Aim of the document

The follwing text is intened to clarify the *Cubesat Space Protocol* (CSP).

1 CSP General Overview [1]

The CSP protocol is intented for communications between different embedded systems. However, it is mainly oriented for a **CubeSat** use.

The protocol is written in C by the *Aalborg University* along with the company *GomSpace*.

Traditionally, CubeSats used and still use a *master/slave* architecture for its internal communication. However, this protocol offers a different approach. It is based in a Service Oriented Architecture (SOA), i.e., it is based in a style where services are provided to the rest components using a protocol over a network. A SOA protocol has to be independent of the vendors, products and technologies. Therefore, it can be seen as a lighter version of the popular *TCP/IP*.

In a nutshell, the protocol aim is to bring a **light**, yet **simple** to implement among the different satellite subsystems, version of the TCP/IP.

Some advantages of this approach against the common *master/slave* way are the following:

- All susbsystems can communicate between each other.
- A master is always a critcal single point of failure. Implementing a SOA protocol avoids this.
- Abstraction, reusability and redundancy are some other key advantages.

2 Protocol Layers [2]

The protocol is composed of the next four layers:

- **Layer 1.Drivers :** CSP provides CAN, I2C and UART drivers for some platforms. It is easy to add a custom driver though. But, **what is a driver?** It is a piece of software intended to allow communication between a hardware device and some other software. Usually, the driver is written by the manufacturing company as they know how to talk to the device since they have designed it.
- **Layer 2.MAC interfaces :** Defines a frame-format that is suitable for the media.
- **Layer 3.Network Router :** This part is in charge of looking to the header of the packet which contains the delivery and the source adress. The main purpose of the router is to accept incoming packets and deliver them to the right message queue.
- **Layer 4.Transport Layer :** This layer is the one that actually delivers the message to its required destination.

3 A common topology example

A typical way to work is to create two segments, the Space Segment and the Ground Segment.

```
               I2C BUS

  -------------------------------
 /        |       |       |       \
+---+   +---+   +---+   +---+   +---+
|OBC|   |COM|   |EPS|   |PL1|   |PL2|       Nodes 0 - 7 (Space segment)
+---+   +---+   +---+   +---+   +---+
          ^
          |  Radio
          v
        +---+           +----+
        |TNC|  -------  | PC |              Nodes 8 - 15 (Ground segment)
        +---+    USB    +----+
        Node 9          Node 10
```

Other topologies can be implemented as well. Each node uses static-routing. In other words, the routing adresses are assigned durinf the power-up. However, CSP also supports assigning individual route to any node of the system during run-time. That means that the network topology can be recnofigured after being initialized.

4 Maximum Transmission Unit

MTU can be defined as the maximum size of a packet being sent without a previous fragmentation in several sub-packets.

CSP iteslf doesn't constrain this parameter as it adapts itself to the underlaying protocols using the CSP. It is important to mention that CSP adds some header to the packet.

5 Conclusions

Cubesat Space Protocol has been mainly designed for communications between several embedded subsytems inside the CubeSat. It adds a layer of abstaction between, let's say, a system that uses CAN bus protocol and another that uses UART protocol for its own communication.

CSP fits like a glove in terms of adding a TCP/IP inspired protocol aiming for simplicity and subsystem agnosticism. The most common way to use it is with a basic Space-Ground Topology. However, it allows to be reconfigurated.

References

[1] GomSpace. https://github.com/GomSpace/libcsp.

[2] KubOS. http://docs.kubos.co/0.0.1/libcsp/index.html.