LBI-39161

User's Manual

EDACS®
Network Driver

ERICSSON

This publication and the described software are supplied "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose.

This publication may change, and the software described may also improve or change, without prior notice given.

Neither the documentation nor the software described may be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form, except in the manner described in the document.

TRADEMARKS

MS-DOS is a trademark of Microsoft Corporation
Windows is a trademark of Microsoft Corporation
IBM is a trademark of International Business Machines Corporation.
Network Driver and EDACS are trademarks of Ericsson, Inc.
All other company, brand, or product names used are trademarks or registered trademarks of their respective companies.

PREFACE

The EDACS Network Driver is a software device driver intended for application programmers developing mobile data applications for the EDACS radio network. It was designed to work with commercial TCP/IP protocol packages conforming with the NDIS Specification, version 2.01. Users of this document should be familiar with TCP/IP protocol stacks, installing networking software on personal computers, and networking concepts in general.

HARDWARE AND SOFTWARE REQUIREMENTS

To use the EDACS Network Driver, a data terminal requires the hardware and software listed below.

For a DOS based application:

- An IBM compatible personal computer with a minimum 80386SX/DX processor running at 20 MHz or better.
- A 720 KB or 1.44 MB 3.5 inch diskette drive
- A hard disk drive
- One serial port capable of 9600 baud operation (COM1 or COM2)
- MS-DOS version 3.3 or higher

For a Windows based application:

- An IBM compatible personal computer with a minimum 80486SX/DX processor running at 33 MHz or better
- A 720 KB or 1.44 MB 3.5 inch diskette drive
- A hard disk drive
- One serial port capable of 9600 baud operation (COM1 or COM2)
- MS-DOS version 3.3 or higher
- Microsoft Windows version 3.0 or higher

EDACS SYSTEM REQUIREMENTS

The EDACS Network Driver is designed to work solely in an EDACS land line data network. It requires an EDACS Data Gateway (EDG) with an IP host interface. See the EDACS Data Gateway Configuration Reference Manual (LBI-38964) for more information on how to setup the EDG for use with the EDACS Network Driver.

TCP/IP PROTOCOL STACK COMPATIBILITY

Extensive testing was done with a number of commercial protocol stacks in an effort to identify those packages that work well over a wireless network such as EDACS. Three packages are listed below as being compatible with Network Driver and EDACS. This manual also includes information on how to configure and install these stacks to work with Network Driver. No attempt was made to test Network Driver with any version of these packages other than what is listed.

- Wollongong Pathway Runtime and Access v2.0

  The Wollongong Group, Inc.
  1129 San Antonio Road
  P.O. Box 51860
  Palo Alto, California 94303
  TEL: (415) 962-7100 FAX: (415) 969-5547

- Distinct TCP/IP for Windows v3.21
  Distinct Corporation
  12901 Saratoga Ave., Suite 4
  P.O. Box 3410
  Saratoga, CA 95070-1410
  TEL: (408) 366-8933 FAX: (408) 366-0153

- Super TCP/NFS for Windows v4.00 R2
  Frontier Technologies Corporation
  10201 N. Port Washington Rd.
  Mequon, WI 53092
  TEL: (414) 241-4555 FAX: (414) 241-7084

RELATED DOCUMENTATION

The following documents are helpful in understanding the installation and/or operation of the Network Driver.

EDACS Data Gateway Technical Description (LBI-38961)
A detailed description of the EDG capabilities, interfaces, and hardware.

EDACS Data Gateway Configuration Reference Manual (LBI-38964)
A manual describing how to configure the EDG operation and interfaces.

Radio Data Interface Protocol Specification, Version 1.92 (ECX 922)
The protocol specification document for the RDI interface.

Internetworking with TCP/IP, Volume 1, by Douglas E. Comer
A excellent (but unofficial) source of information about Internet Protocol.

GLOSSARY OF TERMS

API

Application Programming Interface - A specification (usually for a specific programming language) or the interface between an application program and a system control program.

ARP

Address Resolution Protocol - A protocol scheme in the IP network definition that allows a host to determine the address of another host through a query process.

IP

Internet Protocol - A connectionless network protocol designed for the ARPA (**A**dvanced **R**esearch **P**rojects **A**gency) network.

MTU

Maximum Transmission Unit - The largest sized data packet that can be sent in each transmission.

Network Layer

Network Layer - The layer, hardware or software, in a protocol stack responsible for controlling the operation of subnets in a network. The network layer is concerned with the routing of data throughout a network as well as congestion.

OSI Model

Open Systems Interconnection Model - A seven layer representation of a network proposed by the International Standards Organization as a model for open communications between two systems.

Protocol Stack

Protocol Stack - A collection of networking protocol layers. TCP/IP is a protocol stack consisting of the TCP transport layer and the IP network layer

RARP

Reverse Address Resolution Protocol - A protocol that allows a machine to determine its IP address at startup. Usually used by diskless machines, the machine sends its hardware address to a server who returns the machine's IP address.

RDI

Radio Data Interface - The Ericsson proprietary hardware and associated message protocol that allows the connection of an RS-232 port to an EDACS radio. The RDI may be a separate component, or may be integrated into the radio itself.

RDT

Remote Data Terminal - An end user device used to communicate with a host data application, through data messages, over a wireless network. This can be a fixed or mobile device.

Subnet

Subnet - A communication interface, consisting of a transmission media and a communication protocol, between entities on a network.

TCP

Transmission Control Protocol - A connection oriented transport protocol used on an IP network. TCP provides a reliable, full duplex stream service.

Transport Layer

Transport Layer - A layer in a protocol hierarchy that provides reliable data transmission from one system to another independent of the network protocol used.

UART

Universal Asynchronous Receiver Transmitter - A hardware device capable of receiving serial data, transposing it to parallel data and transmitting it. It is also capable of receiving parallel data and transmitting it as a serial bit stream.

UDP

User Datagram Protocol - A standard IP protocol that allows two application programs to exchange data through the IP network. UDP uses a datagram structure that includes a protocol number. This allows the sender to direct data to multiple destinations on the same machine.

TABLE OF CONTENTS

TABLE OF CONTENTS (con't)

INTRODUCTION

The EDACS Network Driver is a **M**edium **A**ccess **C**ontrol (MAC) sublayer driver for PCs running MS-DOS. It complies with the Network Driver Interface Specification (NDIS), version 2.01, and manages the communications to EDACS for the mobile data application. Because it is an NDIS driver, Network Driver can be used with "off-the-shelf" protocol stacks written for the PC Ethernet LAN environment.

NETWORK DRIVER AND THE PROTOCOL STACK

In the NDIS structure, a MAC driver is the lowest layer in the protocol stack and directly controls the network hardware. The MAC driver communicates with a protocol driver above it. In terms of the OSI model, a MAC driver is a sublayer within the Data Link Layer.

Network Driver is an NDIS compatible MAC driver which handles all of the communications with the RDI and the EDACS network. Network Driver advertises itself as an Ethernet MAC driver to provide compatibility with commercial TCP/IP protocol stacks. This means that Network Driver communicates with the upper protocol layer through Ethernet frames. Network Driver performs the necessary protocol frame translations between Ethernet and RDI.

NETWORK DRIVER AND THE EDACS NETWORK LAYER

The EDG, with an IP host interface, and Network Driver work together to provide "IP like" network layer services to the data application. This is done by encapsulating messages inside the EDACS Network Layer frame. Both the Network Driver and the EDG handle the translations between the IP frames received from the data application and the EDACS Network Layer frames used over the EDACS network. Since these translations are hidden in Network Driver and the EDG, the EDACS data network appears to the application like an IP network.

Through the EDACS Network Layer, the Network Driver and the EDG together provide the necessary fragmentation support so that higher layer protocols (TCP or UDP) can send messages larger than the 512 byte MTU for EDACS. The EDACS Network Layer also allows mobile-to-mobile and mobile-to-group message routing through the EDG.

MEMORY USAGE

Network Driver is a DOS device driver that DOS loads into memory at initialization. Although it uses approximately 33K bytes of the system memory below the 640K byte DOS barrier, Network Driver can be loaded into the DOS high memory area using any of the popular memory managers. This frees the lower memory area for the application.

PERFORMANCE CONSIDERATIONS

When developing an application program, consideration must be given to the speed of the processor platform and the interrupt latency introduced by the application. Network Driver services the serial port connected to the RDI which runs at 9600 baud. In addition, Network Driver makes use of the timer interrupt to manage the interbyte time-outs required by the RDI protocol. Applications that also use the timer or disable interrupts must take care that the interrupt latency of the serial interrupt is less than one character time, (1 millisecond at 9600 baud) or Network Driver will miss characters.

When using Network Driver with a Windows application in enhanced mode, the interrupt latency becomes more critical. Because Network Driver is a DOS device driver rather than a Windows virtual device (VxD) driver, Windows considers the serial port interrupt a global interrupt. When the serial port generates an interrupt, Windows allows any protected mode handlers (VxD drivers) to try and service it first. Windows then switches the processor to real mode and calls the Network Driver interrupt handler. Windows goes through the same process for the timer interrupt which Network Driver uses. The overhead that Windows adds to each interrupt cycle can increase the interrupt latency to where Network Driver occasionally misses characters when running on slower computers. Network Driver requires an 80486 based computer for Windows applications running in enhanced mode to prevent character loss.

If the RDT is a 80386 based computer, Network Driver can still be used with a Windows application as long as Windows is running in standard mode (win /s). However, many Windows screen drivers and some of the TCP/IP protocol stacks are VxD drivers and can not be used with Windows in standard mode.

RESTRICTIONS

Network Driver retains exclusive control of all hardware and software interrupt vectors associated with its serial ports. Applications should not chain these interrupts. Once it is loaded into memory, Network Driver does not release the serial port until the computer is turned off or rebooted.

Although Network Driver can operate on COM1 or COM2, only one copy of Network Driver can be loaded at a time. Also, only one one protocol stack can be installed to use the Network Driver at a time. Multiple Windows applications using the protocol stack can be loaded, but the performance will not be very good because of the increased traffic through the radio.

INSTALLATION

This section gives the necessary information and instructions needed to install the EDACS Network Driver on a PC using DOS. Experience at installing protocol stacks and NDIS drivers is helpful. It is assumed that the user knows what a device driver is and has a DOS text editor available.

This document is divided into two parts: the general instructions for Network Driver, and specific installation procedures and sample files for third party TCP/IP protocol stacks. The general instructions are valid for any TCP/IP package except when this manual has provided specific installation instructions (e.g. Wollongong Pathway Runtime and Acccess v2.0).

REQUIREMENTS

You will need the following when installing the Network Driver.

- The EDACS Network Driver software disk containing the driver *EDACS.DOS*

- An NDIS compatible network communications software package. This can range from an IP protocol driver with a sockets interface to a full network communications package with an application layer, a TCP or UDP transport layer driver, and an IP protocol driver. The package must include an NDIS compatible Ethernet driver such as "EtherLink II" or "EtherLink3". The Network Driver replaces this driver.

- The protocol stack binding program NETBIND.EXE. This program is included in most NDIS compatible network communications software packages.

GENERAL INSTRUCTIONS

Following are general instructions for installing the EDACS Network Driver. Following a brief overview, each step is explained so that the user will have some knowledge as to what is being accomplished. The overview is meant to give a better understanding of what the installation instructions are performing and why the steps are necessary, but it can be skipped without affecting the installation.

Overview

In order for a PC to communicate over a network, it must have networking software that is properly configured, installed, and initialized. Usually, network drivers are listed in the CONFIG.SYS file and are loaded into memory when the computer is started. However, the drivers do not act together as a protocol stack until an initialization and binding process takes place. This process is managed by the Protocol Manager driver (PROTMAN.DOS or PROTMAN.SYS) and the NETBIND.EXE program.

When it is loaded into memory, the Protocol Manager reads an ASCII file, called PROTOCOL.INI, containing the configuration parameters of all the network drivers and information on how to assemble the drivers into a protocol stack. The Protocol Manager parses the PROTOCOL.INI file into a data area in memory that other NDIS drivers can access. During initialization, each network driver reads its configuration information from the data area created by the Protocol manager. Each driver also tells the Protocol Manager its characteristics, the driver's communication entry points, and the other drivers with which the driver wants to communicate.

After the drivers are loaded into memory, the NETBIND.EXE program is run from the AUTOEXEC.BAT batch file. NETBIND tells the Protocol Manager to bind the MAC layer driver to the protocol drivers. The Protocol Manager goes through its driver list, connecting together those drivers which are adjacent in the protocol stack.

In the case of the EDACS Network Driver, usually only two drivers are bound to form the protocol stack: the EDACS Network Driver itself, and the network driver that implements the TCP/IP protocol. Although NDIS allows the binding of multiple protocols (e.g. Novell Netware) to one MAC driver, Network Driver and the EDACS network expect only TCP/IP packet types. All other networking protocols should not be loaded.

PROTOCOL.INI Parameters

The PROTOCOL.INI file is used to configure the network drivers that load and bind to form the "protocol stack." The file is divided into sections, each with a header enclosed in brackets. The EDACS Network Driver section is separated by the header

```
[EDACS]
```

The configuration parameters for the Network Driver are listed below. The PROTOCOL.INI file will also contain parameters for the other

components of the protocol stack. Refer to the protocol stack documentation for more information on these parameters.

DriverName

The parameter `DriverName` defines the name of the Network Driver for other components in the protocol stack. This line must be present and must read:

```
DriverName = EDACS$
```

NetworkAddress

The `NetworkAddress` parameter defines the IP address for the MDT. This line must be present and must read as

`NetworkAddress = "0200`*`hhhhhhhh`*`"`

where the "*hhhhhhhh*" must equal the hexadecimal representation of the MDT's IP address. For example, if the IP address of the MDT is 147.117.37.64, then the line should read:

`NetworkAddress = "0200`*`93752540`*`"`

Port_Num

The `Port_Num` parameter is used to define the COM port that the Network Driver uses. This line is optional. If present, it should read as

`Port_Num =` *`x`*

where the *x* equals the COM port used. Valid port numbers are 1 and 2. If not present, the Network Driver uses COM1.

PriorityOff

The `PriorityOff` parameter tells the Network Driver to give priority to the RDI when a collision occurs on the RDI interface. This parameter is optional. If present, it should read as

```
PriorityOff
```

The RDI must be configured and have the correct software version for this feature to work correctly. Refer to the software release notes for the

RDI or the *Radio Data Interface Installation Manual* (LBI-38335) for more information.

HostID

Normally, the Network Driver cycles through logical IDs 1 through 15, inclusive, when originating calls. In turn, the EDG automatically expects calls to logical IDs 1 through 15 when a terminal using Network Driver is defined in the EDG configuration file.

Some systems may already have radios assigned in the logical ID space between 1 and 15. The `HostID` parameter allows you to tell the Network Driver to direct calls to the logical IDs listed. This line is optional. If present, it should read as

```
HostID = "LID List"
```

where the *LID List* is either a single logical ID or a comma delimited list. A range of logical IDs can be specified by providing the minimum and maximum values separated by a dash. White space is ignored and valid values range from 1 to 63. The Network Driver cycles through the list given, using each ID in the order they occur in the list. IDs can be repeated. An example is given below.

```
HostID = "2, 10,25,5,7, 30 - 40, 6, 6,6,6"
```

All IDs included in the host ID list must also be added to the EDG configuration table. If a terminal is roaming in a multinode network, its host ID list must be added to all EDGs in the network.

DelayTimer

You can adjust the time between consecutive calls initiated by the Network Driver using the *DelayTimer* parameter. The syntax for the *DelayTimer* parameter is shown below.

```
DelayTimer = t
```

Replace *t* with the number of clock ticks you want the Network Driver to wait. On most PCs, a clock tick equals 55 milliseconds. The *DelayTimer* setting will only take effect if the Network Driver receives a call request from the protocol driver while another outbound call is pending.

Using this parameter is optional. If no value is specified, the Network Driver uses 2 as the default setting for *DelayTimer*. The maximum allowable value is 100 ticks.

ACKATimer

The *ACKATimer* parameter gives control over how long the Network Driver waits after receiving an ACKA message on the RDI interface before initiating the next outbound message transfer. The delay is specified in clock ticks as shown below

```
ACKATimer = t
```

where *t* is a number of clock ticks between 1 and 100. If the parameter is not specified, the Network Driver uses 2 clock ticks as the default setting.

ACKARetry

Using the *ACKARetry* parameter, you can tell the Network Driver to resend outbound messages after receiving an ACKA message on the RDI interface. The syntax for the *ACKARetry* parameter is shown below

```
ACKARetry = x
```

where *x* is the number of attempts the Network Driver should make to successfully send a message. If the parameter is not specified, the default value is 0.

Installing the EDACS Network Driver

The EDACS Network Driver doesn't come with an installation program since no single installation procedure works with all available TCP/IP software packages. The following procedure is written so that the EDACS Network Driver replaces a pre-installed Ethernet driver. The General Installation procedure involves installing a TCP/IP communications package for an Ethernet, copying the EDACS Network Driver to the hard drive, editing the necessary files, and then rebooting the computer. The files can be edited using any ASCII based DOS text editor.

Some network communication packages allow you to install an NDIS driver not included with the package. If you wish to install the EDACS Network Driver during that phase of the installation, follow the vendor installation instructions carefully.

When following the instructions below, replace fields in *italics* with a specific name of your choice. Also, know the general configuration of your computer. For example, if the hard drive is partitioned, know which drive contains CONFIG.SYS and AUTOEXEC.BAT and which drive the network communications software is on.

An outline of each step is listed below. More details are given in later sections. Each step must be followed exactly for the EDACS Network Driver to function properly.

- Install the network communications software
- Copy the EDACS Network Driver software
- Copy the PROTOCOL.INI file
- Edit the CONFIG.SYS file.
- Edit the AUTOEXEC.BAT file.
- Edit the PROTOCOL.INI file.
- Edit the SYSTEM.INI file

Install the network communications software

The EDACS Network Driver works with other vendor's network communications packages. You should verify that the software is installed and working correctly before installing the Network Driver. The easiest way to check out the networking software is to install and configure it to work on an IP network. Then use the software to connect to a host on the IP network. If you are successful, then the network software is probably installed correctly.

If there is not an IP network available, install the software and as described below and continue with the next step. If your network communication package is already installed on your PC and has been verified to work correctly, skip this step.

a) Install the network communications software according to the vendor's instructions. Be sure that the Ethernet adapter card you select matches the card in your system. If you do not have a card installed, select any card because the Ethernet driver will be replaced by the Network Driver.

b) Do not install support for any other networking communications protocols, such as Novell Netware. Most network software packages want to install a network protocol such as Novell Netware. If prompted for a network protocol, choose the network that you are presently using. This information will have to be removed later in steps 3 and 4.

c) If you have an Ethernet card present in your system, verify that the network communications package is working properly. Familiarize yourself how to use your network communications software. This will make troubleshooting easier if a problem is encountered in trying to get the EDACS Network Driver initially working.

d) Determine which Protocol drivers and MAC (Ethernet) driver tie to which sections in the PROTOCOL.INI file. (The PROTOCOL.INI file can usually be found in the same directory as the network communications software.) As stated in the overview, the EDACS Network Driver itself and the Protocol Driver(s) that implements the TCP/IP protocol are the only drivers needed for the EDACS Network. Other drivers that are unnecessary may cause problems in the future if you try to operate them with the Network Driver.

Copy the EDACS Network Driver software

a) Create a directory for the EDACS Network Driver software. The suggested path and name is "C:\EDACS". If another path and name is desired, remember the path and directory name, they will be needed later. This newly created directory will here on be referred to as the "EDACS directory".

b) Copy the file EDACS.DOS from the diskette to the EDACS directory.

Copy the PROTOCOL.INI file

Copy the proper PROTOCOL.INI file to the EDACS directory. This preserves the original in case you want to remove the EDACS Network Driver later.

Edit CONFIG.SYS

With the EDACS Network Driver software copied, the next step is to change your system's configuration file so that it will load the EDACS Network Driver on start-up.

a) Copy CONFIG.SYS to a different filename so that it can be recovered later. Open the CONFIG.SYS file in a text editor and change the following line

DEVICE=*path*\PROTMAN.DOS /I:*path*

to

DEVICE=*path*\PROTMAN.DOS /I:*EDACS directory*

b) Now you need to replace the Ethernet driver with the EDACS Network Driver. To do this, change the following line

```
DEVICE=path\Ethernet Driver
```

to

```
DEVICE=EDACS directory\EDACS.DOS
```

Ethernet Driver refers to the driver that the network communications software chose depending upon the type of Ethernet Adapter Card you specified during installation. Some common names are ELNK3.DOS and ELNKII.DOS. If you are unsure of the name of the Ethernet driver, don't guess. Refer to the literature that came with the network communications software.

c) Finally, remove lines that load network device drivers that don't use Internet Protocol (IP). The EDACS Network Driver only supports IP.

Edit AUTOEXEC.BAT

a) Copy your AUTOEXEC.BAT file to a different file name so that it can be recovered later.

b) Edit the AUTOEXEC.BAT file with your editor and remove lines which invoke network communications programs that don't use Internet Protocol (IP). For example, you want to remove any lines that load Novell's Netware protocol driver IPX.COM for communications over Ethernet and its network workstation shell NETX.EXE.

c) Check AUTOEXEC.BAT to ensure that it contains a line invoking the NETBIND program. This looks like:

```
path\NETBIND
```

Your network communications software should have added this line during its installation process. If there isn't a line that invokes the NETBIND program, add it to the AUTOEXEC.BAT file.

Edit PROTOCOL.INI

The next step is to change to PROTOCOL.INI file so that it contains the configuration information for the Network Driver.

a) Edit the PROTOCOL.INI file located in the EDACS directory and insert a Protocol Manager section. This configures the Protocol

Manager for EDACS Network operation. Insert the following three lines at the beginning of PROTOCOL.INI if not already present:

```
[PROTMAN]
DriverName = PROTMAN$
Dynamic = NO
```

b) Just below the protocol manager section, insert an EDACS section to as shown below to configure the Network Driver. Replace “hhhhhhh” with the IP address of the terminal.

```
[EDACS]
DriverName = EDACS$
NetworkAddress = "0200hhhhhhhh"
```

c) Locate the `Bindings` statement for the TCP/IP protocol driver that you installed to use with the Network Driver. Change the `Bindings` statement to refer to the Network Driver name as shown below.

```
Bindings = EDACS
```

d) Remove the sections in the PROTOCOL.INI file that refer to drivers you removed in the steps above.

Edit SYSTEM.INI

If the network communications software you are using runs in Windows, then you must make the following changes to the SYSTEM.INI file in the Windows installation directory. Normally this file is found in the C:\WINDOWS subdirectory.

In the `[386Enh]` section, you must disable Windows from taking control of the COM port used by the EDACS Network Driver. Insert the following lines

```
[386Enh]
...
Comxirq = -1
```

where *x* is the number of the COM port used by the EDACS Network Driver (1 or 2). Do not use the Windows Control Panel to view the settings of the port assigned to the Network Driver. This will enable the port again.

Remove any lines in the `[386Enh]` section such as:

```
[386Enh]
...
Com<x>AutoAssign = #
```

where *x* is COM port number used by the EDACS Network Driver and # is a integer value.

USING NETWORK DRIVER WITH SPECIFIC VENDOR PROTOCOL STACKS

The following sections contain information that is useful in installing and using several protocol stacks. These instructions are specific to the version number listed for each package. Newer releases could change the order of execution or validity of these instructions.

Table 1 - Protocol Stack Characteristics Summary

	Wollongong Pathway 2.0	Distinct TCP/IP	Frontier Technologies Super TCP/NFS
IP Fragments	Limited	Yes	Yes
IP Reassembles	Limited	Yes	Yes
TCP Window Advertisement	Yes	Limited (1K minimum)	No
TCP Retry Timer Config.	Yes	Yes	Yes
TCP Disconnect Problems	No	No	No
TCP Segment Size	482	542	536
UDP Message Size	578, 1044, 1526	2.7K	9.1K
API Exists	Yes	Yes	Yes
Winsock Compliant	Yes	Yes	Yes

IP Fragments - Does the IP layer in the software break up large messages to match the MTU of the lower layer?

IP Reassembles - Upon receiving fragments from a lower layer, does the IP layer reassemble them into the original message?

TCP Window Advertisement - Can you configure the TCP layer to tell the receiving TCP layer what window size it is using ?

TCP Retry Timer Configurable - Can you configure the time-out before the TCP layer tries to send a fragment again ?

TCP Disconnect Problems - Are there problems with the disconnect sequence where sockets are not freed?

TCP Segment Size - What is the segment size used by the TCP layer when configured for optimum performance on EDACS ?

UDP Message Size - What is the maximum message size allowed by the UDP layer?

API Exists - Is there an API for the package?

Winsock Compliant - Is the API compliant to the Winsock specification for Windows socket communications.

WOLLONGONG PATHWAY RELEASE 2.0

The Wollongong Pathway Release 2.0 protocol software works with the Network Driver for both UDP and TCP communications. However, you can not configure the package to optimize both protocols for maximum performance. This is the only recommended protocol stack that you can use with a DOS application.

Installation

Unlike most network communications software, Wollongong Pathway requires a different installation procedure to work correctly with the Network Driver. Do not follow the General Installation procedure. Instead, follow the steps below.

Wollongong keeps a copy of the information in the PROTOCOL.INI, CONFIG.SYS, and AUTOEXEC.BAT files which pertain to the Wollongong initialization. It reads this copy every time it initializes rather than reading the files. If you try to change the statements in these files yourself, Wollongong will still try to use the information it stored and nothing will work.

a) Make a PROTOCOL.INI file as shown in Figure 1.

b) Copy the PROTOCOL.INI file, the Network Driver software (EDACS.DOS), and the protocol manager (PROTMAN.SYS) into the same directory on your hard disk or onto a floppy disk. You can find the protocol manager on disk 2 of the Pathway Runtime software.

c) Follow the instructions in the Wollongong Pathway manual to start the installation process. When the installation prompts you for the path to your driver, enter in the path to the place you selected in step b). The installation program will look in the directory you provided and then show EDACS as the driver it found. Select EDACS as your driver.

d) When asked for the hardware interrupt of your NDIS driver, choose the interrupt number corresponding to the serial port you are using for Network Driver. For example, COM1 uses interrupt 4 and COM2 uses interrupt 3.

e) During the installation procedure, Wollongong asks for a default gateway address. The address tells the Wollongong software where to route datagrams with an unknown destination. You must provide an unique IP address on the same network address assigned to the EDACS network. For example, if your EDACS network has the address 190.0.0.0, then the default gateway address must be between 190.0.0.1 and 190.0.255.254, inclusive.

f) When the installation is complete, verify that your PROTOCOL.INI, CONFIG.SYS and AUTOEXEC.BAT look as shown in Figure 1. Add any statements that are missing.

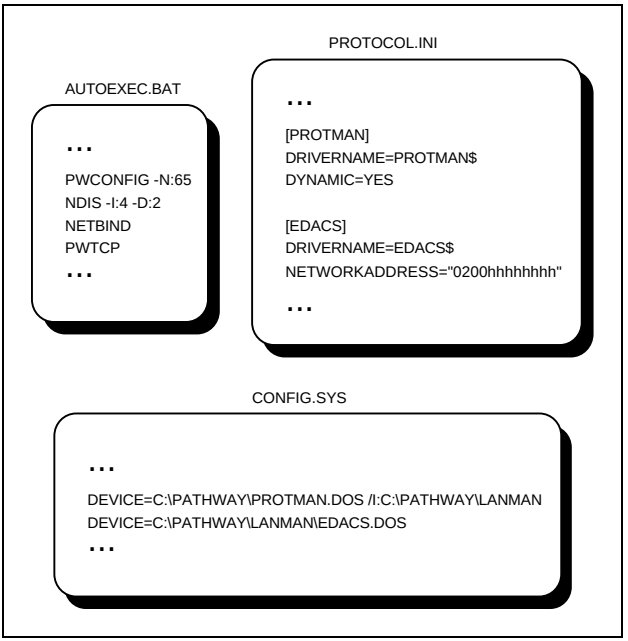

Figure 1 - Configuration Files for Wollongong Pathway 2.0

Once the installation procedure is complete, you need to configure the operating parameters of the network software by running the PWSETUP program from the DOS prompt.

From the main menu of the PWSETUP program, select the Advanced Configuration Setup screen. Set the following parameters as described below. You can check your configuration by looking at the INFO.TWG file created by the Wollongong software.

Packet Buffer Size

When using the UDP protocol, select the packet size available which is greater than or equal to the largest message your application will send. If you send a message larger than the defined packet buffer size, Pathway will truncate the message, transmit the shortened version correctly, and not return an error.

For TCP protocol, select a packet buffer size of 576 bytes.

TCP Segment Size

The TCP segment size is not a configurable parameter in the Wollongong Pathway software. The Pathway TCP layer does respect the 502 byte MTU advertised by the Network Driver and uses a TCP segment size of 482 bytes.

TCP Window Size

The most efficient window size for the EDACS system is 512 bytes. Note that you must select a packet size of 576 bytes to be able to select a window size of 512 bytes. If you are not using the TCP protocol, this parameter can be ignored.

Minimum Retransmission Timer

Set the Minimum Retransmission Timer to a value large enough to allow a TCP segment and acknowledgment to be transmitted before a time-out occurs. Try setting this to 364 ticks (20 seconds) and setting the EDG's **MSG_TIMEOUT** parameter to the default value of 30 seconds. You should also set the Network Driver's **ACKARetry** to 3 so that the

Network Driver retransmits messages that fail because of poor signaling conditions.

Maximum Retransmission Timer

Select a value for the Maximum Retransmission Timer larger than the Minimum Retransmission value. Try setting this to 1818 ticks (100 seconds.

IP Fragmentation and Reassembly

The Pathway IP layer does fragment and assemble messages. However, there is a non-configurable maximum limit of six fragments.

TCP Connect and Disconnect

For a TCP connect-accept sequence, both the sender and the receiver transmit each message twice. Note that a bug in the stack causes one of these messages to be lost if this is the first transmission, so the TCP application must be patient in waiting for a pending connection. The disconnect and reset operations work as expected.

DISTINCT TCP/IP FOR WINDOWS

The Distinct TCP/IP protocol stack is native to Windows and cannot be used for DOS applications. UDP works quite well but is a little limited on reception due to the reassembly timers. TCP does not work well because the minimum 1K byte TCP window size causes collisions which hurt the performance.

Installation

There are specific items during the installation of this stack that aren't covered by the General Installation procedure. Follow installation procedure for Distinct TCP/IP for Windows as stated in the manuals. You can only install Network Driver with this stack by installing the package with another Ethernet driver and replacing it with the Network Driver. Some additional directions are given below.

a) If asked for the hardware interrupt of your NDIS driver, choose the interrupt number corresponding to the serial port you are using for Network Driver. For example, COM1 uses interrupt 4 and COM2 uses interrupt 3.

b) If asked for the I/O Address for the driver, pick the address associated with the serial port I/O address that you are using. For example, if you are using serial port 1 (COM1) then your I/O address is `03F8` hex.

c) During the installation procedure, Distinct TCP/IP asks for a default gateway address. The address tells the Distinct TCP/IP software where to route datagrams with an unknown destination. You must provide an unique IP address on the same network address assigned to the EDACS network. For example, if your EDACS network has the address 190.0.0.0, then the default gateway address must be between 190.0.0.1 and 190.0.255.254, inclusive.

d) When the installation is complete, your PROTOCOL.INI, SOCKET.INI, CONFIG.SYS and AUTOEXEC.BAT should look as shown in Figure 2. Add any statements that are missing.

Figure 2 - Configuration Files for Distinct TCP/IP

Configuring and Using Distinct TCP/IP

Go to the Protocol Menu in the Setup Program. Set the following parameters as described below.

Maximum Transmission Unit

If using UDP, change the MTU size to 552 bytes. If using TCP, change the MTU size to 542.

ARP Retry and Transmission Timer

Set the ARP resend and the ARP time-out values to zero.

IP Fragmentation and Reassembly

The Distinct TCP/IP stack does fragmentation in the IP protocol layer. When transmitting data, the IP layer will fragment messages up to 9K bytes. However, the IP layer will only reassemble messages that are approximately 2.7K bytes in size. If the message exceeds that limit, the IP layer throws out the message. Therefore, do not attempt to send a UDP message from your application greater than 2K bytes.

TCP Segment Size

The TCP segment size is not configurable in the Distinct TCP/IP stack. Selecting an MTU of 542 bytes will provide the best throughput.

TCP Window Size

The TCP Window Size is configurable but the minimum allowable value is 1K bytes. This can cause a problem because the transmitter can send two segments without waiting for an acknowledgment. The second transmitted segment can collide with the acknowledgment for the first segment.

Minimum Retransmission Timer

You can control the retransmission timer with the undocumented line

```
resend = [value]
```

in the *sockets.ini* file where **value** is in seconds. Set the Minimum Retransmission Timer to a value large enough to allow a TCP segment and its acknowledgment to be transmitted before a time-out occurs.

TCP Connect and Disconnect

The TCP connect and disconnect sequences work as expected except that the Distinct TCP/IP stack sends its disconnect segment with the **RST** (reset) bit set in the code bits of the TCP header. This is probably a bug in the stack. Since the disconnect sequence can take a long time, make sure that the stack remains active after applications are closed, otherwise the sockets may not be freed.

FRONTIER TECHNOLOGIES SUPER TCP/NFS V4.00 R2

The Super TCP/NFS protocol stack works only with Windows. Since the stack is a virtual device driver, you cannot use it with Windows in Standard mode. This means you need a fast platform (80486, 33 MHz or better) for the terminal.

Installation

The Network Driver and Super TCP/NFS can be installed faster by using the following instructions instead of the general instructions given above.

a) Create a directory called C:\MSLANMAN.DOS. Create a PROTOCOL.INI file as shown in Figure 3.

b) Follow the instruction in the Super TCP/NFS manual to run the installation program. The Super TCP/NFS program will detect the Network Driver as a possible NDIS MAC driver.

c) When asked to select a driver, pick EDACS as your MAC driver.

d) When defining the `Interface setup`, hit the **`Advanced...`** button.

e) Click on the **`Use Netbind`** box.

f) Go down the scroll bar in the setup program to **`IP setup`**.

g) Click on the **`RIP`** box to disable it.

h) When asked for a default gateway, enter an unique IP address on the same network address assigned to the EDACS network. For example, if your EDACS network has the address 190.0.0.0, then the default gateway address must be between 190.0.0.1 and 190.0.255.254, inclusive.

i) Go down the scroll bar in the setup program to `TCP setup`.

j) Set the `Min Resend` to a value sufficiently large enough to allow a TCP segment and acknowledgment to be transmitted before a time-out. A suggested value is 10000 ms.

k) Follow the **Edit SYSTEM.INI** step from the general installation instructions.

l) When the installation is complete, your CONFIG.SYS, AUTOEXEC.BAT, and PROTOCOL.INI files should look as shown in Figure 3. Add any statements that are missing.

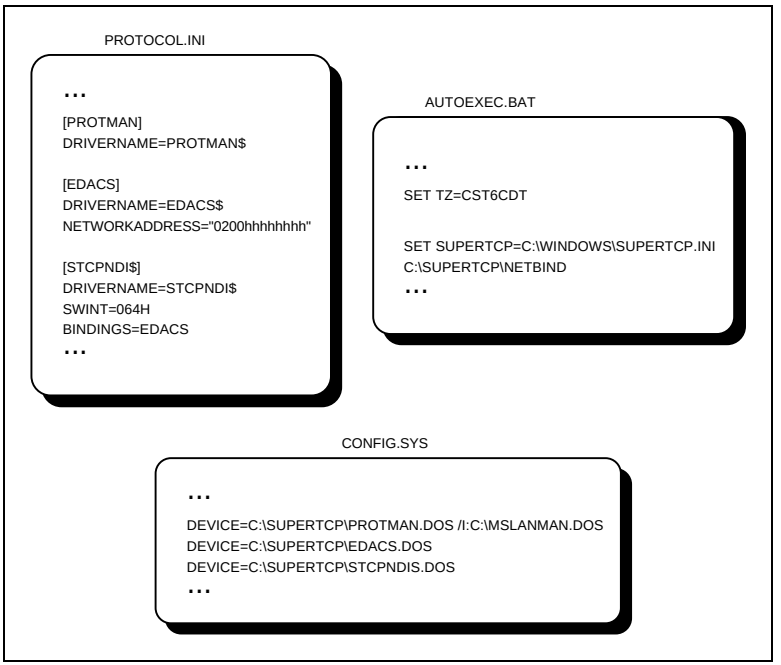

Figure 3 - Configuration Files for Super TCP/NFS v4.00 R2

Run the *SetupTCP* program to configure the operation of the Super TCP/NFS. Select the *IP Parameters* icon in the *SetupTCP* list box to bring up the *IP Parameters* dialog box. To configure the operation of the TCP layer, select the *TCP Parameters* icon to bring up the *TCP Parameters* dialog box.

IP Fragmentation and Reassembly

The IP layer fragments messages up to approximately 9K bytes in size. It will reassemble messages up to the size of the internal frame buffer of the Network Driver or until the reassembly timer expires.

The *Time To Live* (**TTL** setting in the *IP Parameters* dialog box) value is implemented as a function of time rather than gateway hops. If a fragmented IP datagram is not completely received, then the stack will send a *Time Exceeded for a Datagram* ICMP message (type 11). Set the TTL value lower than the round trip transmission time of a message to avoid collisions between messages and ICMP messages.

If you are using TCP, set the **Maximum Packet Size** parameter to 576 bytes.

TCP Segment Size

You can not configure the TCP segment size in the Super TCP/NFS protocol stack, but you can configure the software to negotiate the segment size with the other host. TCP will negotiate segment size up to the maximum packet size set in the *IP Parameters* dialog box.

If you use the Super TCP/NFS stack on both ends of the conversation, TCP sends segments of 536 bytes. It waits for the acknowledgment before sending the next segment.

TCP Window Size

The TCP Window Size is configurable but changes are not reflected in the TCP header during data transmissions.

TCP Connect and Disconnect

The TCP connect sequence is somewhat different from other stacks but still works. When it receives a **SYN** segment to initiate a connection, the

stack's TCP layer responds with its own **SYN** segment but without the **ACK** bit set as normally happens. This causes an extra **ACK** segment in the connection handshake, but the connection sequence will still complete.

The disconnect sequence works as expected.

Minimum Retransmission Timer

You can control TCP retransmissions through the **Van Jacobson Round-Trip Time** section of the *TCP Parameters* list box. The **RTT Initial Value** defines the initial amount of time the TCP layer will wait before initiating a retransmission. The TCP layer will adjust the retransmission time-out between the **RTT Minimum Value** and the **RTT Maximum Value** depending on how responsive the network is.

Set the **RTT Initial Value** to the average round trip time of a message and its acknowledgment. The **RTT Minimum Value** should be set to the minimum expected round trip time while the **RTT Maximum Value** should be some value greater than the setting for the **RTT Initial Value**. If you are experiencing a lot of collisions, raise the initial and minimum settings. Setting the maximum value too high will reduce the performance of your application for terminals in a poor signaling environment.

ARP Retry and Transmission Timer

You can access the ARP configuration parameters through the *MAC Parameters* icon in the *SetupTCP* list box. There is no way to totally disable the ARP mechanism, but you can set the interval between ARP messages to be very large. The Network Driver will absorb all ARP requests from the protocol stack and prevent them from being transmitted over EDACS.

Set the **ARP Time-out** values to 65535. Set the **ARP Time to Live** value to 65535. Finally, set the **ARP Retry Interval** to 65535. This will limit the ARP transmissions to one every 18 hours.

TECHNICAL NOTES

USING TCP ON THE EDACS NETWORK

The *Transmission Control Protocol* (TCP) is a popular transport layer protocol commonly used on IP networks. The purpose of TCP is to provide a buffered reliable full duplex communications path for applications. Unfortunately, the very nature of the services it provides makes it difficult to use over an RF network.

TCP provides reliability by using a positive acknowledgment scheme that retransmits undelivered messages. In other words, a message recipient must send an acknowledgment back to the source indicating that the message was received correctly. If it does not receive an acknowledgment for a message within a configurable time, the sender retransmits the message.

While the TCP acknowledgment scheme works well in full duplex high speed Ethernet networks, there are problems when it is transferred to the EDACS network. First, the half duplex nature of EDACS can cause collisions as messages and acknowledgments fight for use of the RF channel. Collisions tend to force more retransmissions which increases the network congestion. The collision problem is compounded by the slower data transmission speed of EDACS. Finally, the traffic caused by the acknowledgments and retransmissions can quickly saturate the network if there are not enough RF channel resources to support the number of data terminals using TCP.

There are ways to configure the operation of TCP to improve its performance on EDACS. Most protocol stacks allow you to configure TCP operation through two configuration parameters: *TCP window size* and the *TCP minimum retransmission time.* The proper configuration of these parameters, together with the *ACKARetry* and the *ACKATimer* parameters for Network Driver, will greatly improve the communication performance of your application.

TCP Segment Size

The *TCP segment size* defines the size of each packet in a TCP conversation. You want to use this parameter to keep the segment size equal to or below 482 bytes to satisfy the 502 byte MTU of the EDACS network (482 data bytes + 20 byte TCP header = 502). This reduces the traffic on the EDACS network and helps limit message collisions. If the IP

layer in the protocol stack handles message fragmentation using the MTU size advertised by the Network Driver, this parameter can be overlooked. At present, none of the protocol stacks has both a TCP segment parameter and fragments messages in the IP layer.

TCP Minimum Retransmission Time

The *minimum retransmission time* controls when the sender will retransmit a segment if an acknowledgment is not received. In general, you get the best performance from your application on an RF network by generating the least amount of traffic. This means limiting the amount of transmissions caused by the TCP protocol layer. If you set this number too low, the sender's TCP layer will generate retransmissions faster than the host can acknowledge the original messages. This will cause collisions, keep RF channels busy, and generally cause poor performance for all terminals in the network.

To improve performance, set this parameter value greater than the worst case round trip transmission time of a TCP segment and it's corresponding acknowledgment. This will vary depending on the number of channels in your network, the amount of voice and data traffic on your network, and the average size of your messages. Make sure that the EDG's **MSG_TIMEOUT** parameter is set so that messages queued in the EDG are removed before the TCP Minimum Retransmission timer expires. This prevents duplicate messages from congesting the network.

Setting the *ACKARetry* Parameter

TCP provides a reliable communication channel by retransmitting lost segments. However, the previous section recommends setting the minimum retransmission time to a high value to limit the amount of traffic on the EDACS network. This provides a long recovery time for segments that are not delivered because of a failure in the RF channel. To compensate for this, use the *ACKARetry* parameter to make the Network Driver retransmit when an RF failure occurs. For most situations, a setting of one should be adequate.

Be careful not to set the *ACKARetry* value too high. In poor RF signaling conditions, you may find that the Network Driver is still trying to send the segment when TCP's minimum retransmission timer expires and TCP retransmits the segment.

TCP Window Size

Most protocol stacks also allow you to control the *TCP window size*. This defines the amount of data a station can receive without having to send an acknowledgment. However, it does not force the receiving station to wait to send the acknowledgment until the window is full. As long as the receiving station can process a segment before the next one arrives, it will send an acknowledgment back. If it gets more segments before it processes the current segment, the receiving station can wait and send one acknowledgment for multiple messages up to the TCP window size. On an Ethernet based IP network where the communication bit rate is high, setting the TCP window size to a multiple of the network MTU improves throughput because a station can send multiple TCP segments before having to stop and wait for the acknowledgment.

In RF networks such as EDACS, the transmission time of a segment is much greater than the processing time of the receiver. As a result, the optimum TCP window size for EDACS is the size of a TCP segment. This forces the transmitting TCP layer to wait for an acknowledgment for each segment before sending the next segment and eliminates the collisions between TCP segments and acknowledgments.

Limiting the Application Connection Sequence

Many applications use a lengthy connection sequence to establish communications between the remote terminal application and the host application. Of course, the connection sequence includes the three handshake sequence used by TCP to establish its communications channel as well as the acknowledgments generated by TCP for each message the application sends. The situation becomes worse if the application's messages do not fit in the TCP segment size because TCP generates multiple acknowledgments for each application message. Often the application presents its peak load to the network during this sequence. As a result, there is a long period where the user is locked out of the application while it tries to complete the connection sequence.

The problem worsens if the user population tends to start up their applications at the same time, e.g. at the beginning of a work shift. Try to make the application's connection sequence as simple as possible with the remote and host applications exchanging a few messages.

USING TELNET AND FTP

Most protocol stacks include two TCP applications: *Telnet*, a terminal emulator; and *FTP*, a file transfer program. Because of their nature, neither program will work in the EDACS network.

Telnet generates at least two data calls for each character typed at the terminal. If *Telnet* is configured for host echo mode, the number of calls goes up as the host echoes the character typed in another data call and the TCP layer generates its acknowledgments.

FTP tries to open two TCP connections, one for the data transfer and the other for control messages. However, so many collisions occur in trying to open the connections, neither connection can complete and *FTP* times out.

USING THE WINSOCK API

The following are guidelines for developers using the Winsock API to create applications running over the EDACS network with the Network Driver.

UDP Calls

socket()

No special changes are needed here.

sendto()

Most protocol stacks will limit the amount of data you can send with this call.

recvfrom()

The buffer from which data will be read must be larger than the datagram in the receive queue of the protocol stack.

bind()

The port number supplied must be unique and known by the receiving application.

setsockopt()

SO_RCVBUF and SO_SNDBUF can be reset to the application buffer sizes.

WSAAsyncSelect()

No special changes needed here.

closesocket()

No special changes needed here.

TCP Calls

socket()

No special changes needed here.

connect()

The port number given for the connection must be unique and known by the receiving application. A *select()* call should follow since this will always return a WSAEWOULDBLOCK, assuming this is a non-blocking socket (Winsock default). Some protocol stacks make more than one connection attempt if the initial one fails. The application should disable the application window, mouse and keyboard functions until the connection attempt succeeds or fails because the connection process is very long.

send()

The *send()* call rarely sends the whole data stream in one call. Therefore, it is necessary to enclose the *send()* call in a loop. The *send()* call returns a 0 when complete. As with a *connect()* call, disable the application window, mouse and keyboard functions until the *send()* call completes or fails. If you wish to allow other applications on the same PC to receive messages, put a *PeekMessage()* loop within the *send()* loop.

listen()

No special changes needed here.

accept()

No special changes needed here.

recv()

The receiving buffer doesn't have to be large at all, nor does it need to be enclosed in a loop. This is due to the fact that each TCP segment is sent with a separate *send()* call. Therefore, the protocol stack generates a separate window message for each TCP segment it receives.

bind()

No special changes needed here.

setsockopt()

Do not use SO_LINGER or SO_KEEPALIVE options if possible.

WSAAsyncSelect()

No special changes needed here.

closesocket()

Put a *select()* statement here to wait for the completion of the *closesocket()* call. Also, disable the application window, mouse and keyboard functions.

select()

The time-out structure should have values that reflect the EDACS network.

Ericsson Inc.
Private Radio Systems
Mountain View Road
Lynchburg, Virginia 24502
1-800-528-7711 (Outside USA, 804-528-7711)

Printed in U.S.A.