Chabu

The channel bundle

About this document

The original GDoc:
https://docs.google.com/document/d/1Wqa8rDi0QYcqcf0oecD8GW53nMVXj3ZFSmcF81zAa8g/edit?usp=sharing

Abstract

This is a protocol to run on top of a protocol like TCP. Chabu would be in the OSI-model in layer 6, the presentation layer.

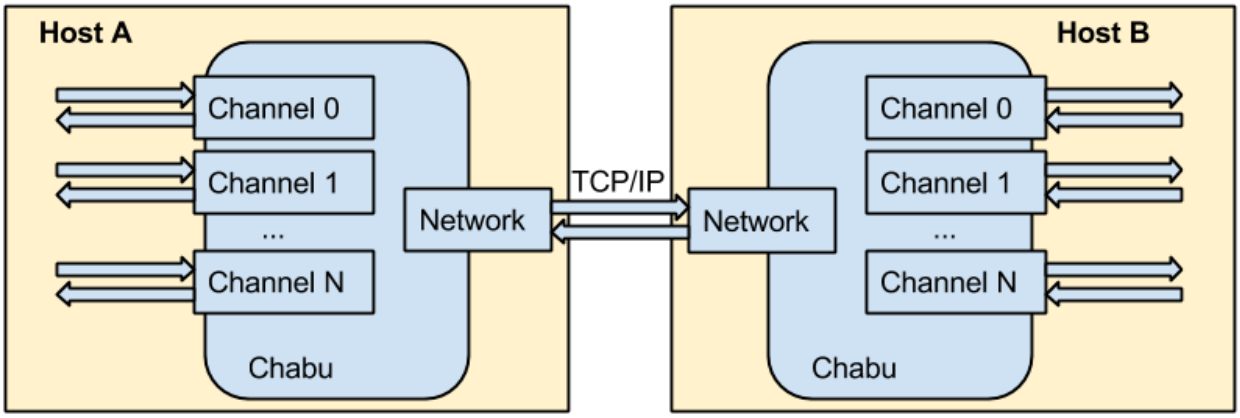


Image source

Features

Implement multiple streams in parallel

Have flow control for each stream

A channel is like a tcp connection, having a single «a to b» and a «b to a» stream.

Chabu multiplexes several channels through a single channel connection to the other host. Each channel has its own «a to b» and «b to a» stream with its own flow control. That means, if the receiver does not consume a streams data, the other streams are not blocked.
There are up to 64k channels.

What it does not do

Does not handle packet loss
Does not handle data corruption

Global Parameters

There are some definitions that must be the same at both communication partner sides. Those parameters are exchanged at communication start. If there is a incompatible mismatch, the connection will be aborted.

Protocol version

The current value is «1».

Version	What changed
1	Initial version

Byte order

The default is «big endian», known as network byte order.
The byte order can be defined for the driver at compile time.
Note: In this document, big endian is used for the examples.

Maximum Payload Size

The default is 1400.
Valid values are 1 .. 0xFFFF.
This influences the amount of memory needed to be held in transmitter and receiver.

Coding

Integral types

This protocol makes use of unsigned integer types: UINT8 and UINT16.

Blocks

Block are the parts transmitted over the tcp connection.

Structure

Type	Name	Meaning
UINT32	HEAD	channel id
UINT32	PS	payload size
UINT32	SEQ	source send sequence number
UINT32	ARM	source receive arm number
UINT8 ...	Payload	The raw bytes to be transferred

CID - Channel Id
0 .. 255

PS - Payload size
The payload is the number of bytes contained in the block. The next block starts immediately after the payload.

SEQ - Sequence block number
The absolute index the block. The first block starts with 0.
If PS is 0, the SEQ is not used.
If SEQ reaches UINT16_MAX it wraps around to 0.

ARM - Arming block number
At startup, before the first ARM is received, it is initialized with UINT16_MAX.
The absolute index, up to what index the opposite endpoint can send blocks with ISEQ.
E.g. if ARM is 10, the sender can send up to block with SEQ=10.

Start up

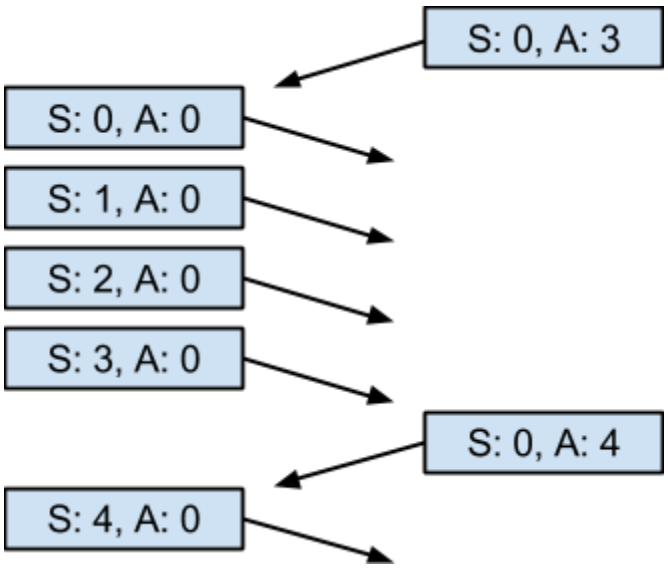
Protocol parameter specification
In the beginning both endpoints send the protocol parameter specification and compare to the values received from the opposite endpoint.
If there is a mismatch, the connection is aborted.

Type	Name	Meaning
UINT8	Version	Protocol version
UINT8	BO	Byte order: 0 = little endian, 1 = big endian
UINT16	MPS	Maximum Payload Size
UINT8	MCID	Maximum channel id

Reading is done in 2 steps. First the «Version» is read, because depending from this, the amount of bytes to follow depends.

Data handling
In the beginning each endpoint has ARM = UINT16_MAX, so no endpoint is allowed to send payload.
When an endpoint is ready to receive data, it sends a block with PS = 0 and a ARM value to signal up to how many blocks it is ready to receive.

Flow Control



Chabu integration

The under laying TCP layer

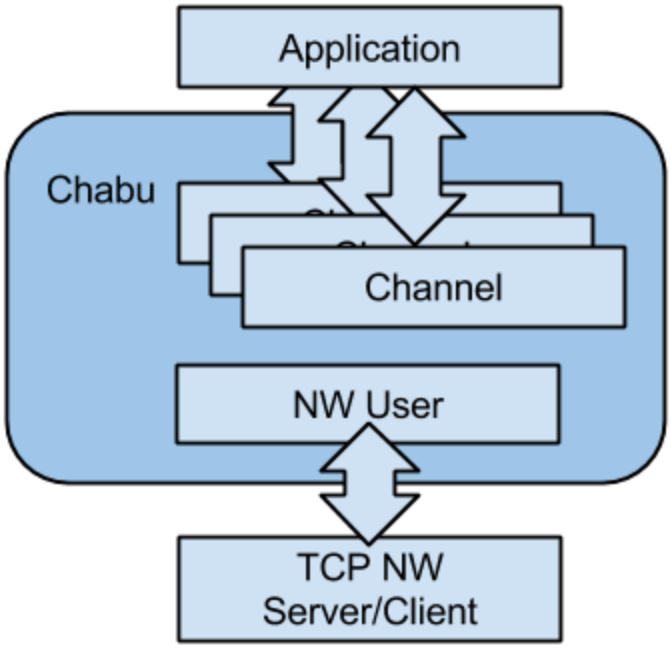
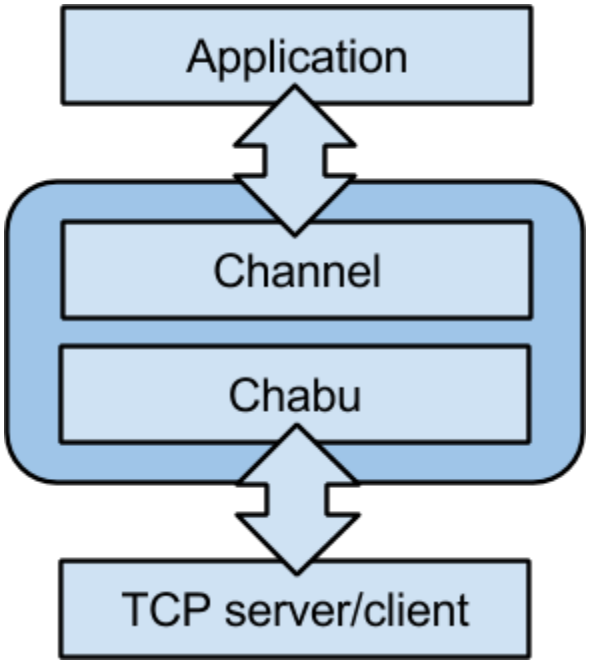


Image source



The interface

The interface between network to chabu and chabu-channel to application is the same.
The lower level implement the `INetwork` interface, the upper layer the `INetworkUser` interface.

INetwork

The network is calling the user to transfer date to be receive or to be transmitted.
But it provides methods to be notified for a user request. These methods can be called from any thread.

Function	
`void evUserRecvRequest()`	The network user want to be called to receive. This is used if not all data available was processed in the last evRecv() call, or if the user just want to be run in the network context.
`void evUserXmitRequest()`	The network user wants to transmit data.

INetworkUser

These methods are calswled by the network from the networking thread.
They shall not block.

Function	
`void evRecv( ByteBuffer )`	Called to provide received data for processing. The buffer may contain only a piece of the received data and it must be consumed completely until the later data will be available.
`boolean evXmit( ByteBuffer )`	Called by the network to request data for transmission. The result boolean tells if a flushing should be done.

The network interface

Configuration Example

FTP like service

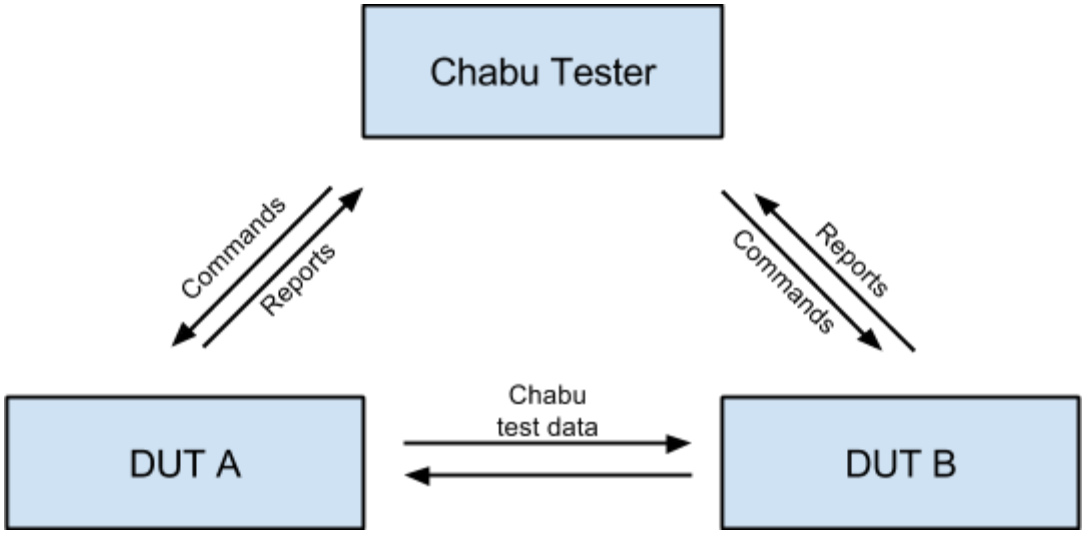
FTP uses 2 TCP connection. One for the control commands and one for the data transfer. With Chabu, this can be just like 2 channels over a single TCP connection.

Channel 0: used for the control commands. Those are expected to be very short always. So length of 1 receive buffer on both side seems to be OK.
Channel 1: used for data transfer. Here the bandwidth shall be high. To avoid time loss because of round trip time to receive the next ARM, the count of rx buffers is higher. E.g. 10.

Test suite

Three applications communicate with each other.

1. Chabu Tester that tells the DUTs what to do and collect the report data.
   As start parameter it needs the 2 ports DUT A+B are listening for the control connection.
2. DUTs to send and receive over a chabu connection.



The DUTs can be either be part of the same Java VM as the Chabu Tester or it can external processes (Java, C#, C) started as executables. Those might be started by the Tester or are already running (e.g. debugging).

Encoding

All commands and reports have a leading length as UINT32.
All commands and reports start with theirs command/report ID, which is a UINT8.

These parameter types are used in the commands and reports:

Type	Name	Meaning
UINT8	CMD RES	Command Result
UINT8	VERSION	
UINT64	TIME SCHED_TIME	Time in ns, e.g. System.nanoTime() This is not a absolute time, but a constantly increasing one.
UINT16	CID	Channel
UINT32	TX_CNT RX_CNT RX_BUF_CNT	Number of bytes to transfer or consume
UINT16+UINT8[]	MSG	String with leading length, UTF-8 encoded

Commands

Command: Time broadcast

Time broadcast, T is the current time when command is generated. This command will be produced regularly, so the receiver can implement a synchronization.

1. CMD = 0
2. TIME

Command: DUT connect

1. CMD = 1
2. SCHED_TIME

Command: DUT disconnect

Close application

1. CMD = 2
2. SCHED_TIME

Command: Close Application

Close application

1. CMD = 3
2. SCHED_TIME

Command: Connect

Start Connection

1. CMD = 4
2. SCHED_TIME
3. MCID
4. MPS

Command: Await connection

Await connection, act like server, waiting for connection

1. CMD = 5
2. SCHED_TIME

Command: Close Connection

Close connection

1. CMD = 6
2. SCHED_TIME

Command: Setup Add channel

Add channel

1. CMD = 7
2. SCHED_TIME
3. RX_BUF_CNT

Command: Setup Activate

Create the Chabu instance and set global parameters

1. CMD = 8
2. SCHED_TIME
3. BYTE_ORDER: true=BE
4. MAX_PAYLOAD

Command: Channel action

Action

1. CMD = 9
2. SCHED_TIME
3. CID
4. RX_CNT
5. TX_CNT

Command: Channel create stats

Action

1. CMD = 10
2. SCHED_TIME
3. CID

Results

Result: Protocol version

Set the version, that the DUT understands. So the Tester can adjust the commands accordingly or abort the connection.

1. RES = 0
2. VERSION

Result: Error

Whenever a problem occurs, this record is generated.

1. RES = 1
2. ERROR_CODE
3. MESSAGE

Result: Channel stats

All 100ms, for all channels that had rx/tx event, on result record is produced.

1. RES = 2
2. TIME
3. CID
4. RX_CNT
5. TX_CNT

Test procedure

The master can either run with a predefined plan or with a random schedule.
A result file is written. It contains the values as CVS and can be examined in Excel