ACAVs politischer und gesellschaftlicher Blog
Artikel rund um Deutschland, Politik,
Wenden und Gesellschaft

ESP8266 TCPIP problems

Many users,included myself, complain about the reboot or the "busy" problem while using one of the ESP8266 modules.

Behaviour if CIPSEND and IPD occur at the same time

Low level problem:

If TCP data is transmitted, it can happen, that the data is divided into more than one epacket,even if the amount of data is very small (happens also for 10 bytes !)).

Then, it seems that there is a bug in the ESP8266 binary firmware, so that (in contrast to what is expected) the packets will not be joined and treated as one, but putted out on the terminal packet by packet as received.
Now, there's another bad programmed part (fortunately in the "user space"),
wich will automaticly send TCP data to the UART if it arrives, regardless of serial incoming data to be send.

The incoming serial data will then call a handler wich will crash the system.

How to avoid it, the simple way:

This workaround only functions if it is in your hand to create the TCP packets to be send to the ESP8266.
Don't try to start a TCP packet by "CIPSEND" if the received data is not complete !
To avoid this, use something special to really set an endmark to your created packet (on the PC), like <CRLR> <CRLF> or values < 0×10.
Then only start sending data by "CIPSEND" if the whole packet arrived (detected by the endmark!)
Don't assume a TCP packet is send (and arrive at the ESP8266) in one block! It will NOT be the case.

Before doing that, I was only able to send ~ 14000 packets with in total ~500kBtye of data until the system crashes.
After setting the endmark correctly and detecting it, I aborted the test after almost 500.000 packets with in total 20MByte, packets were send every 20ms

This behaviour can be tested by:
1) Sending some bytes from the PC to the module via TCP in one packet
2) If the data send in 1) arrived on the micro connected to the ESP8266,
send a large number of bytes back to the PC (not more than 2kBytes!)
3) While CIPSENDing this data, send a single byte from the PC to the ESP8266
4) This single bytes will trigger an *IPD action wich then crashes the ESP8266

How to avoid it, the "hard" way:

The solution from the preceeding paragraph will not work, if the sender will not wait but is sending packet after packet,like Webbroser do (esp. Android browser send many stuff not necessary, but there's no chance to stop that,a Firefox running on Windows in contrast will NOT show this behaviour and will work fine !)

For the following methods, you must be able to change code, compile it and download it to the ESP8266 !

The All-or-nothing solution:

1) create a volatile flag variable in a header file, and include it in all .c files necessary
2) set the var to “DONT STOP” for default in the user_main
3) set the var to “STOP” if a IPD packet arrives
4) set the var to “DONT STOP” if a IPD packet is processed and putted out
5) in the UART RX INT, test if “STOP” and if it is set, exit without further processing
6) after putting out all IPD data, send also a “SEND INT” (instead of the “Send Ok”) to show, that CIPSEND was aborted
The HOLD-ON solution:
1) create 2 volatile flag variables in a header file, and include it in all .c files necessary
1a) call them (e.g.) STATUS_CIPSEND and STATUS_IPD
2a) in the beginning of the “CIPSEND” function set STATUS_CIPSEND to TRUE
2b) shortly before posting data to the OS, set STATUS_CIPSEND to POST
2c) in the end of the “CIPSEND” function set STATUS_CIPSEND to FALSE
2d) shortly before sending data to OS, check if STATUS_IPD=SEND
2d1) -> if yes, don’t POST data and exit (without changing the state)
2d2) -> if no, POST data and exit (like before modifications)

3a) if an IPD packet arrives, set the var to STATUS_IPD=SEND
3b) if the IPD packet is putted out on the serial line, set the var to STATUS_IPD=NONE
3c) check if STATUS_CIPSEND=POST
3c1) -> if yes, post the data like it would be done in the CIPSEND function
3c1) -> if no, nothing is to do

That was a short description. For further information, download the code and see the readme file !
At least, a warning:
Don’t try to only overwrite your files with the ones downloaded from here !
You must check it and compare what’s changed in contrast to your code.
There are some specials I added, like different start and endchars for every packettype.
This files are based on the 0.92 SDK, so your code may differ from no to very much, but the problem remains the same.
Perhaps using GPIOs is a solution ?
