

ClairGeo-M Communication Protocol

Version:1.0

TraffiClair Technology Co., Ltd.

General Protocol Format:

Header (4 bytes) + N1 Encoding (4 bytes) + Length (2 bytes) + Data Segment (n bytes) + CRC16 (2 bytes)

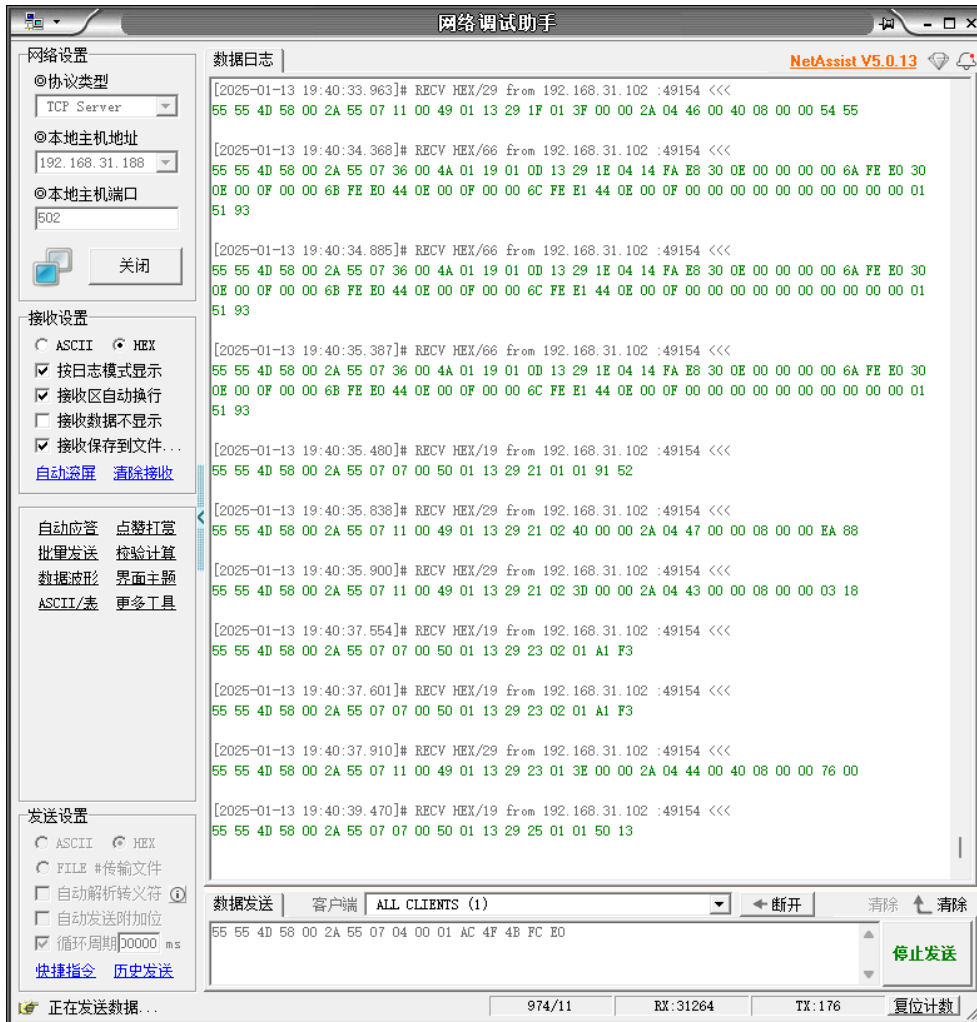
- **Length:** 2 bytes, indicating the length of the data segment.
- **Checksum (CRC16):** 2 bytes, calculated starting from the N1 encoding, covering the N1 encoding, length, and data segment (highlighted in blue).

Network Transmission Connection:

- Initiator: N1
- Request: 0x01
- **Responder:** Central Server
- Command: 0x01 + 0xAC + 0x4F + 0x4B

Example:

- Device Encoding: 07552A00
- **Device Name:** Mawan Tunnel North-to-South Exit
- Server (TCP Server): 172.16.3.78, Port: 502
- Scheduled Transmission: 55 55 4D 58 00 2A 55 07 04 00 01 AC 4F 4B FC E0 (sent every 5 minutes).



Example:

- Device Encoding: 075529FF
- Device Name: Mawan Tunnel South-to-North Entrance
- Scheduled Transmission:

55 55 4D 58 FF 29 55 07 04 00 01 AC 4F 4B FF C1 (sent every 5 minutes).

Protocol Breakdown (Hex):

1. Header (4 bytes): 55 55 4D 58
2. N1 Encoding (4 bytes): FF 29 55 07 (Little-endian representation of 075529FF)
3. Length (2 bytes): 04 00 (Data length = 4 bytes)
4. Data Segment (4 bytes): 01 AC 4F 4B (Command: 0x01 + Response AC 4F 4B)

5. CRC16 (2 bytes): FF C1 (Checksum covering N1 encoding, length, and data segment)

Transmission Details:

- Protocol: TCP
- Server: 172.16.3.78:502
- Interval: 5 minutes (repeating)

(Note: The checksum FF C1 is calculated over the N1 encoding, length, and data segment as per the protocol definition.)



Real-Time Data Packet Structure

Format:

0x49 + AP ID (1 byte) + Hour (1 byte) + Minute (1 byte) + Second (1 byte) +
 Lane ID (1 byte) + Real-Time Speed (2 bytes) + Vehicle Type (1 byte) +
 Vehicle Length (1 byte) + Headway Time (1 byte) + Gap Distance (2 bytes) +
 Occupancy Time (4 bytes)

✓ N1 Data Sample:

[2025-01-13 19:29:07.913]# RECV HEX/29 from 192.168.31.102 :49153 <<<
 55 55 4D 58 00 2A 55 07 11 00
 49 01 13 1E 05 02 26 00 00 2A 04 2A 00 40 08 00 00
 0D 32

✓ Packet Breakdown:

- Header (Fixed): 55 55 4D 58
- Device ID: 00 2A 55 07
- Length: 11 00 → 17 bytes of data
- Data Payload:

Byte(s)	Value	Description
1	49	Packet type (real-time data)
2	01	AP ID
3	13 (19)	Hour
4	1E (30)	Minute
5	05 (5)	Second
6	02	Lane ID
7-8	26 00 (0x0026 = 38)	Real-time speed (km/h)

Byte(s)	Value	Description
9	00	Vehicle type
10	2A (42)	Vehicle length (in decimeters?)
11	04 (4)	Headway time (seconds or deciseconds?)
12-13	2A 00 (42)	Gap distance (cm or dm?)
14-17	40 08 00 00 (0x00000840 = 2112)	Occupancy time (ms?)

- CRC: 0D 32

✓ Notes:

- Each real-time data packet is **sent 4 times consecutively**.
- **Speed** is likely in km/h (38 km/h in this case).
- **Vehicle type, length, gap, and occupancy time** interpretation may depend on calibration or system design.
- CRC is calculated as described using the provided CRC16 algorithm, covering only the data payload (p+4 to len-6).

✓ Modify Timed Data Interval

1. Establish a TCP Client Connection

- **Target Host IP:** The IP address of the front-end device
- **Port:** 12310

2. Example Devices:

- **Ma Wan Tunnel South-North Entrance:** 172.16.6.4
- **Ma Wan Tunnel North-South Exit:** 172.16.3.77

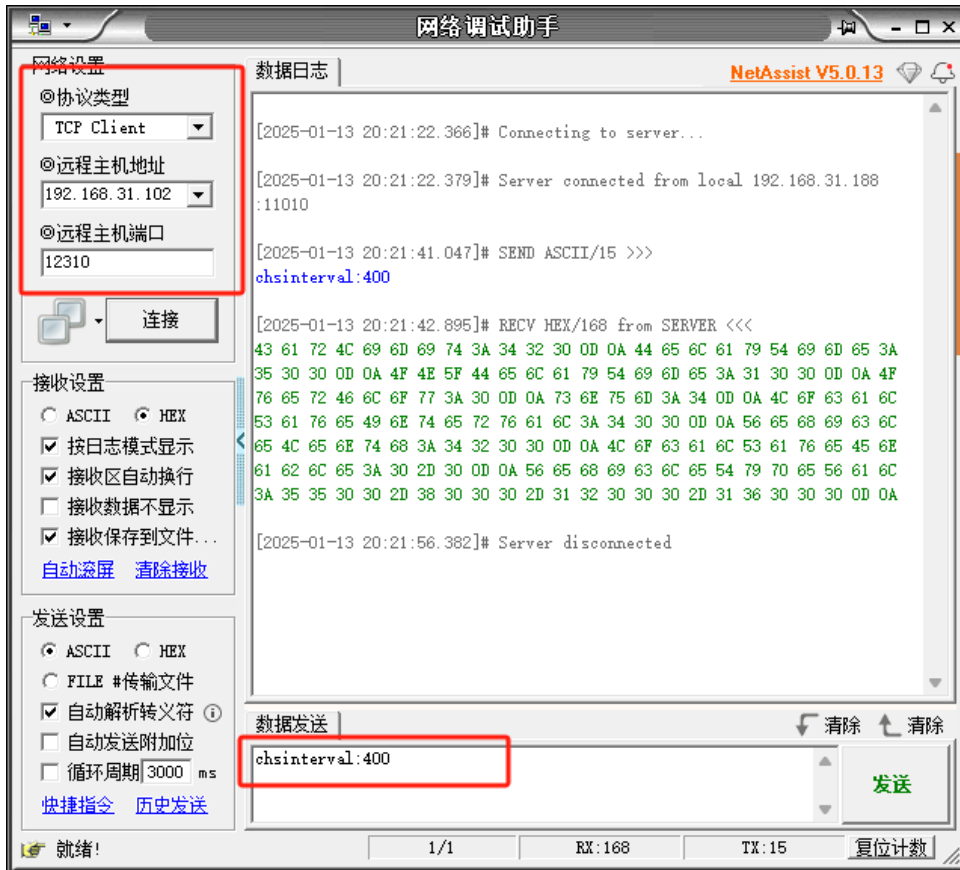
3. Send the Following ASCII Command:

chsinterval:400

- 400 is the new interval value in **seconds**

4. Note:

- This command changes how often the device sends **timed status data**.
- Replace 400 with any other desired value in **seconds**.



0x44: Periodic Traffic Flow Data Format

Format:

0x44 + AP ID (1 byte) + Year (1 byte) + Month (1 byte) + Day (1 byte) + Hour (1 byte) + Minute (1 byte) + Second (1 byte) + Lane Count n (1 byte)
 + [Detector ID (2 bytes) + Vehicle Class Flow (2 bytes * 5 classes) + Occupancy (2 bytes) + Average Speed (2 bytes)] * n lanes

Sample Data:

[2025-01-13 19:29:03.369]# RECV HEX/85 from 192.168.31.102 :49153 <<<
 55 55 4D 58 00 2A 55 07 49 00 44 01 19 01 0D 13 1E 00 04

14 FA 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 6A FE 07 00 00 00 00 00 00 00 00 00 00 35 00 00 00
 6B FE 07 00 00 00 00 00 00 00 00 00 00 31 00 00 00
 6C FE 07 00 00 00 00 00 00 00 00 00 00 31 00 00 00
 26 C7

Parsed Result:

- **Header:** 55 55 4D 58
- **Device ID:** 00 2A 55 07
- **Payload Length:** 49 00 (73 bytes)
- **CRC:** 26 C7

Data Section (Traffic Flow):

44 01 19 01 0D 13 1E 00 04

- 0x44: Flow data flag
- 01: AP ID
- 19: Year = 2025 (0x19 = 25 + 2000)
- 01: Month = 01
- 0D: Day = 13
- 13: Hour = 19
- 1E: Minute = 30
- 00: Second = 0
- 04: 4 lanes

Then comes data for 4 detectors (lanes), each 18 bytes:

[Detector ID (2B) + Class Flow (2B * 5) + Occupancy (2B) + Avg Speed (2B)]

Periodic Status Information:

Each status message is transmitted 4 times in a row.

Message Structure:

0x4A +

AP ID (1 byte) +

Year (1 byte) +

Month (1 byte) +

Day (1 byte) +

Hour (1 byte) +

Minute (1 byte) +

Second (1 byte) +

Number of Detectors n (1 byte) +

[Detector ID (2 bytes) +

Signal Strength RSSI (1 byte) +

Battery Voltage (2 bytes, in millivolts) +

Packet Loss Rate (1 byte, as %) +

Number of Received Packets (2 bytes) +

Detector Status (1 byte)

] * n +

Number of RP m (1 byte) +

[RP ID (2 bytes) +

Signal Strength RSSI (1 byte) +

Battery Voltage (2 bytes) +

Reserved (1 byte) +

Reserved (2 bytes) +

RP Status (1 byte)

] * m +

AP Status (4 bytes) +

SD Card Status (1 byte) +

RTC Status (1 byte) +

Restart Flag (1 byte) +

Configuration Change Flag (1 byte)

Notes:

- Battery Voltage: Unit is millivolts (mV)
 - Packet Loss Rate: Expressed as percentage
-

Status Codes:

Detector Status (1 byte):

- 0x00 – Normal
- 0x01 – Packet loss > 10%, lost packets < 10
- 0x02 – Packet loss > 10%, lost packets ≥ 10
- 0x04 – No status packets or events for 2 hours, marked as faulty

RP Status (1 byte):

- 0x00 – Normal
- 0x01 – No data in current statistics cycle
- 0x02 – No data for 2 hours, marked as faulty

AP Status (4 bytes):

- 0 – Working normally
- 1 – No wireless data received for 5 minutes

SD Card Status (1 byte):

- 0 – Normal
- 1 – SD card failure
- 2 – SD card space less than 10 MB

RTC Status (1 byte):

- 0 – Normal
- 1 – RTC time error

Restart Flag (1 byte):

- 0 – No restart occurred
- 1 – Restarted by SM software
- 2 – Restarted by MPD software
- 3 – Restart due to TCP memory block allocation failure
- 4 – Restart triggered by central platform
- 5 – Restart due to ETH BSP config failure
- 6 – Scheduled restart
- 7 – Restart due to mismatch in IP, MAC, or config read from AP
- 8 – Restart due to change in server IP or port
- 9 – Restarted for other reasons

Configuration Change Flag (1 byte):

- 0 – No change in intersection configuration
- 1 – Intersection configuration changed

Sample Data:

[2025-01-13 19:29:04.387]# RECV HEX/66 from 192.168.31.102 :49153 <<<

55 55 4D 58 00 2A 55 07 36 00 4A 01 19 01 0D 13 1E 00 04

14 FA E7 30 0E 00 00 00 00

6A FE E1 30 0E 06 0F 00 00

6B FE E1 44 0E 00 0F 00 00

6C FE E2 44 0E 00 0F 00 00

00 00 00 00 00 00 00 00 01

Parsed Result:

- **Header:** 55 55 4D 58
- **Device ID:** 00 2A 55 07
- **Payload Length:** 36 00 (54 bytes)
- **CRC:** C9 35

Data Section (Status):

4A 01 19 01 0D 13 1E 00 04

- 0x4A: Status data flag
- 01: AP ID
- 19: Year = 2025
- 01: Month = 01
- 0D: Day = 13
- 13: Hour = 19
- 1E: Minute = 30
- 00: Second = 0
- 04: 4 detectors

Each detector:

- Detector ID (2B)
- RSSI (1B)
- Battery Voltage (2B) in mV
- Packet Loss % (1B)
- Packet Count (2B)
- Detector Status (1B)

Then:

- RP Count = 00
- RP data: None
- AP Status (4B): 00 00 00 00 = Normal

- SD Card Status (1B): 00 = Normal
- RTC Status: 00 = Normal
- Restart Flag: 00 = No reboot
- Config Change Flag: 01 = Configuration changed

CRC Checking:

```

/**
 * Function Name: checkdata
 * Function Purpose: To verify whether the received data packet is valid
 * Parameters:
 *     @p : uint8_t*, pointer to the received data buffer
 *     @len : int, total length of the received data (including header, data section, and CRC)
 * Return Value:
 *     int, returns 1 if the data is valid, 0 if invalid
 * Function Logic:
 *     1. Extract the CRC checksum from the last two bytes of the data
 *     2. Calculate the CRC16 value over the data segment
 *         (excluding the 4-byte header and the 2-byte CRC)
 *     3. Compare the calculated CRC with the extracted CRC
 *     4. Verify the header starts with fixed bytes: 0x55 0x55 0x4D 0x58 ("UUMX")
 *     5. Return 1 if all checks pass, otherwise return 0
 */
int checkdata(uint8_t *p,int len)
{
    int tr = 1;

```

```

uint16_t crc = 0, crcget = 0;

crcget = p[len-2];
crcget = crcget<<8;
crcget |= p[len-1];

crc = crc16(crc, p+4, len-6);

tr = tr &&(crc == crcget);
tr = tr && ((p[0] == 0x55)&&(p[1] == 0x55)&&(p[2] == 0x4d)&&(p[3] == 0x58));

return tr;
}

/**
 * crc16 - compute the CRC-16 for the data buffer
 * @crc:    previous CRC value
 * @buffer: data pointer
 * @len:    number of bytes in the buffer
 *
 * Returns the updated CRC value.
 */
unsigned short crc16(unsigned short crc, unsigned char const *buffer, int len)
{
    crc = 0;
    while(len--)
        crc = crc16_byte(crc, *buffer++);
}

```

```
    return crc;  
}
```

```
unsigned short  crc16_byte(unsigned short crc, const unsigned char data)  
{  
    return(crc >> 8) ^ crc16_table[(crc ^ data) & 0xff];  
}
```