Gathering Sensor Data in Home Networks with IPFIX

Thomas Kothmayr, Corinna Schmitt, Lothar Braun, Georg Carle

Institut für Informatik, Technische Universität München
Garching bei München, Germany
kothmayr@in.tum.de, {schmitt,braun,carle}@net.in.tum.de

Abstract. New developments in military, health and home areas call for new approaches for data acquisition in real-time. Such application areas frequently include challenging requirements for collection, processing and analysis of environmental data. Wireless Sensor Networks can collect such environmental data efficiently. Collected sensor node data needs to be transmitted in an efficient way due to limitations of sensor node resources in battery power and available bandwidth. In this paper, we present a method for efficient transmission of sensor measurement data using the IETF standard IPFIX. We show that its template based design is suitable for efficient transmission of senor data with low bandwidth consumption. In this paper, we present the protocol and its implementation in Wireless Sensor Networks (WSNs). Additionally, a header compression scheme is introduced which further reduces communication cost during data transmission.

1 Introduction

Research efforts for wireless sensor technologies become more and more important due to the number of devices in use. Common sensor nodes are only equipped with low-cost hardware and are limited in available bandwidth, memory and battery power. Therefore, communication within a sensor network needs to be very efficient. As bandwidth is limited and data transmission exhausts battery power, transmitting sensor measurement data with little overhead is necessary.

Home networks have an additional requirement. Adding new sensor nodes into a home network should be performed without any (or only minimal) manual reconfiguration of the network. Additionally, Wireless Sensor Networks (WSNs) should be seamlessly integrable into an existing infrastructure.

Concerning resources, similar constraints can be found in the field of network monitoring. Although network monitors are usually equipped with a lot of memory and processing power, they have to observe and process a lot of traffic. Generating, encoding and transmitting information about the observed traffic needs to be implemented at low cost in order to preserve most of the available resources for the monitoring itself.

Therefore, Claise et al. developed the IP Flow Information Export protocol (IPFIX) [3], which is used for transmitting monitoring data. It was standardized

by the Internet Engineering Task Force (IETF) in 2008. The protocol was designed to transport flow and packet data, but can also be used for transmitting arbitrary data in an efficient way. It has a template-based concept for encoding measurement data.

In this paper, we present the protocol IPFIX and analyse how it can be used for efficient transmission of sensor data within a Wireless Sensor Network. Furthermore, we will discuss how IPFIX can be embedded into home networks with an infrastructure of wireless sensor nodes.

The remainder of this paper is organized as follows: Section 2 presents the IPFIX protocol and discusses its properties, focusing on the special constrains of wireless sensor nodes. Afterwards, Section 3 describes how IPFIX on wireless sensor nodes can be deployed in the context of IP based home networks. Furthermore, we will discuss how data security methods and compression can be integrated with IPFIX. Afterwards, we will describe our implementation approach of IPFIX in Section 4. Finally, Section 5 will discuss related work before conclusions are drawn in Section 6.

2 The IP Flow Information Export protocol

2.1 The Protocol

IPFIX [3] was developed by the Internet Engineering Task Force for transmitting flow information between a network monitor and flow data collectors. Communication takes place between an *Exporter* and a *Collector* in IPFIX terminology. IPFIX is specified as a PUSH-Protocol with an exporter periodically transmitting data to one or more collectors.

This design choice seems to be suitable for WSNs because wireless sensor devices tend to disable their wireless network device as long as possible in order to save energy. Aggregating sensor data with a request-response protocols may fail in these scenarios.

For IPFIX, a template based design was developed to exchange measurement data with little overhead. Measurement data is exchanged in so called *Records*. The protocol distinguishes, amongst others, between *Template Records* and *Data Records* as shown in Figure 1.

Data Records contain the measured data while Template Records contain meta information about the information which is transmitted in the Data Records. This meta information covers the type and the length of the measurement data. An Exporter sends a Template Record only once to its Collector to announce the structure of the upcoming data records. The Template Record is stored by the Collector for decoding incoming Data Records. An unique ID, called *Template ID*, is assigned with every Template Record and it is sent to the collector. Further Data Records will reference this ID.

As shown in Figure 1, each Template Record describes the encoding of the transmitted sensor measurement values in a Data Record. A Record may contain several several *fields* where a field corresponds to a measurement type like

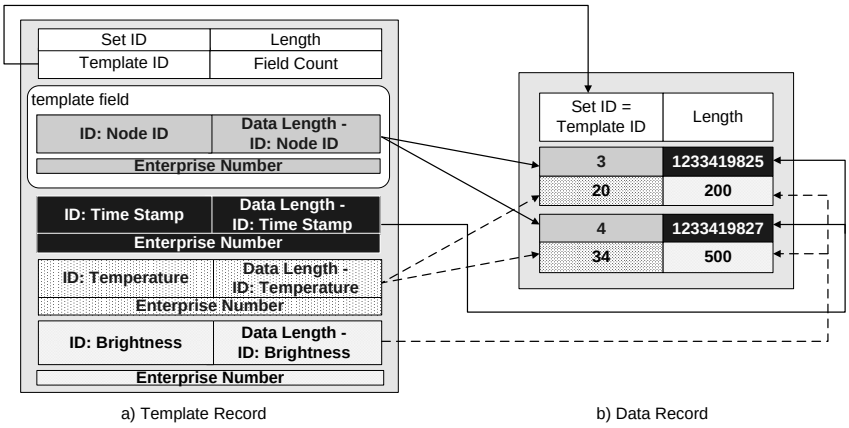

Fig. 1. IPFIX Records with decoding pointers

brightness or humidity. Each field in the Template Record describes the type and length of the corresponding field in the Data Record (Figure 1 shows 4 template fields). The type is uniquely described with a *Type ID* and an *Enterprise ID* in a template field. The Type ID specifies the type of data while the Enterprise ID denotes the organization which issued the Type ID. IPFIX standardized several IDs which are necessary to exchange traffic measurement data like sourceIPv4Address or destinationIPv4Address. The ID field consists of 16 Bits, where IDs 1-32767 are reserved for these traffic measurement data types [21].

If vendors want to exchange different data, for example sensor measurements, new IDs located above ID 32767 must be used. Hence, if the most significant bit for all these IDs is set to 1, a Collector concludes to see a *non standard ID*. In the next step the *Enterprise ID* (EID) will be checked by the Collector in order to find the organization which issued the ID. Each vendor has to register an Enterprise ID with the Internet Assigned Numbers Authority (IANA) [8] which will ensure that any vendor can be uniquely identified. Each vendor can specify up to 32767 own IDs for their data, because 15 bits are left for the Type ID field. We can use this facility to transmit sensor measurement data over IPFIX. It is necessary to register an Enterprise ID for sensor measurement data. Afterwards, we can specify our own standard IDs for common sensor measurement data (e.g. temperature value).

A template field also contains a *length* which announces the length of the transmitted data field in the Data Record. The length is declared in a 16 Bit counter, which allows very long data fields to be included in a Data Record. The fields length is important for a Collector when it decodes a Data Record as Data Records do not include any meta information about the measurement data.

As Figure 1 shows, several template fields may be included into one Template Record, e.g. time stamp and brightness measurement data. If different data should be transmitted to different nodes the Exporter needs more than one

Template Record. Different Template Records are also needed if aggregated and non-aggregated data should be transmitted.

Sensor nodes act as Exporters and transmit their measurement data using IPFIX. When the sensor node boots up, it has to announce a Template in order to announce its measurement data to the Collector. This has to be done only once, as a Collector has to buffer the Template and can use it to decode Data Records. Data Records do not have to contain anything but the measurement data as all meta information has been already sent in the Templates. They only have to contain the number of transported data fields as well as the template ID which is necessary for decoding the record. If a template announces two types of measurement data, e.g. light and temperature, it forms the template record {light, temperature}. Therefore, data records also consist of the tuple {light value, temperature value}. As a consequence, both values need to be included into the record. Several data records can be put into a single message. All records within a packet that can be decoded with a single template, form a so called *Data Set* as shown in Figure 1.

An Exporter transmits a Data Record. The Collector will look up the Template ID and uses the corresponding template to decode the data as illustrated in Figure 1. A Pointer will be hold by the IPFIX parser which points into the Data Set after the length of the Data Set field. The length of the first field will be looked up in the Template Record and the appropriate numbers of bytes will be read. The data type can be identified by its Type ID. Afterwards, the pointer will be advanced by the length of the given field. Then, the next field will be read in the same way.

This template based approach will ensure that meta information about the transmitted data is sent only once. Thus, meta information does not need to be transmitted with every measurement report by the sensor nodes. This in turn results in smaller packets.

Both producing as well as parsing IPFIX Data Records is very easy. The header which contains, amongst others, the Template ID and the number of measurement fields is produced by a sensor node. In the next step the measurement data is packed into the Template in the announced order. A Collector has to read the Template ID and can then read one data field after another as specified in the Template. This process is easy to implement (see Section 4) and has very low processing needs as only pointers need to be moved over the data record. If a pre-defined (hard coded) template is used, this process can be implemented even on very small motes. Multiple templates could be used if the nodes would have more resources, to allow measurement data analysis as done by the base station or servers.

2.2 Identifying measurement data of sensors

Sensor measurement data is identified by the *Type ID* and the *Enterprise ID* (EID). To enhance interoperability they need to be standardized. For today's home networks, typical environmental data can be measured by sensor nodes. Therefore, standard Type IDs can be issued. Up to now, no EID for sensor node

data exists, thus it must be chosen and registered by IANA. This EID can then be used to identify sensor node measurement data. Also, new IDs describing typical sensor data as shown in Table 1 must be standardized. Semantics and type length need to be included in the ID standardization in order to ensure interoperability. New generations of sensor nodes will have the ability to measure other types of data which will result in new IDs. Each vendor can register their own EID and specify their own IDs if he wants to include proprietary data types. However, as the common base for transmitting data is still IPFIX, IPFIX interoperability between devices in the network is enhanced.

Table 1. Possible IDs Space for Sensor Measurement Data

ID	Purpose	Length	Range
1	Node-ID	2 bytes	0 - 65535
2	Temperature	2 bytes	-40 - 123.8C
3	Seismic Data	2 bytes	-2g - 2g
4	Brightness	2 bytes	0 - 10000 Lux
5	Humidity	1 byte	0 - 100% RH
6	Barometric Pressure	1 byte	300 - 1100 mbar

2.3 Data compression and aggregation on top of IPFIX

Due to limited resources on sensor nodes, minimizing data during transmission is desired. Therefore, aggregation can be performed on the IPFIX data in order to reduce the overall amount of data. At first, several measurement results from one or several sensor nodes can be aggregated within a single data packet. Therefore, less packets need to be transmitted which saves energy on the sensor nodes. This kind of data aggregation technique works on arbitrary data without considering measurement context. Additional aggregation techniques can be deployed which consider application context as introduced by Przydatek et al. [20]. Aggregator nodes in the WSN need to be equipped with hardware because they have to store the templates of the child nodes. If a WSN is composed of many uniform nodes which use the same template, all nodes can perform data aggregation.

Another possibility to reduce the transmitted data amount is data compression. The authors in [17] showed that flow and packet measurement data can be compressed with simple methods resulting in smaller packet, which further helps to reduce bandwidth consumption. Thus, it can be reasonable to perform compression on sensor measurement data, too. However, this approach focuses on compressing the actual IPFIX payload. Since typical packet sizes in WSNs are small, the IPFIX header introduces a big source of overhead. Therefore we will introduce an approach to minimize this overhead by compressing the IPFIX header in Section 4.

The wireless part of the network must be connected to a wired infrastructure at some point as described in Section 3. This wired infrastructure is usually based on IP. Hence, using IP within the WSN seems to come natural. Additionally,

IPFIX was standardized to work on IP. By using IP in WSNs, wireless nodes can be addressed by nodes in the wired infrastructure. This enables data transmission from the wired infrastructure into the WSN.

In order to optimize IPv6 for the use in WSNs, 6LoWPAN was developed for wireless sensors and was standardized by the IETF [15]. Harvan et al. implemented an 6lowpan/IPv6 stack on top of 802.15.4 networks [7]. 802.15.4 provides two types of addresses with a length of 16 or 64 bit. Depending on the used hardware, the transmitted payload with 6LoWPAN can be up to 127 bytes for one frame. Larger IPv6 packets need to be fragmented in order to be transmitted within the 6LoWPAN network. As IPv6 has a header size of 40 bytes, too much payload size is occupied by header information. Therefore, a header compression scheme has been standardized resulting in a 2 bytes sized 6LoWPAN header. Similar compression mechanism can be used for the transport headers. An 8 bytes sized UDP can be down sized to four bytes using this compression scheme. The IPv6 compression mechanism is called HC1 and the UDP compression mechanism is called HC_UDP. Without this compression, only 50-66 bytes are left for the data payload, depending on the address types in the 802.15.4 Header. With compression, there is space for 94-110 bytes which nearly doubles the available space for payload [15].

3 Application in Home Networks

Wireless Sensor Networks can perform valuable tasks in home networks. Home networks have additional requirements to WSNs, compared to other applications of sensor networks.

One requirement is the seamless integration into the existing infrastructure. Additionally, users might want to buy devices from different vendors and deploy them into their network. Therefore, devices of different vendors should interoperate and integrate themselves into the existing infrastructure of the Wireless Sensor Network. Handcrafted vendor specific protocols are unlikely to fulfill this requirement. Instead, a common standard for data transmission, like IPFIX, is needed to achieve interoperability.

We will now present why IPFIX is a suitable protocol for the deployment in home networks. Furthermore, we will present how to use application aware data compression techniques to reduce the overall data amount in the network.

Our proof-of-concept implementation is implemented in the context of the Eureka Celtic Project "Autonomic Home Networking" (AutHoNe) [2].

3.1 Application Scenario

The sensor network which is deployed in our home network is supposed to collect environmental data. This data comprises temperature and lighting measurements at the moment and is used to control the lighting and temperature conditions within the house.

Therefore, every room contains several sensors which are linked by a low-power IEEE 802.15.4 wireless mesh. As home networks are usually based on the IP protocol, sensor nodes should support IP, too [5]. This can be achieved by using 6LoWPAN [7], an IPv6 standard for IEEE 802.15.4 networks. As IPFIX was designed to run on top of IP, it is not necessary to adapt IPFIX to work with other network layer protocols. 6LoWPAN is an adequate solution for sensor nodes as it meets sensor node requirements by defining header compression mechanisms for IPv6 packets transmitted over IEEE 802.15-based networks.

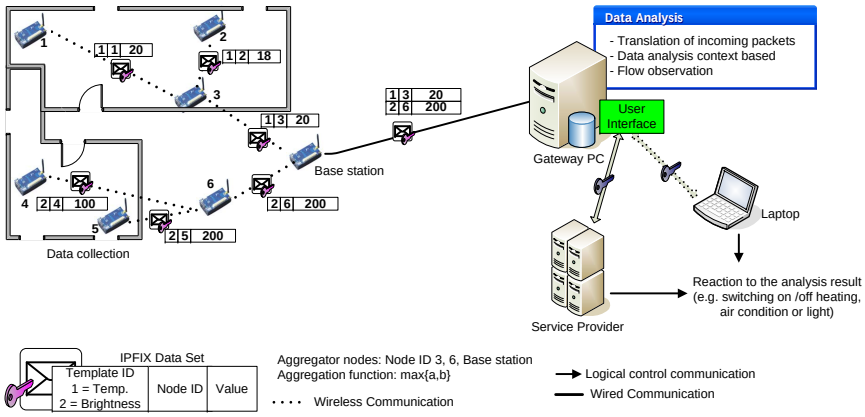

Fig. 2. Overview of application scenario

Figure 2 presents our application scenario. We assume to have several rooms which are equipped with sensor nodes. The sensor nodes are able to measure temperature and light, and are able to build a meshed network to transmit the measurement data to a central server. The server is able to analyse the data and to control the heating and lighting system. For our testbed we use the IRIS motes from Crossbow Technology Inc. [4] as node hardware. The IRIS mote which is used in our setup has the dimensions of 58 x 32 x 7 mm, without the battery pack. Thus, it does not leave much room for the micro controller, flash memory (128kb) and RF transceiver, all of which are located on this board. The available sensor boards have sensors for temperature, brightness and humidity among others.

3.2 IPFIX for Data Transmission

As wireless nodes boot up in the scenario, they will use the 6LoWPAN auto configuration features to obtain an address. Using this address, they will announce their templates to a central server. This server either needs to be configured on the sensor nodes or a special address in the IEEE 802.15.4 can be chosen to address the server.

Afterwards, all nodes in a room measure the current temperature and send their measurement results to the server. During this process, aggregation can be

performed by *aggregator nodes*. All nodes that are able to parse IPFIX messages and have enough resources for holding at least three IPFIX messages in memory can be used as aggregator nodes.

Since transmission is performed on the mesh network, these nodes can aggregate their measurement data with other received measurement data into a single packet. Application specific aggregation for home networks can be performed. In our home network, heat control can be activated for each room depending on the measured temperature. For each room, only minimum, maximum or average temperatures are needed for a decision on whether to turn on the heating or the air condition. IPFIX messages that travel through the network can therefore be aggregated as suggested by Przydatek et al. [20].

Adding devices from different manufactures into the WSN can be done, if they support IPFIX. If all of them use only standard IDs, interoperability between all devices is ensured. If some device vendor wants to specify their own data format, they can register their own EID and issue own IDs. These devices can still be integrated into the home network, as other nodes in the WSN do not need to know the semantics of the new IDs.

3.3 Security in IPFIX transmissions

Measurement data security and data integrity can be integrated as well. IPFIX copes with these security issues by specifying that every IPFIX device needs to support TLS (on stream based transport protocols) or DTLS (on datagram based transport protocols). Fouladgar et al. developed Tiny 3-TLS [6], a TLS handshake sub-protocol for sensor nodes, which can be used for securing IPFIX data transmission. This conforms to the security considerations from IPFIX.

Other protocols can also be used to assure data security and message authentication in WSNs. TinySec [9], for example, offers an encryption mode where data payload is encrypted and the packet itself is authenticated by a MAC. Another approach using the same idea as TinySec was developed by Luk et al. [13], called MiniSec. It is a secure sensor network communication architecture which modifies the common packet structure of TinyOS and combines features from TinySec and ZigBee [22] to perform low energy consumption and high security.

These protocols can be used instead of TLS, if an existing WSN already implements one of these protocols. However, using TLS is highly recommended.

4 Implementation of IPFIX for Wireless Sensors

In this section we want to characterize the problems and challenges we need to face during the implementation of IPFIX for Wireless Sensors. For Wireless Sensor Networks, two problem fields exist: Environment and Hardware.

In home networks, the environmental problems can be ignored because the network is deployed indoors. We know where each sensor is located and what kind of measurements can be conducted. The distance between the nodes is short, thus no environmental blockage must be taken into account.

Hardware limitations are way more concerning. As described in Section 3 IRIS motes from Crossbow Technology Inc. [4] are used in our application scenario. These nodes have several limiting factors, such as only 128kb flash memory, 512kb measurement flash and 8kb RAM. Together with the limited power supply of wireless senors the computational capacity is quite limited. These limitations should be kept in mind for the design decisions described in the upcoming section.

4.1 Design goals and implementation decisions

A sensor node has to perform the following tasks:

- Gather data from all sensors.
- Encode measurement results in IPFIX packets and transmit them to the base station.
- Perform in-network aggregation to reduce the amount of network traffic and preserve energy.

The receiving end at the Gateway PC has to perform these tasks: First receiving and parsing IPFIX packets on a Gateway PC must be guaranteed. And secondly the acquired data must be transfered to a home networking infrastructure.

Figure 3 shows all components involved in this process. Both ends, the sensor node as well as the receiving gateway are mapped on the Figure.

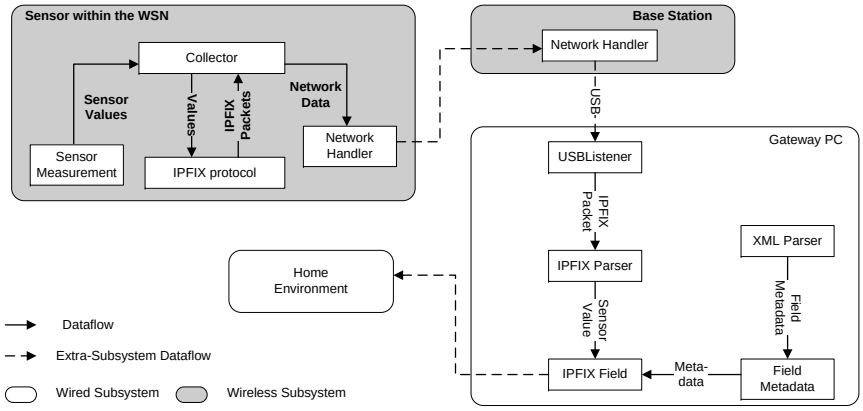

Fig. 3. Data Flow in all components

The node's sensors are queried periodically to generate new sensor data. These raw values are transmitted to the *tinyIPFIX library*, which encodes them into Data Records. The location within the Data Record is specified by an IPFIX template. The template is generated and sent automatically when the node

boots. After all sensors have been queried and the IPFIX packet is ready for transmission, it is sent to the *Base Station* via a multihop network. The Base Station listens for incoming packets from nodes in the network and transmits their payload to the Gateway PC over an USB port. On the Gateway PC, there a Collector waits for transmissions on the USB port. The IPFIX messages sent via USB are parsed according to the matching IPFIX templates, the sensor values are extracted as shown in Figure 1 and transferred to the home environment.

Currently the program for the sensor nodes consists of three main operative components, `ControllerC`, `tinyIPFIXC` and `NetworkHandlerC`. `ControllerC` is the main module of the program, it periodically queries the sensors and passes their reported values to `tinyIPFIXC`, an implementation of IPFIX for TinyOS 2.x. After all connected sensors have reported their values, `ControllerC` receives a byte array containing the finished IPFIX message, which it passes on to `NetworkHandlerC`. `NetworkHandlerC` implements the network communications in a transparent way, so that transmission protocols may be exchanged as needed. Currently, communication is based on the *Collection* protocol of TinyOS. We plan to migrate this to 6LoWPAN in the future. Figure 4 shows a simplified version of the application's wiring. As mentioned in [12], components need to be explicitly wired together.

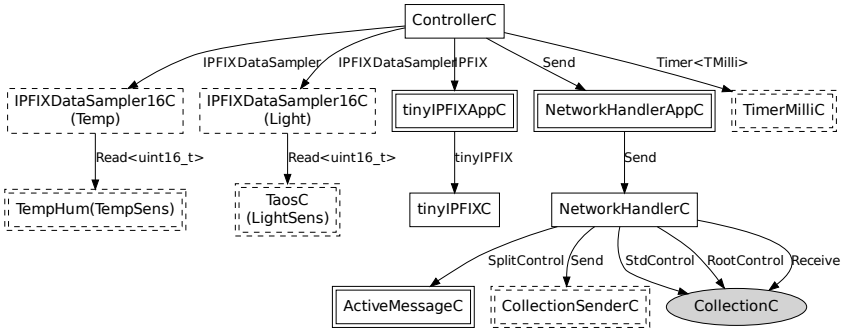

Fig. 4. Simplified wiring of the mote's program

The interface for acquiring sensor data was designed with the following goals: Additional readings need to be added without major changes to the application code, sensor readings should always be linked to an IPFIX *Field* and *Enterprise ID*. Finally, it should be possible to automatically generate IPFIX templates based on the connected sensors.

To generate the IPFIX template, the node queries all connected sensors about their Field ID, Enterprise ID and field length at startup. Similarly, to generate a data record, the node issues a read command to all connected sensors periodically. The sensors return their values after a certain latency and not necessarily in the same order as the read commands were issued. Therefore, a sensor needs to be associated with their respective Field ID, EID and field length.

This was addressed by designing a bidirectional interface called `IPFIXDataSampler` which is implemented by several generic modules. Generic

components can be instantiated with parameters [12]. This allows for a consistent linking of an IPFIX Field ID / Enterprise ID combination with a sensor, since every instance of `IPFIXDataSampler` can report exactly one value. One simply has to pass the according values when creating an instance of the module which provides the interface. `IPFIXDataSampler` defines two commands which are answered by two events. One is `command void report()` with `event void reportBack()` being the according event. `reportBack()` is used to register all providers of `IPFIXDataSampler` with `CollectorC`. Directly after booting, `CollectorC` issues the report command to all connected samplers. When they report back, it uses the information provided by `reportBack()` to create a new field definition in an IPFIX template, thereby addressing the design goal of automatic template creation. The second command is `command read()` which prompts the implementing module to return a reading of the connected sensor. This reading is reported back by `event void readDone()`.

```
 1  configuration ControllerAppC{}
 2  implementation{
 3      components ControllerC as App;
 4      ...
 5      components new IPFIXDataSampler16C(0x80A0,0xF0AA00AA) as Temp;
 6      components new IPFIXDataSampler16C(0x80A2,0xF0AA00AA) as Light;
 7      components new TempHumc() as TempSens, new TaosC() as LightSens;
 8
 9      Temp.Sensor -> TempSens;
10      Light.Sensor -> LightSens;
11
12      App.Sampler -> Temp;
13      App.Sampler -> Light;
14      ...
15  }
16
17  module ControllerC {
18      ...
19      uses interface IPFIXDataSampler as Sampler;
20      ...
21  }
22  implementation {...}
```

Fig. 5. Example of wiring IPFIXDataSampler providers to CollectorC

IPFIX does not transmit the data type of a field, instead it must be recognized based on the respective field ID, so this implementation can ignore the type and simply proceed working with a network order (big endian) byte array. However, functionalities that perform additional computation, such as e.g. mathematical aggregation functions like `SUM()` or `AVG()` must reconstruct the data type. The design goal of flexible extension is addressed by multiple wiring. In traditional languages, the concept of multiple callers to a single method implementation is commonplace. Since nesC interfaces are bidirectional, this also allows for multiple calls to a single method call, meaning multiple methods can be invoked with a single command. The ability to have multiple callers is described as Fan-in and the concept of multiple calls is called Fan-out [12]. By simply wiring multiple components providing `IPFIXDataSampler` to `ControllerC` one can make effective use of the Fan-out concept as shown in Figure 5.

4.2 IPFIX Header compression

Since IPFIX was designed for conventional networks, some extensions and changes have to be introduced to increase its efficiency in WSNs. Border gateways between the WSN and the wired network need to translated from compressed IPFIX to standard IPFIX. These border gateways are called IPFIX mediators in IPFIX terminology [11].

One of the problems when deploying IPFIX in sensor networks is the overhead introduced by the relatively large header which is at least 20 bytes in size (16 bytes from the Message Header + 4 bytes from the Set header) as is shown in Figure 6. However, the maximum size of a packet being transferred with an IEEE 802.15.4 network is 127 bytes. To address this issue, a header compression scheme was devised, which will be introduced in this section.

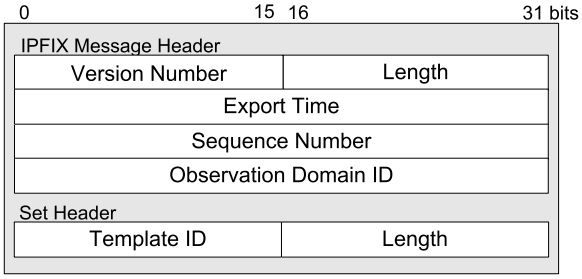

Fig. 6. IPFIX Headers

The idea behind our approach to header compression is to define the length of the fields separately in a pre header which is shown in Figure 7. First the Version field from the original IPFIX header is shortened to 5 bits, this leaves room for the IPFIX version to increase from version 10 to version 31. The definition of the length of the fields *Message Length*, *Export Time*, *Sequence Number* and *Observation Domain ID* follows. A value of 0 in the designated bit(s) means that the field is allowed 1 byte in the subsequent header, a value of 1 means 2 bytes, etc.. The next two bits are designated for the *Template Offset*. Decoders of IPFIX messages are expected to keep track of the sequence in which they received templates from the IPFIX exporters. A value of 0 in the Template Offset bits means that the decoder should use the template it has received last, a value of 1 means the template before that and a value of 2 means two templates before the last one. If this offset is given for a data message, 2 bytes for the Set ID can be saved. If template offset is set to 3 (both bits are one) it is ignored and a proper statement of the template ID is expected in the header. The next bit is called the *Single Set Flag*. It indicates whether the message contains only a single IPFIX set. If this is the case, the explicit statement of set length in the header can be omitted since this value can be computed from the total message length. The last bit in the pre header is the *Template Set Flag*. If it is set to one, the first set in the message is a template set which is defined to have Set ID = 2. Thus, the two bytes for definition of the set ID can be omitted.

In the best case scenario, all header fields can be fitted to 1 byte and the Set Header can be fully omitted. The possibility to shorten the Message Length and Observation Domain ID to 1 byte is fairly obvious. Most messages will be shorter than 255 bytes, in fact if they are transmitted in a single packet, they have to be smaller than 127 bytes with our hardware. Since the Observation Domain ID usually refers to the *Node ID*, a value of 1 byte can accommodate 256 nodes which represents a WSN of medium scale. The Sequence Number can also be shortened to 1 byte, since a rollover after 255 messages is non problematic due to the low data sampling rate of typical WSNs. For the time stamp, a value of 1 byte could refer to the time that has passed since the last full UTC time stamp has been sent. Since the field length can be different with every package sent, it is possible to only transmit a full 4 bytes time stamp periodically and suffice with a delta value in between. For the best case, this method can achieve a reduction in header size from 20 bytes to 6 bytes, or a compression of $81,25\%$. Figure 8 gives an example of the best case, which is actually fairly common since it shows the transmission of a Data Record referencing the last sent template set. In the worst case however, header size may increase to 33 bytes when all header fields are defined to be their original length.

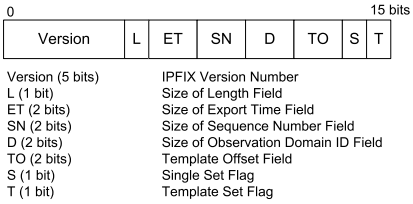

Fig. 7. The IPFIX pre header defining the length of the subsequent header

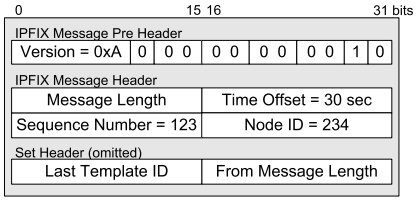

Fig. 8. Best case header for the IPFIX header compression

4.3 Receiving End

To process the data from the WSN to the interface of a home network environment, a gateway is needed. It must parse incoming IPFIX data according to templates generated by the nodes, enrich the received data with meta information (e.g.data type, storage location, etc.) and convert sensor specific values to a general, more abstract data type. In our implementation meta information is stored in a XML-file and fields are matched via Field ID and Enterprise ID.

When an IPFIX template is received, the Gateway creates a new instance of `Field` for every data item defined in the template, based on available meta information. Each instance may contain information about the data type of the field, it's name (for pretty printing), a flag whether or not updates should be passed on to the home network and a simple formula that can be used to perform computations on the received value. Formulas currently support addition, sub-

traction, multiplication, division and square roots. They may contain a variable x which is substituted for the received value when the expression is evaluated.

5 Related Work

In 2003 ZigBee was developed for wireless personal area networks [22]. It is a communication protocol based on the IEEE 802.15.4 standard. It was developed for small-scale isolated ad-hoc networks and limited to a single radio standard. Today it is a standard which is used nearly everywhere. But it requires more resources than the 6LoWPAN approach we are using as described in 2.3. ZigBee has a code size with mesh of 32-64K, requires 8K RAM, produces 8-16 bytes overhead, and supports 802.15.4 and no transport layer. 6LoWPAN has a code size with mesh of 22K, requires only 4K RAM, produces only 2-11 bytes overhead, and supports 802.15.4++ and UDP/TCP [16]. Finally, 6LoWPAN requires less resources than ZigBee, thus more resources are left for additional computations and transmissions.

In contrast, 6LoWPAN was developed for scalable networks as an end-to-end part of the Internet. It is applicable to any low-power and low-rate wireless radio. The used IP protocols tie together heterogeneous networks. ZigBee itself is not a standard, it is a special interest group, called ZigBee Alliance [22]. The IETF supports open, long-lived standards and this will be archived by 6LoWPAN which works with modified IPv6 protocols and stacks. Together with the home network scenario using IP addresses for communication we decided to implement 6LoWPAN on the IRIS motes.

As Kimura and Latifi discussed in [10], many algorithms for data compression exist but cannot adapt to the constraints of Wireless Sensor Networks. Thus, special algorithms were developed to compress the transmitted data like Coding by Ordering, Pipelined In-Network Compression, Low-complexity Video Compression and Distributed Compression.

The basic idea of the algorithm *Coding by Ordering* [18] is to drop data at the aggregation node. This can happen if the transmitted data is unique, and the order is irrelevant for the application. Now it is possible to use the transmitting order to transmit additional information to the receiver. This algorithm can be provided by an aggregator node in the network.

Arici et al. developed an compression algorithm called *Pipelined In-Network Compression* in 2003 [1]. This algorithm is also based on aggregator functionality. The sensor measurements are sent to an aggregator node and buffered. During the buffer period the incoming packets are combined and redundant data is deleted before ongoing transmission. The transmitted data uses a shared prefix which can be used for node IDs and time stamps to reduce space in new packets. Depending on the prefix length the data compression can be quite efficient.

The algorithms *Low-Complexity Video Compression* [14] and *Distributed Compression* [19] deal with data compression of visual data. the first algorithm is based on block changing and JPEG data compression. The second algorithm deals with the usage of side information to encode source information. This compression scheme can be applied to lossless and lossy compression schemes.

6 Conclusion

In this paper we introduced a concept to connect a wireless infrastructure to a wired home network scenario. This can be achieved by implementing 6LoWPAN on the sensor nodes to bring IP communication to a wireless infrastructure. In the next step we integrated IPFIX into the WSN and showed the applicableness for home networks in cooperation with 6LoWPAN.

At first, IPFIX defines a efficient data format for transmitting sensor measurement data using low bandwidth. Generating and parsing IPFIX data can be performed with little processing power, thus saving energy on the nodes. Arbitrary aggregation techniques can be deployed to further reduce the transmitted data.

If standard template IDs are issued, interoperability between different devices from different manufacturers can be ensured. At the same time, vendors can register its own enterprise and type IDs to build custom devices. These devices can still interoperate with other devices. By using IP on the network layer below IPFIX, wireless sensor networks can easily be integrated in existing home networks. Therefore, new sensor nodes can be easily deployed and new functionality to the network can be added in an automatic fashion.

To reduce the amount of data traffic within the network and to reduce the energy consumption of the network we introduced a concept of header compression for IPFIX and combined it with header compression of 6LoWPAN to increase the payload capability of each packet. Those compression functions can be combined with aggregation algorithms to gain more efficiency in the transmissions.

Acknowledgment

The presented work is part of the AutHoNe project which is partly funded by the German Federal Ministry of Education and Research under grant agreement no. 01BN070[2-5]. The project is being carried out as part of the CELTIC initiative within the EUREKA framework.

References

1. T. Arici, B. Gedik, Y. Altunbasak, and L. Liu. PINCO: a pipelined in-network compression scheme for data collection in wireless sensor networks. In *Proceedings of the 12th International Conference on Computer Communications and Networks (ICCCN)*, pages 539–544, Oct. 2003.
2. Autonomic Home Networking DE Project Page. http://www.authone.de, 2009.
3. B. Claise, S. Bryant, G. Sadasivan, S. Leinen, T. Dietz, and B.H. Trammell. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information (RFC 5101). Technical report, The Internet Engineering Task Force (IETF), January 2008.
4. Crossbow Technologies Inc. http://www.xbow.com/, 2009.
5. K. Das. IPv6 and Wireless Sensor Networks. *IPv6.com Tech Spotlight*, 2008.

6. S. Fouladgar, B. Mainaud, K. Masmoudi, and H. Afifi. Tiny 3-TLS: A trust delegation protocol for wireless sensor networks. *Lecture Notes in Computer Science*, 4357:32, 2006.
7. M. Harvan and J. Schönwälder. TinyOS Motes on the Internet: IPv6 over 802.15.4 (6lowpan). *PIK - Praxis der Informationsverarbeitung und Kommunikation*, 31(4):244–251, Okt.-Dez. 2008.
8. Internet Assigned Numbers Authority. http://www.iana.org/, 2009.
9. C. Karlof, N. Sastry, and D. Wagner. TinySec: a link layer security architecture for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 162–175. ACM New York, NY, USA, 2004.
10. N. Kimura and S. Latifi. A survey on data compression in wireless sensor networks. In *Proceeding of the International Conference on Information Technology: Coding and Computing (ITCC)*, volume 2, pages 8–13 Vol. 2, April 2005.
11. A. Kobayashi, B. Blaise, and K. Ishibashi. IPFIX Mediation: Framework. Technical report, The Internet Engineering Task Force (IETF), October 2009.
12. P. Levis and D. Gay. TinyOS Programming, July 2009.
13. M. Luk, G. Mezzour, A. Perrig, and V. Gligor. MiniSec: a secure sensor network communication architecture. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 479–488. ACM New York, NY, USA, 2007.
14. E. Magli, M. Mancin, and L. Merello. Low-complexity video compression for wireless sensor networks. In *Proceedings of the International Conference on Multimedia and Expo (ICME)*, volume 3, pages 585–588, Washington, DC, USA, 2003. IEEE Computer Society.
15. G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. IPv6 over Low Power Wireless Personal Area Networks (6LowPAN) - RFC 4944. Technical report, The Internet Engineering Task Force (IETF), September 2007.
16. G. Mulligan. The 6lowpan architecture. In *Proceedings of the 4th workshop on Embedded networked sensors (EmNets)*, pages 78–82, New York, NY, USA, 2007. ACM.
17. G. Münz and L. Braun. Lossless Compression for IP Flow Information Export (IPFIX). The Internet Engineering Task Force (IETF), Internet-Draft (work in progress), draft-muenz-ipfix-compression-00, 2008.
18. D. Petrovic, R.C. Shah, K. Ramchandran, and J. Rabaey. Data funneling: routing with aggregation and compression for wireless sensor networks. In *Proceedings of 1st IEEE International Workshop on Sensor Network Protocols and Applications*, pages 156–162, May 2003.
19. S.S. Pradhan, J. Kusuma, and K. Ramchandran. Distributed Compression In Dense Sensor Networks. *IEEE Signal Processing Magazine*, 19:51–60, 2002.
20. B. Przydatek, D. Song, and A. Perrig. SIA: Secure information aggregation in sensor networks. *J. Comput. Secur.*, 15(1):69–102, 2007.
21. J. Quittek, S. Bryant, B.Claise, B. Aitken, and J. Meyer. Information Model for IP Flow Information Export (RFC 5102). 2008.
22. ZigBee Alliance. ZigBee specification. Technical Report. Document 053474r06 Version 1.0, ZigBee Alliance, June 2005.