Architecture for the T-COM Network Software

# Condor Gao

Contents

[Condor Gao 1](#_Toc11762817)

[Introduction 3](#_Toc11762818)

[Figure 1: Block Diagram Showing Major Communication Components in End-to-End System 3](#_Toc11762882)

# Introduction

The goal for T-COM network is to achieve a back to back connection from the device through the internet and to the cloud server. The data will be store in the database runs by the server and waiting for the further analysis.



Figure 1: Block Diagram Showing Major Communication Components in End-to-End System

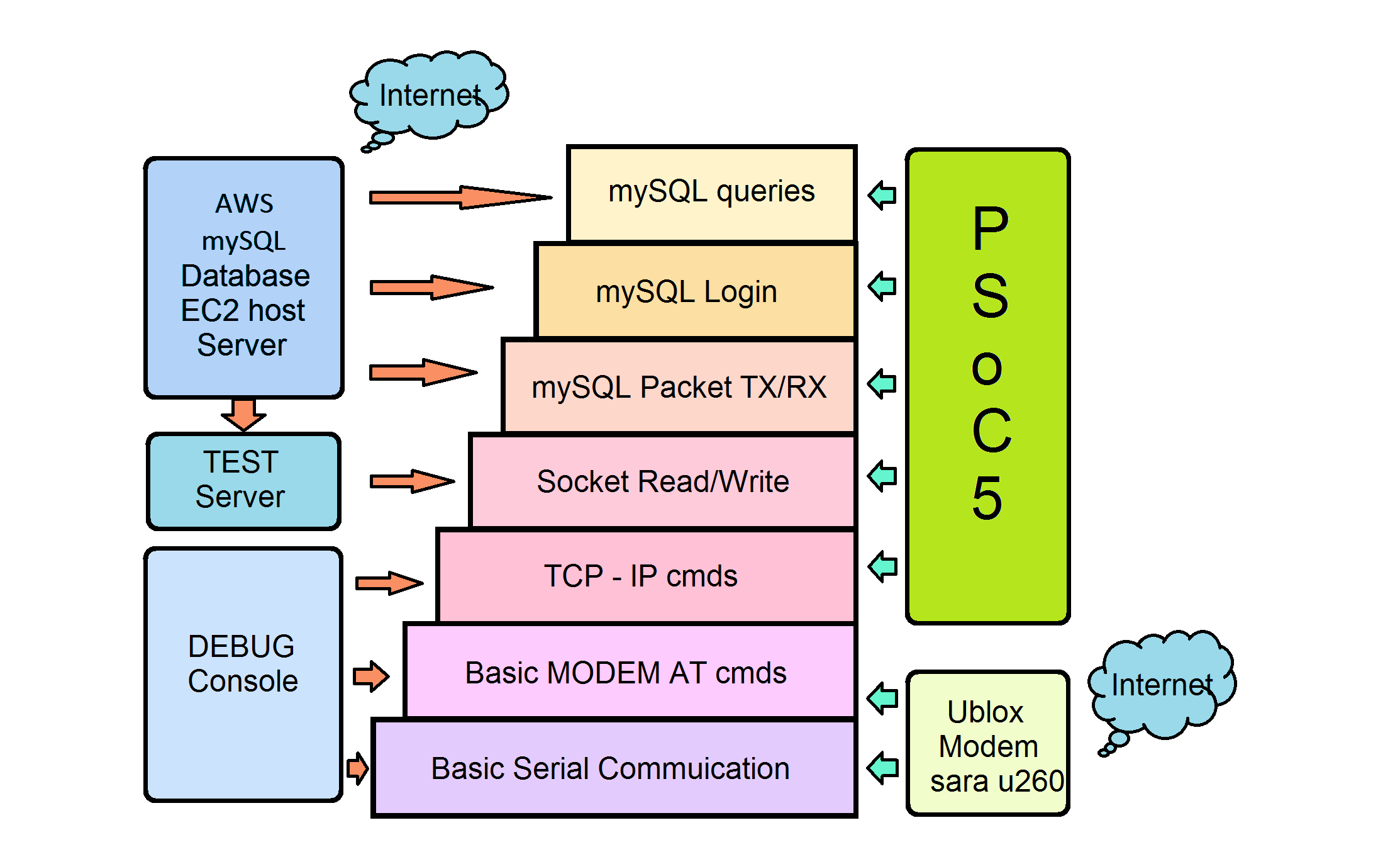


Figure 2: Wireless Cloud Communication Stack

# PHY/Datalink Layer

Serial communication with SARA MODEM

The PHY layer, from the standpoint of the MCU, involves serial data and handshake signals: TXD, RXD, RTS, CTS. Commands are based on an “AT” protocol, usually, command uses the syntax like:

AT+<cmd>=<values>

# Network/Session Layer

## Transport Control Protocol (TCP)

We need to emulate the socket library API calls, socket(), connect(), write(), and read().

### socket()

Create a socket:

AT+USOCR=<protocol>,<local\_port> /r

Implemented using Ublox SARA U2 on T-COM as:

// PDP = Packet Data Protocol

// PSD = Packet Switched Data

send\_at\_cmd\_txt( "CFUN", "1" ); // Set Module Functionality

// 1 (factory-programmed value): sets the MT to full functionality

send\_at\_cmd\_txt( "UPSDA", "0,3" ); // Packet Switched Data Action

// PSD profile ID 0, 3: activate; it activates a PDP context with

// the specified profile, using the current parameters

send\_at\_cmd\_txt( "USOCR", "6" ); // Create Socket, 6: TCP, 17: UDP

### connect()

Connect to a socket:

sprintf( cmd\_str, "0,\"%d.%d.%d.%d\",%d", a,b,c,d,port );

// put cmd in one string<socket\_id>,"<ip\_addr>",<port>

send\_at\_cmd\_txt( "USOCO", cmd\_str); // Connect Socket to the server on a specific port

Note that <socket\_id> is provided in response to AT+USOCR. And also when the socket is in use, the next USOCR will atomically create the socket using next slot (SARAu260 provide up to 7 slots).

### write()

#### Write socket data:

send\_at\_cmd\_txt( "USOWR", "0,12,\"Hello World!\"" ); // params: <socket\_id>,<num\_bytes>,"<data>"

### read()

The Arduino Ethernet library has a call:

client->read();

which gets one byte from the socket stream.

It would be inefficient for us to issue AT+USORD for each byte. So, for example, to implement MySQL\_Packet::read\_packet()

Read socket data:

### AT+USORD=<socket>,<length>

Response:

### +USORD: <socket>,<length>,<data in the ASCII [0x00,0xFF] range>

### OK

Example:

AT+USORD=3,16

+USORD: 3,16,"16 bytes of data"

OK

# Glue Layer

int

wait\_for\_substr\_or\_error( char substr[], int timeout )

{

// If we encounter ERROR or timeout, then return a value of -1

// If we succeed, return value of 0

int substr\_len;

int error\_len;

int substr\_match\_idx;

int start\_time;

int time\_elapsed;

char err\_str = "ERROR";

int err\_idx;

uint8\_t the\_char;

char debug\_str[256];

start\_time = millis();

substr\_len = strlen( substr );

sprintf( debug\_str, "start\_time=%d, timeout=%d, waiting for %d OKs\n", start\_time, timeout, n );

UART\_DEBUG\_PutString( debug\_str );

do

{

time\_elapsed = millis() - start\_time;

while (UART\_GetRxBufferSize() == 0) {} // wait until at least 1 char in UART buffer

the\_char = UART\_ReadRxData();

UART\_DEBUG\_PutChar( the\_char );

if (the\_char == substr[substr\_match\_idx])

++substr\_match\_idx;

else

substr\_match\_idx = 0;

if (the\_char == err\_str[err\_idx])

++err\_idx;

else

err\_idx = 0;

} while (substr\_match\_idx < substr\_len && time\_elapsed < timeout && !error\_flag);

if (time\_elapsed >= timeout)

{

UART\_DEBUG\_PutString( "TIMEOUT!!!\n" );

return -1;

}

else return 0;

}

void

parse\_usord\_resp( int req\_len, int \*resp\_len, uint8\_t buffer[] )

{

// Wait for the string:

// +USORD:

// Wait for:

// <socket\_num>,

// Wait for:

// <resp\_length>,"

// Wait for:

// <# of bytes = resp\_len>

// Wait for "

// Wait for OK

}

void

wait\_until\_at\_least\_n\_bytes\_in\_stream( int n )

{

do

{

// Issue:

// AT+USORD=<socket>,0

//

// Parse response:

// get <# of bytes = resp\_len>

} while (resp\_len < n);

void MySQL\_Packet::read\_packet() {

byte local[4];

wait\_until\_at\_least\_n\_bytes\_in\_stream( 4 );

issue\_usord( 4, local ); // Get 4 bytes from the stream in local

}

// Get packet length

packet\_len = local[0];

packet\_len += (local[1] << 8);

packet\_len += ((uint32\_t)local[2] << 16);

bytes\_received = 0;

do

{

req\_len = min(MAX\_PKT\_LEN, packet\_len);

wait\_until\_at\_least\_n\_bytes\_in\_stream( req\_len );

issue\_usord( req\_len, buffer + bytes\_received ); // Get 4 bytes from the stream in local

bytes\_received += req\_len;

packet\_len -= req\_len;

} while (packet\_len > 0);

# Database Access Layer

Describe MySQL API

Give examples of key subroutines.