

LCD Menu System

The menus, along with the web server, which is described later, allow access to all of the Net Butler's features. The two pushbuttons next to the LCD select which menu and submenu to display, and pressing both buttons at the same time performs a menu-specific action. The display is automatically updated if the status of an item being displayed changes, such as from a timer-based action or from clicking on a button on one of the Net Butler's web pages. The following table shows all the menus and the various actions that can be performed:

Menu	Submenu	Display	Action when both buttons pressed
Main	Status	System status & activity	Toggle activity indicator latched mode
	Version	Software version	Execute bootloader
Weather	12-hour	Current day's weather	Send 12-hour weather update request
	5-day	Upcoming week's weather	Send 5-day weather update request
System	Wifi	Wifi network status	Toggle wifi enabled / disabled
	Printer 1	Printer 1 (USB) power status	Toggle printer 1 on / off
	Printer 2	Printer 2 (External) power status	Toggle printer 2 on / off
Network	IP address	IP address	Force DHCP lease to expire
	Netmask	Netmask	Force DHCP lease to expire
	Gateway	Gateway IP address	Force DHCP lease to expire
	Nameserver	Nameserver IP address	Force DHCP lease to expire
Conflict	-	IP conflict device's MAC address	-

Debug Port

It's not necessary to connect to the serial port to use the debugging features. The serial port character I/O routines also can read and write packets to a TCP connection with its TELNET server. It waits for an incoming connection, then treats input and output data as if it were coming from the serial port. It sends a TCP keep-alive packet every 30 seconds if there's no output data being sent. That lets the server automatically drop a connection if the client is no longer active. I found this is needed for when the laptop goes into sleep mode, since it resets all network connections when it wakes back up without ever closing them down properly.

There were no sockets available after all the other features were written, so I chose to share the ICMP socket used for destination unreachable processing with the TELNET server. Since they can't both use the socket at once, the TELNET server is only enabled when ICMP processing is disabled. This is done with the Network Protocols section of the Configuration Settings web page described later on.

Most application and protocol routines have one or more debug flags that can be set to enable messages, which show their activity and state transitions. These can be set or cleared by sending either an uppercase or lowercase letter, respectively, in the range A-P. These are useful when debugging, but they can also help to analyze some unexpected change in system behavior, such as from changes or upgrades to external devices that communicate with the Net Butler. For this reason, I've kept the debugging code enabled in the final version of the software, so the messages can be activated at any time.

DHCP Client

The Net Butler can be configured in either DHCP or static IP mode. The DHCP client attempts to locate a DHCP server on the network and negotiate a lease for an IP address. If it fails, it can either fall back to

static IP mode (if Dhcp Enabled mode is set), or retry forever until it succeeds (if Dhcp Required mode is set). The Link LED will flash before configuration has been completed, and be on solid afterwards.

I originally wrote the protocol to match the specifications in RFC 2131, then made some minor tweaks after trying it with a couple of servers, one of which didn't follow all the details of the specification.

When DHCP is first started, it is the only network protocol that is active. Once it completes and an IP address has been assigned, the rest of the protocols and applications are enabled. When the lease renewal time comes up (after half of the 24 hour lease has elapsed, in my network) the DHCP client runs without interrupting the rest of the system. If it's unable to renew the lease, then it lets the remainder of the existing lease run out. At that point, it interrupts all other network operation while it starts over to obtain a new lease I configured the RG's DHCP server to assign it the same fixed IP address each time it asks for one.

DNS Proxy

All devices on the network send their DNS requests to the Net Butler, which passes them on to the RG. The RG has its own DNS proxy, which allows it to resolve names for devices on the local network, and pass others on to the network provider's nameservers. Sometimes those nameservers can be sluggish, and our DNS proxy can detect the problem and send requests to one or more alternate nameservers instead. This is more convenient, and I've found to be more reliable, than the retry mechanisms built into the various devices on the network.

The domain name block list is stored in flash memory, between the end of the program code and the start of the bootloader. It's at a fixed location, so it's not affected if new code is downloaded. The list consists of a pointer array that points to the beginning of each entry, followed by the entries. Each entry consists of a length byte and the domain name in the encoded format that's used in DNS records. When the list is updated, the entire area of memory is erased and rewritten with the new data.

The DNS activity log is stored in RAM, and consists of a pointer array that points to the beginning of each entry, and a buffer containing the entries. Each new entry is appended to the end of the existing entries when it is first created. The entries contain the time the entry was created and when it was last accessed. The time is simply the number of minutes since the system was last reset. There is also a sequential number that increments on each access, which is used when entries are sorted by access time, to place entries in the proper order when there are several with the same time. The entries also have a counter that increments on each access, a byte containing the blocked and grouped flags, and a byte with the length of the name field. This is followed by the domain name, in DNS-record encoded format.

Web Client

When the wifi application gets a request to change the network status, it sends out a POST request with the exact data the RG expects to see as if I had clicked a button on its network configuration web page. Then it sends out a GET request to a different RG page to check the network status, and to see if there are any wireless devices currently active on the network.

The HTTP Client protocol engine navigates to a page and supplies data to the application as it arrives. It handles the header status codes according to the HTTP specification, including handling retries due to server errors and redirected web pages. It can generate both GET and POST requests. Basic operation is with a state machine that sequences through the process of looking up a host address, connecting to the host, sending the request and optionally the POST data, receiving the web page header and data, and finally disconnecting.

The data receive process was too complex to handle with the main state machine, so it has a separate control flow. One of the key features of the client is that it can handle arbitrarily large web pages without buffering them. The receive process hands off a small section of each page at a time to the web client application using a callback function, as each block comes in. The application then scans the page for any

fields it needs to collect data from. Since a field might cross a block boundary, the code needs to save some data from the end of each block and paste it onto the beginning of the next block before handing it off to be scanned. That prevents a field from being missed if it's split across blocks. The value returned by the application's callback function specifies how many bytes to save from the current block. It needs to be set to the size of the largest field that's currently being searched for. A return value of zero means that the application is done and no more data is needed.

The protocol engine uses a simple DNS client to look up web addresses. It's completely separate from the DNS Proxy application. It uses a simple state machine and provides a status code to the Web Client to indicate when it has finished, or if an error has occurred.

Web Server

The web server provides access to all of the Net Butler's features in one place. Like the web client, it consists of two parts: an application that generates dynamic web pages, and a protocol engine that manages the connection to the client. There are five different pages available. They can be seen in Screens.pdf.

- Home Page – Network settings display, Wifi and printer control, Weather display
- Configuration Settings – System configuration for network, protocols, and applications
- ARP Request Table – ARP Reply Server request table showing all devices on the network
- DNS Block List – DNS Proxy domain name block list display and update
- DNS Activity Log – DNS Proxy domain name activity log display

Pages are generated from the raw data each time they are requested. Each page consists of dynamic data combined with string constants. Rather than build a complete page in a large buffer, I wrote the low-level routine `sendbuf()`, based on WIZnet's original `TCP send()` function, that buffers data directly in the TCPIP core's transmit buffer until enough has been collected to send out as a TCP packet. It automatically sends out the packet while it collects more data to send, without any intervention by the caller. When the page is finished, calling `sendbuf_end()` sends out any data remaining in the buffer.

The configuration settings page is generated differently from the others. Since the page is so large, and much of the data is made up of repetitive patterns, it's stored as a specially formatted data table. The table is made up of text data, which is output unchanged, and 1-byte control codes, which are expanded into text fields. For example, `0x02` expands to "`<td>`". The special control code `0x01` is converted into an HTML form input field, which is automatically filled in with the value of an item from the system's stored configuration data. A separate pair of arrays specifies which configuration item to use, and the format of the item, for each input field. The name of each form input field, which is part of the data returned by the browser when the form is submitted, includes a number that is the index into each of these arrays.

When a submit button is pressed on a page, the browser will send an HTTP POST request to the web server. Each page has its own routine to parse the data and take appropriate action.

The HTTP Server protocol engine's state machine handles the details of the connection. It listens for incoming connections, looking for valid GET and POST requests. For a POST request, or a GET request with posting data included with the URL (following a question mark), it calls the application's POST callback function with the data. Then it calls the GET callback to generate the web page that is sent to the client. The GET function can output part or all of the web page, and returns with a true value when the page is complete, which lets the server know to disconnect and start waiting for a new incoming connection.