# BACnet over KNX

Wolfgang Granzer Wolfgang Kastner

Automation Systems Group
Institute of Automation
Vienna University of Technology

Treitlstraße 1-3, A-1040 Vienna, Austria
{w,k} @ auto.tuwien.ac.at

The Building Automation and Control Network Protocol (BACnet) has been developed to provide a solution for building automation and control systems of all sizes and types. While BACnet specifies an application model as well as different routing services, the underlying network medium is not defined. BACnet messages can, in principle, be conveyed over any network. However, a number of network types are recommended in the BACnet standard. Since KNX with all its benefits has not been considered as a network medium for BACnet, we provide an approach that uses KNX TP 1 as network medium for BACnet. Additionally, a first proof-of-concept implementation is presented in this paper.

## 1 Introduction

In [1], a standard model for all kinds of Building Automation Systems (BAS) is described. In this model, the system functionality is divided into three levels which are ordered hierarchically. At the field level, environmental data are measured and parameters of the environment are physically controlled. Automatic control is performed at the automation level whereas global configuration and managements tasks are realized at the management level.

Nowadays, the standard three level functional hierarchy model can be implemented as a flatter, two-level architecture [2]. This is for two reasons. First, so called intelligent field devices incorporate more functionality than ordinary ones. Second, information technology (IT) and its infrastructure became accepted not only at the management level, but also at the automation level. Consequently, functions of the former automation level are split being reassigned either to field devices (e.g., implementing controller functionality) or management devices (e.g., realizing process data monitoring).

As a result, today's communication networks for BAS are implemented following a two-tiered model. Smart sensors, actuators and controllers are interconnected by multiple field networks. These field networks are coupled by a backbone network which is home to management stations requiring a global view of the entire BAS. For the field level, robust, low-bandwidth and cost efficient field bus protocols are well established. For the backbone level, IP based protocols are preferred due to reasons of economy and easy integration of management stations with (office) local area networks and the Internet. A protocol that can be used at both levels is the *Building Automation and Control Network Protocol (BACnet)*.

While BACnet is more popular at the backbone level, it can also be used at the field level, where it is limited to the features of the underlying protocols. BACnet defines only the network and application layer. The underlying network medium (i.e. physical and data link layer) is not part of the standard. However, to increase the compatibility of BACnet devices, the use of different physical/data link layer combinations (called *network options*) for BACnet has been standardized. Up to now, KNX TP1 with all its benefits (e.g., CSMA/CA, link-powered devices, free topology, small network stack footprint for end-devices and routers) has not been considered as a network option for BACnet. For this reason, we provide a solution called *BACnet over KNX (BACnet/KNX)*.

The paper starts with an introduction into BACnet. Next, a possible solution to use KNX as network option for BACnet is described. Based on this approach, an implementation of a prototype network that demonstrates the simultaneous use of BACnet and KNX is presented. Afterwards, a conclusion and an outlook on future work is given.

## 2 BACnet

In 1987, a project committee of the American Society of Heating, Refrigerating and Air Conditioning Engineers (ASHRAE) started the development of BACnet. The main objective was to provide a solution for building automation and control systems of all sizes and types. The development effort was completed in 1995, when BACnet was first published as an ANSI/ASHRAE standard. Later in 2003, it became a CEN and ISO standard [3]. The BACnet specification is under continuous maintenance and further development. The current version is BACnet 2004 [4].

The protocol architecture of BACnet consists of four layers corresponding to the physical, data link, network, and application layer of the ISO/OSI model. While BACnet specifies the network and application layer, a specific physical and data link layer is not laid down in the standard. In principle, any physical and/or data link layer combination could be used for BACnet message exchange. To increase the compatibility between BACnet devices, five different *network options* with particular ranges of speed and throughput have been defined. These five network options are:

- Ethernet
- ARCNET
- Master-Slave/Token-Passing (MS/TP)
- LonTalk
- Point-to-Point (PTP)

In addition to these network options, *BACnet/IP* allows the use of BACnet communication over IP networks. For tunneling, a special device called *BACnet/Internet Protocol Packet-Assembler-Disassembler (B/IP PAD)* encapsulates the BACnet message into a UDP packet. The packet is transmitted to the destination B/IP PAD where it is reconverted into a BACnet message. Another possibility to use IP networks is to employ UDP as native data link layer protocol. To achieve this, the *BACnet Virtual Link Layer (BVLL)* was defined. For broadcast communication covering multiple IP subnets, a special device called *BACnet Broadcast Management Device (BBMD)* is required.

Figure 1 gives an overview of all available network options and BACnet/IP.

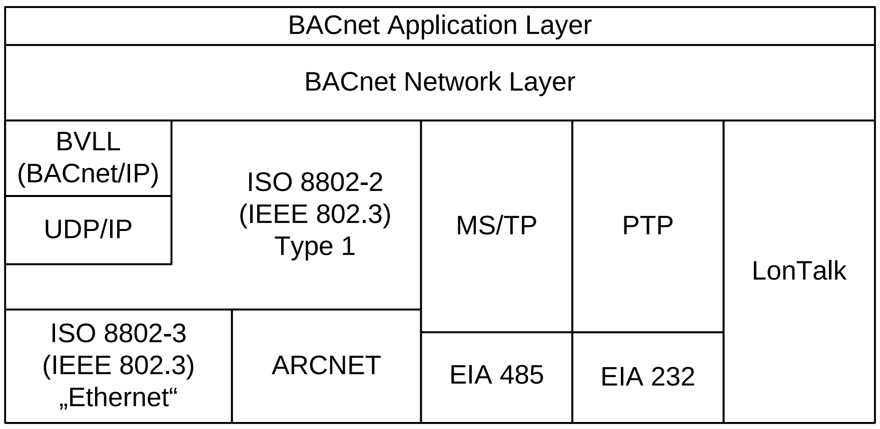


Figure 1: BACnet network options

## 2.1 Network topology

In BACnet, each device is attached to a so called *physical segment*. To extend the maximum cable length, physical segments are interconnected by repeaters or bridges. An interconnection of such segments is called a *BACnet network*. These BACnet networks can be linked together to form a *BACnet internetwork*. To achieve such an interconnection, BACnet routers are necessary.

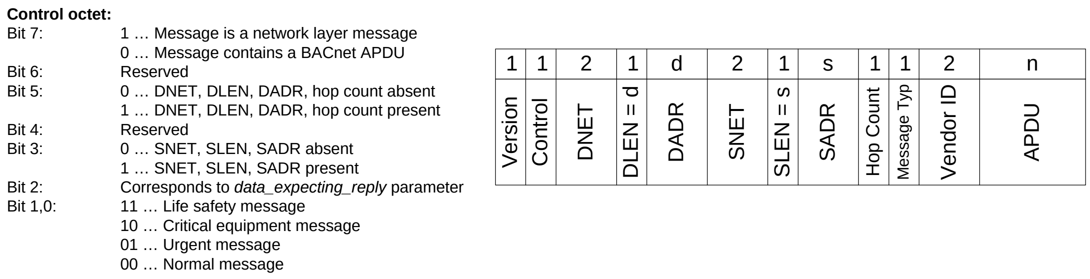
Each BACnet network has a 2 byte network address, which uniquely identifies the network in the BACnet internetwork. To identify a device within a BACnet network, each device has its local address. Since this local address corresponds to the layer 2 address of the used BACnet network option (i.e., the MAC address), the length of the local address varies.

## 2.2 BACnet network layer

The BACnet network layer provides an unacknowledged connectionless communication service to the application layer. The two service primitives are called `N-UNITDATA.request( destination_address, data, network_priority, data_expecting_reply)` and `N-UNITDATA.indication( source_address, destination_address, data, network_priority, data_expecting_reply)`. `N-UNITDATA.request()` is used by the application layer to send a BACnet message. `N-UNITDATA.indication()` indicates the application layer the reception of a new BACnet message. The destination address and (if present) the source address consist of an optional network number and the local address. The optional network number addresses a device located at a remote BACnet network. The local address allows to uniquely identify a device within its local BACnet network and consists of a device address and a link service access point. Since the MAC address of the data link layer is used as the device address, its length varies according to the chosen network option. Also, the structure of the link service access point is depending on the network option.

To specify the priority of BACnet messages, four different `network_priority` levels are defined. These levels are called `normal`, `urgent`, `critical equipment` and `life safety` and have to be mapped to the data link priority level of the underlying network option. The parameter `data_expecting_reply` indicates whether a reply to this service primitive is expected or not.

The general frame format of a BACnet NPDU[1] is shown in Figure 2. The structure of a specific NPCI depends on the used service. The presence or absence of NPCI parts is specified by the control field. Bit 7 indicates whether the NPDU contains a BACnet APDU or the message is a network layer message. Bit 5 defines the absence or presence of the fields DNET, DLEN, DADR, and hop count. Bit 3 indicates whether the source address information (SNET, SLEN and SADR) is present or not. Finally, the bits 1 and 0 define the `network_priority`.



**Control octet:**

| | |
|---|---|
| Bit 7: | 1 ... Message is a network layer message |
| | 0 ... Message contains a BACnet APDU |
| Bit 6: | Reserved |
| Bit 5: | 0 ... DNET, DLEN, DADR, hop count absent |
| | 1 ... DNET, DLEN, DADR, hop count present |
| Bit 4: | Reserved |
| Bit 3: | 0 ... SNET, SLEN, SADR absent |
| | 1 ... SNET, SLEN, SADR present |
| Bit 2: | Corresponds to *data_expecting_reply* parameter |
| Bit 1,0: | 11 ... Life safety message |
| | 10 ... Critical equipment message |
| | 01 ... Urgent message |
| | 00 ... Normal message |

| 1 | 1 | 2 | 1 | d | 2 | 1 | s | 1 | 1 | 2 | n |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Version | Control | DNET | DLEN = d | DADR | SNET | SLEN = s | SADR | Hop Count | Message Typ | Vendor ID | APDU |

Figure 2: BACnet NPDU

## 2.3 BACnet application layer

In BACnet, an *application process* is defined as the functionality that performs information processing required for a particular application. Each application process is divided into an *application program* and an *application entity* [4]. The application program is the part outside the application layer and is not specified by BACnet. The application entity refers to the part within the application layer and is responsible for the BACnet communication. The interaction between the application program and the application entity is handled via the *application program interface (API)* not defined by BACnet.

The application entity is divided into two parts. The *BACnet application service element (ASE)* consists of a set of application services that are classified into 5 different categories:

- Alarm and Event
- File Access
- Object Access
- Remote Device Management
- Virtual Terminal

These application services are implemented as unconfirmed or confirmed application services. Unconfirmed services do not expect a message reply whereas confirmed services are based on a client/server communication model. The service user (BACnet client) sends a service request

[1]According to the OSI model, NPDU denotes the network protocol data unit that consists of a network protocol control information (NPCI) and a network service data unit (NSDU). Corresponding abbreviations are used for the other protocol layers.

to the service provider (BACnet server) that replies with a positive/negative acknowledgment or with a message containing the result of the request.

The second part of an application entity is called *BACnet user element*. It is responsible for providing the API to the application program as well as the implementation of the service procedure portion of each application service. Figure 3 shows the application model.

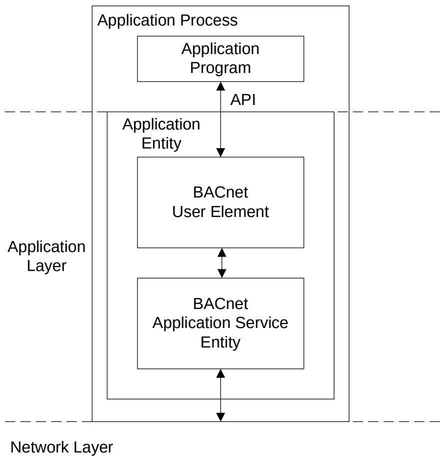


Figure 3: BACnet application model

The internal data structures used in a BACnet device are left open by the BACnet standard. To be able to exchange data between devices of different vendors, the network-visible representation of the data structures has been defined. This network-visible part of a single data element is called a *BACnet object*. Currently, 25 different object types are defined. Each object represents a collection of a set of properties. Each property has a dedicated data type that defines the size and the encoding of the particular data element. To represent a list of data elements, a data type called *BACnetArray* is also available. Up to now, nearly 200 different properties are defined. Three of them must be present in each object: `object-identifier`, `object-name` and `object-type`. To access and manipulate BACnet objects, different object access services are available. The most important ones are `ReadProperty` used to read the value of a property, and `WriteProperty` used to set a new property value.

For further details on BACnet see [2] and [4].

## 3 KNX as network option for BACnet

BACnet is based on a client/server communication model. A point-to-point communication service is necessary to send a request to a server and optionally receive a response. For this task, the BACnet application layer provides confirmed communication primitives. These primitives are built upon the unconfirmed communication service of the BACnet network layer which in turn requires (at least) an unacknowledged data link communication service.

To use the KNX data link and physical layer as new BACnet network option, the KNX `L_Data` service [5] is sufficient. The corresponding parameters of the `L_Data` service are the source and destination address, a message priority, the user data (i.e., LSDU) to be transferred, and a Boolean value specifying whether a layer 2 acknowledgment is mandatory or not. For setting up the required point-to-point communication within a local BACnet network communication, KNX provides the individual addressing scheme. In our approach, the BACnet destination address simply replaces the KNX individual address. Both, BACnet and KNX distinguish between 4 different priority levels, however, with different semantics. Since it has to be guaranteed that BACnet high priority messages are served first, the following mapping was chosen: `BACnet normal` to `KNX low`, `BACnet urgent` to `KNX normal`, `BACnet critical equipment` to `KNX urgent`, and `BACnet life safety` to `KNX system`. The remaining BACnet user data are embedded into the LSDU part of the `L_Data` Service. Although a KNX layer 2 acknowledgment for BACnet/KNX frames is not mandatory, its activation is recommended to increase the robustness.

This straight-forward mapping of BACnet NPDUs into the KNX LPDUs works as long as only BACnet/KNX devices are located in a KNX segment. However, in case of a mixed operation (i.e., BACnet/KNX and standard KNX devices share the same segment), problems will emerge. Consider, for example, a BACnet/KNX broadcast message is transmitted. Such a BACnet/KNX broadcast message causes a misinterpretation in standard KNX devices, since those devices are not able to differentiate between a "mapped" BACnet/KNX telegram and a standard KNX message. A possible solution would be to define a new LPDU type for BACnet/KNX messages. However, these messages could not be routed by standard KNX routers (couplers), since the new LPDU type is unknown to them. To overcome this limitation, a new TPDU type called `T_DATA_BACNET_REQ_PDU` has been defined.

Using this new TPDU type, an unwanted interference between BACnet/KNX and KNX messages is avoided. The encapsulation (replacement of the TSDU and mapping of parameters) is handled by a so called *BACnet/KNX Virtual Link Layer* (BKVLL) located between the KNX transport and the BACnet network layer (cf. Figure 4).

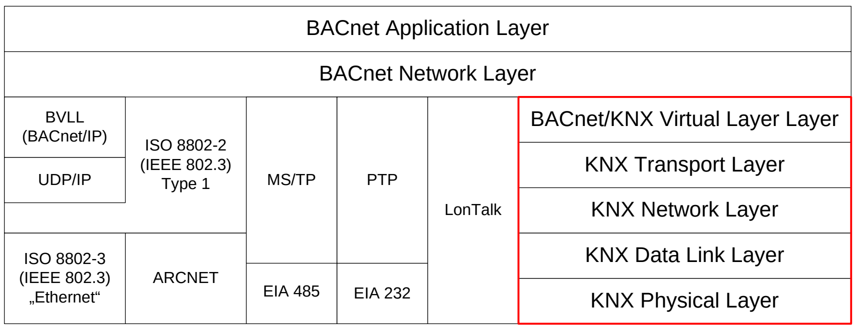


Figure 4: BACnet/KNX as new network option

BACnet supports message segmentation at the application layer. The maximum amount of APDU segments may vary and can be negotiated between the sending and receiving device. Due to the fact that different network options can be used in BACnet, the maximum length of a single APDU within a single message segment may vary too. According to the BACnet specification, the maximum APDU length is limited by the maximum APDU length transmittable by the device, the maximum APDU length accepted by the remote peer and the maximum NPDU length permitted by the local, remote or any intervening network segments between the sending and the receiving device. In addition to these constraints, BACnet demands a maximum APDU length of at least 50 octets. Since the APDU length of standard KNX frames is limited to 15 octets, their use without any further mechanisms is not sufficient.

One possible solution is to introduce segmentation at the data link layer performed independently of the segmentation at the BACnet application layer. This message segmentation has to be implemented by the BKVLL. However, the main drawback of this solution is the resulting overhead. Since the maximum NPCI length of a BACnet NPDU (in the case of a transmission between two BACnet/Ethernet peers) is 24 octets, the resulting NPDU length is 74 (assuming a maximum APDU length of 50). Together with 1 octet that is necessary for a segmentation sequence number, this leads to a minimum length of 75 octets. Therefore, this BACnet NPDU would have to be split into 5 KNX standard messages since a single standard KNX TSDU may contain only 15 octets. This would result to an overhead of 55%. [2]

To avoid the overhead of the data link layer segmentation, the KNX extended frame format can be used. Using KNX extended frames, the KNX TSDU can contain up to 254 octets. Considering a worst case NPCI length of 24 octets, an APDU length of 230 octets is possible which is sufficient to avoid the need for a data link layer segmentation. Figure 5 shows the frame structure of the proposed solution.

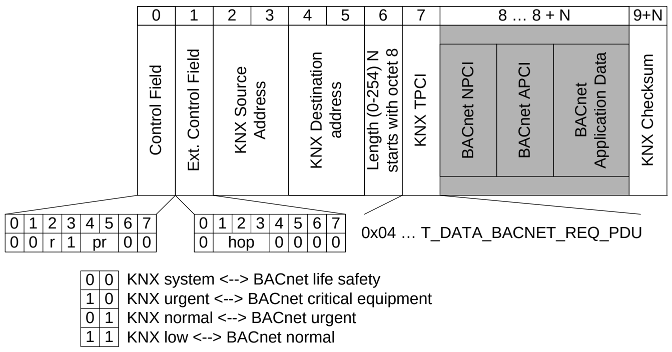


Figure 5: BACnet/KNX frame format

## 4 Prototype implementation

As a first proof-of-concept, a prototype BACnet internetwork has been set up. The internetwork consists of a BACnet/Ethernet network (i.e., Ethernet is used as network option for BACnet) and a KNX TP 1 network. Due to the presented encapsulation mechanism, BACnet communication is possible without disturbing the standard KNX communication. Figure 6 shows the experimental setup.

To demonstrate the mixed operation, different devices are integrated into the prototype network. At the BACnet/Ethernet network, a BACnet client is connected via Ethernet. The client runs the Visual Test Shell (VTS) [6], an open-source Windows application for BACnet conformance testing. The VTS supports the network options `Ethernet`, `PTP`, `ARCnet`, `MS/TP` as well as `BACnet/IP`. It is used to send and receive BACnet messages.

[2]In this case, the overhead is calculated as follows: $overhead = (KNXLPCIlength + KNXNPCIlength + KNXTPCIlength + sequenceID)/(BACnetNPDUlength) * 100$.

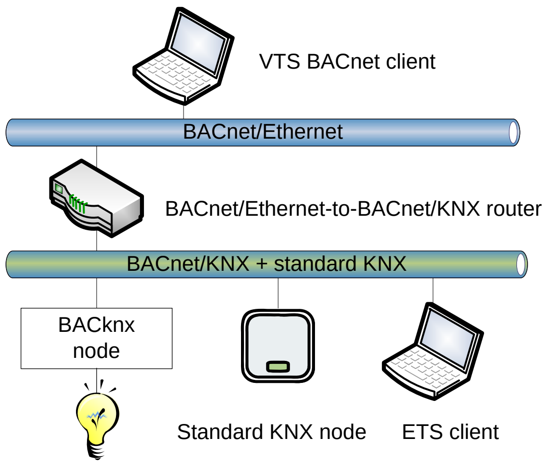


Figure 6: Prototype implementation

The KNX TP 1 network contains a standard KNX light switch, an ETS client (running on a standard laptop), and a special device called *BACknx node*. The BACknx node supports both the KNX operation mode as well as a BACnet operation mode (using KNX as network option). For testing purposes, a LED is attached to the BACknx node associated with a generic application object (cf. Section 4.1). The state of the object can be changed using BACnet as well as KNX services. On the one hand, the VTS client can alter it by sending a BACnet `PropertyWrite`. On the other hand, the KNX light switch can turn on/off the LED by sending a KNX `A_GroupValue_Write`. Additionally, the current state of the object can be read by the VTS (using BACnet `ReadProperty`) or by the ETS (via KNX `A_GroupValue_Read`). Also, the KNX services `A_PropertyValue_Read` and `A_PropertyValue_Response` can be used to read the current value of the binary object.

To interconnect the Ethernet network to the KNX TP 1 network, routing is necessary. For this reason, the design and implementation of a simple *BACnet/Ethernet-to-BACnet/KNX router* is part of the presented work. Its task is to receive messages from either the Ethernet or the KNX network, to convert them into messages suitable for the destination network and finally to forward the converted messages to the destination network (cf. Section 4.2).

## 4.1 BACknx node

The main hardware components of the BACknx node are a MSP430 microcontroller, a Twisted Pair - Universal Asynchronous Receive Transmit (TP-UART) and several digital I/O ports with some connected peripherals (LEDs and buttons). The TP-UART is used to access the KNX TP 1 medium.

The software implementation of BACknx is shown on the left hand side of Figure 7. It is divided into different software modules. A simple *Hardware Abstraction Layer (HAL)* provides an abstraction of the underlying hardware components (UART and timer). On top of it, a hardware independent *TP-UART* stack including the feature to handle KNX extended frames supports a performant implementation of the BACnet/KNX protocol. The *protocol stack* resting above consists of two major parts: A lean *KNX stack* supports basic services for group communication,

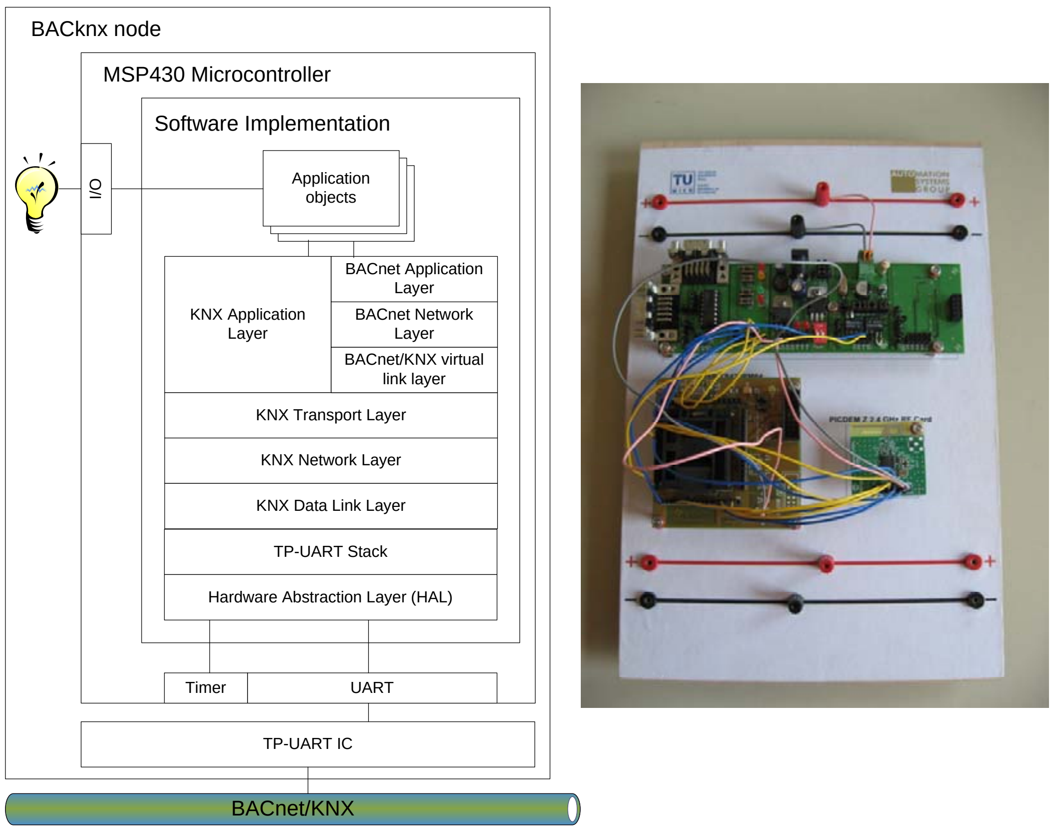


Figure 7: BACknx node

i.e., `A_GroupValue_Read`, `A_GroupValue_Write` and `A_GroupValue_Response` as well as some connection oriented services (e.g., `A_PropertyValue_Read`, `A_PropertyValue_Write` and `A_PropertyValue_Response`). Second, a simple *BACnet stack* supports the BACnet services `Who-is`, `I-am`, `ReadProperty`, and `WriteProperty`. The BACnet stack is placed on top of the BACnet/KNX virtual link layer being responsible for replacing the TSDU part of a KNX message with the BACnet NPDUs as well as providing the mapping (cf. Section 3).

To avoid the communication overhead of a data link layer segmentation, the KNX extended frame format is used. In principle, the TP-UART supports the use of extended frames. However, due to the format of the `U_L_DataContinue-Service` of the TP-UART, a maximum frame length of 62 octets, i.e. a KNX APDU length of 53 octets is possible [7]. Therefore, a data link layer segmentation is still necessary. To achieve a maximum BACnet APDU length of 50 octets (as required by the BACnet specification), splitting a BACnet NPDU into at most 2 KNX messages is sufficient. Compared to the data link layer segmentation approach that would be necessary if standard KNX frames are used, the resulting overhead is significantly smaller.

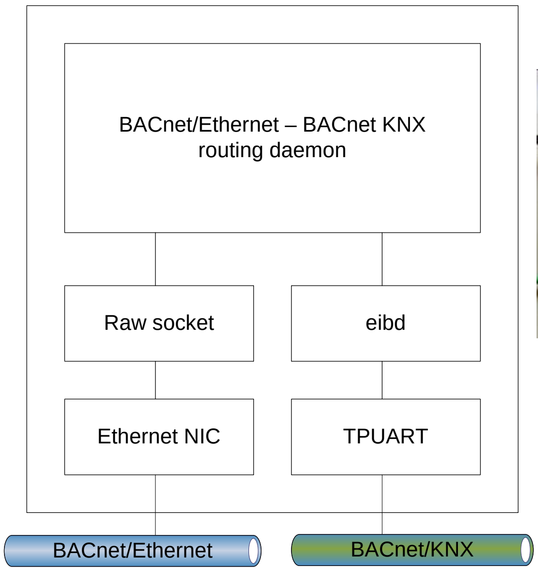
On top of the KNX and BACnet stack, a simple, generic object model has been implemented. In this model, different objects can be defined that encapsulate the process data. Each object includes a set of properties that in turn consist of an array of data elements. In addition to this array, each property has a field specifying the size of a single data element. The semantics as well as the encoding of the data elements have not been defined in this model yet.

The main advantage of the proposed solution is that each application object can be accessed using KNX and BACnet/KNX simultaneously. For example, the BACnet service `ReadProperty` and the KNX service `A_Property_Read` can both be used to read the data elements of a single object.

## 4.2 BACnet/Ethernet-to-BACnet/KNX router

To interconnect the BACnet/Ethernet to the BACnet/KNX network, a simple BACnet/Ethernet-to-BACnet/KNX router has been developed. It is implemented on an ARCOM VIPER embedded PC that is based on a 400MHz XScale processor. The embedded PC provides an integrated 100 MBit Ethernet controller, several EIA 232 interfaces and a USB host controller. To connect to the BACnet/Ethernet network, the onboard Ethernet controller is used. Connection to KNX TP1 has been realized with a TP-UART board that is connected to one of the EIA 232 interfaces.

Embedded Linux has been chosen as operating system for the router device. A *routing daemon* routes the network traffic between the BACnet/Ethernet and the BACnet/KNX network. It simply waits for incoming BACnet/KNX and BACnet/Ethernet messages, converts them into destination network messages and finally forwards them to the destination network. Incoming KNX frames are captured using *eibd* [8, 9] whereas incoming BACnet frames are captured using a local *UNIX raw socket*. Figure 8 shows the software design as well as a picture of this routing device.



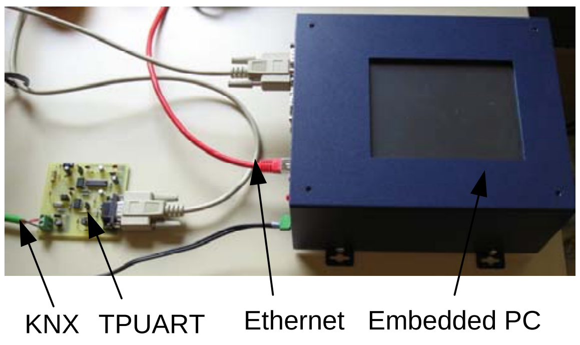


Figure 8: BACnet/Ethernet-to-BACnet/KNX router

## 5 Conclusion and future work

This paper presents an approach to use the KNX physical and data link layer as network option for BACnet. The main advantage of the proposed solution is that standard KNX and BACnet/KNX devices can share the same network medium without interfering each other.

As a first proof-of-concept, a simple prototype BACnet internetwork containing a BACnet/Ethernet as well as a BACnet/KNX network has been implemented. Besides standard KNX and BACnet devices, a special device called BACknx has been integrated into this prototype internetwork. BACknx has been developed to support both, the KNX as well as the BACnet protocol. Currently, both protocol stacks provide only basic communication services. As a next step, these stacks shall be improved by providing support for more advanced communication services. Additionally, the implemented object model shall be enhanced.

To be able to route the network traffic between the two networks, the development of a so called BACnet/Ethernet-to-BACnet/KNX router was part of the project. Currently, this router provides only basic routing based on static routing tables. Therefore, providing support for BACnet router auto-configuration and router table maintenance is planned.

Furthermore, a VTS plugin shall be developed that provides the opportunity to use KNX as a BACnet network option.

## References

[1] "Building Automation and Control Systems (BACS) – Part 2: Hardware", IS0 16484-2, 2004.
[2] W. Kastner, G. Neugschwandtner, S. Soucek, and H. M. Newman, "Communication Systems for Building Automation and Control", *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1178–1203, 2005.
[3] "Building Automation and Control Systems (BACS) – Part 5: Data Communication Protocol", ISO 16484-5, 2003.
[4] "BACnet – A Data Communication Protocol for Building Automation and Control Networks", ANSI/ASHRAE 135, 2004.
[5] "KNX Specification", Version 1.1, 2004.
[6] "Visual Test Shell for BACnet", http://sourceforge.net/projects/vts/, 2007, version 3.4.10.
[7] Siemens, "Technical Data EIB-TP-UART-IC", 2001, version D.
[8] M. Koegler, "Free Development Environment for Bus Coupling Units of the European Installation Bus", Master's thesis, Vienna University of Technology, Institute of Computer Aided Automation, 2005.
[9] M. Koegler, "eibd", http://www.auto.tuwien.ac.at/ mkoegler/index.php/eibd.