EcoNode Series Gateways

C/C++ Development Guide

Debugging and Performance Analysis

Table of Contents

5 Porting Commonly Used Open Source Resources

We have ported some commonly used third-party resources, and these will be released in the form of compressed files. Next, we will use the already ported software as examples to explain the method for porting software packages. However, different open source projects have their own characteristics, so actual porting work will require hands-on experimentation and reading through project files and code to troubleshoot errors.

5.1 Ported Tools

The open-source resources listed in the table below have been ported into our toolchain package and are ready for direct use. For specific usage instructions, please refer to the documentation available on the official website.

Name	Version	Description
paho-mqtt	1.3.9	MQTT protocol client implementation
openssl	1.1.1q	Encryption and decryption
libbzip2	1.0.6	Compression library for bz2 files
libpcap	1.10.1	Packet capture library for network cards
libnet	9.0.0	Packet injection library for network cards
alioss	-	Alibaba Cloud OSS interface SDK
libiconv	2.6.1	Character set conversion library
libcurl	4.8.0	Common protocol library supporting http, ftp, etc.
liblzma	5.2.6	Compression algorithm
libuv	1.4.44	Famous asynchronous networking library used in Node.js
libpcre	8.45	PCRE regular expression library for C programming
libssh2	1.0.1	SSH library supporting SSH1 and SSH2
libtinyxml	2.6.2	A simple XML library
libzstd	1.5.3	Compression algorithm
zlib	1.2.12	Compression algorithm
rapidjson	-	C++ JSON library

Figure 31: Ported Third-Party Libraries

5.2 Porting **Method**

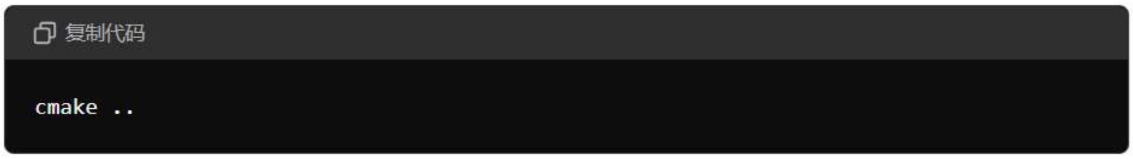
The following content explains how to port software using examples of successfully ported projects. Generally, porting work involves various issues, and you may need to read project files and sometimes source code to make necessary modifications for successful porting. The examples provided are specific to particular projects and may not apply to all porting tasks, but the overall process is similar. Engineers need to analyze specific issues and carefully troubleshoot any problems that arise during porting.

After completing the porting, ensure to back up the modified code for future use. It is also advisable to lock the library versions. If you have the resources, maintaining a branch for your own use is recommended.

5.2.1 Porting CMake-based Project Management Software Packages

libzip uses CMake for project management. To compile, follow these steps:

1. Create a temporary directory for compilation using mkdir, for example: mkdir build.
2. Enter the directory and execute CMake:

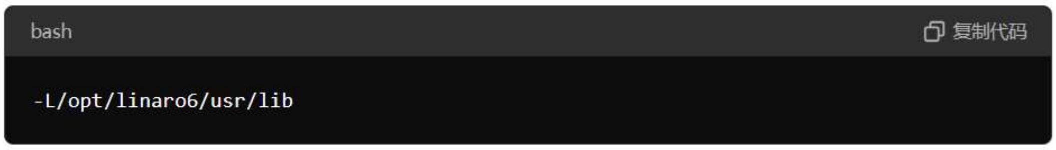
   

   ```
   cmake ..
   ```

   Here, .. indicates that the CMakeLists.txt file is in the parent directory. -D is used to define or modify variables. In this case, it modifies the C compiler to use the ARM 32-bit version and sets the installation directory to /opt/linaro6/usr.

   This command will check the compilation environment and verify that dependencies are correct. The execution time may be longer, and the command will generate control files in the build directory. If there are configuration errors, the process will stop and notify you. Usually, modifying and reconfiguring will resolve the issue. If errors persist even after correct modifications, and the configuration is correct, delete all contents in the build directory and reconfigure after adjusting the path.

3. Execute make all to compile. This process might take a long time and could encounter errors. If the CMake configuration for linking is not handled correctly, you can modify the CMakeLists.txt file or use a shortcut by directly modifying the link command. Locate the link.txt file in the CMakeFiles directory within the subdirectories, and modify the paths as needed, for example:

   ```bash
   -L/opt/linaro6/usr/lib
   ```

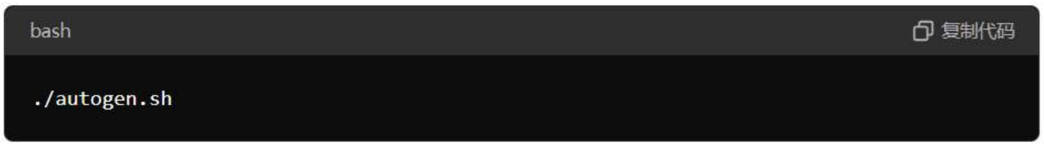
   Then, use the -l option to adjust the library names being linked.

4. Recompile after making the modifications, and once compilation is complete, use make install to install the software.

5.2.2 Porting Autoconf-based Project Management Software Packages

libcurl uses Autoconf for project management and provides autogen.sh to generate the configure script. Follow these steps for porting, using 32-bit porting as an example:

1. Execute autogen.sh to generate configure:

   ```bash
   ./autogen.sh
   ```

   Verify the creation of the configure script using ls configure.

2. Execute configure --help to view configuration parameters. Configure the project according to your requirements, or proceed with the default configuration:

```bash
./configure
```

This command may take some time and generate extensive logs. The final output should indicate that the Makefile has been created, confirming that the project configuration is normal. The host parameter specifies the prefix for the compiler to be used, and the prefix parameter specifies the installation directory.

3. Execute make all to compile. For faster compilation, you can use multiple processes by adding the -j parameter, for example:

```css
make all -j4
```

The number 4 indicates that four processes will compile simultaneously. This number should be based on your computer's CPU cores and disk speed; a higher number does not always mean faster compilation. The process will generate extensive logs. If there are no errors, a dynamic library will be produced. Otherwise, the compilation will stop and show errors.

4. After compilation is complete, execute make install to install the software.

5.2.3 Porting Makefile-based Project Management Software Packages

tinyxml uses Makefile for project management. This project controls the compilation into executable files. Additionally, Makefile does not support cross-compilation by default, so manual modifications to the Makefile are required for compilation.

1. Modify the Makefile for the compiler settings.
2. Adjust the compilation target section by removing the test program code and changing the output from executable files to dynamic libraries.
3. Execute make all. After completion, copy the output results and *.h files to your library directory.

5.3 Recommended Common C/C++ Open Source Libraries

The following table lists some recommended C/C++ open source projects. We have already ported most of these. For those not yet ported, select and port them as needed.

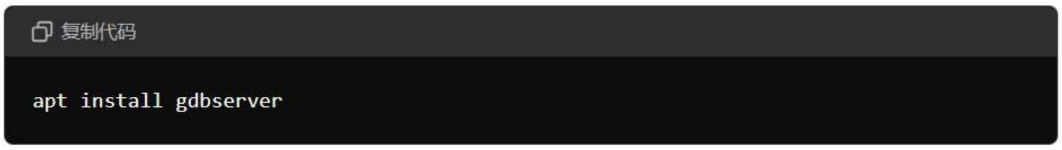
Name	Description	Acquisition URL
boost	A comprehensive C++ standard library providing a wide range of algorithms and frameworks	boost.org
libcurl	A widely-used network protocol framework, very simple to use	curl.se

6 Debugging and Performance Analysis

Linux provides a rich set of debugging and performance analysis tools. Our toolchain includes gdbserver and gdb. On the gateway, you can install various tools like valgrind, tcpdump, etc.

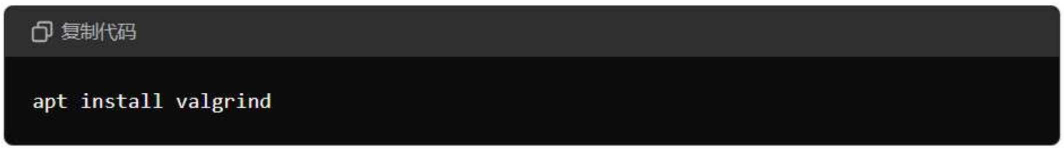
gdbserver runs on the gateway. You can either upload gdbserver from the toolchain to the gateway or install it directly using the apt command. Use the following commands to install these tools:

- To install gdbserver:



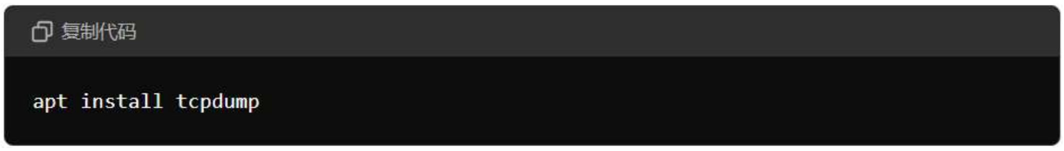
```
apt install gdbserver
```

- To install valgrind:



```
apt install valgrind
```

- To install tcpdump:



```
apt install tcpdump
```

6.1 Using gdbserver and gdb

gdb is a powerful debugging tool with many commands for debugging. Below are some commonly used gdb commands:

- **run (r)**: Start the program and run it. You can provide arguments, e.g., run arg1 arg2.
- **break (b)**: Set a breakpoint at a specified function, line number, or address. For example, break main, break 10, or break *0x0804844.
- **delete (d)**: Delete a breakpoint. You can delete a specific breakpoint, e.g., delete 1, or all breakpoints, e.g., delete.
- **continue (c)**: Continue execution until the next breakpoint or the end of the program.
- **next (n)**: Step over to the next line of code without entering functions.
- **step (s)**: Step into the next line of code, including entering functions.
- **print (p)**: Print the value of a variable, e.g., print x.
- **display**: Continuously print the value of a variable, e.g., display x.
- **watch**: Watch a variable's value for changes, e.g., watch x.
- **backtrace (bt)**: Print the current function call stack.
- **info**: Display information about the program's state, e.g., info breakpoints, info locals.
- **set**: Set the value of a variable, e.g., set x=10.
- **finish**: Execute the current function until it finishes.
- **quit**: Exit gdb.

These are some commonly used gdb commands. gdb offers many other commands and options, which can be explored using the help command.

6.1.1 Command Line Remote Debugging

To debug gateway programs using gdb and gdbserver, follow these steps:

1. **Start gdbserver on the target machine**:

   ```php
   gdbserver <host>:<port> <program> <program_args>
   ```

   Here, <host> is the IP address for gdb to connect to, <port> is the port number, <program> is the program to be debugged, and <program_args> are the arguments for the program.

2. **Start gdb on the development machine**:

   ```php
   gdb <program>
   ```

   This example shows how to debug a 32-bit program.

3. **Set the target machine's gdbserver connection information in gdb**:

   ```ruby
   target remote <host>:<port>
   ```

   Where <host> and <port> match those used when starting gdbserver.

4. **Set breakpoints, watch variables, and configure other debugging information in gdb.**
5. **Run the program in gdb**:

   ```arduino
   run
   ```

6.1.2 Remote Debugging with Qt Creator

To set up remote debugging in Qt Creator:

1. Open Qt Creator and load the project you want to debug.
2. Click on **"Tools"** -> **"Options"**.
3. In the Options dialog, select **"Build & Run"**, then go to the **"Debugger"** tab.
4. Click **"Add"** and choose **"GDB Server"**.
5. In the **"GDB Server"** dialog, enter the IP address and port number of the target machine's gdbserver, e.g., 192.168.1.100:1234.
6. Click **"OK"**.
7. To start remote debugging in Qt Creator, click on **"Debug"** -> **"Start Debugging"** -> **"Remote Debug"**.

8. In the **“Remote Debug”** dialog, select the GDB Server you set up earlier.
9. Click **“OK”**.
10. Perform remote debugging in Qt Creator. Set breakpoints, watch variables, and use other debugging features.
11. Click on **“Debug”** -> **“Continue”** or **“Step Over”** to proceed with debugging.
12. When debugging is complete, click **“Debug”** -> **“Stop”** to end the debugging session.

6.2 Multithreaded Debugging

When debugging multithreaded programs with gdb, consider the following points:

- **Enable Multithreading Support**: When compiling the program, make sure to enable multithreading support. For example, use the -pthread option with gcc.

```bash
gcc -pthread -o my_program my_program.c
```

- **Set Debugger Options**: When starting gdb, configure the set follow-fork-mode child option to continue debugging in the child process.

```gdb
set follow-fork-mode child
```

- **Use Thread-Related Commands**: gdb provides several commands for handling threads, such as:
  - info threads: View information about the current threads.
  - thread N: Switch to thread number N.
  - thread apply N command: Apply a command to all threads.

Here’s an example of a multithreaded program and the steps to debug it with gdb:

1. **Compile the Program and Start gdb**.
2. **Set the set follow-fork-mode child Option**.
3. **Set Breakpoints and Run the Program**.
4. **View Thread Information**.

```gdb
info threads
```

5. **Switch to a Specific Thread**.

```gdb
thread N
```

6. **Set Breakpoints and Execute Commands in the Specific Thread**.
7. **Switch Back to the Main Thread**.

8. **Set Breakpoints and Execute Commands in the Main Thread**.

During multithreaded debugging, be mindful of synchronization issues between threads, such as locks and condition variables, and note that multithreaded debugging may impact program performance.

6.3 Valgrind Installation and Memory Analysis

Valgrind is a tool used for detecting memory leaks, memory errors, and threading issues in programs. It supports various platforms and programming languages, including C, C++, and Java. Valgrind includes several tools, with Memcheck being the most commonly used.

Memcheck can detect memory errors such as accessing uninitialized memory, accessing freed memory, and out-of-bounds memory access. It tracks memory usage during program execution and provides a report upon completion.

To install Valgrind, use the following command:

```bash
apt install valgrind
```

Steps for memory detection with Valgrind:

1. **Compile the Program with Debug Information**: Add the -g option to enable debug information.

   ```bash
   gcc -g -o my_program my_program.c
   ```

2. **Run the Program with Valgrind**:

   ```bash
   valgrind --leak-check=full ./my_program
   ```

3. **View the Report**: Valgrind will output a report containing detailed information about memory errors, including the type of error, location, and error message.

   Example report:

   ```text
   ==1234== Invalid read of size 4
   ==1234==    at 0x4005F2: main (my_program.c:5)
   ==1234==  Address 0x4a3e8d0 is 0 bytes after a block of size 100 alloc'd
   ```

Besides Memcheck, Valgrind includes other tools such as Cachegrind (for cache issues) and Helgrind (for threading issues). These tools can be used by specifying different --tool options.

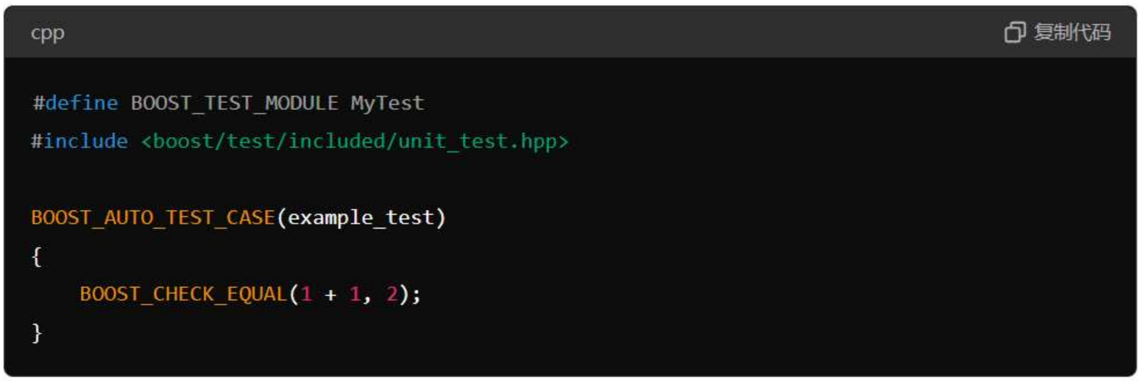
6.5 Unit Testing

Unit testing can be performed using Boost Test. Boost Test is a C++ unit testing framework that is part of the Boost libraries and is used for writing and running unit tests. It provides a simple interface to define test cases, test suites, and fixtures, as well as to assert and output test results.

Features of Boost Test include:

- **Automatic Test Discovery**: No manual registration of test cases and suites is needed.
- **Test Fixtures**: Allows for setup and teardown operations before and after tests.
- **Rich Assertion and Checking Tools**: Verifies the correctness of test results.
- **Multiple Output Formats**: Includes XML, JUnit, HTML, etc., for generating test reports.
- **Integration with Other Frameworks**: Can be integrated with other testing frameworks like Google Test.

Here’s a sample program demonstrating Boost Test usage:



```cpp
#define BOOST_TEST_MODULE MyTest
#include <boost/test/included/unit_test.hpp>

BOOST_AUTO_TEST_CASE(example_test)
{
    BOOST_CHECK_EQUAL(1 + 1, 2);
}
```

6.6 Network Communication Packet Capture and Analysis

During development, network packet capture and analysis are often required. Use tcpdump on the gateway and Wireshark on the PC for packet capture. You can also develop your own capture tools using libpcap.

6.6.1 Wireshark Packet Capture

- **Install Wireshark**:
  - For Windows, download Wireshark from Wireshark's website and install it using the Windows Installer. Installation details are not covered here.
- Ubuntu Installation and Configuration for Packet Capture

1. **Choose the Target Network Interface**:
   - Select the network interface with communication activity. Interfaces with ongoing traffic will be marked with waveform diagrams of the data being exchanged. Choose the correct network interface based on your PC's configuration and double-click to enter the packet capture page.
2. **Set Up Filtering Rules**:
   - By default, Wireshark displays all data, which can be overwhelming and make it difficult to find relevant information. Use filtering rules to narrow down the data to what you're interested in. For example, you can filter by address, port, or protocol.
   - **Filter by IP**: Use the ip filter to specify IP addresses, for example:

```text
ip == 192.168.1.1
```

This filters packets where either the source or destination IP address is 192.168.1.1. You can also filter by source or destination IP specifically:

```text
ip.src == 192.168.1.1
ip.dst == 192.168.1.1
```

Filter by Port: TCP and UDP support port filtering, for example:

```text
tcp.port == 80
```

This filters packets with TCP port 80. You can also filter by source or destination port:

```text
tcp.dstport == 80
```

- **Filter by Protocol**: Wireshark supports filtering by over 1000 protocols. For example, to filter HTTP packets:

```text
http
```

Common protocols include IP, ARP, UDP, TCP, ICMP, HTTP, SMTP, FTP, DNS, SSL, Modbus-TCP, etc.

- **Filter by MAC Address**: Use eth filters for MAC addresses, for example:

```text
eth.dst == A0:00:00:04:C6:77
```

This filters packets with the destination MAC address A0:00:00:04:C6:77.

- **Logical Operations Between Rules**: Wireshark supports logical operations between rules. For example, to filter HTTP protocol and port 3000:

```text
http && tcp.port == 3000
```

To filter packets with port 3000 but not a specific MAC address:

```text
eth.dst != A0:00:00:04:C6:77 && tcp.port == 3000
```

Logical operators supported are && (AND), || (OR), and ! (NOT).

3. **Start and Stop Packet Capture**:
   - See the interface below for commands to start, stop, and restart packet capture. When starting capture, if there are existing packets, Wireshark will prompt whether to save them. Choose to save or discard based on your needs.
4. **Open Saved Files**:
   - Wireshark can open saved packet capture files, which is useful for analyzing data captured with tcpdump on a gateway and then transferred to a PC for analysis.
5. **Packet Analysis**:
   - After selecting a packet in the packet list area, its details are displayed in the packet details area. This area is divided into two sections: packet structure and hexadecimal data representation. Review the packet structure to understand its content, especially for custom protocols.

6.6.2 tcpdump Packet Capture

tcpdump is a convenient command-line tool for packet capture and debugging in Linux. Similar to Wireshark, tcpdump allows you to configure filtering rules and capture packet data. By default, tcpdump displays captured packets in the console, which is not ideal for detailed analysis. Therefore, it also supports saving data to files for further analysis, which can be opened in Wireshark.

- **Install tcpdump**:
  - By default, tcpdump may not be installed on a gateway. Install it using the following command:

```bash
sudo apt install tcpdump
```

After entering Y, the installation will start.

- **tcpdump Commands**: The command format for tcpdump is as follows:

  Capture Options:

  - -c: Specify the number of packets to capture.
  - -i interface: Specify the interface to listen on. By default, it captures from the first network interface.
  - -n: Display addresses as numeric values, not hostnames (i.e., skip hostname resolution).
  - -nn: Also display port numbers as numeric values, not service names.
  - -P: Specify whether to capture incoming, outgoing, or both types of packets (in, out, inout, default is inout).
  - -s len: Set the packet capture length to len. If not set, the default is 65535 bytes. If the length is insufficient, packet truncation may occur, indicated by [|proto] in the output. Capture length should be as small as possible while capturing the necessary data to avoid packet loss.

Output Options:

- -e: Include data link layer header information, such as source and destination MAC addresses.
- -q: Quick print mode; reduces protocol-related information to keep lines short.
- -X: Output packet headers in both hexadecimal and ASCII.
- -XX: Output packet headers in both hexadecimal and ASCII with more detail.
- -v, -vv, -vvv: Produce increasingly detailed output during analysis and printing.

Additional Functional Options:

- -D: List available interfaces for packet capture, including their numerical IDs and names.
- -F: Read capture expressions from a file. Overrides other expressions provided on the command line.
- -w: Write captured data to a file instead of standard output. Can be combined with -G time to rotate files every time seconds. Use -r to read and analyze these files.
- -r: Read packets from a file. Use - to read from standard input.

Example Command:

- To capture packets on interface eth0 with destination address 192.168.1.27 and port 3000, and save the data to packet.pcap:

```bash
tcpdump -i eth0 dst host 192.168.1.27 and port 3000 -w packet.pcap
```

6.6.3 Custom Packet Capture Tool Using libpcap

libpcap is a packet capture library that provides a C function interface for systems that need to capture packets from network interfaces. The library offers a consistent programming interface across different platforms. Programs written using libpcap can be used across different platforms where libpcap is installed.

Using libpcap for packet capture allows for the handling of specific protocol content but can be more complex and time-consuming.

6.1 libpcap Packet Capture Framework

Header File:

```c
#include <pcap.h>
```

Find Network Devices

The function pcap_findalldevs() is used to retrieve a list of network devices that can be used with pcap_open_live() or pcap_lookupnet(). If the function fails, it returns NULL, and an error message is stored in errbuf.

```c
pcap_findalldevs(pcap_if_t **alldevsp, char *errbuf);
```

Obtain Network Address and Netmask

The function pcap_lookupnet() retrieves the network number and netmask for a specified device. netp and maskp are pointers to bpf_u_int32. If the function fails, it returns -1, and an error message is stored in errbuf.

```c
int pcap_lookupnet(const char *device, bpf_u_int32 *netp, bpf_u_int32 *maskp, char *er
```

int pcap_lookupnet(const char *device, bpf_u_int32 *netp, bpf_u_int32 *maskp, char *errbuf);

bpf_u_int32 is a 32-bit unsigned integer. Use inet_ntoa() to display addresses in a.b.c.d format.

1. **Open Network Device**

   The function pcap_open_live() opens a network device for packet capture. The device parameter specifies the network device to open. snaplen defines the maximum number of bytes to capture, with 65535 being the maximum value. promisc specifies whether to set the interface to promiscuous mode (1 for promiscuous mode). to_ms specifies the timeout (in milliseconds), with 0 indicating no timeout. ebuf is used to store an error message if pcap_open_live() fails and returns NULL.

```c
pcap_t *pcap_open_live(const char *device, int snaplen, int promisc, int to_ms, char *
```

pcap_t *pcap_open_live(const char *device, int snaplen, int promisc, int to_ms, char *ebuf);

The pcap structure is defined in pcap-int.h in the libpcap source code, and you typically use its pointer type (pcap_t).

6.2 Compile and Set Filter Conditions

Use pcap_compile() to set filter conditions. Example filter expressions include:

- src host 192.168.1.1: Only capture packets with source IP address 192.168.1.1.
- dst port 80: Only capture packets with TCP or UDP destination port 80.
- not tcp: Only capture packets not using TCP.
- tcp[13] == 0x02 and (dst port 22 or dst port 23): Only capture TCP packets with the SYN flag set and destination port 22 or 23 (13th byte of the TCP header).
- icmp[icmptype] == icmp-echoreply or icmp[icmptype] == icmp-echo: Only capture ICMP ping requests and responses.

- ether dst 00:e0:09:c1:0e:82: Only capture packets with Ethernet MAC address 00:e0:09:c1:0e:82.
- ip[8] == 5: Only capture IP packets with TTL=5 (8th byte of the IP header).

```c
int pcap_compile(pcap_t *p, struct bpf_program *fp, const char *str, int optimize, bpf
```

int pcap_compile(pcap_t *p, struct bpf_program *fp, const char *str, int optimize, bpf_u_int32 netmask);

The fp parameter is a pointer to a bpf_program structure, which is assigned in pcap_compile(). optimize controls the optimization of the resulting code. netmask specifies the local network's netmask; set it to 0 if unknown. On error, it returns -1.

Use pcap_setfilter() to set a compiled filter program:

```c
int pcap_setfilter(pcap_t *p, struct bpf_program *fp);
```

2. **Capture and Read Packets**

   Capture and process packets using pcap_loop() or pcap_dispatch().

   - pcap_loop() captures and processes packets. cnt specifies the maximum number of packets to process before returning. cnt = -1 processes all packets in a buffer. callback is a function with three parameters: a u_char pointer, a pointer to pcap_pkthdr, and a u_char pointer to the packet data.

     ```c
     int pcap_loop(pcap_t *p, int cnt, pcap_handler callback, u_char *user);
     ```

     pcap_dispatch() is similar to pcap_loop(), but it returns after processing cnt packets or encountering an error, without considering read timeouts.

     ```c
     int pcap_dispatch(pcap_t *p, int cnt, pcap_handler callback, u_char *user);
     ```

   - pcap_next() reads the next packet, similar to pcap_dispatch() with cnt set to 1, but does not return the pcap_pkthdr structure.

     ```c
     u_char *pcap_next(pcap_t *p, struct pcap_pkthdr *h);
     ```

   The pcap_pkthdr structure includes:

   - struct timeval ts: Timestamp with seconds and microseconds since 1900.

   - bpf_u_int32 caplen: Length of captured data.
   - bpf_u_int32 len: Actual length of the packet.

  user is a user-defined parameter passed to the callback function.

  On success, pcap_loop() and pcap_dispatch() return the number of packets read. 0 indicates no packets were captured. On error, return -1, and use pcap_perror() or pcap_geterr() to retrieve the error message. Return -2 indicates that pcap_breakloop() was called.

6.3 Close File and Release Resources

Use pcap_close() to close the packet capture file and release resources.

```c
void pcap_close(pcap_t *p);
```

6.7 Serial Communication Packet Capture and Analysis

Gateways typically use RS232 and RS485 serial ports. RS485 is a bus structure that allows easy insertion of taps for monitoring data communication. RS232 is point-to-point, so monitoring requires an RS232 to RS485 converter to tap into the communication.

6.7.1 Using stty, cat, and echo Commands

When serial communication is not working, you can use the stty, echo, and cat commands to configure and test serial communication, eliminating potential interference.

stty Command

The stty command is used to configure and display terminal parameters.

Syntax:

```sh
stty [-F DEVICE | --file=DEVICE] [SETTING]...
stty [-F DEVICE | --file=DEVICE] [-a|--all]
stty [-F DEVICE | --file=DEVICE] [-g|--save]
```

Examples:

- View serial port parameters:

```sh
stty -F /dev/ttyS1 -a
```

Disable input echo:

```sh
stty -F /dev/ttyS1 -echo
```

Enable input echo:

```sh
stty -F /dev/ttyS1 echo
```

Set terminal baud rate to 9600:

```sh
stty -F /dev/ttyS1 9600
```

Options:

- -F DEVICE: Specify the device file (e.g., /dev/ttyS1).
- -a or --all: Display all current settings.
- -g or --save: Save the current settings to be restored later.
- -e: Disable echo (show input characters as they are typed).
- 9600: Set the baud rate to 9600 bps.

Explanation of Settings:

- **Special Characters**: Control characters and special symbols.
- **Special Settings**: Includes options like -echo (disable input echo).
- **Control Settings**: Adjust terminal control settings.
- **Input Settings**: Configure how input is handled.
- **Output Settings**: Configure how output is handled.
- **Local Settings**: Define local terminal settings.
- **Comprehensive Settings**: General settings for terminal operation.

cat Command

The cat command can be used to read data from the serial port and display it on the terminal.

Options:

- -A: Display all file contents including non-printable characters.
- -b: Number non-empty lines, overriding -n.
- -e: Equivalent to -vE.
- -E: Display end-of-line characters.
- -n: Number all lines.
- -s: Suppress multiple adjacent empty lines.
- -t: Equivalent to -vT.
- -T: Display tabs as ^I.
- -v: Show non-printable characters using ^ and M.

Example Usage:

To read and display data from the serial port:

```sh
cat /dev/ttyS1
```

echo Command

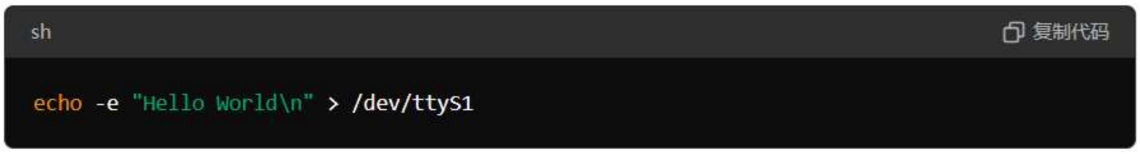
The echo command sends data to the serial port.

Options:

- -n: Do not output a newline after the string.
- -e: Enable interpretation of escape characters (e.g., \n for newline, \t for tab).

Example Usage:

To send data to the serial port:



```sh
echo -e "Hello World\n" > /dev/ttyS1
```

This sends the string "Hello World" followed by a newline to the serial port /dev/ttyS1.

6.7.2 RS232 Self-Test

To verify RS232 communication, connect the TX (transmit) and RX (receive) lines together. By doing this, you can send data out on TX and receive it on RX, allowing you to independently test the functionality of the RS232 serial port hardware and drivers. However, this method does not confirm the overall system’s operational integrity.

Procedure:

1. **Connect TX and RX:** Physically connect the TX pin to the RX pin on the RS232 port.
2. **Send Data:** Use serial communication software to send data through the port.
3. **Receive Data:** Check if the sent data is correctly received.

Note: This test ensures that the serial port hardware and drivers are functioning correctly but does not guarantee that the entire system operates properly.

6.7.3 RS485 Bypass Packet Capture

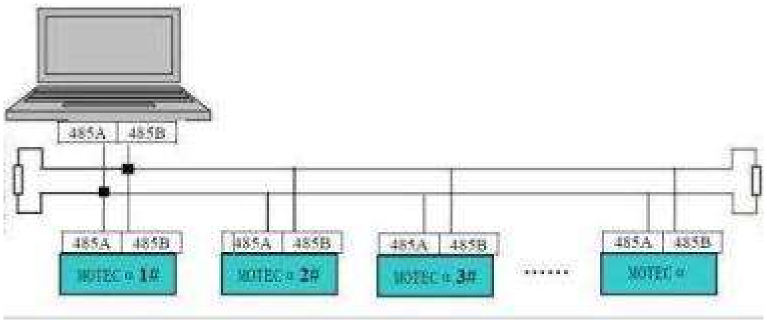
Taking advantage of the RS485 bus structure, you can tap into the bus to monitor and capture data. Connect a computer to the RS485 bus and use serial communication tools to read and analyze the bus data to ensure that it is functioning correctly. The connection setup is shown below:

Procedure:

1. **Connect to RS485 Bus:** Use an RS485 to USB converter or similar interface to connect the computer to the RS485 bus.
2. **Use Serial Communication Tools:** Employ tools such as serial port monitors or sniffers to capture and analyze data transmitted over the RS485 bus.

Diagram: *Include a diagram showing the connection between the RS485 bus, converter, and computer.*

Note: This method allows you to analyze data directly from the bus to verify communication, ensuring that the data transmitted is correct.



6.7.4 Virtual Serial Port Remote Analysis

Use a serial port passthrough tool to open the target serial port remotely. The passthrough tool will convert the serial data into network data, enabling analysis by monitoring the data on the server.

Procedure:

1. **Open Serial Port:** Use the serial port passthrough tool to access the target serial port.
2. **Data Conversion:** The tool will convert serial data to network data.
3. **Server Monitoring:** Analyze the data on the server to check for correct transmission and reception.

Tools:

- **Serial Port Passthrough Tools:** Software that facilitates the conversion and transmission of serial data over a network.

Note: This approach enables remote analysis of serial communication by leveraging network connectivity to monitor and troubleshoot data flow.

6.8 File Transfer

During development and debugging, it is often necessary to transfer files between the development machine and the gateway. We support three methods for file transfer: Zmodem protocol, SCP protocol, and FTP protocol.

6.8.1 Zmodem File Transfer Method

To use Zmodem for file transfer, the sz and rz commands are required on the gateway side. Typically, these commands are pre-installed on the gateway when it is shipped. You can check if these commands are installed by running sz --help. If they are not installed, use the command apt install lrzsz to install them.

Uploading Files to the Gateway:

1. Ensure you are connected to the gateway using RockTerminal.
2. Enter the rz command in the command line.
3. A file selection dialog will pop up in RockTerminal, allowing you to choose the file to upload.

Downloading Files from the Gateway:

1. Use the command sz <filename> on the gateway to download a file to your PC.
2. During the download, a file save dialog will not appear. The file save path can be configured in the software settings.

6.8.2 SCP Protocol for File Transfer

SCP protocol is based on SSH protocol. On the computer side, you can use WinSCP for file transfer. You can download WinSCP from WinSCP.

Connecting to the Gateway Using WinSCP:

1. Select SCP protocol.
2. Enter the gateway's IP address. The port number usually uses the default setting. If your gateway uses a custom SSH port, enter the correct port number.
3. Enter the username and password, which are provided when you purchase the gateway.
4. Save the settings and click the connect button.

After connecting, you will see both the local and remote file directories. You can upload or download files and modify file attributes as needed.

For detailed usage, refer to the WinSCP help documentation.

6.8.3 FTP Protocol for File Transfer

The gateway can also use FTP protocol for file transfer. By default, the gateway does not have an FTP client or server installed. If you prefer using FTP, we recommend installing an FTP client on the gateway and an FTP server on the PC.

Installing FTP Client on the Gateway: Use the command apt install ftp to install the FTP client.

Installing FileZilla Server on the PC: Download FileZilla Server from FileZilla Project. FileZilla is a cross-platform software that can be installed on both Linux and Windows. It is easy to configure. After installation, you can use FTP commands on the gateway for file transfer.

Configuring FileZilla Server:

1. Set up user information.
2. Add users and set usernames and passwords.
3. Ensure users are activated and require a password for login.
4. Add FTP directories and configure directory mapping.

Click "Apply" or "OK" to complete the configuration. The FTP server will be ready for use.

Using FTP for File Transfer:

1. Connect to the server using the ftp command.
2. After successful connection, input the username and password when prompted.
3. Switch to binary mode for file transfer (important).
4. Switch to passive mode.

After completing these steps, you can transfer files. Use the help command at the FTP prompt to see available commands. Commands like ls for listing files, cd for changing directories on the server, and lcd for changing local directories are useful.

Uploading Files to the Gateway:

- Use the get command to download files from the server (PC) to the gateway.

Downloading Files to the PC:

- Use the put command to upload files from the gateway to the server (PC).