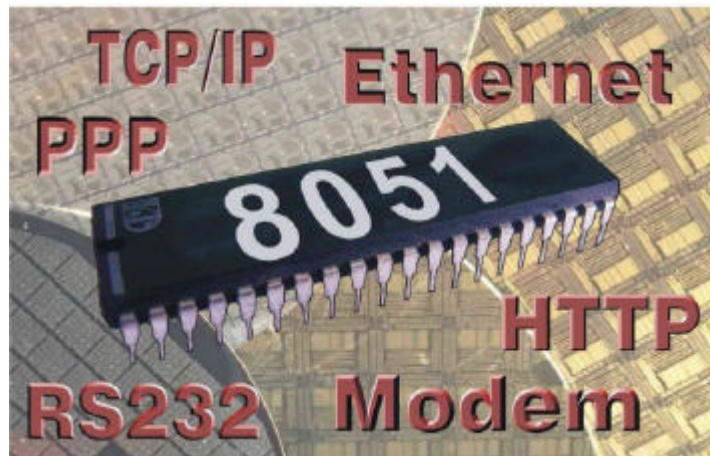


TCP/IP - An Introduction for 8 & 16 bit Microcontroller Engineers



By C. L. Stephens

Computer Solutions Ltd

1a New Haw Road

Addlestone

Surrey, KT15 1BU, England

Tel: +44 (0)1932 829460

Website: www.computer-solutions.co.uk

Computer Solutions Ltd make no warranty of any kind with regard to this material, including, but not limited to the implied warranties of merchantability and fitness for a particular purpose. Computer Solutions Ltd shall not be liable for errors contained herein nor for incidental or consequential damages in connection with the furnishing, performance, or use of this material. This manual is a tutorial and as such its contents are intended for explanation purposes not as definitions.

CMX-MicroNet is a registered trademark of CMX Systems Inc, RTIP and Webster are registered trademarks of EBSnet Inc all other trademarks are acknowledged.

This document may be copied in printed or electronic form for individual study purposes but may not be reproduced for sales or marketing purposes without specific written permission from Computer Solutions Ltd. If so copied it must not be modified in any way.

I would like to thank Paul Bosselaers the MicroNet product manager and its lead programmer for helpful discussions and for reviewing much of this document although any errors remain the responsibility of the Author.

Version 1.0 April 2002

Some sections of this manual contain material copyright CMX Systems Inc and Catalyst Inc, these have been used with their permission.

CMX-MicroNet, CMX-TCPIP and EBS-RTIP may be purchased outside the USA from:

Computer Solutions Ltd

**1a New Haw Road
Addlestone, Surrey
KT15 2BZ, England**

**Tel: 01932 829460
www.computer-solutions.co.uk
sales@computer-solutions.co.uk**

In the USA the CMX products (MicroNet, CMX-TCPIP, CMX-RTX) are available from:

CMX Systems Inc

12276 San Jose Blvd.
Suite 119
Jacksonville
FL 32223, USA

Tel: +1(904)8801840
Email: cmx@cmx.com
Website: www.cmx.com

In the USA the EBSnet products (RTIP, Webster) are available from:

EBSnet Inc

P.O. Box 873
Groton
MA 01450, USA

Tel: +1(978)448 9340
Email: sales@ebsnetinc.com
Website: www.ebsnetinc.com

Table of Contents

1. INTRODUCTION.....	5
1.1. OBJECTIVES	5
1.2. TYPICAL APPLICATION	6
<i>Example 1 Micro to Micro.....</i>	<i>6</i>
<i>Example 2 Micro to PC</i>	<i>6</i>
<i>Example 3 Remote dial-up Micro.....</i>	<i>7</i>
<i>Example 4 Multi Drop Networks.....</i>	<i>7</i>
2. TRANSMISSION CONTROL PROTOCOL	7
3. USER DATAGRAM PROTOCOL	9
4. MAKING THE CONNECTION	10
4.1. IP ADDRESS AND HOSTNAMES.....	10
4.2. SERVICE PORTS	11
4.3. SOCKETS.....	11
4.4. BLOCKING VS. NON-BLOCKING SOCKETS.....	12
4.5. CLIENT-SERVER APPLICATIONS	12
5. DRIVERS LAYERS AND STACKS	14
5.1. PHYSICAL LAYER	14
5.2. LINK LAYERS - SLIP & PPP.....	15
5.3. MODEMS.....	15
5.4. LINK LAYERS – ETHERNET	16
6. HIGHER LEVEL PROTOCOLS.....	16
6.1. FILE TRANSFER PROTOCOL (FTP).....	16
6.2. GETTING AN IP ADDRESS (BOOTP, DHCP & TFTP).....	17
6.3. WEB SERVER (HTTP).....	17
6.4. JAVA AND BEYOND	18
6.5. SIMPLE MAIL TRANSFER PROTOCOL (SMTP).....	19
6.6. POST OFFICE PROTOCOL (POP3).....	19
6.7. SIMPLE NETWORK MANAGEMENT PROTOCOL (SNMP)	19
6.8. PING	19
7. CHOOSING A PROTOCOL.....	20
<i>Example 1 Micro to Micro.....</i>	<i>20</i>
<i>Example 2 Micro to PC</i>	<i>20</i>
<i>Example 3 Remote dial-up Micro.....</i>	<i>21</i>
<i>Example 4 Multi Drop Networks.....</i>	<i>21</i>
8. CMX MICRONET FEATURES AND LIMITATIONS.....	22
9. INTEGRATING CMX-MICRONET	23
9.1. USING TCP/IP FOR COMMUNICATIONS	23
9.2. USING A SERVER	23
9.3. CMX-MICRONET WITH AN RTOS	24
9.4. ALLOCATING ADDRESSES.....	24
10. FURTHER READING AND BROWSING	25
11. COMMON MNEMONICS	26

1. Introduction

1.1. Objectives

The aims of this white paper are twofold, firstly to act as an introduction to the TCP/IP protocols for Embedded Engineers who may not have used them before in applications. Secondly we will go into some detail of the CMX-MicroNet implementation of TCP/IP. MicroNet provides an implementation of TCP/IP that has been optimised for use on small 8 and 16 bit CPUs that may have very little memory available and so would not be able to use large conventional stacks. In order to achieve a small footprint some limitation on the stacks operations have been made and we explain what these are in the relevant sections.

To provide an example of the memory required for MicroNet here are some ROM & RAM figures for the AVR, a 16 bit CPU and the 8051 an 8 bit CPU.

AVR		8051 (Keil)	
ROM		ROM	
UDP/IP + core	4918 bytes	UDP/IP + core	5367 bytes
or		or	
TCP/IP + core	8184 bytes	TCP/IP + core	8340 bytes
or			
UDP + TCP/IP + core	9102 bytes	PPP	2883 bytes
		Ethernet	3055 bytes
PPP	3804 bytes	Modem	480 bytes
Ethernet	2972 bytes	HTTP Server	3013 bytes
Modem	442 bytes	FTP Server	3967 bytes
HTTP Server	2932 bytes	Virtual file	932 bytes
Virtual file	1436 bytes	DHCP Client	3297 bytes
		SMTP Client	3260 bytes
RAM (not counting size of buffers)		RAM (not counting size of buffers)	
UDP/SLIP	50 bytes	UDP/SLIP	55 bytes
TCP/PPP/HTTP	712 bytes	TCP/PPP/HTTP	620 bytes

Computer Solutions Ltd also sells the EBS-RTIP fully compliant stack, which is more appropriate should you have a CPU with a significant amount of available memory. For comparison TCP/IP alone on the RTIP stack takes ~ 64K ROM and 32K of RAM. This stack is available as a portable system written in C and modelled on a PC implementation to run standalone or under your own OS. Alternatively we can supply it badged as CMX-RTIP, configured for specific embedded CPUs to operate under CMX-RTX the Real Time Operating System. This stack also supports many of the options not available with MicroNet as well as some of the more complex protocols such as SSL, IMAP, NFS who's details we have left out of this introductory tutorial. Even if you decide to use this larger stack we hope you will still find this a helpful introduction to the subject of TCP/IP protocols and their use in embedded applications.

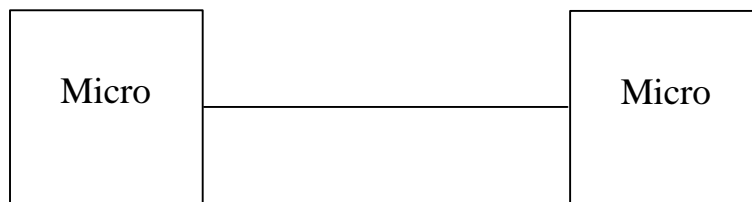
Sections 2 – 4 of this introduction to the topic are loosely based (with permission) on the introduction to TCP/IP found in Catalyst Inc's SocketWrench manual (see further reading).

1.2. Typical Application

Some of the applications, which we see MicroNet being used in, are directly and permanently connected to the Internet. Others will connect to the Internet intermittently to exchange messages or to upload and download files. But the majority of the applications that benefit from using MicroNet will consist of devices that are connected to one another or to PC's by RS232, dial up telephone lines, Ethernet or other hardware connection techniques to form local networks (sometimes known as intranets). Much of the interest among embedded engineers is simply the desire to use the well-trying family of protocols and application packages developed for the Internet within their local networks. These protocols are collectively referred to as Internet Protocols, IP or, from the name of the most commonly used - TCP/IP. This name is sometimes loosely used to encompass application packages and other more advanced protocols based on TCP/IP such as File Transfer Protocol (FTP, and its sibling TFTP Trivial File Transfer Protocol), Mail (SMTP – to send POP3 – to receive) and Web Browsing (HTTP).

We will start by giving a few examples where embedded systems might use these protocols so that you can be considering typical scenarios that will later be used to highlight the relevance of different features of the protocols.

Example 1 Micro to Micro



In these examples the application will run on a microprocessor or increasingly on multiple microprocessor configurations used either locally to share the load, or over longer distances to put intelligence close to sensors or actuators.

Example 2 Micro to PC

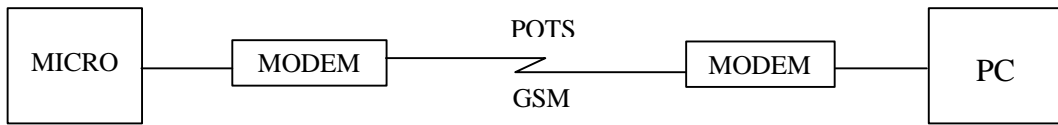


There are two reasons why we may be looking at this -

Where the micro performs data acquisition or control and the PC data processing - in which case the PC's presence may be a necessary part of the total system.

Where the PC may be used as a convenient terminal to support complex set-up or diagnostic capabilities – without TCP/IP these PC's would typically run an RS232 terminal emulator to communicate with a custom built command line interpreter software module within the application.

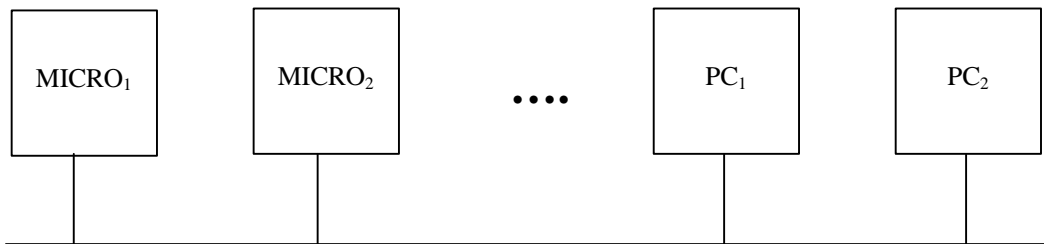
Example 3 Remote dial-up Micro



In this example the micro is remote from the PC and the connection is made via Plain Old Telephone Services (PTOS) or GSM radio. This could simply replace the local link for infrequent maintenance functions or we can think up more automated systems where the PC dials a whole array of outstations to gather information whenever it needs it.

The normal interface to a modem is via the RS232 port of the micro using the Hayes “AT” command sequences to control the modem. Radio modems with RS232 interfaces and similar command sets are now available that allow low cost units to link to the GSM radio network and these can then communicate via TCP/IP. The GSM network is typically less reliable than cable but has advantages of portability, low usage costs and minimum installation charges.

Example 4 Multi Drop Networks



Ethernet or other multi drop protocols

As Ethernet cards have become standard on PC’s their price has fallen dramatically. The availability of Ethernet chips that are easily interfaced to both 8 and 16 bit micros, at less than \$10 each, along with the ease and familiarity of using networked PC’s will make this an increasingly popular communication media for industrial systems.

It is also clear that the software components required to control an Ethernet network could be easily adapted to operate multi station Transport Layers such as RS485, CAN and other proprietary multi drop links by the provision of appropriate hardware drivers.

2. Transmission Control Protocol

When two computers wish to exchange information over a network, there are several components that must be in place before the data can actually be sent and received. Of course, the physical hardware must exist, which is typically either an Ethernet link or a serial communications port for direct or dial-up connections.

Beyond this physical connection, however, computers also need to use a *protocol* that defines the parameters of the communication between them. In short, a protocol defines the "rules of the road" that each computer must follow so that all of the systems in the network can exchange data. One of the most popular protocols in use today is TCP/IP, which stands for Transmission Control Protocol/Internet Protocol.

By convention, TCP/IP is used to refer to a suite of protocols, all based on the Internet Protocol (IP). Unlike a single local network, where every system is directly connected to each other, an *internet* is a collection of networks, combined into a single, virtual network. The Internet Protocol provides the means by which any system on any network can communicate with another as easily as if they were on the same physical network. As our applications may run on a wide range of computers from 8 bit to 32 we will refer to each system on the network as a CPU, be it a large Unix Server (host), a PC or a microprocessor irrespective of its power. Each CPU is assigned a unique 32-bit number, which can be used to identify it over the network. Typically, this address is broken into four 8-bit numbers (octets – values 0 – 255) separated by periods. This is called *dot-notation*, and looks something like "192.43.19.64". The first one, two or three of these octets are used to identify the network that the CPU is connected to, and the remainder identifies the CPU itself. This gives three "classes" of addresses, referred to as "A", "B" and "C". The rule of thumb is that class "A" addresses are assigned to very large networks, class "B" addresses are assigned to medium sized networks, and class "C" addresses are assigned to smaller networks (networks with less than 250 hosts). Addresses starting 224-255 are reserved for testing.

When a CPU sends data over the network using the Internet Protocol, it is sent in discrete units called *datagrams*, also commonly referred to as *packets*. A datagram consists of a header followed by application-defined data. The header contains the addressing information that is used to deliver the datagram to its destination, much like an envelope is used to address and contain postal mail. And like postal mail, there is no guarantee that a datagram will actually arrive at its destination. In fact, datagrams may be lost, duplicated or delivered out of order during their travels over the network. Needless to say, this kind of unreliability can cause a lot of problems for software developers. What's really needed is a reliable, straightforward way to exchange data without having to worry about lost packets or jumbled data.

To fill this need, the Transmission Control Protocol (TCP) was developed. Built on top of IP, TCP offers a reliable, full-duplex byte stream that may be read and written to in a fashion similar to the use of a serial port. The advantages of this are obvious: the application programmer doesn't need to write code to handle dropped datagrams, and instead can focus on the application itself. And because the data is presented as a stream of bytes, existing code can be easily adopted and modified to use TCP.

TCP is known as a *connection-oriented* protocol. In other words, before two programs can begin to exchange data they must establish a "connection" with each other. This is done with a three-message handshake in which both sides exchange packets and establish the initial packet sequence numbers (the sequence number is important because, as mentioned above, datagrams can arrive out of order; this number is used to ensure that data is received in the order that it was

sent). When establishing a connection, one program must assume the role of the *Client*, and the other the *Server*. The Client is responsible for initiating the connection, while the Server's responsibility is to wait, listen and respond to incoming connections. Once the connection has been established, both sides may send and receive data until the connection is closed.

TCP is therefore ideal for situations where the link may be noisy and data integrity is not checkable at the application level. Of course, this error checking and recovery means that the TCP option code is larger but then the application does not need to do further checking. The application does however have to cope with the situation where TCP times out and no data gets through such as line breaks or failure of the other processor. If using IP is like sending a letter, with no knowledge of whether it got to the recipient, TCP is like a telephone conversation – while you are both still talking you know that the message has got through.

3. User Datagram Protocol

Unlike TCP, the User Datagram Protocol (UDP) does not present data as a stream of bytes, nor does it require that you establish a connection with another program in order to exchange information. Data is exchanged in discrete units called datagrams, which are similar to IP datagrams. In fact, the only features that UDP offers over raw IP datagrams are port numbers and an optional checksum.

UDP is sometimes referred to as an *unreliable protocol* because when a program sends a UDP datagram over the network, there is no way for it to know that it actually arrived at its destination. MicroNet's main application area is use on microprocessors that may have limited resources. It is therefore all too possible that a slow micro may not be able to keep up with a fast PC sending multiple UDP packets and that, without a handshake to limit the speed to that of the slowest device, packets may be lost. This means that the sender and receiver must typically implement their own application protocol on top of UDP. Much of the work that TCP does transparently (such as acknowledging the receipt of packets, re transmitting lost packets, checking the order in which packets are received and so on) must be performed by the application itself.

With the limitations of UDP, you might wonder why it is used at all. Well UDP has the advantage over TCP in three critical areas: code size, transmission speed and packet overhead. To make TCP a reliable protocol, it goes to great lengths to insure that data arrives at its destination intact, and as a result it exchanges a fairly high number of packets per Kbytes of data. UDP doesn't have this overhead, and is, therefore considerably faster than TCP. In those situations where speed is paramount, or the number of packets sent over the network must be kept to a minimum, UDP is the solution. A further advantage of UDP is that it can be used to send "One To Many" messages.

Examples of where UDP might be adequate is in a system that reported air temperatures every second – if one was missed the receiving CPU might reasonably assume the temperature had not changed – obviously if it did not get data for 6 hours that assumption would not be valid! If the CPU has a mechanism

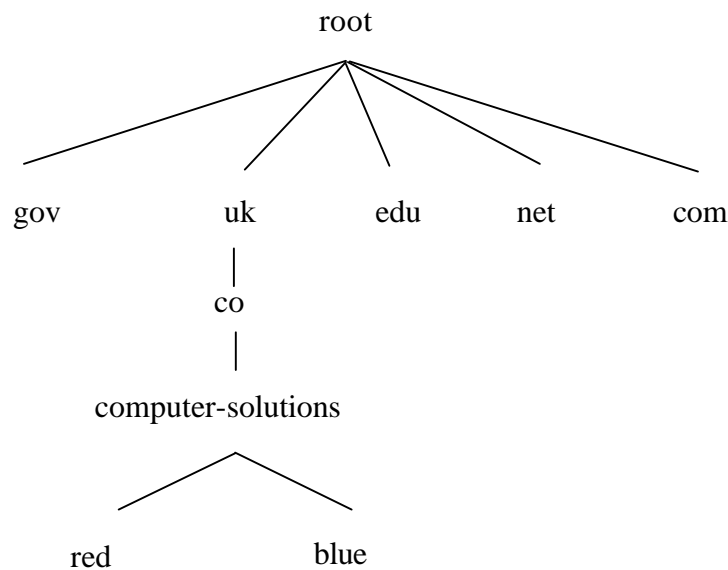
for requesting a particular data point then in effect the application has its own built in error recovery mechanisms and UDP's other advantages might outweigh TCP's security.

4. Making the connection

4.1. IP Address and Hostnames

In order for an application to exchange data with a remote process, it must have several pieces of information. The first is the IP address of the CPU that the remote program is running on. Although this address is internally represented by a 32-bit number, it is typically expressed in either dot-notation or by a logical name called a *hostname*. MicroNet currently needs each micro to either be hard coded with an IP address in dot notation or be allocated an IP address and has no facilities for using hostnames.

Like an address in dot-notation, hostnames are divided into several pieces separated by periods, called *domains*. Domains are hierarchical; with the top-level domains defining the type of organization that network belongs to, with sub-domains further identifying the specific network.



In this figure, the top-level domains are "gov" (government agencies), "com" (commercial organizations), "edu" (educational institutions) "net" (Internet service providers) and a number of country specific abbreviations "UK" standing for the United Kingdom. The *fully qualified domain name* is specified by naming the host and each parent sub-domain above it, separating them with periods. For example, the fully qualified domain name for the "red" host would be "red.computer-solutions.co.uk". In other words, the system "red" is part of the "computer-solutions" domain (a company's local network), which in turn is part of the "co" domain (a domain used by registered companies) which is part of the UK domain which contains all domains registered in the United Kingdom.

In order to use a hostname instead of a dot-address to identify a specific system or network, there must be some correlation between the two. This is accomplished by one of two means: a local host table or a name Server. A host table is a text array that lists the IP address of a host, followed by the names that it's known by. On your PC you will find the host table in a file such as C:\Windows\hosts and you can edit it to allow your PC to communicate with micros running MicroNet by name and not by number.

A name Server, on the other hand, is a program running somewhere on a network which can be presented with a hostname and which will return that host's IP address.

4.2. Service Ports

In addition to the IP address of the remote CPU, an application also needs to know how to address the specific program on the CPU that it wishes to communicate with. This is because large processors running TCP/IP will typically be multi-tasked and running a number of different links. This is accomplished by specifying a *service port*, a 16-bit number that uniquely identifies an application running on the CPU.

On the PC instead of numbers service names are usually used. Like hostnames, service names are usually matched to port numbers through a local file, commonly called C:\Windows\Services. This file lists the logical service name, followed by the port number and protocol used by the Server.

A number of standard service ports and names are used by Internet-based applications and these are referred to as *well-known services*. These services are defined by a standards document and include common application protocols such as FTP, POP3, SMTP and HTTP. Port numbers 1 – 1023 are reserved for well-known services. Client port numbers are called *ephemeral ports* and values between 1024 and 5000 are usually used. When setting up an application specific link (say a straight TCP link between a micro and a PC) you should avoid using the well-known services range. Remember that a service name or port number is a way to *address* an application running on a remote host. Because a particular service name is used, it doesn't guarantee that the service is available, just as dialling a telephone number doesn't guarantee that there is someone at home to answer the call.

4.3. Sockets

MicroNet provides an API that will allow the application programmer to send individual TCP or UDP packets. However, it is anticipated that most users will employ the socket interface which is significantly simpler to program.

The previous sections described what addressing information a program needs to communicate over a TCP/IP network. The next step is for the program to create what is called a *socket*, a communications end-point that can be likened to a telephone. However, creating a socket by itself doesn't let you exchange information, just like having a telephone in your house doesn't mean that you can talk to someone by simply taking it off the hook. You need to establish a

connection with the other program, just as you need to dial a telephone number, and to do this you need the *socket address* of the application that you want to connect to. Once a connection has been established to a socket in the addressee the applications programmer need only consider the data to be read/written.

Using the socket interface is very simple, the program uses *mn_open* to make the connection and get a socket.

```
socket = mn_open(dest_ip, src_port, dest_port,  
                client, TCP, recv_buff, buff_len);
```

then *mn_send* or *mn_receive* to pass the data

```
status = mn_send(socket, msg_ptr, msg_len);  
  
status = mn_rcv(socket, buff_ptr, buff_len);
```

Once the connection is no longer required

```
status = mn_close (socket);
```

4.4. Blocking vs. Non-Blocking Sockets

One of the first issues that you'll encounter when developing your Socket application on a PC is the difference between blocking and non-blocking sockets. Whenever you perform some operation on a socket, it may not be able to complete the operation immediately if not then when does it return control back to your program? For example, a read on a socket cannot complete until the remote host has sent some data. If there is no data waiting to be read, one of two things can happen: the function can wait until some data has been written on the socket, or it can return immediately with an error that indicates that there is no data to be read.

The first case is called a *blocking socket*. In other words, the program is "blocked" until the request for data has been satisfied. When the remote system does write some data on the socket, the read operation will complete and execution of the program will resume. The second case is called a *non-blocking socket*, and requires that the application recognize the error condition and handles the situation appropriately.

For historical reasons, the default behaviour is for socket functions to "block" and not return until the operation has completed and this is how MicroNet's sockets operate. However, using blocking sockets in Windows can introduce some special problems and you are warned that a different strategy will need to be adopted in any Windows application that communicates with a micro.

4.5. Client-Server Applications

Programs written to use TCP are developed using the *Client-Server model*. As mentioned previously, when two programs wish to use TCP to exchange data, one of the programs must assume the role of the Client, while the other must assume the role of the Server. The Client application initiates what is called an *active*

open. It creates a socket and actively attempts to connect to a Server program. On the other hand, the Server application creates a socket and passively listens for incoming connections from Clients, performing what is called a *passive open*. When the Client initiates a connection, the Server is notified that some process is attempting to connect with it. By *accepting* the connection, the Server completes what is called a *virtual circuit*, a logical communications pathway between the two programs. Data may now be interchanged between the two usually as a result of requests that originate at the Client. It is useful to think of the Client as being in control of the Server when looking at an overall design. Many of us automatically think of large computers when we say Server but it is this control aspect that determines whether the embedded micro or the CPU it is connected to is the Server.

It's important to note that in order to keep overheads low when MicroNet accepts a connection to a socket (if the port numbers match) it allocates it to the socket that was listening. If required MicroNet will open additional sockets as part of the Server's function – for example Web Browsers may use multiple sockets, one for each element on the page. On larger CPUs such as the TCP/IP stack running on a PC, without the constraints imposed by limited resources, the original socket may remain listening for additional connections and the link may communicate via a new socket. When the Server no longer wishes to listen for connections, it closes the original passive socket.

To review, there are five significant steps that a program, which uses TCP, must take to establish and complete a connection. The Server side would follow these steps:

- Create a socket.
- Listen for incoming connections from Clients.
- Accept the Client connection.
- Send and receive information.
- Close the socket when the Client has finished or when the Server wishes to no longer be available.

In the case of the Client, these steps are followed:

- Create a socket.
- Specify the address and service port of the Server program.
- Establish the connection with the Server.
- Send and receive information.
- Close the socket when finished, terminating the conversation.

Only steps two and three are different, depending on whether it is a Client or Server application.

5. Drivers, Layers and Stacks

Most engineers are familiar with the concept of a driver - lets define it loosely as “the combination of function calls and Interrupt Service Routers necessary to operate a piece of hardware at the message level”. These functions (with a bit of documentation) allow someone unfamiliar with the details of the hardware to write application programs that will use the device. However, there is usually only one way to use a driver (OK two, read or write). A stack is much more, as it must be a set of co-operating programs written to work together in many different combinations. Their underlying structure is provided by a commonly agreed set of protocols (or message standards) each level of the stack hiding the messy detail of the level below as we become more and more application oriented. The need for something more complex than a conventional driver also arises from the nature of communications networks in that we may have to sustain multiple interactions going on at the same time. On a heavily loaded system the Stack will make extensive use of a Real Time Operating System (RTOS) if it is available to simplify its handling of the slow I/O devices.

The TCP/IP stack is broken down into layers as shown below.

Layer	Examples	Function
Client Application	DHCP, FTP, WEB, YOUR APP	Do Something Useful
Transport	TCP, UDP	Send a Message
Internet	IP, PING, ARP	Component parts Connect, Data, handshake
Link	PPP, SLIP	Customise for & talks to specific hardware
Physical	RS232, Ethernet	The hardware chip so voltage, frequency, signalling techniques

5.1. Physical Layer

The most commonly used physical connection to a microprocessor is the RS232 serial connection. For long distances this is converted from its absolute voltage, bit signalling form to a voltage independent, frequency modulated form by a modem. For high speed connections between PC's the most common form of connection is 10Mbit Ethernet using CSMA/CD (Carrier Sense, Multiple Access with Collision Detection).

All these are examples of different “Physical Layers” and their definitions include voltages, signalling mechanisms and timing details. Each different mechanism will have particular characteristics that require data to be specially formatted for it and this is done by the link layer. The link layer handles such problems as how to

manage both Escape and Control characters that may perform functions as well as their binary equivalent as part of data.

5.2. Link Layers - SLIP & PPP

SLIP stands for Serial Line Internet Protocol and is one of the simplest conventions for sending TCP/IP packages along a serial line. You would typically use it on a low noise RS232 link between two fixed processors.

It suffers from the following problems –

- Each end assumes it knows the identity of the device at the other end.

- Only a single conversation can go on at one time.

- There is no protection against data corruption at this level and it relies on the levels above to detect transmission errors.

However the overhead added by SLIP is minimal.

PPP stands for Point to Point Protocol and is the link layer protocol most commonly used for TCP/IP package communications over modems. If you dial an ISP (Internet Service Provider) to link to the web or send your emails you are probably using PPP.

PPP is superior to SLIP as its message structure contains a word (called the Frame Check Sequence or FCS.) that provides error detection. It also defines a number of options including a protocol for configuring and testing lines.

The PPP Client on dialling may be allocated an IP address by the Server. This is necessary as many Servers (such as those used by Internet Service Providers) allocate a different IP address every time you dial them (dynamic IP addressing). A simple name and password dialogue can then take place, which is supported in MicroNet by simple character recognition/output data transfer. MicroNet will also support a password option built into the protocol – Password Authentication Protocol (PAP). There are a number of other PPP options which the Server may ask if they are supported but which, if not available, will not be used. These options are supported by the RTIP stack and include CHAP CSLIP and Van Jacobson Compression.

MicroNet supports SLIP and optionally PPP using the integral UART port of the microprocessor.

5.3. Modems

The MicroNet Modem functions are a part of the core TCP/IP stack and can support both Dial in and Dial out functions along with commands to simplify logging into an ISP. All necessary Modem functions must be executed prior to the start of a PPP dialogue.

5.4. Link Layers – Ethernet

Each Ethernet interface card or chip has a 48bit unique physical address called its MAC or OUI. The Ethernet Link Layer includes two protocols that are used to perform translations between IP addresses and physical addresses. Protocols called ARP (Address Resolution Protocol) and RARP (Reverse Address Resolution Protocol) perform the translations. RARP is not currently supported by MicroNet

If you ask to talk to IP xxx then ARP will look in its table for the MAC that it knows is associated with that IP address. If it fails to find a MAC associated with the IP address then a message is sent to all systems on the network asking if they are the IP xxx. If one successfully acknowledges with its MAC then the table is updated and an address returned. It is this MAC address that then appears in the Ethernet packet as the destination address.

MicroNet provides Ethernet support for a number of chips the most popular of which is the Cirrus Crystal CS8900 Ethernet controller. This is a nice chip that is easy to interface to embedded processors with substantial transmit and receive buffers that take much of the load of running Ethernet off the processor. We can provide circuit diagrams showing both 8-bit I/O mapped and 16-bit memory mapped configurations. We also support the Realtek 8019, the SMSC 91c111 and the SMCS 9196.

A similar strategy or a simple fixed table could be adopted for other networks such as RS-485 or CAN where a translation was required between an IP address and a protocol specific system identity.

6. Higher Level Protocols

The basic TCP or UDP protocols can be used to exchange messages between CPUs but in addition there are many other protocols built upon TCP and UDP that can be used in embedded systems. These are also sometimes referred to as Application Packages.

6.1. File Transfer Protocol (FTP)

This application copies a complete file from one CPU to another – it does not allow one CPU to read individual records at a time from files held on the other CPU (NFS Network File System does that). As most embedded systems do not include a hard disk, MicroNet includes with this option a simple virtual file system that can be ROM, RAM or flash based. In this virtual file system the programmer manages the location of the data and the file system acts as a look up table for named data sets. This file system puts all file names into a single root directory.

FTP operation requires an underlying TCP layer. MicroNet currently provides an FTP Server that allows a remote PC to control the micros operation. Having first checked the incoming file request against a table of users and passwords the Server will read and write files from and to the micro's file system as directed by

the PC. An FTP Client running on the micro which would take control of the interaction is not currently available – call Computer Solutions Ltd if you are interested in this facility.

6.2. Getting an IP address (BOOTP, DHCP & TFTP)

If a device on a network is to successfully communicate with the other devices on the network it must have a name – its IP address. There are many practical reasons why it is better to have the network allocate an IP address rather than have it built into the device ROM or manually set up. As has already been explained PPP Servers may allocate an IP address whenever a connection is made. When using multi drop links such as Ethernet, the most common ways of allocating IP addresses are BOOTP and Dynamic Host Configuration Protocol (DHCP) Servers. For either of these a Server running on the network is required (MicroNet only provides Clients, RTIP provides Client and Server options, PC based Servers are commercially available).

BOOTP is the simpler protocol, the Client requests an IP address and at the same time can ask for a named file transfer. This file is often used to load the application code into the device. DHCP is more sophisticated in that the Client can ask for an IP address which will only be valid for a limited time, after which it will have to re-apply. The time can be infinite or if required the device can apply for a time extension. DHCP can also request a file transfer which is done with TFTP.

In order too minimise the code space required in the embedded device before an initial file transfer is made, these protocols use the simpler TFTP Client for this file transfer. Unlike FTP, TFTP does not have any password handling and uses UDP rather than TCP for the data transfers. It makes little attempt at optimising transfer times but does implement a handshake to overcome UDP's inherent unreliability. MicroNet implements TFTP's read file function in order to support BOOTP but does not implement the write file functions. Once the file is received it is up to the application program to handle any system specific actions required should it be desirable to start executing the files contents or saving it to flash memory.

6.3. Web Server (HTTP)

While it may sound like a major commitment of resources to put a “Web Server” which we usually associate with large UNIX systems on a micro in fact the Web Server is significantly smaller than the rest of the TCP/IP stack. When used with a windows browser such as Explorer, it adds significant capabilities to that micro. The MicroNet Web Server option also includes the virtual file system that holds the web pages to be requested/browsed.

By linking a PC running Windows and a web browser to a micro running the Web Server we immediately have a sophisticated color Graphical User Interface (GUI) whose use will be immediately familiar to a technically literate audience.

The MicroNet Server can supply the full range of HTML information including fonts, colors graphics (.gif, .jpg), tables, frames, borders, buttons, check boxes, drop down menus, radio buttons, text boxes and Java applets.

Of course, these web pages can contain hyperlinks that point to other pages held in the MicroNet Web Servers file system making it possible to manage complex dialogues between the user and the microprocessor (ie. operator help manuals). To reduce load on the micro this manual could be held on the PC or on the Internet and accessed via an off micro hyper-link.

If you are familiar with the type of Server used by your ISP to host your companies web pages, you will know that it is possible to insert statements into a web page that force programs to run on the Server when the page is requested/browsed. These are called Server side programs and good examples are CGI scripts and hit counters.

In the same way MicroNet provides the ability to link to user supplied functions that execute on the microprocessor. Typically these might be

A function that is executed to start/stop a machine when a button was pressed on a web page.

A function that was executed while the web page was being assembled and which picked up a variable in the microprocessor, converted it to an ASCII string and put it into the resulting web page sent to the browser. This would display the current value of that variable every time the web page was requested.

A function that accepted a character string when a text box is filled and a web "Submit" operation occurs. The function will typically convert the text string to a value, validate it and then use it to set some application variable on the microprocessor.

Down at the microprocessor level if the file system is held in RAM or flash it is possible for the web pages themselves to be dynamically created or modified. Simple examples might be to have a number of different graphical representations of a tank of liquid and to change the image name to point to one of eleven images showing - empty, 10%, 20% ... full. Another example is to have a function that changes the html text that set the colour of the names of those items in alarm states.

A web browser is not available within the MicroNet suite as its complex GUI is not considered viable on the small memory range micros that it is targeted at. On larger processors the Webster embeddable browser and PEG the embedded windows package will operate with the full RTIP and CMX-TCPIP stacks.

6.4. JAVA and beyond

It is even possible to use JAVA when the Web Server is an 8051! How? Well the JAVA scripts or applets can execute on the browsing PC even though they are held within web pages or as files on the microprocessor. In this way a stream of characters representing data logged by the micro could be requested and

converted in the PC into a graph by a JAVA applet, which the PC requested from the microprocessor. This JAVA applet would be sent to the PC, by a browser request, as part of the normal processing of an .html file. The JAVA applet once running would interact with the user to ask for the time interval over which he wished to view the data. The JAVA applet would then send a text string to the microprocessor specifying the required date range and accept a binary file with the data from the microprocessor. Or the applet can set up a TCP/IP link for more complex interactions. It would then be up to the JAVA applet to format the graph.

Please note this example is not provided as a MicroNet demonstration but has been outlined here to show the range of possibilities that exist once a Web Server is available, it is certainly possible but also non trivial.

6.5. Simple Mail Transfer Protocol (SMTP)

Based on TCP this Client application allows a micro to send an email to any Server that supports it (most ISP's will accept SMTP). The MicroNet SMTP option is currently limited to one recipient, per email and one us-ascii text attachment <998 bytes long.

6.6. Post Office Protocol (POP3)

POP3 Client is the most common way of receiving emails and is supported by most ISP's. It allows the Client to request the number of emails waiting for it, their origin, size and the subject line of each message. It then can retrieve selected messages and delete them from the Server. MicroNet does not currently support this option – call Computer Solutions Ltd if you are interested in this facility.

6.7. Simple Network Management Protocol (SNMP)

This protocol based on UDP is used to manage devices attached to a network. It is typically used for devices such as routers and Servers. It provides network managers with the ability to access and change network settings as well as to be informed when specific events occur. At first sight this would seem to be the ideal way to communicate with embedded control or data acquisition devices, however, the needs of network management are so specific (and the budget of those managing large networks are so large) that we have not been able to successfully identify any network management tools that appear to be applicable to the general embedded TCP/IP market.

SNMP is not yet available for MicroNet, call Computer Solutions Ltd if you have an interest in this option.

6.8. Ping

Not a protocol, but a useful program (usually found in C:\Windows on your PC) that will send a message to a host on the network and time the round trip message time. MicroNet can respond to a ping but cannot initiate one. This is a very

useful diagnostic aid used to check the existence of stations on the network. The only part of ICMP (Internet Control Message Protocol) that MicroNet supports is the Ping response.

7. Choosing a Protocol

So back to our examples.....

Example 1 Micro to Micro



For the Micro to Micro link we can use the socket interface and the TCP protocol to ensure reliable communications between the micros over long distance serial links or faster short range Ethernet. If the application has facilities for device polling with good message handshakes we may choose to use UDP. Another application where UDP should be considered is voice or video over IP. UDP's lower overhead will improve throughput and the application does not have time to recover poor data but the human observer is good at ignoring occasional blips.

Example 2 Micro to PC

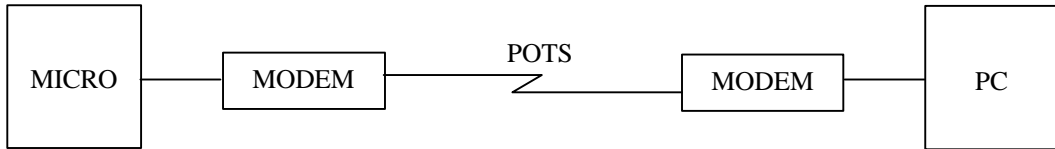


The same arguments apply to Example 2 where we are talking to a PC but be warned that the micro may be easily overwhelmed by a flood of UDP messages pumped out by a 1700 MHz PC. If large amounts of data are to be exchanged or the application structure suits it we might use FTP, the File Transfer Protocol, which is an optional MicroNet module.

In Example 2b where the PC is used as a terminal providing a full time operator's consol, it would be possible to write a GUI application on the PC in C++ or Delphi that communicated via the Windows Socket interface to TCP/IP. But depending on the requirement and particularly if the link is more transient (set-up / diagnostics) then the possibility exists of replacing the conventional ASCII character based keyboard dialogue with a Web Server that provides a convenient ready-made Windows front end. Using this Web Server it is possible to set or read variables and to control the application by commanding particular functions to be executed. This may also be more easily customised to suite a range of users and applications. Note that to run Explorer without modems at each end requires

the PC to run a Null Modem driver – availability of this for the common operating systems is currently under investigation.

Example 3 Remote dial-up Micro



Because of the modem lines low reliability TCP will be the protocol of choice for all applications. This system could use a custom PC based application program communicating via Sockets or TCP/IP datagrams but is also ideally suited to running an FTP Server if you just want to get in values, say from a data logger. Using a Web Server on the micro makes remote operator control very simple, as all that is required is to run a conventional web browser such as Internet Explorer on the PC and PPP on the micro. But there is much more we can do - for example using SMTP it would be possible for the micro to dial an ISP (also needs PPP) and leave an email each night giving the day's results. A simple procedure would then allow all emails from a number of machines to be merged. The micro might send an email to indicate alarm conditions and there are services that will convert such an email into a text message sent to a designated mobile phone or to a pager. Emails from a central computer could be used to set new limits or, using an attachment, to provide new operational software to be loaded into the micro's Flash.

Example 4 Multi Drop Networks



Ethernet or other multi drop protocols

All the applications described for the last example apply when we have one micro and one PC but in addition we now have a whole new set of system possibilities.

Distributed applications with micros sharing data

Multiple PC's using Web Servers able to inspect any part of the plant (but do you really want anyone on the network to be able to control that Steel Mill press?)

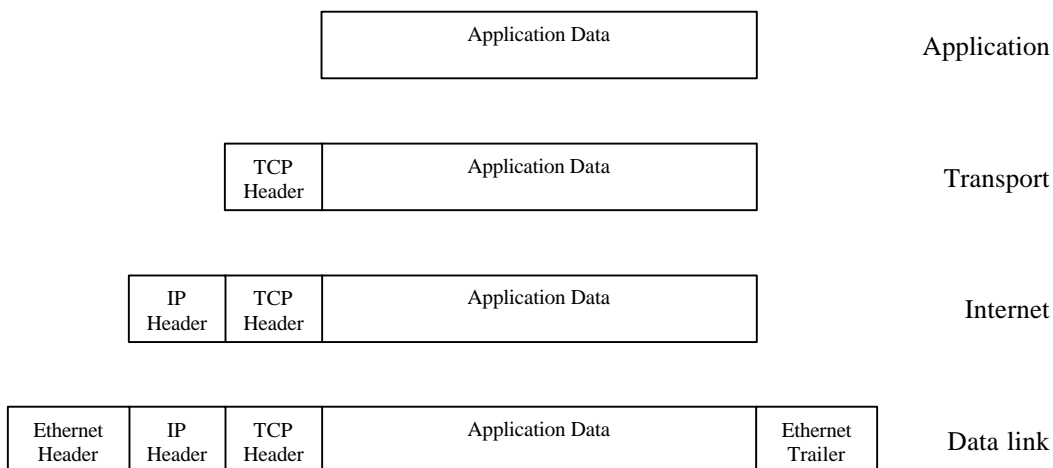
Care must however be taken as Ethernet is non deterministic – there have been discussions on the Real Time Newsgroup where people have said things such as “Ethernet is fine for control so long as you don't load the line by more than 10% of its bandwidth.” Can you guarantee that your customer will put in a fresh line

for this application or will he be unable to resist putting it onto the same Local Area Network as the sales office PC's? And what will be the effect on your application when someone in Marketing transfers that high resolution Photoshop image of the company's latest full-page advertisement across the network at a critical point in your control loop?

8. CMX MicroNet Features and limitations

CMX-MicroNet is a TCP/IP implementation designed to work with processors that have a small amount of ROM and RAM. It is also highly configurable, enabling the developer to further minimize resource usage.

Many stacks including those sold for use with larger CPUs are quite profligate in their use of resources – both CPU time and RAM. In particular when processing information down the stack from application to high level Protocol to low level Protocol to driver each layer may add heading and trailing data as shown on the next page.



Many do this by creating a new buffer, loading the header, copying the data it is given into the buffer and then adding any trailing data. This buffer is then passed down to the next level of protocol. The result is a lot of CPU cycles used to copy characters and a high RAM requirement. Both MicroNet and RTX have a “No Copy” mode which avoids this unwelcome overhead.

CMX-MicroNet supports up to 16 sockets using TCP or UDP. All 16 sockets can be PPP sockets, SLIP sockets or Ethernet sockets, but not a mixture of PPP sockets, SLIP sockets and Ethernet sockets at the same time. The RS232 link can either be a direct cable link or through a modem.

In order to reduce code and data sizes a number of departures were made from the full RFC standards in that not all of the options are supported. TCP options are ignored, in particular it should be noted that it does not support sliding windows and it will not accept an out of order packet – it does not acknowledge higher number packets until all intermediate packets are received. Other limitations in the options available have been identified in the description of these protocols.

9. Integrating CMX-MicroNet

Unlike most TCP/IP stacks and in order to minimise the memory footprint, CMX-MicroNet does not require nor include a real time operating system (RTOS). The question therefore arises as to what programming strategies are available for building an application that uses the CMX-MicroNet functions?

9.1. Using TCP/IP for Communications

In many cases the stack will be used to provide communication via PPP, UDP or TCP links to other computers as part of an application. If this is a new project being written from scratch it is easy to design the application code so that it does not conflict with the MicroNet packages requirements. If you are adding MicroNet on top of an already existing application then this will be a little more complex. In either case it is valuable to know some more about how MicroNet allocates CPU time.

Firstly, the peripherals such as UARTS or Ethernet chips are driven under interrupts with this “driver code” putting its data into or getting it from character buffers. So if your application can be driven completely from interrupts it can easily co-exist with a high level routine that performs the necessary TCP/IP code.

Secondly, it is important to understand which MicroNet functions retain control until I/O is completed and which return without having to wait for any I/O.

UDP send returns once the buffer is set up it does not wait

UDP receive does not return until the input transmission is completed

TCP send sets up the buffer to be transmitted and immediately goes to waiting to receive the acknowledgement

TCP receive waits for the message and returns once it has started to send an acknowledgement

So in effect the system either returns control after setting up a write or it is sitting waiting for a read to complete. While it is waiting for a read to complete it polls the received byte count. Each time it does so unsuccessfully (message not yet complete) it executes the callback routine (*mn_app_recv_idle*) so any high level code put into that routine can be executed during the time that the TCP/IP interaction is taking place.

The Third alternative is to reduce the receive message time-out so that any receive functions return shortly after the call has been issued. CPU time can then be used for application code but in this case it is necessary to add extra code to establish when a receive function is complete and to identify a true error such as a broken line.

9.2. Using a Server

When using a Server such as the HTTP Web Server there is only a single application function *mn_web_Server* and CMX-MicroNet only drops out of

that should the line go down. This is fine for the fully interrupt driven application but especially if retro fitting CMX-MicroNet to a current application another form of CPU sharing needs to be used.

Within each of the Servers a callback routine is provided that will execute at any time that the Server is inactive. For example, for the Web Server this is *mn_app_http_Server_idle*.

The application code can easily be placed in this idle routine, however care should be taken that any single execution of the idle routine does not include long time delays, as that will reduce the perceived responsiveness of the Server.

9.3. CMX-MicroNet with an RTOS

CMX-MicroNet has been designed to operate successfully under an RTOS should one be available. This could be a homegrown RTOS, CMX-RTX (available for most CPUs) or any proprietary RTOS. If this is being done, the timer ISR will need to be recoded to make use of the RTOS clock. CMX-MicroNet only supports one socket call in operation at a time so it is not possible for multiple tasks to be executing socket calls at the same time. The best way of preventing this is to perform all the CMX-MicroNet functions from within one task dedicated to the communications functions. Should you need more complex multi-task structures then you should probably consider using the full CMX-TCP stack which can support multi tasked access to multiple active sockets.

9.4. Allocating Addresses

When doing the detailed designing of a product you will soon get to the question “But what address does this unit have and how/when do I set it?” If you are using PPP into a dial up Server then it is easy as you will be allocated an IP number whenever you dial in.

When using multidrop networks remember that the IP addresses are all local to the network they are on so if you have multiple outstations running on a dedicated network you can build the IP addresses into the outstation’s ROM and set the system up to run with those addresses. This is simple but naïve – if you have a spare device you will need to blow fresh PROMs before you can use it – OK put the address on jumpers or in flash with some configuration procedure. If you have a PC or RTIP system on the network then it is much better to set it up with a BOOTP or DHCP Server so that addresses are allocated automatically.

If you are using Ethernet then you must allocate each unit with a unique 48 bit address (MAC) – again fine if your equipment is the only thing on the network but as soon as someone puts a PC with a network card on it you have a potential contention.

MAC addresses are allocated by IEEE and if you plan to produce a significant number of units you should consult them. For small numbers of units it may be worth asking your Ethernet chip supplier for a set of MACs as each chip will require one. As a last resort there are a number of alternative strategies that break the rules but for which the chance of two numbers conflicting are small.

10. Further Reading and browsing

Books

TCP/IP for Dummies Leiden & Wilensky Published by IDE Books, ISDN 0-7645-0063-5 - partly an introduction to managing IT Servers but still a good starting place.

TCP/IP Illustrated Vol. 1 R. Stevens Published by Addison Wesley ISBN 0-201-63346-9 – showing its age (1994 so the web only gets 4 lines in 555 pages) but still the best place to go if you want to know about the details of TCP message structures.

Internetworking with TCP/IP by Douglas E. Comer

RFC 1180 – A TCP/IP Tutorial <http://www.faqs.org/vfcs/>

Web sites

Embedded TCP/IP Grasping the fundamentals – a short article on TCP/IP basics for embedded use

<http://www.embeddedtechnology.com/read/n120000720/195152>

RFC/STD/FYI/BCP that's Request for Comment / Internet Standards / For Your Information / Best Current Practice. These documents are the way the TCP/IP protocol standards and recommended ways of working are promulgated <http://www.faqs.org/rfcs/>

Applications programs

Note that at the time of writing Computer Solutions are still investigating a range of programs that can be used with Internet enabled embedded targets but the following look promising.

Catalyst's SocketWrench is a free, general purpose Windows library that includes ActiveX components which drive the Windows Sockets – So you can insert an object into your C++ or VB application. They also have an even better one that they charge for <http://www.catalyst.com/products/wrench.html>

Borland's DELPHI database, Pascal and C++ based programming languages provides socket calls that can be used on the PC end of applications as well as FTP and SMTP capabilities.

Tal's TCPWedge software claims to input and output TCP/IP data from any IP addresses directly into any Windows 95 or NT application. Excel, Access, FoxPro, Oracle, Wonderware, Intellution, other MMIs, LIMS, SPC programs, control applications <http://www.taltech.com>

Dr. Herbert Hanewinkel's website has useful, low cost, DHCP/Bootp and TFTP Servers available for Windows 9x / Me / NT / 2000 / XP go to <http://www.heha.cjb.net/homee.htm> select "Shop" then "Internet Tools".

11. Common Mnemonics

ARP	Address Resolution Protocol	Translates an internet address into a hardware address
BOOTP	Bootstrap Protocol	For remote booting of diskless devices
CGI	Common Gateway Interface	A standard that allows web pages to invoke and pass data to Server programs
CHAP	Challenge Handshake Authentication Protocol	PPP password that allows both ends to check for a valid connection
CSLIP	Compressed Serial Line Interface Protocol	Compressed version of SLIP
DHCP	Dynamic Host Configuration Protocol	Allocates IP addresses dynamically
DNS	Domain Name Server	A program / computer that converts a domain name into its IP address
FTP	File Transfer Protocol	An application that transfers files across the network
HTML	Hyper Text Mark-up Language	The standard used to create web pages
HTTP	Hyper Text Transfer Protocol	The protocol used to transmit web pages
ICMP	Internet Control Message Protocol	Used to report errors from IP level and above
IGMP	Internet Group Management Protocol	Used when broadcasting to groups that exist across routers.
IMAP	Internet Mail Access Protocol	A relatively new advanced email Server that allows users to hold and manage their emails on the Server.
IP	Internet Protocol	The mechanism for delivering packets across the network
IPv6	The next proposed version of Internet Addressing Protocol	Able to address many more items on a network this extension to IPv4 is still a few years away from normal use.
IPCP	IP Control Protocol	PPP's control protocol
IPSec	Secure Internet Protocol	A secure version of IP
IIOB	Internet Inter ORB Protocol	Protocol for passing object data
ISP	Internet Service Provider	A company that links an end user to the Internet backbone
LCP	Link Control Protocol	PPP option negotiation

MIB	Management Information Base	A table of devices used by SNMP
MIME	Multi Purpose Internet Mail Extensions	Standard for adding diverse types of files to emails
MFS	MicroNet File System	Used by Microsoft OSs to link file systems across networks.
MTA	Mail Transfer Agent	An application for sending emails
MUA	Mail User Agent	An application for creating and reading emails
NFS	Network File System	A protocol for controlling remote file systems
NTP	Network Time Protocol	Used to set the clocks on multiple machines across a network
ORB	Object Request Broker	Standard for object data interchange
PAP	Password Authentication Protocol	PPP password
PING	A diagnostic program	It tests the connection with a specified URL
POP3	Post Office Protocol	A mail protocol for retrieving email that can include attachments
PPP	Point to Point Protocol	Provides links between computers on a network
RARP	Reverse Address Resolution Protocol	Allows a host to obtain its IP address by sending its hardware address
RIP	Router Internet Protocol	Used to control Routers
SLIP	Serial Line Interface Protocol	Simple protocol often used with RS232
SMTP	Simple Mail Transfer Protocol	Protocol for sending and receiving emails. Messages must be text
SNMP	Simple Network Management Protocol	Protocol used by computers that monitor and manage network activity to communicate with one another and the computers they are monitoring Comes in 3 versions: v1 Simple – Simple v2 Some Security v3 Advanced Security & Encryption
SSL	Secure Socket Layer	Sits below TCP, UDP and uses encryption and certification to provide commercial levels of security.
TCP	Transmission Control Protocol	Puts data into packets for delivery by IP
TELNET		An application that allows logging into remote computers

TFTP	Trivial File Transfer Protocol	TFTP is a simpler version of FTP which does not require valid username and password.
UDP	User Datagram Protocol	Protocol at the internet layer. UDP does not guarantee reliable, sequenced packet delivery. So if data does not reach its destination, UDP does not retransmit as TCP does