CHAPTER 3

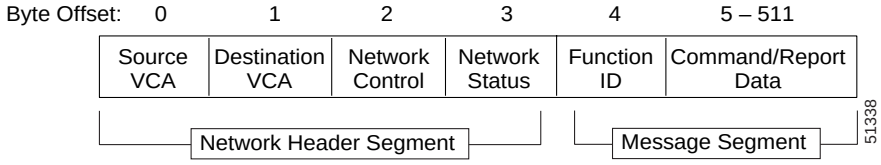
Host Communications

This chapter provides an overview of the Ethernet implementation used by the VCO/4K for communications with host applications. (Administration and NFS communications are described in Chapter 2, “Ethernet Installation and Configuration.”)

VCO/4K System Message Structure

Depending on the command or report, a VCO/4K system message can contain 6 to 512 bytes of information. Figure 3-1 illustrates the system message structure.

Figure 3-1 System Message Structure



Each system message consists of a network header segment and a message segment. The network header segment contains routing, control, and processing status information used by application processes in the host and system. The message segment contains the system command or report. For more information about these segments, refer to the *Cisco VCO/4K Extended Programming Reference* or *Cisco VCO/4K Standard Programming Reference*.

Note The following special considerations pertain when using Ethernet for system host communication:

Because of the Transmission Control Protocol (TCP) stream orientation, all VCO Commands and Reports must be preceded by a 2-byte length count.

Messages are transmitted in network byte order (also called Big Endian), with the most significant byte (MSB) being transferred first. This is opposite from the byte ordering expected by most PC Ethernet implementations.

Communications Interfaces

TCP/IP protocol standards do not require a specific interface between the application programs and TCP/IP; therefore, the interface may vary from system to system. Because the TCP/IP protocol software resides in the operating system, the operating system must determine the interface requirements.

The VCO/4K supports connections established by the host operating system's socket interface. Socket interfaces typically provide host applications with library routines for network functions, including:

- Manipulation of IP Addresses
- Accessing the domain name system
- Performing network byte order conversion
- Obtaining host, network, protocol, and network service information

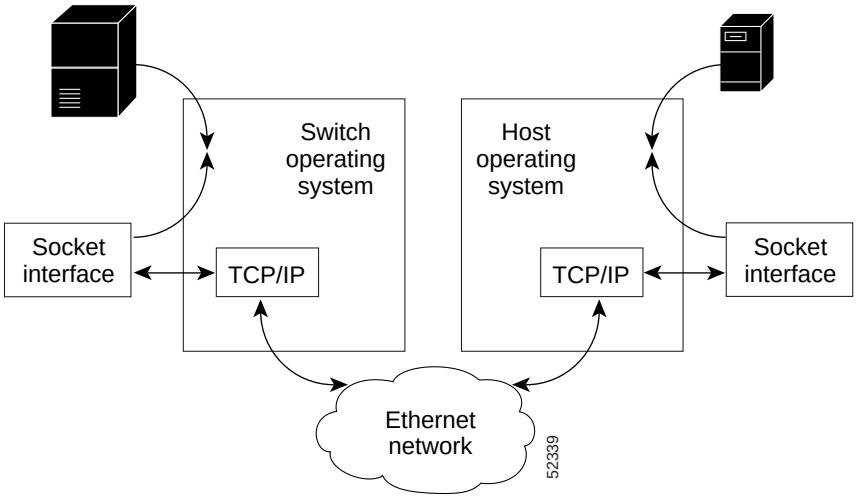
The socket interface provides the structure for the application's communication link with the VCO/4K. This interface controls the system sockets that provide end-points for TCP/IP's Transport Layer communications. All TCP communication links are active socket-to-passive socket connections:

- TCP establishes a connection between the active socket of a host and a passive socket of the VCO server application.
- Up to eight host-link, passive sockets can be configured for the single cable connection (through the CPU-TM's DB-15 port) to the Ethernet network, each host connection with its own port number.

Note In addition to the host application socket connections, the VCO/4K also supports TCP/IP connections with a host computer through the Telnet protocol.

Figure 3-2 shows a conceptual view of the communication path between the VCO/4K and a controlling host computer.

Figure 3-2 Socket Interface Conceptual Diagram



Socket Connections and Host Programming

Sockets are logical connections that provide the end points for all Transport Layer protocol communication links. For TCP transport protocol socket-to-socket connections, sockets are either passive (i.e., located at the server end of the connection) or active (i.e., located at the client end of the connection). The VCO/4K is a server in all of its TCP socket-to-socket links and uses only passive sockets for those links. A passive socket cannot initiate a connection; it uses a Listen state to inform TCP when it is ready to connect with the active socket of the client application.

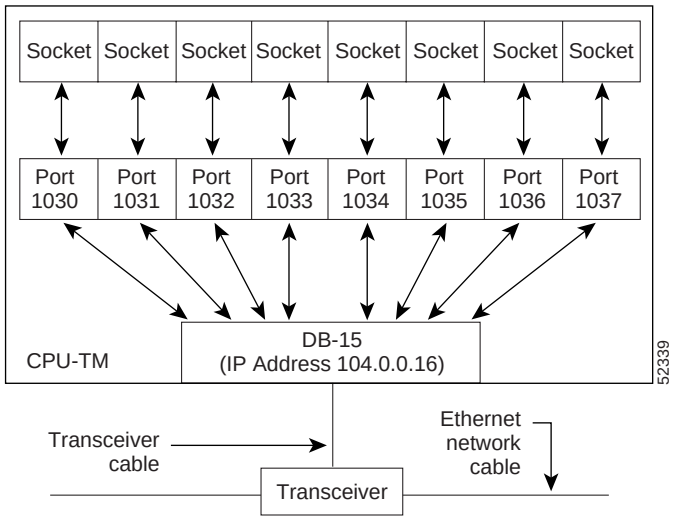
The VCO/4K uses the following types of sockets for its Ethernet network communication links:

- Stream sockets—Used with the TCP Transport Layer's links, with a host, for command and report data communications. The VCO/4K also supports the Telnet protocol.
- Datagram sockets—Used with the UDP Transport Layer's NFS protocol links with a file server.

Figure 3-3 shows a conceptualized view of the system host-link socket connections to the Ethernet network through the CPU-TM DB-15 port. These host-link stream sockets are bound to TCP port address values which, in turn, are bound to the specific IP address value for the CPU-TM's DB-15 Ethernet network connection. The TCP port and the IP address values together define a host connection.

Note The port and IP address values shown in Figure 3-3 are for illustration purposes only. The actual address values differ from system to system.

Figure 3-3 Multiple Socket Connection to an Ethernet Network



Socket Interface Library Routines

Library routines can be bound into an application program for any of the following functions:

- Host information access
- Network information access
- Network service information access
- Protocol information access

- Network byte order conversion
- IP address manipulation
- Domain system access

Host Socket Application Example

The following sections discuss a pseudocode example that creates a C-socket application for a VCO/4K connected to a UNIX-based host. Other operating systems use similar Ethernet system calls; therefore you can modify this example for different Ethernet hosts.

Opening a Connection with the VCO/4K

The VCO/4K provides the option to require a host password before establishing a socket with the host. For more information about the enabling a host password, refer to the *Cisco VCO/4K System Administrator's Guide*. The following example shows how to open a socket with a password.

Note If password connection fails, the VCO/4K issues an error message and closes the socket. Subsequent host attempts to write a message to the socket are rejected.

```
VCO_Open
(VCO_Name,VCO_Port,Local_Port)
{

        VCO_Entry = gethostbyname (VCO_Name)
        Connected = FALSE
        while Not Connected
        {
                Create a socket using 'socket()'
                Connect to the VCO using address from VCO_Entry and
                VCO_Port
                if error
                {
                        Close the socket.
                        if( error !=ETIMEDOUT && error != EADDRINUSE )
                                return the error
                        else
                                Wait and try again.
                }
                else
                {
                        if doing password verification
```

```
                    {
                        Receive “Password: “ string using ’recv()’
                        note: no message length sent by VCO
                        if error
                            return error
                        Send Password as specified in
                        VCO Host Config
                        using ’send()’ note:do not send
                        message length
                    }
                }
                Connected = TRUE
            }
            Save opened socket descriptor for program usage.
            return success
```

Reading from a Socket Connected to a System

Single UNIX **recv()** calls can read a single system message, part of a system message, or multiple system messages. The following pseudocode illustrates how to handle each of these situations.

Note This example assumes a variable called *Rx_Stream* that holds data already read from the socket, but not yet processed by the application.

```
VCO_Recv (message)
{

    if data in Rx_Stream
    {
        Get length of next message, first 2 bytes of data –
        Convert to machine byte order as required.

        if all of next message is in Rx_Stream
        {
            Copy next message from Rx_Stream to message
            parameter

            Remove message from Rx_Stream

            return message length.
```

```
                }

        }

        Append data to Rx_Stream using'recv()' call.

        if error
                return error.

        return VCO_Recv(message) – Recursive call
        to parse stream data.

}
```

Writing a Message on a System Socket

The following pseudocode illustrates the process of writing a message on a system socket.

```
VCO_Write(message,message_len)
{
        Convert message length to network byte order.

        Transmit message length using 'send()'.

        if error
                return error.

        Transmit message using 'send()'.

        if error
                return error.

        return success.
}
```

System Socket—Host Application Structure

The following example illustrates the basic software model for a host application, based on sockets using the subroutines created in the previous examples.

```
main()
{
        VCO_Open("VCO_1",2000,2001)

        Forever

        {

                Connected = TRUE

                while Connected
                {
                        VCO_Recv(message)

                        if no errors
                        {

                                Application specific processing.

                                if need to respond to message
                                {
                                        VCO_Write(response,response_length)
                                        if error
                                                Connected = FALSE
                                }
                        }
                        else
                                Connected = FALSE
                }

                Close VCO socket.
                Reopen the socket, VCO_Open("VCO_1",2000,2001)
        }
}
```

Reestablishing Host Connections

To minimize delays when reconnecting to a VCO socket after a connection has been torn down, do not use the “bind” function in your host application code. In addition, perform the following steps.

Step 1 From the Master Console, type **C** from the System Configuration menu to access the Host Configuration screen.

Step 2 Set the Rem Port value to 0.

Step 3 Set the Reset Time value to 1 second.

System Error Messages

Error messages are written to the VCO/4K system log file. For more information and a list of all error messages, refer to *Cisco VCO/4K System Messages.*