Embedded Software lab, RaspNet Protocol-defintion

Stefan Naumann

March 31, 2017

Contents

1 Preface

A protocol in computer communications is a set of rules for every participant of the communication. It ensures, that data sent by the sender can be received and properly computed on the receiving end.

The ISO/OSI layering model depicts several layers of protocols, which are needed to pack and unpack the data within the internet-context. There are for example TCP, IP, Ethernet or UDP. The separation of functionality into layers is important for better interoperability between different implementations. It enables the developer to implement one layer at a time and test smaller parts, rather than having to implement the whole architecture at once.

User-applications, for example a web-browser, which encapsulated its data in HTTP, are represented by the the top layer of the ISO/OSI-model. This data-stream will be handed over to the TCP-layer, which adds some information needed for identifying the package. It is also responsible for resending, if the package did not reach the receiver. At the IP-layer the header with receiver- and sender-address is added and the package is handed over to the data link layer. This one adds the physical MAC-address before the physical layer sends the data as single bits over the wire.

Table 1: ISO/OSI-layering model (see [2])

Layer	**Function**
7. Application	High-level API, incl. Resource Sharing, remote file access, directory service, virtual terminals
6. Presentation	Translation between network service and application; character encoding, data compression and en-/decryption
5. Session	Managing session, i.e. exchange of information back-and-forth
4. Transport	Reliable transmission of data segments between points in the network; segmentation, acknowledgement, multiplexing
3. Network	Structuring and managing a multi-node network; addressing, routing, traffic control
2. Data Link	Reliable transmission of data frames between two connected nodes
1. Physical	Sending and receiving raw bit streams over the physical medium

This document describes a simple communication protocol between Raspberry Pi computers with the Gertboard-extension board, by using its ATMEL-processor for the communication. The protocol uses a unidirectional logical token-ring infrastructure and it uses four ports of the ATMEL-processor for communication.

The protocol clarifies the handling of addresses and basic reliability aspects. It also implements a broadcast-mechanism. There is no concept of combining packages to a bigger message, if needed the user has to implement that on top of the described stack.

2 Basic structure

RaspNet contains of several layers of protocols. Frames are send serialized over one wire. The second wire is used as clock-signal. Therefore there are two wires used between two subsequent nodes in the ring (see Figure 1), which can send up to 260 bytes (255 payload + 4 CRC + 1 size) per frame.

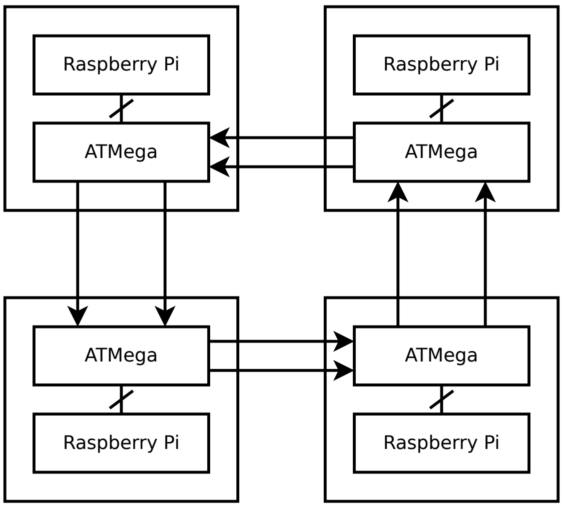

Figure 1: Architecture of the logical token ring network with four nodes

The following sections describe the layers of the protocol. The numbering of the layers is oriented to the ISO/OSI-model.

2.1 Layer 1 - Physical Layer

The lowest layer describes how bits are sent over the wires. There are two wires used from one node to the next. The first is used as data-pin, the second as clock-signal. Whenever the clock-signal changes (from 1 to 0 or from 0 to 1) there is a new bit there for reading on the data-pin. The data-signal may change in the middle of two clock-changes, so there is a phase shift between data and clock-signal.

A HIGH-signal level on the data-pin identifies a logical 1, a LOW-signal level identifies a logical 0 when reading the signal at the data-pin. The clock-signal needs to be toggled even if there are no data-bits to send. In this case the data-signal needs to be pulled towards a logical 0.

Figure 2 depicts the timing-behaviour for receivable data of '01' on time 2 and 4.

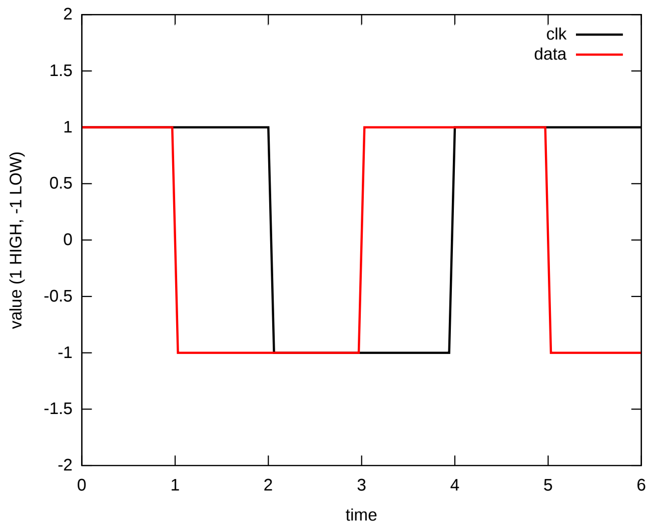

Figure 2: Timing behaviour of layer 1

2.2 Layer 2 - Data Link Layer

Using the service of layer 1 to send bits, layer 2 bundles them together to send network-frames. A network-frame begins with a preamble of '01111110' (a zero, six ones, a zero). Whenever the receiver reads this pattern, when not reading the payload of another package, a new network frame begins.

Figure 3 shows the structure of the header for layer 2 (lengths are given in bit).

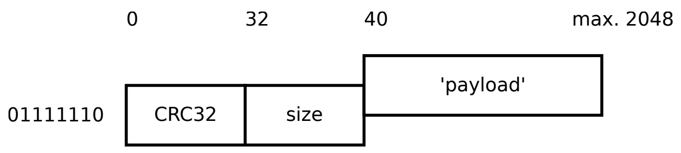

Figure 3: Basic structure of the frame

The header-fields serve the following purposes:

1. **CRC32** *32 Bit.* A 32 bit long CRC32-checksum over the payload only. Must be checked when receiving a package, if it's wrong the receiver discards the frame.
2. **Size** *8 Bit.* Size of the payload-data in Byte (8 Bit).

Figure 4 shows an automaton with the different states of the layer 2 protocol. From the starting-state (idle), there are two possibilities, either the layer 2-implementation receives data from the RPi, then it will calculate the checksum and send the frame (using layer 1), or it receives data from layer 1, i.e. from another node in the network.

Then it will read the header and *size* bytes of payload. It will calculate the CRC-value of the payload and compare the calculated value with the previously read one. If they match, the payload will be transferred to layer 4, if not it will discard the frame. Both actions will lead to the idle-state.

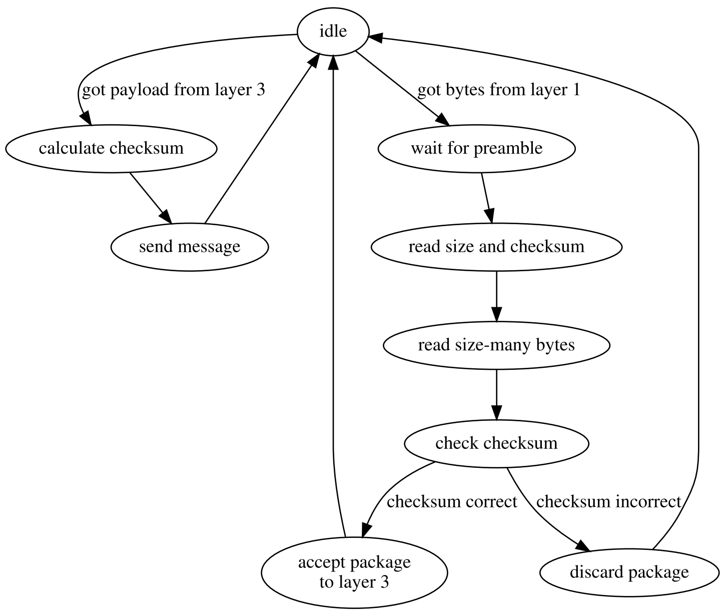

Figure 4: Protocol automaton for a layer 2 implementation

2.2.1 Securing the transmission - CRC32

For securing the transmission a CRC32-procedure (Cyclic Redundancy Check) is used. This algorithm is based on polynomial division with remainder. The polynomial for CRC32 [1] [3] is

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$$

This is transferred into the bit-representation: `1 0000 0100 1100 0001 0001 1101 1011 0111`. This word is more than 32 bit long. The data-stream will be divided by this generator-polynomial in a mod(2)-ring. The remainder will be your CRC32-value.

For checking the correct transmission one calculates the remainder of the received package and checks it with the received CRC32-value. If they match there is either no error in transmission or there is a unlikely error, which cannot be found by the CRC32-method.

Calculating the remainder may be done like long division. Take the numerator (dividend) and add 32 bit as remainder to the bitstream. Put the denominator (divisor) under it, with the first '1' of both numbers lining up. Do a bitwise-XOR-operation on both numbers. If the numerator is larger than the denominator you only need to XOR the present bits. Do this recursively until the bits in the original bitstream are all zero, with only the added 32 bits having ones in them.

Given the payload `0x74 65` the following CRC8-calculation may be done. The generator polynomial of CRC8 is: $x^8 + x^5 + x^4 + 1$ (`1 0011 0001`). This calculation is based on adding n (the degree of the polynomial) bits to the payload, which serve as remainder. In the case of CRC8 we added 8 bits as remainder.

```
     0x74      0x65
  0111 0100 0110 0101
G  100 1100 01
   011 1000 001
G   10 0110 001
    01 1110 0000
G    1 0011 0001
     0 1101 0001 0
G      1001 1000 1
       0100 1001 11
G       100 1100 01
        000 0101 1001 | 00
G            100 1100 | 01
             001 0101 | 0100
G              1 0011 | 0001
               0 0110 | 0101 00
G                 100 | 1100 01
                  010 | 1001 010
G                  10 | 0110 001
                   00 | 1111 0110
                           0xf6
```

Therefore CRC-8 remainder of the input data `0x74 65` is `0xf6`.

A valid CRC-32 remainder for `0x74 65 73 74` is `0x0b 70 ed 28`. Keep in mind, that a CRC-value is always as long as the degree of the generator-polynomial (therefore in CRC-32 32 bit).

2.3 Layer 3 - Network Layer

The third layer handles addressing and routing of packets. Based on the correct layer2-network-frames layer 3 handles addressing, therefore ensuring, that packets are retransmitted through the ring if the packet needs to reach other recipient(s) or handing the packet to layer 4. Figure 5 shows the layout of the header of layer 3 (lengths are given in Bytes).
The header-fields serve the following purposes:

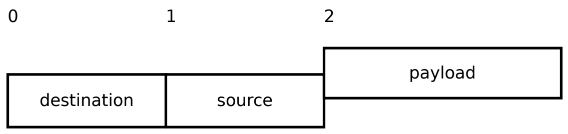

Figure 5: Basic structure of the layer3-packet

1. **destination** *1 Byte.* The destination address of the package.
2. **source** *1 Byte.* Source address of the package.

Addresses An address is a number between 1 and 255 ($2^8 - 1$). Every address has to be unique in the network. The address zero (0x00) is used as broadcast-address. The addresses are assigned statically at start-up of the system.

The following cases may occur during execution:

Whenever a layer 3 implementation receives a package, it checks the destination address. If the address is equal to the address gotten during initialization, the packet will be handed over to layer 4.

Reading the broadcast-address indicates that every node in the token ring needs to receive the package. There are no ACK-messages for broadcast packages. At the point, where the sender receives its own package, it knows that every addressee has received the message properly and notifies layer 4 of the successful sending operation. If the package is lost or corrupted on the way through the ring, layer 4 needs to care about resending or ignoring the fault. Every other node needs to relay the package through the ring.

Whenever the layer 3 implementation receives a message with its own address or the broadcast-address set as source-address, it will discard the package.

Due to the organization of the header it may be possible to stream the packets through the ATMegas without buffering them in whole. A CRC-check is not performed at the intermediate-nodes between the sender and the receiver. The frame is received and read until the destination-address. If the destination differs from the own address, the ATMega can begin relaying the read Bytes to the next node, while still receiving the remaining ones. An communication-error can be detected on the receivers end. This behaviour is also possible for broadcast-message, but every node has to check the destination address. If the receiving node sent the broadcast message relaying must not be done.

Priorities Packages that need to be relayed should be handled with a higher priority than own sending-wishes of the node. Sending-wishes should be handled in order of their arrival. This ensures, that own sending-wishes may not block the ring and congestions should not occur. If the node has no time to prepare its own sending-wishes, it cannot generate additional load onto the network. Also prioritizing packages from other nodes helps them holding their time-constraints.

2.4 Layer 4 - Transport Layer

The layer 3-packets are used for transporting packages in layer 4. The package contains an identification number and some flags. Figure 6 shows the layer 4 header (lengths are given in Byte).

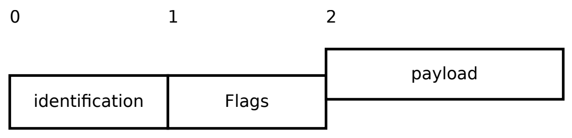

Figure 6: Basic structure of the layer 4-header

The header-fields serve the following purposes:

1. **identification** *1 Byte.* Identification number for that package.
2. **flags** *1 Byte.* Flags for signalling.
   a) **0x01 – ACK**. Acknowledge. Indicates to the sender that the receiver got the package.
   b) **0x02 – DGRAM**. Message is datagram. Do not acknowledge the message, it will not be resend if lost or corrupted.
   c) **0xfc**. Reserved for future use.

Layer 4 provides the functionality to detect corrupt or lost messages via the means of a time-out. It therefore can ensure, that a package is resend if needed. The acknowledgement and resend-functionality can be disabled by setting the DGRAM-flag, so the application can decide whether a resend is necessary.

Not trying to resend lost messages may be useful when the message is a snapshot of a value, for example a sensor-value that is read in a cyclic manner. Not receiving a value may be better than trying to resend an old value, which block the communication channel for other sensors.

Identification The identification number will be increased with every sent message at the senders side. Therefore after 256 sent messages identification numbers start to get reused. The sender needs to save sent packages until an ACK-package was received if the package is not a datagram. The receiver needs to save the identification-number of the last received package for each sender, because it may be possible that a sender resends its package due to an ACK-package taking to long to reach the sender. Layer 4 does not guarantee, that a message is received only once, nor that it reaches its destination at all.

Acknowledgement The ACK-package has a set ACK-Flag, the identification-number of the original package and does not have a layer 4-payload. After sending a package, a timer is started. If the sender receives an ACK-message within a specific amount of time (set at startup), the message can be deleted off the ATMega; if not it will be resent. The sending ATMega indicates an error to the sending Pi after five unsuccessful tries.

The amount of time for waiting after the last bit is transmitted will be received on initialization from the Raspberry Pi. A good value might be:

$$t_w = n \cdot l_1 \cdot Wc(Layer1) \cdot 2$$

with t_w time for waiting; n number of nodes in the network; l_1 the amount of latency on the cable for the longest possible message and $Wc(Layer1)$ the worst-case time for layer 1 to send and receive the longest possible frame. It must be multiplied with two to consider the latency of the ACK-messages and possible waiting time of the package due to already begun transmissions on retransmission-side.

3 API between ATMega and RPi

For the communication between the ATMega processor and the Raspberry Pi we reuse the layer 2-implementation and use a one-Byte big payload for transmitting data (characters). This protocol therefore ensures correctness inside one byte but not whether a communication partner has received a character nor if it received the correct order.

The protocol consists of several commands, indicated by a single character, followed by its parameters. The ATMega may answer with `Y` or `N`. Due to the asynchronous nature of communicating processors, transmissions may be interrupted by a communication-wish by the other. Interrupting an ongoing transmission is not allowed. The ATMega may drop any command it has received while sending a package to the Pi.

The following formal grammar describes the commands of the serial-protocol:

```
S = ( 'A' ADDRESS | 'S' | 'D' LENGTH DATA | 'B' ADDRESS | 'P' )
ADDRESS = NUMBER NUMBER NUMBER
LENGTH = NUMBER NUMBER NUMBER

NUMBER = ( '0' | '1' | ... | '9' )
HEXNUMBER = ( NUMBER | 'a' | 'b' | 'c' | 'd' | 'e' | 'f' )
DATA = <payload-data>

-- hint: PRESCALER: the allowed values greatly depend on the timer you use on ↩
    the ATMega, here: the 8-bit timers: timer1 and timer3.
```

The following commands are allowed:

- **A** Address - give the ATMega an address for itself
- **S** Status - ask the ATMega for its status ("OK" -> it's okay to send another package)
- **D** Data - transmit payload-data of the given length [0,255] to the ATMega.
- **B** Bind - specify the target address for the next package for sending.
- **P** Push - send the package (payload-data given the bound address)

On first initialization the ATMega receives an address from the Raspberry Pi. Then messages can be transferred using the Bind, Data and then Push-commands. Every command will be acknowledged or denied by the ATMega using `Y` or `N` characters.

The ATMega indicates a received message by `R`, followed by the address of the source-node (as 3 digit string) size in bytes (as 3 digit string) and the payload:

```
S = 'R' ADDRESS LENGTH DATA
-- rest as above
```

References

[1] Wikipedia. Cyclic redundancy check - wikipedia. URL: https://en.wikipedia.org/wiki/Cyclic_redundancy_check. [date accessed: May 13, 2016].

[2] Wikipedia. Osi-modell - wikipedia. URL: https://en.wikipedia.org/wiki/OSI_model. [date accessed: May 13, 2016].

[3] Wikipedia. Zyklische redundanzprüfung - wikipedia. URL: https://de.wikipedia.org/wiki/Zyklische_Redundanzpr%C3%BCfung. [date accessed: May 13, 2016].