Communication Protocol over Ethernet between NIOS driven embedded applications and control system

Document History

Date	Who	Description
2006-09-06	Marek Penno	Create the Document
2006-09-07	Marek Penno	Removed Packet Header Version 1 Overworked some details of Packet Header Version 2
2007-03-23	Stefan Weisse	Adopted to Network Queue v1 RC1
2007-03-30	Stefan Weisse	Finalized to Network Queue v1 RC1
2007-08-29	Stefan Weisse	Updated to Network Queue v1 RC2

Requirements

There exist several applications at DESY, which are using the NIOS2 Processor and need to communicate with the control system DOOCS. These applications are for example the Heavy-Mover, Neutron-Detector, Gamma-Detector and XFEL Interlock. Together, they have the need of two different communication methods to the control system:

- request and reply method – for each request, a single reply is send
- trap or signalling method – signalling (i.e. data can be send without a request)

Additionally, following requirements exist:

- optional, simple authentication of requests
- predefined set of data types
- ability to 'switch off' the network layer of the protocol stack and do direct inter-application communication

Using UDP Protocol

UDP will be used as network transport layer. The reasons are:

- avoid TCP overhead
- failure-proof (prototype tested) in case of irregularities at network, stack or peer side
- simple to implement with look on embedded systems (e.g. Nios2)
- proven to be working stable on Nios2 system

If demands are heavy, the new protocol was designed in a way that it can be easily adapted to TCP socket transport.

Packet Format

Each packet is structured in four parts:

- **General Packet Header**
  - contains information about packet version and size
  - with the help of the general packet header, later, older clients are able to handle also unknown, newer packets, and simply skip them without blow-up in confusion
- **Specific Packet Header**
  - contains information about data format, authenticity and more
- **Packet Data**
  - contains the payload data of the packet, e.g. DOOCS property values or control data
- **Packet Tail**
  - marks the end of the packet

Important Remark: Each data field in the packet is encoded in **Little Endian** format where applicable.

General Packet Header

The General Packet Header is needed to have a common understanding about the packet data received and transmitted. It contains general information like packet size and header version. This minimal information is needed by the packet processing to e.g. skip unknown packet types.

If the header version is known, the dedicated packet layout (Specific Packet Header) gets interpreted.

- Packet Magic, 16 Bit unique ID, same for all future Packet Versions, marks the beginning of a new Packet, agreed was magic 0xA51C
- Packet Size, 16 Bit
- Packet Version, 16 Bit
- Packet Special Bits, 16 Bit, e.g. marking for 'packet want acknowledge' or 'packet is acknowledge packet'

Specific Packet Header Version 2

The Specific Packet Header Version 2 was chosen to transport the payload from local to remote peer. The possible appended data is carried in a list of Variables / Properties. This header type consists about following elements:

- **Authorisation ID**, 32 Bit
  - used, to determine the requestors access privilege
  - if NULL, packet has 'anonymous' access privilege or no authorisation is used
- **Packet Index**, 16 Bit
  - This value increased for every packet send to one host. It can be used to detect lost packets and fill some statistic information
- **Request ID**, 16 Bit
  - This field is filled with a non-null value by the sender and is f.ex. increased for every request packet. The reply packet on the request must contain the same request ID. For trap packets this field is leaved NULL
- **Subsystem or Device ID**, 16Bit
  - Tells the receiver, which subsystem of the embedded system gets addressed
  - The Subsystem ID is used as a prefix for properties
- **Num of Datablocks**, 16 Bit
  - Contains the number of Datablocks, that are carried within the Packet data
- **Physical Packet Type**, 8 Bit
  - Available physical packet types are request, response and telegram
- **Logical Packet Type**, 8 Bit
  - Available logical packet types are get, set (also known as put) and call for requests and responses
  - Available logical packet types are update and trap for telegrams
- **Packet Type Param 1**
  - See proposal further down

Remarks to Specific Packet Header Version 2:

- The fields Physical and Logical packet type must be set by the user to appropriate values and are interpreted internally in source code.
- The fields Request ID, Packet Index and Num of Datablocks are used internally and can not be set freely by the user code.
- The other fields (Subsystem/Device ID, Packet Type Param1 and Authorization ID) are optional. Optional does **not** mean that they are not encoded in any packet. Optional is a marker for the user that one can freely use these fields for ones own demands.

Packet Data for Specific Packet Header 2

The Packet Data consist about a list of variables / properties. Each variable is represented by a block which contains following fields:

- total Length of the block in bytes, 16 Bit
- variable ID, 16 Bit → a substitution for the variable name
- data type of the variable, 16 Bit
- number of elements of data type of the variable, 16 Bit
- data itself, encoded in **Little Endian** format and padded to 32bit value

The variable ID on the side of the embedded system is decoded by a simple constant.
On the side of the meta server, a file gives information about which variable ID and subsystem ID correspond to which DOOCS property. For any data type (by type this is naturally the case for binary and text arrays) there can be N elements transferred as 1-dimensional array.

Any data is padded to full four bytes! This implementation detail was chosen by performance reasons. This means that for one element of an 8 Bit wide data type transferred, 3 Bytes are totally wasted!
Also strings and char arrays are padded to four bytes by performance reasons.

At the moment, the following data types are supported:

Data type	**ID# (hex)**	**type space req.**	**used packet space**
binary array (char array)	0x01	>=1	>=4 (1 per element)
Single Float value	0x02	4	4 per element
Single Double value	0x03	8	8 per element
Single 8 Bit signed integer	0x04	1	>=4 (1 per element)
Single 8 Bit unsigned integer	0x05	1	>=4 (1 per element)
Single 16 Bit signed integer	0x06	2	>=4 (2 per element)
Single 16 Bit unsigned integer	0x07	2	>=4 (2 per element)
Single 32 Bit signed integer	0x08	4	4 per element
Single 32 Bit unsigned integer	0x09	4	4 per element
Text array	0x0A	>=2	>=4 (1 per element)

Packet Tail

The Packet Tail signals the end of the packet. It simply consists of a single constant 16 bit value. It was agreed to use the value 0xC15A.

Proposal: Physical and Logical Packet Type Structuring

The idea of the structuring of the 16 bit packet type into physical and logical type is to define sets of meaningful calls. The physical handling, where applicable, can be (and was) done internally inside the Network queue fully transparent to the user.

- Higher Byte defines the physical Packet Type, Lower Byte defines the Logical Packet Type
  - available physical packet types:

    - '**Request**', will cause a response packet logical packet types:
      - 'get' <property(s)>, causes response 'get property'
      - 'set' <property(s)>, causes response 'set property'
      - 'call' <parameter(s)>, causes response 'call'

    - '**Response**', is send as a effect of a request packet logical packet types:
      - 'get property' answer, <property list filled>, error code
      - 'set property' answer, error code
      - 'call' <return parameters>
        - RPC like technique, packet carries a set of variables as parameters, including Function ID

    - '**Telegram**', is sent autonomous and asynchronous logical packet types:
      - 'Trap', signals about a event that has arise, Trap ID is put into Packet Type Param1
      - 'Update', to signal that several properties have updated, contains list of updated property(s)

Proposal: Packet Type Parameters

The idea is to use the ***Packet Type Param 1*** for encoding of user message types.

For telegram + trap type, this field may contain a trap message id.

For response packets, this could be a result or error code of the request that was issued.

- For Packets of Type 'Telegram, Trap', field contains the Trap-ID
  - 0x0000 – 0x0FFF – reserved System Traps
    - 0x0000 – System start up
    - 0x0001 – System shut down
    - 0x0002 – System entered fault state
    - 0x0003 – System leaved fault state
    - 0x0004 – Authentication failed
    - 0x0100 – Update DOOCS Properties
  - 0x1000 – 0xFFFF – free for user defined trap codes

- For Packets of Type 'response'
  - 0x0000 - no error
  - Protocol Errors 0x100-0x1FF
    - 0x0101 - data type not supported
    - 0x0102 – packet data corrupted
  - Server Error 0x200-0x2FF
    - 0x0200 – requested operation not available
    - 0x0201 – requested data not available
    - 0x0202 - server busy, request dropped
    - 0x0203 – internal error
  - Access Error 0x0300-0x03FF
    - 0x0300 – access denied
    - 0x0301 – data is read only
  - 0x1000 - 0xFFFF – free for user defined error codes