Using LabVIEW with TCP/IP and UDP

Overview

Internet Protocol (IP), User Datagram Protocol (UDP), and Transmission Control Protocol (TCP) are the basic tools for network communication. The name TCP/IP comes from two of the best-known protocols of the internet protocol suite, the Transmission Control Protocol and the Internet Protocol.

With TCP/IP you can communicate over single networks or interconnected networks (Internet). The individual networks can be separated by large geographical distances. TCP/IP routes data from one network or Internet computer to another. Because TCP/IP is available on most computers, it can transfer information between diverse systems.

LabVIEW and TCP/IP

You can use the TCP/IP protocols with LabVIEW on all platforms. LabVIEW includes TCP and UDP functions for creating client or server VIs.

IP

IP performs the low-level service of moving data between computers. IP packages data into components called datagrams. A datagram contains, among other things, the data and a header that indicates the source and destination addresses. IP determines the correct path for the datagram to take across the network or internet and sends the data to the specified destination.

IP cannot guarantee delivery. In fact, IP may deliver a single datagram more than once if the datagram is duplicated in transmission. Programs rarely use IP because they use TCP or UDP instead.

UDP

UDP provides simple, low-level communication between processes on computers. Processes communicate by sending datagrams to a destination computer or port. A port is the location where you send data. IP handles the computer-to-computer delivery. Once the datagram reaches the destination computer, UDP moves the datagram to its destination port. If the destination port is not open, UDP discards the datagram. UDP shares all the delivery problems of IP.

Use UDP in applications where reliability is not critical. For example, an application might transmit informative data to a destination frequently enough that a few lost segments of data are not problematic.

Using UDP

Because UDP is not a connection-based protocol as TCP is, you do not need to establish a connection with a destination before you send or receive data. Instead, you specify the destination for the data when you send each datagram. Operating systems do not report transmission errors.

Use the UDP Open function to open a port. The number of simultaneously open UDP ports depends on the operating system. UDP Open returns a UDP Network Connection refnum, which is an opaque token used in all subsequent operations that pertain to that port.

Use the UDP Write function to send data to a destination and use the UDP Read function to read that data. Each write operation requires a destination address and port. Each read operation contains the source address and port. UDP preserves the datagram bytes that you specified for each command you send.

In theory, you can send datagrams of any size; however, you would not typically use UDP to send large datagrams because it is not as reliable as TCP.

When you finish all communications on a port, use the UDP Close function to free system resources.

TCP

TCP ensures reliable transmission across networks, delivering data in sequence without errors, loss, or duplication. TCP will retransmit the datagram until it receives an acknowledgment.

Setup

Before using TCP/IP, confirm that you have the correct setup, which varies depending on the platform you use.

- **Windows UNIX** TCP/IP is built in. You do not need to use a third-party product to communicate using TCP/IP. Assuming your network is configured properly, no additional setup for LabVIEW is necessary.
- **Macintosh** LabVIEW networking requires Open Transport, included in Macintosh OS 7.5 and later. LabVIEW no longer supports Mac TCP.
- **UNIX** TCP/IP is built in. Assuming your network is configured properly, no additional setup for LabVIEW is necessary.

Using TCP

TCP is a connection-based protocol, which means that sites must establish a connection before transferring data. TCP permits multiple, simultaneous connections.

You initiate a connection either by waiting for an incoming connection or by actively seeking a connection with a specified address. In establishing TCP connections, you have to specify the address and a port at that address. A number between 0 and 65,535 represents a port. UNIX reserves port numbers less than 1,024 for privileged applications. Different ports at a given address identify different services at that address.

Use the TCP Open Connection function to actively establish a connection with a specific address and port. If the connection is successful, the function returns a TCP Network Connection refnum that uniquely identifies that connection. Use this connection ID to refer to the connection in subsequent function calls.

You can use the following methods to wait for an incoming connection:

- Use the TCP Listen VI to create a listener and wait for an accepted TCP connection at a specified port. If the connection is successful, the VI returns a connection ID and the address and port of the remote TCP.
- Use the TCP Create Listener function to create a listener and then use the Wait on Listener function to listen for and accept new connections. Wait on Listener returns the same listener ID that was passed to the function and the connection ID for a connection. When you finish waiting for new connections, use TCP Close to close a listener. You cannot read from or write to a listener.

The advantage of using the second method is that you can cancel a listen operation by using TCP Close. This is useful when you want to listen for a connection without using a timeout, but you want to cancel the listen when another condition becomes true. You can close the listen VI at anytime.

When you establish a connection, use the TCP Read and TCP Write functions to read and write data to the remote application.

Use the TCP Close Connection function to close the connection to the remote application. If unread data remains and the connection closes, that data may be lost. Use a higher level protocol for your computer to determine when to close the connection. Once a connection is closed, you cannot read from it or write to it again.

TCP Versus UDP

TCP is the best method to use if you want reliable data transmission. UDP is a connectionless protocol with higher performance, but it does not ensure reliable data transmission.

TCP Client Example

The following steps are a generalized description of how to create a TCP client using functions from the **Communication»TCP** palette.

1. Use the TCP Open Connection function to open a connection to a server. You must specify the internet address of the server and the port for the server.

   The address identifies a computer on the network. The port identifies a communication channel on the computer that the server uses to listen for communication requests. When you create a TCP server, you specify the port that you want the server to use for communication.
2. Use the TCP Write function to send the command to a server.
3. Use the TCP Read function to read results from the server.

   With the TCP Read function, you must specify the number of characters you want to read.

   Use the following methods to handle different sized commands:

   - Precede the command and the result with a fixed **size** parameter that specifies the size of the command or result. In this case, read the **size** parameter and then read the number of characters the **size** parameter specifies. This method is the most flexible.
   - Make each command and result a fixed size. When a command is smaller than the size you specify, you can pad it to the fixed size. This option is the most efficient.
   - Follow each command and result with a specific terminating character. Read data in small units until you reach the terminating character.
4. Use the TCP Close Connection function to close the connection to the server.

Timeouts and Errors

When you design a network application, consider carefully what should happen if something fails. For example, if the server crashes, how would each of the client VIs handle it?

One solution is to make sure that each VI has a timeout. If something fails to produce results, after a certain amount of time the client continues execution. In continuing, the client can try to reestablish execution or report the error. If necessary, it can shut down the client application.

TCP Server Example

The following steps area generalized description of how to create a TCP server using functions from the **Communication»TCP** palette.

1. Use the TCP Listen VI to wait for a connection. You must specify the port. This port must be the same port that the client attempts to access. Refer to the *TCP Client Example* section for more information about using the TCP Listen VI.
2. If a connection is established, read from that port to retrieve a command. As discussed in the *TCP Client Example* section, you must decide the format for commands. If a size parameter precedes commands, read the length field first and then read the amount of data the length indicates are remaining.
3. Use the TCP Write function to return results. As discussed in the *TCP Client Example* section, the data must be in a form that the client can accept.
4. Use the TCP Close Connection function to close the connection.

342028A-01 Jul00