message.**

These are nearly all the standard VSpy defaults for *Echo Request* messages. The only item changed is that we have put the value 1 into the *Identifier* field, since that's what the Windows ping utility does. The *Sequence Number* field is filled in dynamically.

► **Switch to Application Signals.**

The single application signal, *Sequence Counter*, is used to keep track of the sequence number of the last *Echo Request* sent so they are all uniquely identified, just like with the normal ping utility.

► **Switch to Function Blocks.**

The simple script here (Figure 89) is triggered when the Node A pushbutton is detected as being pressed down. If it is, the script increments *Sequence Counter*, puts its new value into the *Sequence Number* field and transmits an *Echo Request*. It then waits a full second before sending the next one (if the button is still down).

Step	Description	Value	Comment
1	If	{Switch 1 (Value) :neo0-sw0-0-index(0)}	// Use pushbutton to trigger sending Echo Request from Node A.
2	Set Value	{Sequence Counter :sig0-index(0)} = {Sequence Counter :sig0-index(0)} + 1	// Increment sequence counter.
3	Set Value	{Sequence Number (Value) :out0-sig22-0} = {Sequence Counter :sig0-index(0)}	// Set Sequence Number in Echo Request to current value of internal sequence counter.
4	Transmit	Echo Request	
5	Wait For	= 1000 ms	// Wait for a second to avoid rapid-firing Echo Requests on a single pushbutton press.
6	End If		

Figure 89: Ping Request Function Block Script.

That's it, let's download it to Node A.

► **Send CoreMini to Node A.**

Part 3.4B Set Up EEVB Node A to Respond to Echo Request Messages with Echo Reply Messages

As usually is the case, Node B will have complementary functionality to that of Node A.

► **Load 3.4 Ping Reply:** Discard changes, if prompted.

Once again we'll start by taking a look at the details of how the setup works.

► **Switch to Messages Editor.**

On the receive side we have the same *Echo Request* message definition as in the Ping Request setup transmit side; no surprise there. The transmit message is the *Echo Reply* that this node will send. Notice that the *Destination* value is zero, since that is filled in based on the *Echo Request* received.

► **Switch to Tx Panel.**

► **Click the Echo Reply Message.**

This message has the *Source* values appropriate to Node B and a *Type* that has been set to value 0 (for *Echo Reply*) rather than 8 (for *Echo Request*).

► **Switch to Function Blocks.**

This script does more and so is more complex (Figure 90). First we wait until an *Echo Request* message is seen. We check the *Type* field to make sure this is an *Echo Request* (though this is only a formality, since of course we already know what Node A is sending). When a request is seen we fill in four fields in the *Echo Reply* using values from the *Echo Request*: the *Identifier* and *Sequence Number* fields are copied directly, while the *Destination MAC Address* and *Destination IP Address* fields come from their *Source* counterparts in the *Echo Request*.

Step	Description	Value	Comment
1	Wait Until	{Echo Request (Present) :in0-0}	// Wait for an Echo Request to be received from Node A.
2	Set Value	{Echo Request (Present) :in0-0} = 0	// Clear Present flag to avoid triggering a reply more than once per request.
3	If	{Type (Value) :in0-sig18-0} = 8	// In theory this could be a different type of ICMP message; ensure it is an Echo Request before proceeding.
4	Set Value	{Identifier (Value) :out0-sig21-0} = {Identifier (Value) :in0-sig21-0}	// Copy Identifier field from Echo Request.
5	Set Value	{Sequence Number (Value) :out0-sig22-0} = {Sequence Number (Value) :in0-sig22-0}	// Copy Sequence Number from Echo Request.
6	Set Value	{Destination MAC Address (Value) :out0-sig0-0} = {Source MAC Address (Value) :in0-sig1-0}	// Set Destination MAC Address of Echo Reply to Source MAC Address of Echo Request.
7	Set Value	{Destination IP Address (Value) :out0-sig17-0} = {Source IP Address (Value) :in0-sig16-0}	// Set Destination IP Address of Echo Reply to Source IP Address of Echo Request.
8	Transmit	Echo Reply	
9	End If		
10			

Figure 90: Ping Reply Function Block Script.

This script goes to Node B.

► **Download CoreMini to Node B.**

Part 3.4C Set Up Vehicle Spy 3 on the PC to Monitor Ping Message Exchanges

Next, let's load the special setup for the PC that we'll use to watch the ping simulation.

► **Load 3.4 Ping Monitor:** Again, discard changes if prompted.

You will immediately see something new: a gray screen with a gauge that looks somewhat like a simplified car speedometer. We'll get back to this shortly. Let's start by, as usual, looking at the defined messages.

► **Switch to Messages Editor.**

Here we have definitions for the *Echo Request* and *Echo Reply* messages on the receive side. There's nothing on the transmit side since we aren't transmitting anything.

► **Switch to Application Signals.**

We have three signals defined here:

- **Ping Time:** The most recently calculated simulated ping time.
- **Update Button:** Controls whether or not the graphical panel updates.
- **Most Recent Sequence Number:** Stores the *Sequence Number* of the last *Echo Reply* message seen.

The use of these signals will be clarified in a moment.

► **Switch to Function Blocks.**

This setup's script (Figure 91) watches for *Echo Request* and *Echo Reply* messages and updates the graphical panel we saw when we first loaded this setup. First, it checks the graphical panel's update button, doing nothing unless it's on. If it is, the script waits until it sees an *Echo Request* message and the *Echo Reply* message that follows. It then calculates the

Ping Time application signal as the difference between the *Echo Reply* timestamp and that of the *Echo Request*, rounding and storing the value in milliseconds to three decimal places. Finally, it updates the *Most Request Sequence Number* field.

Step	Description	Value	Comment
1	If	{Update Button :sig2-index(0)}	// Do not update if the update button is off.
2	Wait Until	{Echo Request (Present) :in0-0}	// Wait for receipt of Echo Request sent from Node A to Node B.
3	Set Value	{Echo Request (Present) :in0-0} = 0	// Prevent multiple triggering.
4	Wait Until	{Echo Reply (Present) :in1-0}	// Wait for Echo Reply from Node B to Node A.
5	Set Value	{Echo Reply (Present) :in1-0} = 0	// Ensure the previous step isn't triggered again until another Echo Reply is seen.
6	Set Value	{Ping Time :sig1-index(0)} = int(({Echo Reply (Update Rate (abs)) :in1-3} - {Echo Request (Update Rate (abs)) :in0-3}) * 1000000 + .5) / 1000	// Set current ping time to difference between timestamps of Echo Reply and Echo Request messages.
7	Set Value	{Most Recent Sequence Number :sig3-index(0)} = {Sequence Number (Value) :in1-sig22-0}	// Update display of most recent sequence number based on value in Echo Reply.
8	End If		

Figure 91: Ping Monitor Function Block Script.

Part 3.4D Monitor Ping Message Exchanges Using Messages View

Before we go back to the graphical panel, let's look at the message exchange in the way we usually do.

- **Switch to Messages View.**
- **Filter for ICMPv4 Messages:** Enter ICMPv4 in the Protocol filter field.
- **Enter Scroll Mode.**
- **Enable Details View:** It may be turned off from an earlier lab.
- **Go Online.**

Nothing happens. Remember that Node A doesn't send *Echo Request* messages unless its pushbutton is held down.

- **Hold Down the Node A Pushbutton.**

You should see several *Echo Request* / *Echo Reply* pairs show up (Figure 92). The sets will be separated by about one second from each other, with the time between request and reply roughly 1 to 2 milliseconds.

- **Go Offline.**

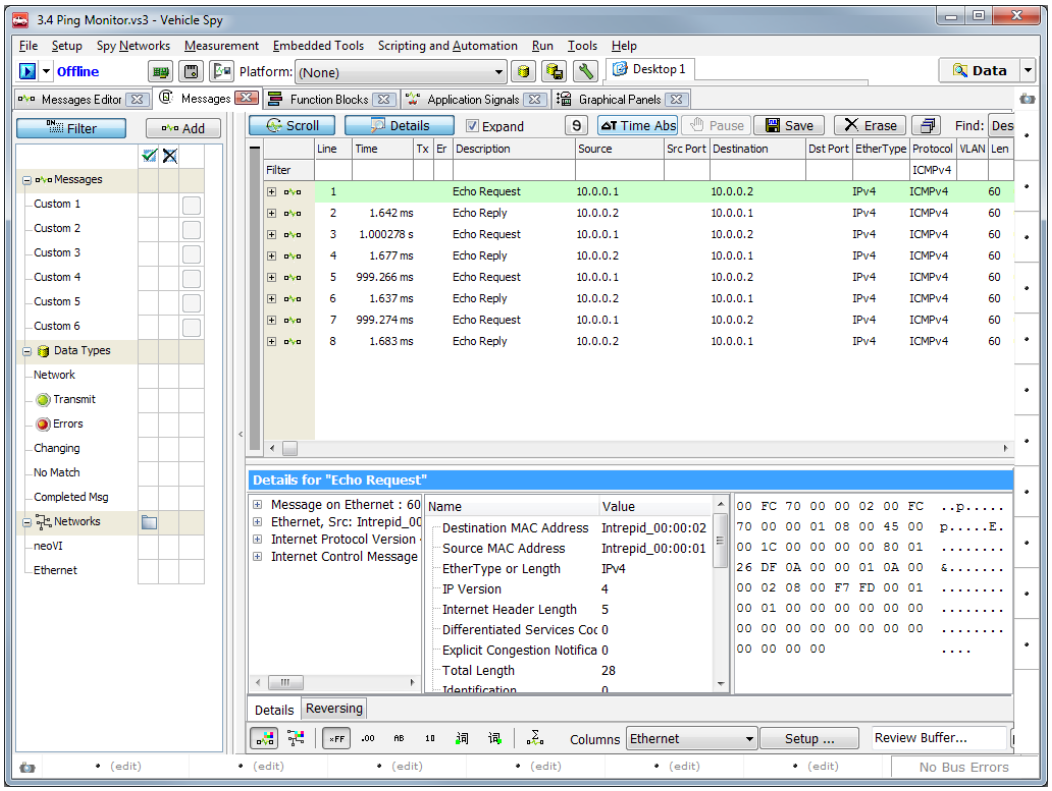


Figure 92: Ping Message Exchange. Node A sends *Echo Request* messages once per second when its pushbutton is held down; Node B sends back *Echo Reply* messages after receiving them.

Part 3.4E Monitor Ping Message Exchanges Using the Graphical Panel

Alright, let's go back to that fancy new screen now.

- **Switch to Graphical Panels:** Click the Graphical Panels tab.

Graphical panels allow you to create your own graphical interfaces to Vehicle Spy 3 scripts and functions. You can mix and match standard graphical user interface (GUI) features such as buttons and text blocks with unique items specific to Vehicle Spy 3.

In this example we've created a simple interface to allow us to monitor the activity of the EEVB as it sends ping messages back and forth. The meter in the center monitors the most recently observed ping time calculated by the function block script, presenting it in a visual fashion. Below it is a readout showing the sequence number of the last observed *Echo Reply*. And below that is the *Update* button; when it is pressed down (on) the meter and the sequence number field update; when it is up (off) they are frozen.

- **Go Online.**
- **Turn the Update Button Off (If Necessary).**
- **Hold Down the Node A Pushbutton.**

Echo Request and *Echo Reply* messages are being generated, as we saw above, but the graphical panel doesn't update continuously because *Update* is off.

- **Press the Update Button to Turn it On.**
- **Press the Node A Pushbutton:** Hold it down for a few seconds.

Now the display will update every second as new messages come in (Figure 93). The "speedometer" will show the most recent ping time both graphically and as a text value, while the box below updates with the sequence number of the last exchange.

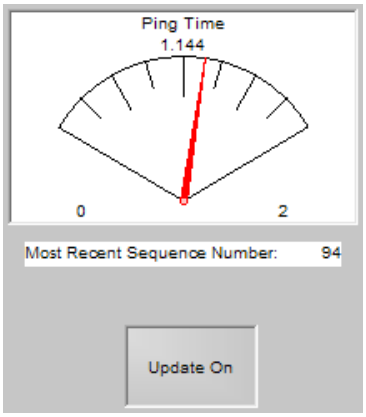


Figure 93: Graphical Panel Ping Display. When the *Update* button is on, the ping time for each *Echo Request* / *Echo Reply* exchange is shown graphically, along with the sequence number of the most recent transaction.

- **Turn the Update Button Off.**
- **Hold Down the Node A Pushbutton For Five Seconds.**
- **Turn the Update Button On.**

Notice that the display "jumps" when updating is re-enabled, showing the results of the last transaction that took place while updates were off.

As you can see, this is a more "civilized" way of looking at data and controlling how Vehicle Spy 3 works. And this is just a tiny taste of what you can do with graphical panels.

- **Go Offline.**
- **Unlock the Graphical Panel:** Click the button at the bottom right of the screen.

The panel now switches into edit mode, where you can add or remove controls or change the properties of existing ones. Along the bottom you will see icons corresponding to the various items you can put into a graphical panel.

- **Click the Ping Time Meter Control.**

After selecting this control, its properties are displayed on the right side. Notice that the *Signal* field entry is **Ping Time**, which is the application signal we defined whose value is displayed here.

- **Change Needle Color:** Double-click the red rectangle next to the NeedleColor property. Select a different color, say green.

The color of the needle on the meter changes to the color you chose; Figure 94 shows the result of the change, as well as the overall look of the graphical panel editing layout. Feel free to tinker with the many other controls available here; you'll discover that you have nearly unlimited ability to tailor the graphical panel to suit your needs.

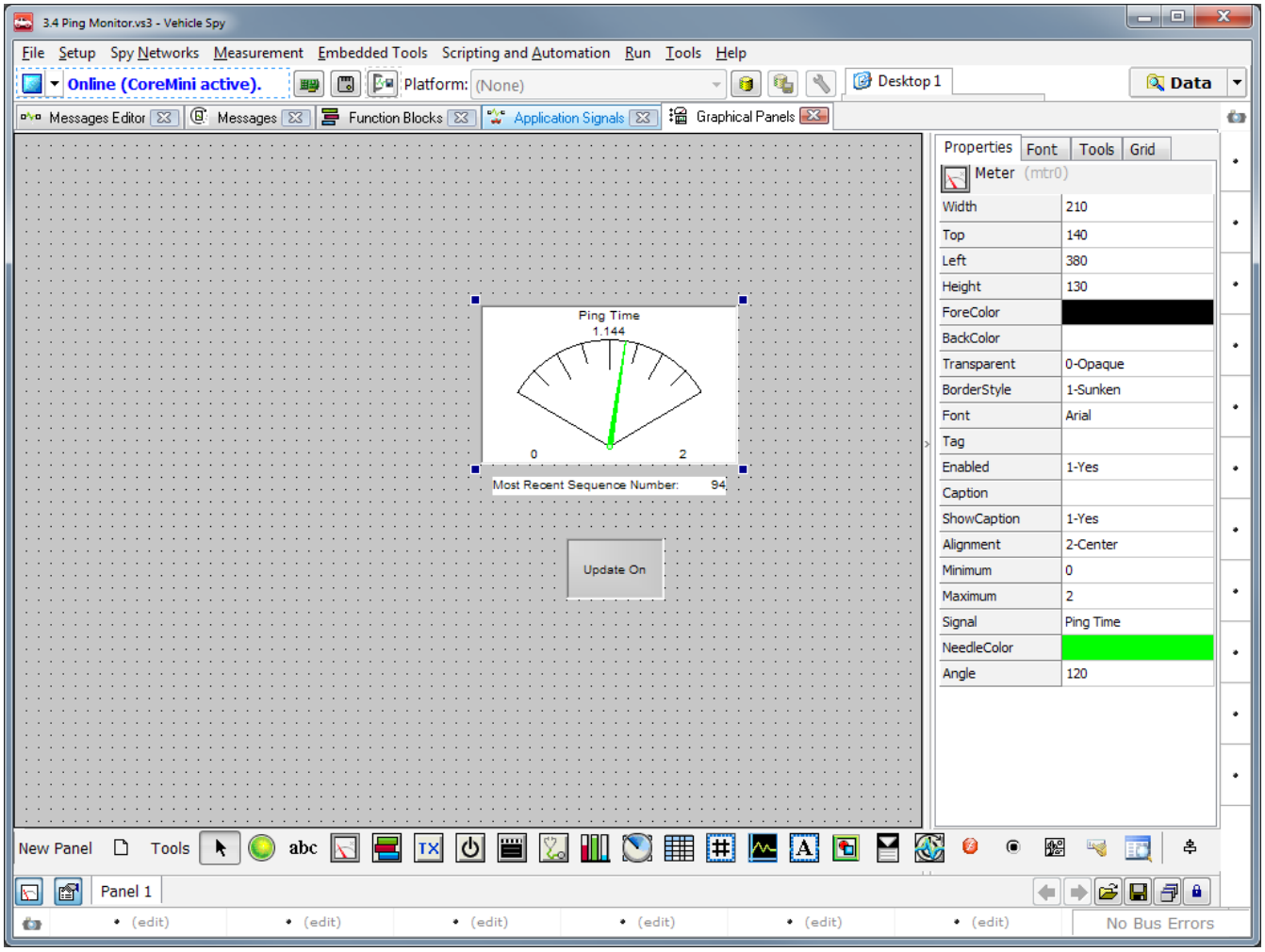


Figure 94: Editing a Graphical Panel.

Lab 3.5 Manual Ping from PC to EEVB Using RAD-Moon

In Lab 2.6 we adapted the ARP exchange we had set up in the EEVB to allow the PC to send *ARP Request* messages and have one of the EEVB nodes respond with *ARP Reply* messages. This was made possible by Intrepid's RAD-Moon media converter, which allows a standard Ethernet device such as a PC network interface card to talk directly to the EEVB BroadR-Reach nodes. We will now make the same sort of modification to our ICMP ping simulation, changing Lab 3.4 so that instead of Node A sending the *Echo Request* messages, we send them from Vehicle Spy 3 on the PC.

As was the case with Lab 2.6, this lab is optional, since it requires a RAD-Moon to be completed. You will also need a PC with a standard wired Ethernet connection for this lab to work properly. An Ethernet hub or switch will make the hardware setup easier but is not required.

This lab assumes you have continued from Lab 3.4 and so already have the EEVB nodes running the appropriate scripts for the ping simulation.

Part 3.5A Set Up the Ethernet EVB and RAD-Moon for Bidirectional Communication

You need to change the physical configuration of the PC and EEVB for this lab so that the two are linked through the RAD-Moon. Follow these steps, which should already be familiar if you did Lab 2.6.

- **Connect USB Cable to RAD-Moon.**
- **Disconnect BroadR-Reach Cable from EEVB Node A.**
- **Connect BroadR-Reach Cable to RAD-Moon.**
- **Connect Ethernet Cable to RAD-Moon and PC Jack or Hub/Switch.**

This process is described in detail in Part 2.6A; please refer there if you need step by step instructions.

Part 3.5B Load Vehicle Spy 3 Setup for Manual Ping

First, let's make sure we remember to change the Ethernet interface.

- **Change Ethernet Interface Selection:** On the *Logon Screen*, select the Ethernet interface corresponding to the PC's Ethernet port.

Assuming you have continued here after doing Lab 3.4, Node B should already be set to look for *Echo Request* messages and respond with *Echo Reply* messages, and we won't be using Node A. This means we don't need to change anything on the EEVB at all; we just need a new setup for the PC, which this time has been provided for you with a pre-made file. This is essentially a combination of the 3.4 Ping Request and 3.4 Ping Monitor setups, since we are now having VSpy perform both of those functions.

► **Load 3.5 Ping Request and Monitor.**

The graphical panel that loads will look familiar from Lab 3.4, but you can see that the *Update* button has been replaced by a *Send Ping* button. In the previous lab the *Echo Request* messages were generated by a pushbutton on the EEVB; the *Send Ping* button serves that role here.

► **Switch to Messages Editor.**

The *Echo Request* definition is on the transmit side and the *Echo Reply* message on the receive side, as you'd expect. These are the same as they were in earlier labs.

► **Switch to Application Signals.**

The *Ping Time* and *Most Recent Sequence Number* application signals are the same as the ones we used in 3.4 Ping Monitor. *Sequence Counter* is used to generate sequentially-numbered *Echo Request* messages as in 3.4 Ping Request.

► **Switch to Function Blocks.**

This script (Figure 95) is also a combination of ones we've seen before. First, it increments the *Sequence Counter* variable and then sends an *Echo Request* message with the new value in its *Sequence Number* field. It waits for an *Echo Reply* message, calculates the *Ping Time* signal as the difference between the *Echo Reply* and *Echo Request* timestamps, and updates the *Most Request Sequence Number* field.

Step	Description	Value	Comment
1	Set Value	{Sequence Counter :sig4-index(0)} = {Sequence Counter :sig4-index(0)} + 1	// Increment sequence counter.
2	Set Value	{Sequence Number (Value) :out0-sig22-0} = {Sequence Counter :sig4-index(0)}	// Set Sequence Number in Echo Request to current value of internal sequence counter.
3	Transmit	Echo Request	// Send Echo Request.
4	Wait Until	{Echo Reply (Present) :in1-0}	// Wait for Echo Reply from Node B to Node A.
5	Set Value	{Echo Reply (Present) :in1-0} = 0	// Ensure the previous step isn't triggered again until another Echo Reply is seen.
6	Set Value	{Ping Time :sig1-index(0)} = int(({Echo Reply (Update Rate (abs)) :in1-3} - {Echo Request (Update Rate (abs)) :out0-3}) * 1000000 + .5) / 1000	// Set current ping time to difference between timestamp of Echo Reply and Echo Request messages.
7	Set Value	{Most Recent Sequence Number :sig3-index(0)} = {Sequence Number (Value) :in1-sig22-0}	// Update display of most recent sequence number based on value in Echo Reply.
8	Stop	n/a	
9			
10			

Figure 95: Ping Request and Monitor Function Block Script.

One noteworthy difference here is that this script's start type is **Manual**. This is because we only want this script to run when we tell it to using the *Send Ping* button on the graphical panel.

Part 3.5C Generate and Monitor Ping Message Exchanges Using the Graphical Panel

Before we go back to our graphical panel, let's set up *Messages View* to show only the messages we want.

► **Switch to Messages View.**

- **Set Custom Filter:** Enter Echo in the Description filter field so only our special messages are shown.
- **Enter Scroll Mode.**

Okay, let's go back to our graphical panel now.

- **Switch to Graphical Panels.**
- **Go Online.**
- **Press the Send Ping Button.**

You should see the "needle" on the *Ping Time* meter display a time somewhere in the range of 1 to 3 milliseconds, though it may be less. The *Most Recent Sequence Number* displayed will be 1 (Figure 96).

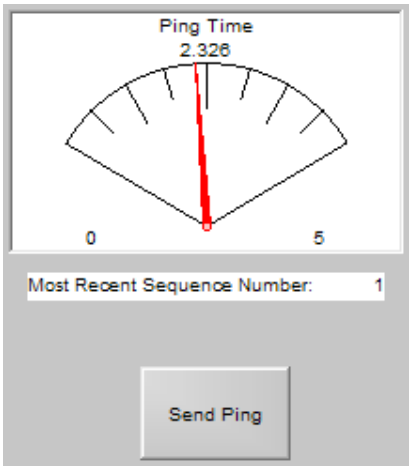


Figure 96: Graphical Panel Manual Ping Display.

In fact, if you go offline and then online, the sequence number shown will again be 1 after the *Send Ping* button is pressed. This is different from the way it worked when we were sending from EEVB Node A, because Vehicle Spy 3 application signals are cleared each time you go online, while those in EEVB CoreMinis are only reset if the board is power-cycled.

- **Switch to Messages View.**

You will now see the *Echo Request* and *Echo Reply* message exchange that was summarized in the graphical panel display (Figure 97). The elapsed time between the messages should match that shown in the graphical panel, as is the case in our figures.

	Line	Time	Tx	Er	Description	Source	Src Port	Destination	Dst Port	EtherType	Protocol	VLAN	Len
Filter					Echo								
	1				Echo Request	10.0.0.1		10.0.0.2		IPv4	ICMPv4		42
	2	2.326 ms			Echo Reply	10.0.0.2		10.0.0.1		IPv4	ICMPv4		60

Figure 97: Manual Ping Echo Request / Echo Reply Exchange.

- **Switch to Graphical Panels.**
- **Press the Send Ping Button Again.**

A new ping value is displayed and the sequence number increments. Notice that the ping times are generally longer here than they were in the previous lab. This makes sense, because it takes more time for an *Echo Request* to travel from the PC through the RAD-Moon and to EEVB Node B than it takes for a message to move between Node A and Node B of the board.

Part 3.5D Restore EEVB-Only Configuration

We're done with this lab, so let's restore our normal hardware configuration.

- **Go Offline.**
- **Disconnect USB Cable from RAD-Moon.**
- **Disconnect BroadR-Reach Cable from RAD-Moon.**
- **Connect BroadR-Reach Cable to EEVB Node A.**
- **Disconnect Ethernet Cable from RAD-Moon and PC or Hub.**
- **Reconnect Ethernet Cable to PC Ethernet Port:** If you disconnected your main network connection for this lab, be sure to restore it.
- **Select Ethernet EVB in Ethernet Interfaces List.**

Lab 3.6 Simulating a Routing Problem with ICMP Time Exceeded Messages

In the introduction to ICMP at the start of this section of the Lab Manual, we mentioned that there were two classes of messages defined within the protocol. The *Echo Request* and *Echo Reply* messages we've been using over the last few labs are examples of ICMP *informational messages*, which were created specifically for network tools such as ping. The others are ICMP *error messages*. We've seen that these can also be employed in a utility role, when we examined how they are used to implement traceroute. However, their primary function, as their name implies, is to allow the communication of error conditions among nodes.

In this lab, we will simulate an example of a network with a problem condition that causes ICMP *Time Exceeded* messages to be generated. Specifically, we'll be using the same subtype of this message used in traceroute—messages created when the *Time To Live* field of a message drops to 0 while it is in the process of being routed.

Recall that routers are devices that connect together different networks into internetworks, passing messages intended for distant hosts from one network to the next until they reach their intended destinations. Most computer users are familiar with the small routers used to move traffic over a home or office Internet connection; there are also much larger ones that route millions of packets per second to implement the Internet itself.

To demonstrate ICMP error messages we will set up a mock routed network consisting of three devices: EEVB Node A, EEVB Node B and a PC running Vehicle Spy 3. Node A will regularly generate IPv4 messages that are ultimately intended for the PC. Node B will act as the "router", taking the datagrams that Node A sends, repackaging their data, and sending a new datagram containing the data to the PC. We'll use the pushbutton on Node A to select between two different *Time To Live* values in the transmitted messages, allowing us to alter the behavior of the "router" dynamically.

Bear in mind that no actual routing is taking place here, because all of these devices are on the same local network, meaning that the PC can see Node A's messages without the need for Node B to "route" anything. However, as we have done in other cases, we're going to pretend that routing is taking place for the sake of illustration.

Part 3.6A Set Up EEVB Node A to Send IPv4 Messages with Variable Time To Live Values

Naturally, we will start with Node A, setting it up to send IPv4 data packets. We've created a setup file for you, which we'll briefly examine before downloading it to the EEVB.

- ► **Load 3.6 Message Transmit.**
- ► **Switch to Messages Editor.**

This node only sends data, so there are no receive messages defined. On the transmit side there is an IPv4 message defined called *Lab 3.6 Data Packet*; this is a "raw" IPv4 message because it doesn't contain an encapsulated higher-layer protocol message such as one from

UDP or TCP. Instead we have two custom fields: *Lab 3.6 Packet Number*, which is simply a sequential counter to allow us to distinguish among adjacent packets, and *Lab 3.6 Data Field*, which can carry an arbitrary 16-bit data value.

► **Switch to Application Signals.**

The application signal *Packet Counter* is the means by which the function block script keeps track of what number to put in the next message it sends.

► **Switch to Function Blocks.**

This script (Figure 98) first fills in *Lab 3.6 Data Field*; we could have chosen anything to put here, but we decided just to use the value of the Node A potentiometer, which you may recall reads as a value of 0 to 4,095. You can thus alter what data is sent in each packet by changing the position of the potentiometer dial, and the variations in sampling the potentiometer mean that the data value will also fluctuate over time naturally. The script then increments the *Packet Counter* application signal and stores that value in the *Packet Number* message field. The script tests the value of the Node A pushbutton; if it's held down, the *Time To Live* value of the IPv4 message is set to 1, and otherwise it is made 2. The packet is then transmitted, and a delay of 5 seconds imposed.

Step	Description	Value	Comment
1	Set Value	{Lab 3.6 Data Field (Value) :out0-sig22-0} = {Analog Input 1 (Value) :neo0-ai0-0-index(0)}	// Set Data field to be equal to potentiometer input (value from 0 to 4095).
2	Set Value	{Packet Counter :sig0-index(0)} = {Packet Counter :sig0-index(0)} + 1	// Increment packet counter.
3	Set Value	{Lab 3.6 Packet Number (Value) :out0-sig21-0} = {Packet Counter :sig0-index(0)}	// Set IPv4 message packet number to current counter value so packets are uniquely numbered.
4	If	{Switch 1 (Value) :neo0-sw0-0-index(0)}	// If switch is pressed down, set TTL to 1, otherwise 2.
5	Set Value	{Time To Live (Value) :out0-sig13-0} = 1	
6	Else		
7	Set Value	{Time To Live (Value) :out0-sig13-0} = 2	
8	End If		
9	Transmit	Lab 3.6 Data Packet	
10	Wait For	= 5000 ms	

Figure 98: Message Transmit Function Block Steps.

► **Download CoreMini to Node A.**

Part 3.6B Set Up EEVB Node B as "Router" of IPv4 Messages Coming from Node A

Now let's take a look at the setup and program logic for our simulated "router" that will run on EEVB Node B.

► **Load 3.6 Message Routing.**

► **Switch to Messages Editor.**

There is one receive message definition, for *Lab 3.6 Data Packet*, the message sent regularly by Node A. There are two transmit messages. The first, *Lab 3.6 Data Packet Routed*, has basically the same definition as *Lab 3.6 Data Packet* but with different field names; this is the message that Node B sends when it successfully "routes" a packet from Node A to the PC. The

Lab 3.6 ICMP Time Exceeded message is sent if this fails due to the *Time To Live* field in *Lab 3.6 Data Packet* hitting zero.

► **Switch to Function Blocks.**

The script (Figure 99) is triggered by receipt of a *Lab 3.6 Data Packet* message. When one arrives, we check its *Time To Live* field. If it's 1, then this "routing" operation would reduce it to 0, so the message has expired; we send a *Lab 3.6 ICMP Time Exceeded* message back to Node A. In this message we include the number of the packet that was dropped. (Note that in real ICMP the first portion of the full IPv4 message would be included in this message, but we've skipped this step to reduce complexity.)

If the *Time To Live* field is 2 or greater, we are going to "route" the message. We copy *Lab 3.6 Packet Number* and *Lab 3.6 Data Field* from the received message to their corresponding areas in the new message. We also copy the *Time To Live* field, decreasing it by one. Then we send the *Lab 3.6 Data Packet Routed* message.

Step	Description	Value	Comment
1	Wait Until	{Lab 3.6 Data Packet (Present) :in2-0}	// Wait for incoming IPv4 data packet from Node A.
2	Set Value	{Lab 3.6 Data Packet (Present) :in2-0} = 0	// Clear flag to avoid extra triggering.
3	If	{Time To Live (Value) :in2-sig13-0} = 1	// If TTL field is 1, decrementing it would make it 0, so drop packet and send Time Exceeded message.
4	Set Value	{Lab 3.6 Expired Packet Number (Value) :out2-sig21-0} = {Lab 3.6 Packet Number (Value) :in2-sig21-0}	// Specify the number of the packet that expired.
5	Transmit	Lab 3.6 ICMP Time Exceeded	// Send Time Exceeded message.
6	Else		// Otherwise, "forward" the message.
7	Set Value	{Lab 3.6 Packet Number Routed (Value) :out1-sig19-0} = {Lab 3.6 Packet Number (Value) :in2-sig21-0}	// Copy packet number to routed message.
8	Set Value	{Lab 3.6 Data Field Routed (Value) :out1-sig20-0} = {Lab 3.6 Data Field (Value) :in2-sig22-0}	// Copy data field.
9	Set Value	{Time To Live (Value) :out1-sig13-0} = {Time To Live (Value) :in2-sig13-0} - 1	// Set Time To Live to 1 less than that of the incoming message.
10	Transmit	Lab 3.6 Data Packet Routed	
11	End If		

Figure 99: Message Routing Function Block Steps.

► **Send CoreMini to Node B.**

Part 3.6C Set Up Vehicle Spy 3 on the PC to Monitor Message Traffic

We'll now load a simple setup that has all of the message definitions described above, so we can watch the traffic sent by the EEVB nodes. Again, all devices are on the same network, enabling us to "spy" on traffic that Node A is sending to Node B and vice-versa, as we always do. In a true routing situation, Node A and the PC would be on different networks and this would not be possible, but again, this is just a simulation.

► **Load Lab 3.6 Message Monitor.**

► **Switch to Messages View.**

► **Set Filter for Lab 3.6 Messages:** Enter Lab 3.6 in the Description column filter box.

► **Enter Scroll Mode (If Necessary).**

► **Go Online.**

Every five seconds you should see a *Lab 3.6 Data Packet* message going from 10.0.0.1 to 10.0.0.2, followed immediately by a *Lab 3.6 Data Packet Routed* message sent from 10.0.0.2 to 10.0.0.3 (Figure 100). This represents the normal “routing” of our message by Node B.

	Line	Time	Tx	Er	Description	Source	Src Port	Destination	Dst Port	EtherType	Protocol	VLAN	Len
Filter					Lab 3.6								
⊞	1				Lab 3.6 Data Packet	10.0.0.1		10.0.0.2		IPv4			60
⊞	2	1.299 ms			Lab 3.6 Data Packet Routed	10.0.0.2		10.0.0.3		IPv4			60
⊞	3	5.000344 s			Lab 3.6 Data Packet	10.0.0.1		10.0.0.2		IPv4			60
⊞	4	1.331 ms			Lab 3.6 Data Packet Routed	10.0.0.2		10.0.0.3		IPv4			60
⊞	5	5.000320 s			Lab 3.6 Data Packet	10.0.0.1		10.0.0.2		IPv4			60
⊞	6	1.308 ms			Lab 3.6 Data Packet Routed	10.0.0.2		10.0.0.3		IPv4			60

Figure 100: Data Packet Transmission and “Routed” Retransmission.

► **Switch to Signal List.**

We’ve created a signal list to more easily allow us to examine some of the more important fields (signals) in our messages (Figure 101). The first three are from *Lab 3.6 Data Packet*, and the last three from *Lab 3.6 Data Packet Routed*.

Signal	Value	Update
Time To Live (Value)	2	/
Lab 3.6 Data Field (Value)	2352	/
Lab 3.6 Packet Number (Value)	9	/
Time To Live (Value)	1	/
Lab 3.6 Data Field Routed (Value)	2352	/
Lab 3.6 Packet Number Routed (Value)	9	/

Figure 101: Selected Fields Data Packet and “Routed” Data Packet.

► **Slowly Rotate the Node A Potentiometer.**

You will see the *Lab 3.6 Data Field* value change as you move the potentiometer, and the *Lab 3.6 Data Field Routed* value will change to match it.

Notice also that the *Lab 3.6 Packet Number* and *Lab 3.6 Packet Number Routed* fields increase by 1 with each message exchange. Also, the *Time To Live* value for *Lab 3.6 Data Packet* (the first signal in our list) is 2, while it has been reduced to 1 for *Lab 3.6 Data Packet Routed* (the fourth signal). Again, these are all as we expected.

► **Switch to Messages View.**

► **Hold Down the Node A Pushbutton.**

You will see that each *Lab 3.6 Data Packet* message is now being followed by a *Lab 3.6 ICMP Time Exceeded* message (Figure 102). This continues as long as the pushbutton is held down.

	Line	Time	Tx	Er	Description	Source	Src Port	Destination	Dst Port	EtherType	Protocol	VLAN	Len
Filter					Lab 3.6								
⊞	81	4.999650 s			Lab 3.6 Data Packet	10.0.0.1		10.0.0.2		IPv4			60
⊞	82	2.033 ms			Lab 3.6 Data Packet Routed	10.0.0.2		10.0.0.3		IPv4			60
⊞	83	4.999636 s			Lab 3.6 Data Packet	10.0.0.1		10.0.0.2		IPv4			60
⊞	84	2.020 ms			Lab 3.6 Data Packet Routed	10.0.0.2		10.0.0.3		IPv4			60
⊞	85	4.999599 s			Lab 3.6 Data Packet	10.0.0.1		10.0.0.2		IPv4			60
⊞	86	1.050 ms			Lab 3.6 Data Packet Routed	10.0.0.2		10.0.0.3		IPv4			60
⊞	87	5.000602 s			Lab 3.6 Data Packet	10.0.0.1		10.0.0.2		IPv4			60
⊞	88	1.048 ms			Lab 3.6 ICMP Time Exceeded	10.0.0.2		10.0.0.1		IPv4	ICMPv4		60
⊞	89	5.000605 s			Lab 3.6 Data Packet	10.0.0.1		10.0.0.2		IPv4			60
⊞	90	1.058 ms			Lab 3.6 ICMP Time Exceeded	10.0.0.2		10.0.0.1		IPv4	ICMPv4		60
⊞	91	5.000579 s			Lab 3.6 Data Packet	10.0.0.1		10.0.0.2		IPv4			60
⊞	92	1.085 ms			Lab 3.6 ICMP Time Exceeded	10.0.0.2		10.0.0.1		IPv4	ICMPv4		60

Figure 102: Data Packet and Simulated ICMP Time Exceeded Messages. When the Node A pushbutton is held down, the *Time To Live* field value in *Lab 3.6 Data Packet* changes from 2 to 1. When Node B sees this, instead of "routing" the message to the PC, it sends back a *Lab 3.6 ICMP Time Exceeded* message to Node A.

► **Switch to Signal List.**

Notice that the *Time To Live* value at the top (the one in the original *Lab 3.6 Data Packet*) is now 1 rather than 2. (The other *Time To Live* value stays at 1 rather than 0 because it is showing the previous value from the last *Lab 3.6 Data Packet* that was received before the pushbutton was held down.)

► **Release Node A Pushbutton.**

The *Lab 3.6 Data Packet Routed* messages resume being shown.

► **Go Offline.**

Congratulations, you've completed Section 3 of the Intrepid Ethernet EVB Lab Manual!

Section 4 TCP/IP User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) Data Exchanges

In this, the final section of the EEVB Lab Manual, we move up again in the TCP/IP protocol stack to the Transport Layer, also known as layer 4 in the OSI Reference Model. The two primary protocols found here are the *User Datagram Protocol* (*UDP*) and *Transmission Control Protocol* (*TCP*). UDP and TCP work in conjunction with the Internet Protocol to carry nearly all Internet traffic.

In this section you'll learn about these two "siblings", analyze their behavior within Vehicle Spy, and work with simulations using the protocols in the Ethernet EVB and Vehicle Spy 3. You'll achieve these objectives:

- Learn about UDP and TCP and how they work, how they differ, and why both are important.
- Learn about ports and how their software-level addressing permits multiple processes to function on a device.
- Capture and analyze UDP and TCP messages.
- Generate UDP messages and a TCP connection message exchange.
- Examine and run a UDP-based version of the Ethernet EVB Input Output example first encountered in the EEVB User's Guide.
- See an example of a custom application protocol based on UDP.
- Work with a complex simulation of the TCP connection establishment and termination process.

As we have in the last few sections, we will start with a brief summary of the TCP and UDP protocols for those who may not be familiar with them. If you want a more complete description of these technologies, they are covered in Part IV of Intrepid's book *Automotive Ethernet - The Definitive Guide*, which is included in your Ethernet EVB package.

Why Two Transport Layer Protocols?

The Internet Protocol is the basis for the Internet and other TCP/IP networks, but it is not sufficient by itself to implement an internetwork. IP is an unreliable, unacknowledged and unmanaged protocol, so it doesn't provide a way for a device sending data to ensure that it was received, to detect errors in transmissions, or control the rate at which data is sent. Without these capabilities built in at the lower levels, thousands of applications would be forced to implement them, which would be tremendously inefficient.

It makes sense to implement these services at the Transport Layer so higher layers can make use of them. However, not all applications need these functions, and in some cases using them would actually be detrimental, due to the additional complexity and network bandwidth required

to provide them. For this reason, TCP/IP uses two different core Layer 4 protocols: TCP, which implements these services for applications that require them, and UDP, which does not.

By means of analogy, TCP is a fully-loaded luxury performance sedan with a chauffeur, roadside service and GPS. It provides lots of frills and comfort, good performance, and virtually guarantees you will get where you need to go without any problems. In contrast, UDP is a stripped-down race car; the only goal is speed, and while you probably will get to your destination quickly, you might also have a breakdown in the middle of the road somewhere.

An Overview of UDP

Our race car, UDP only really performs two complementary tasks. When transmitting, it takes data from higher-layer protocols, prefixes a header to that data, and sends it the IP layer. When receiving, it takes data passed to it by IP, performs a simple error check on the UDP header data, then removes that header and sends the data to the application. As mentioned above, UDP provides no service guarantees; it simply packages the data and sends it.

The header format for UDP, which can be found in Figure 103, has only four fields. The *Source Port* and *Destination Port* fields are used for process-level addressing, which we'll discuss in a moment. The *Length* field specifies how many bytes are in the message, and *Checksum* provides limited error-detection capabilities for the UDP header and also some of the IP header fields as well. It does not allow errors in the actual data to be detected.

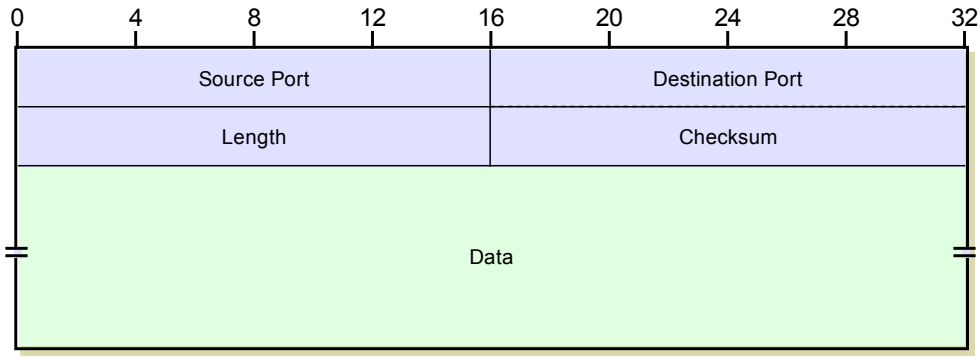


Figure 103: UDP Header Format.

UDP is usually used in applications where some or all of the following are true:

- Loss of a message won't unduly affect operation of the protocol.
- Receiving a retransmission of a lost message wouldn't be useful (usually because the messages are time-sensitive).
- Simplicity is essential.
- Speed of transmission is of paramount importance.

UDP is also used whenever multicasting is required, because TCP only supports unicast communications between two devices.

An Overview of TCP

TCP is a full-featured Transport Layer protocol that provides all the functions needed by a typical application for the reliable transportation of data across an arbitrary internetwork. It is used by most high-level protocols so that they can focus on the application they are implementing, without having to worry about being sure that the data they send will be correctly received.

TCP is a complex protocol, so we can only describe it in broad strokes. Here is a brief overview of its main operational characteristics.

Connection-Based Operation

TCP is a connection-oriented protocol, which means that all communication takes place in the context of a logical connection between a pair of device. There are three basic phases:

- **Connection Establishment:** A device that wants to communicate contacts another device using a special message type. The two establish a logical connection by exchanging configuration messages.
- **Connection Use:** The connection can be used to send data bidirectionally for an arbitrary amount of time (milliseconds to hours, or even longer in some cases).
- **Connection Termination:** When either side no longer wants to talk to the other, another exchange of special messages occurs to tear down the connection while ensuring that no pending data transmissions are lost.

TCP is designed so that multiple connections can be supported simultaneously by any device.

We will see an example of a real TCP connection in Lab 4.1, and work with a simulation of the TCP connection establishment and termination processes in Lab 4.4.

Byte-Oriented Data Transfer

TCP/IP protocols like IP and UDP are designed to send data in discrete chunks, which we call messages or packets. An application protocol using one of these will generally send the data in blocks, so they can be packaged and transmitted whole.

TCP is unique because it is said to be *stream-oriented* rather than message-oriented. This means that an application protocol using TCP can simply send data as a stream of bytes, without having to break it into discrete pieces. TCP takes care of managing the bytestream, packaging and then transmitting (and if necessary retransmitting) each byte in messages based on what is most efficient for the needs of the connection at any given time.

TCP data is sent in messages that are commonly called *segments*. The TCP header (Figure 104) is much longer than the UDP header because of all of the additional services that TCP provides. We will discuss many of these fields when we look at TCP messages in Lab 4.1.

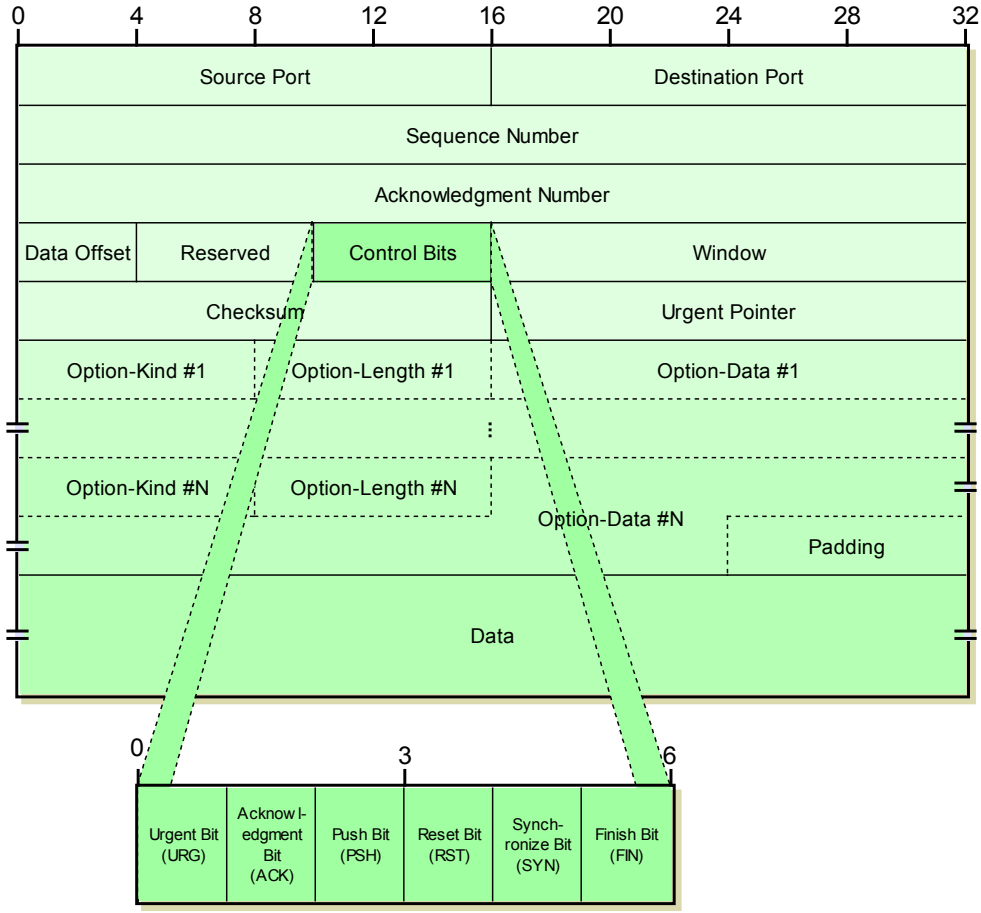


Figure 104: TCP Header Format.

Reliable Operation and Flow Control

TCP includes an extensive set of mechanisms to ensure that data gets from source to destination reliably and consistently. The key to its operation in this regard is a special *sliding window acknowledgment system*, which allows each device to keep track of which bytes of data have been sent and to confirm receipt of data received from its partner in the connection. Unacknowledged data is eventually retransmitted automatically, and the parameters of the system can be adjusted to suit the needs of both the devices and the connection.

This same system also provides buffering and flow control capabilities between devices, to handle uneven data delivery rates, bottlenecks and other problems.

Process-Level Addressing: TCP and UDP Ports

We've already seen how IP addresses are used to uniquely identify devices on an internetwork (including the Internet). In modern networks, though, devices are complex: they are usually using many different applications and protocols. For example, your home computer may be running a web browser, an email client, a chat client and many other types of network software. Even considering just your web browser, it probably has multiple tabs open with data being sent to and received from many web servers simultaneously.

We need an additional level of addressing to keep all of these different communications separate from each other, while allowing all of them to share network links. This is accomplished in TCP/IP at the Transport Layer using TCP and UDP *ports*, which are 16-bit numbers that uniquely identify a particular software process on a device. If an IP address can be likened to a street address, then a port number is sort of like the name of a person who resides at that address. Much as the name allows letters and packages sent to a home or business to ultimately be directed to the correct actual recipient, the port number does the same with UDP and TCP messages sent to an IP address.

Another important function of ports is that they solve the problem of how a client knows which software process on a server should receive its request. A central authority maintains a database of *well-known ports* for common protocols; for example, the Hypertext Transfer Protocol (HTTP) used on the World Wide Web has a well-known port number of 80. When a client wants to send a request to a web server, it transmits it to that server's IP address with a port number of 80. The client generates a random port number (from outside the reserved range) and the server replies back to it as shown in Figure 105.

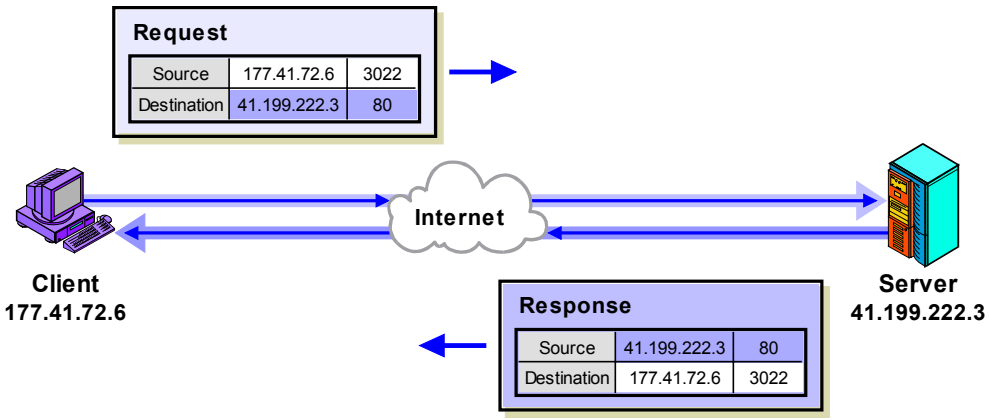


Figure 105: Client/Server Communication Using Port Numbers.

The combination of an IP address and a port number is called a *socket*. A pair of sockets, as shown in the two hypothetical messages in Figure 105, uniquely identifies a particular connection or exchange of data between two devices.

Lab 4.1 Analyzing UDP and TCP Messages and Exploring the TCP Column Display

We'll continue the pattern established in earlier sections of the Lab Manual, starting out our look at UDP and TCP with real traffic captured on a live Ethernet connection. We will explore how to use filters to narrow down the large number of messages that can arrive using these two protocols, and discuss the fields found in both message types. We'll also generate our own UDP and TCP traffic using two common application-level protocols: the *Domain Name System* (*DNS*) and *Hypertext Transfer Protocol* (*HTTP*).

In earlier labs where we've worked with live (non-simulated) TCP/IP data we have mentioned the need for a real Ethernet network in order to see traffic, and recommended an Internet connection. Here the Internet link is actually mandatory to duplicate our results, since this lab works with real web servers and IP addresses.

Part 4.1A Go Online and Work with Filtering UDP and TCP Messages

As we often do at the start of a new lab manual section, we'll restart Vehicle Spy 3 to ensure that any settings from previous labs have been cleared.

- ► **Close Vehicle Spy 3.**
- ► **Start Vehicle Spy 3.**

Let's now select the Ethernet interface through which we connect to the Internet, and also load our standard custom column setup so that *Messages View* is set to show Ethernet traffic.

- ► **Select Internet Connection Ethernet Interface:** Choose the Ethernet interface through which you connect to the Internet. This will normally be the (non-EEVB) interface with the highest packet rate.
- ► **Load 1.1 Custom Column Setup.**

Just as was the case with IP, finding UDP and TCP traffic is far from difficult: the vast majority of messages on a conventional TCP/IP network use one or the other. Thus, we will see some immediately as soon as we go online.

- ► **Enter Scroll Mode (If Necessary).**
- ► **Go Online.**

Also as was the case with IP, not only is it not difficult to find UDP/TCP messages, viewing them is somewhat like trying to drink from a firehose. One option is to collect data for a while and then go offline, but usually we will instead want to apply column filters; this will "sift" the data and allow us to focus more finely on what we are interested in at a particular time.

To begin with, we can set a filter that will exclude any messages that are not UDP or TCP.

- ► **Set UDP and TCP Protocol Filter:** In the Protocol column filter box, enter UDP,TCP.

Assuming that you are on an active Internet connection, your VSpy setup is still probably scrolling madly, because messages using neither UDP nor TCP typically account for less than 5% of Internet traffic. Thus the “firehose” persists; we will need to narrow down further.

Let’s go offline so we can see more easily the impact of future filter additions.

► **Go Offline.**

Before proceeding, let’s quickly look at the *EtherType* column in VSpy. Even though we didn’t enter a filter here asking to show only IP messages, that’s what you will almost certainly see in every line. Recall that in Section 3 we mentioned that IP messages rarely appear by themselves; they nearly always encapsulate a higher-layer protocol. Most of the time this is either UDP or TCP. Thus, when you filter for UDP and TCP in the *Protocol* column, you will be inherently selecting most of the IP traffic as well. This will be a mix of versions (IPv4 and IPv6) but you can filter to show just one or the other, as we did in Lab 3.1.

Back to filtering. Obviously the first thing we can do is show only UDP *or* TCP; let’s choose TCP in this case.

► **Set Protocol Filter to TCP:** Change the Protocol column filter to just TCP. You can also select TCP from the drop-down box in the filter row for that column.

UDP messages are now suppressed, which will remove some portion of the total messages being shown. However, you will probably still have quite a lot showing.

One common way of narrowing down what you are looking for is to use IP address filtering. For example, many networks use the IP address range 192.168.x.x for local devices, including the network we used in creating this manual. We can use this to quickly isolate all of the messages either originating from, or coming into, the local network.

► **Add Source Filter:** Add the filter 192.168 to the Source column in the filter row.

Since we put this filter in the Source column, we will see requests coming from our local network. It still leaves a lot of messages, but it’s more manageable than before.

Many different applications “share” UDP and TCP to send their data, so a useful way of narrowing traffic down further is to focus on a particular application layer protocol. For example, suppose we want to see all messages sent from our local network using HTTPS, the secure version of the Hypertext Transfer Protocol often used for accessing online banking and other sensitive transactions online. These requests normally use TCP and are sent to port 443 on the destination server.

► **Add Dst Port Filter:** Add a Dst Port filter of 443.

We will now only see HTTPS requests from our local address (Figure 106). Naturally, we could narrow things down even further by specifying the IP address of the remote server we are interested in, a feature we’ll explore later in this lab.

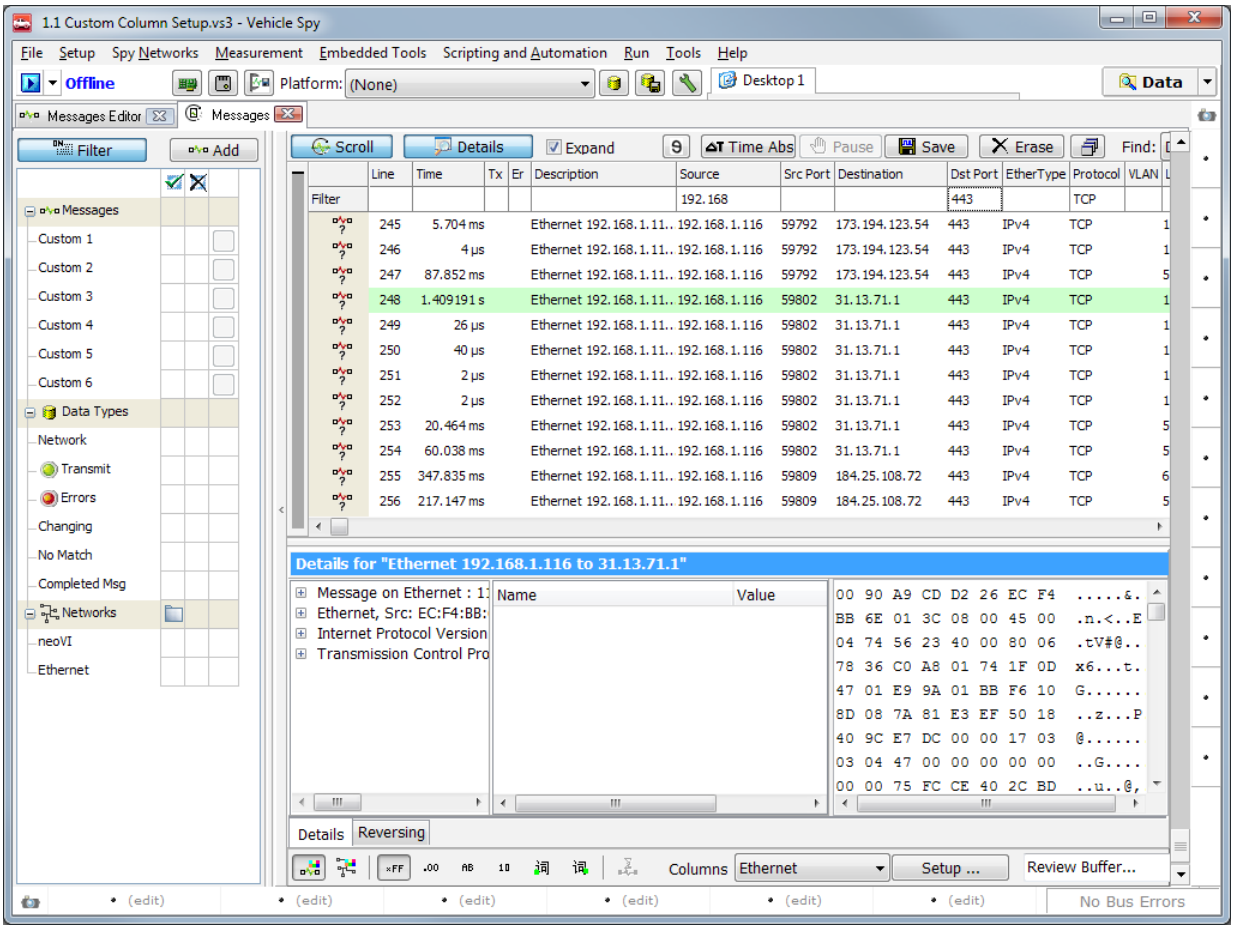


Figure 106: Filtering for Local HTTPS Requests in Messages View. In this example we have specified a *Source* filter value of 192.168 and a *Dst Port* value of 443, which means we will see only HTTPS requests from the local network. As it happens, only one device is active on the network, with an IP address of 192.168.1.116. It has made a number of requests to several IP addresses, such as 31.13.71.1 (which belongs to Facebook.)

Part 4.1B Analyze a UDP Message

We're going to look at both UDP and TCP messages in detail in this lab. Since UDP messages are (much) smaller and simpler than TCP ones, it makes sense to start with them.

You should have a good number of UDP messages already captured in VSpy from the previous part of the lab, though the filters we entered have caused them to all be hidden. Let's fix that now.

- **Set IPv4 EtherType Filter:** Enter IPv4 as an EtherType column filter.
- **Set UDP Protocol Filter:** Change the Protocol column filter to UDP.
- **Clear Other Filters:** Erase the filters previously entered in other columns.

We want *Details View* for this part of the lab, so make sure it is turned on, and then select a UDP message.

► **Enable Details View (If Necessary).**

► **Select UDP Message:** Any one will do.

You should see four lines in the information pane on the left side of Details View, the first three of which will be *Message on Ethernet* (the general entry VSpy uses for Automotive Ethernet messages) followed by *Ethernet* and *Internet Protocol Version 4*, which are the lower-level encapsulations of our UDP message. The fourth line will say *User Datagram Protocol* and will list the source and destination ports used by the particular message you selected.

► **Select the User Datagram Protocol Header.**

You will see 8 bytes highlighted in the byte display on the right, corresponding to the bytes in the UDP header.

► **Expand the UDP Message Header:** Click the ⊞ button to the left of User Datagram Protocol in the *Details View*.

Your *Details View* window should now look something like what is shown in Figure 107.

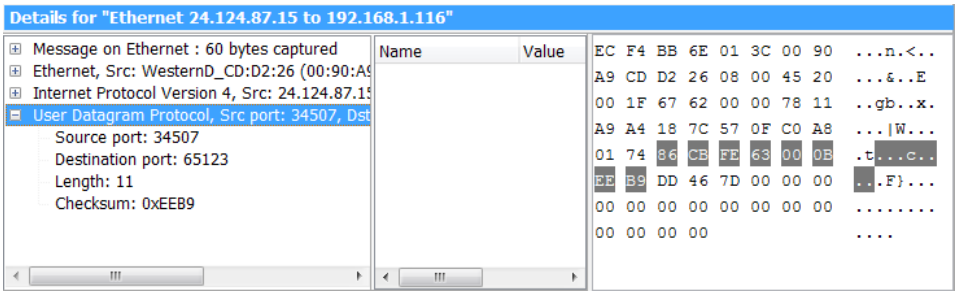


Figure 107: UDP Message in Details View.

In the introduction to this section we described UDP as something of a “stripped down” protocol, and you can see exactly how “lean and mean” it is by how few fields are present in the header:

- **Source Port:** Identifies the process that created the UDP message on the originating device.
- **Destination Port:** Specifies the process intended to receive the message on the destination device.
- **Length:** The length of the complete UDP message, including the 8-byte header and the data that follows.
- **Checksum:** A 16-bit checksum calculated based on the UDP header and also some of the fields in the encapsulating IP header. It is recalculated by the recipient and checked against the original, allowing simple transmission errors to be detected.

And that’s pretty much it for UDP message fields!

Part 4.1C Generate and Analyze UDP DNS Messages

Filtering a standard Internet connection, as we did above, makes it easy to capture and examine UDP messages. However, UDP is used by many different protocols, and the messages from them will be mixed together in the traffic stream. If we want to isolate the UDP datagrams created by a particular higher-level protocol, we can use port number filters, as we did before. Even here, though, we may have many different message exchanges happening concurrently; in our example we filtered for a *Destination Port* value of 443, but we could have several web browser tabs or windows accessing different servers on that port at the same time. You can actually see this in Figure 106, which shows at least three different HTTPS exchanges occurring simultaneously.

The best way to get a really good feel for how UDP works is to move from passively viewing the messages created by other programs to actively generating our own. This way we will know exactly what we are looking for, and can easily tie the messages that we see to the actions that create them.

The example we will use deals with an important problem that is solved so seamlessly on the Internet that most people never even think about it. All communication between hosts on the Internet is oriented around the use of IP addresses, those strings of numbers that identify source and destination devices; we've worked with them a few times already. However, people don't like having to remember lots of numbers; they prefer working with names. If you want to get to the Intrepid website, for example, it's much more convenient to remember to type **intrepidcs.com** into a browser tab than it is to remember **54.225.225.147**. Yet the number is what's needed to actually make the request.

A mechanism is required to convert the site name into its corresponding IP address, and this is provided in the form of the TCP/IP Domain Name System. DNS servers maintain databases of domain names (like **intrepidcs.com**) and IP addresses (like **54.225.225.147**), allowing one to be converted to the other and vice-versa. If you're thinking that this process sounds similar to the address translation function performed by ARP, then you're quite right. Domain names can be considered a type of address, and so DNS performs a translation service similar to ARP, though it works at a different level, and in a very different way.

DNS is a complex system with many different components, and uses both TCP and UDP. The name conversion function described above uses UDP, sending requests containing names to be converted, and receiving back replies containing IP addresses.

You should currently have Vehicle Spy 3 offline, with *Messages View* set to filter for UDP messages. We'll now use a web browser to generate a DNS request and look at it and the resulting reply.

- **Load Web Browser:** Start up any web browser.
- **Enter Test Page Name:** Enter the following web address into your browser, but do not yet hit *Enter* to load it: http://www.intrepidcs.com/ae/eevb/test.txt.

Try to perform the following steps as quickly as possible, to make it easier to spot the data we are interested in:

- **Go Online:** Go online within Vehicle Spy 3.
- **Load Test Page:** Press *Enter* to load the page mentioned above.
- **Go Offline.**

DNS requests are sent to port 53, so we can find the DNS request for **intrepidcs.com** by entering that value as a filter. After it appears, select the message and examine it.

- **Enter Dst Port Filter for DNS:** Enter 53 as a Dst Port filter.

In theory you should see only one entry, corresponding to the request we sent to Intrepid's site. However, it is possible you will see more than this if other processes are active on your computer. In this case click around until you find the right one.

- **Select the Intrepid DNS Request Message:** Select the UDP message corresponding to the request we sent to **www.intrepidcs.com**. You will be able to see the name clearly in the byte display on the right side in *Details View*, as shown in Figure 108.

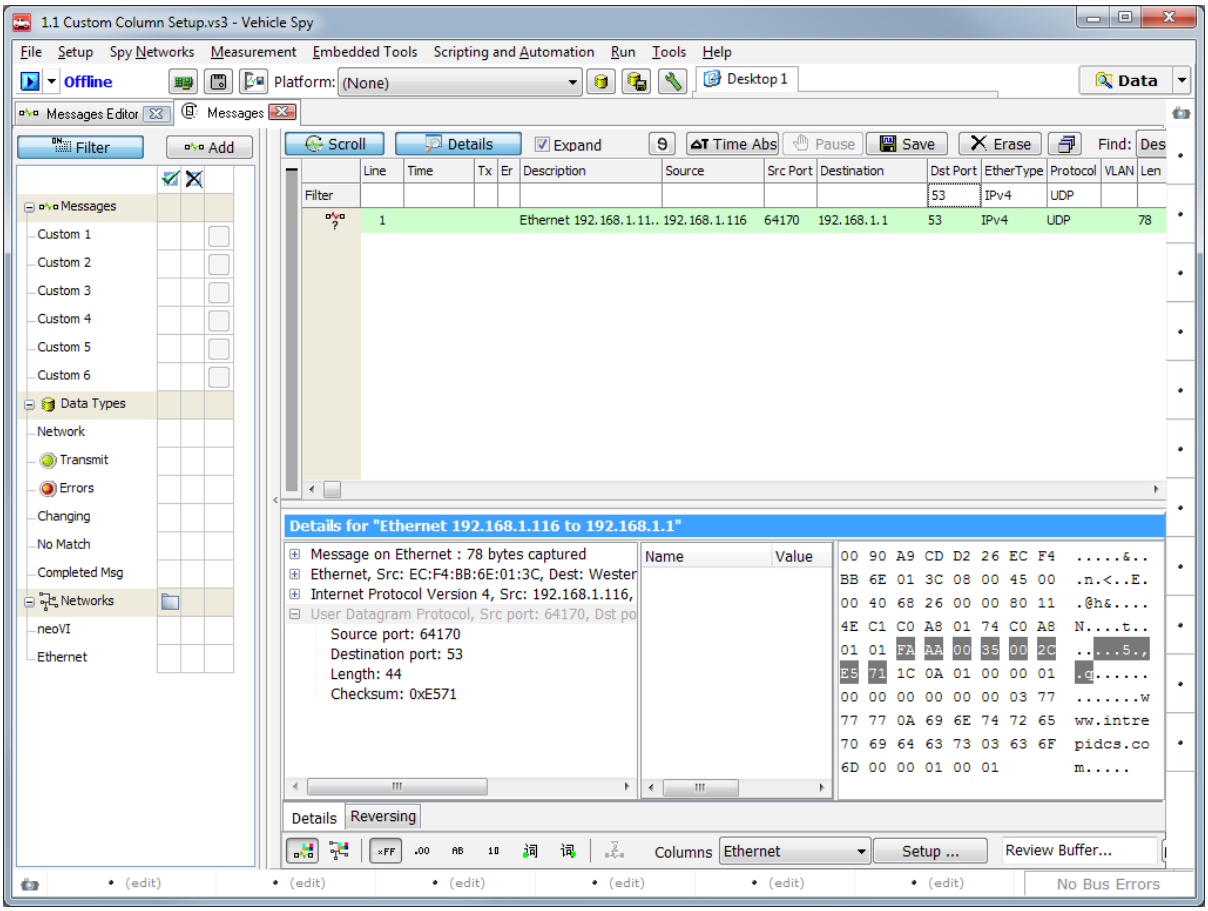


Figure 108: DNS Request in Messages View. Note that in this example the *Src Port* value is 64170; this will be the *Dst Port* value for the reply to this request. You can also see **www.intrepidcs.com** in the message, the name we are translating.

Now, make a note of the *Src Port* field value for this message, which is 64170 for the example shown in Figure 108. In most TCP/IP protocols, when a request is sent from a particular source port, the reply is sent back to that port. We can use this to quickly find the matching reply to the request we sent.

- **Change Dst Port Filter:** Change the value of this filter to match the *Src Port* of the request we just looked at. So again, in the example above we would enter 64170.
- **Click the DNS Reply Message:** A single UDP message should now appear; select it.

If you examine the byte display of this message you will again see the string www.intrepidcs.com in it. Now look at the last 4 bytes in the message: you should see the hex values 36 E1 E1 93; converted to decimal, these are 54 225 225 147, the IP address of the Intrepid web server.

- **Remove Dst Port Filter.**

You should now see something similar to Figure 109, though you may have many more messages, depending on what was running on your machine during this exercise. Removing the port filter allows us to see the full exchange, with the DNS request (line 3) and reply (line 4) adjacent to each other. In this case, it took about 29 ms for the request to be received and processed, and the reply sent back.

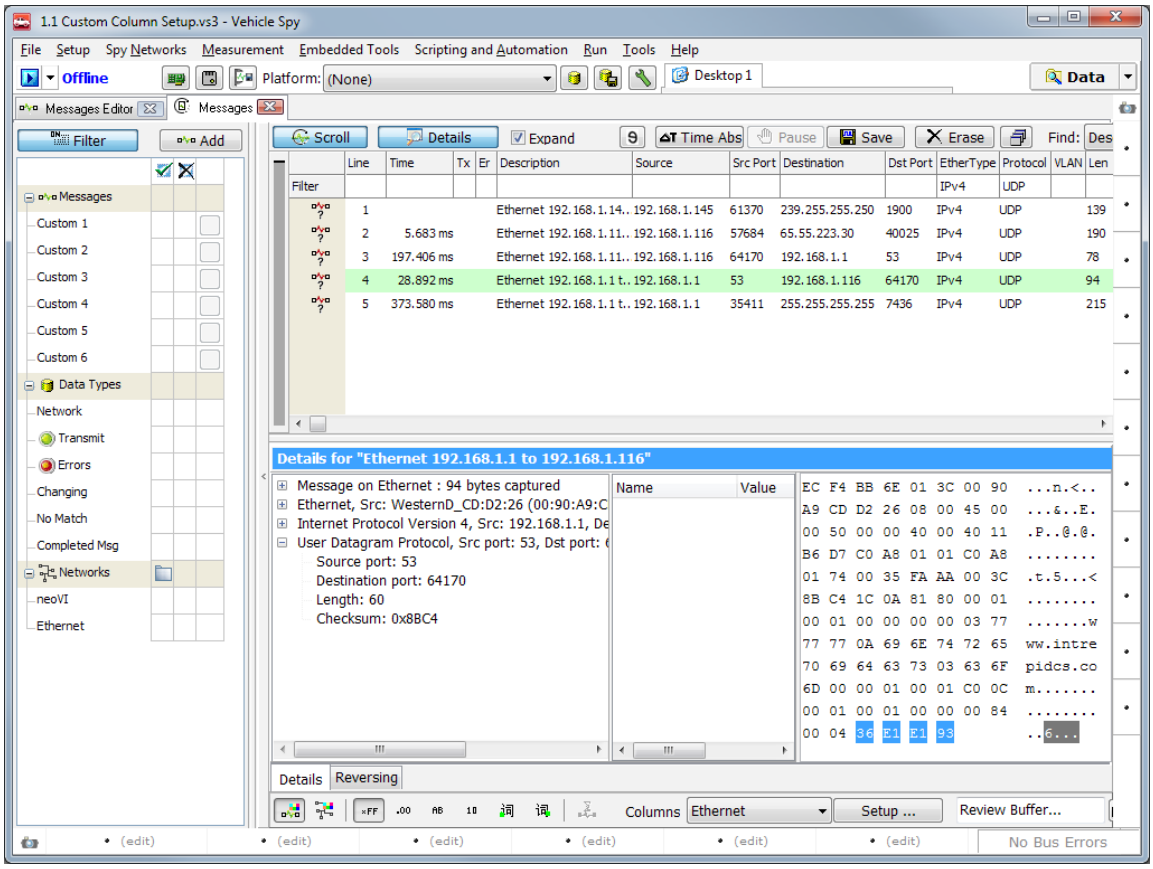


Figure 109: DNS Message Exchange. Line 3 shows the DNS request and line 4 the response. You can see that the *Src Port* for the request became the *Dst Port* for the reply. The last four bytes, which we manually highlighted before taking the screenshot, contain the IP address of www.intrepidcs.com.

- **Clear Column Filters:** Remove all column filters that are currently present within Messages View.

Part 4.1D Analyze a TCP Message

Let's now turn our attention to TCP messages. We'll begin again by simply selecting one and looking at its constituent fields.

You should have a number of TCP messages already in the Vehicle Spy 3 buffer. If you do not, go online for a few seconds and that should be enough to capture a few.

- **Set IPv4 EtherType Filter:** Enter IPv4 as the EtherType column filter.
- **Set UDP Protocol Filter:** Enter the Protocol column filter TCP.
- **Select Any TCP Message.**

As with the UDP message, we have four lines in the information pane, and again the first three are *Message on Ethernet*, *Ethernet* and *Internet Protocol Version 4*. The fourth line will now say *Transmission Control Protocol*; as was the case with UDP, this line will show the message's source and destination ports.

- **Select the Transmission Control Protocol Header.**

The TCP header is much longer than the UDP one, so this time you'll see 20 bytes highlighted in the byte display.

- **Expand the TCP Message Header:** Click the ⊞ button to the left of Transmission Control Protocol in the *Details View*.

Like the IPv4 header, the one for TCP has a special field dedicated to holding flags, which we should also expand so we can see them:

- **Expand the IPv4 Flags Field:** Click the ⊞ button to the left of Flags in the *Details View* area.

On a small screen you will probably find that there is insufficient space to see the entire TCP message headers, especially with the *Flags* field opened up. If so, increase the size of *Details View* so you can see all of the fields properly.

- **Expand Details View (If Necessary).**

Your Vehicle Spy 3 window should now appear similar to that shown in Figure 110.

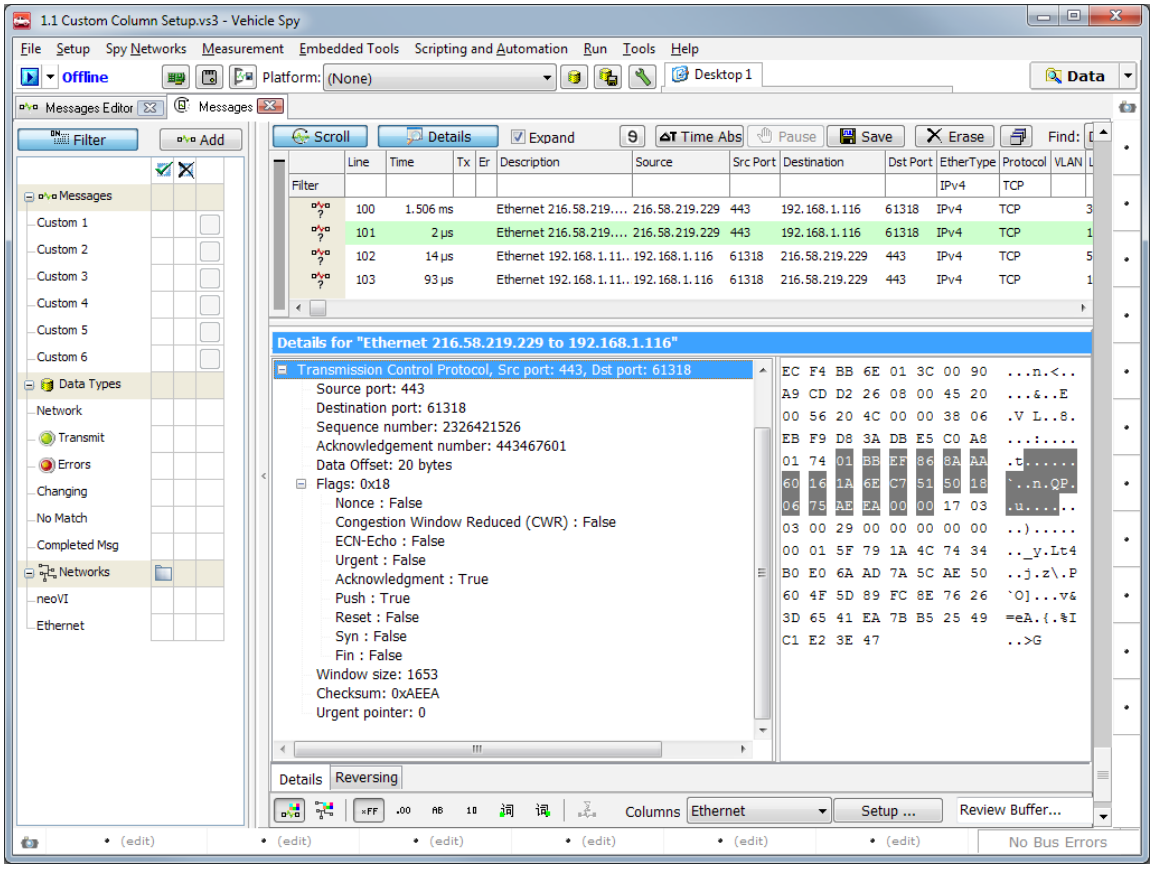


Figure 110: TCP Message Details.

Here's a summarized description of TCP headers and what they are used for:

- **Source Port:** Identifies the process that created the TCP message on the sending device.
- **Destination Port:** Specifies the intended recipient process on the destination device.
- **Sequence Number:** A numeric tag corresponding to the first byte of data in the message.
- **Acknowledgement Number:** Indicates the number of the data byte most recently acknowledged as having been received from the other device in this TCP connection.
- **Data Offset:** The size of the TCP header; this is stored internally in 32-bit increments; Vehicle Spy 3 shows it as a total number of bytes, which is usually 20.
- **Flags:** A set of nine status flags that control the operation of the protocol. The ones of most interest to us are:
  - **Urgent:** When set, indicates that part of the message contains data to be prioritized.
  - **Acknowledgment:** Indicates that this message is acknowledging the receipt of data from the other device.

- **Push:** Tells the other device to immediately send data to the receiving process rather than buffering it.
- **Reset:** Used to reset a connection when a problem is detected.
- **Syn (Synchronize):** Used to initiate a connection.
- **Fin (Finish):** Used to begin the process that terminates a connection.

- **Window Size:** Tells the other device how much data the sending device is willing to receive at one time.
- **Checksum:** A 16-bit checksum for transmission error protection.
- **Urgent Pointer:** Specifies where in the message urgent data can be found (when the Urgent flag is set).

If you increased the size of *Details View*, you can restore it back to roughly its original size now.

► **Restore Details View Window Size (If Necessary).**

Part 4.1E Using the TCP Column Display to Monitor a TCP Connection and Data Exchange

Many of the header fields described above play an essential role in the implementation of TCP's connection management and reliable data transfer capabilities. A full explanation of them requires understanding the protocol itself in detail; this is beyond the scope of this Lab Manual. However, we can learn a fair bit about these fields and how they are used simply by observing them in action. To do this, we will now set up a simple scenario that will let us see how a TCP connection is created, used and then terminated.

> **Note:** We will also learn more about how TCP connections work in Lab 4.4, where we will simulate the logic involved in connection establishment and termination using Vehicle Spy and the EEVB.

VSpy includes a tool that will make it easier to monitor these messages exchanges: a *Messages View* column set tailored specifically to TCP. We should still have some TCP messages in the buffer, so let's turn on this column set and see how it looks.

► **Switch to TCP Column Set:** Find the Columns drop-down box near the bottom of Messages View and switch it from Ethernet to TCP.

This set includes four TCP-specific columns, which are located to the right of the *Len* column. If you have VSpy currently set to a relatively small window size, it will be beneficial to expand it for the purposes of this part of the lab, if that's possible. Otherwise, you may wish to shrink

some of the other columns, such as *Description*, so you can see the special columns. The *Flags* column is especially important, so be sure you can see it easily.

Figure 111 shows an example of the Messages View with the TCP column set applied, with the window size and column widths adjusted to show the special TCP fields. This configuration has been saved as *4.1 TCP Column Setup* and included in the special zip file containing EEVB setups, so you should find it in the My Setups tab of the *Logon Screen*.

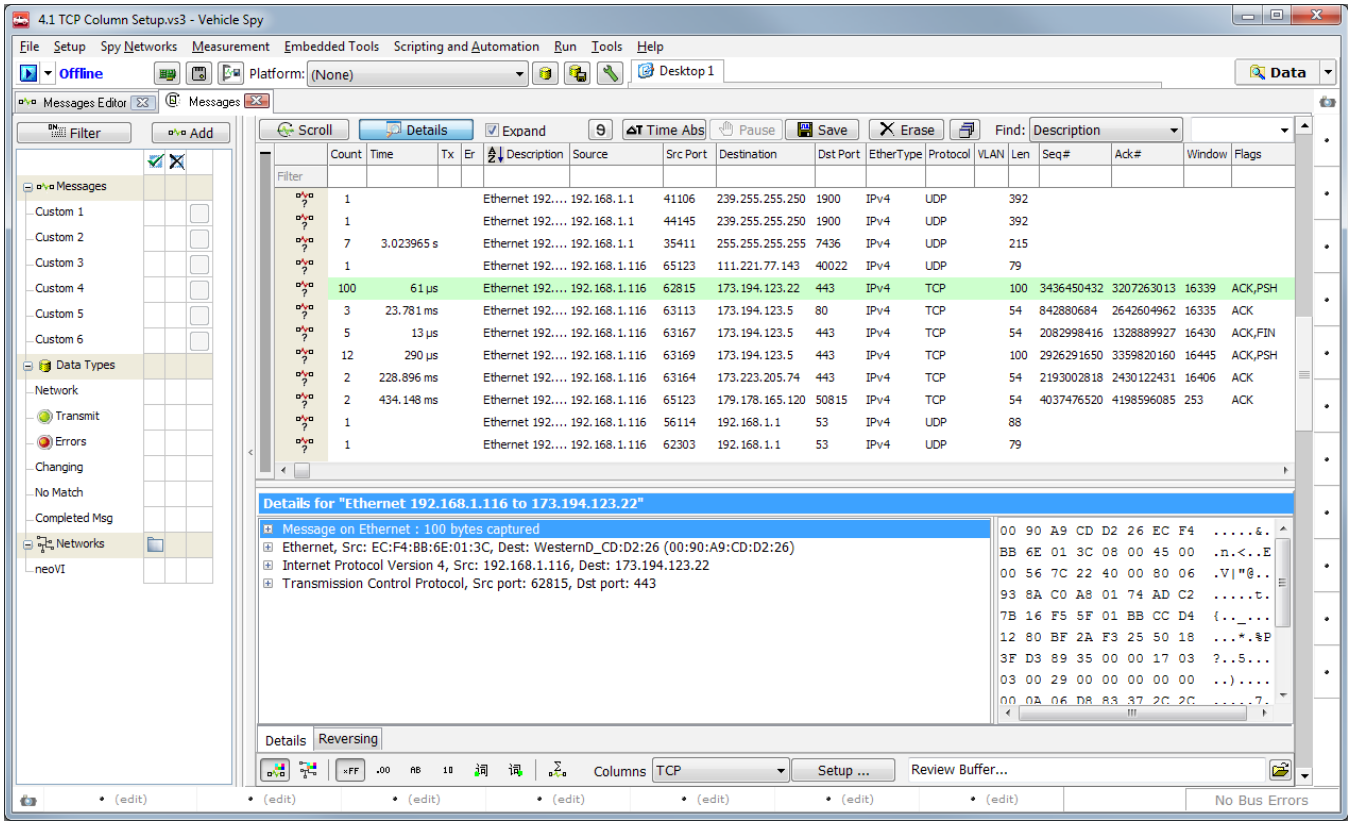


Figure 111: TCP Column Set in Messages View.

For our exchange we are going to load the same web page that we did earlier in the lab when we generated UDP messages. This time, however, instead of looking at the DNS messages generated to resolve the web page name, we will look at the actual web request and reply itself. Standard web pages use the Hypertext Transfer Protocol (HTTP), which is carried over TCP.

A typical web page contains many elements, including text and graphics, which means many messages going back and forth between web browser and server. This would be hard to follow, so instead, we are going to request a simple text file. Loading this file involves the web browser initiating a TCP connection to the server, sending an *HTTP Request*, receiving an *HTTP Reply*, and then shutting down the TCP connection.

Let's give it a go.

- **Load Web Browser:** Start up any web browser.
- **Enter Test Page Name:** Enter the following into your browser, but do not yet hit *Enter* to load it: http://www.intrepidcs.com/ae/eevb/test.txt.
- **Enter Description Column Filter:** Enter 54.225.225.147 as a Description column filter.
- **Clear Other Column Filters (If Necessary).**
- **Enable Scroll Mode (If Necessary).**

You may recognize the IP address in the column filter as corresponding to Intrepid's web server. Entering the filter in the *Description* field will ensure that we can isolate the transaction to ICS's site while ignoring other traffic on the connection.

Okay, do these next three steps together, as quickly as possible:

- **Go Online:** Go online within Vehicle Spy 3.
- **Load Test Page:** Press *Enter* to load the web page.
- **Go Offline.**

You should now see a total of 10 messages displayed in Vehicle Spy 3, as shown in Figure 112. Below we describe each message and what it does to give you an idea of how TCP operates. To help explain what is happening, we make reference to various field values in these messages as seen in the figure; the numbers will of course be different on your machine (and will change each time this exercise is done).

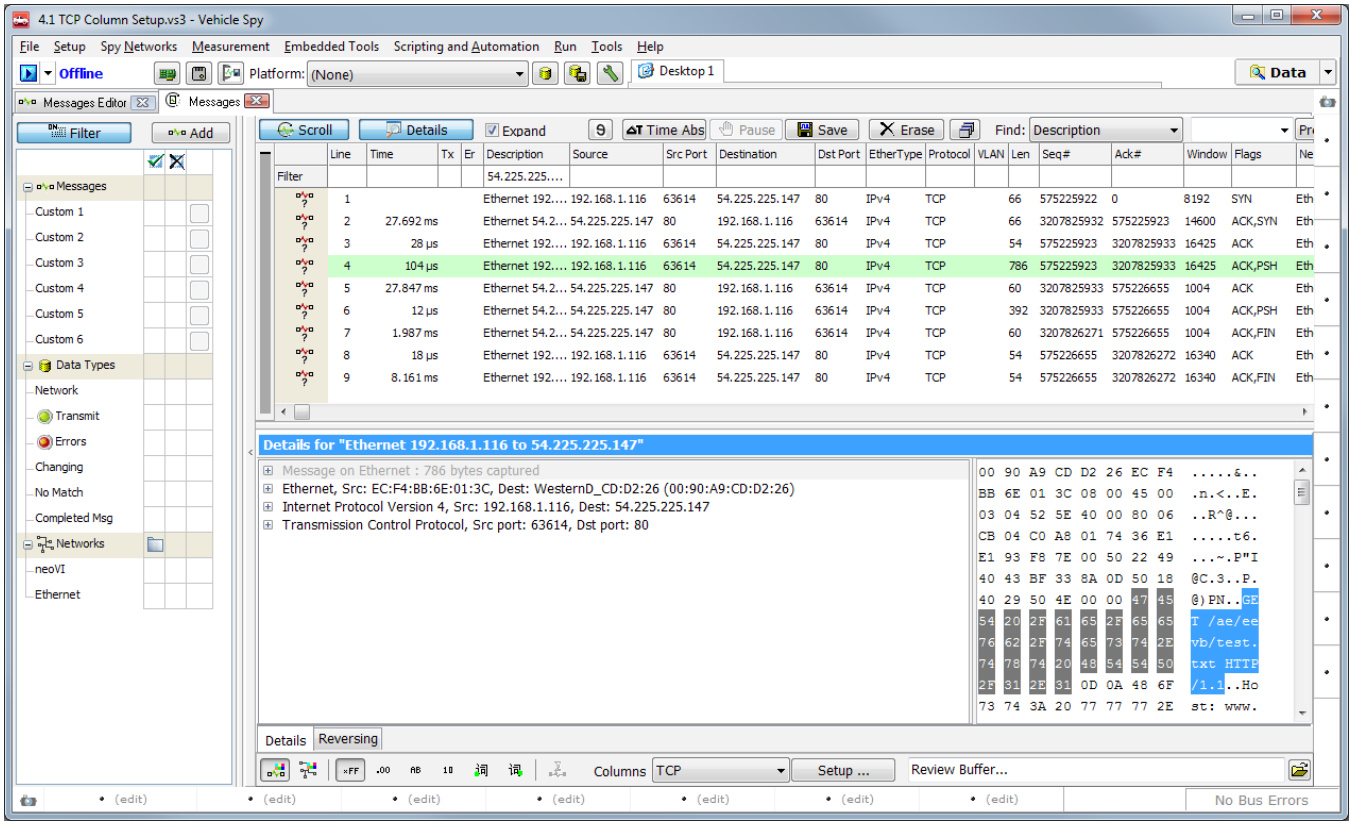


Figure 112: TCP Connection and HTTP Exchange in Messages View. The first three messages establish a TCP connection between the two devices; the next three are used by the client to request the web page and the server to send it; and the final four terminate the connection. Message 4, which contains the web request, has been selected here, and the bytes of the actual request string highlighted.

The first three messages comprise the TCP "three-way handshake" used to establish a connection:

1. **Client SYN:** Sent by the client to the server to request a connection; notice the *SYN* flag is set, the *Seq#* is 575225922 (a number chosen by the client at random) and *Ack#* is 0 (because nothing has been sent yet, so nothing is acknowledged).

2. **Server SYN+ACK:** Sent in response from the server to the client, this message acknowledges the client's *SYN* message; notice that its *Ack#* is 575225923, which is 1 higher than the previous message's *Seq#*. The *Seq#* field here is the server's sequence number.

3. **Client ACK:** Sent by the client to acknowledge the server's *SYN* in the previous message. The *Ack#* is 1 higher than message 2's *Seq#*, and the *Seq#* matches the *Ack#* sent in message 2.

The next three messages comprise the actual HTTP exchange:

4. **HTTP Request:** This is the actual request sent by the browser. HTTP sends much of its data as plain text, so if you click the message you can see much of the information

the browser is sending. The following is the actual HTTP command, which is also highlighted in Figure 112: **GET /ae/eevb/test.txt HTTP/1.1**. Other lines contain additional parameters required for the request. Notice that the *PSH* flag is set for this message, telling the recipient TCP implementation to send it right through to the web server without waiting for subsequent data.

5. **Request Acknowledgment:** An acknowledgment of the client's request, sent by the server.
6. **HTTP Response:** A message containing the requested page. Click this message and scroll down in the byte display and you will find the actual text of the file near the end: **This is a simple text file for testing Ethernet within Vehicle Spy 3.**.

The last four messages are used to terminate the connection:

7. **Server ACK+FIN:** The server has no more data to send, so it sends a message with the same *Ack#* as in message 6, and sets the *FIN* flag.
8. **Client ACK:** Client acknowledges the server's desire to close the connection.
9. **Client ACK+FIN:** Client tells server it is ready to close the connection.
10. **Server ACK:** Server acknowledges the client; connection is now closed.

We're now done with this lab.

- **Close Web Browser.**
- **Resize VSpy Window:** If you made the VSpy window larger for this part of the lab, you can return it to its default size now.

Lab 4.2 Transmitting Input/Output Data Using UDP

If you've been following the Lab Manual from the beginning and doing all of its experiments, you are pretty much an EEVB veteran by now! And if you have done all of the labs up to this point, you probably also did the initial demo that's described in the EEVB User's Guide, which we used to illustrate the basic operation of the board. You may recall that the demo in question programmed both EEVB nodes to send a simple raw Ethernet message every 100 milliseconds, and allowed you to control the flash rate of the board's LEDs using its potentiometers.

In this lab we will revisit this demo, using a variation of the initial demo that replaces the raw Ethernet messages with equivalent versions using IP and UDP. This serves as an example of how UDP might be used in a real TCP/IP network, and is also more representative of how Automotive Ethernet messages actually work than the raw Ethernet messages are.

Since you are now experienced with VSpy, this time around we will go through all of the message definitions and function block scripts to explain how the demo works in detail. Part of this will be demonstrating how this single setup implements different behavior in Node A and Node B, which is accomplished in a slightly different way than we have done before in the Lab Manual.

Part 4.2A Load and Analyze UDP Input Output Script

As described above, this demo works in much the same way as the one in the User's Guide. We'll start as we often do, by loading the setup file that contains the message definitions and function blocks. We then walk through the setup and explain each component.

- **Load 4.2 UDP Input Output:** As usual, you can find this file under the My Setups tab on the *Logon Screen*.

Vehicle Spy 3 will present you with a screen that looks nearly identical to the one you saw with the User's Guide demo, showing *Messages View* in the top half of the screen, and a pair of signal plots on the bottom.

If you have a relatively small screen and you find that *Details View* is mostly obscuring the main *Messages View* window, please turn it off.

- **Disable Details View (If Necessary).**

Let's also close the signal plots for now, so we have more room to examine our setup.

- **Close Signal Plot Windows:** Click the small  button on the top right of each of these small windows.

Let's take a look at the message definitions in this file.

- **Switch to Messages Editor, Transmit Side.**

You will see two messages defined here, named *Node A UDP Data* and *Node B UDP Data* (Figure 113). These are the same as the messages in the User's Guide demo, but notice from their summaries that they are defined as IPv4/UDP messages based on their *EtherType* and *Protocol* field values. The *Source* and *Destination* values are IP addresses, with the two nodes containing complementary values of 10.0.0.1 and 10.0.0.2. These, along with the port numbers, are all default values that come from the UDP Ethernet packet template.

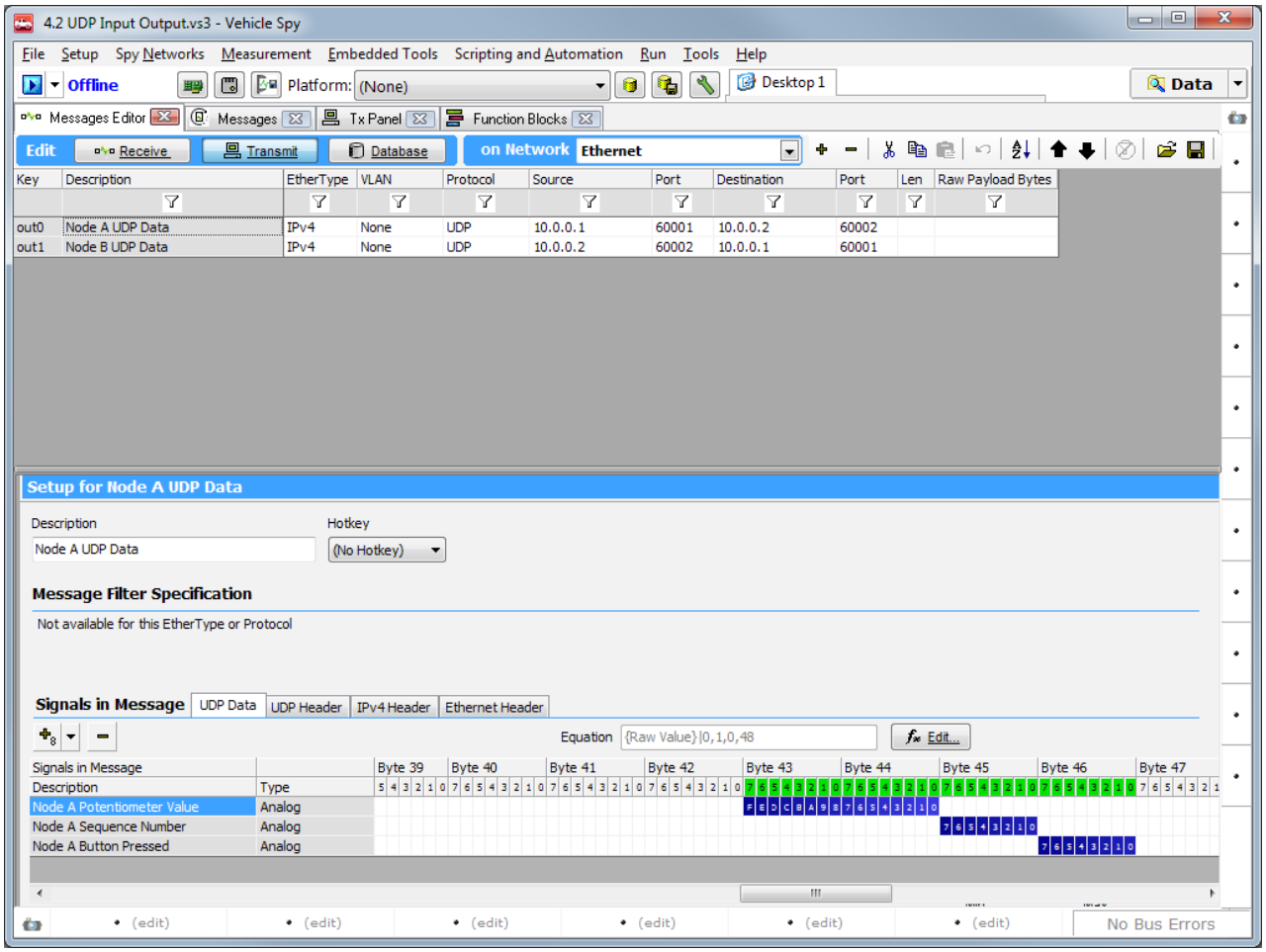


Figure 113: UDP Input Output Message Definitions.

At the bottom of the screen you can see that there are four tabs defined for this message. From right to left (reflecting the order they appear in the message itself) these are: *Ethernet Header*, *IPv4 Header*, *UDP Header* and *UDP Data*. The three data signals shown should look familiar, as they are the same ones we saw in the earlier demo upon which this one is based. However, they are now in the *UDP Data* section, nested under the UDP, IPv4 and Ethernet headers. This also means that they are much "deeper" within the message.

- **Scroll the Signal Display to Find the Data Signal Bytes:** Move the scroll bar to the right until the signal bytes are visible.

As you can see in Figure 114, these signals now occupy bytes 43 to 46 of the message, as opposed to bytes 15 to 18; the difference of 28 corresponds to the 20-byte IPv4 and 8-byte UDP header lengths.

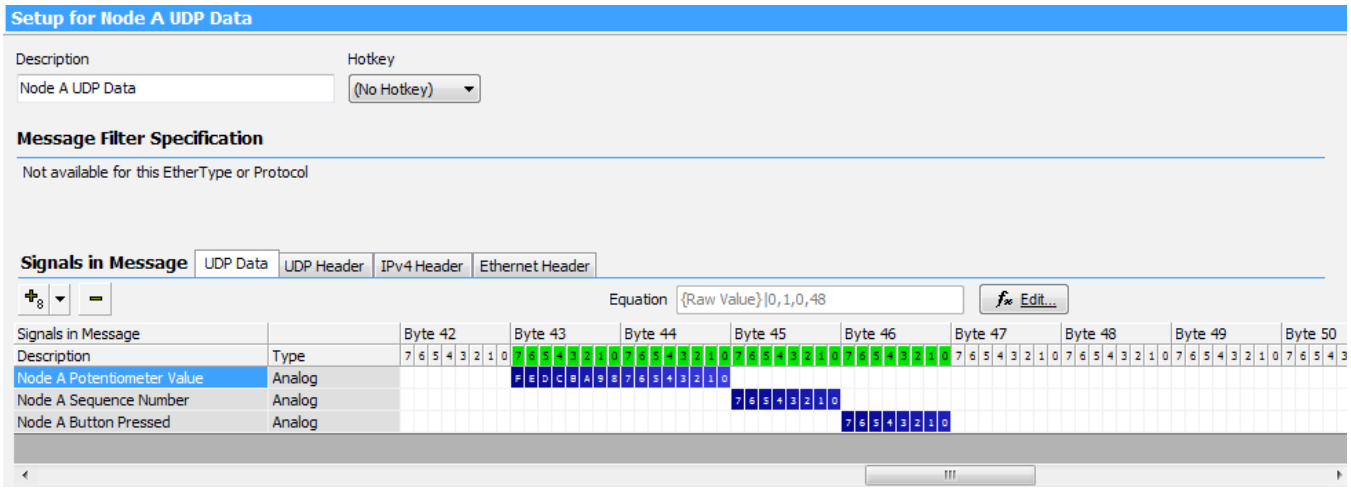


Figure 114: UDP Input Output Data Signals.

► **Switch to Receive Side.**

The message definitions here are identical to the ones on the transmit side; they were copied here to allow the messages coming from the EEVB to be properly decoded in VSpy.

► **Switch to Function Blocks.**

Here we see five different function block scripts, which makes the setup seem rather complex, though it's really not. Let's go through them one at a time.

► **Click the Node A Send UDP Data Script.**

This script (Figure 115) sets the three transmit message signals for the *Node A UDP Data* message. The *Node A Sequence Number* field is set by incrementing its previous value, while the *Node A Potentiometer Value* and *Node A Button Pressed* signals are set to their respective physical input device readings. The message is then transmitted, and the script waits for 100 milliseconds before running again.

Step	Description	Value	Comment
1	Set Value	{Node A Sequence Number (Value) :out0-sig23-0} = {Node A Sequence Number (Value) :out0-sig23-0} + 1	// Increment node sequence number.
2	Set Value	{Node A Potentiometer Value (Value) :out0-sig22-0} = {Analog Input 1 (Value) :neo0-ai0-0-index(0)}	// Set potentiometer value to current reading of analog input 1.
3	Set Value	{Node A Button Pressed (Value) :out0-sig24-0} = {Switch 1 (Value) :neo0-sw0-0-index(0)}	// Set pushbutton status based on switch 1 value.
4	Transmit	Node A UDP Data	
5	Wait For	= 100	

Figure 115: Node A Send UDP Data Function Block Script.

► **Click the Node B Send UDP Data Script.**

Of course, the *Node B Send UDP Data* script is the same as the one above, but runs on Node B instead of Node A.

► **Click the Node A LED Blink Script.**

As you have undoubtedly figured out on your own, this script (Figure 116) flashes one of the user-controllable LEDs on Node A. It starts by toggling the status of LED1, turning it off if it is currently on and vice-versa. It then waits for an amount of time based on the read potentiometer value of the node. The potentiometer produces a value of 0 to 4,095, which is divided by 4.095 resulting in a value of 0 to 1000 milliseconds. The value 20 is then added to produce a minimum flash time of 20 ms; this was done because a flash rate any smaller than about 20 ms causes the LED to flash too quickly for most people to notice (it just looks like it is solidly lit).

Step	Description	Value	Comment
1	If	{LED 1 (Value) :neo0-ld0-0-index(0)}	// Check current value of LED1, where a value of 0 means it is off and 1 means it's on.
2	Set Value	{LED 1 (Value) :neo0-ld0-0-index(0)} = 0	// Current value is 1, so change it to 0.
3	Else		
4	Set Value	{LED 1 (Value) :neo0-ld0-0-index(0)} = 1	// Current value is 0, so change it to 1.
5	End If		
6	Wait For	= {Analog Input 1 (Value) :neo0-ai0-0-index(0)} / 4.095 + 20	// Wait for 20 to about 1000 milliseconds, based on the value of the node potentiometer (which ranges from 0 to 4095).

Figure 116: Node A LED Blink Function Block Script.

► **Click the Node B LED Blink Script.**

And again, this script does the same as above for Node B, except this time using LED2 instead of LED1 (just for variety).

► **Select the Node Initialization Script**

We typically want to have different functionality running in each of the nodes of the EEVB. Thus far we have accomplished this in two ways.

The simplest method, which we used early in the Lab Manual, was just to have two different setup files. However, this has a few drawbacks, such as the need to duplicate message definitions, and having to swap back and forth when sending CoreMinis to the nodes.

In Lab 3.2 we introduced the idea of reading the EEVB serial number to allow a single script to act differently based on whether it is running on Node A or Node B. This works well, but is only well-suited to short scripts or ones where very few of the steps change depending on node. Longer scripts can get cumbersome if it's necessary to check which node we are running on many times.

In this setup (see Figure 117) we introduce a third option, which is ideal for more complex programs. We define separate function block scripts to run on Node A and on Node B, as we did with the two pairs of Send UDP Data and LED Blink scripts. Then we have a fifth script,

here called Node Initialization, that queries the device and then starts other scripts appropriate to the node it determines it is running on.

The first step in the script does the actual serial number comparison. If true, the two Node A scripts are started, and if false, the two Node B scripts. You'll also see that there are two statements to control LEDs: these turn off the LEDs that aren't going to be flashed, which means setting LED2 to zero on Node A, and LED1 to zero on Node B. Notice that the final step here is a Stop command, because we don't want this script to run continuously, only once.

Step	Description	Value	Comment
1	If	{Device Serial Num} mod 2 = 0	// If serial number is even, this is Node A.
2	Set Value	{LED 2 (Value) :neo0-ld1-0-index(0)} = 0	// Turn off LED2 on Node A since we use LED1 on that node for display.
3	Function	Start Node A Send UDP Data	
4	Function	Start Node A LED Blink	
5	Else		// Otherwise, it is Node B.
6	Set Value	{LED 1 (Value) :neo0-ld0-0-index(0)} = 0	// Turn off LED1 on Node B since we use LED2 on that node for display.
7	Function	Start Node B Send UDP Data	
8	Function	Start Node B LED Blink	
9	End If		
10	Stop	n/a	// Run this script only once.

Figure 117: Node Initialization Function Block Script.

One final part of the puzzle is necessary for this setup to work. Take a look at the *Start Type* column in the summary list of the function block scripts, and you'll see that the *Node Initialization* script is set to **Immediate Embedded Only**, while the others are **Manual**. This ensures that when downloaded to the EEVB, only the *Node Initialization* script runs; it then starts the other nodes as appropriate.

Part 4.2B Reload and Run UDP Input Output Script

Now that we know what this script does, let's run it and take a look at the output it produces. We'll actually start by reloading the setup file; this is the easiest way to get back the mixed setup with *Messages View* on top and the two signal plots on the bottom.

- ► **Load 4.2 UDP Input Output.**

Next, send the script to both nodes.

- ► **Send CoreMini to Node A.**
- ► **Send CoreMini to Node B.**

As was the case with the User's Guide demo, you should now see LED1 on Node A blinking at a rate you can change with its potentiometer, and the same for LED2 on Node B.

- ► **Set Message Filter:** Enter UDP in the Description filter field, so we only see our messages.
- ► **Turn Scroll Mode Off (If Necessary).**
- ► **Go Online.**

You should now see a display similar to Figure 118, which again, will be very familiar if you did the similar demo in the User's Guide.

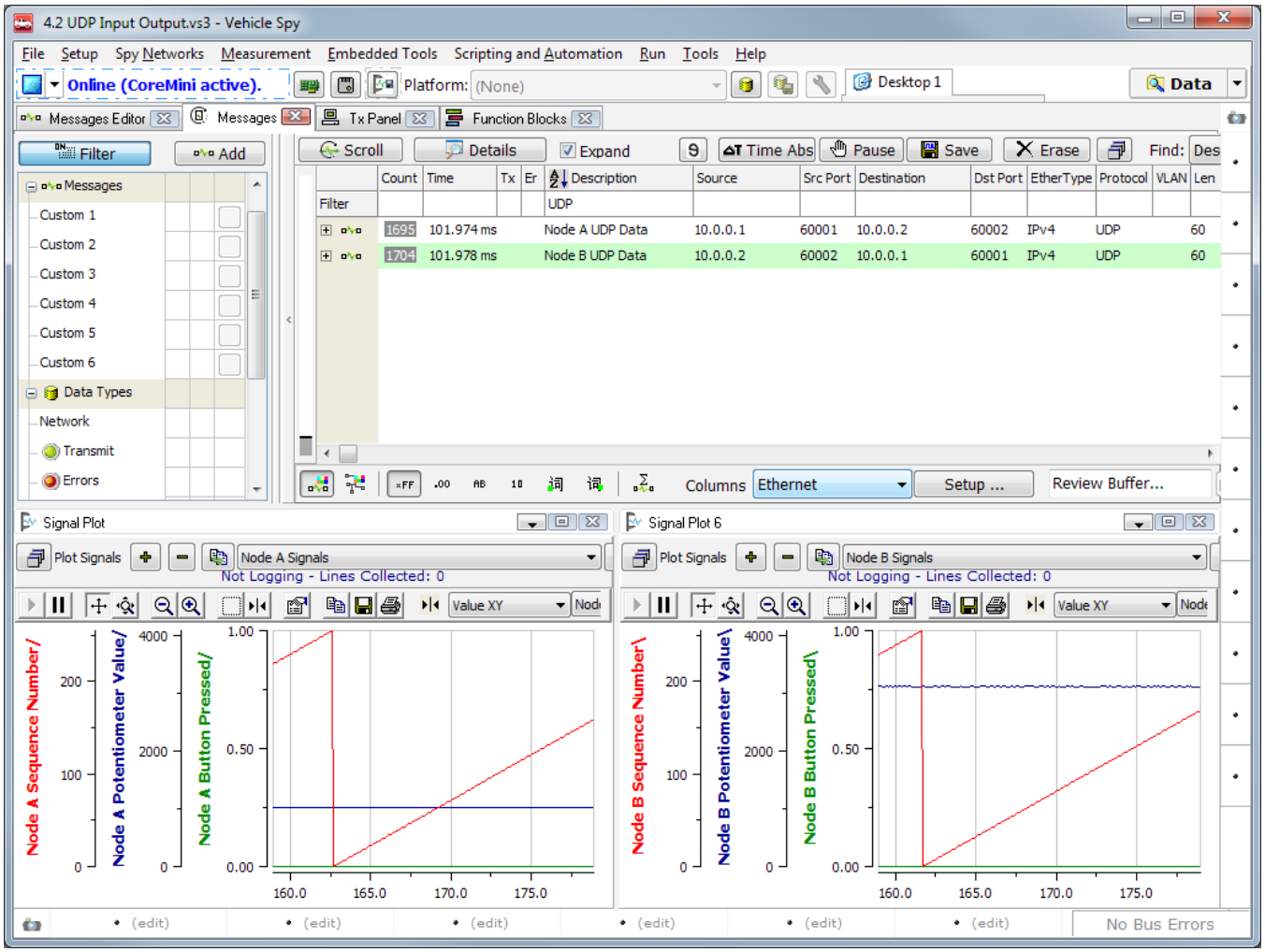


Figure 118: UDP Input Output Messages and Signal Plots.

Let's now take a look at those messages in more detail. Speaking of which, we should turn *Details View* back on; once again, we will close the signal plots to make more room.

- ► **Close Signal Plot Windows.**
- ► **Enable Details View.**
- ► **Select Node A UDP Data Message.**

On the left side of *Details View* you will see the four lines we expect for a UDP message, the first being the standard one for Automotive Ethernet messages, and the next three containing this message's nested headers: Ethernet, IPv4 and UDP. As always, you can expand these headers to see the fields they contain. The header fields and their current values are also

shown in the Name/Value section in the middle of *Details View*, since this message is being recognized and fully decoded by VSpy.

► **Scroll Down in the Name/Value Section of Details View.**

Near the bottom of the screen you will see our three data fields, with their values changing in real time: the sequence number counts up, the potentiometer value fluctuates slightly or changes more if you move the dial, and the button value changes from 0 to 1 if the button is pressed. Of course, we can do the same for the Node B UDP Data message.

► **Go Offline.**

Lab 4.3 Creating a Simple Custom UDP Message Exchange Protocol

UDP is a common choice for simple messaging protocols because it is simple itself and allows messages to be sent over IP internetworks with a minimum of fuss or overhead. As mentioned in the introduction to this section, it is ideal for time-sensitive messaging, and also in cases where it doesn't matter if data is lost, because it can just be transmitted again.

In the previous lab we used UDP to send data generated by the EEVB's input devices, as well as a sequential counter. This is not really how most TCP/IP protocols work, however, because the same data is sent automatically, and there is no interaction between devices. In this lab we will use a custom-designed UDP protocol based on the classical client/server method of communication: the client sends requests, and the server responds to them.

Our creation, which we have dubbed a *headline server*; it is intended to be a simplified version of a news service. Here's how it works.

The client, which runs on EEVB Node A, sends requests for news headlines to the server:

- There are five types: Global News, National News, Local News, Weather, and Sports.
- The headline requested is specified by the position of the potentiometer dial, which is split into five roughly-equal regions.
- A new headline request is sent every 5 seconds. In a real implementation of this sort of service, requests would instead be triggered by users, but this method makes it easier for us to see the client/server exchanges in action.
- In addition to the regular headline requests, there are two special requests. The first asks for a repeat of the most recently-requested headline (ignoring the current potentiometer setting). The second asks for the date/timestamp of the current headline data. These are sent in alternating fashion when the node's pushbutton is held down.

The server is designed to run in Vehicle Spy on the PC so that we can easily monitor the requests coming in and responses going out:

- A set of five application signals holds the various headlines. These can be manually edited to whatever values you like.
- Additional application signals hold the value of the last headline and the date/timestamps for the special requests.

As we've seen in earlier labs, this is again a simplified version of this particular messaging functionality. In a real implementation we would probably focus on making the protocol more flexible by allowing more types of data to be requested. As mentioned, requests would come from actual users rather than just being generated automatically. We'd also have a slightly more elegant way of selecting the headline we want, since the EEVB's inputs are quite limited. Finally, a proper server would use a database for the information it provides rather than hard-coded variables. All that said, this cut down implementation will give you a good idea of the sort of exchanges you can design using TCP/IP within Automotive Ethernet.

Part 4.3A Load and Analyze Headline Client Script

Let's begin by looking at the headline client script, which generates requests. As usual, we'll load the script, then walk through the setup before sending it to the EEVB to run.

► **Load 4.3 Headline Client:** This file can be found with the other lab example scripts under the My Setups tab on the *Logon Screen*.

You will begin in the *Messages Editor* on the transmit side. There is only one message defined here: *Headline Request*, a UDP message using VSpy-default IP addresses and port numbers. In the signal area at the bottom of the screen you will find that it has three custom fields:

- **Operation:** Defines what sort of request is being sent.
- **Headline Code:** For regular headline requests, which headline is wanted.
- **Request Number:** A sequential counter to identify each request as it is sent

As was the case for the messages in Lab 4.2, these signals begin at byte 43, because the Ethernet, IPv4 and UDP headers combine to take up 42 bytes.

► **Scroll the Signal Display:** Move the scroll bar to the right to find the signal bytes.

You should now see something similar to the display in Figure 119.

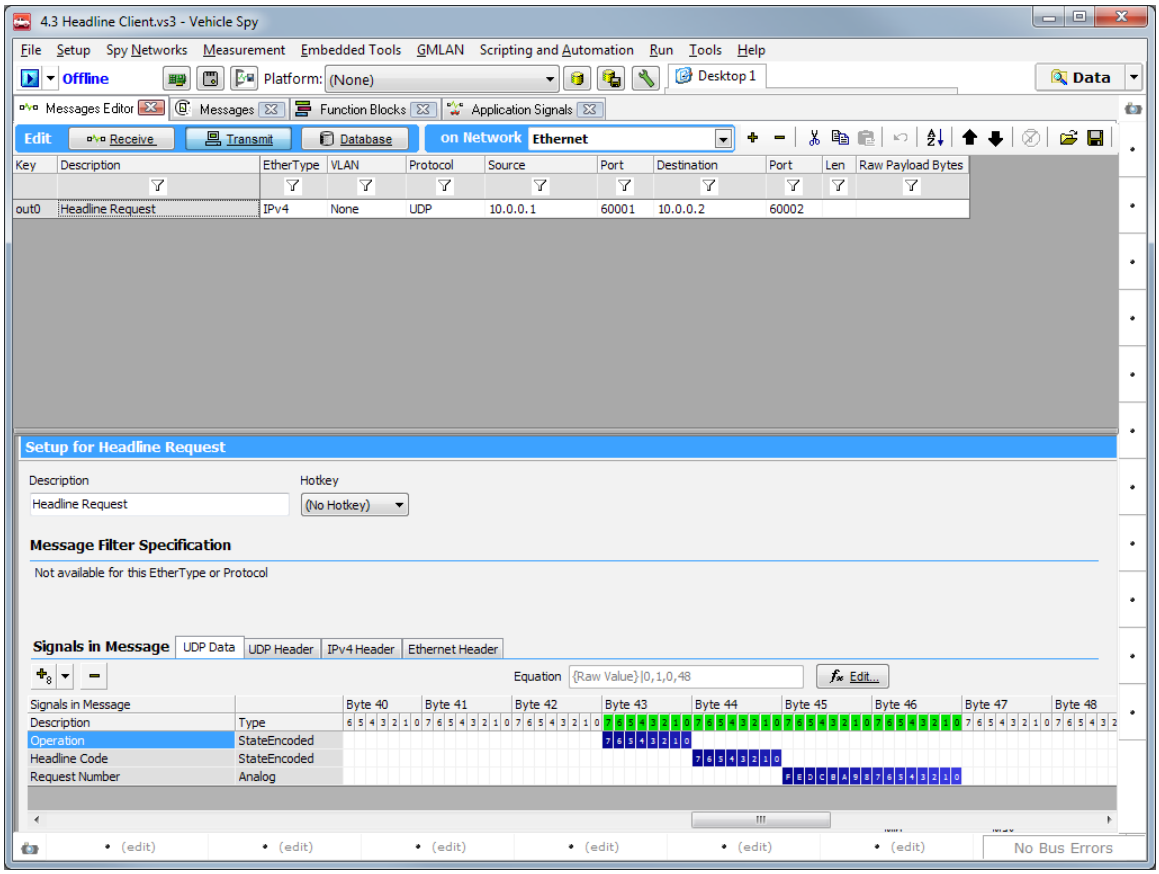


Figure 119: Headline Request Message Definition and Signals.

Note that two of these messages are *state-encoded*, which means that they carry integers that correspond to specific string values. We'll see later on that when these messages are displayed, the strings are shown in addition to the numbers, making the data values more meaningful and the signals easier to understand.

Let's take a look at the state values for the *Operation* field.

► **Edit Signals for the Operation Field:** Select the *Operation* field, then click the Edit... button above the signal byte display.

A dialog box appears. Near the bottom you will see a set of *State* and *Raw Value* pairs, showing the state encoding for this signal (Figure 120). There are three values, though you will need to scroll the small box containing the values to see them all: **RequestHeadline** (1), **LastHeadline** (2) and **RequestTimestamp** (3). These of course correspond to the three possible requests that the client makes.

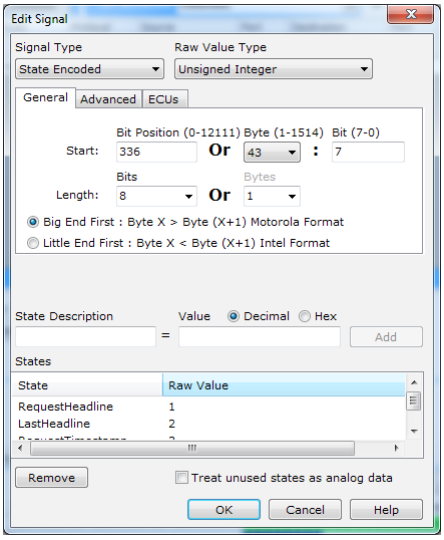


Figure 120: Messages Editor Edit Signal Dialog Box.

► **Switch to Receive Side.**

Here we see a copy of the transmit message. This client only transmits, it doesn't receive or process data from other devices. The copy of the transmit message is here only so we can take a look at the data before we load the headline server script; otherwise, it would not really be necessary either.

► **Switch to Function Blocks.**

There is only one function block for this script, which is shown in Figure 121. The script begins by incrementing the *Request Number* field, which labels each request with a sequential number. If the node's pushbutton is currently up, a normal request is generated by setting the *Operation* field to a value of 1, and then setting the *Headline Code* field to a value from 1 to 5 depending on the potentiometer setting. If the pushbutton is held down, then either a request for the previous headline or the timestamp is sent, with the requests alternating based on the

value of the *Request Timestamp Next* application signal. After filling in data for the request, it is sent, and then a delay of 5 seconds occurs before the process repeats. Please refer to the comments in the function block for a detailed description of each step.

Step	Description	Value	Comment
1	Set Value	{Request Number (Value) :out0-sig25-0} = {Request Number (Value) :out0-sig25-0} + 1	// Sequentially increment request number field; first request will be 1.
2	If	{Switch 1 (Value) :neo0-sw0-0-index(0)} = 0	// By default (node pushbutton up) send a normal headline request.
3	Set Value	{Operation (Value) :out0-sig22-0} = 1	
4	Set Value	{Headline Code (Value) :out0-sig23-0} = int({Analog Input 1 (Value) :neo0-ai0-0-index(0)}/820) + 1	// Generate a headline number from 1 to 5. This is done by dividing the current node potentiometer value into roughly 5 equal areas so the number changes as the dial is rotated.
5	Else		// If the node's pushbutton is held down, alternate between requesting for the previously sent headline
6	Set Value	{Headline Code (Value) :out0-sig23-0} = 0	// For both of the special request types, the Headline Code should be set to 0 since it is not used.
7	If	{Request Timestamp Next :sig1-index(0)} = 1	// Flag is 1, so request the timestamp next while button is down, then change flag to 0.
8	Set Value	{Operation (Value) :out0-sig22-0} = 3	
9	Set Value	{Request Timestamp Next :sig1-index(0)} = 0	
10	Else		// Flag is 0, so request previous headline, then set flag to 1.
11	Set Value	{Operation (Value) :out0-sig22-0} = 2	
12	Set Value	{Request Timestamp Next :sig1-index(0)} = 1	
13	End If		
14	End If		
15	Transmit	Headline Request	// Send the request.
16	Wait For	5.000000 sec	// Wait for a number of seconds equal to the value of the Request Send Rate variable.

Figure 121: Headline Client Function Block Script.

Part 4.3B Run Headline Client Script

Let's send the script to Node A and then quickly take a look at the messages it is sending. We will disable *Details View* for the moment as well, since these messages have many fields (including headers) and this will make it easier to see them.

- **Send CoreMini to Node A.**
- **Switch to Messages View.**
- **Set Message Filter:** Enter Headline in the Description filter field, so we only see our messages.
- **Turn Scroll Mode Off (If Necessary).**
- **Disable Details View.**
- **Go Online.**
- **Expand the Headline Request Message to Show Signals.**

Vehicle Spy should now appear similar to Figure 122.

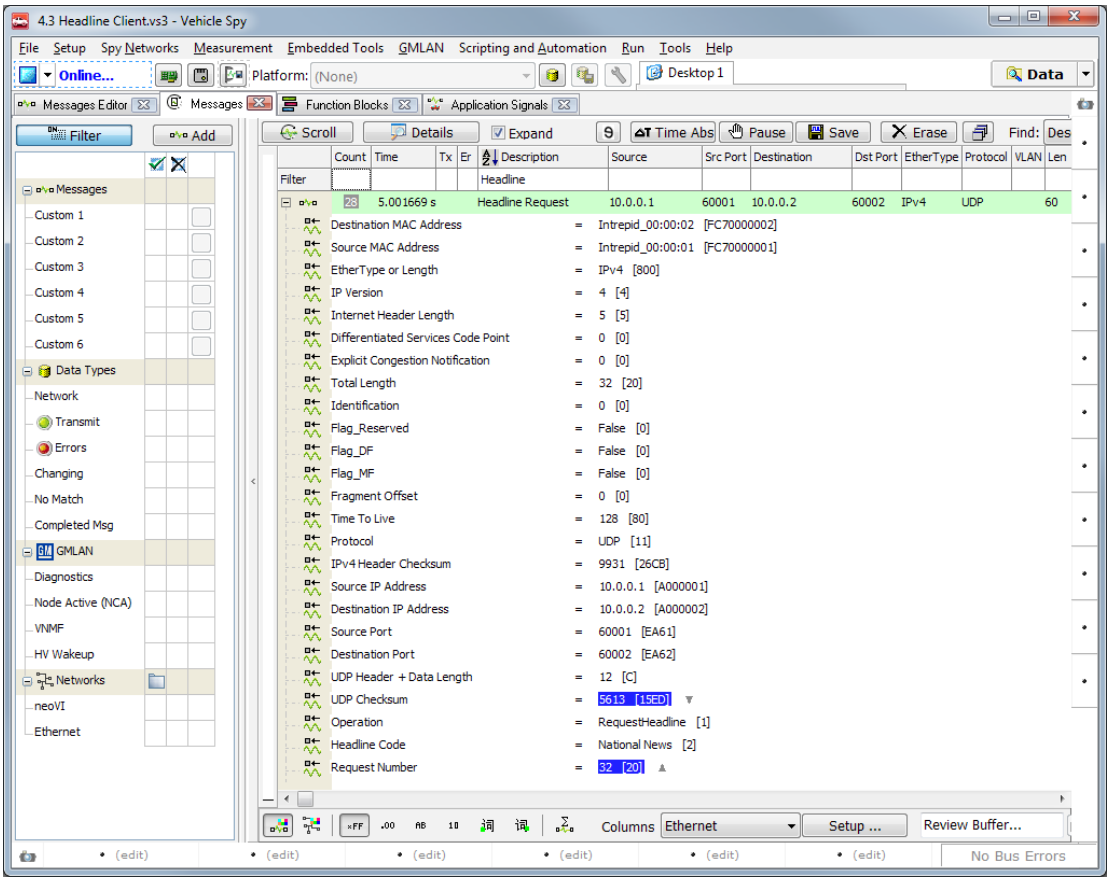


Figure 122: Headline Request Messages Sent by EEVB Node A.

A new *Headline Request* will be sent approximately every 5 seconds, with the *Request Number* field increasing each time a new message is received. Notice that the *Operation* and *Headline Code* fields are shown with both their string descriptions and raw values in square brackets.

► **Turn the Node A Potentiometer.**

Moving the dial will alter the *Headline Code* value, allowing you to select one of the five predefined types.

► **Hold Down the Node A Pushbutton.**

While the button is depressed, the *Operation* value will change to alternate between LastHeadline and RequestTimestamp.

► **Release the Node A Pushbutton.**

Now the Operation value will go back to sending RequestHeadline again.

► **Go Offline.**

Part 4.3C Load and Analyze Headline Server Script

Let's now look at the server side of the equation.

► **Return to Logon Screen.**

► **Load 4.3 Headline Server.**

Once again we start in the *Messages Editor*, transmit side, where there is a single message unsurprisingly called *Headline Response*. The first three signals are the same as those in the *Headline Request* message; the fourth is a text field which is used to carry the actual data requested.

► **Switch to Receive Side.**

On the receive side we have a definition of the *Headline Request* message, which is what this script responds to. It is the same as the definition in the headline client setup. We don't need a copy of the *Headline Response* message here because it is transmitted by VSpy and not an EEVB node, so VSpy will automatically decode it in *Messages View*.

► **Switch to Application Signals.**

The first five signals here contain the five different headlines that can be requested. These are text values that you can change to whatever you wish. The remaining two signals carry the timestamp (which again can be manually modified) and a variable that is set to the value of the last headline requested by the function block script we are about to examine.

► **Switch to Function Blocks.**

The function block script for this setup can be found in Figure 123; it is long, but not hard to understand if you look at it in sections:

- **Steps 1 and 2:** Trigger the script and clear the *Present* flag to ensure it runs once per request.
- **Steps 3 to 8:** Set various values in the outgoing message based on comparable values in the message received.
- **Steps 9 to 13:** Check to see if either of the "special" requests have been received. If so, sets the outgoing text response field to the saved last headline value or the date/timestamp.
- **Steps 14 to 26:** Processes a standard headline request, checking which headline was asked for and then setting the text response field accordingly.
- **Step 27:** Transmits the response.

Step	Description	Value	Comment
1	Wait Until	{Headline Request (Present) :in4-0}	// Wait for receipt of a headline request message.
2	Set Value	{Headline Request (Present) :in4-0} = 0	// Clear present flag to avoid duplicate triggering.
3	Set Value	{Destination MAC Address (Value) :out2-sig0-0} = {Source MAC Address (Value) :in4-sig1-0}	// Set destination MAC address, IP address and port number to source values of incoming message.
4	Set Value	{Destination IP Address (Value) :out2-sig17-0} = {Source IP Address (Value) :in4-sig16-0}	
5	Set Value	{Destination Port (Value) :out2-sig19-0} = {Source Port (Value) :in4-sig18-0}	
6	Set Value	{Request Number (Value) :out2-sig25-0} = {Request Number (Value) :in4-sig25-0}	// Copy request number from source to outgoing message to enable matching of requests and responses.
7	Set Value	{Operation (Value) :out2-sig22-0} = {Operation (Value) :in4-sig22-0}	// Copy Operation and Headline Code for reference.
8	Set Value	{Headline Code (Value) :out2-sig23-0} = {Headline Code (Value) :in4-sig23-0}	
9	If	{Operation (Value) :in4-sig22-0} = 2	// Request is asking for the previously sent headline.
10	Set Value	{Response Text (Value) :out2-sig26-0} = {Last Headline :sig6-index(0)}	
11	Else If	{Operation (Value) :in4-sig22-0} = 3	// Request for last update timestamp.
12	Set Value	{Response Text (Value) :out2-sig26-0} = {Update Timestamp :sig7-index(0)}	
13	Else		// Normal headline request.
14	If	{Headline Code (Value) :in4-sig23-0} = 1	// Decide what to send back based on the Headline Code value.
15	Set Value	{Response Text (Value) :out2-sig26-0} = {Global News Headline :sig1-index(0)}	
16	Else If	{Headline Code (Value) :in4-sig23-0} = 2	
17	Set Value	{Response Text (Value) :out2-sig26-0} = {National News Headline :sig2-index(0)}	
18	Else If	{Headline Code (Value) :in4-sig23-0} = 3	
19	Set Value	{Response Text (Value) :out2-sig26-0} = {Local News Headline :sig3-index(0)}	
20	Else If	{Headline Code (Value) :in4-sig23-0} = 4	
21	Set Value	{Response Text (Value) :out2-sig26-0} = {Weather Headline :sig4-index(0)}	
22	Else		
23	Set Value	{Response Text (Value) :out2-sig26-0} = {Sports Headline :sig5-index(0)}	
24	End If		
25	Set Value	{Last Headline :sig6-index(0)} = {Response Text (Value) :out2-sig26-0}	// Save the headline we just selected in the Last Headline app signal so it is available if requested.
26	End If		
27	Transmit	Headline Response	// Send the response back to the client.

Figure 123: Headline Server Function Block Script.

Part 4.3D Run Headline Server Script

We will be running this script right within VSpy, so we don't need to send anything to the EEVB, just set up *Messages View* to optimally display the message exchange. *Details View* should already be off from earlier steps in this lab, but if not, please disable it so you can more easily see message details. Scroll mode should also be off for the same reason.

- ► **Switch to Messages View.**
- ► **Set Message Filter:** Enter Headline in the Description filter field, so we only see our messages.
- ► **Go Online.**

You should see two messages showing up every 5 seconds: headline requests and matching responses.

- ► **Expand the Headline Response Message to Show Signals.**

Now you can see the fields in the response messages, which will look like Figure 124. Notice that the copying of the request parameters to the output (*Operation* and *Headline Code*) makes it easy to see that the correct response is being sent to each request.

- ► **Turn the Node A Potentiometer.**

As you move the potentiometer, the *Headline Code* value will change, with the message in *Response Text* changing to match it.

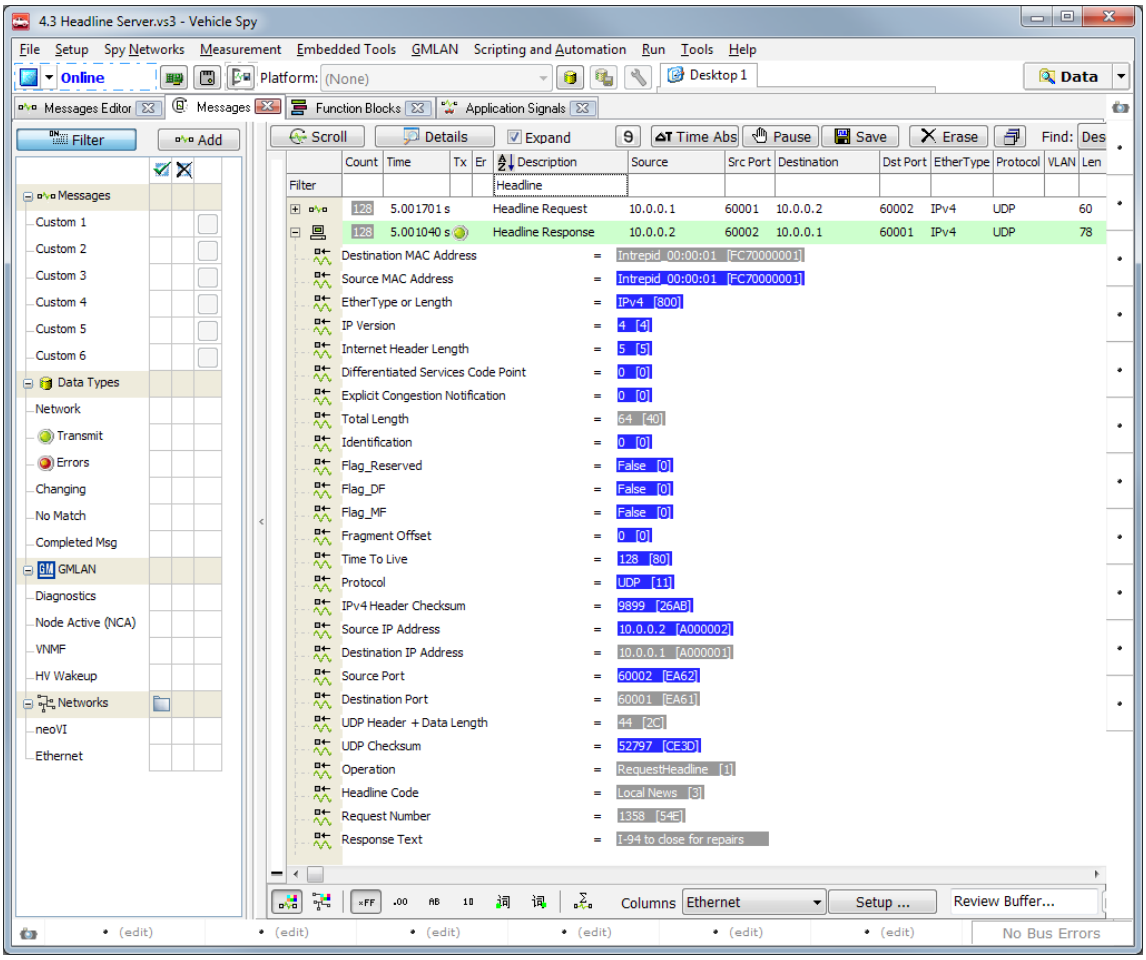


Figure 124: Headline Request Messages Sent by EEVB Node A.

- **Hold Down the Node A Pushbutton.**
- **Turn the Node A Potentiometer.**

While the button is held down you will see the *Operation* value alternate between **LastHeadline** and **RequestTimestamp**, and again, the *Response Text* field will change appropriately as well. Notice that changing the potentiometer has no effect while the pushbutton is depressed.

- **Release the Node A Pushbutton.**

Now the *Operation* value will go back to sending **RequestHeadline** again. If you changed the node potentiometer by enough to alter the requested *Headline Code*, the *Response Text* will change to match the new headline type requested.

We're now done with this lab, so let's reset things back to be closer to their usual configuration.

- **Turn Details view**
- **Go Offline.**

Lab 4.4 Simulating TCP Connection Establishment and Termination

The headline server we examined in Lab 4.3 is a great candidate for using UDP for two primary reasons: the messages exchanged are short and simple, and it doesn't really matter very much if they don't make it from source to destination. On occasion, a request might be lost, or it might be received but the response lost. In these cases the user would simply send the request again, and it would probably work fine, the lost messages representing merely a minor inconvenience.

Imagine instead, however, that we were trying to transfer a large file between two devices. Each file would need to be broken into smaller pieces for transmission, and each and every one of those pieces would need to be correctly received—and reassembled in the right order—for the file transfer to be considered successful. This is a job far better suited to TCP, because it keeps track of all of the data sent between two devices, ensuring that every byte transmitted by the sender is either received correctly, or is flagged for retransmission. TCP also contains facilities to manage large data flows such as might occur when moving files around.

As discussed in this section's introduction, the benefits that TCP provides come with a small cost: slightly lower performance (due to protocol overhead) and *significantly* greater complexity. In fact, TCP is so much more complex than UDP that it would be impractical for us to simulate its use with the EEVB. Implementing all of the intelligence necessary to manage TCP data exchanges would result in a setup file so large that it would be more confusing to you than helpful.

Instead, we have decided to focus on one important element of the operation of the TCP protocol: connections. We already saw an example of an actual TCP connection in Part 4.1E. Now we will discuss connections in more detail, and then show how they can be simulated using the Ethernet EVB.

An Overview of TCP Connections

Because TCP keeps track of data exchanges, it isn't possible to have one device just start sending data to another "out of the blue". The two devices must first establish a logical connection, which is used to let one device notify the other that it wishes to communicate, and to make any necessary preparations before data transmissions begin. This can be thought of as analogous to making a telephone call before beginning to speak. When either device no longer wants to communicate, it can terminate the connection, which is like hanging up the phone.

Both the client and server transition among various *states* as the connection is established, used and then torn down. The state describes the "status" of the connection as seen by each device, and changes as the connection evolves. In our simulation, EEVB Node A will play the role of the client, while the server script runs on Node B. Each has different states.

Connection Establishment

Both devices initially begin in the CLOSED state, and the process of connection establishment proceeds as follows:

E1. Server Waits for Incoming Connection Requests: The server prepares the necessary data structures for a connection and then transitions to the LISTEN state, where it sits waiting to be contacted by a client.

E2. Client Initiates Connection: The client transmits a TCP message with the *SYN* flag set, which indicates a desire to open a connection. This message contains the client's initial *Sequence Number* value, which is chosen randomly. As soon as this is sent, the client transitions to the SYN-SENT state.

E3. Server Receives, Processes and Acknowledges Connection Request: The server receives the client's *SYN* message, processes it, and responds with a *SYN+ACK* message. This message acknowledges the client's request and provides the client with the server's initial *Sequence Number* and possibly other parameters. The server transitions to the SYN-RECEIVED state.

E4. Client Receives, Processes and Acknowledges Server Acknowledgment: The client receives the server's *SYN+ACK* message and processes it. The client sends an *ACK* message back to the server and then transitions to the ESTABLISHED state.

E5. Server Receives and Processes Client Acknowledgment: The server receives the client's acknowledgment and transitions to the ESTABLISHED state.

The connection is now established and data can be sent by either device.

Since this process involves the exchange of three messages—SYN from client to server, SYN+ACK from server to client, and ACK from client back to server—it is known as a *three-way handshake*. The message exchanges and state transitions for both client and server are summarized graphically in Figure 125.

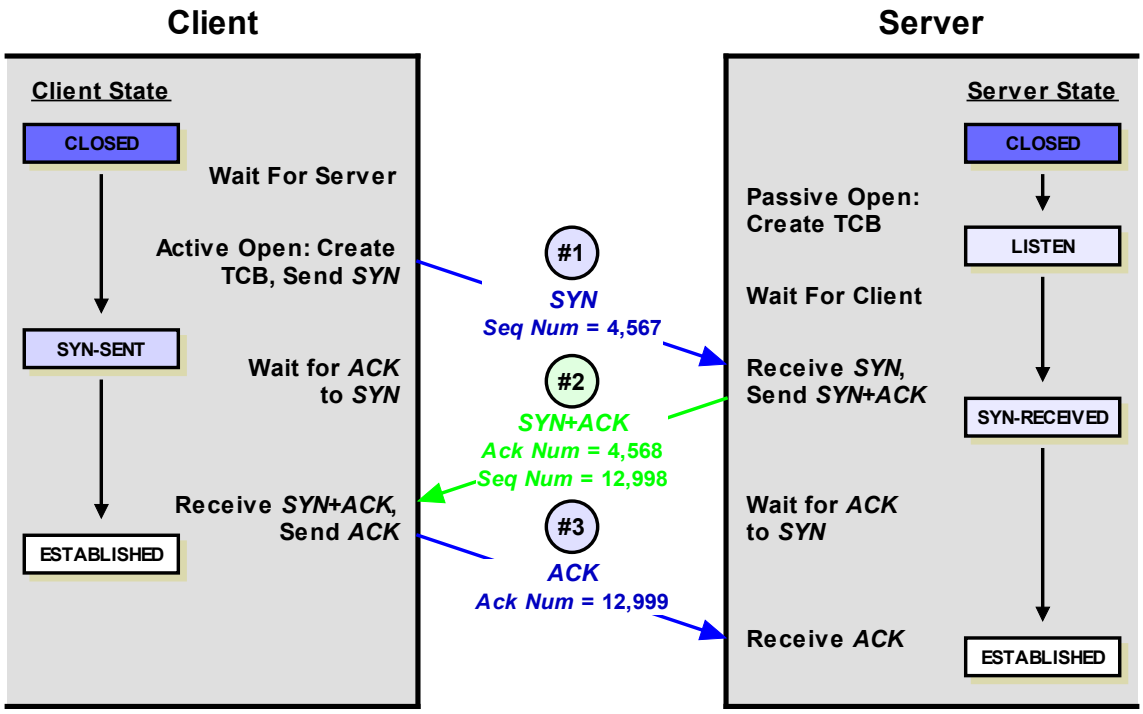


Figure 125: TCP Three-Way Handshake for Connection Establishment.

Connection Termination

A TCP connection can remain open for anywhere from a fraction of a second to many days, depending on the application. Either device can initiate connection termination when it decides it no longer wishes to communicate. We will assume here that the server initiates termination, perhaps because it has completed transferring all of the data it has available for the client. Here's what happens in this case:

T1. Server Sends Termination Request to Client: The server sends a TCP message with the *FIN* (for "finish") flag set. It transitions to the FIN-WAIT-1 state. (This message will usually also have the ACK flag set as well, acknowledging previously-received data, so it is sometimes called *FIN+ACK*.)

T2. Client Receives and Acknowledges Termination Request: The client receives the server's message and responds with an *ACK* message. The client transitions to the CLOSE-WAIT state.

T3. Server Receives Client Acknowledgment: The server receives the *ACK* and transitions to the FIN-WAIT-2 state. The server's side of the connection is now closed, but it must wait for the client before formally closing the connection.

T4. Client Waits for Application: The client TCP software waits for the application using the connection to signal that it is done and ready to close. (The server doesn't do this since its application is what triggered termination.)

T5. Client Sends Termination Request to Server: The client sends a *FIN* message, then transitions to the LAST-ACK state. (Again, this will often be a *FIN+ACK* message.)

T6. Server Receives and Acknowledges Termination Request: The server receives the *FIN* and responds with an *ACK*, then transitioning to the TIME-WAIT state.

T7. Client Receives Server Acknowledgment: The client receives the server's *ACK* and transitions to the CLOSED state.

T8. Server Waits and then Closes: The server waits for a specified time (set by the TCP implementation) to give the client time to receive its *ACK*. It then also transitions to the CLOSED state.

This is sometimes called a four-way handshake, though it's really a pair of two-way handshakes that occur in sequence. It is depicted in Figure 126.

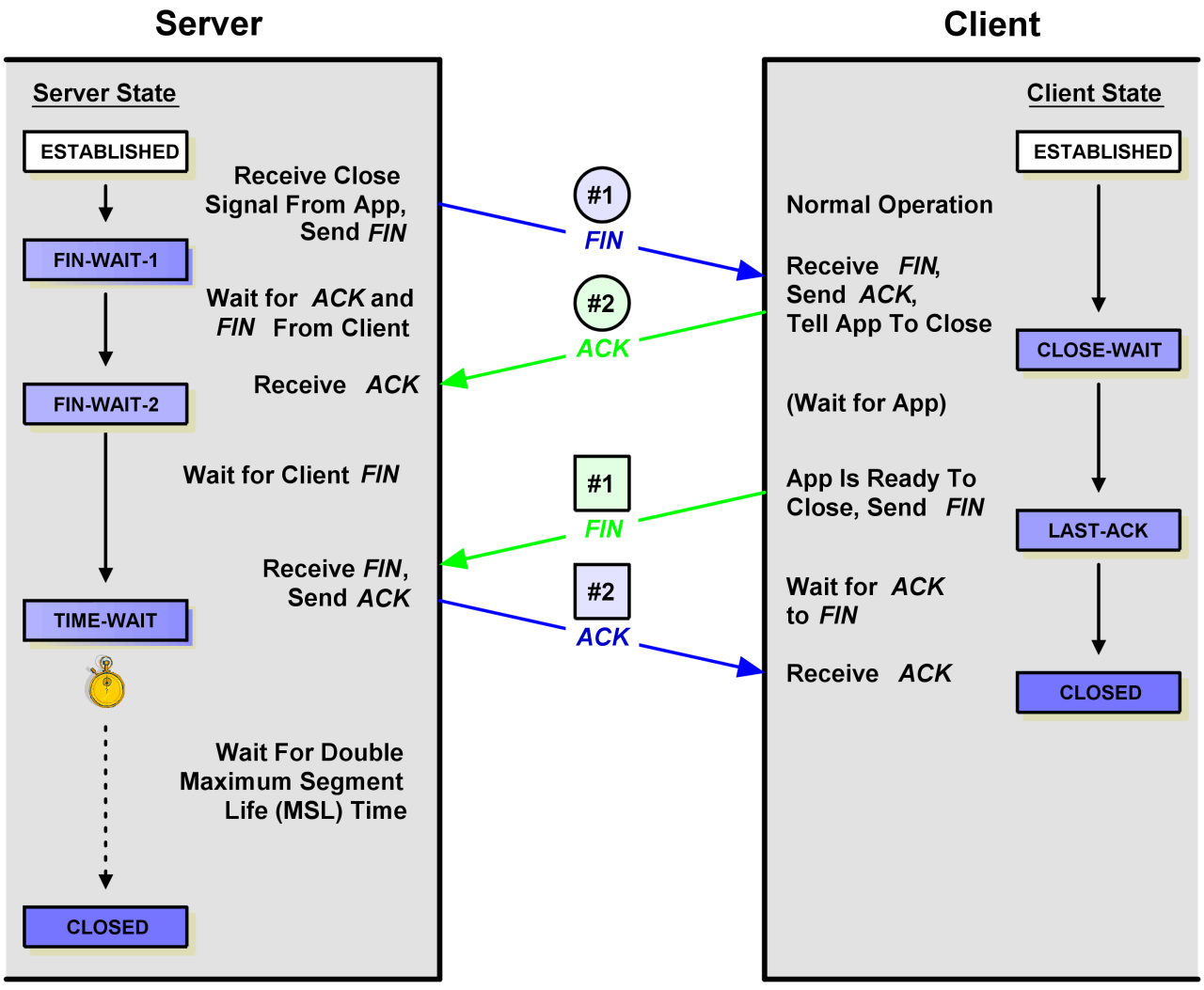


Figure 126: TCP Connection Termination.

Part 4.4A Load and Analyze TCP Connection Client Script

Let's start with the client script, which as mentioned earlier will run on EEVB Node A.

► **Load 4.4 TCP Handshake Client:** You can find this alongside the other lab example scripts on the *Logon Screen,* under the My Setups tab.

You will start in the *Messages Editor* on the transmit side, where you will see... a lot of messages! The first one in the list is the most important: it is a generic TCP message transmitted from the client to the server, and has no custom fields because we are only using control fields in the header for the handshaking process. The other 12 entries are "diagnostic messages" that we will use later in this lab to illustrate more completely how the handshaking works.

► **Switch to Receive Side.**

There is one receive message, which is the complement of the main transmit message. This is used to identify transmissions to the client from the server.

► **Switch to Application Signals.**

A number of application signals are used to control the operation of the client:

- **Client State:** The current status of the client, which can be CLOSED [0], SYN-SENT [1], ESTABLISHED [2], CLOSE-WAIT [3] or LAST-ACK [4].
- **Client Sequence Number:** The client's current *Sequence Number* field.
- **Server Acknowledgment Number:** The number from the last *Acknowledgment* field sent to the server.
- **Client Default Window Size:** The default value for the *Window Size* field, specifying the maximum amount of data the client can handle. This is an important parameter in TCP, but is not actually used in the simulation logic, so you can changed it to any value you like.
- **Connection Closed Timestamp:** Used to ensure a delay between closing the last connection and starting a new one.
- **Client Application Close Delay:** Specifies the number of seconds that the client waits in Step T4 before it sends its termination request.
- **Entered CLOSE-WAIT Timestamp:** Used to ensure the correct amount of time elapses in Step T8 before the client closes the connection.
- **LED Flash Counter:** Used to indicate visually the state of the client using an EEVB LED.

► **Switch to Function Blocks.**

There are six function block scripts in this setup. Five of these implement the client behavior described in the overview of connection establishment and termination. Each script controls the operation of the client in a particular state, and its name indicates the transition between states that occurs each time that script is run to completion. Here are these five scripts and the step numbers they implement:

- **Begin Initiation (CLOSED --> SYN-SENT):** Sends the *SYN* message to the server (Step E2).
- **Complete Initiation (SYN-SENT --> ESTABLISHED):** Sends *ACK* back to server (Step E4).
- **Respond to Server Termination (ESTABLISHED --> CLOSE-WAIT):** Receives and acknowledges the server's termination request (Step T2).
- **Begin Client Termination (CLOSE-WAIT --> LAST-ACK):** Waits for the (simulated) application to close and then starts the client side of connection termination (Steps T4 and T5).
- **Complete Client Termination (LAST-ACK --> CLOSED):** Receives acknowledgment of the client's request and closes the connection (Step T7).

The last script is a special indicator program that allows you to see the value of the Client State application signal. Every two seconds, this script flashes the node's LED1 a number of times equal to the value of that signal. The client will only remain in some states for a fraction of a second, but others are "stable". For example, you can use this to easily see if the client believes it is in the CLOSED state (value of 0) or the ESTABLISHED state (value of 2).

Let's send this setup to Node A of the EEVB.

► **Send CoreMini to Node A.**

As soon as you do this, both LEDs should go dark. The client stays in the CLOSED state until we initiate a connection, and it has a Client State value of 0; the LED indicator function block therefore doesn't flash LED1 at all.

We really need both the client and server halves running for this simulation to be meaningful, so we won't actually do anything else until we examine and download the server script as well.

Part 4.4B Load and Analyze TCP Connection Server Script

The server script is similar to the client one, but not exactly the same since you can see that both connection establishment and termination are asymmetric.

► **Load 4.4 TCP Handshake Server.**

Again we begin in the *Messages Editor* on the transmit side, and again we have a lot of messages. As with the client, the primary message is the first one, while 13 others are used for the optional diagnostic feature discussed later in the lab.

► **Switch to Receive Side.**

This script is the one we will leave running within VSpy to monitor the message transactions between the two EEVB nodes. Accordingly, we need copies of all of the transmit messages for both the server and client setups here, for a total of 27.

► **Switch to Application Signals.**

The application signals are similar to those used in the client script, though again there are differences due to the two sides not behaving in exactly the same way:

- **Server State:** The current logical status of the server: CLOSED [0], LISTEN [1], SYN-RECEIVED [2], ESTABLISHED [3], FIN-WAIT-1 [4], FIN-WAIT-2 [5] or TIME-WAIT [6] .
- **Server Sequence Number:** The server's current *Sequence Number* field value.
- **Client Acknowledgment Number:** The number from the last *Acknowledgment* field sent to the client.
- **Server Default Window Size:** The default value for the *Window Size* field, specifying the maximum amount of data the server can handle.
- **Server Connection Close Delay:** The number of seconds to wait between the TIME-WAIT and CLOSED states.
- **Client Application Close Delay:** How long the client waits in Step T4 before it sends its termination request.
- **Entered TIME-WAIT Timestamp:** Marks the time the TIME-WAIT state began so the correct wait duration is applied.
- **LED Flash Counter:** Used to indicate visually the state of the client using an EEVB LED.

► **Switch to Function Blocks.**

The function block scripts in this setup are crafted in much the same way as they were in the client setup. Again, each block performs the necessary operations for the server in a particular state before changing to a different one, and the names reflect the state transitions. There are more scripts than for the client because there are more server states. The last script shows the value of the Server State application signal in Node B LED1.

The scripts that define the operation of the server are as follows:

- **Prepare for Initiation (CLOSED --> LISTEN):** Gets the server ready to receive incoming connection requests (Step E1).
- **Respond to Initiation (LISTEN --> SYN-RECEIVED):** Responds to the server's *SYN* with a *SYN+ACK* (Step E3).

- **Complete Initiation (SYN-RECEIVED --> ESTABLISHED):** Receives the client's acknowledgment and establishes the connection (Step E5).
- **Begin Server Termination (ESTABLISHED --> FIN-WAIT-1):** Starts the process of terminating the connection from the server side (Step T1).
- **Complete Server Termination (FIN-WAIT-1 --> FIN-WAIT-2):** The server receives acknowledgment of its termination request, now waits for the client side to close (Step T3).
- **Respond to Client Termination (FIN-WAIT-2 --> TIME-WAIT):** Server responds to the client's termination request (Step T6).
- **Wait to Close Connection (TIME-WAIT --> CLOSED):** Server waits and then closes the connection (Step T8).

The last script is a status indicator as on the client. It will normally show the server in the CLOSED state (no flashes, value of 0), LISTEN state (1 flash) or ESTABLISHED state (3 flashes).

We'll now send this setup to EEVB Node B.

► **Send CoreMini to Node B.**

You should see this node's LED1 begin to flash once every two seconds shortly after the CoreMini is received. The server automatically transitions to the LISTEN state so it is ready to receive incoming connection requests from the client.

Part 4.4C Observe TCP Connection Establishment Handshake

Let's now run our simulation and see how it works.

First, recall those extra message definitions and our mention of a special diagnostic feature? We want to begin with that mode off as it is easier to start by looking at just the basic message exchange. The sending of these extra messages is controlled by the potentiometers on each node: they are triggered by the dial being in the lower half of its range (0 to 2047) and disabled if it is in its upper half (2048 to 4095).

► **Turn Both Node Potentiometers to Maximum Value:** Turn the dials clockwise until they are at or near their maximums.

Now let's get ready to observe the messages coming from the two nodes.

► **Switch to Messages View.**

► **Set Message Filter:** Enter TCP in the Description filter field to suppress messages we aren't interested in.

► **Turn Scroll Mode On (If Necessary).**

► **Ensure that Flags Column is Visible:** This may require widening the Vehicle Spy window or changing column widths.

► **Go Online.**

You won't see anything at first, because we must manually trigger the connection process.

► **Press the Node A Pushbutton.**

Three messages will appear very quickly, separated by only a few milliseconds (Figure 127). These are the messages in the connection establishment "three-way handshake": *SYN* from client to server, *SYN+ACK* from server to client and *ACK* from client to server, as seen in the *Flags* column. Note also the interaction of the *Seq#* and *Ack#* fields: the client selects a random sequence number in its *SYN* message, and the server acknowledges it by incrementing the value and sending it back in the *Ack#* field of its *SYN+ACK*. The same happens with the server's sequence number in the *SYN+ACK* and *ACK* messages.

The client and server are now both in the ESTABLISHED state; Node A's LED1 flashes twice, and Node B's LED2 flashes three times.

	Line	Time	Tx	Er	Description	Source	Src Port	Destination	Dst Port	EtherType	Protocol	VLAN	Len	Seq#	Ack#	Window	Flags	Network
Filter					TCP													
	1				TCP Segment - Client to Server	10.0.0.1	60001	10.0.0.2	60002	IPv4	TCP		60	172641	0	1024	SYN	Ethernet
	2	4.866 ms			TCP Segment - Server to Client	10.0.0.2	60002	10.0.0.1	60001	IPv4	TCP		60	169138	172642	2048	ACK,SYN	Ethernet
	3	3.110 ms			TCP Segment - Client to Server	10.0.0.1	60001	10.0.0.2	60002	IPv4	TCP		60	172642	169139	1024	ACK	Ethernet

Figure 127: TCP Connection Simulation Establishment Handshake.

Pay close attention to VSpy as we now close the connection.

► **Press the Node B Pushbutton.**

You should see four messages appear in two groups (messages 4 to 7 in Figure 128). The first is the server's connection termination exchange (Steps T1 to T3) and the second is the client's (Steps T5 to T7). Between these is the application close delay, which in this simulation is 3 seconds (Step T4); you can see this in the *Time* field for message 6 in the figure. The server wait time before closing the connection (Step T8) can sometimes be observed by watching the Node B LED, which should flash 6 times (TIME-WAIT state) at least once before the connection closes. The server will then quickly transition back to LISTEN so a new connection can be established again (one flash every two seconds).

	Line	Time	Tx	Er	Description	Source	Src Port	Destination	Dst Port	EtherType	Protocol	VLAN	Len	Seq#	Ack#	Window	Flags	Network
Filter					TCP													
	1				TCP Segment - Client to Server	10.0.0.1	60001	10.0.0.2	60002	IPv4	TCP		60	172641	0	1024	SYN	Ethernet
	2	4.866 ms			TCP Segment - Server to Client	10.0.0.2	60002	10.0.0.1	60001	IPv4	TCP		60	169138	172642	2048	ACK,SYN	Ethernet
	3	3.110 ms			TCP Segment - Client to Server	10.0.0.1	60001	10.0.0.2	60002	IPv4	TCP		60	172642	169139	1024	ACK	Ethernet
	4	26:06.372514			TCP Segment - Server to Client	10.0.0.2	60002	10.0.0.1	60001	IPv4	TCP		60	169139	172642	2048	ACK,FIN	Ethernet
	5	2.464 ms			TCP Segment - Client to Server	10.0.0.1	60001	10.0.0.2	60002	IPv4	TCP		60	172642	169140	1024	ACK	Ethernet
	6	3.003855 s			TCP Segment - Client to Server	10.0.0.1	60001	10.0.0.2	60002	IPv4	TCP		60	172642	169140	1024	ACK,FIN	Ethernet
	7	2.506 ms			TCP Segment - Server to Client	10.0.0.2	60002	10.0.0.1	60001	IPv4	TCP		60	169140	172643	2048	ACK	Ethernet

Figure 128: TCP Connection Simulation Establishment and Termination Messages.

> **Note:** It is possible in some cases to end up with the two nodes in incompatible states, which will cause the simulation to stop working properly. This can be corrected by pressing the reset buttons on both nodes, which will cause their scripts to start over again in the CLOSED state so they function correctly.

Part 4.4D Observe Handshake Details Using Diagnostic Messages

Okay, let's now come back to all those extra messages that we told you to ignore earlier in the lab. The purpose of these messages is to help you see exactly what is happening as a connection is created or torn down in the client, the server, or both. The extra messages are sent for each major occurrence within either device, such as receiving a message, transmitting a message, or commencing or completing a delay.

Let's start with the client.

- **Turn Node A Potentiometer to Minimum Value:** Turn the dial counter-clockwise until it is near the minimum.
- **Clear the Messages Display:** Press the Erase button.
- **Press the Node A Pushbutton.**

You will now see the same three-way connection establishment handshake as before, but with extra messages for each major action taken by the client.

- **Press the Node B Pushbutton.**

The termination closes, as before, and again extra messages appear tracing the client's behavior. The full exchange with the client's 12 diagnostic transmissions is in Figure 129.

	Line	Time	Tx	Er	Description	Source	Src Port	Destination	Dst Port	EtherType	Protocol	VLAN	Len	Seq#	Ack#	Window	Flags
Filter					TCP												
	1				TCP Client: Start connection establishment	10.0.0.1	60001	255.255.25...	1	IPv4	TCP		60	0	0	65535	
	2	2.998 ms			TCP Segment - Client to Server	10.0.0.1	60001	10.0.0.2	60002	IPv4	TCP		60	172641	0	1024	SYN
	3	1 µs			TCP Client: Sent SYN (--> SYN-SENT)	10.0.0.1	60001	255.255.25...	2	IPv4	TCP		60	0	0	65535	
	4	4.182 ms			TCP Segment - Server to Client	10.0.0.2	60002	10.0.0.1	60001	IPv4	TCP		60	669006	172642	2048	ACK,SYN
	5	1.789 ms			TCP Client: Received SYN+ACK	10.0.0.1	60001	255.255.25...	3	IPv4	TCP		60	0	0	65535	
	6	2.033 ms			TCP Segment - Client to Server	10.0.0.1	60001	10.0.0.2	60002	IPv4	TCP		60	172642	669007	1024	ACK
	7	996 µs			TCP Client: Sent ACK (--> ESTABLISHED)	10.0.0.1	60001	255.255.25...	4	IPv4	TCP		60	0	0	65535	
	8	80 µs			TCP Client: Connection established	10.0.0.1	60001	255.255.25...	5	IPv4	TCP		60	0	0	65535	
	9	2.592983 s			TCP Segment - Server to Client	10.0.0.2	60002	10.0.0.1	60001	IPv4	TCP		60	669007	172642	2048	ACK,FIN
	10	791 µs			TCP Client: Received FIN+ACK	10.0.0.1	60001	255.255.25...	6	IPv4	TCP		60	0	0	65535	
	11	2.990 ms			TCP Segment - Client to Server	10.0.0.1	60001	10.0.0.2	60002	IPv4	TCP		60	172642	669008	1024	ACK
	12	2 µs			TCP Client: Sent ACK (--> CLOSE-WAIT)	10.0.0.1	60001	255.255.25...	7	IPv4	TCP		60	0	0	65535	
	13	1.043 ms			TCP Client: Start delay for application close	10.0.0.1	60001	255.255.25...	8	IPv4	TCP		60	0	0	65535	
	14	2 µs			TCP Client: Start client connection termination	10.0.0.1	60001	255.255.25...	9	IPv4	TCP		60	0	0	65535	
	15	980 µs			TCP Segment - Client to Server	10.0.0.1	60001	10.0.0.2	60002	IPv4	TCP		60	172642	669008	1024	ACK,FIN
	16	968 µs			TCP Client: Sent FIN+ACK (--> LAST-ACK)	10.0.0.1	60001	255.255.25...	10	IPv4	TCP		60	0	0	65535	
	17	2.196 ms			TCP Segment - Server to Client	10.0.0.2	60002	10.0.0.1	60001	IPv4	TCP		60	669008	172643	2048	ACK
	18	865 µs			TCP Client: Received ACK	10.0.0.1	60001	255.255.25...	11	IPv4	TCP		60	0	0	65535	
	19	1.963 ms			TCP Client: Client connection closed (--> CLOSED)	10.0.0.1	60001	255.255.25...	12	IPv4	TCP		60	0	0	65535	

Figure 129: TCP Connection Handshaking with Extra Client Status Messages.

Now let's repeat this to show the server side of the transactions.

- ► **Turn Node A Potentiometer to Maximum Value.**
- ► **Turn Node B Potentiometer to Minimum Value.**
- ► **Clear the Messages Display:** Press the Erase button.
- ► **Press the Node A Pushbutton.**
- ► **Press the Node B Pushbutton.**

The results can be seen in Figure 130.

	Line	Time	Tx	Er	Description	Source	Src Port	Destination	Dst Port	EtherType	Protocol	VLAN	Len	Seq#	Ack#	Window	Flags
Filter					TCP												
⊞	1				TCP Segment - Client to Server	10.0.0.1	60001	10.0.0.2	60002	IPv4	TCP		60	43161	0	1024	SYN
⊞	2	1.060 ms			TCP Server: Received SYN	10.0.0.2	60002	255.255.25...	1002	IPv4	TCP		60	0	0	65535	
⊞	3	3.065 ms			TCP Segment - Server to Client	10.0.0.2	60002	10.0.0.1	60001	IPv4	TCP		60	667252	43162	2048	ACK,SYN
⊞	4	929 µs			TCP Server: Sent SYN+ACK (--> SYN-RECEIVED)	10.0.0.2	60002	255.255.25...	1003	IPv4	TCP		60	0	0	65535	
⊞	5	2.936 ms			TCP Segment - Client to Server	10.0.0.1	60001	10.0.0.2	60002	IPv4	TCP		60	43162	667253	1024	ACK
⊞	6	95 µs			TCP Server: Received ACK (--> ESTABLISHED)	10.0.0.2	60002	255.255.25...	1004	IPv4	TCP		60	0	0	65535	
⊞	7	1.992 ms			TCP Server: Connection established	10.0.0.2	60002	255.255.25...	1005	IPv4	TCP		60	0	0	65535	
⊞	8	3.331839 s			TCP Server: Start server connection termination	10.0.0.2	60002	255.255.25...	1006	IPv4	TCP		60	0	0	65535	
⊞	9	1.957 ms			TCP Segment - Server to Client	10.0.0.2	60002	10.0.0.1	60001	IPv4	TCP		60	667253	43162	2048	ACK,FIN
⊞	10	993 µs			TCP Server: Sent FIN+ACK (--> FIN-WAIT-1)	10.0.0.2	60002	255.255.25...	1007	IPv4	TCP		60	0	0	65535	
⊞	11	1.925 ms			TCP Segment - Client to Server	10.0.0.1	60001	10.0.0.2	60002	IPv4	TCP		60	43162	667254	1024	ACK
⊞	12	1.076 ms			TCP Server: Received ACK (--> FIN-WAIT-2)	10.0.0.2	60002	255.255.25...	1008	IPv4	TCP		60	0	0	65535	
⊞	13	3.001760 s			TCP Segment - Client to Server	10.0.0.1	60001	10.0.0.2	60002	IPv4	TCP		60	43162	667254	1024	ACK,FIN
⊞	14	111 µs			TCP Server: Received FIN+ACK	10.0.0.2	60002	255.255.25...	1009	IPv4	TCP		60	0	0	65535	
⊞	15	3.003 ms			TCP Segment - Server to Client	10.0.0.2	60002	10.0.0.1	60001	IPv4	TCP		60	667254	43163	2048	ACK
⊞	16	960 µs			TCP Server: Sent ACK (--> TIME-WAIT)	10.0.0.2	60002	255.255.25...	1010	IPv4	TCP		60	0	0	65535	
⊞	17	66 µs			TCP Server: Start server close delay	10.0.0.2	60002	255.255.25...	1011	IPv4	TCP		60	0	0	65535	
⊞	18	2.961 ms			TCP Server: Server close delay completed	10.0.0.2	60002	255.255.25...	1012	IPv4	TCP		60	0	0	65535	
⊞	19	1.970 ms			TCP Server: Server connection closed (--> CLOSED)	10.0.0.2	60002	255.255.25...	1013	IPv4	TCP		60	0	0	65535	
⊞	20	2.002 ms			TCP Server: Passive open (--> LISTEN)	10.0.0.2	60002	255.255.25...	1001	IPv4	TCP		60	0	0	65535	

Figure 130: TCP Connection Handshaking with Extra Server Status Messages.

You can of course turn diagnostics on for both nodes, though the result is a lot of client and server messages intermixed, which is hard to decipher.

- ► **Go Offline.**

Congratulations, you've completed the Intrepid Ethernet EVB Lab Manual!

