Good news, everyone!

mTCP

Version: 2015-07-05

M. Brutman (mbbrutman@gmail.com)
http://www.brutman.com/mTCP/

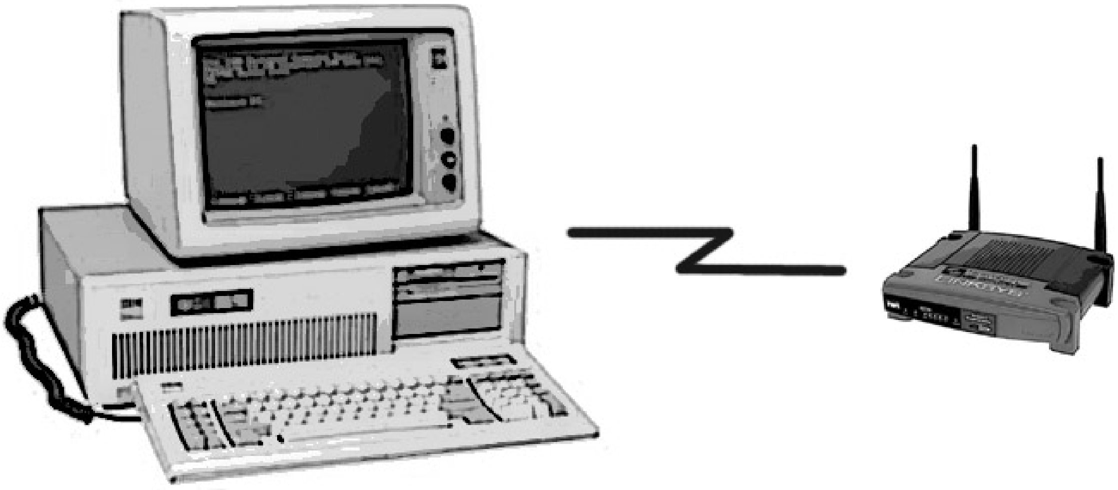


Table of Contents

Introduction and Setup

mTCP Applications

Advanced Topics

Introduction and Setup

Introduction

What is mTCP?

mTCP is a set of TCP/IP applications for personal computers running DOS. The applications are designed to run well on older 16 bit x86 compatible computers. Applications include:

- a DHCP client for auto-configuring your machine for use on your network
- an FTP client for transferring files across your network (or further!) at high speed
- a high performance FTP server that supports multiple connected clients, anonymous FTP and more
- HTGet for fetching files or web pages from web servers
- HTTPServ for serving files on the web using HTTP 0.9, 1.0 or 1.1
- IRCjr, an Internet Relay Client for chatting on IRC networks
- the Netcat utility for sending and receiving data easily across the network
- the Ping command which is used for checking basic network connectivity
- the PktTool utility for working with packet drivers and sniffing packets on the network
- SNTP, a Simple Network Time Protocol client for synching your clock with Internet servers
- SpdTest for figuring out how fast your system/network card is
- a Telnet client for connecting to Unix systems, BBS systems, and anything else that supports Telnet

The TCP/IP code is compiled into each application allowing for each application to choose the features that they need. This approach allows for better performance and per-application customization not possible with other approaches, such as a DOS TSR.

The TCP/IP code takes a "framework" approach to applications, allowing you to quickly get to the important parts of your application without getting bogged down in the details of initialization, error handling, etc.

Features

Some of the design goals of mTCP are:

- Configuration flexibility: A lot of features are enabled or disabled at compile time using #defines so that the TCP/IP code can be tailored to each application.
- High performance: Even on the slowest machine dating back to 1981 one can get raw TCP socket performance well in excess of 70KB/sec. (This is dependent upon the Ethernet card being used.)
- Small space requirements: The library is compact without cutting key features. Buffer sizes are configurable. Most of the applications run comfortably on a 256KB machine and many will run with less memory.
- Extensive tracing: Nobody likes a bug and a comprehensive tracing mechanism helps find and squash bugs after the code leaves my hands.
- Robustness: The library and applications are pretty rigorously tested. Some of the applications have been left running for weeks at a time with no unexplained memory leaks or crashes. The correctness of the TCP/IP and other protocols are checked against a variety of target machines.
- Usability: DHCP configuration makes it easy to use on a modern network. Command line options and the configuration file are not made needlessly complicated.

Some of the more advanced features of the mTCP TCP/IP library are:

- Automatic detection and retransmit of lost packets using measured round trip times.
- Support for multiple open sockets.
- Support for listening sockets to write server applications.
- DNS resolving (using UDP packets only at the moment).
- TCP zero window support.
- IP fragment reassembly and automatic sending of UDP fragments.

And some of the current limitations:

- Minimal support for IP and TCP header options
- Only one gateway may be configured
- Only one nameserver may be configured

In general, I implemented a set of features that makes sense for a small machine. There is enough implemented to allow the machine to interoperate well with a variety of other machines and not violate specifications.

Tested machines/environments

mTCP is not particulary fussy about what it runs on. As long as you have some variant of DOS and a machine (physical or emulated) to run it on, mTCP should work.

General requirements:

- An IBM PC compatible with an 8088 or better CPU. Virtual machines and emulators are included.
- 128KB to 384KB of RAM available depending on the program.
- DOS 2.1 or newer. (DOS 3.3 or newer is recommended.)
- A network card or device that has a “class 1” (Ethernet) packet driver.

Here are some of the machines, network cards, and environments that I have tested mTCP with:

Machines:

- IBM PCjr (4.77Mhz 8088 class machine)
- IBM PC and IBM PC XT (4.77Mhz 8088 class machine)
- Compaq Portable (4.77Mhz 8088 class machine)
- IBM PS/2 Model 25 (10Mhz 8086 class machine)
- IBM AT (8Mhz 80286 class machine)
- IBM PS/2 L40SX (25Mhz 80386 class machine)
- Generic 80386-40 clone
- Compaq Deskpro 433i (66Mhz 80486DX-2 class machine)
- Generic Pentium 133 clone
- Aptiva E3N (300Mhz AMD K6-2 class machine)

CPUs: 8088, 8086, V20, 80286, 80386, 80486, Pentium, etc ...

Video cards: Monochrome, CGA, EGA and VGA

Ethernet cards: The only requirement is a good packet driver. Cards that I have personally tested are:

- Xircom PE3-10BT (parallel port Ethernet adapter)
- 3Com 3C503 (ISA)
- Novell NE1000 (ISA)
- Novell NE2000 (ISA)
- Western Digital/SMC 80x3 series (ISA)
- LinkSys LNE100 (PCI)
- Intel EtherExpress 8/16

Operating Systems:

- IBM PC-DOS and MS-DOS version 2.1 should work well. Version 3.3 and above are recommended.
- FreeDOS 1.0 and 1.1

Virtual environments:

- DOSBox using the H-A-L 9000 "megabuild" (http://home.arcor.de/h-a-l-9000/)
- Windows NT4, 2000, and XP DOS command windows using SwsVpkt
- VMWare
- VirtualBox

Licensing

Early versions of mTCP were distributed in binary only form. This made sense as I was doing a lot of development and things were churning quickly.

On May 27th 2011 I open-sourced the code and made it available under the terms of the GNU General Public License version 3. All versions from 2011-05-27 through 2013-05-23 are open source with readily available source code. Please see the "copying.txt" file in those distributions for the full text of the license that covers those versions.

Starting with this release (2015-07-05) I am going back to distributing mTCP as binary files only. This is a personal decision based on my needs and my experiences. I may change my mind and release source code again in the future, but at the moment mTCP does not include source code.

(Source code for the 2013-05-23 version will remain available indefinitely.)

Packaging

mTCP is distributed as a set of binaries and this documentation. The binaries and documentation are in separate files because it is assumed that you will want to read the documention on a more modern machine.

Binaries

There are two flavors of the binaries available to choose from:

- Standard binaries: These are normal DOS EXE files that consume about 980KB on disk.
- UPX compressed binaries: These are the standard binaries compressed using “Ultimate Packer for eXecutables” (http://upx.sourceforge.net/). UPX compressed binaries take less space on disk (590KB) and take just a little longer to get running. These are ideal for systems where space is tight.

Once running both sets of binaries are equivalent; there are no performance or feature differences between them.

Documentation

This PDF file is the documentation for mTCP. There may be additional information available at the mTCP home page, but everything that you need to get started should be here.

The documentation used to be provided in TXT files, one for each program and several others for setup and miscellaneous topics. At the time I decided to consolidate the documentation in this PDF file there were 20 TXT files consuming 328KB of space. That became unwieldy. Moving to PDF has enabled me to improve the formatting and usability of the documentation.

I consider the documentation as important as the code. If you find a problem with the documentation or have a suggestion to make it better please let me know.

Support and contact information

Links to the latest mTCP code and documentation can be found at the home page at http://www.brutman.com/mTCP/. If you have questions, comments, feature requests or bug reports I can be reached at mbbrutman@gmail.com.

If you are having a problem getting the mTCP applications running please try to give me a good idea of your machine setup, your network configuration, and the symptoms you are experiencing. If I don't know the answer off the top of my head we can use the trace facility and try to figure it out. (See the chapter on Debugging for how to do some basic debugging and generating traces.)

If you are looking for enhancements to mTCP to support a commercial use please send me an email with what you are looking for. I have never accepted any money, beer or cookies for my work, however I am proud to have added new features in exchange for a donation to an animal shelter. (You get to choose the shelter!).

Setup

Introduction

mTCP is both a TCP/IP stack and a set of applications that use that TCP/IP stack. Before you can use the mTCP applications some setup is needed. You need to:

- Make sure your Ethernet card is ready to use.
- Tell mTCP where to find a setup file.
- Place some networking values in the setup file.
- Add some more configuration values to the setup file depending on what programs you want to use.

Most of this work is required to be done just once when you first set mTCP up. After that, using mTCP is quick and easy.

Hardware setup

All networking depends on having a connection from one computer to another. There are many ways to make that connection – serial ports, parallel ports, Ethernet, Token Ring, etc. It would be too complex for any program to try to know how to use every possible connection type so some simplifying assumptions are usually made.

Packet drivers

mTCP is designed to work with Ethernet cards. But instead of having code to talk directly to every possible Ethernet card mTCP has code to work with “packet drivers.”

A packet driver is a device driver that provides a simplified programming interface for Ethernet cards. Ethernet cards are often very different from each other; a packet driver is used to hide those differences. With packet drivers networking programs become simpler to write because instead of trying to know the details of every card, programs just need to know how to talk to packet drivers. The full packet driver specification can be found at http://www.crynwr.com/packet_driver.html.

In particular, mTCP is designed to work with “class 1” packet drivers. This set of packet drivers includes most Ethernet cards. Any packet driver that looks like a class 1 packet driver will work with mTCP. This includes “shims” that convert other types of Ethernet device drivers (ODI or NDIS) to class 1 packet drivers and packet drivers that make other hardware (serial ports) look like Ethernet cards. *(Even the definition of “card” is loosely used; there are Ethernet devices that connect using the printer parallel port that can be made to look like Ethernet cards through the use of a packet driver.)*

Packet drivers that do not provide a class 1 interface such as such as class 3 (Token Ring) or class 8 (ArcNet) are not compatible with mTCP.

For a general discussion on packet drivers see http://www.brutman.com/Dos_Networking/ .

Loading the packet driver

Packet drivers are usually loaded from the DOS command line as terminate and stay resident utilities. Each packet driver is specific to a model of Ethernet card so every one will be slightly different. However most have some common options that are required.

Software interrupt

Since a packet driver defines a high level programming interface for working with an Ethernet card, each packet driver can be told where to put the entry point to that interface so that higher level programs like mTCP applications can find it. The technical term for that entry point is “software interrupt”. When you load a packet driver you tell it which software interrupt to install its services at.

Software interrupts are generally written as hexadecimal numbers and are limited in range between 0x60 and 0x7F. You need to pick one that is not in use already but since most DOS software does not these software interrupts 0x60 is usually available.

The next set of parameters is often required to tell the packet driver where the Ethernet card is located in your system

I/O port

Each Ethernet card needs an I/O port address that can be used to send the card data and commands. The I/O port is usually set by a jumper on the card or a software configuration utility. It is extremely important to specify the correct I/O port address; if the address is incorrect the packet driver will not be able to control the Ethernet card.

Hardware interrupt

A hardware interrupt is a signal from your Ethernet card to the computer that lets the computer know when the Ethernet card has new data that needs to be received. When a new packet arrives the Ethernet card will send the signal using the hardware interrupt, waking up the loaded packet driver. The packet driver will then do the work to move the data from the Ethernet card to the main memory of the computer, and possibly signal any higher level software (mTCP) that new data is available.

The hardware interrupt number that the Ethernet card uses is usually set by a jumper on the card or a software configuration utility. The hardware interrupt number is often required on the packet driver command line so that the packet driver knows which hardware interrupt it should be listening to. If you do not use the hardware interrupt number that the card is set to use it will look like you can not receive any data, as the Ethernet card will be signaling that new work is available on one hardware interrupt while the packet driver will be listening on a different interrupt number.

Memory window address

Some Ethernet cards use shared memory to make the transfer of data between the Ethernet card and the computer faster. This shared memory is usually 8KB to 16KB in size and cards that use it are usually much faster that cards that use only an I/O port address.

If your Ethernet card uses shared memory there should be hardware jumpers or a software configuration utility to set the shared memory address. It is also possible that the shared memory address used by the hardware is set by the packet driver when the packet driver loads. Please check the packet driver for your card to see which system it uses for shared memory, if applicable.

Note for ISA bus systems: It is extremely important that any hardware resources that you use are not shared with another piece of hardware and that they are correct. This is a general ISA bus rule. Using the wrong hardware interrupt number, I/O port address or shared memory region will result in flaky system behavior or crashes.

Sample packet driver command lines

Here are some sample packet driver command lines with an explanation of their parameters.

```
ne2000 0x60 3 0x300
```

ne2000 is the packet driver for “NE2000” and compatible network cards. (NE2000 is a 16 bit ISA card that is the successor to the NE1000 card, which was a popular 8 bit ISA card.) On this command line we are telling the ne2000 packet driver to install its services at software interrupt 0x60, to listen to hardware interrupt 3, and to communicate with the NE2000 card at I/O port 0x300.

```
smc_wd 0x61 0x03 0x280 0xD000
```

smc_wd is a packet driver for the SMC/Western Digital 8003 series Ethernet cards. The smc_wd packet driver is being told to install its services at software interrupt 0x61, to listen to hardware interrupt 3, and to communicate with the Western Digital card at I/O port 0x280. The last parameter tells the packet driver to set the card up to use the memory at D000:0000 as a shared memory area for data buffers between the card and the computer.

```
pe3pd SINT=60 LPT=2 INT=5 NON
```

pe3pd is the packet driver for the Xircom PE3 series of Ethernet adapters. The Xircom PE3 is interesting in that it is an Ethernet device but it attaches to your computer using the standard printer parallel port.

On this command line we are telling the pe3pd packet driver to install its services at software interrupt 0x62, to find the Xircom device on the second parallel port, to listen to hardware interrupt 5 (which should be associated with that parallel port), and to not attempt to use the parallel port as a bi-directional parallel port. Note the different way in which the software interrupt is specified and the extra parameter that is specific to the parallel port.

More information on the Xircom PE3 Ethernet device can be found online at http://www.brutman.com/Dos_Networking/xircom_pe3.html.

Finding packet drivers

Packet drivers are specific to models of Ethernet cards. Older ISA bus cards almost always had packet drivers available for them. Many older PCI cards also had packet drivers available too.

If you have the original driver diskette or CD-ROM that came with your Ethernet card that might have the packet driver included with it. Here are some other places to look:

- http://www.crynwr.com/drivers/ has a collection of open source packet drivers for older cards
- http://www.georgpotthast.de/sioux/packet.htm has some packet drivers for newer Ethernet cards

If your card does not have a packet driver available it might have an ODI driver instead. ODI drivers can be made to look like packet drivers using a program called ODIPKT.COM. See ftp://ftp.hs-koblenz.de/pub/pc/msdos/networking/odi/odipkt/ to find a copy of ODIPKT.COM.

If your card only has an NDIS style driver look for DIS_PKT.DOS. A discussion of how to use it can be found at http://www.freedos.org/wiki/index.php/Networking_FreeDOS_-_NDIS_driver_installation.

Creating the mTCP configuration file

The mTCP configuration file is a simple text file that tells mTCP applications the networking parameters and other configuration values that they need to run. You only have to spend time once to create the file; after that you may change it or add to it but the basics will remain the same.

Creating the minimal configuration file for use with DHCP

The smallest mTCP configuration file consists of line line. It tells the mTCP programs where to find the software services installed by the packet driver for the Ethernet card.

For example, this would be a valid mTCP file for a machine that has a working Ethernet card and a packet driver for it loaded using software interrupt 0x62:

```
PACKETINT 0x62
```

Notice that there are no networking parameters in the file. The only thing that this file does is tell mTCP to use software interrupt 0x62 to find a packet driver.

This is where the mTCP DHCP client comes in. Instead of requiring you to add the networking parameters to the configuration file yourself, the DHCP program can be run and it will add the networking parameters automatically.

Before we can run the DHCP program we need to set a DOS environment variable that tells mTCP programs where the mTCP configuration file can be found:

(Assuming the configuration file is located in C:\MTCP\MYCONFIG.TXT)

```
set MTCPCFG=C:\MTCP\MYCONFIG.TXT
```

Now when you run the DHCP program the following will happen:

- The DHCP program will load and look for an environment variable called MTCPCFG.
- The filename pointed at by the MTCPCFG environment variable will be read.
- DHCP will look for the “PACKETINT” line in the file and see that it is supposed to use software interrupt 0x62 to communicate with a loaded packet driver.

- DHCP will make a connection to the packet driver so that it can send and receive packets.
- DHCP will use the packet driver to send and receive packets with your DHCP server. This is part of the process of negotiating a DHCP lease for your machine. (The lease is the right to use an IP address for a limited period of time.)
- DHCP will update the configuration file so that it looks like this when it is done:

```
DHCPVER DHCP Client version May  2 2015
TIMESTAMP ( 1432571386 ) Mon May 25 12:29:46 2015
PACKETINT 0x60
IPADDR 192.168.2.146
NETMASK 255.255.255.0
GATEWAY 192.168.2.1
NAMESERVER 192.168.2.1
LEASE_TIME 86400
```

Notice that two lines were added at the top and five more lines were added at the bottom. The five lines at the bottom are the networking parameters given to your machine by the DHCP server. The two lines at the top are added so that it is easy to determine that DHCP set the networking parameters.

Setting your own networking parameters using static configuration

If you are in an environment without a DHCP server or if you do not want to use DHCP then you can set the networking parameters in the mTCP configuration file by hand. The basic steps are the same except instead of using DHCP to add the networking parameters you will put them in the file when you create it.

Here is a sample file:

```
PACKETINT 0x60
IPADDR 192.168.1.10
NETMASK 255.255.255.0
GATEWAY 192.168.1.1
NAMESERVER 192.168.1.1
```

This sample is similar to the DHCP created version except that it is missing the DHCP specifics.

Reminder: To use mTCP programs you still need to set the MTCPCFG environment variable to the location of this file.

Networking related configuration parameters

What follows is a detailed description of the networking related parameters that are valid to use in the mTCP configuration file.

`PACKETINT <x>`

Required: Yes

This is the software interrupt number that the packet driver is listening on. This has to match the software interrupt number that you told the packet driver to use. It is specified in hexadecimal notation. (eg: 0x60)

`IPADDR <x.x.x.x>`

Required: Yes (Automatic when using DHCP)

This is the assigned IP address of the machine. IP addresses need to be unique on your network. If two machines try to share the same IP address neither machine will be able to communicate reliably.

The value can be provided explicitly or it can be provided by the DHCP program. If you are using DHCP then IPADDR will be added if it is not found or overwritten with a new value when a new lease is assigned.

`NETMASK <x.x.x.x>`

Required: Yes (Automatic when using DHCP)

This is the assigned network mask for your local area network. The network mask is used to determine if a target machine for a packet is on the same local area network as your machine or on a remote network. Packets that are destined for a remote network will be sent to the machine specified by the GATEWAY configuration parameter.

The value can be provided explicitly or it can be provided by the DHCP program. If you are using DHCP then NETMASK will be added if it is not found or overwritten with a new value when a new lease is assigned.

`GATEWAY <x.x.x.x>`

Required: No, but recommended (Automatic when using DHCP)

This is the IP address of the machine that will be used to forward packets destined for remote networks. This is usually the IP address of a router. If this is not provided or it is set incorrectly you will only be able to send and receive packets that are on your same local network.

The value can be provided explicitly or it can be provided by the DHCP program. If you are using DHCP then GATEWAY will be added if it is not found or overwritten with a new value when a new lease is assigned.

`MTU <x>`

Required: No

MTU stands for “Maximum Transmission Unit”, which is the largest number of bytes that can be sent by your networking hardware in a single packet.

For Ethernet this value is typically 1500 bytes, but that value might not work well on mixed networks where other networks have smaller MTU sizes. A setting of 576 should be safe for almost all environments. SLIP and PPP users will have a smaller value depending on their SLIP or PPP connection.

Please see the separate section on MTU for a more detailed discussion.

`NAMESERVER <x.x.x.x>`

Required: No, but recommended (Automatic when using DHCP)

This is the IP address of the name server on your network, which is responsible for translating human readable machine names (www.brutman.com) to IP addresses. If this value is incorrect then converting machine names to IP addresses will not work.

The value can be provided explicitly or it can be provided by the DHCP program. If you are using DHCP then NAMESERVER will be added if it is not found or overwritten with a new value when a new lease is assigned.

`NAMESERVER_PREFERRED`

Required: No

If you are using DHCP and it is assigning you a nameserver but you want to use a different nameserver, use this option. For example, if DHCP is assigning you a nameserver run by your ISP and you prefer to use Google DNS instead then this setting will tell mTCP to use Google DNS instead of the default nameserver:

```
NAMESERVER_PREFERRED 8.8.8.8
```

DHCP will not overwrite this preference.

Side note: Google DNS is fast, reliable, and does not return bogus results when a name does not resolve, unlike many ISP nameservers.

`HOSTNAME`

Required: No

The host name of your machine. The host name can be up to twenty characters, and should conform to internet standards. (Avoid punctuation and you should be fine.)

Other configuration parameters

The mTCP configuration file is used for more than network configuration. Other mTCP applications may read it to look for their configuration settings.

In general, you can put anything you want in the configuration file. Lines that the DHCP client don't understand will be left unaltered. Here are the rules:

- Blank lines are OK.
- Comments start with a # character in the first position of the line.
- Lines should never exceed 140 characters.
- If you use the DHCP client it will rewrite the configuration file and write new values for IPADDR, NETMASK, GATEWAY, and NAMESERVER. The previous values will be lost, but no other lines in the file will be altered.
- Other mTCP programs generally read but do not alter the file. They will skip lines that they do not understand. (And there is no reason why they can't write the file; none of the do that at the moment.)

Here is a sample of a more complex mTCP configuration file:

```
DHCPVER DHCP Client version May  2 2015
TIMESTAMP ( 1432571386 ) Mon May 25 12:29:46 2015

# Parms for my machine
#
packetint 0x60
mtu 1500
hostname DOSBox

# IRCjr parms
#
ircjr_nick Zoidberg
ircjr_user ZoidBerg
ircjr_name John F Zoidberg
ircjr_connect_timeout 30
ircjr_register_timeout 60

# FTP parms
#
ftp_connect_timeout 15
ftp_tcp_buffer 8192
ftp_file_buffer 16384

# DHCP generated settings will appear here
#
IPADDR 192.168.2.102
NETMASK 255.255.255.0
GATEWAY 192.168.2.1
NAMESERVER 24.159.193.40
LEASE_TIME 86400
```

Note that the DHCP client adds/replaces the first two lines of the file and the last five lines of the file. The rest of the lines in the file are kept as is for the other applications.

Configuration parameters for each application are documented by that application. The sample file above is just a sample; see the individual programs to see which configuration parameters they support.

Advanced topics

The MTU setting

MTU stands for 'Maximum Transmission Unit' and it tells the TCP/IP stack how big of a packet it can send on the local network. Bigger packets generally mean less overhead.

In a world where all networks are built using the same technology the MTU setting would not be needed. But that is not our world. Ethernet predominates, but there are other physical transports out there that you might not be aware of that you are using. Your Ethernet traffic can be transmitted over many different things that don't look like Ethernet before it reaches its final destination.

TCP/IP uses 'fragments' to deal with this problem. If a packet is too big to pass from one network to another the gateway machine will break the packet into fragments suitable for the next network. It is possible that a single packet will be broken into fragments multiple times over the course of its journey. The receiving side is responsible for gathering all of the fragments for an IP packet together and reassembling them.

While this process works it is not ideal. It causes a performance loss as the packets are fragmented by the gateways and a bigger performance loss when the fragments reach their final destination and have to be reassembled. In addition to the performance loss, it requires quite a bit of memory too. And losing a fragment hurts performance even more because all of the other fragments for that packet have to be thrown away.

In the real world fragments are rare. Most modern servers using TCP sockets do some form of path MTU discovery to determine the largest packet that can be sent without causing fragments to be created. For a TCP socket it works very well and you will have to work hard to try to get a server to send you a packet that needs fragmenting.

On the other hand, that mechanism is not applicable to UDP packets. A server sending a UDP packet has no choice about how to 'chunk' the data to avoid fragmentation; it has to send what the application says to send. Most UDP based programs avoid sending large packets that will be fragmented, but there are exceptions. (NFS and programs sending streaming video data are two notable exceptions.)

mTCP supports the reassembly of fragmented packets, with some restrictions. There is a limit on the total size of the fragmented packet that is set at compile time, and there is a limit to the number of fragments that can be reassembled at the same time. Currently those limits are set at 1500 bytes per packet, and 4 packets at a time. While low, those limits will protect you against a server on a standard Ethernet that goes through a misconfigured gateway that causes fragments to be created. (For UDP applications that require larger packets that can be set at compile time when such an application is created. There are no such applications available at the moment.)

Even though mTCP can reassemble fragments there are still all sorts of fragment related dangers floating around out there:

- Some TCP/IP servers set the 'Don’t Fragment' bit on their packets. If the packet needs fragmentation at a gateway and this bit is set, the packet effectively gets dropped at the gateway. mTCP will have no knowledge and no way to get around the dropped packets, other than mysterious timeouts from the application. While the server might be trying to avoid additional overhead caused by fragmentation, it is really doing a disservice by forcing these packets to be dropped. (I'd rather get the data slowly than not get it at all.)
- Some home router/firewalls throw fragments away rather than try to process them correctly. This is especially true when NAT (Network Address Translation) is used.

The good news is that you have some control over the problem.

Applications that use TCP sockets will tell the remote side what their MSS (Maximum Segment Size) is. (MSS is based on MTU.) This happens during the socket connect and once the remote side knows your MSS it will not knowingly send you a packet that needs to be fragmented. (Something in the middle might cause fragmentation, so it is still possible. MTU path discovery is supposed to avoid that danger by actively probing all gateways between the server and your machine.)

If you are having problem with dropped packets on a TCP socket, set your MTU to 576. That is guaranteed to work on all gateways and all routers. This is also the default for mTCP if you do not specify an MTU size.

UDP should not have this problem; if an application knowingly sends a large packet it will not turn on the Don't Fragment bit in the IP header. Doing so would effectively break the application, as the data would never be able to get out. On a large UDP packet fragmentation is the only solution; there is no concept of MSS. Most UDP packets used by mTCP never exceed 512 bytes of data, so fragmentation is not an issue.

For best performance set MTU has large as you can. For Ethernet that value is usually 1500 bytes. This minimizes the overhead of the protocol headers and minimizes interrupt handling.

If you are having strange connection issues you might be bumping into a fragmentation problem. Many times there is no warning message because the packets just simply don't arrive, and the ICMP messages get sent directly to the server, not the mTCP code. If this happens set your MTU to 576 and try again.

SLIP and PPP connections generally have smaller MTU settings which may cause fragmentation. This is unavoidable; consider it part of the joy of using low speed point-to-point links. Your local gateway that is providing the SLIP or PPP connectivity will generate fragments that mTCP will have to reassemble.

SLIP and PPP notes

(Please see the chapters on PPP and SLIP for detailed instructions on how to setup PPP and SLIP connections, including how to use a Linux machine as the gateway.)

I designed and implemented everything all of this code assuming Ethernet. Quite a bit down the road somebody pointed out that my code was inaccessible to an entire class of people using SLIP (Serial Line IP) and PPP, which is often used on dial-up.

I've tested the code using SLIP and PPP. Here are the key things you need to be aware of.

- You need to use a packet driver that simulates Ethernet. SLIPPR and CSLIPPER can do this for SLIP connections, and DOSPPP can do this for PPP connections. For SLIPPR use the "ether" option to do this. For DOSPPP use the EPPPD program.

- mTCP assumes that ARP is necessary, which is completely wrong on a point to point connection. It will not get a response to an ARP query for the gateway, and it will time out. To avoid this set the following environment variable:

  ```
  set MTCPSLIP=true
  ```

  Do this for both SLIP and PPP.

- This environment variable stuffs a bogus ARP entry into the mTCP ARP table for your gateway. That allows it to immediately send packets to the gateway without sending an ARP request. The Ethernet address in the sent packets will be wrong, but nothing is looking at it anyway.

- I've tested configurations using static addressing. You will have to find a way to pass the IP address, netmask, etc. to mTCP using it's configuration file.

- It is beyond the scope of this document to explain addressing in a point-to-point configuration. Here is something to keep in mind though.

  If you use the smallest possible netmask (255.255.255.252) you can easily create a configuration that works. With that netmask only the last two bits are significant. Considering the last two bits only:

```
<30 bits of netmask>.0 - reserved address meaning 'this host'
<30 bits of netmask>.1 - usable, good for the gateway address
<30 bits of netmask>.2 - usable, good for your address
<30 bits of netmask>.3 - IP broadcast address for this network
```

Setting up your netmask and addresses like this is simple and should avoid routing problems. Basically you force mTCP to always send things to the gateway because nothing else is on your subnet.

SLIP and PPP generally use smaller MTU settings than other networks because their transport layer is slower. Be sure that the mTCP MTU setting reflects what your SLIP or PPP connection is using. Depending on the application, servers and gateways you may experience some fragmentation, especially on UDP packets. (TCP generally avoids this by using the MTU setting to compute MSS.)

How the configuration file works

At the beginning of time, mTCP used environment variables for all of the configuration. This worked well until the IRC client came along and I had to start increasing the size of the DOS environment space. Then came the DHCP client - I couldn't find a way to set environment variables from within the DHCP client and make them stick for the next program without writing out a file. At that point the configuration file was born.

When the DHCP client starts up it reads the PACKETINT and HOSTNAME parameters from the configuration file and ignores the rest. (HOSTNAME is optional.) It then tries to communicate with the DHCP server on your network. If it gets an address from the DHCP server it will write a new configuration file. All of the contents of the old configuration file are copied, except for the IPADDR, NETMASK, GATEWAY, NAMESERVER, and LEASE_TIME parameters which get updated with new values at the end of the file.

Having DHCP write a configuration file has an interesting side effect. It means that at its core, every mTCP application assumes static network addressing. They all read the configuration file to get the network parameters, and they can't tell if the network parameters were set by a human or a machine.

At start-up each application looks for the MTCPCFG environment variable and tries to read the configuration file. It makes one pass to get the networking parameters that it needs to initialize the TCP/IP stack.

If an application has other configuration parameters that it needs to read it will reopen the configuration file and scan for them. The format for a parameter is a single token that represents the parameter name, a space for a delimiter, and then the value of the parameter. The entire rest of the line is assumed to be the parameter. Lines must be under 75 characters in total length.

mTCP and Virtual Environments

If you want to try DOS and mTCP but don't have an old computer to dedicate to it, don't worry! All of this runs fine in virtual and emulated environments. Here are some notes to get you started.

If you find that your virtual machine is chewing up a lot of CPU please read the section on DOS Power control. Using the wrong power/idle management program will reduce your CPU usage but also hurt the usability of mTCP.

DOSBox

DOSBox is an x86 PC emulator designed to run old DOS games on modern hardware running modern operating systems. While Windows XP has a DOS-like command line it is not sufficient to run many old games. This is where DOSBox takes over and shines.

While relatively full featured, DOSBox is not a full machine emulator. DOS is emulated just enough to make the old games run. The command prompt recognizes a few major commands; you may need to add more from your favorite DOS version. DOSBox can be configured to share part of the host operating system filesystem so that moving files in and out is easy.

By default there is no networking support in DOSBox. To get an emulated network device you need to use a special build of DOSBox created by "HAL9000". The HAL9000 "megabuilds" add NE2000 emulation allowing networking applications to run, plus a few other goodies. When using the NE2000 emulation DOSBox will appear to be physically connected to the network. (This is commonly known as "bridged" mode.) I have been using the megabuilds for over four years now and I am very pleased with their quality.

I do most of my functional testing with DOSBox so I know that it works well with mTCP. The biggest function missing from DOSBox is proper Ctrl-Break and Ctrl-C handling. mTCP lets you break out of some long running applications using Ctrl-Break and Ctrl-C so under DOSBox that feature might not work. But for the most part DOSBox is a great way to run mTCP.

DOSBox main site	http://www.dosbox.com/
HAL9000 DOSBox megabuilds	http://home.arcor.de/h-a-l-9000/
NE2000 packet driver for use with DOSBox	http://www.brutman.com/mTCP/NE2000.COM

VirtualBox

VirtualBox is an open source program that creates a virtual x86 machine within your host operating system. You define the amount of memory, the size of the virtual hard drive, the amount of video memory, networking support, and a few other attributes to create a virtual machine. Then you boot the virtual machine and install DOS on it, just like you would do on a bare machine that you are installing for the first time. The virtual machine emulation takes advantage of modern CPU features making it extremely fast - almost as fast as running directly on the hardware without the host operating system in the way. Virtual machines slow down when doing I/O operations but it is not enough to worry about.

VirtualBox has a variety of network cards that it can emulate and it can make your virtual machine look like it is directly on the network ("bridged"), behind a firewall ("NAT", or Network Address Translation), and a few other options. I use "bridged" mode and the AMD PCNet device, which is the same device used by VMWare Player. If you use a different emulated device be sure to find the correct packet driver for it.

All of the mTCP applications run well except for PING, which runs but seems to suffer from a timing problem. I reprogram the interval timer to get higher resolution timings than are normally possible on DOS and it looks as though PING is tripping on this. It will eventually complete but it takes much longer between ping packets than it should. (The packet timings are accurate, but the wait time between packets is too long.)

VirtualBox main site	http://virtualbox.org/
AMD PCNet packet driver	http://www.brutman.com/mTCP/PCNTPK.COM

VMWare Player

VMWare Player is a closed source (but free) virtual x86 machine that is very similar in operation to VirtualBox. (VMWare predates VirtualBox by quite a bit.) You define your virtual machine, install DOS, and then run it side by side with your host operating system.

The networking works basically the same way as it does in VirtualBox - "bridged" makes your virtual machine look like it is directly connected to the LAN. "NAT" (Network Address Translation) makes it look like your virtual machine is hiding behind a firewall on your host operating system. "Host-only" is the most limited, and it allows you to communicate only with the host operating system. VMWare emulates some form of AMD PCNet Ethernet device.

I am using version 3.1.3 on Windows XP. Everything works perfectly, including PING, so VMWare Player probably does a better job of emulating the PC timer hardware that I am reprogramming.

VMWare Player main site	http://www.vmware.com/products/player/
AMD PCNet packet driver	http://www.brutman.com/mTCP/PCNTPK.COM

SwsVpkt

SwsVpkt is a virtual packet driver that allows DOS networking applications to run directly under Windows NT4, 2000 and XP in a command window.

SwsVpkt works by letting those applications send RAW Ethernet packets to and from the network using a network card on the machine. Unlike the other solutions on this page SwsVpkt does not give your code the illusion of having a separate network card with a unique MAC address. Your DOS application will be sending and receiving packets using the actual MAC address of the card you select to use. If you use the same card that your host operating system is using you need to configure mTCP to use a different IP address than what your host is using. This makes DHCP unusable as your DHCP server will see the MAC address and give mTCP the same address that it gave your host operating system. As a result, static configuration of the IP address in the mTCP configuration file is required.

Here are some usage notes:

- Use "`swsvpkt -l`" to list available adapters.
- Use "`swsvpkt -a<n> <packet_int>`" to load swsvpkt and use a specific adapter. <n> is the adapter to use and you need to choose one that is connected to your network. <packet_int> is the

software interrupt that mTCP will use to talk to swsvpkt.

- Be sure that you setup your mTCP configuration file by hand and that the address you choose is different than the address being used by the host machine!

On my machine "`swsvpkt -l`" finds four network adapters it can bind to:

```
E:\DOSPROGS\MTCP>swsvpkt -l

Packet driver for Windows VDM version 1.05, updated Fri May 19 12:59:33 2006
Copyright (C)2005,2006 Lawrence Rust. All rights reserved.

Found interface:
0: VMware Virtual Ethernet Adapter
1: VMware Virtual Ethernet Adapter
2: Intel(R) 82562V-2 10/100 Network Connection (Microsoft's Packet Scheduler)
3: NOC Extranet Access Adapter (Microsoft's Packet Scheduler)
```

To run I use `swsvpkt -a2 0x60`:

```
E:\DOSPROGS\MTCP>swsvpkt -a2 0x60

Packet driver for Windows VDM version 1.05, updated Fri May 19 12:59:33 2006
Copyright (C)2005,2006 Lawrence Rust. All rights reserved.

Found interface:
0: VMware Virtual Ethernet Adapter
1: VMware Virtual Ethernet Adapter
2: Intel(R) 82562V-2 10/100 Network Connection (Microsoft's Packet Scheduler)
3: NOC Extranet Access Adapter (Microsoft's Packet Scheduler)

Installed on SWI 0x60, using IRQ 7.
```

The mTCP applications seem to work normally when run using swsvpkt. I have noticed that swsvpkt leaves the command window in a strange state - command line editing no longer works after I have run it. I'm not quite sure what is going on, so if you know what it is please drop me a line. (Other command line windows are not affected.)

SwsVpkt main site	http://www.softsystem.co.uk/products/swsvpkt.htm

mTCP Applications

DHCP client

Introduction

A DHCP client is a small utility that helps you automatically configure the networking parameters for a machine, allowing it to use your network. This is an alternative to "static" addressing where you set the network parameters for your machine manually.

Most networks have a router that has a DHCP server built in, making DHCP the preferred way to configure a machine for networking. This DHCP client can be used to automatically configure the networking parameters for all of the mTCP applications and with some cleverness it can be used for other networking stacks as well.

Special hardware or software requirements:

None

Setup instructions

DHCP is built using the mTCP library. The detailed setup instructions for mTCP can be found in SETUP.TXT.

Here is the quick story:

1. Load your packet driver. If your packet driver loads correctly it will probably report the MAC address of your Ethernet hardware.

2. Create a configuration file for mTCP. The configuration file is a text file that all mTCP apps will read when they start. The smallest possible configuration file has one line in it to tell mTCP where the packet driver is loaded. Here is an example:

   ```
   packetint 0x60
   ```

   Just replace the 0x60 with whatever software interrupt your packet driver is set to use. There are more things you can add to the configuration file but this will get you started.

3. Set the MTCPCFG environment variable to the full path and name of the configuration file. For example:

   ```
   set MTCPCFG=c:\mtcp\tcp.cfg
   ```

And now you should be ready to run DHCP.

Using DHCP

The DHCP command line format is:

```
dhcp [options]
```

where options are:

`-help`	Show a help message
`-retries <n>`	Retry n times before giving up (default=3)
`-timeout <n>`	Set timeout for each attempt to n seconds (default=10)

When DHCP runs it will send a series of UDP packets and try to negotiate a DHCP lease with a DHCP server on your network. If DHCP is successful it will set a return code of 0, show the network settings it received on the screen, and write the new network settings to the mTCP configuration file.

If DHCP fails it will set a return code of 1.

You can check the return code in a batch file using ERRORLEVEL. Here is a simple example:

```
ne2000 0x60 3
set mtcpcfg=e:\mtcp\tcp.cfg
dhcp -retries 3 -timeout 20
echo off
if errorlevel 1 goto Errmsg
echo DHCP worked
goto End
:Errmsg
echo DHCP failed!
:End
```

You should run this program at least once a day, and before running any other mTCP programs. This program will write the settings it gets into a configuration file that the other mTCP programs will look for and use. (You do not need to run DHCP before each mTCP application.)

DHCP lease duration

Remember that a DHCP lease has an expiration time; your router will not let you use an address forever. The normal lease duration is a few hours but it can be much shorter than that.

DHCP stores the time the lease was granted and the expiration time in the mTCP configuration file. When an mTCP program starts it will check the current time and the lease expiration time. It will issue a warning and refuse to start if the lease is going to expire within an hour.

If your DHCP server issues short leases (less than an hour) this will be a problem. There is a configuration override that you can add to the mTCP configuration file to override that threshold. The name of the value is DHCP_LEASE_THRESHOLD and the value is the number of seconds to set the threshold to. For example:

```
DHCP_LEASE_THRESHOLD 1800
```

will change the threshold to 1800 seconds so that if your DHCP lease expiration is further than 30 minutes out into the future you will not get a warning.

DHCP failures

DHCP is often the first thing that runs after a user does the basic mTCP configuration. As a result it is often the first thing to fail. The most common form of failure is a timeout. Below are some factors that can cause timeouts:

- a misconfigured card
- using the wrong packet driver
- bad parameters to the packet driver
- cabling/network issues

A detailed discussion of these factors can be found in the Debugging section.

In the unlikely event that you can not use DHCP you can still use mTCP applications; you will have to do the network parameter setup manually though. See the Setup section for details. (I have not yet met a DHCP server that mTCP does not work with so being forced to do a manual setup is unlikely.)

Configuration parameters

Normally configuration parameters are read-only and are used to set the behavior of a program. DHCP is unique in that it writes some configuration parameters to the mTCP configuration file. These parameters are networking related parameters.

Parameters provided by the DHCP server:

These configuration parameters will be overwritten by the values that your DHCP server sends.

```
IPADDR <x.x.x.x>
```

This is the assigned IP address of your DOS machine.

```
NETMASK <x.x.x.x>
```

This is the assigned network mask for your DOS machine.

```
GATEWAY <x.x.x.x>
```

If provided, this is the IP address of the default gateway that your DOS machine will use to connect to machines that are not on the same network segment. (While optional, it is almost always provided.)

```
NAMESERVER <x.x.x.x>
```

If provided this is the IP address of the DNS server that will be used to resolve host names to IP addresses. If this is not provided host names will not be able to be converted to IP addresses.

```
LEASE_TIME <x>
```

This is the number of seconds that the DHCP lease is valid for. mTCP applications will check for lease expiration when they start, but they will not check again while they are running. So if your DHCP lease is due to expire soon there will be a warning at start-up.

Configuration parameters:

```
DHCP_LEASE_THRESHOLD <x>
```

This option sets when mTCP applications will warn you that your lease is about to expire. Please see DHCP lease duration for how it works.

```
NAMESERVER_PREFERRED <x.x.x.x>
```

This option can be used to tell mTCP applications to use a different nameserver than the one provided by your DHCP server. For example, if you want to use Google public DNS servers instead of your ISP's nameservers you could use the following:

```
NAMESERVER_PREFERRED 8.8.8.8
```

The NAMESERVER option set by your DHCP server will continue to be updated, but mTCP programs will use NAMESERVER_PREFERRED if it is set.

Recent changes

2015-04-10	Add preferred nameserver support
2014-06-01	Add DHCP_LEASE_THRESHOLD configuration parameter
2013-03-29	Add numerical timestamp to the DHCP timestamp message to make it easier to compute when the lease expires.

DNSTest

Introduction

This is a simple utility for resolving machine names to IP addresses. It uses UDP only, which is standard for DNS. If you are curious there is a lot of debug output available using a command line option.

Special hardware or software requirements:

None

Setup instructions

DNS name resolving depends on the NAMESERVER configuration setting in the mTCP configuration file being set correctly. If you are unsure of what to set NAMESERVER to, Google runs a public DNS system that is fast and reliable. The IP addresses for the Google DNS servers are 8.8.8.8 or 8.8.4.4. See https://developers.google.com/speed/public-dns/ for details.

Using DNSTest

Usage is like this:

```
dnstest -name <your.machinename.here> [options]
```

Options are:

-help	Show a help message
-norecurse	Do not request a recursive DNS lookup
-timeout <n>	Timeout after <n> seconds
-verbose	Turn on verbose debug messages

The -norecurse option requires some explanation. Normally mTCP will send DNS requests with the 'recursion requested' flag turned on in the request. This makes the nameservers do a lot of the work necessary to resolve a name. If you set this option that flag will not be set and you will see a lot more activity on your machine to do the same DNS lookup. It is interesting to play with but is inefficient for normal lookups. Use this option with the -verbose option to see the full effect.

Here is some sample output:

```
dnstest -name www.brutman.com
mTCP DNSTest by M Brutman (mbbrutman@gmail.com) (C)opyright 2009-2013
  Version: May  2 2015

Timeout set to 10 seconds, DNS Recursion = on
Press [ESC] or [Ctrl-C] to quit early

Machine name www.brutman.com resolved to 208.73.32.24
Elapsed time in seconds: 0.055
```

If you specify the -verbose options you will see a lot more output!

Recent changes

2010-06-21 Ported to Open Watcom
2009-01-18 Make resolving more robust when recursive mode is turned off.
2008-12-30 First public version

FTP client

Introduction

This is an FTP client like many other FTP clients you have probably used before. Features include fairly high performance, low memory requirements, and compatibility with a wide selection of FTP servers.

Special hardware or software requirements:

None

Using FTP

FTP uses the following syntax:

```
ftp [-port <n>] <ftp_server_addr>
```

Options are:

`-help`	Show a help message
`-port <n>`	Connect using port <n> instead of port 21

You have to specify the FTP server address on the command line. After the control socket is opened you will be prompted for a user ID and password. After that you get a command line just like on other FTP command line clients.

Here are the commands:

dir [<filespec>]	directory list (full details)
ls [<filespec>]	directory list (names only)
pager <n>	pause ls or dir output after approximately n lines
cd [<directory>]	change to a directory on the server
cdup	move up to the server parent directory
pwd	print the current server directory
lcd [<directory>]	change the local directory
lmd [<directory>]	make a local directory
mkdir [<directory>]	make a directory on the server
rmdir [<directory>]	remove a directory on the server

ascii	change the file transfer mode to ASCII
image, binary or bin	change the file transfer mode to binary
get <server_file> [<new_local_file>]	get a file from the server
put <local_file> [<new_server_file>]	put a file on the server
delete <server_file>	delete a file on the server
prompt	toggle prompting for mget, mput and mdelete on or off
mget <server_filespec>	get multiple files matching server_filespec
mput <local_filespec>	put multiple files matching local_filespec
mdelete <server_filespec>	delete multiple files matching server_filespec
rename <from_file> <to_file>	rename a file on the server
xfermode [<new_mode>]	set PORT or PASSIVE transfer mode (see section below)
quote <remote command>	send a raw command to the FTP server to execute
quit	quit this program
shell	shell out to DOS
interactive	switch input from a script file to the keyboard

The program captures Ctrl-Break and Ctrl-C. If you use Ctrl-Break during a file transfer or directory listing the current operation will stop (prematurely) and you will be returned to the command line after a brief delay. If you use Ctrl-Break or Ctrl-C at the command line the program will end. (This might take up to 20 seconds depending on the state of the sockets, so please be patient.)

The shell command should be used with care. With the shell command you can do local directory listings, rename local files, look inside of files, and do most normal things. Keep in mind that while you are at the DOS command prompt the mTCP code is not servicing the socket connection - if you stay in DOS too long you might lose your connection to the FTP server.

Transfer modes

The xfermode command requires a little explanation.

In the beginning of time FTP servers always sent data by doing a connect to the FTP client, which was listening for the incoming connection on a default port. The FTP server in NCSA telnet does this. I will call this 'Classic' mode and you can tell this FTP client to handle data transfers this way.

'Classic' mode is only useful for ancient FTP servers that can't do anything else. Most modern FTP servers will not connect to an FTP client on the default ports and most firewalls/routers wouldn't allow it anyway.

'Port' mode is the current default for many clients. Instead of having the FTP server connect to the client on default ports, the client will tell the FTP server exactly what port it should use for the connection. This is slightly better than Classic mode, and should be usable on more FTP servers, firewalls and routers.

'Passive' mode is probably the best and most compatible mode to use with modern FTP servers. It turns things around by making the client connect to the FTP server to make data connections instead of having the FTP server connect to the client. (From the standpoint of the FTP server it is 'passive' and it waits for a connection from the client.) This will work with almost any firewall and router. This is the mode I'd recommend using too. Passive is the default mode for this client.

DOS filename limitations

As this FTP client runs under DOS, it is subject to DOS path and filename limitations.

When specifying a filename on the server you can use whatever filenames are legal for the server. This may require the use of quoting filenames (see below).

When dealing with local files you must obey the rules for DOS. For example, DOS filenames must fit the 8.3 naming convention and some characters are forbidden.

The FTP client will not try to check and reject invalid DOS filenames; it will just let DOS deal with them. Sometimes this results in filenames being truncated to fit the 8.3 format. Other times it might result in an error and an uncompleted operation because DOS refused to work with the filename. Please be aware of your filenames and try not to break DOS too badly.

One important exception: the FTP client will allow you to use either a forward slash or a backward slash as a directory delimiter for local DOS filenames.

Filenames and Quoting

DOS does not allow a filename to have a space in it. However, you may find that FTP servers running real operating systems do support filenames and directory names with spaces in them.

FTP supports adding quotes around filenames and directory names so that you can work with these servers. (You still have to provide a legal DOS filename for receiving files.)

If you need to specify a filename that has the double quote character in it use two double quote characters in a row. This will result in one double quote character being sent. If you have to deal with embedded spaces at the same time, add an extra set of quotes to enclose the entire parameter.

Examples:

You enter:	The server sees:
"Spacey name"	**Spacey name** *(All one parm)*
""QuoteChar""	**"QuoteChar"** *(All one parm)*
""Quote Char""	Uh oh .. you get **"Quote** as one parm and **Char"** as the second parm. That is because the two sets of double quotes collapse into a single double quote, but do not do anything to mark it all as one parameter.
"""Quote Char"""	**"Quote Char"** *(All one parm)*. Note the usage of the triple quotes.

Command line editing

If you are using FTP interactively you can take advantage of the command line editor. It does all that you would expect it to do: backspace, delete, insert, home, end, etc. It also includes a ten line command buffer so that you can recall older commands easily without retyping them.

Scripting support

FTP will take input from a file redirected to stdin. This allows you to automate some simple tasks, like logging in to fetch a file.

Here are some notes for using this support:

- Because you have to start FTP with stdin redirected from a file, you will probably have to include your login information as the first two lines of the script file.
- FTP will interpret "end of file" as "quit". So when the script ends, so does FTP.
- If you don't want FTP to quit when it hits the end of the script file, add the "interactive" command to the end of the file. This will tell FTP to start reading from the keyboard again.

Advanced Configuration Parameters

The following are optional configuration settings that you can put in your mTCP configuration file. See SETUP.TXT for how to make changes to that file.

```
FTP_CONNECT_TIMEOUT <n>
```

Set the timeout for a socket connection to n seconds. The default timeout is 10 seconds.

```
FTP_TCP_BUFFER <n>
```

Set the size of buffer used by TCP for receiving data to n bytes. The default is 8192 bytes. For best performance this should be set to a multiple of 1024 (1KB). The default is reasonable compromise between size and performance. You can go lower to conserve memory or up to 16384 bytes to improve performance.

```
FTP_FILE_BUFFER <n>
```

This controls the size of the buffer used to store data when receiving a file. The default is 8192 bytes. For best performance this should be set to a multiple of 1024 bytes. The default is a reasonable compromise between size and performance. The maximum size is 32768 bytes.

```
FTP_SEND_FILE_BUFFER <n>
```

As of this release this configuration option is ignored. The value set by FTP_FILE_BUFFER is used for both sends and receives.

```
FTP_MLIST_BUFFER <n>
```

When you use MGET, MPUT or MDELETE FTP has to fetch a list of files from the server or from your local machine. This buffer holds that list of files. If the list of files is too long the MGET, MPUT or MDELETE command will fail.

One solution is to change your filespec so that the list of be transferred is smaller, and use multiple MGET or MPUTS to do what you need. The other solution is to increase this buffer size. The default size is 4096 bytes, which is plenty for most usage. The maximum size is 16384 bytes.

Recent changes

2015-05-29 Change parsing of PASV responses to work around poorly configured servers.

2015-02-27 Add mdelete command; make help text more helpful. Remove separate SEND_FILE_BUFFER option to simplify setup. Remove length restrictions on local filenames to allow for paths to be specified with filenames. And some code cleanup.

2013-03-28 Allow for larger user commands; add some ability to handle packets while waiting for user input; support screens larger than 80x25; Fix 125 message handling.

2011-09-23 Make passive the default xfermode. Fix a small display bug when editing the current command.

2011-01-02 Fix handling of multi-line error messages

2010-11-27 Miscellaneous TCP library changes to improve handling when buffers are full. Add sending and receiving rate at end of transfers. Watcom runtime workaround: improve compatibility with older machines.

2010-10-20 Add lmd command (local make directory). Add command line editing with a 10 command recall buffer

FTP server

Disclaimer

While I make nearly every attempt to ensure that my code is free from bugs and safe to run on your system, I am not perfect and things can happen. You are encouraged to test this code in isolation on a machine that you can afford to reinstall to get comfortable with it. While that's extreme, it's probably the only prudent thing to do with any new code.

Use this, and anything else that I write at your own risk. I take no liability for anything that might happen to you, your computer, your network, house, descendents, pets, etc.

Introduction

This is an FTP server for DOS. Some of the features include support for passive and active connections, support for multiple users at the same time, a sandbox mode which is useful for setting up anonymous FTP, and good compatibility with a wide variety of FTP clients.

Special hardware or software requirements:

The FTP server can run in as little as 190KB if you just have one FTP client connecting at a time. If you want to support multiple connected clients more memory will be needed.

Before you fire it up ...

Setting up and running an FTP server is a little more involved than running the other mTCP applications. You need to consider your security and performance requirements. Security is especially important; an FTP user with full privileges to your system can do serious damage, accidental or otherwise.

Quick Start Instructions

1. Configure mTCP and ensure that you have basic network connectivity. You should be able to run the other mTCP applications first.
2. Add the FTPSRV_PASSWORD_FILE and FTPSRV_LOG_FILE configuration parameters to the mTCP configuration file. (FTPSRV_LOG_FILE is optional but recommended.)
3. Create a minimal password file. (See the included sample file, ftppass.txt)
4. Run. You should be able to use the FTP server from machines on your home network with no further configuration.
5. If you are planning on offering anonymous FTP on the internet or using it with FTP clients on the other side of a firewall, read the rest of this file.

FTP server background information

Active connections vs. Passive connections

FTP clients connect to FTP servers on a socket connection known as the control connection. The client sends commands and gets return codes on the control connection. FTP servers usually listen for new control connections on port 21, although other non-standard ports are possible.

When an FTP client and server wish to exchange data they will open a new socket connection that will be used for that one data transfer. A data transfer is a directory listing, a file send or a file receive. (Yes, even a directory listing requires a new socket connection.) After the data transfer is done the new socket is closed, leaving only the original control connection. Data connections are not used for more than one data transfer.

There are two ways the new socket connection for data can be created. An "active" connection is when the client tells the server where to connect to, listens on a port, and waits for the server to open the new socket connection. A "passive" connection is when the client asks the server to listen on a port and the client opens the new socket connection. Note that "active" and "passive" are from the point of view of the server.

In the real world active connections are more difficult to support because most home users are behind firewalls. Firewalls cause problems because they usually are permissive about letting machines make outgoing connections to the rest of the world but they do not readily let incoming connections in. This is a problem because when in active mode, an FTP server will have to make a new incoming data connection back to the client. Firewalls have special code to allow this incoming connection but it often doesn't work correctly for a variety of reasons.

On the other hand, passive mode connections almost always work. If your client was able to create the control connection to the FTP server in the first place, then it probably can create other connections to the FTP server. Passive mode is generally preferred by FTP clients because it works with a wider variety of firewalls.

If you have a firewall it is probably optimized to allow FTP clients make connections to the outside world. Running an FTP server behind a firewall becomes a problem because you need to allow incoming connections from the outside world. (We'll talk about that more later.)

UserIDs, Anonymous FTP, and permissions

Anonymous FTP allows anybody to connect to and use your FTP server. There is no authentication required. While useful for distributing data, anonymous FTP can be dangerous when not configured correctly.

In addition to anonymous FTP the mTCP FTP server supports named FTP accounts that require a password be entered. Named accounts provide more security but they are not foolproof either - FTP transmits passwords in the clear and a named user can still do accidental damage to your system.

To give you some control and security the mTCP FTP server has the following features:

- The ability to lock a user into a specific part of the directory hierarchy. This ability can be used to create "sandboxes" that can limit what users can see and do.
- The ability to restrict uploads to a particular directory whether the user is in a sandbox or not.
- The ability to set permissions for individual operations that might be destructive in some way, such as

delete, make directory, remove directory, rename, and store (store, store unique, or append) files.
- mTCP FTP server does not let you overwrite files when uploading. You must explicitly delete a file before replacing it.

It isn't difficult to setup the server in a secure way. But it does require a little planning.

Configuration parameters

There are no command line options that are used when starting the FTP server; all configuration parameters are set in the mTCP configuration file. The following configuration parameters are recognized:

Required parameters:

```
FTPSRV_PASSWORD_FILE <path_and_filename>
```

This should be set to a full path and filename of a password file. The password file format is explained in "Setting up the password file."

Optional parameters:

```
FTPSRV_LOG_FILE <path_and_filename>
```

This should be set to a full path and filename of a file that will be used to log user connections and file transfers. The file will be appended to each time it is opened.

If you do not specify this option then no logging of user activity will be done.

```
FTPSRV_SESSION_TIMEOUT <seconds>
```

This configuration parameter allows you to set the inactivity timeout for FTP sessions. The default is 180 seconds (three minutes). The maximum is 7200 seconds (two hours). This configuration parameter is optional.

```
FTPSRV_CONTROL_PORT <port number>
```

The standard port for an FTP server to listen to for new control connections is port 21. You may specify an alternate port if 21 is not available or usable. A lot of ISPs will block incoming access to port 21, so to accept connections from the outside world you may need to use a high numbered port such as 2021. This is not standard so you will need to make it clear to any possible client what the new port number is.

```
FTPSRV_EXT_IPADDR <numerical ip address>
```

If you are behind a firewall and you are accepting passive connections from the outside world you will need to use this option to tell the FTP server what your external (public) IP address is. This allows the FTP server to put the correct connection information in the response to the PASV command sent from the client.

See the section on firewall considerations for more information.

`FTPSRV_PASV_BASE <port number>`

If you are accepting passive connections you can use this option to specify a starting port number for passive data connections to use. The default starting port number is 2048.

Being able to set this value might be helpful for configuring your firewall.

`FTPSRV_PASV_PORTS <number of ports to use>`

This option can be used to set the number of ports used for passive data connections. The default is to use 1024 different ports.

Being able to set this value might be helpful for configuring your firewall.

`FTPSRV_CLIENTS <number of clients>`

This option is used to set the maximum number of clients that can be connected at one time. The default is three clients. The maximum number of connected clients is 10.

`FTPSRV_MOTD_FILE <path_and_filename>`

This should be set to a full path and filename of a file that will be used to great new users when they first login to the FTP server. The file should consist of plain ASCII text with lines no longer than 70 characters. The total file size should be no more than 600 bytes.

`FTPSRV_FILEBUFFER_SIZE <n>`

Each connected client has a buffer used for reading and writing files. The default size is 8KB. You can set it between 4KB and 16KB in 1KB increments. Larger buffer sizes perform slightly better, but on small systems the total amount of buffer space can add up.

`FTPSRV_TCPBUFFER_SIZE <n>`

Each connected client has a buffer used for receiving data on the TCP/IP data socket. The default size is 8KB. You can set it between 4KB and 16KB in 1KB increments. Larger buffer sizes perform slightly better, but on small systems the total amount of buffer space can add up.

Buffer sizes below 8KB might cause flow control problems with faster systems.

`FTPSRV_PACKETS_PER_POLL <n>`

This is an advanced option that only makes sense on smaller systems.

The FTP server checks the packet driver for new incoming packets to process each time through the main program loop. The most efficient way to process packets is to scoop up any available packets that might be available and process them all at the same time. While efficient, this can delay the sending of outgoing packets for a short time. On slow systems that delay can cause flow control problems with faster systems.

This setting tells the FTP server how many packets to process each time through the main program loop. The default is 1, which is safe for any system. If you want to experiment with larger values for better performance you can set it as high as 10.

```
FTPSRV_EXCLUDE_DRIVES <drive_letters>
```

This option allows you to exclude specific drive letters from the scan for valid drive letters that is done when FTPSRV starts. This is useful if you want FTPSRV things like network drives or floppy drives. For example:

```
FTPSRV_EXCLUDE_DRIVES AB
```

will tell FTPSRV to ignore drive letters A and B, making them unavailable to users even if they have diskettes in them.

Choosing the number of clients

It doesn't take a lot of CPU to have a connected client. In theory a low end machine can handle many connected clients. The problems start when those clients try to transfer large amounts of data.

If you have multiple clients transferring files at the same time the clients will be competing with each other for the following:

- CPU time
- Disk I/O
- Network I/O

Clients that are browsing directory listings are not a burden on the system. Clients that are actively transferring files will split these resources. Depending on the speed of your system multiple active clients may reduce transfer speeds to an unacceptable level.

At the lowest end of the spectrum, an 8088 based PC XT with the original MFM hard drive and a reasonable Ethernet card should be able to service one file transfer at a rate of 30KB/sec. Multiple clients will split that bandwidth.

A higher end system like a 386-40 with an IDE hard drive can service one file transfer at a rate around 400KB/sec. Multiple clients splitting this bandwidth is obviously not as much of an issue.

Besides the available bandwidth memory requirements are another consideration. Each connected client requires around 11KB. If a file transfer is in progress the client will require another 8KB for the length of time the transfer is in progress. Since it is unlikely that every client will try to transfer a file at the same time, you can probably get by with less than the worst case memory scenario. I haven't measured it precisely, but you should not need more than 400KB in the worst case.

Setting up the password file

The location of the password file is set by the FTPSRV_PASSWORD_FILE configuration parameter described in "Configuring the mTCP FTP server."

The password file is used to define which users are allowed to connect to the FTP server and what directories and functions they can use. There is one entry per line. Entries are in the following format:

```
   userid password sandbox_directory incoming_directory permissions
```

`userid`

This can be up to 10 characters in length.

`password`

Passwords can be up to 10 characters in length. You may choose any reasonable mix of characters for a password; one notable exception is that spaces are not allowed.

If you use the keyword "[email]" (without the quotes) the FTP server will prompt for an email address instead of a password and will permit any user input except a blank line. The "[email]" keyword is used to implement anonymous FTP; be sure to read "Setting up Anonymous FTP" before setting up anonymous FTP.

FTP passwords are always transmitted "in the clear" on the wire, so they are not terribly secure.

`sandbox_directory`

If you wish to restrict a user to a specific directory (and all of its subdirectories) set this field to be the fully qualified drive letter and path of the directory. If you do not want a user restricted use the keyword "[none]" without the quotes but with the brackets.

The drive and path format is not in the standard DOS format. The format used by the program is:

/drive_X/path

where X is a DOS drive letter. For example, D:\FTP\INCOMING is expressed as /drive_d/ftp/incoming.

When a sandbox is in use the user will be unaware that they are being restricted to a sandbox. They will just see the designated directory as the top of the directory hierarchy. For example, if you specify a sandbox directory of "/drive_c/ftpdata" the user will only see that directory as "/". The FTP server automatically puts "/drive_c/ftpdata" in front of any requests, and it prevents the user from trying to escape the sandbox.

The sandbox must be specified using a full path, including drive letter, in the format described above.

If a user is not restricted to a sandbox they are free to move to any directory and drive letter on the system. To switch drives the user must use a path in the format described above. For example, to switch to drive E: the user would enter "cd /drive_e/". This syntax looks like a standard Unix path, which provides the best compatibility with the widest range of clients.

(Previous versions of the FTP server used a more DOS-like syntax. But compatibility with clients was a nightmare. This works much better now ... -Mike)

`incoming_directory`

If you want to restrict uploads to a specific directory enter the directory name here. If the user tries to upload a file in a different directory the request will be denied.

If the user is also in a sandbox then the incoming directory specified here should be a relative pathname within the sandbox. For example, if the sandbox is "/drive_c/ftpdata" and this parameter is "/incoming", then all uploaded files will be deposited in "/drive_c/ftpdata/incoming". (The user will see it as "/incoming" due to the way the sandbox works.)

If the user is not in a sandbox then the incoming directory should be an absolute pathname including a drive letter in the same format described in "Sandbox directory".

If the user is not to be restricted then use the keyword "[any]" (without the quotes).

`permissions`

This field is used to control permission to commands that might alter the filesystem. You can choose to grant permission to all commands by using the "all" keyword (without the quotes). If you don't use "all" you must grant permissions to each command individually.

The commands you can grant permission to are:

DELE	delete a file
MKD	create a directory
RMD	remove a directory
RNFR	rename from (this is used to start a file rename)
STOR	store (receive a file from the client)
APPE	append (create a new file or append to an existing file)
STOU	store unique (receive a file form the client, but the server picks a unique filename.)

If you do not grant permission to a command the user can not use it. The other FTP commands for changing directories, getting directory lists, changing modes, etc. are always available.

Here is a sample password file that demonstrates a named user with no restrictions, a named user with restrictions on uploading, an anonymous account with uploading, and a read-only user with no uploading:

```
brutman    passw0rd  [none]        [any]                 all
tester     b0guspw   [none]        /drive_e/ftp/incoming  stor stou
anonymous  [email]   /drive_e/ftp  /incoming              stor stou
readonly   [email]   /drive_e/ftp  [any]
```

The first user has no restrictions on directories and permission to use all of the filesystem altering commands. The second user can see the entire system but is restricted to only uploading in one directory, and they can only put files - they can't delete, create or remove directories, or append to existing files. The third user is restricted to one directory and its subdirectories (a sandbox) and may only upload to one directory within that sandbox. The last user can only look in one directory and its subdirectories (a sandbox) and may not upload. (Ignore the [any] field, it's just holding a place.)

Setting up Anonymous FTP

Setting up anonymous FTP is fairly easy but you should be aware of the potential security problems if you do it wrong.

First, as an operating system DOS has no real user permission system. The FTP server can do anything, and by extension your users can do anything that the FTP server allows them to. The operating system does nothing to help keep things secure.

An anonymous user account is created when you set the password field of a user account to "[email]". The userid is traditionally named "anonymous" or "ftp", but it can be any name if the password field is set to "[email]". Setting the password field to "[email]" turns off all password checking for that user and simply records what the user enters for the password as their email address.

It is absolutely necessary to lock anonymous users into a sandbox. If you do not, anonymous users will be able to look in any drive and any subdirectory on your system. Depending on how you set the permissions to the FTP commands, it would be possible for an anonymous user to trash your system.

If you plan on letting anonymous users upload you should designate a directory within the sandbox that will be used to hold all uploaded files. That allows you to easily identify and sort new files in a single place instead of having them spread through the sandbox.

The recommended user permissions for anonymous users are STOR and STOU. This lets them upload files with a unique name that they choose, or with a unique name that the server chooses. mTCP FTP server will not overwrite an existing file so both STOR and STOU are safe for existing data. APPE (append) is not safe; it can alter existing files, so it is not recommended for anonymous FTP users.

If you do not plan on accepting uploads from anonymous users then do not even give them STOR and STOU. They will still be able to retrieve files from anywhere in the sandbox.

Below are sample lines that can be used in the password file to implement anonymous FTP.

```
anonymous [email]   /drive_e/ftpdata  /incoming stor stou
ftp       [email]   /drive_e/ftpdata  /incoming stor stou
```

These lines assume that the e:/ftpdata is the sandbox that the FTP users will be using. Store (STOR) and Store Unique (STOU) are granted, and the user has to be in the /incoming directory to upload files. (With the sandbox taken into account the full path for the incoming directory is /drive_e/ftpdata/incoming.)

Two different users with the same sandbox, incoming directory and permissions are used because anonymous FTP usually honors both "anonymous" and "ftp" as usernames.

It is possible to assign different users to different sandboxes; there is no limitation on the number of sandboxes on a system. (The sandbox and upload directory are a per user setting.) You can use this to have different classes of anonymous users; perhaps one class is not allowed to upload, while a second and slightly more privileged class is allowed to upload.

DOS filesystem limitations

For those of you used to Unix or modern Windows systems this is going to be a shock. Here is the brief run-down on the DOS filesystem and what the limits are:

A fully qualified filename in DOS looks like this:

```
   d:\path\filename.ext
```

The maximum length for a fully qualified filename is 78 characters. The components are:

drive letter	2 characters (letter plus a colon)
path	63 characters including a beginning and ending "\"
filename	13 characters in 8.3 format

The official directory delimiter is always a "\" (backslash).

Valid characters for a filename are all letters, numbers, the following punctuation characters !@#$%^&()-_{}`'~*? and all "high bit" characters numbered ASCII 128 and above. The rendering of high bit characters depends on your active DOS code page - mTCP FTP server assigns no meaning to them at all.

If your user tries to use an invalid path or filename their request will be rejected. Gently remind them that they are on DOS, and that 8.3 isn't just a pain in the rear, it's the law.

To keep compatibility with the largest number of FTP clients the FTP server doesn't show DOS style drive letters and paths to connected users. It rewrites the drive letter part to look like /drive_X/ where X is the drive letter and the path delimiters use forward slashes, not backslashes. The FTP server also presents the image of a root file directory '/' where the subdirectories are the available drive letters.

A connected user must use a full path (in /drive_x/path/ form) to change directories. If a relative path is used it is relative to the current drive letter and path for that user.

Using mTCP FTP server with a firewall

If you plan on allowing users to use your FTP server through a firewall, read on. If you are not allowing access from people outside of your LAN you can skip this section.

Most firewalls are designed with the needs of FTP clients in mind. Now you will be running a server, where things don't work so well.

Choosing a control port to listen on

If your ISP allows it use the standard FTP port (21).

If your ISP does not allow it or you want to improve your security slightly choose an alternate port. ISPs generally don't block ports numbered 1024 and higher. While it works, non-standard ports are not as convenient for your clients to use. (Your users will have to find a way to specify the new port number in their client.)

Setup your firewall to forward the control port

Most firewalls block incoming connections from the internet. This makes sense, as your personal machines behind the firewall will make whatever outgoing connections they need. This doesn't work when you run a server though. You have to be able to allow incoming connections from the outside world to get to your FTP server.

Most firewalls allow you to specify where incoming connections from the outside world should be forwarded to. The forwarding is done based on port numbers. Setup your firewall to forward incoming connections on your chosen control port to the machine running the FTP server.

Setup your firewall to forward data ports

Clients connecting to your FTP server from the outside world are probably behind firewalls that also do not allow for incoming connections. They will probably need to use passive mode for data connections, which means that they expect to make incoming connections to your FTP server.

By default mTCP FTP server will use ports 2048 through 3071 for data connections. (The starting port number and the number of ports to use can be configured if the defaults are not suitable.) You will need to configure your firewall to pass all of these port numbers through to the FTP server.

Tell mTCP FTP server your public IP address

If you are behind a firewall you are probably using a private, non-routable address assigned by your firewall. This address was obtained using DHCP or configured by hand. The firewall is responsible for translating incoming and outgoing IP packets between the private non-routable addresses and the public address assigned by your ISP.

This mapping is called "Network Address Translation" and it is normally transparent to you. There is a snag when running an FTP server; when a client requests a passive connection for a data transfer the server will respond with an IP address and port that the client should connect to. The FTP server knows its IP address, which is the local non-routable address on your home network. Without any intervention that is the address it will give to the connected client requesting a passive mode connection. Clearly that address is wrong for an external client.

To get around this you can use the FTPSRV_EXT_IPADDR configuration parameter to tell mTCP FTP server what address it should send to clients instead of the private, non-routable address that it knows. The address to use is the public address that your ISP has assigned to your firewall/cable modem/DSL modem.

A commercial grade firewall will modify the FTP server passive response automatically, but home grade firewalls will not. When a server is running on a non-standard port almost no firewalls will be able to fix the problem. FTPSRV_EXT_IPADDR gives you a work around.

mTCP FTP server is smart enough to know if a client is on the same internal LAN as it is, or if the client is coming from a different LAN. If the client is on the same LAN as the server then the passive command response will use the local non-routable private address. That way your local machines can use passive mode connections too.

Starting, running and ending the FTP server

This assumes that your packet driver is loaded and that the basic work to setup mTCP is done. (You should have the MTCPCFG environment variable pointing at a valid mTCP configuration file, and you should have tested your network connectivity.)

Starting

To run ftpsrv, just go to the directory where it is located and run it. Ftpsrv will read the mTCP configuration file, and then scan the password file. If all looks good and it can get the memory that it needs, it will start running. If anything fails you should get an error message that tells you what is wrong.

The three most common reasons for it to fail are:

- The mTCP configuration file can not be found or is not correct.
- The password file has an error in it. Not all errors will be spotted, but it has to look like a reasonable password file.
- You don't have enough memory available.

If mTCP starts running it will give you a set of messages, and then it will start to use the screen to log events. Events will also be written to a log file on disk specified by the configuration file, if desired.

Successful start-up will look something like this:

```
mTCP FtpSrv by M Brutman (mbbrutman@gmail.com) (C)opyright 2009-2015
  Version: Sep 28 2011

Opening password file at e:\ftppass.txt
  Password file looks reasonable.

mTCP FtpSrv version (Sep 28 2011) starting

Clients: 10, Client file buffer size: 8192, TCP buffer size: 8192
Packets per poll: 1, TCP sockets: 21, Send buffs: 35, Recv buffs: 40
Client session timeout: 120 seconds
Control port: 21, Pasv ports: 2048-3071
Real IP address: 192.168.2.100, Pasv response IP addr: 8.8.8.8

Press [Ctrl-C] or [Alt-X] to end the server
```

The Pasv response IP address and other parameters will be different for your machine and configuration.

The top line of the screen has a small status line that shows you the total number of connections and the number of active sessions. The rest of the screen is used for log messages.

At this point your server is ready for incoming connections.

Running

The following keys are recognized on the console:

Alt-B	Toggle the beeper that signals when users sign on
Alt-H	Help
Alt-S	Show statistics
Alt-X	Exit

There is not much to do while the server is running. I would turn off the monitor and check on it once in a while by connecting to it with an FTP command and using the SITE STATS or SITE WHO command.

If you are accepting uploaded files you should check for new files and process them once in a while. You can do this by downloading the files to another machine, examining them, and then either moving them using the rename command or deleting them from the server. You should have a rough idea of how much free space you started with; file uploads will fail if you run out of disk space. You can check the amount of available disk space with the SITE DISKFREE command.

With five or less clients configured the mTCP FTP allocated enough TCP/IP sockets to cover every possible combination of connections. After five users it is assumed that not every client will try to do a data transfer at the same time, so the worst case number of TCP/IP sockets are not created. If by some freak accident every connected client does try to do a passive mode data connection at the same time some of them might fail - the failure will be temporary and they will be able to try again.

If more people try to connect than there are configured clients each new connection will get a "421 - please try again later" message.

The default timeout period for an inactive session is three minutes. You can set it as low as one minute or as high as 10 minutes. An inactive user should be disconnected within 10 seconds after exceeding the limit.

Ending

To end the server hit Ctrl-C at the keyboard. Ctrl-C will be recognized within a second, and it will take up to 10 seconds to close all of the current data connections.

Notes on specific FTP clients

The FTP server presents the illusion of a Unix filesystem to connected clients, even while allowing users to switch drive letters. Hiding the fact that it is running under DOS keeps the clients from getting confused. As a result, there should be very few client specific problems.

Some clients send a LIST command assuming that there is a Unix system running /bin/ls on the other end. They send options with the LIST command that are supposed to be interpreted by /bin/ls, but look like a valid DOS filename. (Hint: A '-' character is valid as the first letter of a DOS filename.) The FTP standard does not have a mechanism for sending options on a LIST command so these clients are technically broken.

Previous versions of FTP server did not handle these options and thus it looked like clients could not get directory listings. This version ignores those options, which should improve compatibility with those clients.

When running on a slow machine it is possible that faster clients will flood the TCP/IP socket, causing the TCP/IP flow control mechanisms to kick in. Sometimes they kick in too hard and make file uploads very unpredictable. If this happens you can increase the file buffer and TCP receive buffer sizes used, and if your client supports it throttle the rate at which it sends data. (FileZilla is one client that can do this.)

The standard command line Windows XP FTP client does not support passive mode data connections. On an internal network this is not a problem, but if there is a firewall between you and the Windows XP machine it will be nearly worthless. Get a better client.

Internet Explorer, Firefox and Chrome work well against mTCP FTP server when an FTP URL is used to access the server.

Command line Linux clients have no known problems.

And of course the mTCP FTP client has no known problems. ;-0

Recent changes

2015-05-29	Change the logfile to be optional instead of required
2013-02-13	Enable the SIZE command; make FTPSRV 132 col aware
2012-12-23	Ignore UNIX style /bin/ls options on LIST and NLST
2012-04-29	Reduce memory utilization; add option to exclude drives
2011-09-28	Rewrite drive letter handling; fix MDTM command; rewrite screen handling; add additional configuration parameters; scan for valid drive letters at start-up; add optional MOTD file; misc bug fixes
2011-05-18	Fix parsing error on password file
2011-04-06	Change DOS path delimiter handling to make FileZilla happy
2011-01-02	First released version

Implemented RAW FTP commands

USER	Set user (only valid during initial login sequence)
PASS	Send password (only valid during initial login sequence)

RNFR	rename from (original filename)
RNTO	rename to (new filename)
DELE	delete file
CWD/XCWD	change directory
CDUP/XCUP	move "up" one directory in the hierarchy
PWD/XPWD	print working directory
MKD/XPWD	make directory
RMD/XRMD	remove directory

PASV	request passive mode
PORT	use PORT mode
ABOR	abort current data transfer
LIST	directory list
NLST	name list
RETR	retrieve (send file to client)
STOR	store (receive file from client)
STOU	store unique (receive file, but server chooses a unique name)
APPE	append to existing file (or create a new file)

MODE	set transfer mode (only stream is supported)
STRU	structure (only file is supported)
TYPE	structure (only file is supported)
SIZE	get the file size of a file (BINARY mode only)

HELP	help
ALLO	allocate (obsolete, but not an error)
FEAT	feature negotiation; will respond with MDTM
MDTM	modification type
NOOP	do nothing successfully
STAT	give server status or do a dir listing over the control socket
SYST	identify system type (responds with "215 DOS Type: L8")
SITE	do SITE specific commands

REIN	re-initialize (not supported)
ACCT	account (not supported)
REST	restart position (not supported)

SITE commands:

SITE DISKFREE	show free disk space for a given drive letter
SITE HELP	give help for the SITE command
SITE STATS	respond with some fun statistics
SITE WHO	show who is online

HTGet

Introduction

HTGet is a small utility that can fetch files served from HTTP servers. It uses HTTP 1.1 which is the current HTTP protocol that enables use with virtual hosts. HTTP 1.0 is supported for compatibility with older HTTP servers.

HTGet was inspired by a version written by Ken Yap (ken@syd.dit.csiro.au) in 1997 for the WATTCP stack. This version is a complete rewrite using some of the concepts from the original.

Special hardware or software requirements:

None

Using HTGet

Usage is like this:

```
htget [options] <URL>
```

Options are:

`-h`	Shows help text
`-v`	Print extra status messages
`-headers`	Fetch only the HTTP headers
`-m <file>`	Fetch file only if modified after <file>'s date
`-o <file>`	Write content to <file> instead of stdout
`-pass <user:password>`	Send authorization
`-10`	Use HTTP 1.0 protocol
`-11`	Use HTTP 1.1 protocol (default)

The URL format is:

```
http://yourservernamehere[:port]/path
```

The port parameter is optional - if it is not specified the default is port 80. Authentication information is not included in the URL syntax; this is slightly nonstandard but it makes the code simpler. (See the description of the -pass option below for details on how authentication is supported.)

When you run HTGet it will attempt to connect to the HTTP server, send headers, and receive content. If you do not give any options the downloaded content is sent to STDOUT. You can redirect STDOUT to a file or use the -o option to specify a filename for HTGet to write to directly.

Using -v gives you extra messages that tell you how the data transfer is progressing. The extra output is sent to STDERR to avoid interfering with the content that is being set to STDOUT.

The -m option checks the modification date of the file that was specified with the -o option and sends that date to the server during the request phase. If the content on the server has not been updated since the timestamp on the local file the server will return a "304" status code and no content. If the content on the server has been updated you will get the new content instead. This option is useful for when you only want to update a file based on if it has been updated on the server. To use it you should have the TZ environment variable set. (See SNTP.TXT for notes on how to use the TZ environment variable.)

If the content you are requesting is password protected try using the -pass option. The -pass option takes a string in the form of username:password and encodes it using BASE64 before sending it to the HTTP server. The authentication type is known as "BASIC" - it is not secure but it is widely supported. If the HTTP server requires something more secure then this option probably will not work.

The current HTTP protocol version number is 1.1 which is what this program is using. The big improvement that version 1.1 adds is support for virtual hosts, which are HTTP servers that share the same IP address. Since this program is designed to run on vintage computers it is quite possible that you might run into a vintage web server using HTTP 1.0. These older servers might not understand the headers used by HTTP 1.1; if that happens the request will fail. Try using the -10 option to force the program to fall back to HTTP 1.0 if you suspect you have an ancient server that is confused by HTTP 1.1.

Examples:

```
htget -o google.htm http://www.google.com/
```

This downloads the Google search page and writes it to google.htm.

```
htget -m -o google.htm http://www.google.com/
```

This does the same thing as above, but if the copy on the server has not changed since the local copy was written it will not be downloaded and rewritten.

```
htget -pass john:secret http://someserver.com/protected/secret.file
```

This grabs the password protected file "secret.file" using "john" as the userid and "secret" as the password. The userid and password get converted to BASE64 before being sent to the server. If the server supports HTTP BASIC authentication you will get the file.

```
htget -10 http://yeancient.server.net/readme.txt
```

Force HTTP 1.0 mode for a prehistoric server that does not use HTTP 1.1.

```
htget -headers http://www.google.com/
```

Don't get the content - just look at the headers that would have been sent.

When the server sends content ...

The server can send content even when the server reports a bad return code. The content might be a page that describes the error and tells you what to do.

HTGet will not write output if there is a socket error or if the file has not been modified and you use the -m switch. But due to the way HTTP works, it will write data if it is given content, even if the content is an error message. So don't overwrite anything important assuming that the server will give you a fresh copy - it might get replaced with an error message.

Return codes

The HTTP server always returns a result code which can be used to tell you if the request was successful. Unfortunately, the range of possible return codes from the HTTP server is greater than what can be expressed in a program exit code so HTTP return codes can not be used as exit codes directly.

It is valuable to be able to determine if a request was successful or not using a program instead of having a human make the decision. To make this easier HTGet maps HTTP response codes to a smaller set of return codes. Here are some of the mappings:

0	No socket errors, but unknown HTTP code (should never happen)
1	Socket or protocol error
10	HTTP response code from 100 to 199
20	HTTP response code from 200 to 299 if not shown below
21	200 OK
24	203 Non-Authoritative Information
30	HTTP response code from 300 to 399 if not shown below
32	301 Moved Permanently
35	304 Not Modified
37	307 Temporary Redirect
40	HTTP response code from 400 to 499 if not shown below
41	400 Bad Request
42	401 Unauthorized
43	403 Forbidden
44	404 Not Found
45	410 Gone
50	HTTP response code from 500 to 599 if not shown below
51	500 Internal Server Error
53	503 Service Unavailable
54	505 HTTP Version Not Supported
55	509 Bandwidth Limit Exceeded

The general idea is to give the common response codes a unique return code and compress the rest into a generic return codes. For a complete list of the mappings refer to the source code.

Differences with the original WATTCP HTGet

As mentioned earlier, this is a complete rewrite of the original HTGet. A few variable names survived - not much else. Here are the major differences:

- HTTP 1.1 request support with a switch to use HTTP 1.0 if necessary
- The ability to hit Ctrl-Break, Ctrl-C or ESC to abort a transfer
- Far better performance due to better buffer handling.
- Return codes that try to give a better indication of what happened.
- This version uses mTCP :-)

Recent changes

2011-07-17	Initial version

HTTPServ

Introduction

HTTPServ is a small HTTP (web) server designed for early IBM PCs and similar machines. Features include:

- HTTP/0.9, HTTP/1.0 and HTTP/1.1 support
- HTTP/1.0 and HTTP/1.1 persistent connections
- HTTP/1.1 request pipelining support
- Up to 8 simultaneous requests can be serviced with up to 8 more queued
- Ability to improve performance by serving pre-compressed content
- Easy mapping of incoming long URL path components to DOS 8.3 filenames
- HTTP Basic Authentication
- Improved performance through caching of directories and HTACCESS files
- Optional server status page
- Caching of selected files in RAM
- Logging

HTTPServ is not a replacement for a modern HTTP server but if you are looking for an easy way to make the files on an old machine browsable from modern machine then it should work well for you. And if you are a hardcore retro-computing enthusiast, you can probably serve your static HTML pages directly from an old machine.

Special hardware or software requirements:

- More memory: At least 256KB available memory, 512KB or more is recommended
- A mass storage device; serving from floppies will work but you are going to have a bad time.

HTTPServ needs more memory than most mTCP applications because it is assumed that it should try to handle multiple requests from connected clients at the same time instead of just queuing the requests and making the connected clients wait until it is their turn. You can use configuration options to reduce memory requirements at the cost of some performance and reduced function.

Unless you have very little content to server you will need a mass storage device such as a hard drive or a Zip drive. The faster the device is, the better – it is assumed that most of the objects being served will be served from disk storage and will not be cached in memory.

On older 8088 class machines some versions of DOS are extremely slow to calculate the free space on the hard drive. This can lead to long pauses where HTTPServ is unable to do any work, possibly causing TCP/IP errors. DOS 5 and later versions seem to be affected the most. For these slower machines DOS 3.3 is recommended.

Using HTTPServ

The HTTPServ command line looks like this:

```
httpserv [options]
```

Options are:

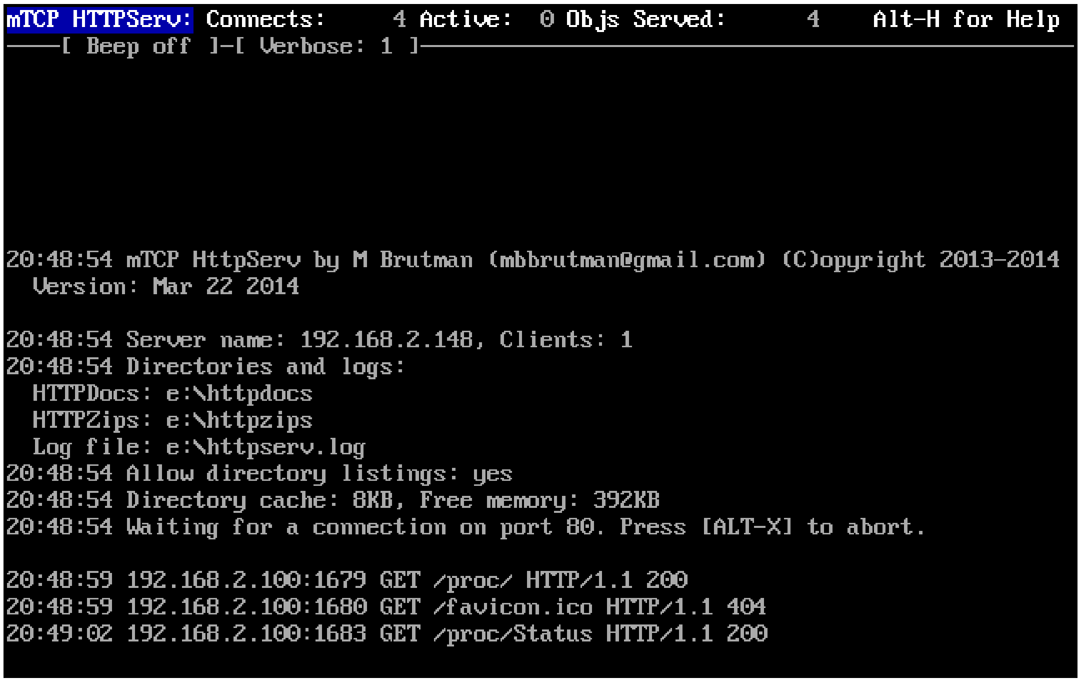
`-help`	Shows help for the command line
`-clients <n>`	Set the number of concurrent clients supported (default=8)
`-dir_indexes`	Enable generating directory listings (default is not to)
`-doc_root <dir>`	Set dir as the document root
`-port <n>`	Use port n for incoming requests (default is port 80)

Although command line options are provided, the configuration options in the mTCP configuration file should be used instead. Command line options are treated as a temporary override for anything found in the configuration file. This allows you to do quick tests or run HTTPServ for quick file transfers without altering your normal configuration options in the configuration file.

For a more detailed discussion of these options see their corresponding configuration file options in the "Configuration options" section.

Console messages and controls

The HTTPServ console is designed to show you what is going on at the moment. At a glance you can see the total number of connections served, the total number of objects served, and the current number of active connections. Recent requests and error messages are shown on the screen.



```
mTCP HTTPServ: Connects:     4 Active:  0 Objs Served:      4    Alt-H for Help
────[ Beep off ]─[ Verbose: 1 ]─────────────────────────────────────────────────






20:48:54 mTCP HttpServ by M Brutman (mbbrutman@gmail.com) (C)opyright 2013-2014
  Version: Mar 22 2014

20:48:54 Server name: 192.168.2.148, Clients: 1
20:48:54 Directories and logs:
  HTTPDocs: e:\httpdocs
  HTTPZips: e:\httpzips
  Log file: e:\httpserv.log
20:48:54 Allow directory listings: yes
20:48:54 Directory cache: 8KB, Free memory: 392KB
20:48:54 Waiting for a connection on port 80. Press [ALT-X] to abort.

20:48:59 192.168.2.100:1679 GET /proc/ HTTP/1.1 200
20:48:59 192.168.2.100:1680 GET /favicon.ico HTTP/1.1 404
20:49:02 192.168.2.100:1683 GET /proc/Status HTTP/1.1 200
```

The following key combinations can be used:

Alt-H	Show help text
Alt-B	Toggle beeping on and off
Alt-V	Set the verbosity level
Alt-X	Shutdown the server

The state of the beeper and the verbosity level are shown on the second line of the screen.

The default verbosity level is 1, which gives you an overview of what each request is and what the return code was. At verbosity level two you can see some of the additional fields from the request, including the user-agent and referer. Verbosity level 3 includes debug messages which are probably not very interesting.

All messages from the server are logged to the log file; the verbosity level just determines what gets shown on the screen. (Well, almost. Debug messages do not get logged unless you are also showing them on the screen.)

Serving static content

Like on other servers, the content that you want to serve is located under a directory called the document root. On HTTPServ you can set the document root using the command line or in the configuration file. All URLs served by HTTPServ are relative to the document root.

Remember, you are on a DOS machine so there are some limits you have to observe:

- DOS filenames and subdirectories have to use the 8.3 filename.ext format.
- DOS filenames are not case sensitive.
- DOS dos not like filenames that contain special characters, like spaces

Serving content is a fairly straightforward process. The URL should identify a directory or filename relative to the document root. Just start at the document root, open the directory or filename specified by the URL, and send the bytes over the socket. If the URL does not exist send back an error message instead. In reality serving is more complex, but this is the basic idea.

If you are designing content from scratch then all of the links within your content that are on the same server will follow the DOS filename conventions. This can make the links look a little goofy because they will be terse compared to other web sites, but it works.

If you want to use URLs that look more like normal URLs HTTPServ has a trick you can use. Your files still have to meet the DOS 8.3 naming requirement but you can define longer aliases that map to the files. That allows you use use links like "http://servername/ThisIsALongSubdir/MyContent.html" instead of "http://servername/long.sub/mcontent.htm". The magic happens each time a request comes in; the parts of the URL are parsed and optional mappings from long names to short DOS 8.3 filenames are consulted. If a mapping exists the server substitutes in the proper DOS 8.3 filename automatically.

To use this feature please see the "Map" directive in the HTACCESS files appendix. Using HTACCESS files can slow things down and they require more memory, but when used correctly the performance and memory costs are minimal. This feature also makes it easier to move existing content to a DOS machine for serving without having to rewrite all of the links within the content.

Serving precompressed static content

Sending data over a network requires both network bandwidth and CPU usage. The CPU usage to send and receive data is roughly proportional to the amount of data being sent.

Compressing data before sending it requires additional CPU usage for the compression step but it can result in less CPU used during the transmit stage and less network bandwidth required. Depending on your network connection and CPU speed, compressing content before sending it can make sense. Modern HTTP servers usually choose to compress content because their CPUs are so powerful and preserving network bandwidth is more important.

HTTPServ can not compress content on-the-fly yet, but it can serve compressed content if you prepare it ahead of time. Any content that you choose to compress can be stored on disk in a directory tree that parallels the HTTPDocs directory hierarchy. This parallel directory tree is known as HTTPZips.

At runtime when a client makes a request HTTPServ will search the normal document root (HTTPDocs) for the file the user requested. If the client indicates that it can accept GZIPed content HTTPServ will then check HTTPZips for a compressed version of the file. If a compressed version of the file is found it will be sent instead of the standard version of the file. If no compressed version is found, the standard version of the file is sent instead.

HTTPServ expects the directory hierarchy in HTTPZips to be the same as it is in HTTPDocs. It will look for the compressed version of the file using the same path, substituting your HTTPZips directory for your HTTPDocs directory when looking for compressed content. If the compressed file is not in the same relative place it will not be found. (Note: HTACCESS files can be used to alter how URLs are mapped to directories. Serving compressed content does not change this; the HTACCESS files in the the normal document root are still used. The only change that happens is that instead of reading the final target out of HTTPDocs HTTPZips will be tried instead.)

HTTPServ also has no idea if the compressed version of a file is equivalent to the standard version of the file. You need to make sure that the two stay in sync; if you update a file in HTTPDocs then you should updated the compressed version in HTTPZips or delete it until you can update it.

To make compressed content available follow these high-level steps:

- Create an HTTPZips directory and subdirectories under it that mirrors your HTTPDocs directory hierarchy.
- Gzip files that will compress easily; human readable text files are good candidates. JPG, ZIP archives, and other files that are already compressed will not benefit from further compression.
- DOS filenames are limited to the standard 8.3 format. Convention states that gzip'ed files should have a ".gz" suffix on them, but that is not possible in DOS. That is why HTTPZips is a separate parallel directory hierarchy - it is assumed that everything located in HTTPZips has a ".gz" suffix on it.

Pre-compressing the content requires additional disk space and some extra file management. But if you have the space I recommend doing it; it cuts down on the amount of CPU time required for requests that can be served from the compressed copies, giving the clients faster response time and letting you serve more potential clients.

The use of HTTPZIPS is completely optional. If you are only an occasional user of HTTPServ then it probably is not worth the extra work.

Protecting files and directories

HTTPServ supports HTTP Basic Access Authentication. With "basic auth" there are no cookies, sessions or login pages. But there really is no security either; the userID and password is transmitted by the web browser basically "in the clear." This means that basic auth is best suited to things that you want to lightly restrict access too, but nothing that would be harmful if it were more widely available.

To protect a file using basic auth you mark the file as belonging to a realm. A realm is a tag (often a description) that allows you to logically group files that need protection together. Each realm has a password and a list of users associated with it.

When a user requests access to a protected file a challenge is sent back asking for a userID and password. The challenge generally appears as a dialog box on the web browser with a message that includes the realm the file is part of. The user then enters a userID and password and the browser retries the request, sending the information along in an additional HTTP header. If the entered userID and password are correct for the realm the user is granted permission to view the content.

Web browsers often cache the entered userID and password for a few minutes. If the user tries to access another file protected by the same realm the caching allows the web browser to retry the request without prompting the user for the userID and password again. (If the realm different or not in the cache then the user will get the dialog box.)

Files and directories marked with a particular realm do not need to be located near each other; realm is a tag that can be applied wherever it is needed.

HTTPServ lets you protect a file or a directory using the FileAuth directive in the HTACCESS.--- file where the file or directory resides. One very important thing to be aware of is that if you mark a directory as being protected, it is assumed that if the user gets the userID and password correct that everything under that directory (files and subdirectories) is also available. This is true even if those files and directories have different protection specified - that protection is ignored because the user has been granted access at a higher level. This is a shortcoming of basic auth - only one authorization header can be sent for a request so it is not possible to provide layers of protection for files and directories along a URI path.

Realms are defined in a text file that you provide. The HTTPSERV_REALMS configuration directive in the mTCP configuration file tells HTTPServ where to read the file. Each line in the realms file defines a realm, the password and a list of users authorized to the realm. Below is a sample REALMS.TXT file:

In your mTCP configuration:

```
HTTPSERV_REALMS e:\realms.txt
```

In e:\realms.txt:

```
"Server functions" s3cr3t   root
"Restricted Files" passw0rd brutman eddie
"Restricted Dirs"  pfft123  brutman warez anon
```

In this example three different realms are declared. The realms have spaces in their names so they are surrounded by quotes to indicate that both words are part of the realm name. Next comes the password and after comes a list of users that are valid for the realm. All fields are delimited by spaces and have the following length restrictions:

- realm name: Up to 50 characters (not including quotes)
- passwords: Up to 12 characters
- userids: Up to 16 characters

The length of each line is limited so don't go overboard. (As of this writing the maximum line length is 255 bytes.)

Do not locate your realms file under HTTPDOCS where it can be served. That's just an accident waiting to happen.

Special server functions

Besides serving static files HTTPServ has some special functions that it can perform. The special functions can be used to show status pages or make the server perform operations such as caching content in memory or performing a shutdown.

By default all of the functions are enabled but there is no way to get to them. There is no hard-coded URL that you can use and find the functions. To enable a function you create an pseudo filename entry for it in a directory using the HTACCESS file for that directory. This seems complicated but it lets you define what the URL path to use to get to function and allows you to protect the function using the existing HTTP Basic Authentication method.

The appendix on HTACCESS files shows you the syntax for enabling these functions. The appendix called Sample files for exposing the special functions gives a concrete example of how to set these special functions up. What follows next is a description of the functions.

HTTPServ Status Page

This is a useful but harmless page that gives you statistics such as how long the server has been running, how much memory is available, how many objects have been served, TCP/IP statistics, etc.

HTTPServ Clients Page

This page shows you the IP address of currently active connections, and if possible what request is being served on each of them. You can only see active requests which limits the usefulness of the page. (In the future this page might be removed.)

File cache functions

HTTPServ lets you use RAM to cache files. If you choose the files to cache carefully this can really speed things up, especially on older systems with slow disk access.

Three cache functions are provided - one to read a file into memory, one to tell you what files are in memory, and one to clear all of the files from the cache. The cache functions are implemented with one handler that uses

URL parameters to pass commands to the server.

To add a file to the cache send a parameter with the name "add" and the URL of the file to cache as the value of the parameter. HTTPServ will process the URL and resolve it to a DOS file, including doing any long name to DOS filename mapping specified in the HTACCESS files. For example:

```
http://192.168.2.148/proc/cache?add=%2Fpcjr.css
```

In this example I have a subdirectory called /proc hanging off my HTTPDOCS directory. The HTACCESS file for /proc defines a pseudo-filename called "cache" to enable the caching function. The "/pcjr.css" object (relative from the document root) is to be added to the cache. The string looks horrible because the special characters need to be escaped in the URL. An easier way to do this is to provide a simple HTML file in the directory with a form that looks like this:

```
<form action="cache" method="get">
  <INPUT type="text" name="add" />
  <INPUT type="submit" />
</form>
```

The form version is much easier to use and the browser handles escaping the special characters for you. The form method must be "get" as HTTPServ does not handle the post method yet.

When you add a file to the cache HTTPServ will check the document root to see if it exists. If the file does exist it will be read into memory and served from memory from that point forward. If a gzipped version is available in HTTPZips that version will be cached instead; most modern clients will request compressed content and this saves memory in RAM.

Remember, even though DOS is not case sensitive URL paths are. Make sure you use the proper combination of upper and lower case when specifying the URL paths of files to cache.

To see the list of files in the cache send a parameter with the name "show" and no value. For example:

```
http://192.168.2.148/proc/cache?show=
```

A form can also be used to simplify this:

```
<form action="cache" method="get">
  <INPUT type="hidden" name="show" value="">
  Show cached files: <INPUT type="submit"/>
</form>
```

The cache clear command is similar; the parameter name to use is "clear". Individual files can not be cleared from the cache; the entire cache must be cleared at the same time. This is done to prevent memory fragmentation in the DOS heap.

Remote shutdown function

The remote shutdown (quit) function does what you think that it does. Be careful; if you enable this you must protect it with HTTP Basic Authentication or anybody who finds the link will be able to shut your server down remotely. And there is no "Are you sure?" prompt, so if you do invoke this function the server will shut down and return to DOS.

Serving to the public Internet

Teaching you how to run a public HTTP server is beyond the scope of this document. I assume a basic knowledge of networking and security. These are just a few reminders and things you should be aware of.

Security, Security, Security ...

If you connect a machine to a network you open it up to a wide array of possible problems. Viruses or malware may try to infect your machine. People might try to find sensitive information on your machine. People might try to crash your machine just for fun.

I have made great efforts to protect against these things but nothing is perfect. You are responsible for not exposing sensitive information; if you don't want it accidentally published don't put a web server on the machine. I have tried to protect against buffer overruns, malicious URLs, etc. but this work pretty much never ends which is why it needs to be tested and bugs need to be reported. And lastly, if somebody decides to DoS (Denial of Service) your DOS machine with a cell phone or a faster machine, there is not much that can be done.

You take responsibility for running a web server. It can be fun, but it takes some care. Here are some things I've done to make HTTPServ safer:

- If something is protected with HTTP Basic Authentication then the default is to fail to serve the file unless the userid and password for the realm match up.
- HTACCESS.--- files are never served.
- The server aggressively defends against buffer overflow attacks.
- The server does not allow access to special filenames, like LPT1:, $CLOCK, etc. If it is not a file or a directory, it will not be served.
- The server times idle connections out automatically, and more quickly if connections are limited.
- A fair amount of input checking is done to weed out mal-formed requests.

There are probably things that will break and I will fix them as they come up.

Your ISP probably hates private servers

Most residential grade internet service providers prohibit running servers on your network connection. The usual method of enforcement is to block incoming requests to "well known ports" that correspond to services that you might want to run.

The usual way around this restriction is to run your service on an alternative port. For example, the standard port for HTTP traffic is port 80. Incoming requests to port 80 will be blocked on your connection but if you choose to use port 8080 instead the traffic will probably not be blocked. (And for real vintage computing street cred, run on port 8088.)

The HTTPSERV_PORT configuration option can be used to tell HTTPServ what port to use for incoming requests. If you use a non-standard port the port must be included in any URLs that point to your server. For example, while http://yourserver.net/ works for a server running on the standard HTTP port the same server using port 8080 would need a URL that looks like http://yourserver.net:8080/ .

Configuring your home firewall

Most people have a home firewall of some sort that allows you to have multiple machines in your house sharing the same connection from your ISP. While this is a valuable function the firewall also serves another valuable purpose - it generally blocks incoming requests from arbitrary machines on the Internet. One reason for doing this is that a request from an unknown machine might be hostile. The other reason for doing this is that the firewall really does not know which machine on your network should handle a particular incoming request. Dropping those incoming requests helps improve security but it becomes a problem when you are trying to run a web server; all of the traffic to the web server looks like incoming requests from possibly hostile machines on the Internet.

Most home firewalls can be configured to allow incoming requests on the public IP address on specific ports to be sent to a specific machine and port number on your internal network. This solves the technical problem of where to route incoming requests to. The forwarding is done on a port by port basis so you generally tell the firewall to only forward anything coming in on port 80 to the specific machine in your network running the web server. Requests on other ports still get dropped.

Keep in mind that your ISP is probably blocking incoming requests to port 80 and that you probably have to use an alternate port, such as 8080. If that is the case then your port forwarding should also reflect that; incoming requests to your cable/DSL modem on port 8080 will need to be forwarded to your HTTP server. The HTTP server can be running on any port; they do not have to match. This allows you to setup HTTPServ to run on port 80 on your private LAN and only people connecting from outside will have to use the non-standard port that you chose to get around your ISP. (The firewall handles the forwarding of the traffic including the mapping of ports automatically.)

Telling people your server name

On your internal network your server name is the IP address of the machine or a hostname if you have DNS configured internally. But users outside of your LAN on the other side of your firewall will see the world differently. They will connect to your server using the IP address assigned to you by your ISP and they will also probably have to use a non-standard port number.

In general this is not a problem. A request will come to your firewall on the public address. Your firewall will forward the request to your server. The server will respond, and everything is good. The problems arise when your server has to tell the outside person what address to use for followup requests. For example, if a person specifies a URL that maps to a subdirectory then they are supposed to provide a trailing slash ('/') to indicate that it is a directory. If they omit the trailing slash the server will respond with a 301 to tell them the correct URL to use, and those headers will also include the server name. If the server name is not accurate the followup request will fail.

The default server name is the IP address of the machine and the port you are listening on. This works fine on your internal LAN but it does not work for users on the other side of your firewall.

To get around this set the HTTPSERV_SERVERNAME configuration option to be the IP address and port that you are telling people to connect to. For example:

- If the IP address of your HTTP server on your internal LAN is 192.168.1.124 and it is listening on port 80 and …
- Your cable/DSL modem (firewall) has a public address of 67.43.212.82 and …

- Your ISP blocks port 80 so you are using port 8080 for connections from the public Internet and you have set your router up to forward them to your HTTP server, then ...
- Your HTTPSERV_SERVERNAME should be set to 67.43.212.82:8080 .

That way, if your HTTP server has to respond with a 301 HTTP header it will give the correct address to use for the followup connection. If you don't set HTTPSERV_SERVERNAME then it uses the local IP address of the machine (192.168.1.124:80 in this example) which makes no sense to an outside user.

Miscellaneous tips

Choosing files to cache in memory

Since most clients support gzipped content you should cache only gzipped versions of your content to stretch your memory; clients that do not support gzipped content are rare.

Cached files need to be under 64KB in size; if a file is above that size the request will be rejected. You get more bang-for-the-buck by caching several small files instead of fewer large ones. The reason for this is that there is overhead to finding and opening a file and that overhead is the same whether the file is large or small. As a fraction of total transfer time that overhead is larger for small files than it is for large files. So it makes sense to try to eliminate that overhead for small files.

Files that are used across multiple pages are good candidates for caching. This includes CSS files, small logos and icons, etc. Your site's home page and any pages that you think will be visited often are also good candidates for caching.

Log file considerations

If you are running a server for your own personal use on a private LAN then you probably do not need to have the log file enabled. Not having a log file may provide some modest speed improvement. It will also cut down on the required memory.

If you are running a public server then you really should have logging turned on.

The data written to the log file is pretty verbose. Make sure that the drive the log file resides on has enough space; a few megabytes is a safe number to start with. During my large scale test HTTPServ served 7175 requests and generated a logfile that was 6.4MB in size. That is an average of 900 bytes for each request.

In the future logs will either be written in a compressed format at runtime or there will be a "log rolling" function added to periodically close the log file, compress it, and start another one.

Do not write your log file to a RAM disk. If the machine crashes you will lose the log file.

Using disk caching software

If you are on a machine that has expanded or extended memory that can be used to cache disk operations then you should use it. Such caching software will generally cache the most used disk blocks, automatically removing blocks from the cache that are no longer being used. This makes more sense than using conventional memory to cache specific files using the cache function of HTTPServ; you have much more memory available to you if you have expanded or extended memory.

The only danger might be that the log file writes would also be cached; if something crashes or you lose power you might lose some of your logs.

HTTPServ Configuration options

HTTPServ can be run using command line arguments only; no further configuration is needed unless you want to use advanced options. If you want to set configuration options they can be placed in the mTCP configuration file pointed at by the MTCPCFG environment variable.

The following options can be used to configure HTTPServ behavior:

`HTTPSERV_HTTPDOCS`	Set the document root
`HTTPSERV_HTTPZIPS`	Set the document root for pre-compressed files
`HTTPSERV_LOG_FILE`	Set the location of the log file
`HTTPSERV_REALMS`	Set the location of the realms file for HTTP Authentication
`HTTPSERV_SERVERNAME`	Set the name used when sending responses to clients
`HTTPSERV_PORT`	Set the port used for incoming HTTP requests
`HTTPSERV_VERBOSITY`	Control how verbose the on-screen messages are
`HTTPSERV_DIR_INDEXES`	Set the default for generating directory indexes
`HTTPSERV_DIR_CACHE`	Set how much memory is used for caching HTACCESS files
`HTTPSERV_CLIENTS`	Set the number of concurrent clients to support

Each configuration option is explained in more detail below:

```
HTTPSERV_HTTPDOCS
```

Equivalent command line option: -doc_root

Format:

```
HTTPSERV_HTTPDOCS <fully qualified dos directory name>
```

Every web server has a "document root" that is a directory that serves as the root of the directory tree for files being served by the server. This configuration option is used to tell HTTPServ which directory to use as the document root.

The document root should be provided as a fully qualified path including a drive letter. Files in the directory and subdirectories within the doc root can be seen by people browsing the server; files outside of the doc root are hidden and not available.

Examples:

```
HTTPSERV_HTTPDOCS d:\httpdocs
HTTPSERV_HTTPDOCS c:\
HTTPSERV_HTTPDOCS e:\public\toshare
```

If you do not set this configuration option then you can set it on the HTTPServ command line using the "-doc_root" option. If both the -doc_root option and HTTPSERV_HTTPDOCS are specified then the -doc_root will have priority and be treated as though it was overriding HTTPSERV_HTTPDOCS.

```
HTTPSERV_HTTPZIPS
```

Equivalent command line option: Not applicable (see below)

Format:

```
HTTPSERV_HTTPZIPS <fully qualified dos directory name>
```

HTTPSERV_HTTPZIPS tells HTTPServ where your pre-compressed files are. Pre-compressed files allow for better performance at the cost of additional disk space. See Serving precompressed static content for the details on how it works.

Example:

```
HTTPSERV_HTTPZIPS d:\httpzips
```

The use of HTTPZIPS is completely optional.

HTTPSERV_HTTPZIPS is only valid if the document root is set using HTTPSERV_HTTPDOCS in the configuration file. If you use the -doc_root command line option to temporarily change the document root then serving precompressed content will be disabled. No equivalent command line option for HTTPSERV_HTTPZIPS is provided.

```
HTTPSERV_LOG_FILE
```

Equivalent command line option: Not applicable

Format:

```
HTTPSERV_LOG_FILE <fully qualified dos file name>
```

If you are going to run any sort of public server you probably want a log file so you can keep track of which IP addresses are connecting to your server and what they are doing. This configuration option is used to tell HTTPServ which file to use as the log file.

HTTPServ always appends to this file. As of this writing HTTPServ does not compress log files or rotate them so you should locate your log file on a drive letter that has plenty of available space and you should periodically check to see if the log files needs to be compressed and moved offline. Do not locate your log file in an area of the disk being served by HTTPServ - that is a potential security problem.

HTTPSERV_LOG_FILE is optional. Using it can slow the server down a little bit but if you are allowing traffic from outside of your network you really should use it. Please see Log file considerations for more discussion on the log file and logging techniques.

```
HTTPSERV_REALMS
```

Equivalent command line option: Not applicable

Format:

```
HTTPSERV_REALMS <fully qualified dos file name>
```

If you are using HTTP Basic Authentication to protect any of your files or directories then you need a realms file to define the password for each realm you are using in the HTACCESS files. Please see Protecting files and directories for the details on how to use this file.

`HTTPSERV_SERVERNAME`

Equivalent command line option: Not applicable

Format:

```
HTTPSERV_SERVERNAME <alternative_server_name>
```

HTTPServ knows the IP address of the machine that it is running on but it does not really know what the hostname of the machine is. Hostname can be ambiguous, especially for machines that are DHCP clients or that are behind firewalls. The hostname that you tell clients to use may have nothing to do with the hostname the machine is configured for, especially if port forwarding is involved.

If HTTPServ is going to serve traffic to machines outside of your immediate LAN or to machines that are on the other side of a firewall that changes network addresses then you probably did some configuration to get the requests routed to HTTPServ. Whatever host name you are telling the clients to use to connect to HTTPServ should be set on this configuration option.

One way in which this is used is when the client requests a directory but does not include a trailing slash ('/'); that requires HTTPServ to send a redirect (301 HTTP header) to let the client know that the correct URL includes a trailing slash. The redirect message needs to have a server name that the client can use.

Example:

(Assume your machine is behind a firewall that uses NAT and the address is 192.168.1.124)

```
HTTPSERV_SERVERNAME bogus.dyndns.org:8080
```

In this example you are using a dynamic DNS service and port forwarding to provide connectivity to your machine from the outside world. The real IP address does not make sense to clients outside of your network so if your server needs to send a response that includes the server name, it will use "bogus.dyndns.org:8080" instead of 192.168.1.124.

See the section entitled Serving to the public internet for a more detailed explanation of this and other firewall related issues.

`HTTPSERV_PORT`

Equivalent command line option: -port

Format:

```
HTTPSERV_PORT <n>
```

By default HTTPServ listens on the standard WWW port, which is port 80. If you need to use an alternative port use this option or the "-port" option on the command line.

`HTTPSERV_VERBOSITY`

Equivalent command line option: Not applicable

By default HTTPServ prints a minimal amount of information on the console about each request it is serving. If you would like to see more detail for each request you can set the verbosity level higher. Supported levels are "low" (the default), "medium" and "high".

Example:

```
HTTPSERV_VERBOSITY medium
```

"Medium" is actually pretty verbose. "High" is used for debug messages.

`HTTPSERV_DIR_INDEXES`

Equivalent command line option: -dir_indexes

By default HTTPServ will not generate directory listings for directories; this is for security reasons. If you would like to change the default to enable directory listings then use this option.

Enabling directory listings by default is safe and harmless if you are the only person using the server. But if you are going to allow other people to browse your machine you should disable directory listings and provide a static, per directory file that links to what you want users to be able to see. That gives you an opportunity to provide descriptions for each file next to the link to it. For some cases like a large shareware repository that might not be practical. Use your best judgment.

Also keep in mind that just because a file does not have a link to it does not mean that an external user won't be able to see it. They can always guess at links. If you don't want something served, don't put it on your server.

Legal values for this option are "yes" and "no." A setting for a specific directory in its HTACCESS file overrides this global setting.

Example:

```
HTTPSERV_DIR_INDEXES yes
```

That will enable directory indexes by default across all directories.

`HTTPSERV_DIR_CACHE`

Equivalent command line option: Not applicable

Serving files using HTTP is a fairly expensive operation for an old machine, especially when HTACCESS files are used. There may be an HTACCESS file in any directory and each HTACCESS file may have a lot of directives. Serving a file requires walking the entire path in the URL and examining each directory and possible HTACCESS file along the way.

To make this reasonable on slower machines the directory structure under HTTPDocs and the contents of HTACCESS files are cached in memory. The cache can be set from 8KB to 63KB in size.

Example:

```
HTTPSERV_DIR_CACHE 32
```

That will set the directory cache to 32KB in size.

If you make extensive use of HTACCESS files you should increase this value to the maximum setting. You can run with less but it might slow things down. The cache is fixed in size and not very sophisticated; to keep the programming and memory management reasonable some compromises were made. If the cache overflows it gets erased and rebuilt starting with the data from the current request. If the cache is too small it will keep getting erased, which is almost like having no cache at all.

`HTTPSERV_CLIENTS`

Equivalent command line option: -clients

Format:

```
HTTPSERV_CLIENTS <n>
```

Modern web browsers open multiple connections to a web server at the same time to speed the processing of requests. To the web server each connection is an independent transaction so one clients with multiple connections is treated the same as many clients each with one connection.

HTTPServ allows you to set the number of simultaneously connected clients that it can serve requests to. The default is eight, meaning that eight different clients (or one client being a pig) can connect at the same time, issue a request, and have the request processed. HTTPServ will do its best to service the requests; interleaving the processing. This is better than just simply queuing the requests and processing them one at a time. However, it is not magic - when interleaving the processing everything takes longer.

If more than <n> requests arrive at the same time the additional requests will be queued and made to wait.

HTACCESS Files

(Note: The actual filename used for an HTACCESS file is "HTACCESS.---" . That should be obscure enough to avoid collisions with real filenames.)

In any directory that has content being served by HTTPServ you can create a special file called HTACCESS that can be used to tell HTTPServ to perform special functions. These special functions include marking files and directories as protected using HTTP Basic Authentication, mapping parts of a URL to standard DOS 8.3 filename format, providing a default file to display when the user navigates to a directory (instead of providing a directory list), etc.

An HTACCESS file in a directory only affects the handling of names in that directory. (Names includes names for real files and names for subdirectories in that directory.) If you navigate away from that directory the HTACCESS file is no longer relevant; nothing is inherited from it. (Well, almost nothing ... read the description of the FileAuth directive for details.)

Each line of the file has a directive and some options. All fields on the line are delimited by a single space. Below is a description of the directives.

Map Directive

HTTP URLs are supposed to loosely map to subdirectories and files on the HTTP server. Exceptions to this include parameters in the URL and paths that refer to generated content. As HTTPServ runs on DOS it expects to serve URLs that are composed of legal DOS file and directory names. This becomes a problem when you consider that DOS file and directory names are at most 8 bytes long with an optional 3 byte extension - the dreaded 8.3 format. DOS file and directory names are also not case sensitive.

Map lets you get around these limits by allowing you to define long, case sensitive names that are aliases for DOS 8.3 names. This is useful if you want to host an existing web site where the existing HTTP URLs do not conform to DOS 8.3 naming conventions. Of course the files themselves have to conform to DOS 8.3 naming conventions but you do not have to go into the files and change all of the links to match or update all of the existing URLs.

The Map directive has the following format:

```
Map NiceLongName filename.ext
```

where:

- NiceLongName is a case sensitive name up to 51 bytes in length. If this name contains spaces please wrap it in quotes.
- filename.ext is the DOS 8.3 filename that should be used in place of NiceLongName. This name is not case sensitive.

For example:

```
Map Downloads.html dls.htm
```

That example says that if the URL from the user has "Downloads.html" as the filename in it that HTTPServ should convert that to "dls.htm" instead.

When HTTPServ is processing a URL it "walks" through each part of the URL and considers it on a component by component basis. The mapping if long names to short names is done at each stage of the walk. Consider the following URL:

```
http://yourserver.com/Reference/Downloads/Downloads.html
```

In the path part of the URL we find three parts- "Reference", "Downloads", and "Downloads.html". Reference and Downloads refer to subdirectories and Downloads.html refers to a file. But none of these are valid DOS 8.3 filenames that HTTPServ can read.

To handle this URL we can do the following. Assume that your server is configured with "E:\HTTPDOCS" as your HTTP document root and that E:\HTTPDOCS\REF\DLS\DLS.HTM is the full path that should be served for that request. "E:\HTTPDOCS" is the root of your server and it has an HTACCESS file that includes a Map directive:

```
Map Reference REF
```

That allows HTTPServ to move into the real REF subdirectory when it sees "Reference" at the first level of the URL. Inside of the REF subdirectory there is another HTACCESS file that includes another Map directive:

```
Map Downloads DLS
```

That allows HTTPServ to move into the DLS subdirectory when it sees "Downloads" at the next level of the URL. Inside of the DLS subdirectory we need one more HTACCESS file with one more Map directive:

```
Map Downloads.html DLS.HTM
```

That allows HTTPServ to substitute DLS.HTM as the file to be sent to the web browser.

Mapping only happens once per URL component. You can not have one mapping map to another long name, which in turn maps to something else; if that were allowed a user mistake could lead to an endless loop.

FileAuth Directive

The FileAuth directive lets you mark a file or directory as being protected by an HTTP Basic Authentication realm. (See "Protecting Files and Directories" above for an overview.) The format of the FileAuth directive is:

```
FileAuth filename.ext realm
```

where:

- filename.ext is the DOS file or directory that you are protecting in DOS 8.3 format. Only DOS 8.3 format file and directory names may be protected; if you are using long names in your URLs you need to use MAP directives to convert them to DOS 8.3 format to be able to serve them.
- realm is a string that describes what you are protecting. Most web browsers show this string to the user when prompting for a userid and password. The realm must be defined in a "realms" file that your mTCP configuration tells HTTPServ to use. If the realm is not found in that file nobody will be able to authenticate and see the file or directory.

Example:

```
FileAuth secret.txt "Sensitive Files Area"
```

The example shows that "secret.txt" is marked as belonging to realm "Sensitive Files Area" . Only users listed in the realms file who know the password will be able to see secret.txt.

GeneratedPage Directive

As of this writing HTTPServ has two special pages and one special function that can be accessed using a URL. By default these pages/functions are not available. You can use the GeneratedPage directive to make them available as a page in the directory where the HTACCESS file is located.

The format is:

```
GeneratedPage filename.ext special_function
```

where

- filename.exe is a DOS 8.3 format filename that will be used as the name of the page/function. Yes, it is a little strange to require a DOS 8.3 format name for something that does not actually exist but it makes the code simpler.
- special_function names the special page or function. The current pages/functions are:

serverstatus	generate a page of statistics for the server
clients	generate a page that shows the connected clients
serverquit	end the server immediately

Example:

(Assume these directives are in E:\HTTPDOCS\SPECIAL\HTACCESS.---)

```
Map Status status
Map Clients clients
Map QuitQuitQuit quit
GeneratedPage status serverstatus
GeneratedPage clients clients
GeneratedPage quit serverquit
FileAuth clients "Slighty sensitive"
FileAuth quit "Admin operations"
```

In this example all three special pages/functions are active. The server status page can be obtained by going to http://yourserver.com/Status or http://yourserver.com/status because we have made the page available as "status" and providing a mapping for "Status". The clients page is similar - it has a DOS 8.3 name and a mapping for "Clients". Clients is also marked as being in the "Slightly sensitive" realm so it requires a userID and password to view. The quit function is also available through two names and it is extremely dangerous so it is marked as belonging to the "Admin operations" real. Presumably your realms setup file has less users authorized to "Admin operations" than to "Slightly sensitive".

DirectoryIndex Directive

If a user uses a URL that turns out to be a directory the server may either generate a list of files in the directory, it may refuse access to the list of files in the directory, or it may serve a default file that exists in the directory. This directive lets you specify that a file should be served and which file that is. The format is:

```
DirectoryIndex filename.ext
```

If the user uses a URL that ends in the directory where this is set then whatever file specified on this directory will be served to them. This is the mechanism that lets files such as "index.html" or "default.html" be served when you first navigate to a web site and do not specify a filename in the URL.

If you use this directive the user will not be able to generate a directory list for the directory, even if that is explicitly allowed.

IndexAllowed Directive

The IndexAllowed directive allows you to specify if HTTPServ should generate a file list or not if a user specifies a URL that resolves to a directory. The format is

```
  IndexAllowed yes|no
```

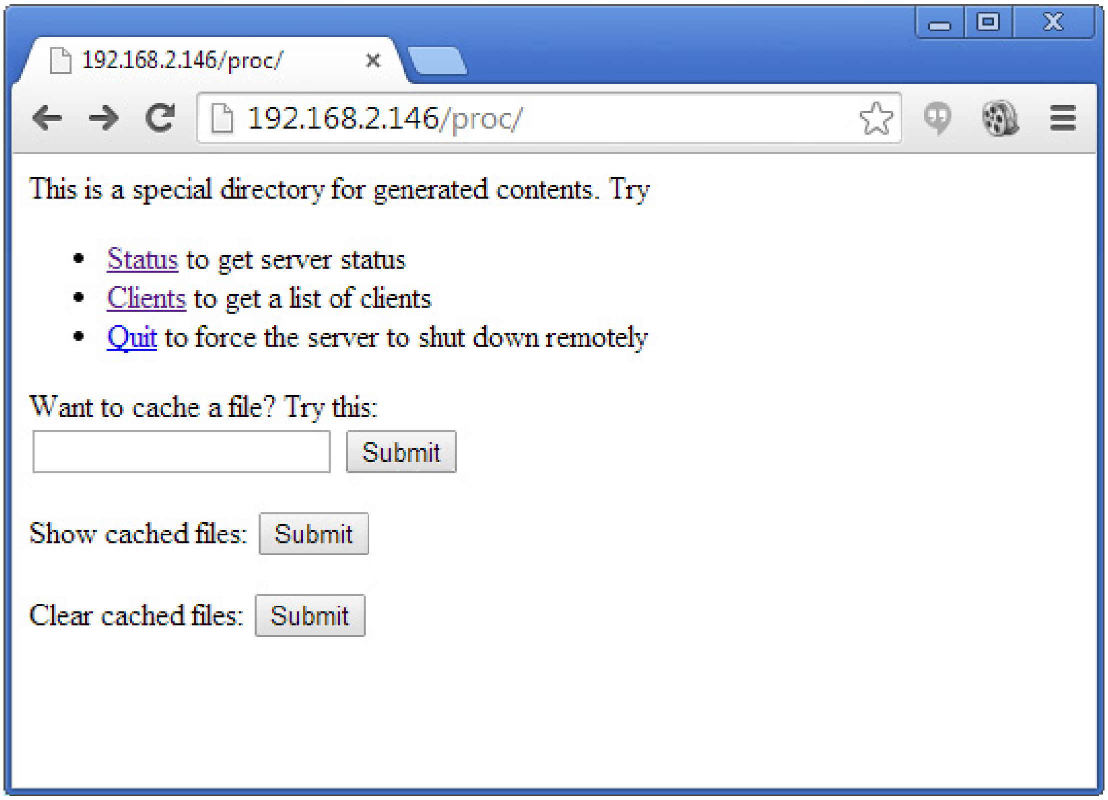
Yes and no are used to explicitly allow or deny a directory list to be generated for a given directory. If no directive is provided then the default is "no" unless that is overridden in the mTCP configuration file using the HTTPSERV_DIR_INDEXES configuration option.

Sample files for exposing the special functions

Let's assume the following:

- Special functions live in a directory called "proc" that is a subdirectory directly under your HTTPDOCS directory.
- All functions are enabled.
- The clients page, cache and quit functions are protected by HTTP Basic Authentication.
- When a user goes to that directory a default file will be served to them to make things easier.

When a user navigates to the proc directory we want them to see something like this:



This is a special directory for generated contents. Try

- Status to get server status
- Clients to get a list of clients
- Quit to force the server to shut down remotely

Want to cache a file? Try this:

Show cached files:

Clear cached files:

Here is a simple HTML file we will call default.htm that will be served to the user when they go to /proc on your server:

```
<html>
  <head></head>
  <body>
    This is a special directory for generated contents.  Try
    <ul>
      <li><a href="Status">Status</a> to get server status
      <li><a href="Clients">Clients</a> to get a list of clients
      <li><a href="Quit">Quit</a> to force the server to shut down remotely
    </ul>
    Want to cache a file?  Try this:
    <form action="cache" method="get">
      <INPUT type="text" name="add" />
      <INPUT type="submit" />
    </form>
    <p>
    <form action="cache" method="get">
      <INPUT type="hidden" name="show" value="">
      Show cached files: <INPUT type="submit"/>
    </form>
    <p>
    <form action="cache" method="get">
      <INPUT type="hidden" name="clear" value="">
      Clear cached files: <INPUT type="submit"/>
    </form>
  </body>
</html>
```

And here is the HTACCESS file for the /proc directory. It does the following:

- Tells HTTPServ to serve default.htm by default when no filename in the directory is specified.
- Defines the four "pseudo filenames" that represent the four server functions.
- Declares three of the functions to be protected by the "Server Functions" realm for HTTP Basic Authentication.

```
directoryIndex default.htm
indexAllowed   no
GeneratedPage  status  serverstatus
GeneratedPage  quit    serverquit
GeneratedPage  clients clients
GeneratedPage  cache   cachecmds
FileAuth       quit         "Server Functions"
FileAuth       clients      "Server Functions"
FileAuth       cache        "Server Functions"
```

Remember, DOS filenames are not case sensitive. The HTML file is using "Status" for a reference, which refers to "status" on the GeneratedPage directive in the HTACCESS file. And there really is no file in /proc called "status" - it is a pseudo filename that results in a status page being generated.

IRCjr

Introduction

This is a fairly simple IRC client that I have been working on since the summer of 2008. Originally it started as a bare bones program that let you connect to one channel on one server. All messages came out on the same screen and sending private messages to other users was a hassle. But, it worked!

Things have changed quite a bit since then. The program has grown in size since those early versions but there are a lot of improvements that make the program much easier to use. IRCjr now supports multiple open channels and private conversations, Client to Client Protocol (CTCP) messages, etc.

This version adds the following features:

- mIRC color code support
- PASS command support
- logging to a different file for each open session
- larger supported backscroll buffer sizes
- 132 column support (if your video card supports it)
- improved TCP/IP lost packet and retransmit support
- many small bug fixes

Enjoy!

Special hardware or software requirements:

None

Setup

IRC networks require you to specify a nickname that other people will know you by, a user name which is the userID that you use on your local machine, and a real name. The nickname is the most important of these, as your nickname will be used the most. The user name and real name can not be verified or trusted and most people probably don't provide accurate information for them anyway.

The nickname, user name, and real name are set by adding lines to the mTCP configuration file. The parameter names are:

IRCJR_NICK	Your nickname for IRC
IRCJR_USER	Your username on this machine
IRCJR_NAME	Your name in real life

Here is an example that shows how to use these:

```
# IRCjr parameters for nickname, user name, and real name
#
ircjr_nick Zoidberg
ircjr_user Jzoidberg
ircjr_name John Zoidberg
```

The official IRC protocol specification states that nicknames are a maximum of 9 characters long. That is routinely ignored and you will find that your favorite IRC servers support much longer nicknames. IRCjr allows you to use up to 40 characters for a nickname but if your server does not support nicknames that long it may truncate the nickname or give you an error message.

Some IRC servers are picky about the user name and real name and want you to use something plausible or they will reject your connection. For example, if you pick a single word for your real name it is possible that the server will assume that you are a "bot" and reject your connection. If you are having trouble connecting to a server check the error messages carefully and see if it is complaining about these fields.

Starting IRCjr

If your packet driver is loaded, the mTCP environment variable is set, and you have run DHCP to get a network address then you are ready to run IRCjr. (If you need help with any of those, please see SETUP.TXT.)

The command line arguments are:

```
ircjr [-port <n>] <server_name> [<channel>]
```

The server name is always required. The channel is optional - you can use the /join command to join a channel once you are connected to the IRC server.

Be sure that the packet driver is loaded first. When the program first starts it is going to initialize the TCP/IP stack and try to find the IP address of your IRC server. It is going to use the DNS system to do this. If you have not setup your TCP/IP environment variables correctly you will not be able to connect to your IRC server. (If you know the numerical IP address of your server you can use it directly to avoid the DNS lookup.)

If all goes well you will get a connection to the IRC server. If you specified a channel on the command line you will be signed onto that channel automatically by the program. At any time during the connection process you can hit the [ESC] key to quit.

The optional -port <n> can be used to connect to IRC servers running on non-standard ports.

Here are some examples:

```
ircjr irc.slashnet.org #vc
```

connects to my favorite channel on my favorite IRC network.

```
ircjr -port 2000 irc.privateserver.net
```

shows how to use the -port option to connect to a server with a non-standard IRC port.

Using IRCjr

Basic knowledge of IRC is assumed. But just in case ...

IRC stands for "Internet Relay Chat" and it is a system that allows clients to connect to a common server and chat with each other. Servers allow for the creation and management of different "channels" that clients can join. The channels are usually named after a topic of interest. Users may also chat directly with each other out of the view of others.

Servers can be connected to other servers, allowing for IRC networks that span the globe. Each network has its own policies regarding user behavior, and even within a channel there might be "operators" who enforce policies on that channel. Operators have the ability to remove people from channels, so be polite to them. ;-0

Screen layout

The IRCjr screen is split into two areas - the user input area and the dialog area. The dialog area shows the running conversation in a channel or a private chat session. The user input area is where you can compose your comments before sending them to the other users or where you can enter IRC commands.

A status line separates the two areas of the screen. The status line tells you which session you are viewing, which sessions are active, and also has indicators for the various toggle settings.

During normal usage the text that you enter will be sent to the other users as part of the conversation. The text that you enter will be interleaved in the conversation with text from other users in the order that the server receives it.

Sending IRC commands

It is possible to send IRC commands to the IRC server. IRC commands meant for the server start with a '/' character at the start of the line. IRC commands are used for many different reasons; here are some of the common ones:

/join	Join an IRC channel (eg: /join #vc)
/msg	Send a private msg to another user (eg: /msg Leela Hello!)
/query	Same as /msg, but open a new session to do it
/names	See who is in the channel (eg: /names #vc)
/part	Leave a channel (eg: /part #boring)
/nick	Change your nickname (eg: /nick newnickname)
/list	List the channels on a server
/quit	Disconnect from the server and end IRCjr

A hint on the /list command - on a large server or server network the list of open channels can be hundreds or thousands of lines long. You can't see all of the output on the screen or in your backscroll buffer, but you can use the LOG toggle to save the list to a file and read it later.

Some less common but still fun commands are:

/info	Get server information
/motd	Read the message of the day on the server
/whois	Get information about a particular nickname
/away	Mark yourself as away (or back)

IRCjr processes the commands that it understands and passes unfamiliar ones straight to the server. This lets you try to use any IRC command even if IRCjr does not recognize it.

Sending CTCP commands

Besides commands for the server there are "Client To Client Protocol" (CTCP) commands you can use to interact directly with other IRC clients. They are:

/me	Send an "Action" command
/ctcp version	Find out what client a user is running
/ctcp ping	Ping a client to see if they are active
/ctcp time	Find the local time for another user
/ctcp userinfo	Get the user name for a nickname

The /me CTCP command is pretty simple to use:

`/me ducks and runs!`	*Sends a message that you are ducking and running*

The other CTCP commands require you to provide a target (a nickname or even a channel name). For example:

`/ctcp version twinkie`	*(find what IRC client user "twinkie" has)*
`/ctcp time otherguy`	*(find out what time otherguy thinks it is)*

IRCjr Hotkeys

Besides the IRC commands there are special key combinations that make IRCjr perform tricks:

ALT-B	Toggle the new message beeper
ALT-H	Display the help window
ALT-C	Close current session/window
ALT-L	Toggle session logging on and off
ALT-S	Show TCP/IP statistics
ALT-T	Toggle timestamps on incoming messages
ALT-X	Exit IRCjr
PgUp	Go back in time through the backscroll buffer
PgDn	Go forward in time through the backscroll buffer
Alt 0-9	Switch to sessions 0 thru 9 (if applicable). (Session 0 is always the "Server" session. Channels and private chats use sessions 1-9.)

Session/Window handling

Each open channel and private chat has its own virtual session. The main window only displays one session at a time, so to flip between the virtual sessions you use Alt and a number key.

When you first start you are in a session reserved for server messages. This window is known as the "Server" session. You can get to it at any time by hitting Alt-0. It can not be closed.

When you join a new channel a new virtual session is created and you are automatically switched to that new session. The same thing happens if somebody sends you a private message or if you send somebody a private message.

The name of the current session is shown on the status indicator line. Also on the status indicator line there is a sequence of digits that show you how many sessions are open and what state they are in:

- Normal digit: a session with no recent activity
- Bright digit: the session currently being displayed
- Reverse digit: a session that has new activity in it

You can flip directly to any session by hitting Alt and the number that represents the session.

You can have up to 10 sessions including the server session. After that trying to create a new session will fail and whatever messages that are sent to that channel will be put on the Server session. If a new session is required and it can't be created because you have run out of memory those messages will go to the Server session too. You should probably close a session to make room for the new session; sending messages will be difficult from the server session.

Sessions can be closed by hitting Alt-C while in the session. If it is a session for a channel an IRC /PART command will be sent automatically.

Logging sessions

Previous versions of IRCjr logged all messages to a single file and you could not control logging on a per session basis. Those flaws have been fixed in this version.

If you turn logging on in a session a new file with the name of the session and the extension "irc" will be created. If the file already exists it will be appended to so that you do not lose your previous log. If the session name is longer than eight characters the filename will only use the first eight characters - this is a limitation of DOS.

By default the current directory is where the log file will be created or opened. You can specify a directory to use for logs by adding a configuration parameter to the mTCP configuration file. (How to do that is described later on.)

When you turn logging off the file is closed. You may resume logging again at any time by using Alt-L. An indicator on the status line tells you if logging is on or off for the session that is currently open.

IRCjr does not try to detect when you connect to channels with the same name on different servers and turn logging on. The same log file name will be used in that case. IRCjr does write the server name and full channel name each time it starts logging so this should not be a large problem.

Using attributes (bold, reverse, etc) and mIRC color codes

IRC is generally a plain text service with not too many extras. You can highlight your messages in a few different ways by using a few special keys:

Bold:

> You can select parts of your messages to appear in bold by turning bold on where you want it to start and turning bold off when you are done. To turn bold on press Ctrl-B - a special "block" character will appear to signal that you have inserted a special attribute command. After turning bold on, type what you want to appear in bold and then press Ctrl-B to turn bolding off. (The Ctrl-B behaves like an on/off toggle.)
>
> Your IRCjr text display really is not setup to display a true boldface font. IRCjr simulates bolding by making the text brighter. (True bolding would show a thicker font.)

Reverse video:

> You can select parts of your messages to appear in reverse video by turning it on and off in the same way that bold is used. To turn reverse video on press Ctrl-R and to turn it off after you are done using it press Ctrl-R again. (Ctrl-R works like an on/off toggle.)

Italics:

> You can select parts of your messages to appear in italics by turning italics on and off in the same way that bold is used. To turn italics on press Ctrl-I and to turn it off after you are done press Ctrl-I again. (Ctrl-I works like an on/off toggle.)
>
> IRCjr can send the italics code, but it can not display italics. When somebody sends italics to IRCjr it inserts the string "`<italics on>`" where the italics attribute starts and "`<italics off>`" when it stops.

Underline:

> You can select parts of your messages to appear underlined by turning underlining on and off around the parts of the text that you want underlined. To turn underlining on press Ctrl-U and to turn it off when you are done with it press Ctrl-U again. (Ctrl-U works like an on/off toggle.)
>
> IRCjr can send the underline code but unless you are running on an IBM Monochrome Display or a Hercules adapter it can not show underlined text. (This is a hardware limitation - CGA, EGA and VGA cards can not show underlined characters in text mode.) When something sends the underline attribute to IRCjr it inserts the string "<underline on>" where the attribute starts and "<underline off>" when the attribute stops.

mIRC color codes

> If you really want to get fancy you can insert color codes into your text.

There are 16 different colors you can use and you can specify both the foreground and background color. To specify a color code start by pressing Ctrl-K. When you do a pop-up color chart with the color code numbers will show up. To specify the foreground color enter the number of the color on the chart. If you want to specify a background color then add a comma and the number of the background color. When you are done, just start typing your message and the pop-up color chart will disappear. The background color is optional and does not have to be used.

To turn color off just enter a Ctrl-K by itself.

For example, the following sequence of keys will give you the word "Hello" written in red on a yellow background:

```
<Ctrl-K>4,8Hello<Ctrl-K>
```

While this sequence will just give you the word "Red" with the default background color:

```
<Ctrl-K>4Red<Ctrl-K>
```

Some people fine color codes annoying, so don't go overboard.

Reset (turn all off)

To disable any attributes that might have been set use Ctrl-O. The special "box" character will be shown where the Ctrl-O was inserted. After that point the text you send should be normal with no bold, reverse, italics, underlining or color.

Selecting your display mode and size

IRCjr uses whatever the current display mode is on your active screen. So if you want to use a particular number of rows and columns, setup the display mode first using your favorite utility and then start IRCjr.

CGA and MDA users do not have any choices - only 80x25 is available.

EGA users can use 80x25 or 80x43 text mode.

VGA users can use 80x25, 80x43 or 80x50 text mode.

SVGA users may be able to use a 132 column screen if their hardware supports it. (DOSBox with SVGA_S3 emulation has a 132x43 mode which is beautiful; use mode 54 to get it. Other cards will vary.)

If you have a monochrome display and a color display on the same machine choose the display you want to use using the DOS MODE command first. IRCjr will detect the current active display and use it.

Optional Configuration Parameters

You only need the three configuration parameters explained earlier to start using IRCjr. But there are other configuration parameters you can use to alter IRCjr's behavior, customizing it to your needs. Here is the list of optional configuration parameters you can use:

IRCJR_PASS	Specify a connection password
IRCJR_CONNECT_TIMEOUT	Set the timeout period for the socket connect
IRCJR_REGISTER_TIMEOUT	Set the timeout for the registration process
IRCJR_BACKSCROLL	Set # of backscroll lines for each channel
IRCJR_BACKSCROLL_CHAT	Set # of backscroll lines for each chat session
IRCJR_BACKSCROLL_SERVER	Set # of backscroll lines for the server session
IRCJR_COLOR_SCHEME	Override the default color scheme
IRCJR_TIMESTAMPS	Turn timestamps on at program start
IRCJR_LOGGING_DEFAULT	Turn on logging at program start
IRCJR_LOG_DIR	Specify a directory for log files
IRCJR_NICK_UPDATES	Specify where nickname updates go to
IRCJR_QUIT_UPDATES	Specify where quit notifications go to

And now for the details on how to use these ...

IRCJR_PASS

IRCJR_PASS allows you to send a "connection password" during the connection and registration process. Connection passwords can be used to authenticate registered nicknames on many servers. Keep in mind that connection passwords are not very secure; they are transmitted in the clear.

IRCJR_CONNECT_TIMEOUT

IRCJR_CONNECT_TIMEOUT controls how long IRCjr will wait for a TCP/IP socket connection to an IRC server. The default is 30 seconds, which should be long enough for almost anything. But if your connection is poor and you need more time you can set it to something longer.

IRCJR_REGISTER_TIMEOUT

IRCJR_REGISTER_TIMEOUT controls how long IRCjr will wait for a server to recognize the client after the TCP/IP socket is created. The default is 30 seconds which is usually long enough, but on a busy server you might need to allow for more time.

IRCJR_BACKSCROLL, IRCJR_BACKSCROLL_CHAT, IRCJR_BACKSCROLL_SERVER

The IRCJR_BACKSCROLL_* settings are used to tell IRCjr how much memory to reserve for backscroll buffers. The backscroll buffer allows you to see lines that have scrolled off of the current screen. The backscroll buffers are a great feature but they require more memory.

The default backscroll buffer settings are:

Server session:	50 lines
Channels:	150 lines
Private chats:	75 lines

These are reasonable for most users. With these settings:

- Just connecting to a server requires ~164KB
- Connecting and joining a channel requires ~191KB
- Connecting, joining a channel and having one private chat requires 206KB

This is a bit more than in the original IRCjr but still manageable. On an 512KB system you can be on nine different channels with a generous amount of backscroll in every channel.

You can set the backscroll buffers to be larger, but you are limited by how much RAM you have available. When IRCjr runs out of memory you will not be able to open new sessions to chat in new channels or receive messages from private users. It will put private messages in the server window, which is ugly but will work. You will probably be happier if you close some idle sessions to get some memory back, or reduce the number of backscroll lines by using these settings. (If you are really tight on memory you can disable a class of backscroll buffers by setting them to 0.)

`IRCJR_COLOR_SCHEME`

IRCJR_COLOR_SCHEME only has one setting at the moment - CGA_MONO. This is useful for machines that have CGA cards but use an LCD or monochrome monitor where the different shades of grey, green or amber might be difficult to distinguish. Setting this will give you a high contrast color scheme.

`IRC_TIMESTAMPS`

IRC_TIMESTAMPS is used to turn timestamps on when the program first starts. The default is to start without timestamps turned on. While the program is running you can always turn the timestamps on by using ALT-T (a toggle setting), but if you usually do that then you can use this setting to make that the default.

`IRCJR_LOGGING_DEFAULT`

IRCJR_LOGGING_DEFAULT allows you to specify the default for the logging state. Normally the program does not log the contents of sessions unless you turn logging on using Alt-L. If you see this "on" the logging will be started by default as soon as the program starts.

`IRCJR_LOG_DIR`

IRCJR_LOG_DIR allows you to specify a directory path where log files will be written to. By default log files are written to whatever the current directory is when IRCjr starts. If you want to specify a directory enter it here in standard DOS format with a drive letter and backslashes as the path delimiter. (A full or relative path can be specified. The path must end in a backslash.)

Log files are always appended to and never deleted or overwritten.

`IRCJR_NICK_UPDATES`

IRCJR_NICK_UPDATES allows you to tell IRCjr where to send nickname updates generated by other users. Four options are available:

none	No nickname updates will be shown anywhere

server	Nickname updates will be shown on the server session
current	Nickname updates will be shown on the current open session
all	Nickname updates will be shown on all sessions

Previous versions of IRCjr used the equivalent of "all" which could get annoying on busy channels. The current default is "none".

`IRCJR_QUIT_UPDATES`

IRCJR_QUIT_UPDATES allows you to tell IRCjr where to send the notifications that are generated when other users disconnect from the server you are using. The same four options as used by IRC_NICK_UDPATES are available.

Previous versions of IRCjr used the equivalent of "all" which could get annoying on busy channels. The current default is "none".

Here is an example of an mTCP configuration file with these parameters set:

```
DHCPVER DHCP Client version Apr 26 2009
TIMESTAMP Sun Apr 26 17:59:54 2009

# Parameters for my machine; your machine will be different
#
packetint 0x60
mt 576
hostname DOSBox

# IRCjr parms
ircjr_nick Zoidberg
ircjr_user Jzoidberg
ircjr_name John F Zoidberg

# All parameters after this are optional

# IRC connection password
ircjr_pass secretgoeshere

# Use these for really slow servers
ircjr_connect_timeout 45
ircjr_register_timeout 60

# Setup for large backscroll buffers

ircjr_backscroll 375
ircjr_backscroll_chat 150
ircjr_backscroll_server 100

# Use this if you need a high contrast screen
ircjr_color_scheme cga_mono

# Turn timestamps on at the start
ircjr_timestamps on

# Turn on logging by default and log to a specific directory
ircjr_logging_default on
ircjr_log_dir e:\data\irclogs\

# Send nickname updates and quit notifications to the server session
ircjr_nick_updates server
ircjr_quit_updates server

# DHCP generated settings will appear here ...
```

Obviously, substitute in values that make sense for your machine. "Zoidberg" is a great name but it is probably already taken.

Recent changes

2013-03-30 Add options for where to send nick updates and quit messages.
2013-03-01 Add 132 col support, mIRC color codes, PASS command, improved logging, additional configuration options, more CTCP commands, misc bug fixes
2012-04-29 Fixes for compatibility with more servers
2011-09-15 Add ability to edit input before sending it

2011-05-20	Client To Client Protocol support; Parse additional server messages; Minor cosmetic and bug fixes
2011-04-06	Multiple session/virtual window support; Add high contrast color scheme setting; Small performance enhancements.
2010-11-27	Misc TCP library changes to improve handling when buffers are full; Watcom runtime workaround: improve compatibility with older machines

Netcat

Introduction

Netcat (nc) is a utility that can send and receive data using a TCP/IP socket. It has the ability to open a connection to another machine or to listen for incoming connections. Input can be entered interactively through the keyboard or redirected in from a file. Output can appear on the screen or redirected to a file. Nc can be used to send large files or short messages, and it can be called from within batch files too.

Any status messages that netcat creates that are not part of the data received from the socket are sent to stderr. This ensures that if you redirect the socket output to a file that only the data that came across the socket will be in that file.

Special hardware or software requirements:

None

Using Netcat

Netcat uses the following syntax:

```
nc -target <ipaddr> <port> [options]
nc -listen <port> [options]
```

The first form of the command is used to create a socket connection to another machine, as a client program normally would. The second form of the command is used to make nc wait for an incoming connection from another machine, as a server program normally would.

Interested in getting started quickly? Skip down to the section entitled 'Examples'. The discussion of options is fairly long.

Netcat Options

Options are:

`-help`	Show basic help text
`-verbose`	Print extra status messages
`-bin`	Treat stdin and stdout as binary streams
`-telnet_nl`	Send and receive newline (NL) as Telnet newline (CR/LF)
`-echo`	Turn on local echoing
`-w <n>`	Wait <n> seconds for traffic after stdin closes
`-nocorc`	Do not Close on remote close

`-srcport <port>`	Use <port> as the source port when connecting

Usage:

First, some background definitions:

Newline character (NL):

This varies depending on the computer and operating system being used.

- For MS-DOS: A newline is composed of a CR/LF pair
- For Unix: A newline character is an LF
- For internet programs using the Telnet NVT standard: A newline is composed of a CR/LF pair.

Socket 'closes'

In this context a socket is a connection to another machine using TCP/IP. The socket is a full duplex connection - each side can send at the same time, and each side is treated as a peer.

When a socket connection is "closed" it means that one one side of the connection has signaled that it is done sending data. The socket still exists, and the side that did the close is still free to receive data from the socket - it just can't send any more data. The other side of the connection is free to send more data to the half that did the close, but it should not expect to be able to read any more data from the closed half.

It is only when both sides have closed their half of the socket that the socket connection is terminated and cleaned up.

Most programs communicating via a socket use a protocol over the socket that warns both sides when the connection is going to be closed. That allows them to close their half of the socket and end the application while being assured that the other side is going to close and exit too. Netcat's default behavior is to close and exit when the other side closes, but this can be overridden.

Netcat default operation:

With no extra options netcat will behave as follows after a connection is made:

- Input will come from the keyboard and be sent over the socket
- Data received on the socket will be written to the screen
- Pressing Ctrl-Z at the keyboard will specify the end of input and close your half of the socket connection. It will also terminate the socket connection and end netcat even if the other side of the connection had not closed their half of the socket. (It is expected that the other side will close when they see you close.)
- Pressing the Enter key will generate a Unix newline char (LF)
- Unix newline characters received on the socket will be converted to MS-DOS newline characters (CR/LF)
- If the remote side does a close on their half of the socket netcat will close your half of the socket and exit. (This can be overridden.)
- Each keystroke is sent in a separate packet. There is no line editing capability because that would block

the receipt of incoming data from the socket.

There are no surprises here, unless you come from an entirely DOS universe where newline characters are always a CR/LF pair instead of the Unix style newline which is an ASCII LF.

Redirecting stdin and stdout from/to files

You can specify that stdin should be read from a file or that stdout should be written to a file. Just use the normal DOS redirection characters to do this. Netcat behavior when stdin/stdout are redirected is similar to its default behavior, except for these changes:

- If netcat detects that stdin is redirected from a file it will try to read large chunks of the file in a single read operation and then send those chunks in large packets. This is more efficient than the normal behavior when netcat is reading from the keyboard.
- If netcat detects that stdin is redirected from a file it will close its half of the socket when EOF is hit on stdin. This will cause netcat to terminate the socket connection and exit whether the other side had closed their half of the socket or not. (There is an option to change this behavior - see "-w".)

Netcat options (details)

`-verbose`

Verbose mode gives you extra status message that tell you when the local side or the remote side closed their connection. These messages are written to stderr so they will not affect data that you redirect to a file on stdout.

`-bin`

MS-DOS can treat files as either BINARY or TEXT.

In BINARY mode:

- All characters are read and written exactly with no changes.
- The End of File (EOF) is determined by the file size, not a Ctrl-Z char. If you are reading from the keyboard (and not a redirected file) then you need to use Alt-X to close the connection, as Ctrl-Z will not be interpreted as EOF.

In TEXT mode:

- When writing on stdout Unix newline characters (LF) will be converted to MS-DOS newline characters (CR/LF).
- When reading from stdin MS-DOS newline characters (CR/LF) will be converted to the Unix newline character (LF).
- A Ctrl-Z will be interpreted as End of File

If you are transferring binary data, use -bin to ensure that netcat and MS-DOS do not alter your data in any way. If you are working with normal ASCII text then use the default, although binary mode might work too.

Binary mode is going to perform much better than the normal text mode because in binary mode no attempt is made to process the incoming and outgoing characters, saving on processing time.

Note: You can use -bin even if you are not redirecting stdin and stdout to a file. The behavior change is the same whether you are redirecting or not.

`-telnet_nl`

The standard for sending newline characters across the internet using the Telnet "network virtual terminal" (NVT) is to send a CR/LF pair. This is because some machines use CR, some use LF, and some use other oddball characters.

If you are connecting to a service that uses the Telnet NVT standard then turn this option on. This will convert your outgoing newline characters to the Telnet standard of CR/LF. Services like HTTP, FTP, SMTP, etc. all use the Telnet NVT standard so they will probably work better with this option. Incoming CR/LF pairs will be translated to normal newlines.

This option can not be mixed with -bin. If you need binary mode, then turning on -telnet_nl makes not sense.

(Slightly fun trick .. MS-DOS uses the same newline characters as the Telnet NVT standard. So you can probably use -bin in place of -telnet_nl. But -bin changes other things, so it is probably best not to mess with it.)

This option does not turn netcat into a Telnet client - it only changes the way that newline characters are sent and received. It will not do Telnet option negotiation, which is an important part of the Telnet protocol that a real Telnet server will need. But this option will help you connect to FTP, SMTP and other servers that use the Telnet NVT standard but not do Telnet option negotiation.

`-echo`

By default no echoing of characters is provided. If you turn this on any characters from stdin will be echoed to stdout.

If the other side is echoing your characters and you turn this on you will see things twice.

Keep in mind that because we are echoing to stdout with this option, your local input will show up in stdout even when you redirect stdout to a file.

`-w <n>`

The default is to close the socket connection and exit from netcat when EOF is detected on stdin. This can be a problem if there was data in flight from the other side at the time the socket was closed. (You will miss the data that was in flight. This is more likely to happen when stdin is redirected from a file, as netcat will send it very quickly and hit EOF possibly before the other side has had a chance to do everything.)

To get around this the -w option tells netcat how long to wait for data to come from the other side after your side of the socket hits EOF on stdin. This can give netcat a chance to receive the last of the data from the other side before closing the connection. <n> is the number of seconds to wait.

`-nocorc`

Corc stands for "Close On Remote Close", so -nocorc means don't do that.

By default if netcat detects that the other side closed its half of the socket connection it will close the socket and exit. This will happen even if EOF was not yet detected on stdin.

With this option netcat will ignore the close from the other side until EOF is detected on stdin.

This option probably is not very useful for most people, but if you are experimenting with 'half closed' sockets it can be very useful.

`-srcport <n>`

By default if you are connecting to another machine netcat will use a random port number as the source port on your machine. If for some reason you need to specify a source port for your machine you can use this option.

Keyboard Handling

By default netcat reads interactively from the keyboard. This is normally straightforward, but there are some things you should know.

In binary mode:

- Ctrl-Z is sent as ASCII 26 and does not mark EOF. (You must signal 'EOF' using Alt-X.)
- Enter is sent as ASCII 13 (CR).
- Ctrl-Enter is sent as ASCII 10 (LF).
- All other keys are sent exactly as they are read.

In text mode:

- Ctrl-Z means EOF for stdin, and will close your half of the socket.
- Enter is sent as a Unix newline (LF)
- Ctrl-Enter is also sent as ASCII 10 (LF). (No change from above)
- To send a raw CR use Ctrl-M.

In text mode with -telnet_nl turned on the Enter key changes slightly:

- Enter is sent as a telnet NVT newline (CR/LF)
- To send a raw CR use Ctrl-M. To send a raw LF use Ctrl-J Or Ctrl-Enter.

Special keys:

Whether you are reading from the keyboard or not the following keys will always be honored:

Alt-E	Toggle local echoing on and off
Alt-H	Help

Alt-X	Close your half of the socket connection
Alt-S	Show Status
Ctrl-Break	Unconditional exit

Alt-S is really handy, as it will show you if the other side has closed their half of the connection or not. If you are doing a large file transfer it can also show you how many bytes have been sent and received.

Environment variables

Two environment variables are provided to help tailor netcat to your memory and performance requirements.

`READBUF`

Default 8192, Range 512 to 32768

Use: Sets the size of the buffer used to read from stdin when stdin is redirected from a file on your local disk. Bigger numbers generally result in higher performance. Use a smaller number if you are tight on memory and need to cut back on what netcat whats to use.

If you are low on memory 4KB is a reasonable setting that will not hurt performance too much.

This environment variable has no effect when you are reading from the keyboard.

`WRITEBUF`

Default 8192, Range 512 to 32768

Use: Sets the size of the buffer used to write to stdout when stdout is redirected to a file on your local disk. Bigger numbers generally result in higher performance. Use a smaller number if you are tight on memory and need to cut back on what netcat whats to use.

If you are low on memory 4KB is a reasonable setting that will not hurt performance too much.

For both variables DOS likes it best if you use a power of 2, or at least a multiple of 512 bytes. Other numbers will cause extra work for DOS and limit performance.

Examples (and Quickstart!)

```
nc -target 192.168.2.10 7
```

Opens a socket connection to 192.168.2.10 on port 7. Your keystrokes will be sent and data coming back will be displayed on the screen. (On a Unix machine port 7 is the echo service, so you should see everything that you send.)

```
nc -target 192.168.2.5 1234 < input.txt
```

Opens a TCP/IP socket to 192.168.2.5 port 1234 and sends the file 'input.txt' to the other machine instead of sending your keyboard input. The program automatically ends after input.txt has been transferred.

```
nc -target 192.168.2.5 1234 -w 5 < input.txt
```

Same as above, but wait 5 more seconds after EOF is detected on stdin so that any response from the other side can be read.

```
nc -listen 2000 > newfile.txt
```

Listens on port 2000 for a connection from a remote machine. When the connection is established any incoming data is sent to newfile.txt. When the remote machine closes the connection netcat exits automatically.

```
nc -listen 3000 -nocorc > newfile.txt
```

Similar to above, but netcat does not automatically exit until it is told to. (This is not terribly useful when redirecting files, but is useful when using netcat interactively.)

```
nc -target www.cnn.com 80 -telnet_nl -verbose -echo
```

Connect to the HTTP server at www.cnn.com on port 80. Automatically convert newlines to telnet NVT newlines, enable verbose status messages, and start with local echoing turned on.

(If you send "GET /" without the quotes and press Enter, you will get a web page back!)

Network printing

If you have a network attached printer that supports “raw” printing on TCP/IP port 9100 then it is possible to send files from your DOS machine to the printer for printing using the following steps:

- Send your output to a DOS file instead of a printer. The output file needs to be in a format the printer understands, such as PCL for printers that use Printer Control Language or PostScript for printers that use that. (This implies that your DOS program has a suitable printer driver for your printer.)
- Send the output file to the printer using the netcat command:

```
nc -target <printer address> 9100 -bin < <filename>
```

Substitute <printer address> with the IP address of your printer and <filename> with the filename your program generated when you said "print to file." For example, on my printer:

```
nc -target 192.168.2.20 9100 -bin < testfile.txt
```

will print testfile.txt over the network to the printer. (testfile.txt gets redirected on stdin to netcat.)

If you are unsure if your printer listens on port 9100 then just try the steps above anyway; at worst case nothing will happen.

Limitations

This version of Netcat does not support UDP. (Most Unix versions do.)

Recent changes

2010-11-27 Misc TCP library changes to improve handling when buffers are full. Watcom runtime workaround: improve compatibility with older machines

2010-06-21 Ported to Open Watcom

2010-04-22 Maintenance release for the underlying TCP library. TCP checksums should be computed correctly all of the time now, and are much faster.

2010-02-08 Complete rewrite

Ping

Introduction

This is a DOS version of the 'Ping' utility commonly found on systems that support TCP/IP. Ping allows you to do a basic check to see if a remote system is alive and responding to TCP/IP even if it is not accepting connections on specific ports. Ping is also useful for measuring how far away (also known as the ping latency) of a remote system - lower ping times mean better performance.

Special hardware or software requirements:

None

Using Ping

Usage is like this:

```
ping [options] <your.machinename.here>
```

Options are:

`-help`	Show a help message
`-count`	Tell ping how many ping packets to send
`-size`	Set the size of the test data in each packet

By default ping will send four packets and each packet will have a 32 byte test message in them. The total packet size will be 60 bytes, including the TCP/IP headers.

You can tell ping to send more packets or to send bigger packets. The test data portion of a packet can be up to 256 bytes in size.

Ping sends one packet per second.

Here is some sample output:

```
mTCP Ping by M Brutman (mbbrutman@gmail.com) (C)opyright 2009-2015
Version: May  2 2015

ICMP Packet payload is 256 bytes.

Packet sequence number 0 received in 151.30 ms, ttl=45
Packet sequence number 1 received in 155.55 ms, ttl=45
Packet sequence number 2 received in 154.70 ms, ttl=45
Packet sequence number 3 received in 149.60 ms, ttl=45
Packet sequence number 4 received in 148.75 ms, ttl=45

Packets sent: 5, Replies received: 5, Replies lost: 0
Average time for a reply: 151.98 ms (not counting lost packets)
```

You can press CTRL-C or CTRL-BREAK to stop. If you ping a non-existent machine you will either get a DNS error message or an ARP failure.

Fun notes

Normal timer resolution under DOS is limited to 55 milliseconds. This is how fast the BIOS timer clicks, and it works out to about 18 times a second.

Ping can measure time down to 0.85 milliseconds. To do this it reprograms the hardware timer to generate interrupts faster, and then only passes the correct number of interrupts to the BIOS code. This way BIOS keeps correct track of the time and everything else works as it should, while Ping can get 1 millisecond accuracy while it needs it.

This technique is highly accurate - try it!

Recent changes

2012-04-29	Fix packet length problem
2010-06-21	Ported to Open Watcom
2009-09-30	Higher resolution timing
2009-09-25	Initial version

PktTool

Introduction

Pkttool is a small utility that can:

- scan memory for loaded packet drivers
- read statistics from a loaded packet driver
- capture and display packets flowing on your network (packet sniffer)

The first two functions are useful when trying to diagnose problems with an Ethernet card on your system. The packet sniffer function is an advanced function that can help you diagnose problems on your network.

Special hardware or software requirements:

None

Using PktTool

No setup is required. Unlike most of the mTCP programs, pkttool does not use the mTCP configuration file. All you need is to have your packet driver loaded.

Pkttool has three primary commands, each of which may have options. The commands are:

scan	scan memory for loaded packet drivers
stats	read statistics from a loaded packet driver
listen	capture and display packets flowing on your network

Scanning for packet drivers

To scan for loaded packet drivers use the scan command:

```
pkttool scan

mTCP pkttool by M Brutman (mbbrutman@gmail.com) (C)opyright 2008-2015
Version: Apr 12 2015

Scanning:

Details for driver at software interrupt: 0x60
  Name: NE2000
  Entry point: 0192:03CE
  Version: 11   Class: 1   Type: 54  Interface Number: 0
  Function flag: 2  (basic and extended functions)
  Current receive mode: packets for this MAC and broadcast packets
  MAC address: AC:DE:48:88:99:AA
```

If any packet drivers are detected information about them will be displayed.

If the Trumpet DOS TCP/IP TSR is found in memory you will be warned about it. mTCP is not compatible with Trumpet but both can be used if you have each one assigned to a different packet driver.

After the pkttool program ends the DOS ERRORLEVEL variable will be set to the number of packet drivers found, or 0 if none were found.

Displaying packet driver statistics

To display the statistics tracked by each packet driver use the stats command:

```
pkttool stats <x>
```

where <x> is the software interrupt that the packet driver was loaded at. It is a hexadecimal value ranging from 0x60 to 0x80.

And here is some sample output:

```
pkttool stats 0x60

mTCP pkttool by M Brutman (mbbrutman@gmail.com) (C)opyright 2008-2015
Version: Apr 12 2015

Details for driver at software interrupt: 0x60
  Name: NE2000
  Entry point: 0192:03CE
  Version: 11   Class: 1   Type: 54  Interface Number: 0
  Function flag: 2  (basic and extended functions)
  Current receive mode: packets for this MAC and broadcast packets
  MAC address: AC:DE:48:88:99:AA

All statistics are from when the packet driver was loaded.

Packets in:           10  Packets received
Packets out:           4  Packets sent
Bytes in:           2700  Bytes received (includes Ethernet headers)
Bytes out:          1210  Bytes sent (includes Ethernet headers)
Errors in:             0  Receive errors
Errors out:            0  Sending errors
Packets lost:        135  Lost due to no handler, out of buffers, etc.
```

Listening for packets on the network:

The most powerful feature of pkttool is the ability to listen for packets being received and to give you a quick idea of what they are.

Under normal circumstances an Ethernet card is setup to listen only for packets that are addressed directly to it or packets that are sent for broadcast to the entire network. A special mode known as "promiscuous mode" can be used to let the packet driver receive all packets on the wire, regardless of the destination. Pkttool uses promiscuous mode to let you see those packets, making it a valuable debugging tool.

To run pkttool in listen mode use the following syntax:

```
pkttool listen <x> [-v] [-raw] [-s <n>] [-outputfile <filename>]
```

where:

<x> is the software interrupt that the packet driver was loaded at. It is a hexadecimal value ranging from 0x60 to 0x80.

-v means you would like to see some basic output that shows each incoming packet. (Think of it as "verbose mode".)

-raw means that you want to see the raw hexadecimal data for each incoming packet.

-s <n> says to capture only n bytes when using -raw mode. So if you have large packets and you only care about the first 128 bytes, this option will let you only capture the first 128 bytes of those larger packets.

-outputfile <filename> instructs pkttool to write its output to the specified file.

When started in listen mode, pkttool will switch your Ethernet card into promiscuous mode and start listening for traffic. Some of the traffic on your network that you might see includes:

- ARP requests
- IPv4 traffic
- IPv6 traffic
- Network switch management traffic

Not all possible traffic will be visible; what you will be able to see will depend on your network switches and routers.

Using the -v option:

If you use the -v option you will see output like this for each incoming packet:

```
16:45:34.06 To: 30:85:A9:68:E6:60 From: 00:21:9B:29:79:BE Len: 80 T:0800
IPv4: From: 192.168.2.100 To: 216.58.216.132 Len: 66 Type: (17) UDP

16:45:34.11 To: 00:21:9B:29:79:BE From: 30:85:A9:68:E6:60 Len: 131 T:0800
IPv4: From: 216.58.216.132 To: 192.168.2.100 Len: 117 Type: (17) UDP

16:45:34.72 To: 33:33:00:00:00:0C From: 00:21:9B:29:79:BE Len: 208 T:86DD
IPv6: From: FE80:0000:0000:0000:FD79:90D5:2FD7:832E
        To: FD79:90D5:2FD7:832E:FF02:0000:0000:0000
       Len: 154 Type: (17) UDP

16:45:35.21 To: FF:FF:FF:FF:FF:FF From: 00:21:9B:29:79:BE Len: 60 T:0806
ARP: Request: 192.168.2.100 is looking for 192.168.2.129
```

For each packet you will see a timestamp, the destination and source MAC address, and the Ethernet packet type. (If you see an Ethernet packet type that is less than 0x0600 that is an older style Ethernet 802.3 frame as opposed to a newer style Ethernet II style frames.)

If the packet type is ARP, IPv4, or IPv6 you will get some additional information from the packet.

Using the -raw option:

If you use the -raw option you will be able to see the data in the packets. That data might contain user IDs and passwords that are not encrypted. Be careful and use this responsibly! You do not want to leak a password because you shared a cool packet trace ...

Each packet will be displayed as a raw hexadecimal dump:

```
0000  30 85 A9 68 E6 60 00 21 9B 29 79 BE 08 00 45 00   0..h.`.!.)y...E.
0010  00 42 5D 06 00 00 80 11 69 D9 C0 A8 02 64 D8 3A   .B].....i....d.:
0020  D8 84 DE FA 01 BB 00 2E 38 D6 0C F0 0C 29 5A 1F   ........8....)Z.
0030  DE 33 EF C2 7A AB 07 81 46 03 21 43 40 D5 69 C4   .3..z...F.!C@.i.
0040  9C 72 D8 5E 08 E3 C2 9E A6 E4 09 48 B8 3C F4 41   .r.^.......H.<.A

0000  FF FF FF FF FF FF 00 21 9B 29 79 BE 08 06 00 01   .......!.)y.....
0010  08 00 06 04 00 01 00 21 9B 29 79 BE C0 A8 02 64   .......!.)y....d
0020  00 00 00 00 00 00 C0 A8 02 81 0C F0 0C 29 5A 1F   .............)Z.
0030  DE 33 EF C2 7A AB 07 81 46 03 21 43               .3..z...F.!C
```

In this example the first packet is an IPv4 packet. The next packet is an ARP packet.

Dumping the hexadecimal for each packet like this takes time and your computer might not be able to keep up. If you see warnings about packets being dropped then consider using the -s option to not dump all of the contents of each packet, or just don't use the -raw option at all. If you are writing to an output file you can improve your speed by writing to a file that is on a RAM disk.

Pkttool will continue to listen for traffic until until you press [ESC] to end the program. At the end of the program you will get some output that summarizes the traffic that was seen. Here is a sample of that output:

```
Tracing active for 2 minutes, 8 seconds

Packets received:            1018
Packets dropped:                0
Ethernet broadcast packets:    73
Unique MACs detected:           4

MAC addr: A4:70:D6:FD:F1:22  Packets sent: 10  Bytes sent: 910
  Packets:     10  Bytes:     910  Type: 0800 Ipv4

MAC addr: 00:1B:A9:7D:8A:8A  Packets sent: 35  Bytes sent: 6330
  Packets:     34  Bytes:    6270  Type: 0800 Ipv4
  Packets:      1  Bytes:      60  Type: 0806 ARP

MAC addr: 00:21:9B:29:79:BE  Packets sent: 545  Bytes sent: 109741
  Packets:    381  Bytes:   92317  Type: 0800 Ipv4
  Packets:     88  Bytes:   12864  Type: 86DD Ipv6
  Packets:     76  Bytes:    4560  Type: 0806 ARP

MAC addr: 30:85:A9:68:E6:60  Packets sent: 428  Bytes sent: 149225
  Packets:    356  Bytes:  144565  Type: 0800 Ipv4
  Packets:     63  Bytes:    3780  Type: ---- (IEEE 802.3 packet)
  Packets:      5  Bytes:     300  Type: 0806 ARP
  Packets:      4  Bytes:     580  Type: 888E EAP over LAN


All hosts combined: Packets sent: 1018  Bytes sent: 266206
  Packets:    781  Bytes:  244062  Type: 0800 Ipv4
  Packets:     88  Bytes:   12864  Type: 86DD Ipv6
  Packets:     82  Bytes:    4920  Type: 0806 ARP
  Packets:     63  Bytes:    3780  Type: ---- (IEEE 802.3 packet)
  Packets:      4  Bytes:     580  Type: 888E EAP over LAN


MAC addresses and their possible IP addresses (based on ARP traffic):

00:1B:A9:7D:8A:8A  ->  192.168.2.20
00:21:9B:29:79:BE  ->  192.168.2.100
30:85:A9:68:E6:60  ->  192.168.2.1
```

Here you can see that packets from four different machines were spotted by pkttool. Besides the expected ARP and IPv4 traffic there was IPv6 traffic, unidentified 802.3 style packets, and four "Extensible Authentication Protocol over LAN" packets.

Pkttool scans ARP traffic to try to build a mapping of MAC addresses to IP addresses. That table appears at the end. It is possible that you will have many more machines on your network than are identified in this table. The table only uses ARP traffic, which is the most reliable way to detect which MAC address has which IP address assigned. If you run pkttool for a long enough time you will have enough ARP traffic to build a fairly complete table.

Recent changes

2015-04-10	New!

SNTP (Simple Network Time Protocol)

Introduction

SNTP is a small utility that lets you fetch the current time and date from an NTP server on the internet. This is useful if your machine does not have a battery backed clock/calendar or if you want to adjust for the drift that is inherent in most older clock/calendar solutions.

Special hardware or software requirements:

None

Using SNTP

Usage is like this:

```
sntp [options] <ntp.server>
```

Options are:

`-help`	Show a help message
`-port <n>`	Contact server on port <n> (default=123)
`-retries <n>`	Number of times to retry if no answer (default=1)
`-set`	Set the system time
`-timeout <n>`	Seconds to wait for a server response (default=3)

The NTP server parameter can be any NTP server on the network. I recommend using 'pool.ntp.org'. Details on this collection of servers can be found at http://www.pool.ntp.org/ .

Before running SNTP you should set the 'TZ' environment variable in DOS. The date and time that you get from the NTP server will be in Universal Coordinated Time and needs to be adjusted for your local timezone. The TZ environment variable tells the code what your local timezone is and how to make that adjustment.

Here are some example timezone settings:

```
set TZ=EST5EDT    (US Eastern Standard, 5 hour offset, DST is observed)
set TZ=CST6CDT    (US Central Standard, 6 hour offset, DST is observed)
set TZ=UTC-1      (One hour east of GMT, somewhere in Europe)
set TZ=CET-2CDT   (Two hours east of GMT, DST is observed)
```

Some of you might note that the time offset is opposite of what Windows uses. That is because the program is using the POSIX standard for setting the TZ environment variable. Sorry, that is just the way it is. If you are west of GMT the offset is positive and if you are east of GMT the offset is negative.

If you want to see the full rules for setting the TZ environment variable including how to set it for complex daylight savings rules, please see the Open Watcom documentation:

http://www.openwatcom.org/ftp/manuals/current/clib.pdf

Look for the section entitled "The TZ Environment Variable."

SNTP uses the UDP protocol to contact the NTP server. UDP is not a reliable way to send packets on the network, so packets are often lost. The -retries option can be used to improve the chances that SNTP will be able to get a response from an NTP server.

By default SNTP only tells you the time that it received from the NTP server. If you want the time to be set on your system use the -set option.

Examples:

```
set TZ=CST6CDT                    Set timezone to US Central Standard

sntp pool.ntp.org                 Get the time from pool.ntp.org
sntp -retries 3 pool.ntp.org      Same as above, but retry up to three times
sntp -set pool.ntp.org            Set the system time to the NTP time
```

Miscellaneous notes

SNTP is accurate to within about one second. NTP servers will return time stamps with much higher resolution but after you deal with network jitter and delays it is not as accurate. Combine that with DOS which has a default timer resolution of 55ms and there is just no point in trying to make it more accurate. One second is great for most of us on older hardware.

Recent changes

2010-08-21	Initial version

Spdtest

Introduction

This program can be used to test the speed of your machine when sending and receiving data on TCP sockets. It has no practical use other than for benchmarking so don't invest too much time with it.

The basic idea is to send or receive data using the TCP library without doing any extra processing on the data. The data will not be copied, written to disk, or manipulated in any way. It will be fully processed by the TCP library though, just like normal data - no shortcuts like not computing the checksums are taken. This allows you to determine the maximum speed you should expect when sending or receiving data. Actual applications will be slower because they presumably do work.

I use this program to test the speed of new versions of the mTCP code. I also use it to figure out the relative speed of different machines and Ethernet cards.

Special hardware or software requirements:

None

Using Spdtest

Spdtest uses the following syntax:

```
spdtest -target <ipaddr> <port> [options]
spdtest -listen <port> [options]
```

The first form of the command tells spdtest to create a socket connection to another machine. The second form of the command is used to make spdtest wait for an incoming connection from another machine.

Options are:

`-help`	Show basic help text
`-srcport <port>`	Use <port> as the source port when connecting
`-receive`	Do a receive only test
`-send`	Do a send only test
`-mb <n>`	Megabytes to send during a send test

Spdtest requires another machine to either send it data or to blast data at. To ensure a good benchmark the other machine should be far faster than the machine you are testing. For example, I use a 200Mhz Linux machine to test my slower PCs against.

The other machine should be running netcat. If spdtest is doing a send test then the other machine should be set to receive and throw away the data as quickly as possible. If spdtest is doing a receive test then the other machine should be sending a fixed amount of data, hopefully as fast as it can.

Tips for getting accurate results:

- Both machines should be on the same network
- The network should be relatively quiet
- The machines should be plugged into a switch, not a hub
- The test should run at least 10 seconds, and possibly up to a minute.
- The longer you run the more accurate the results will be.
- Timing should be done with a stopwatch and on the other machine. Some Ethernet card and packet driver combinations hold off interrupts too long, distorting time on the machine running spdtest.
- Check the ending messages for network errors. The test should run cleanly; network errors are a sign of other problems.

Telnet

Introduction

This is a small telnet client like other telnet clients that you have probably used before. Besides communicating with a telnet server using the telnet protocol it also emulates a standard 'ANSI' terminal and has uploading and downloading capability using Xmodem and Ymodem Batch. This particular telnet client features excellent screen updating performance and a small memory footprint.

Special hardware or software requirements:

None

Using Telnet

Telnet uses the following syntax:

```
telnet [options] <telnet_server_addr> [port]
```

where <telnet_server_addr> is the name or numerical IP address of the telnet server you wish to connect to and [port] is an optional port to connect to use. By default the port is set to 23, which is the standard telnet server port.

Options are:

`-help`	Show basic help text
`-debug_ansi`	Create telnet.log with some extra debug info
`-debug_telnet`	Create telnet.log with some extra debug info
`-sessiontype <telnet|raw>`	Force telnet mode or raw mode

Under normal operation if you connect to port 23 on a server you will be operating in telnet mode. This means that the telnet client will expect to receive telnet options from the server, and will reply and try to negotiate option settings. If you connect to any port besides port 23 you will be operating in raw mode, where the telnet client will not respond to telnet options and it will not try to negotiate option settings.

If you need to connect to a real telnet server on a non-standard port then use the -sessiontype option to force the telnet into telnet mode, even though you are not connecting to the standard telnet port. You probably will never need to specify raw mode but it is there if you need it.

For example:

```
telnet bbs.retroarchive.org
```

connects to a machine called 'bbs.retroarchive.org' using the standard telnet port (23). It expects to find a telnet server at that port and it will negotiate telnet options with that server.

```
telnet example.com 5500
```

connects to a machine called example.com on port 5500. This is not the standard telnet port so no telnet options negotiation will be done. If that machine really were running a telnet server at that port you could force telnet mode using:

```
telnet -sessiontype telnet example.com 5500
```

To make the screen performance tolerable on older machines telnet will take incoming data and render the current screen on a virtual buffer before trying to repaint the real screen. This approach uses a little bit more memory but it dramatically improves the performance of the screen handling, especially when scrolling large amounts of data. You will notice that the screen will pause and stop updating while the machine is getting flooded with incoming data - this is normal and it is keeping you from dying a slow and agonizing death while the display adapter scrolls. A nice side-effect of the virtual buffer approach is that you get backscroll capability for free - if something does scroll past the screen you can hit Page Up and Page Down to browse around.

The following special keys are recognized while telnet is running:

PgUp or PgDn	Go up and down through the backscroll buffer
Alt-H	Show a combined help and status screen
Alt-B	Toggle sending DEL chars when the Backspace key is used on and off
Alt-E	Toggle local echoing on and off
Alt-N	Toggle between sending [Enter] as CR/NUL, CR/LF, CR or LF
Alt-W	Toggle automatic wrapping around the right margin on and off
Alt-R	Refresh the screen from our local virtual buffer. (Should not be needed unless I have a screen drawing problem)
Alt-D	Download a file using Xmodem or Ymodem Batch
Alt-U	Upload a file using Xmodem or Ymodem Batch
Alt-F	Drop to DOS to run some commands (make it quick!)
Alt-X	Exit the program (Ctrl-Break does this too) Telnet will wait for the server to close the connection so that you don't lose any last bytes that were being sent. If this is taking too long hit Alt-X again and mTCP Telnet will exit immediately.

When you use one of the toggle options a single beep means it was turned off, while a beep followed quickly by a higher pitched beep means it was turned on.

The following special keys are sent to the server to handle:

- Cursor keys (Up, Down, Left Right)
- Home
- Insert
- Back-Tab (shift Tab)

Telnet protocol features and limitations

This isn't a fully compliant telnet client implementation. It has the following limitations:

- SGA (Server Go Ahead) must be enabled on both sides of the connection. Most servers expect this behavior, as a TCP/IP socket is a full duplex connection.
- This client will not do local line editing.
- This client does not support the telnet Data Mark command. This is not a big issue, but it means that you don't have an easy way to tell the server to squelch a flood of output if it happens. At Ethernet speeds this should not be a big problem.

It does properly support the following telnet options:

- Binary transfer (used during file transfers)
- Echoing - it will request or let the server offer to do echoing. You can also turn on local echoing.
- Terminal type - it will send the terminal type if asked for it
- Window size - it will send the window size in rows and columns if asked

Other options such as passing the environment variables are not supported, but that is not a big deal considering that you are connecting from a DOS machine. (If you have a burning desire for a missing option let me know.)

Uploading and Downloading files

In ye old days one would use a terminal emulation program with a modem to gain access to a BBS or multi-user timesharing system. If you wanted to download a file you used something like Xmodem, Kermit, or Zmodem. Downloading was slow and agonizing, which made the reward for getting a good file that much more precious.

Now people use HTTP and FTP to transfer files over broadband. It just works. It's not interesting. Most people don't even know that FTP or HTTP exists - they are just "getting a file."

In order to give you a taste of what it was like to transfer files back in the old days I have finally added Xmodem and Ymodem to telnet! Many telnet BBSes support uploading and downloading through telnet connections and Unix systems have supported it using the sx/sb and rx/rb commands for years. It is not as fast as FTP but you can do it right from telnet without quitting to run another program. On faster machines the speed difference is not noticeable.

Protocols:

Here are your options for uploading and downloading:

> **Xmodem:** This is the classic protocol designed by Ward Christensen in 1977. It sends your file in chunks of 128 bytes; if your file size is not a multiple of 128 bytes it will be padded to the next 128 byte multiple. Xmodem does not preserve file sizes or timestamps. You may only transfer one file at a time. The data is protected with a one byte checksum which is probably completely inadequate for unprotected data links but is not necessary over telnet as TCP/IP does its own error detection and correction.
>
> **Xmodem CRC:** This is a slight improvement on Xmodem. Instead of protecting your data with a 1 byte

checksum it uses a 2 byte CRC. It can be used with Xmodem packets that are 128 bytes in length (the standard) or 1 KB in length (an extension).

Xmodem 1K: This is major performance improvement on standard Xmodem. By sending 1KB at a time instead of 128 bytes throughput is dramatically improved. The speed of Xmodem is limited by how quickly the receiving side can tell the sending side "go ahead, I got that last packet" or "that last packet looked bogus, please resend." Xmodem 1K reduces that overhead by a factor of eight, resulting in much less "dead air" on the wire. Xmodem 1K is almost always used with the CRC option.

Ymodem Batch: Ymodem is an improvement over Xmodem in several ways. It sends the filename, file size and timestamp to the remote side in a special header packet so that you don't have to re-enter the name or worry about the file size and timestamp being altered. Ymodem is a batch protocol so it can be setup to send multiple files at a time. Ymodem uses the CRC option; normally 1KB packets are used but 128 byte packets might be seen near the end of a file transfer.

To download a file (or files) give the server the command to start sending the file(s) and then press Alt-D. You will need to select the protocol to use - be sure this matches what the other side is going to use. If you have a mismatch the file transfer will probably fail. If you choose one of the Xmodem variants you will have to enter the filename to save the file to and the saved file size will be rounded up to the next 128 byte size. The file timestamp will also be set to the current time, as Xmodem does not preserve the file timestamp. If you download using Ymodem Batch just select the protocol and the transfer will start. The filename is automatically received, the file size will be exact, and the original file timestamp from the server size will be preserved.

(Almost - If you run using DOSBox the file timestamp is not preserved because of a limitation in DOSBox.)

If you try to download a file that already exists on your system you will be prompted to see if you really want to write over the existing file. If the file can not be over written for some reason then the transfer is aborted. This can happen if the filename happens to be the same as a directory on your local machine.

To upload a file give the server the command to start receiving the file and then press Alt-U. Select a protocol to use, and enter the name of the file to send.

When uploading and downloading use the ESC key to abort the file transfer. Xmodem and Ymodem transfers are normally canceled by sending a special character (Ctrl-X) at least two times. mTCP Telnet will do this for you, but if you need to cancel a file transfer manually just hit Ctrl-X a few times and be patient. (You might need to do this if you started the file transfer on the other side and changed your mind before starting it on the mTCP Telnet side.)

All file operations happen in the current directory. See the section on "Shelling to DOS" for details on how to change directories and manage files.

Limitations on file transfers:

When receiving using Ymodem you can tell the other side to send more than one file at a time. But when sending you can only setup mTCP Telnet to send just one file at a time.

Filenames must be in 8.3 format. The sending and receiving of paths is not supported.

There is no Ymodem G. Ymodem G support on other systems is often implemented wrong, or does not exist at all. It is just not worth the hassle.

Notes for Linux/Unix:

If you have lrzsz package installed then you have the following commands available:

- rx - receive on the server side using Xmodem
- rb - receive on the server side using Ymodem Batch
- sx - send from the server side using Xmodem
- sb - send from the server side using Ymodem Batch

All of these commands have a bad habit - they write to stderr which in turn appears on stdout. And that can mess up your transfer. To get reliable transfers from these commands pipe the output of stderr to a file or /dev/nul. For example:

```
sx dosfile.exe 2> stderr.txt
```

That will send the file dosfile.exe from Linux to the mTCP Telnet client, piping the output on stderr to a file called stderr.txt. (This assumes you are using bash for your shell. If you use a different shell figure out how to pipe stderr using that shell.)

The send commands (sx and sb) might not use 1KB packet sizes or CRC by default. Not using CRC is not an issue over telnet, but not using 1KB packets can slow you down quite a bit. Use the "-k" option to setup for 1KB packets. (And using CRC never hurts either.)

Another neat trick is to enable a raw TTY session before you start your file transfer. This will prevent the terminal driver from interpreting characters and injecting or removing extra characters from the data stream while your transfer is in progress. To enable raw mode use "stty raw", and to return things to normal use "stty sane".

This sample sequence illustrates the safest way to transfer a file using Unix:

```
stty raw; sx myfile 2> stderr.txt; stty sane
```

That sequence puts the TTY in raw mode and starts a file transfer using Xmodem while piping the extraneous output to a file called stderr.txt. After the transfer the terminal will be put back into sane mode.

```
stty raw; sb -k myfile 2> stderr.txt; stty sane
```

This sequence is similar, except that Ymodem Batch with 1KB blocks is used. This should be much faster than the standard block transfer size, which is 128 bytes.

Notes for Synchronet:

Synchronet BBS is great, but it has a bad habit. When using Ymodem batch it will append extra bytes to your file to insert an advertisement for Synchronet. This kind of defeats the purpose of preserving the original file size. While I appreciate the effort that goes into maintaining Synchronet, this is bad behavior.

Shelling to DOS

Sometimes you may need to suspend telnet and execute DOS commands while still connected to a server. You might need to do this manage files that you have just transferred. The Alt-F (File) key in telnet allows you to do this.

When you use this function telnet will be suspended and you will be dropped at a DOS prompt. You can execute commands here; I would limit it to just a few seconds and only using simple commands. For example, if you need to delete a partially downloaded file, change directories, or check a directory for files this function is ideal. Do not do anything that might alter the screen mode; the screen contents will be restored but telnet is not expecting the mode to change.

While you are at the DOS shell telnet is not processing any incoming Ethernet packets. If you receive something from the other system the connection will time out and you will be disconnected! Keeping that in mind, do not use this function if you are expecting input from the other side. When you are sitting at a Unix command prompt it is probably safe; doing it while a directory listing is coming down is definitely not safe.

Use the "exit" command at the DOS prompt to return to telnet.

ANSI Emulation Notes

Full blown ANSI terminal emulation is a complicated mess. All of the different variations of ANSI emulation over the years have not helped anything either. To put it politely, I've done the best I can and you might still see screen rendering problems.

That being said, I've tried to do a reasonable good job of interpreting ANSI escape sequences and rendering them properly. The ANSI emulation does not depend on ANSI.SYS or any other console device driver - it is all internal to the program. I used a few sources to determine the required set of escape sequences, including the Linux TERMCAP entries for a few ANSI-like terminals, including the generic ANSI terminal definition and a few related PC flavors. The emulation is good enough for me to write this documentation using VI, browse in text mode with Lynx, run the 'screen' program to get a few virtual terminals, use the 'info' command to browse the terminfo writeup, and run some of the more complicated 'system-config-*' commands provided with Fedora.

Here are a few notes to improve your experience:

Unicode:

This telnet client doesn't have Unicode support in it, so sending a three byte sequence to it and hoping that you will get a Unicode character will not work. The easiest way to suppress the bogus Unicode characters is to set your LANG environment variable so that it doesn't trigger Unicode support. For example, on my recent Linux boxes LANG looks like this:

```
echo $LANG
en_US.UTF-8
```

Set it to en_US instead like this: **`export LANG=en_US`**

Codepages:

The standard codepage built into the monochrome card or the CGA card is codepage 437. No other codepages are possible if you are using these two video cards. Set Lynx to use codepage 437 to avoid getting weird characters.

EGA and VGA users may be able to get other codepages loaded into memory using country.sys and other DOS configuration commands. I have not experimented with this yet, but in theory you should be able to use other codepages with telnet by doing that. (If you try this out please let me know how it works!)

Terminal Types:

By default if the telnet server asks type type of terminal is connecting this code will report back as 'ANSI'. This happens during telnet option negotiation.

If you want to experiment with other terminal types supported by your system then you can change the string that gets reported. See the section entitled 'Advanced Setup' for instructions on how to do this.

If you just want to make a temporary change most Unix systems have a TERM variable in the environment that you can alter.

Depending on your system there may be several suitable terminal types to explore. Older Linux systems have over 20 variants of ANSI terminals to choose from. Newer Linux systems might just have 'ANSI' and 'PCANSI' to choose from. Look in /usr/share/terminfo/ for possible termcap definitions to play with. If you want to make a custom termcap for this program contact me and I'll tell you exactly what I implemented.

Monochrome Adapter Users:

The monochrome adapter does not display color, but it does have the ability to underline characters properly. If you are on a true monochrome display adapter (MDA) then you should set your terminal type to 'pcansi-mono', or something similar that tells the server that you have a terminal with true underlining capability. The standard ANSI setting will work, but will not enable underlining.

Special note: Telnet BBSes and MUDs

Not everything out there claiming to be a telnet server is actually running a compliant telnet server that does telnet option negotiation. A lot of telnet BBBes and multi-user dungeons fall into this category. If you are having trouble trying to connect to something that is not a standard telnet server, try using the "-sessiontype raw" option. You will still get ANSI emulation but all of the telnet protocol negotiation code will be disabled. This is done automatically when you connect to anything other than port 23.

Assuming that works, you might need to turn on local echoing using Alt-E.

[Enter] key handling

The telnet standard spells out that a "newline" character is comprised of a CR (carriage return) followed by a LF (line feed). But it is not clear what it means when a user presses "Enter" at a terminal; that is traditionally just a CR that gets sent to the server. This makes the standard ambiguous.

After two years of struggling with the ambiguity I think I have found something that works for most telnet servers. By default pressing the "Enter" key will send a CR/NUL pair. If you need something different for a non-standard telnet server you can use Alt-N to toggle between the following options: CR/NUL, CR, LF, CR/LF. The help screen will show you the current setting.

Regardless of this setting you can always send a CR by pressing Ctrl-M and send a LF by pressing CTRL-J.

Advanced setup

There are some options that you can specify in the MTCPCFG file to override default behavior, such as the initial state of toggles. Below is the list:

`TELNET_VIRTBUFFER_PAGES <n>`

> Telnet makes screen access fast by writing to a virtual screen buffer in memory, and then updating the real screen as needed. <n> is the number of virtual screen pages to store in memory. Valid values are from 1 to 8.

`TELNET_CONNECT_TIMEOUT <n>`

> <n> is the number of seconds to wait for a connection to the Telnet server. The default is 10 seconds.

`TELNET_AUTOWRAP <n>`

> Set <n> to 0 to specify no autowrap or 1 to turn autowrap on. The default is to turn autowrap on.

`TELNET_SENDBSASDEL <n>`

> Set <n> to 1 to send a DEL character (ASCII 127) when the Backspace key is hit instead of sending a Backspace character (ASCII 8). The default is to send DEL.

`TELNET_TERMTYPE <termtype>`

> Report <termtype> as your terminal type during Telnet option negotiation. The default termtype is 'ANSI'.

`TELNET_SEND_NEWLINE <chars>`

> Choose which characters will be sent when you press your Enter key. The valid values for TELNET_SEND_NEWLINE are "CR/LF", "CR", "LF", "CR/NUL", or "AUTO". The default is AUTO, which uses CR/NUL unless telnet binary mode is enabled. If binary mode is active then just a CR will be sent. (You don't have any direct control over binary mode so don't worry about it.) See the section on [Enter] key handling above for details.)
>
> Examples:

```
    TELNET_SEND_NEWLINE AUTO       Default: use CR/NUL or just CR automatically
    TELNET_SEND_NEWLINE CR/LF      Send both CR and LF at all times
```

```
TELNET_SEND_NEWLINE CR        Send just a CR
TELNET_SEND_NEWLINE LF        Send just an LF
TELNET_SEND_NEWLINE CR/NUL    Send CR/NUL
```

Recent changes

2015-01-23 Add 132 column support; make Telnet wait for the server to close the connection so the last bytes being sent are not lost. Code cleanup changes to reduce size.

2013-03-15 Add scroll region support; Add DEC Origin Mode; add a few more CSI and ESC sequences; consolidate screen updating code.

2012-04-29 Fix [Enter] handling; Add Xmodem and Ymodem file transfer

2011-09-11 Fix telnet options bug

2011-05-18 Add Alt-N to toggle between sending [Enter] as CR/LF, CR or LF

2010-11-27 Misc TCP library changes to improve handling when buffers are full. Watcom runtime workaround: improve compatibility with older machines

2010-06-21 Ported to Open Watcom. Note: The CGA snow code has been removed temporarily.

2010-04-22 Maintenance release for the underlying TCP library. TCP checksums should be computed correctly all of the time now, and are much faster.

2010-03-21 Fix a bug involving servers with high numbered telnet options.

2009-12-27 Add local echoing toggle; Bug fix: [Enter] key should send CR/LF, not just CR; Bug fix: Monochrome displays were freezing; Bug fix: Monochrome displays were not underlining; Add -sessiontype command line option and implement raw mode

2009-12-12 Add support for cursor position reporting

Advanced Topics

Debugging problems

Introduction

Welcome to the wonderful world of DOS networking! If you are reading this you are either bored or you have a problem. Hopefully this will help you get past your problem.

Keep in mind there is a wide variety of hardware out there including emulators on modern machines which are technically not even hardware! This guide will give you some pointers on how to get past problems but it is going to take some work.

This guide is broken up into two parts - [1] debugging hardware and packet driver problems and [2] debugging mTCP problems.

Besides this guide check out the instructions for the PKTTOOL.EXE. PKTTOOL can talk to a loaded packet driver and get statistics from it, allowing you to see if the packet driver is seeing any packets "on the wire." There is also a packet sniffer tool that will let you look at the traffic on your network. (Hint: if you never see any traffic, then there is something wrong!)

Hardware and packet driver problems

Note: Some of this might not apply to PCI cards. Parallel port devices like the Xircom PE3 series require the parallel port to be correctly configured. Serial line users (SLIP or PPP) need to ensure that their COM ports are setup correctly. Do what makes sense for your specific setup.

mTCP will not work if your Ethernet card is not setup correctly. Here is a partial list of things that can trip you up:

I/O ports

Like any PC hardware, your Ethernet card needs to be setup to use I/O ports. The I/O ports can not be shared with other cards, either on purpose or by accident. If there is a conflict with another card you will get flaky operation or maybe even random crashes.

If your packet driver loads and reports the correct MAC address for your Ethernet card you probably have the I/O port set correctly. (You might still have conflicts with other cards though.)

A MAC address is a unique identifier that is 48 bits long usually written in hexadecimal notation with a colon separating each byte. For example, 02:31:BE:D1:2F:08. The MAC address can usually be found printed on a sticker on the card. A MAC address that is all 00s (00:00:00:00:00:00) or all FFs (FF:FF:FF:FF:FF:FF) is probably not correct and indicates that the packet driver is not talking to the card.

If the MAC address reported by the packet driver does not match the MAC address printed on the card then it is possible that it has been changed on the card, but it is more likely that you have the wrong I/O port address.

Hardware Interrupts (IRQ)

Your Ethernet card probably needs a hardware interrupt assigned to it so that it can tell the packet driver when a new packet has arrived on the wire. The IRQ has to be set on the card so that it is not shared with any other hardware in your system, including devices that might be on your motherboard.

PC and XT class systems generally have very few interrupts available. IRQ 0, 1, and 6 are usually always used or unavailable. IRQ 2, 3, 4, 5, and 7 may be available depending on what else you have in your system.

AT class systems have more choices for interrupts. Unlike an XT class system, IRQ 2 is not available on an AT class system.

If the IRQ is not set correctly then your card may be able to send packets but each time it receives a packet it will not be able to tell the packet driver about it. This will look like your card is never receiving packets.

Shared Memory

All Ethernet cards use some memory to store incoming and outgoing packets. Simple Ethernet cards keep that memory on the card in a private area that the rest of the computer can not see. Better cards expose this memory to the computer which improves performance but requires you to find a memory range that does not conflict with other cards or motherboard settings.

The NE1000 card is an example of a card that does not use shared memory. (The NE1000 just needs I/O ports and an IRQ.) The Western Digital 8003 series is an example of card that does required shared memory. It might need to show 8KB or 32KB (depending on the card) to the system, requiring that much space free in your memory map.

Cards that use shared memory will generally need space in the area reserved for expansion cards which ranges from C000:0000 to F000:0000. Be sure to pick a range that does not conflict with other devices.

Cabling selection

Besides the basic resource allocation settings above there may be other settings that you have to make on the card. One good example is setting the cable type - some cards make you set jumpers to choose between the AUI port, Thinnet connector, or RJ45 connector. If this is set wrong the card will appear to work normally but will not be able to send or receive data on the wire.

Bad cabling or network connection

Even with a properly configured card you can still encounter problems.

Believe it or not, cables go bad. If you suspect a cabling problem find another cable that you know works and use that instead. Cables are cheap so there is no reason to wrestle with a potentially bad one.

Most Ethernet cards, hubs and switches have indicator lights that tell you if a good connection has been made. Examine your indicator lights and ensure that your hardware thinks there is a connection. Use a known good port on your switch or hub if you even get a hint of a problem with the port you are using.

Some cable modems are configured to only hand out a limited number of addresses and will refuse to hand out more. You may need to reset the cable modem to clear its list of clients. Switches and hubs are generally reliable but sometimes cycling the power (turning them off then on) clears up problems.

Older Ethernet cards pose special challenges. Before modern Ethernet standardized on CAT 5 cabling there was ThickNet and ThinNet cabling. ThickNet (the original Ethernet) was often supported by using the AUI port on a card with an external transceiver, while ThinNet was often directly connected to the card. The card would have to be configured to use one port or the other port and choosing the wrong port will make it look like you have bad cabling.

Even on cards that look like they support twisted pair wiring (CAT5) the signaling might not be correct, and thus not work on modern switches and hubs. For example older WD8003 series cards have both an AUI port for an external transceiver and an RJ45 jack which accepts a CAT5 cable, but it actually uses Lattisnet which is not compatible with Ethernet signaling. It is close enough to make a hub or switch light up the port, but not close enough to enable data transfer.

Packet driver

First, ensure you have the right packet driver for your card! Packet drivers are very specific to models of Ethernet cards. I always try to look for the packet driver from the manufacturer that was shipped with the card. If I can't find that I might use a packet driver from a card in the same chipset family.

A good source for packet drivers for older cards is the Crynwr packet driver collection at http://www.crynwr.com/ . Those guys defined the packet driver specification and did a lot of the early work to make packet drivers available for ISA cards. Source code for their work is available, which is an added bonus.

If your Ethernet card is setup correctly you should know the I/O port and other settings to pass to the packet driver on the command line. Packet drivers can detect some configuration problems or mismatches but not all of them, so be sure you have the settings that match your card.

The PKTTOOL.EXE program might provide some insight as to how the packet driver is behaving. See the directions for it in PROGRAMS/PKTTOOL.TXT.

Collecting mTCP traces

I would rather have you believe that mTCP is perfect, but we know that nothing ever is. But we can strive ...

In general, the mTCP code is too complex for a casual user to try to debug. If I really do have a problem in the code we are going to need to take a trace so that I can see what is going wrong.

Even if mTCP is acting normally you might still be having problems related to your network or setup. To make debugging and fixing problems easier I have included some tracing features in the library.

To enable tracing you need to set two DOS environment variables before running an mTCP program. The LOGFILE variable sets the name of a text file where trace output will be written to. The DEBUGGING variable sets the desired level of tracing to be used. For example:

```
set DEBUGGING=127
set LOGFILE=logfile.txt
```

Those two lines specific log level 127 (to be explained next) and that logfile.txt should hold the trace.

Tracing can be slow so mTCP lets you choose how verbose you want the tracing to be. The number used on the DEBUGGING environment variable controls this. Here are the useful values for DEBUGGING that you can use:

1	Warnings only
3	Warnings and application messages
127	Full tracing except for packet contents
255	Full tracing with some packet contents

The bigger the number the slower the machine will get. Traces can also get quite large. On slower machines tracing will cause a noticeable overhead because of the extra disk I/O.

The number is actually based on a bitmask, so if you want to be selective you can turn on specific types of trace messages. The bit positions are:

0x0001	Warnings
0x0002	General - used by applications
0x0004	ARP - used by ARP
0x0008	IP - used by the IP/ICMP layer
0x0010	UDP - used by UDP
0x0020	TCP - used by TCP
0x0040	DNS - used by DNS
0x0080	Packet dumping

Now that you know the bit variables you might also find it useful to know that setting the DEBUGGING variable can also be done using a hexadecimal value. For example:

```
set DEBUGGING=0x7F
```

Turns on all tracing except for packet dumping, and is equivalent to the "127" level shown in the previous example.

With the debug trace turned on it is possible that you will have a crash and not see all of the debug messages. This is because the debug messages are buffered and depending on the timing the buffer may not be fully flushed at the time of the crash. If you want to be absolutely sure that all trace messages are flushed and visible there is a bit to flip that will aggressively flush the buffers after each trace point is written:

```
0x8000 Turn on aggressive buffer flushing
```

So for example, if you want full packet tracing with normal trace file buffer behavior use:

```
set DEBUGGING=0xFF
```

And if you are paranoid and you want to make sure that you see every trace message, flip the extra bit and use:

```
set DEBUGGING=0x80FF
```

Debugging problems

If you have a trace that you would like me to read send it along with a brief explanation of what you were doing and what your network setup looks like. I have been able to help a few people get up and running this way and I have fixed countless of my own bugs with it.

PPP Setup

Introduction

While mTCP is designed for Ethernet cards it is possible to use it with PPP (Point-to-Point Protocol) connections. This chapter describes how to setup mTCP and a Linux host to use a PPP connection over the serial port.

SLIP connections are similar but are covered in the next chapter.

Prerequisites

- It is assumed that you are familiar with mTCP configuration and that you have some knowledge of Linux system administration. It is possible to use other operating systems by mapping the networking concepts presented here to those other operating systems.

- A known good connection between the machines. This is usually done with a null modem cable. You need to verify this link works with terminal emulation software before you start with PPP - it makes debugging much easier if you start with a known good setup.

- The DOSPPP packet driver. Download a known working version from here:

  http://www.ibiblio.org/pub/micro/pc-stuff/freedos/files/net/dosppp/dosppp06.zip

  Other PPP drivers are possible; I used DOSPPP as an example. One key point is that whatever PPP driver you use should emulate an Ethernet card for mTCP. mTCP is expecting to use a packet driver that is designed for Ethernet; that is known as a "class 1" packet driver. mTCP does not work with other classes of packet drivers, including serial (class 6) and Token Ring (class 3). DOSPPP gets around this limitation by telling mTCP that it is an Ethernet card even though it is actually using the serial port.

Network setup for the examples provided here

My house is connected to the outside world with a cable modem that gives me a dynamically allocated address on the global internet. That is connected to a home router, which then feeds the machines in the house. The router does network address translation (NAT) to provide private addresses within the house.

```
Cable Modem  <->  Router        <->   Machines
96.x.x.x          192.168.2.1         192.168.2.x
```

Normally I use Ethernet connections on my DOS machines so they look like any other PC in the house and can connect to the router directly.

To add a DOS machine using PPP one of the machines in the house has to convert from Ethernet to PPP. I use an older Linux machine to do this. Here is what the network diagram is going to look like:

```
Cable Modem  <->  Router       <->  Linux
96.x.x.x          192.168.2.1       192.168.2.10
                                        |
                                    192.168.2.17 <-> DOS (192.168.2.18)
```

The Linux machine has an Ethernet card and it is assigned address 192.168.2.10. That is a static assignment; it never changes. The DHCP server in the router is told not to give out any addresses below 192.168.2.100 so there will not be a conflict.

To enable PPP the Linux machine will take on a second network address. That network address must be chosen carefully. Here is the process:

- Pick a number that is divisible by 4.
- No machine in your house should have a machine with that number or the next three numbers after it.
- Your Linux machine will have that number + 1
- Your DOS machine will have that number + 2
- The other two numbers will be reserved and unused.
- Do not start with 0 or 252.

For example, the following addresses would be valid in this scheme:

```
Linux            DOS               Reserved
192.168.2.5      192.168.2.6       192.168.2.4 and 192.168.2.7
192.168.2.17     192.168.2.18      192.168.2.16 and 192.168.2.19
192.168.2.161    192.168.2.162     192.168.2.160 and 192.168.2.163
```

Other options are available, but those require advanced routing tricks. Just find a block of four addresses divisible by four that are not in use, follow the rules above, and you will be fine. (I'll tell you the hard way that adds new subnet for a beer or two.)

Establish the PPP connection to the Linux machine

On the Linux side you will be using the PPP daemon. The important parameters to specify are:

- the serial port that will be used
- the speed of the port
- "local" to set the flow control
- "lock" to ensure that other programs do not interfere with the port
- "passive" or "silent" to tell PPPD to wait for a connection instead of timing out. (Passive and silent are slightly different, but both should work.)
- "proxyarp" to make your machine look as though it is visible on the network.
- A reasonable MTU size. 576 should work fine.
- IP addresses for the Linux side and the remote side

Before you fire up the PPP daemon ensure that nothing else is using that serial port. If you have a "getty" process running on that port to allow normal logins you will need to disable it. The serial port has to be clear for PPP. (Use the ps command to look for a getty process that might be using it.)

For example, on my Linux system I use the following command line:

```
pppd ttyS1 9600 local lock passive proxyarp defaultroute mtu 576
192.168.2.17:192.168.2.18
```

(All of that should be on the same line.)

In this setup I'll be running at 9600 bps over my second serial port. The Linux machine will get a new network interface called ppp0 with address 192.168.2.17 and it will be talking to the DOS PC which will have an address of 192.168.2.18.

If successful the Linux machine will return silently to the command line but the PPP daemon will be running in the background waiting for a new connection from the DOS machine. You can verify this using the ps command.

On the DOS machine we need to run EPPPD to complete the connection. Here we need to specify the serial port and the speed to use. I use this command on my DOS machine:

```
EPPPD com2 9600 local
```

EPPPD is the PPP driver with Ethernet emulation. This is necessary for mTCP because mTCP is written with Ethernet in mind and can not use a serial link directly.

My DOS machine will be using the second serial port at 9600 bps using flow control suitable for directly connected machines.

After a few seconds of delay EPPPD responds with:

```
Installed packet driver handler at vector 0x60
```

Now you have a live link between the two machines. If you go back to the Linux machine and run "ifconfig" you will be able to see the new network interface:

```
ppp0  Link encap:Point-to-Point Protocol
      inet addr:192.168.2.17  P-t-P:192.168.2.18  Mask:255.255.255.255
      UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:576  Metric:1
      RX packets:11 errors:0 dropped:0 overruns:0 frame:0
      TX packets:13 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:3
      RX bytes:170 (170.0 b)  TX bytes:158 (158.0 b)
```

EPPPD will also create a batch file called IP-UP.BAT which sets environment variables suitable for WATTCP programs. mTCP won't use those environment variables but some of their values are useful to us to configure mTCP.

Configuring mTCP for PPP

mTCP configuration is done as normal, but with a few changes:

6. You need to set the MTCPSLIP environment variable to something. The actual value does not matter; it just needs to exist. At the DOS command line this command will do the trick:

```
SET MTCPSLIP=true
```

7. You should ensure that your MTU setting in the mTCP configuration file matches what you told the PPP daemon on Linux. I recommend using a value of 576.

8. You need to setup the IP addresses statically because you will not be using DHCP in this setup. In particular you need to set the IPADDR, NETMASK, and GATEWAY variables:

   IPADDR: This is second IP address you used on the PPP daemon line in Linux. In this example it is 192.168.2.18.

   NETMASK: Linux uses a netmask of 255.255.255.255 to signify a point-to-point link. mTCP is thinking "Ethernet" so it wants a netmask that reflects the addressing scheme more closely. If you followed the rules above a netmask of 255.255.255.252 is correct for mTCP.

   If you did not pick your addresses as per the rules above you might be able to ping the Linux machine, but not other machines. Follow those rules carefully!

   GATEWAY: If you plan to send packets past your Linux machine to other machines the Linux machine will have to serve as your gateway. Put its PPP0 address here. In this example it is 192.168.2.17.

If these are set correctly you should be able to run PING on the DOS machine and get a response back from the Linux machine. At 9600 bps on a 386 class machine the time for a standard 32 byte PING packet should be around 150ms.

> *[The math is pretty straight forward - at 9600 bps you get about one byte per millisecond. The standard payload for ping is 32 bytes and you need an IP header, an ICMP header, and four more bytes for the ping identification number and sequence number. The total bytes sent is that overhead (28 bytes) plus the ping payload bytes (32 default) for a total of 60 bytes. PPP adds at least another 8 bytes of framing overhead to that, so the total being sent per packet is closer to 68 bytes. 68 bytes at 9600 bps takes 70.8ms to send and you need to wait for the response which is the same size. So the minimum possible time with no additional overhead is 141ms.]*

If you ping from Linux to DOS you might not get a response; remember mTCP only response when an mTCP application is running. If you want to try to ping from Linux run something simple on the mTCP machine. Netcat in "listen" mode will do the trick:

```
nc -listen 2000
```

That will enable you to test PING from Linux to DOS.

Now you have basic connectivity between the two machines. Telnet, FTP, IRCjr and other applications will work if the target machine is your Linux machine. Next we need to enable some advanced routing to go beyond your Linux machine to other machines on your network and the outside world.

Routing beyond the Linux machine

The "proxyarp" option on the pppd command should make your machine look like it is on the network. If a remote machine tries to find the hardware address of your DOS machine running mTCP the Linux machine will substitute it's own hardware address. In effect, it is your proxy on the network.

This is not sufficient for getting to machines past your Linux machine though. Your Linux machine still considers the PPP link to be a separate network. All we need to do is to tell Linux to route packets from one network to the other. This is done by turning on "IP forwarding."

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Do that as root. After that, when Linux sees a packet from an outside network it will route it to the DOS machine running mTCP.

After that, you should be able to ping other machines on your home network. If you setup your nameserver in mTCP you should be able to get to the outside world as well. And other machines will be able to ping your machine.

Advanced options

What has been presented above is the bare minimum to get a DOS machine running mTCP on a local home network. Your home network might not look like this, requiring more work. Some other options include:

- Running IP masquerading (NAT) on the Linux machine.
- Running the PPP network on a different subnet and having Linux routing to that subnet. This requires advanced routing on other machines in your network if you are using non-routable networks like 192.168.x.x.

All of this is left as an exercise to the interested reader ...

SLIP setup

Introduction

While mTCP is designed for Ethernet cards it is possible to use it with SLIP (Serial Line Internet Protocol) connections. This chapter describes how to setup mTCP and a Linux host to use a SLIP connection over the serial port.

PPP connections are similar but are covered in the previous chapter.

Prerequisites

- It is assumed that you are familiar with mTCP configuration and that you have some knowledge of Linux system administration. It is possible to use other operating systems by mapping the networking concepts presented here to those other operating systems.

- A known good connection between the machines. This is usually done with a null modem cable. You need to verify this link works with terminal emulation software before you start with SLIP - it makes debugging much easier if you start with a known good setup.

- The "EtherSlip" packet driver. Download a known working version from here:

  http://crynwr.com/drivers/pktd11.zip

  (The filename in the zip is ETHERSL.COM. Instructions are in INSTALL.DOC.)

  Other SLIP drivers are possible; I used EtherSlip as an example. One key point is that whatever SLIP driver you use should emulate an Ethernet card for mTCP. mTCP is expecting to use a packet driver that is designed for Ethernet; that is known as a "class 1" packet driver. mTCP does not work with other classes of packet drivers, including serial (class 6) and Token Ring (class 3). EtherSlip gets around this limitation by telling mTCP that it is an Ethernet card even though it is actually using the serial port.

Network setup for the examples provided here

My house is connected to the outside world with a cable modem that gives me a dynamically allocated address on the global internet. That is connected to a home router, which then feeds the machines in the house. The router does network address translation (NAT) to provide private addresses within the house.

```
Cable Modem  <->  Router       <->   Machines
96.x.x.x          192.168.2.1        192.168.2.x
```

Normally I use Ethernet connections on my DOS machines so they look like any other PC in the house and can connect to the router directly.

To add a DOS machine using SLIP one of the machines in the house has to serve as a gateway between Ethernet and SLIP. I use an older Linux machine to do this. Here is what the network diagram is going to look like:

```
Cable Modem  <->  Router       <->  Linux
96.x.x.x          192.168.2.1       192.168.2.10
                                        |
                                    192.168.2.121 <-> DOS (192.168.2.122)
```

The Linux machine has an Ethernet card and it is assigned address 192.168.2.10. That is a static assignment; it never changes. The DHCP server in the router is told not to give out any addresses below 192.168.2.100 so there will not be a conflict. (In this case my DOS machine with address 192.168.2.122 might cause a conflict, but I don't have that many DHCP clients in my house.)

To enable SLIP the Linux machine will take on a second network address. That network address must be chosen carefully. Here is the process:

- Pick a number that is divisible by 4.
- No machine in your house should have a machine with that number or the next three numbers after it.
- Your Linux machine will have that number + 1
- Your DOS machine will have that number + 2
- The other two numbers will be reserved and unused.
- Do not start with 0 or 252.

For example, the following addresses would be valid in this scheme:

```
Linux           DOS              Reserved
192.168.2.5     192.168.2.6      192.168.2.4 and 192.168.2.7
192.168.2.17    192.168.2.18     192.168.2.16 and 192.168.2.19
192.168.2.161   192.168.2.162    192.168.2.160 and 192.168.2.163
```

Other options are available, but those require advanced routing tricks. Just find a block of four addresses divisible by four that are not in use, follow the rules above, and you will be fine. (I'll tell you the hard way that adds new subnet for a beer or two.)

Establish the SLIP connection to the Linux machine

On the Linux side you will be using the SLIP daemon (slattach). The important parameters to specify are:

- the protocol to use (slip, compressed slip, etc.)
- the serial port to use
- the speed to use

Here is an example:

```
slattach -p slip -s 9600 /dev/ttyS1
```

That says to use plain SLIP (not compressed) at 9600 bps over the second serial port on the machine.

I had a devil of a time trying to get my serial port to work. It looked as though it was receiving packets from the DOS machine just fine, but it was not able to send packets because the handshaking on the serial port wires was not correct. I could see this using ifconfig - packets were being received on the serial line interface but not sent.

So besides the parameters above I used the following:

- "-L" to specify a local ("3 wire") connection
- "-m" to not initialize the line

And before running slattach I had to use stty to setup the speed and handshaking for the port. Here are the full commands that I used:

```
stty -F /dev/ttyS1 -clocal -crtscts 9600
slattach -p slip -L -m -s 9600 /dev/ttyS1
```

The stty command sets up the second serial port to not do flow control or look for a carrier detect signal. The speed will be set to 9600 bps. Your results will probably vary from mine ...

slattach creates a network interface but nothing else. Next we have to tell Linux what IP address to use for that interface. The ifconfig command is used for that:

```
ifconfig sl0 192.168.2.121 pointopoint 192.168.2.122 mtu 576 up
```

Here I am using 192.168.2.121 for the IP address on the Linux side and the DOS side is going to use 192.168.2.122. (The spelling of "pointopoint" is deliberately missing a "t" - that is the correct syntax.) The MTU size for the connection is 576, which is a safe value.

On the DOS machine we need to run our SLIP driver to complete the connection. Using EtherSlip as an example:

```
ethersl 0x60 3 0x2f8 9600
```

That tells EtherSlip (ethersl) to install using software interrupt 0x60 on COM2 (IRQ3 and port 0x2f8) at 9600 bps.

EtherSlip doesn't let you just name a COM port to use - it wants you to specify the hardware interrupt and I/O port address of the COM port you are going to use directly. Here are the common port assignments:

```
COM1: IRQ 4, Port 0x3F8
COM2: IRQ 3, Port 0x2F8
```

Your hardware may vary ... but you tested this port with a straight terminal emulator first, right?

If EtherSlip is successful it will print out some status information, including the "MAC" address of the simulated Ethernet card.

Configuring mTCP

mTCP configuration is done as normal, but with a few changes:

1. You need to set the MTCPSLIP environment variable to something. The actual value does not matter; it just needs to exist. At the DOS command line this command will do the trick:

```
SET MTCPSLIP=true
```

2. You should ensure that your MTU setting in the mTCP configuration file matches what you told the SLIP daemon on Linux. I recommend using a value of 576.
3. You need to setup the IP addresses statically because you will not be using DHCP in this setup. In particular you need to set the IPADDR, NETMASK, and GATEWAY variables:

   IPADDR: This is second IP address you used on the ifconfig command in Linux. In this example it is 192.168.2.121.

   NETMASK: Linux uses a netmask of 255.255.255.255 to signify a point-to-point link. mTCP is thinking "Ethernet" so it wants a netmask that reflects the addressing scheme more closely. If you followed the rules above a netmask of 255.255.255.252 is correct for mTCP.

   If you did not pick your addresses as per the rules above you might be able to ping the Linux machine, but not other machines. Follow those rules carefully!

   GATEWAY: If you plan to send packets past your Linux machine to other machines the Linux machine will have to serve as your gateway. Put its SL0 address here. In this example it is 192.168.2.121.

If these are set correctly you should be able to run PING on the DOS machine and get a response back from the Linux machine. At 9600 bps on a 386 class machine the time for a standard 32 byte PING packet should be around 150ms.

If you ping from Linux to DOS you might not get a response; remember mTCP only response when an mTCP application is running. If you want to try to ping from Linux run something simple on the mTCP machine. Netcat in "listen" mode will do the trick:

```
nc -listen 2000
```

That will enable you to test PING from Linux to DOS.

Now you have basic connectivity between the two machines. Telnet, FTP, IRCjr and other applications will work if the target machine is your Linux machine. Next we need to enable some advanced routing to go beyond your Linux machine to other machines on your network and the outside world.

Routing beyond the Linux machine

So far you have a "point to point" connection between the two machines and the DOS machine can not see past the Linux machine. In this section we are going to do a few steps to give your DOS machine connectivity to the rest of the world.

The first step is to turn on "proxy ARP" which is a trick that tells the Linux machine to answer ARP requests from other machines on behalf of your DOS machine, which is not directly connected to the network. The "arp -s" command is used to set this up:

```
arp -s 192.168.2.122 00:90:27:FC:28:82 pub
```

Here we are telling the Linux machine that if it should see an ARP request for IP address 192.168.2.122 (the IP address of the DOS machine) that it should answer with its own MAC address, which is "00:90:27:FC:28:82". You need to use your own MAC address on your machine - use the "ifconfig" command to find out what it is for your Ethernet card. For example, on my machine:

```
ifconfig eth0
```

responds with:

```
eth0  Link encap:Ethernet  HWaddr 00:90:27:FC:28:82
      inet addr:192.168.2.10  Bcast:192.168.2.255  Mask:255.255.255.0
      inet6 addr: fe80::290:27ff:fefc:2882/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:12914 errors:0 dropped:0 overruns:0 frame:0
      TX packets:15950 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:1386899 (1.3 Mb)  TX bytes:1687851 (1.6 Mb)
```

where HWaddr is the MAC address of this particular ethernet card.

After doing this other machines will be able to ping the DOS machine.

This is not sufficient for getting to machines past your Linux machine though. Your Linux machine still considers the SLIP link to be a separate network. All we need to do is to tell Linux to route packets from one network to the other. This is done by turning on "IP forwarding."

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Do that as root. After that, when Linux sees a packet from an outside network it will route it to the DOS machine running mTCP.

After that, you should be able to ping other machines on your home network. If you setup your nameserver in mTCP you should be able to get to the outside world as well. And other machines will be able to ping your machine.

Advanced options

What has been presented above is the bare minimum to get a DOS machine running mTCP on a local home network. Your home network might not look like this, requiring more work. Some other options include:

- Running IP masquerading (NAT) on the Linux machine.
- Running the SLIP network on a different subnet and having Linux routing to that subnet. This requires advanced routing on other machines in your network if you are using non-routable networks like 192.168.x.x.

All of this is left as an exercise to the interested reader.

DOS Idle/Power control for laptops and Virtual Machines

Introduction

Part of the mTCP design requires the application programmer to periodically check for new packets that have arrived so that they may be processed. That check is supposed to be done when the application is not actively working on something else; it is a core part the idle loop in an event driven program.

On old classic hardware running DOS such as the IBM PC or PCjr this polling behavior does no harm. But it does keep the CPU busy and that can show up when using mTCP applications in a virtual environment such as VirtualBox or VMWare. A side effect is higher power usage which is an important consideration for laptop users.

How mTCP minimizes CPU usage

Starting with the 2012-04-29 version mTCP applications will try to minimize unnecessary CPU usage. When in the idle loop each application will check for packets and work to do and if nothing is found they will try to "sleep" for a short period. The mechanisms to do this are:

- Calling the DOS "idle" interrupt 0x28. Traditionally this allows TSR programs to come active while DOS is waiting for a keystroke but it is also a general purpose way for DOS to signal that the machine is generally idle.
- Calling int 0x2F function 0x1680. This is the "VM yield time slice" signal.

The first method (int 0x28) is controversial; many people will claim that it is not needed. It has existed since DOS 2.x and it is documented as being used by DOS to signal to TSRs that there is idle time. After doing a lot of research and testing there is no reason why an application should not use it too, especially if the application does not use DOS for keyboard input (thus depriving DOS the ability to call int 0x28 itself.)

If power saving is enabled in mTCP (which is the default) the calls to int 0x28 will always be made during idle loops.

The second method (int 0x2f 0x1680) is well documented and should work in other environments such as DOS boxes in Windows, OS/2, and other virtual environments. I've tested it in Windows XP when using SwsVpkt.

When mTCP first starts up it will use int 0x2f function 0x1680 to see if the function is supported. If it is supported mTCP will use it during idle loops.

Notes on specific power saving programs

POWER

The standard power management utility for MSDOS and PCDOS is POWER.EXE which is installed as a device driver and controlled from the command line.

While POWER seems to work to reduce CPU consumption at the DOS command line, it does not help with the mTCP idle loop. So it does not harm anything, but it doesn't help either.

POWER works correctly on real hardware and in VMWare. But the current version of VirtualBox seems to have a bug that causes the FTP client to hang up and freeze when POWER is loaded. If you see weird behavior like this under VirtualBox get rid of POWER and use a different utility.

FDAPM

FDAPM is an open source replacement for POWER that ships with FreeDOS and can be downloaded separately.

The FDAPM power control utility detects the calls to int 0x28 to see when DOS is idle. This has a very noticeable effect when running mTCP applications with FDAPM loaded; instead of making a processor fully busy the processor will idle at a low CPU utilization. Here is the command line I use to enable FDAPM:

```
fdapm apmdos
```

If you run that FDAPM will detect calls to int 0x28 and try to adjust your hardware to save power. For more information on FDAPM go to: http://help.fdos.org/en/hhstndrd/base/fdapm.htm

DOSIDLE

DOSIDLE is an older power management utility that can replace POWER. I've not experimented a lot with it, but it works so well that it forced mTCP to save power even when you have told mTCP not to try to save power!

DOSIDLE can be found at http://maribu.home.xs4all.nl/zeurkous/download/mirror/dosidle.html.

Disabling power control

In the unlikely event that you need to disable the power control calls in the mTCP idle loop you can set an environment variable. The name of the variable is "MTCPSLEEP" and an example of how to set it to disable power control is shown:

```
set MTCPSLEEP=0
```

If you do this mTCP will not attempt to call int 0x28 or int 0x2f function 0x1680 while it is in an idle loop.