Independent Submission
Request for Comments: 8272
Category: Informational
ISSN: 2070-1721

C. Schmitt
B. Stiller
University of Zurich
B. Trammell
ETH Zurich

November 2017

TinyIPFIX for Smart Meters in Constrained Networks

Abstract

This document specifies the TinyIPFIX protocol that is used for transmitting smart-metering data in constrained networks such as IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN, RFC 4944). TinyIPFIX is derived from IP Flow Information Export (RFC 7011) and adopted to the needs of constrained networks. This document specifies how the TinyIPFIX Data and Template Records are transmitted in constrained networks such as 6LoWPAN and how TinyIPFIX data can be converted into data that is not TinyIPFIX in a proxy device.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This is a contribution to the RFC Series, independently of any other RFC stream. The RFC Editor has chosen to publish this document at its discretion and makes no statement about its value for implementation or deployment. Documents approved for publication by the RFC Editor are not a candidate for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8272.

carefully, as they describe your rights and restrictions with respect
to this document.

Table of Contents

1. Introduction

Smart meters that form a constrained wireless network need an application-layer protocol that allows the efficient transmission of metering data from the devices to a central analysis device. The meters used to build such networks are usually equipped with low-cost and low-power hardware. This leads to constraints in computational capacities, available memory, and networking resources.

The devices are often battery powered and are expected to run for a long time without having the possibility of recharging themselves. In order to save energy, smart meters often power off their wireless networking device. Hence, they don't have a steady network connection; they are only part of the wireless network as needed when there is data to be exported. A push protocol like TinyIPFIX, where data is transmitted autonomically from the meters to one or more collectors, is suitable for reporting metering data in such networks.

TinyIPFIX is derived from IPFIX [RFC7011]; therefore, it inherits most of IPFIX's properties. One of these properties is the separation of data and its data description by encoding the former in Data Sets and the latter in Template Sets.

Transforming TinyIPFIX to IPFIX as per [RFC7011] is very simple and can be done on the border between the constrained network and the more general network. The transformation between one form of IPFIX data into another is known as "IPFIX Mediation" [RFC5982]. Hence, smart-metering networks that are based on TinyIPFIX can be easily integrated into an existing IPFIX measurement infrastructure.

1.1. Document Structure

Section 2 introduces the terminology used in this document. Afterwards, hardware and software constraints in constrained networks, which will motivate our modifications to the IPFIX protocol, are discussed in Section 3. Section 4 describes the application scenarios and Section 5 describes the architecture for TinyIPFIX. Section 6 defines the TinyIPFIX protocol itself and discusses the differences between TinyIPFIX and IPFIX. The Mediation Process from TinyIPFIX to IPFIX is described in Section 7. Section 8 defines the process of Template Management on the Exporter and the Collector. Section 9 and Section 10 discuss the security and IANA considerations for TinyIPFIX.

2. Terminology

Most of the terms used in this document are defined in [RFC7011]. Each of these terms begins with a capital letter. Most of the terms that are defined for IPFIX can be used to describe TinyIPFIX. This document uses the term "IPFIX" to refer to IPFIX as defined in [RFC7011] and the term TinyIPFIX for the protocol specified in this draft document assuming constrained networks. The prefix "Tiny" is added to IPFIX to distinguish between the IPFIX version and the TinyIPFIX version.

The terms IPFIX Message, IPFIX Device, Set, Data Set, Template Set, Data Record, Template Record, Collecting Process, Collector, Exporting Process, and Exporter are defined as in [RFC7011]. The term IPFIX Mediator is defined in [RFC5982]. The terms Intermediate Process, IPFIX Proxy, IPFIX Concentrator are defined in [RFC6183].

All the terms above have been adapted from the IPFIX definitions. As they keep a similar notion but in a different context of constrained networks, the term "TinyIPFIX" now precedes the defined terms.

The term "smart meter" is used to refer to constrained devices like wireless sensor nodes, motes, or any other kind of small constrained device that can be part of a network that is based on IEEE 802.15.4 and 6LoWPAN [RFC4944].

TinyIPFIX Exporting Process

The TinyIPFIX Exporting Process is a process that exports TinyIPFIX Records.

TinyIPFIX Exporter

A TinyIPFIX Exporter is device that contains at least one TinyIPFIX Exporting Process.

TinyIPFIX Collecting Process

The TinyIPFIX Collecting Process is a process inside a device that is able to receive and process TinyIPFIX Records.

TinyIPFIX Collector

A TinyIPFIX Collector is a device that contains at least one TinyIPFIX Collecting Process.

TinyIPFIX Device

A TinyIPFIX Device is a device that contains one or more TinyIPFIX Collectors or one or more TinyIPFIX Exporters.

TinyIPFIX Smart Meter

A TinyIPFIX Smart Meter is a device that contains the functionality of a TinyIPFIX Device. It is usually equipped with one or more sensors that meter a physical quantity, like power consumption, temperature, or physical tampering with the device. Every TinyIPFIX Smart Meter MUST at least contain a TinyIPFIX Exporting Process. It MAY contain a TinyIPFIX Collecting Process in order to work as a TinyIPFIX Proxy or TinyIPFIX Concentrator.

TinyIPFIX Data Record

A TinyIPFIX Data Record equals an IPFIX Data Record in [RFC7011]. The term is used to distinguish between IPFIX and TinyIPFIX throughout this document.

TinyIPFIX Template Record

A TinyIPFIX Template Record is similar to an IPFIX Template Record in [RFC7011]. The Template Record Header is substituted with a TinyIPFIX Template Record Header and is otherwise equal to a Template Record. See Section 6.3.

TinyIPFIX Set

The TinyIPFIX Set is a group of TinyIPFIX Data Records or TinyIPFIX Template Records with a TinyIPFIX Set Header. Its format is defined in Section 6.2.

TinyIPFIX Data Set

The TinyIPFIX Data Set is a TinyIPFIX Set that contains TinyIPFIX Data Records.

TinyIPFIX Template Set

A TinyIPFIX Template Set is a TinyIPFIX Set that contains TinyIPFIX Template Records.

TinyIPFIX Message

The TinyIPFIX Message is a message originated by a TinyIPFIX Exporter. It is composed of a TinyIPFIX Message Header and one or more TinyIPFIX Sets. The TinyIPFIX Message Format is defined in Section 6.

TinyIPFIX Intermediate Process

A TinyIPFIX Intermediate Process is an IPFIX Intermediate Process that can handle TinyIPFIX Messages.

TinyIPFIX Proxy

A TinyIPFIX Proxy is an IPFIX Proxy that can handle TinyIPFIX Messages.

TinyIPFIX Concentrator

A TinyIPFIX Concentrator is device that can handle TinyIPFIX Messages (e.g., pre-process them) and is not constrained.

TinyIPFIX Proxy

A TinyIPFIX Proxy is an IPFIX Proxy that can handle TinyIPFIX Messages and is not constrained.

A TinyIPFIX Transport Session is defined by the communication between a TinyIPFIX Exporter (identified by an 6LoWPAN-Address, the Transport Protocol, and the Transport Port) and a TinyIPFIX Collector (identified by the same properties).

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Constraints

3.1. Hardware Constraints

The target devices for TinyIPFIX are usually equipped with low-cost hardware; therefore, they face several constraints concerning CPU and memory [Schmitt09]. For example, the IRIS mote from Crossbow Technologies, Inc. has a size of 58 x 32 x 7 mm (without a battery pack) [IRIS]. Thus, there is little space for a micro-controller, memory (128 kb program flash, 512 kb measurement serial flash, 8 kb

RAM, 4 kb configuration EEPROM), and radio-frequency transceiver, which are located on the board. The TelosB motes produced by Crossbow Technologies, Inc. [TelosB] and ADVANTIC SISTEMAS Y SERVICIOS S.L. [Advantic] are similar sized, but offering more memory (48 kb flash, 1024 kb serial, flash, 10 kb RAM, 16 kb configuration EEPROM). The same holds for OpenMote, but the offering is 512 kb flash and 32 kb RAM [openMote].

Network protocols used on such hardware need to respect these constraints. They must be simple to implement using little code and little run-time memory and should produce little overhead when encoding the application payload.

3.2. Energy Constraints

Smart meters that are battery powered have hard energy constraints [Schmitt09]. If two AA 2800-mAh batteries power the mote, they contain approximately 30,240 Joule of energy. If they run out of power, their battery has to be changed, which means physical manipulation to the device is necessary. Therefore, using as little energy as possible for network communication is desired.

A smart-metering device can save a lot of energy, if it powers down its radio-frequency transceiver. Such devices do not have permanent network connectivity; they are only part of the network as needed. A push protocol, where only one side is sending data, is suitable for transmitting application data under such circumstances. As the communication is unidirectional, a meter can completely power down its radio-frequency transceivers as long as it does not have any data to send. If the metering device is able to keep a few measurements in memory, and if real-time metering is not a requirement, the TinyIPFIX Data Records can be pushed less frequently, therefore saving some more energy on the radio-frequency transceivers.

3.3. Packet Size Constraints

TinyIPFIX is mainly targeted for the use in 6LoWPAN networks, which are based on IEEE 802.15.4 [RFC4944]. However, the protocol can also be used to transmit data in other networks when a mediator is used for translating the TinyIPFIX data into the data format used in the other network (e.g., IPFIX). And the protocol is able to map the 6LoWPAN addresses to the addresses used in the other network. This operation typically consists of per-message re-encapsulation and/or re-encoding. As defined [RFC4944], IEEE 802.15.4 starts from a maximum physical layer packet size of 127 octets (aMaxPHYPacketSize) and a maximum frame overhead of 25 octets (aMaxFrameOverhead), leaving a maximum frame size of 102 octets at the media access control (MAC) layer. On the other hand, IPv6 defines a minimum MTU

of 1280 octets. Hence, fragmentation has to be implemented in order to transmit such large packets. While fragmentation allows the transmission of large messages, its use is problematic in networks with high packet loss because the complete message has to be discarded if only a single fragment gets lost.

TinyIPFIX enhances IPFIX by a header-compression scheme, which allows the header size overhead to be significantly reduced. Additionally, the overall TinyIPFIX Message size is reduced, which reduces the need for fragmentation.

3.4. Transport Protocol Constraints

The IPFIX standard [RFC7011] defines several transport protocol bindings for the transmission of IPFIX Messages. Stream Control Transmission Protocol (SCTP) support is REQUIRED for any IPFIX Device to achieve standard conformance [RFC7011], and its use is highly recommended. However, sending IPFIX over UDP and TCP MAY also be implemented.

This transport protocol recommendation is not suitable for TinyIPFIX. A header compression scheme that allows a compression of an IPv6 header from 40 octets down to 2 octets is defined in 6LoWPAN. There is a similar compression scheme for UDP, but there is no such compression for TCP or SCTP headers. If header compression can be employed, more space for application payload is available.

Therefore, using UDP on the transport layer for transmitting TinyIPFIX Messages is RECOMMENDED. Furthermore, TCP or SCTP are currently not supported on some platforms, like on TinyOS [Harvan08]. Hence, UDP may be the only option.

Every TinyIPFIX Exporter and Collector MUST implement UDP transport-layer support for transmitting data in a constrained network environment. It MAY also offer TCP or SCTP support. In the case in which TCP or SCTP MAY be used, power consumption will grow and the available size of application payload compared to the use of UDP May be reduced. If TinyIPFIX is transmitted over a unconstrained network, using SCTP as a transport-layer protocol is RECOMMENDED. TinyIPFIX works independent of the target environment, because it MUST only be ensured that all intermediate devices can understand TinyIPFIX and be able to extract needed packet information (e.g., IP destination address). TinyIPFIX messages can be included in other transport protocols in the payload whenever is necessary, making TinyIPFIX highly flexible and usable for different communication protocols (e.g., Constrained Application Protocol (CoAP), UDP, TCP). TinyIPFIX itself just specifies a messages format for the collected

data to be transmitted.

The constraints on UDP usage given in Section 6.2 of [RFC5153] apply to TinyIPFIX as well. TinyIPFIX is not intended for use over the open Internet. In general, the networks on which it runs are considered dedicated for sensor operations and are under the control of a single administrative domain.

4. Application Scenarios for TinyIPFIX

TinyIPFIX is derived from IPFIX [RFC7011]; therefore, it is a unidirectional push protocol assuming UDP usage. This means all communication that employs TinyIPFIX is unidirectional from an Exporting Process to a Collecting Process. Hence, TinyIPFIX only fits for application scenarios where meters transmit data to one or more Collectors. In case pull requests should also be supported by TinyIPFIX, it is RECOMMENDED not to change the code of TinyIPFIX much to get along with the restricted memory available [Schmitt2017]. Meaning including just a one bit field, called type, to distinguish between push and pull messages would be feasible, but the filtering SHOULD be done by the gateway and not by the constrained device; meaning if a pull is performed, the constrained device is triggered to create a TinyIPFIX message immediately as usual, set the type field to one instead of zero (for a push message), and send message to the gateway. At the gateway, the filtering is performed based on the pull request.

If TinyIPFIX is used over UDP, as recommended, packet loss can occur. Furthermore, if an initial Template Message gets lost, and is therefore unknown to the Collector, all TinyIPFIX Data Sets that reference this Template cannot be decoded. Hence, all these Messages are lost if they are not cached by the Collector. It should be clear to an application developer that TinyIPFIX can only be used over UDP if these TinyIPFIX Message losses are not a problem. To avoid this loss, it is RECOMMENDED to repeat the Template Message periodically, keeping in mind that a Template never changes for a constrained device after deployment. Even when Template Messages become lost in the network, the data can be manually translated later when the Template Messages is re-sent. Including an acknowledgement mechanism is NOT RECOMMENDED due to overhead, because this would require storage of any sent data on the constrained devices until it was acknowledged. In critical applications, it is RECOMMENDED to repeat the Template Message more often.

TinyIPFIX over UDP is especially not a suitable protocol for applications where sensor data trigger policy decisions or configuration updates for which packet loss is not tolerable.

Applications that use smart sensors for accounting purposes for long-term measurements can benefit from the use of TinyIPFIX. One application for IPFIX is long-term monitoring of large physical volumes. In [Tolle05], Tolle et al. built a system for monitoring a "70-meter tall redwood tree, at a density interval of 5 minutes in time and 2 meters in space". The sensor node infrastructure was deployed to measure the air temperature, relative humidity, and photosynthetically active solar radiation over a long-term period.

TinyIPFIX is a good fit for such scenarios. Data can be measured by the sensors of the TinyIPFIX Smart Meter over several 5-minute time intervals; the measurements can be accumulated into a single TinyIPFIX Message. As soon as enough measurements are stored in the TinyIPFIX Message, e.g., if the TinyIPFIX Message size fills the available payload in a single IEEE 802.15.4 packet, the wireless transceiver can be activated and the TinyIPFIX Message can be exported to a TinyIPFIX Collector.

Similar sensor networks have been built to monitor the habitat of animals, e.g., in the "Great Duck Island Project" [GreatDuck] [SMPC04]. The purpose of the sensor network was to monitor the birds by deploying sensors in and around their burrows. The measured sensor data was collected and stored in a database for offline analysis and visualization. Again, the sensors can perform their measurements periodically, accumulate the sensor data, and export them to a TinyIPFIX Collector.

Other application scenarios for TinyIPFIX could be applications where sensor networks are used for long-term structural health monitoring in order to investigate long-term weather conditions on the structure of a building. For example, a smart-metering network has been built to monitor the structural health of the Golden Gate Bridge [Kim07]. If a sensor network is deployed to perform a long-term measurement of the structural integrity, TinyIPFIX can be used to collect the sensor-measurement data.

If an application developer wants to decide whether to use TinyIPFIX for transmitting data from smart meters, he must take the following considerations into account:

1. The application should require a push protocol by default. The timing intervals of when to push data should be predefined before deployment. The property above allows a TinyIPFIX Smart Meter to turn off its wireless device in order to save energy, as it does not have to receive any data.

2. If real-time reporting is not required, the application might benefit from combining several measurements into a single TinyIPFIX Message, causing delay but lowering traffic in the network. TinyIPFIX easily allow the combination of several measurements into a single TinyIPFIX Message (or a single packet). This combination can happen on the TinyIPFIX Smart Meter that combines several of its own measurements. Or, it can happen within a multi-hop wireless network where one IPFIX Proxy combines several TinyIPFIX Messages into a single TinyIPFIX Message before forwarding them.

3. The application must accept potential packet loss. TinyIPFIX only fits for applications where metering data is stored for accounting purposes and not for applications where the sensor data triggers configuration changes or policy decisions, except when Message loss is acceptable for some reason.

4. The application must not require per-message export timestamps (e.g., for auditing). TinyIPFIX removes export timestamps, generally only useful for Template Management operations, which it also does not support, from IPFIX. This is a minor inconvenience, since per-record timestamp Information Elements are also available in IPFIX.

5. Architecture for TinyIPFIX

The TinyIPFIX architecture is similar to the IPFIX architecture, which is described in [RFC5470]. The most common deployment of TinyIPFIX Smart Meters is shown in Figure 1, where each TinyIPFIX Smart Meter can have different sensors available (e.g., IRIS: Temperature, Humidity, Sound; TelosB: Temperature, Bridgeness, Humidity, GPS) building the sensor data.

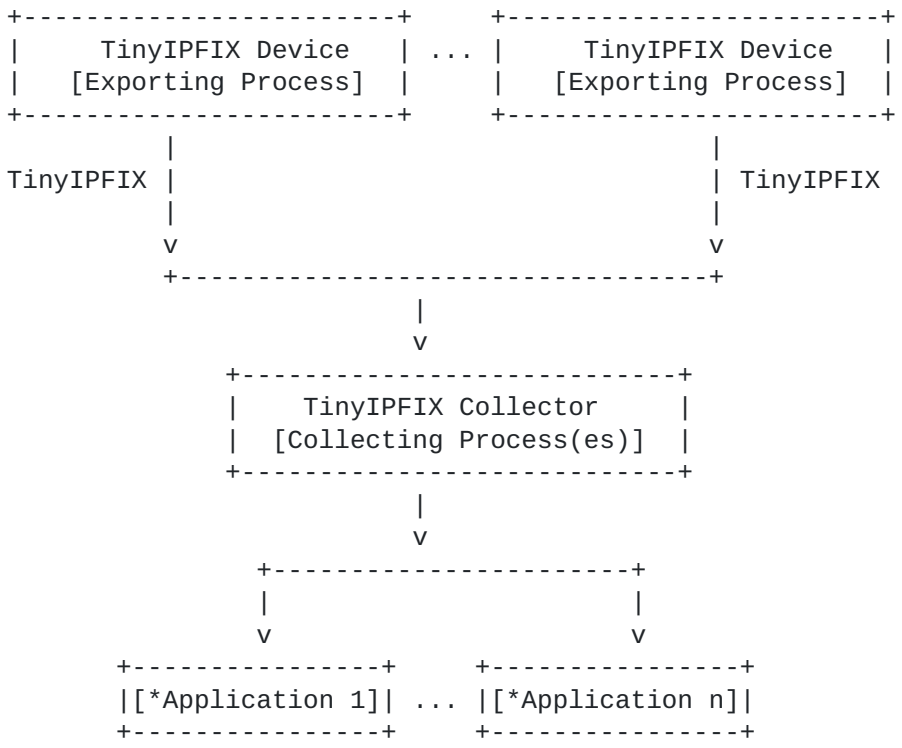

Figure 1: Direct Transmission between TinyIPFIX Devices and Applications

A TinyIPFIX Smart Meter (S.M.) receives measurement data from its internal sensors to create its TinyIPFIX Messages. Then, it encodes the results into a TinyIPFIX Message using a TinyIPFIX Exporting Process and exports this TinyIPFIX Message to one or more TinyIPFIX Collectors. The TinyIPFIX Collector runs one or more applications that process the collected sensor data. The TinyIPFIX Collector can be deployed on unconstrained devices at the constrained network border.

A second way to deploy TinyIPFIX Smart Meter can employ accumulation on TinyIPFIX Messages during their journey through the constrained network as shown in Figure 2. This accumulation can be performed by TinyIPFIX Concentrators. Such devices must have enough resources to perform the accumulation.

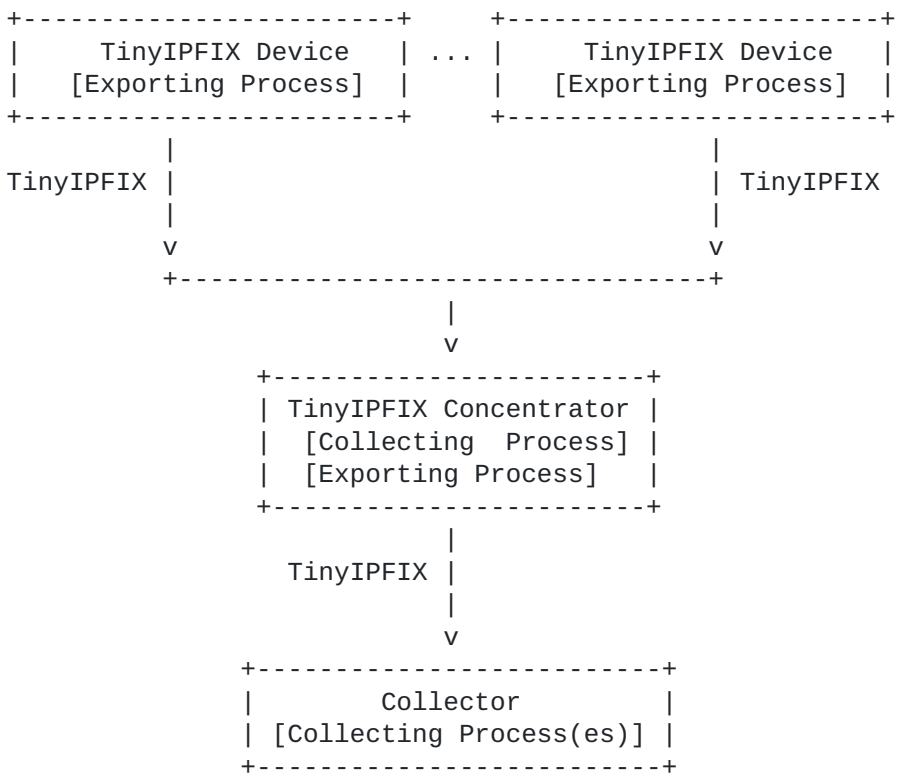

Figure 2: Accumulation of TinyIPFIX

TinyIPFIX Smart Meters send their data to a TinyIPFIX Concentrator, which needs to have enough storage space to store the incoming data. If the TinyIPFIX Concentrator is hosted in a TinyIPFIX Smart Meter, it MAY also be able to collect data from it sensors, if activated. It may also accumulate the incoming data with its own measurement data. The accumulated data can then be re-exported to one or more Collectors. In that case, the TinyIPFIX Concentrator can be viewed as receiving data from multiple Smart Meters: one locally and some remotely.

The last deployment, shown in Figure 3, employs another TinyIPFIX Mediation process.

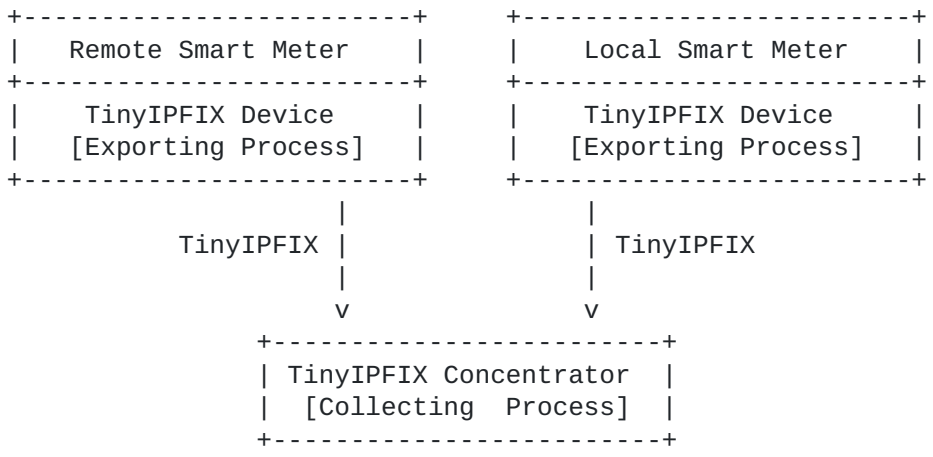

Figure 3: TinyIPFIX Mediator

In this deployment, the TinyIPFIX Smart Meters transmit their TinyIPFIX Messages to one node, e.g., the base station, which translates the TinyIPFIX Messages to IPFIX Messages. The IPFIX Messages can then be exported into an existing IPFIX infrastructure. The Mediation process from TinyIPFIX to IPFIX is described in Section 7.

6. TinyIPFIX Message Format

A TinyIPFIX IPFIX Message starts with a TinyIPFIX Message Header, followed by one or more TinyIPFIX Sets. The TinyIPFIX Sets can be either of type TinyIPFIX Template Set or of type TinyIPFIX Data Set. A TinyIPFIX Message MUST only contain one type of TinyIPFIX Set. The format of the TinyIPFIX Message is shown in Figure 4.

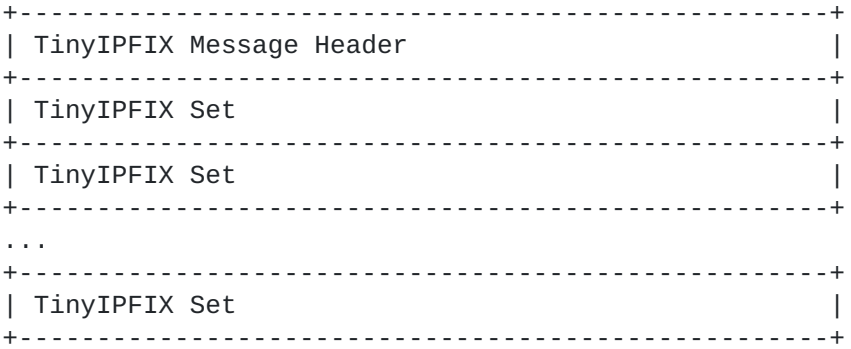

Figure 4: TinyIPFIX Message Format

6.1. TinyIPFIX Message Header

The TinyIPFIX Message Header is derived from the IPFIX Message Header, with some optimization using field compression. The IPFIX Message Header from [RFC7011] is shown in Figure 5.

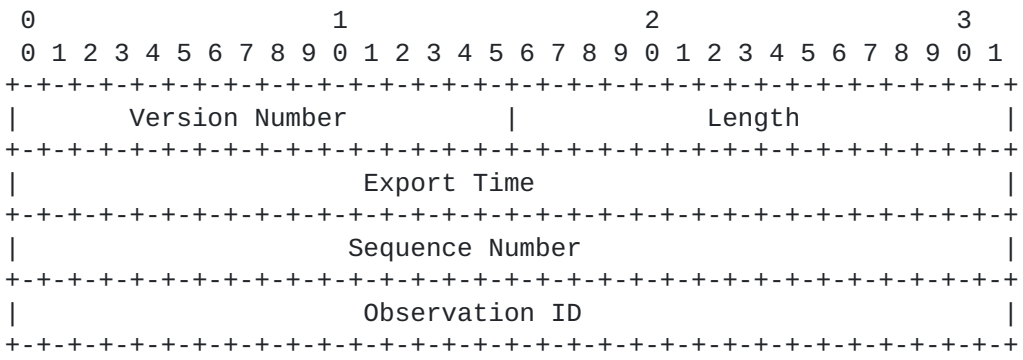

Figure 5: IPFIX Message Header

The length of the IPFIX Message Header is 16 octets, and every IPFIX Message has to be started with it. The TinyIPFIX Message Header needs to be smaller due to the packet size constraints discussed in Section 3.3. The TinyIPFIX Header consists of a fixed part of three octets as shown in Figure 6, followed by a variable part as shown in Figures 7 to 10.

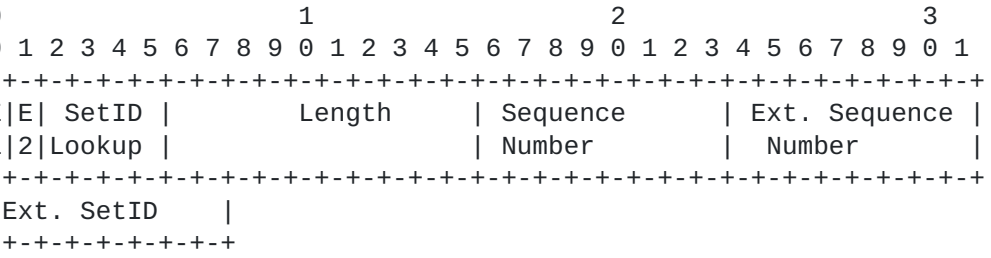

Figure 6: Format of the TinyIPFIX Message Header including Fixed and Optional Parts

The fixed part has a length of 3 octets and consists of the "E1" field (1 bit), the "E2" field (1 bit), the "SetID Lookup" field (4 bits), the "Length" field (10 bits), and the "Sequence Number" field (8 bits). The variable part has a variable length defined by the "E1" and "E2" fields in the fixed header. The four variants are illustrated in Figure 7 to Figure 10 below.

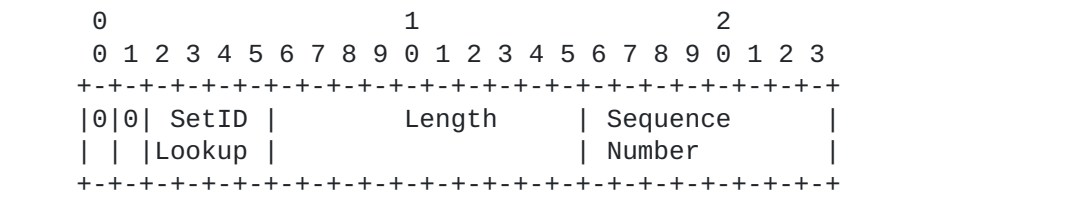

Figure 7: TinyIPFIX Message Header Format if E1 = E2 = 0

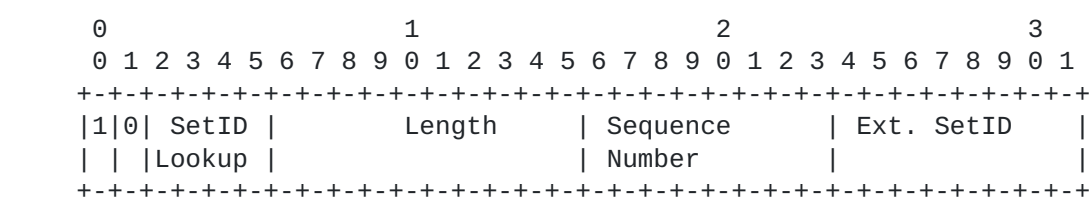

Figure 8: TinyIPFIX Message Header Format if E1 = 1 and E2 = 0

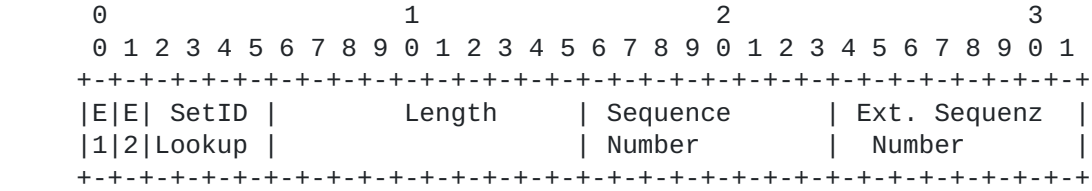

Figure 9: TinyIPFIX Message Header Format if E1 = 0 and E2 = 1

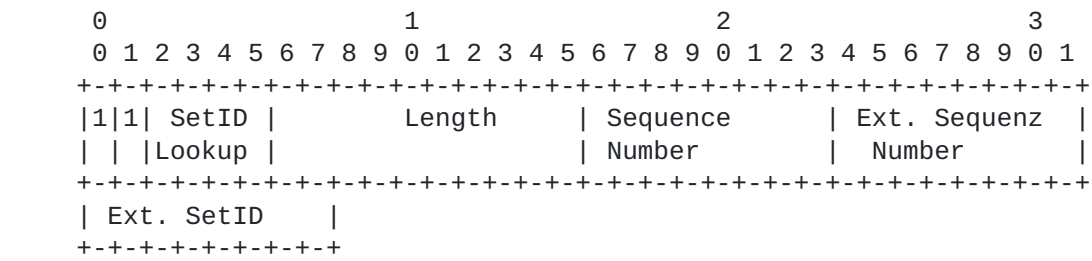

Figure 10: TinyIPFIX Message Header Format if E1 = E2 = 1

The fixed header fields are defined as follows [Kothmayr10] [Schmitt2014]:

E1 and E2

The bits marked "E1" and "E2" control the presence of the field "Ext. SetID" and the presence of the field "Ext. Sequence Number", respectively.

In case E1 = E2 = 0, the TinyIPFIX Message Header has the format shown in Figure 7. The fields Extended Sequence Number and Extended SetID MUST NOT be present.

When E1 = 1, the extended SetID field MUST be present. Custom SetIDs can be specified in the extended SetID field, setting all SetID Lookup bits to 1 (cf. Figure 8.) When evaluated, the value specified in the extended SetID field is shifted left by 8 bits to prevent collisions with the reserved SetIDs 0-255. To reference these, shifting can be disabled by setting all SetID lookup bits to 1.

Depending on the application, sampling rates might be larger than in typical constrained networks (e.g., Wireless Sensor Networks (WSNs), Cyber-Physical-Systems (CPS)); thus, they may have a large quantity of records per packet. In order to make TinyIPFIX applicable for those cases, E2 = 1 is set (cf. Figure 9). This means the Extended Sequence Number field MUST be present, offering 8-bit more sequence numbers as usual. Depending on the constrained network settings, the combination E1 = E2 = 1 is also possible, resulting in the maximum TinyIPFIX Message header shown in Figure 10 where the Extended Sequence Number field and Extended SetID field MUST both be present.

SetID Lookup

This field acts as a lookup field for the SetIDs and provides shortcuts to often used SetIDs. Four values are defined:

Value = 0; Look up extended SetID field, Shifting enabled.

Value = 1; SetID = 2 and message contains a Template definition.

Value = 2; SetID = 256 and message contains Data Record for Template 256. This places special importance on a single template ID, but, since most sensor nodes only define a single template directly after booting and continue to stream data with this template ID during the whole session lifetime, this shorthand is useful for this case.

Value = 3-14; SetIDs are reserved for future extensions.

Value = 15; look up extended SetID field, shifting enabled.

Length

The length field has a fixed length of 10 bits.

Sequence Number

Due to the low sampling rate in typical WSNs, the "Sequence Number" field is only one byte long. However, some applications may have a large quantity of records per packet. In this case, the sequence field can be extended to 16 bit by setting the E2-bit to 1.

Since TinyIPFIX packets are always transported via a network protocol, which specifies the source of the packet, the "Observation Domain" can be equated with the source of a TinyIPFIX packet. Therefore, this IPFIX field has been removed from the TinyIPFIX Header. Should an application require explicit Observation Domain information, each Data Record in the TinyIPFIX data message may contain an Observation Domain ID Information Element; see Section 3.1 of [RFC7011]. The version field has been removed since the SetID lookup field provides room for future extensions. The specification of a 32-bit timestamp in seconds would require the time synchronization across a wireless-sensor network and produces too much overhead. Thus, the "Export Time" field has been removed. If applications should require a concrete observation time (e.g., timestamp), it is RECOMMENDED to include it as a separate Information Element in the TinyIPFIX Records.

6.2. TinyIPFIX Set

A TinyIPFIX Set is a set of TinyIPFIX Template or TinyIPFIX Data Records. Depending on the TinyIPFIX Record type, the TinyIPFIX Set can be either a TinyIPFIX Template Set or a TinyIPFIX Data Set. Every TinyIPFIX Set starts with a TinyIPFIX Set Header and is followed by one or more TinyIPFIX Records.

The IPFIX Set Header consists of a 2-octet "Set ID" field and a 2-octet "Length" field. These two fields are compressed to 1 octet each for the TinyIPFIX Set Header. The format of the TinyIPFIX Set Header is shown in Figure 11.

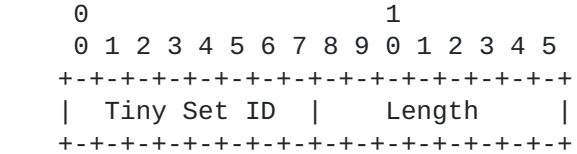

Figure 11: TinyIPFIX Set Header

The two fields are defined as follows:

TinyIPFIX Set ID

The "Tiny Set ID" identifies the type of data that is transported in the TinyIPFIX Set. A TinyIPFIX Template Set is identified by TinyIPFIX Set ID 2. This corresponds to the Template Set IDs that are used by IPFIX [RFC7011]. TinyIPFIX Set ID number 3 MUST NOT be used, as Options Templates are not supported; a TinyIPFIX Collector MUST ignore and SHOULD log any Set with Set ID 3. All values from 4 to 127 are reserved for future use. Values above 127 are used for TinyIPFIX Data Sets.

Length

The "Length" Field contains the total length of the TinyIPFIX Set, including the TinyIPFIX Set Header.

6.3. TinyIPFIX Template Record Format

The format of the TinyIPFIX Template Records is shown in Figure 12. The TinyIPFIX Template Record starts with a TinyIPFIX Template Record Header and this is followed by one or more Field Specifiers. The Field Specifier format is defined as in Section 6.4 and is identical to the Field Specifier definition in [RFC7011].

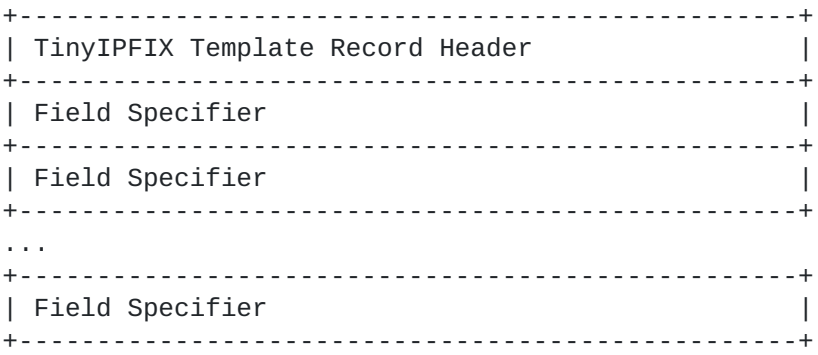

Figure 12: TinyIPFIX Template Format

The format of the TinyIPFIX Template Record Header is shown in Figure 13.

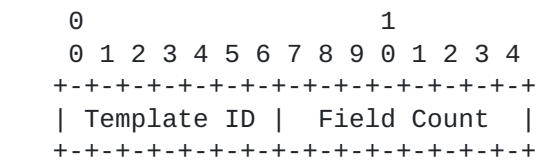

Figure 13: TinyIPFIX Template Record Header

TinyIPFIX Template ID

Each TinyIPFIX Template Record must have a unique TinyIPFIX Template ID (Comp. Temp ID) between 128 and 255. The TinyIPFIX Template ID must be unique for the given TinyIPFIX Transport Session.

Field Count

The number of fields placed in the TinyIPFIX Template Record.

6.4. Field Specifier Format

The type and length of the transmitted data is encoded in Field Specifiers within TinyIPFIX Template Records. The Field Specifier is shown in Figure 14 and is identical with the Field Specifier that was defined for IPFIX [RFC7011].

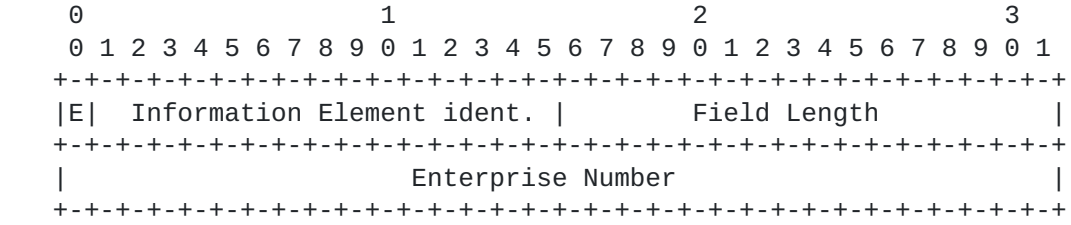

Figure 14: TinyIPFIX Data Field Specifier

Where:

E

Enterprise bit. This is the first bit of the Field Specifier. If this bit is zero, the Information Element Identifier identifies an IETF-specified Information Element, and the four-octet Enterprise Number field MUST NOT be present. If this bit is one, the Information Element Identifier identifies an enterprise-specific Information Element, and the Enterprise Number field MUST be present.

Information Element Identifier

A numeric value that represents the type of Information Element.

Field Length

The length of the corresponding encoded Information Element, in octets. Refer to [RFC7012]. The value 65535 is illegal in TinyIPFIX, as variable-length Information Elements are not supported.

Enterprise Number

IANA Private Enterprise Number of the authority defining the Information Element identifier in this Template Record.

Vendors can easily define their own data model by registering a Enterprise ID with IANA. Using their own Enterprise ID, they can use any ID in the way they want them to use.

6.5. TinyIPFIX Data Record Format

The Data Records are sent in TinyIPFIX Data Sets. The format of the Data Records is shown in Figure 15 and matches the Data Record format from IPFIX.

Figure 15: Data Record Format

7. TinyIPFIX Mediation

There are two types of TinyIPFIX Intermediate Processes. The first one can occur on the transition between a constrained network (e.g., 6LoWPAN) and the unconstrained network. This mediation changes the network and transport protocol from 6LoWPAN preferring UDP to IP/(SCTP|TCP|UDP) and is shown in Figure 16.

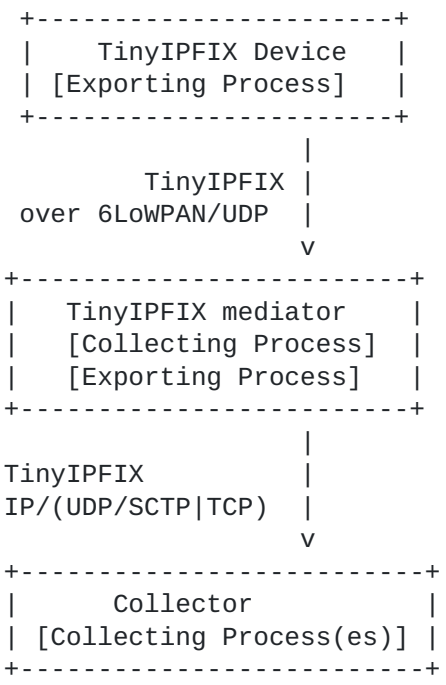

Figure 16: Translation from TinyIPFIX over 6LoWPAN/UDP to TinyIPFIX over IP/(SCTP|TCP|UDP)

The mediator removes the TinyIPFIX Messages from the 6LoWPAN/UDP packets and wraps them into the new network and transport protocols. Templates MUST be managed the same way as in the constrained environment after the translation to IP/(SCTP|UDP|TCP) (see Section 8).

The second type of mediation transforms TinyIPFIX into IPFIX. This process MUST be combined with the transport protocol mediation as shown in Figure 17.

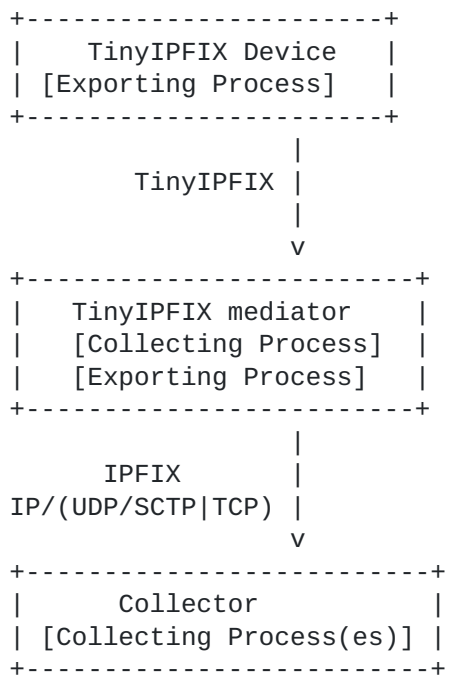

Figure 17: Transformation from TinyIPFIX to IPFIX

This mediation can also be performed by an IPFIX Collector before parsing the IPFIX message as shown in Figure 18. There is no need for a parser from TinyIPFIX to IPFIX if such a mediation process can be employed in front of an existing IPFIX collector.

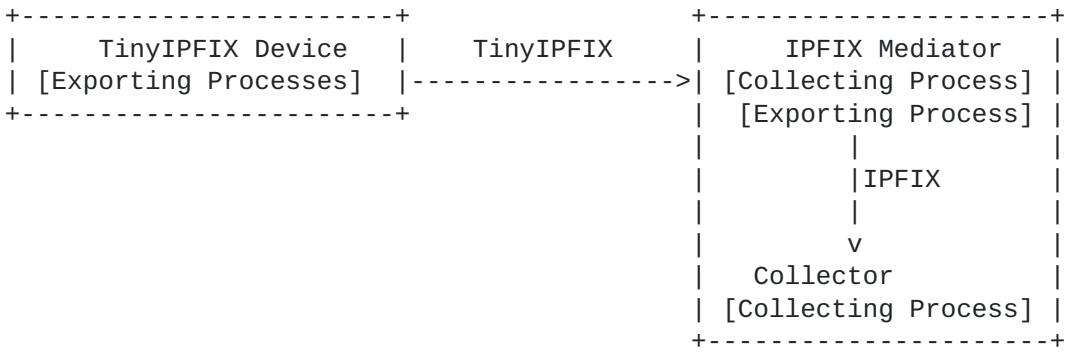

Figure 18: Transformation from TinyIPFIX to IPFIX

The TinyIPFIX Mediation Process has to translate the TinyIPFIX Message Header, the TinyIPFIX Set Headers, and the TinyIPFIX Template Record Header into their counterparts in IPFIX. Afterwards, the new IPFIX Message Length needs to be calculated and inserted into the IPFIX Message header.

7.1. Expanding the Message Header

The fields of the IPFIX Message Header that are shown in Figure 5 can be determined from a TinyIPFIX Message Header as follows:

Version

This is always 0x000a.

Length

The IPFIX Message Length can only be calculated after the complete TinyIPFIX Message has been translated. The new length can be calculated by adding the length of the IPFIX Message Header, which is 16 octets, and the length of all Sets that are contained in the IPFIX Message.

Export Time

The "Export Time" MUST be generated by the Mediator, and contains the time in seconds since 00:00 UTC Jan 1, 1970, at which the IPFIX Message leaves the Mediator.

Sequence Number

If the TinyIPFIX Sequence Number has a length of 4 octets, the original value MUST be used for the IPFIX Message. If the TinyIPFIX Sequence Number has a size of one or two octets, the TinyIPFIX Mediator MUST expand the TinyIPFIX Sequence Number into a four octet field. If the TinyIPFIX Sequence Number was omitted, the Mediator needs to calculate the Sequence Number as per [RFC7011].

Observation Domain ID

Since the Observation Domain ID is used to scope templates in IPFIX, it MUST be set to a unique value per TinyIPFIX Exporting Process, using either a mapping algorithmically determined by the Intermediate Process or directly configured by an administrator.

7.2. Translating the Set Headers

Both fields in the TinyIPFIX Set Header have a size of 1 octet and need to be expanded:

Set ID

The field needs to be expanded from 1 octet to 2 octets. If the Set ID is below 128, no recalculation needs to be performed. This is because all IDs below 128 are reserved for special messages and match the IDs used in IPFIX. The TinyIPFIX Set IDs starting with 128 identify TinyIPFIX Data Sets. Therefore, every TinyIPFIX Set ID above number 127 needs to be incremented by number 128 because IPFIX Data Set IDs are numbered above 255.

Set Length

The field needs to be expanded from one octet to two octets. It needs to be recalculated by adding a value of 2 octets to match the additional size of the Set Header. For each TinyIPFIX Template Record that is contained in the TinyIPFIX Set, 2 more octets need to be added to the length.

7.3. Expanding the Template Record Header

Both fields in the TinyIPFIX Template Record Header have a length of one octet and therefore need translation:

Template ID

The field needs to be expanded from one octet to two octets. The Template ID needs to be increased by a value of 128.

Field Count

The field needs to be expanded from one octet to 2 octets.

8. Template Management

As with IPFIX, TinyIPFIX Template Management depends on the transport protocol used. If TCP or SCTP is used, it can be ensured that TinyIPFIX Templates are delivered reliably. If UDP is used, reliability cannot be guaranteed: template loss can occur. If a Template is lost on its way to the Collector, any following TinyIPFIX Data Records that refer to this TinyIPFIX Template cannot be decoded.

Template Withdrawals are not supported in TinyIPFIX. This is generally not a problem, because most sensor nodes only define a

single static template directly after booting.

8.1. TCP/SCTP

If TCP or SCTP is used for the transmission of TinyIPFIX, Template Management MUST be performed as defined in [RFC7011] for IPFIX, with the exception of Template Withdrawals, which are not supported in TinyIPFIX. Template Withdrawals MUST NOT be sent by TinyIPFIX Exporters.

8.2. UDP

All specifications for Template Management from [RFC7011] apply unless specified otherwise in this document.

TinyIPFIX Templates MUST be sent by a TinyIPFIX Exporter before any TinyIPFIX Data Set that refers to the TinyIPFIX Template is transmitted. TinyIPFIX Templates are not expected to change over time in TinyIPFIX and, thus, they should be pre-shared. TinyIPFIX Devices have a default setup when deployed; after booting, they announce their TinyIPFIX Template directly to the network and MAY repeat it if UDP is used. Hence, a TinyIPFIX Template that has been sent once MAY NOT be withdrawn and MUST NOT expire. If a TinyIPFIX Smart Meter wants to use another TinyIPFIX Template, it MUST use a new TinyIPFIX Template ID for the TinyIPFIX Template.

While UDP is used, reliable transport of TinyIPFIX Templates cannot be, guaranteed and TinyIPFIX Templates can be lost. A TinyIPFIX Exporter MUST expect TinyIPFIX Template loss. Therefore, it MUST re-send its TinyIPFIX Templates periodically. A TinyIPFIX Template MUST be re-sent after a fixed number N of TinyIPFIX Messages that contain TinyIPFIX Data Sets referring to the TinyIPFIX Template. The number N MUST be configured by the application developer. Retransmission and the specification of N can be avoided if TinyIPFIX Exporter and TinyIPFIX Collector use pre-shared templates.

9. Security Considerations

The same security considerations as for the IPFIX Protocol [RFC7011] apply.

10. IANA Considerations

This document does not require any IANA actions.

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <https://www.rfc-editor.org/info/rfc4944>.

[RFC5153] Boschi, E., Mark, L., Quittek, J., Stiemerling, M., and P. Aitken, "IP Flow Information Export (IPFIX) Implementation Guidelines", RFC 5153, DOI 10.17487/RFC5153, April 2008, <https://www.rfc-editor.org/info/rfc5153>.

[RFC5470] Sadasivan, G., Brownlee, N., Claise, B., and J. Quittek, "Architecture for IP Flow Information Export", RFC 5470, DOI 10.17487/RFC5470, March 2009, <https://www.rfc-editor.org/info/rfc5470>.

[RFC5982] Kobayashi, A., Ed. and B. Claise, Ed., "IP Flow Information Export (IPFIX) Mediation: Problem Statement", RFC 5982, DOI 10.17487/RFC5982, August 2010, <https://www.rfc-editor.org/info/rfc5982>.

[RFC6183] Kobayashi, A., Claise, B., Muenz, G., and K. Ishibashi, "IP Flow Information Export (IPFIX) Mediation: Framework", RFC 6183, DOI 10.17487/RFC6183, April 2011, <https://www.rfc-editor.org/info/rfc6183>.

[RFC7011] Claise, B., Ed., Trammell, B., Ed., and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", STD 77, RFC 7011, DOI 10.17487/RFC7011, September 2013, <https://www.rfc-editor.org/info/rfc7011>.

[RFC7012] Claise, B., Ed. and B. Trammell, Ed., "Information Model for IP Flow Information Export (IPFIX)", RFC 7012, DOI 10.17487/RFC7012, September 2013, <https://www.rfc-editor.org/info/rfc7012>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

11.2. Informative References

[Advantic] ADVANTIC SISTEMAS Y SERVICIOS S.L., <https://www.advanticsys.com/>, 2017.

[GreatDuck] Mainwaring, A., Polastre, J., Szewczyk, R., Culler, D., and J. Anderson, "Wireless Sensor Networks for Habitat Monitoring", In Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications ACM, pp. 88-97, DOI 10.1145/570738.570751, 2002.

[Harvan08] Harvan, M. and J. Schoenwaelder, "TinyOS Motes on the Internet: IPv6 over 802.15.4 (6LoWPAN)", DOI 10.1515/piko.2008.0042, December 2008.

[IRIS] Memsic, "Data Sheet IRIS", 2017, <http://www.memsic.com/userfiles/files/Datasheets/WSN/IRIS_Datasheet.pdf>.

[Kim07] Kim, S., Pakzad, S., Culler, D., Demmel, J., Fenves, G., Glaser, S., and M. Turon, "Health monitoring of civil infrastructures using wireless sensor networks", Proceedings of the 6th international conference on Information processing in sensor networks (IPSN 2007), Cambridge, MA, ACM Press, pp. 254-263, DOI 10.1145/1236360.1236395, April 2007.

[Kothmayr10] Kothmayr, T., "Data Collection in Wireless Sensor Networks for Autonomic Home Networking", Bachelor Thesis, Technical University of Munich, Munich, Germany, January 2010.

[openMote] openMote Technologies S.L., 2017, <http://openmote.com>.

[Schmitt09] Schmitt, C. and G. Carle, "Applications for Wireless Sensor Networks", Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies and Applications, Edited by Antonopoulos N., Exarchakos G., Li M., and A. Liotta, Information Science Publishing, Chapter 46, pp. 1076-1091, ISBN: 978-1615206865, 2010.

[Schmitt2014]
Schmitt, C., Kothmayr, T., Ertl, B., Hu, W., Braun, L., and G. Carle, "TinyIPFIX: An efficient application protocol for data exchange in cyber physical systems", Computer Communications, ELSEVIER, Vol. 74, pp. 63-76, DOI 10.1016/j.comcom.2014.05.012, 2016.

[Schmitt2017]
Schmitt, C., Anliker, C., and B. Stiller, "Efficient and Secure Pull Requests for Emergency Cases Using a Mobile Access Framework", Managing the Web of Things: Linking the Real World to the Web, Edited by Sheng, M., Qin, Y., Yao, L., and B. Benatallah, Morgen Kaufmann (imprint of Elsevier), Chapter 8, pp. 229-247, ISBN: 978-0-12-809764-9, 2017.

[SMPC04] Szewczyk, R., Mainwaring, A., Polastre, J., and D. Culler, "An analysis of a large scale habitat monitoring application", Proceedings of the 2nd international conference on Embedded networked sensor systems (SenSys 04), DOI 10.1145/1031495.1031521, November 2004.

[TelosB] Memsic, "Data Sheet TelosB", 2017, <http://www.memsic.com/userfiles/files/DataSheets/WSN/telosb_datasheet.pdf>.

[Tolle05] Tolle, G., Polastre, J., Szewczyk, R., Culler, D., Turner, N., Tu, K., Burgess, S., Dawnson, T., Buonadonna, P., Gay, D., and W. Hong, "A macroscope in the redwoods", Proceedings of the 3rd international conference on Embedded networked sensor systems (SenSys 05), DOI 10.1145/1098918.1098925, November 2005.

Acknowledgments

Many thanks to Lothar Braun, Georg Carle, and Benoit Claise, who contributed significant work to earlier draft versions of this work, especially to the document titled "Compressed IPFIX for Smart Meters in Constrained Networks".

Many thanks to Thomas Kothmayr, Michael Meister, and Livio Sgier, who implemented TinyIPFIX (except the mediator) for TinyOS 2.x and Contiki 2.7/3.0 for 3 different sensor platforms (IRIS, TelosB, and OpenMote).

Authors' Addresses

Corinna Schmitt
University of Zurich
Department of Informatics
Communication Systems Group
Binzmuehlestrasse 14
Zurich 8050
Switzerland

Email: schmitt@ifi.uzh.ch

Burkhard Stiller
University of Zurich
Department of Informatics
Communication Systems Group
Binzmuehlestrasse 14
Zurich 8050
Switzerland

Email: stiller@ifi.uzh.ch

Brian Trammell
Swiss Federal Institute of Technology
Gloriastrasse 35
Zurich 8092
Switzerland

Email: ietf@trammell.ch