TCP/IP planning

2009-10-29 16:10:46 by Nick

Before any further work can be done towards designing the COPTA module, further consideration must be given to the communication part of THAT System. Chris and I spent the day researching UDP/TCP/IP communications. We came up with a preliminary communication framework for the modules in THAT System.

Default IP setup for THAT System modules (Preliminary)

MAC addresses

- Example MAC = 02 : 54 : 48 : EA : 51 : DF
- Byte 6 : Locally administered MAC, unicast mode
- Byte 5 : “T”
- Byte 4 : “H”
- Byte 3 : Randomly generated
- Byte 2 : Randomly generated
- Byte 1 : Randomly generated

IP addresses

- Security, Safety, & Access Modules 192.168.80.xxx
- Utility Management Modules 192.168.82.xxx
- Controller & Bridge Modules 192.168.84.xxx
- Basic Output Modules 192.168.86.xxx
- Basic Input Modules 192.168.88.xxx
- Miscellaneous Modules 192.168.90.xxx

Protocol

- Transmission Control Protocol (TCP)
- Listening Port: 8428

Communication Framework

THAT-compatible Module: TCP server
Master Controller: Computer running TCP client software (Custom THAT software to be created using Python)

Communication Flow

<u>Initial System Setup:</u>

1. Module(s) listens on network (indefinitely)
2. Computer scans all IP address in the range 192.168.80.xxx → 192.168.90.xxx , and it attempts to connect to TCP port 8428 at each address.
3. Computer sends “**@CONF**”
4. Active module responds “**$THAT**” ** Hard-coded into module
5. Computer sends “**@NAME**”
6. Module responds “**$*NAME***” ** Hard-coded into module
7. Computer sends “**@ID**”
8. Module responds “**$*UNIQUE-ID***” ** Hard-coded into module
9. Computer generates a unique hash for the computer-module pair
10. Computer sends “**@*GENERATED-HASH***“
11. Module responds “**$DONE**”
12. Computer stores module IP, NAME, ID, and HASH for future connection

<u>Typical System Communication:</u>

1. Computer sends “**@START**” to known (configured) module
2. Module responds “**$*STORED-HASH***”

3. Computer checks received hash with that on file, and terminates connection if they do not match.
4. Computer sends a ’set’ command
5. Module responds with an acknowledgment reply
– OR –

4. Computer sends a ‘read’ command
5. Module responds with the requested data
6. Computer closes connection with module

** There should be a set of basic, standardized ’set’ and ‘read’ commands that all THAT modules understand. Beyond this standard set, each module will inevitably have an extended command/instruction set; and the creator of each module will have to provide documentation and/or an API for accessing the specific features of that module.