Towards Effective and Efficient Data Management in Embedded Systems and Internet of Things

Abstract

The majority of today low-end and low-cost embedded devices work in dynamic environments under several constraints such as low power, reduced memory, limited processing and communication, etc. Therefore, their data management is critical. We introduce here a general method for data representation, storage, and transmission in embedded systems based on a compact representation scheme and some heuristics. This method has been implemented, tested, and evaluated within a vehicle tracking system that uses an in-house very low cost microcontroller-based telemetry device, which provides for near-real-time remote vehicle monitoring, energy consumption, ubiquitous health, etc. However, our method is general and can be used for any type of low-cost and resource-constrained embedded device, where data communication from the device to the Internet (or cloud) is involved. Its efficiency and effectiveness are proven by significant reductions of mobile data transmitted, as our case study shows. Further benefits are reducing power consumption and transmission costs.

Keywords: mobile device; data storage, compression, and communication; telemetry; remote monitoring; embedded systems; pervasive computing; Internet of Things.

1. Introduction

Information Technology and Communications have become ubiquitous in everyday life and embedded systems have a prevalent role in this process. From smart cities, smart environment and home automation, to industrial control, logistics, smart agriculture, and health and wellness monitoring, to name just a few opportunities, it seems that under the pervasive computing paradigm there is no limit on what we can do to improve various human activities [6, 12, 14, 17, 21]. Ubiquitous sensing allows measuring, inferring, and understanding environmental indicators, from delicate ecologies and natural resources to urban environments. The omnipresent sensors provide for creation of a communicating–actuating network that fuels the Internet of Things (IoT), wherein sensors and actuators blend seamlessly with the environment around us, and the information is shared across platforms in order to develop a common operating picture [12]. Under this paradigm, computers are expected to know everything about things by making sense of the data they have collected independently from humans and by being able to track and count everything, contributing this way to significant reductions with regard to waste, loss, and cost. In terms of the scientist that has coined the IoT term, we would know when things needed replacing, repairing or recalling, and whether they were fresh or past their best. The IoT has indeed the potential to change the world, even more than the Internet did [1, 25]. Building up on that, pervasive computing is expected to make life simpler via digital environments that are able to sense, adapt, and respond to human needs, and in which devices can act as portals into various application-data spaces, not just as repositories of custom software to be managed by users. In such environments, an application is a means by which a user performs a task, and not just software for exploiting a particular device's capabilities. This way, a computing environment becomes *an information-enhanced physical space*, not just a virtual one in which software is

stored and run [19]. Pervasive computing will have a strong impact on members of human societies with respect to both life and work styles, especially regarding information exchanging and sharing [11, 13, 25].

Despite great expectations, nowadays pervasive computing supporting technologies face significant technical issues. For example, many types of devices have strong limitations on memory usage and processor performance, as well as tight constraints on power consumption. In addition, they are expected to be able to handle power shortages, while the applications must be able to resume seamlessly after a shutdown. Furthermore, the footprint of any kind of running hardware and software is to be reduced as much as possible. Therefore, having the two paradigms, i.e. pervasive computing and Internet of Things, in daily life is not yet easy. The specific particular solutions in existence are not customizable, the costs are often prohibitive, and dependability is generally limited. A good example gives the automotive industry, where each car receives an ever-growing number of electronic control units (70-100 per car), and, as a result, software complexity escalates dramatically. Hence, the current design, validation, and maintenance processes and tools can no longer ensure sufficiently reliable systems, at affordable costs, and industries cannot capitalize on the huge potential that emerging hardware and IT&C technologies offer [2, 14]. Moreover, the majority of nowadays embedded systems work in dynamic environments, where the particularities of the computational load cannot generally be predicted in advance. Hence, embedded systems are inherently real-time systems that are required to work under several resource constraints imposed by particular characteristics such as *size, weight, energy consumption, equipment cost, data communication costs, maintenance costs* etc., showing therefore a dynamic behavior. Still, *timely responses to events* have to be provided within precise timing constraints in order to guarantee a desired level of performance. Consequently, efficient resource management is critical for embedded systems [3, 16]. Many of these systems are *data-dominated* and experiments have shown that a significant part of the power consumption is due to *data storage and transfer* [4, 9]. The escalating need for high performance and huge capacity memories of today embedded systems has led to the widespread adaptation of flash memory as main data storage. Therefore, data management on this kind of storage has become critical for mobile embedded applications. As reading, data writing or erasing on flash memories is performed radically different from magnetic disks, new data management approaches are necessary. However, *complex compressions algorithms cannot be used due to the limitations of the IoT embedded devices* (low power requirements, reduced memory, and extremely limited processing and communication capabilities). Hence, simpler but still efficient and effective solutions must be worked out [3, 4, 5] [9] [15, 16, 17, 18] [20] [22].

Further on, we present related work that is somewhat similar to ours. In [16], the authors introduce a scale-downed relational DBMS, called LGeDBMS, which has been specifically designed for data management and easy access to data in embedded mobile systems that use flash memories. LGeDBMS optimizes the flash memory based on the Log-structured File System design principle, has a compact size that is appropriate for consumer electronics appliances, and implements a transaction management mechanism that comply only with atomicity and durability. In [5], the authors investigate the performance issues of flash-memory storage systems by using a dynamic striping architecture and I/O parallelism to speed-up a flash-memory storage system. A bold approach is taken in [17], where the authors point out that traditional data base management systems are not suited for embedded systems due their special requirements and limited resources. They propose to implement highly customizable data management systems that can be created with a Software Product Line approach, i.e. a concrete instance of a DBMS is derived by composing features of the DBMS product line that are needed for a given application scenario. They show that in embedded systems also non-functional properties, such as memory consumption, have to be considered when creating a particular DBMS instance. Further details are available in [18]. Another impressive effort is presented in [20], where the authors present a novel architecture for underwater sensor networks to be used for long-term monitoring of coral reefs and fisheries. The sensor network consists of static and mobile underwater sensor nodes that have various sensing capabilities (cameras or devices to measure water temperature and pressure). The

mobile nodes can locate and hover above the static nodes for data muling and perform network maintenance functions such as deployment, relocation, and recovery.

Despite the significant challenges, the premises that the pervasive computing paradigm becomes a reality exist, especially due to the opportunities provided by embedded systems and IoT. On one hand, the technology evolves with an incredible pace, new architectures and systems appear every day, while on the other, our society is more and more prepared for this major paradigm shift, at various levels. This work aims to make a contribution to that by introducing *a general method for data representation, storage, and transmission in embedded systems and IoT*, which is based on a compact representation scheme and some heuristics. We have already implemented, tested, and evaluated this method within a vehicle tracking system developed in-house over the last years, experimenting through several iterations due to advances in the supporting technologies [7, 8]. The system is in use on more than 500 cars for over 10 years now and we are currently in the process of adapting it for other use scenarios, i.e. access control, remote data and energy consumption monitoring. The system uses an in-house developed *very low cost microcontroller-based telemetry device* that implements this method and that provides for near-real-time remote monitoring of mobile vehicles, energy consumption, ubiquitous health etc. The real data samples presented here are obtained while monitoring remotely mobile vehicles, over long distances, using low bandwidth mobile data communication. Having *near real-time data* from the mobile device and *as much information as possible* about the vehicle, while *keeping low the amount of data transmitted between the device and the Internet* have been some of our main goals. One of *the biggest challenges* in developing this system has been to reduce *the amount of transferred data* while using a very limited device (low computational performance, little available memory, low communication bandwidth). Our solution consists of temporarily keeping the data on the device, transmitting it when possible, allowing retransmission in case of communication errors, and storing it in a compact/compressed form to avoid both wasting storage space and transmitting redundant data. The basic idea is to carefully transform the data in such a way that *a minimal number of bits are necessary to encode it without losing any information*. *The method introduced here is general* and it can be adapted easily, using the appropriate heuristics, for any type of *low-cost and resource constrained embedded or IoT device*, where sensory data communication from the device to the Internet (or cloud) is involved. The same is true for the compact representation scheme that reduces the amount of data stored and transferred, lowering this way both the costs of data processing and transferring. Moreover, the *time to transmit* is also reduced and, consequently, the total energy used for communication is reduced as well, implicitly increasing the battery's life time because it is a well-known fact that most of the power is used during transmissions.The efficiency and effectiveness of the method are illustrated with several evaluations based on real data samples.

The structure of the paper is as follows: the next section gives an overview of our system. Section 3 and 4 include, respectively, our method for data representing, storing, and transmitting and our compact data representation scheme. Several evaluations of the method are included in Section 5. The last section includes some conclusions and future work ideas.

2. Overview of Gipix – a Vehicle Tracking System

In this section, we present briefly our vehicle tracking system (called Gipix). The system can also provide statistical information about different aspects of the recorded data, e.g. the driver's acceleration, braking habits, and driving style, the speed variations, etc. More details about Gipix and our experimenting with it may be found in [7, 8, 23, 24]. Gipix is a system for near real-time vehicle tracking, which offers very accurate positioning based both on state-of-the-art GPS technology and GSM/GPRS data transmission and, at the same time, it can gather and process multiple sensor data from the vehicle. The system's core consists of a data server that processes the maps for the main cities and roads in our country, the monitored vehicles and their tracks, the drivers' related information, various critical events, some predefined tracks, specific reports, etc. It can be used both for individual vehicles and fleets. Gipix collects the data of interest by using a GPS-based embedded device installed on each

monitored vehicle that is able to automatically transmit the vehicle positions and to signal various critical events to both the server and the interested users. To overcome the limitation of the commercially available GPS-based tracking solutions with regard to customization the system has been developed in-house. Its main capabilities include *one-second acquisition interval (for position, speed, and heading), high sensibility (the antenna is able to work in difficult conditions), interconnection with other applications or devices, adaptability to users' needs, local storage of data for areas which are not GSM covered, and positioning without GPS signal if the antenna fails based on the position of the GSM cells.* A customized version can be used for *remote telemetry monitoring of different sensors in a fixed configuration* (position data is ignored and only sensor data is transmitted) that can be used in other pervasive computing applications such as energy consumption or ubiquitous health monitoring, access control, etc. The system's architecture includes a large number of mobile devices that communicate over the Internet to one or several data servers by sending near-real-time data or by retransmitting lost data (Fig. 1). Data is sent over a low speed communication channel, using GPRS, because it offers the best geographical coverage over other communication methods at a reasonable price tag. The end user interacts with the system over the Web using a graphical user interface. The Web server communicates directly to the data server by means of a backend application. The architecture of the mobile embedded device consists of a central microcontroller that is in charge with monitoring all the connected sensors mounted on the device or connected to the vehicle (Fig. 2a). It also handles the data storage and communication between the device and the Internet using a GPRS modem. The accurate location of a vehicle is established once every second using a high-sensitivity GPS receiver. The device is powered either from the vehicle battery (12/24V) or from its own internal backup rechargeable battery, a high-capacity Lithium-Polymer battery, which is capable of powering the device for several days in the absence of the car battery. The device is able to detect situations in which a car has its own battery disconnected or it is moved to a different location. Its current version called Gipix-112 is shown in Fig. 2b.

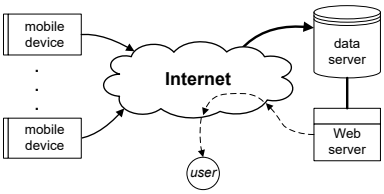
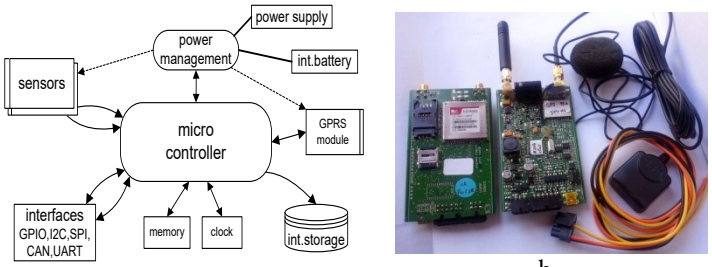


Fig. 1. Gipix vehicle tracking system – main architecture



a. b.

Fig. 2. Gipix embedded device: a. architecture; b. physical implementation.

The tracking device is built around a very low power 16-bit *microcontroller* from the PIC24F Microchip family with integrated 256 Kbytes flash and all the necessary peripheral interfaces embedded (inputs, outputs, serial ports, ADC converters, etc.). It has a number of sensors that allow gathering of abundant information about both the vehicle and the driver, and also detection of any abnormal situation. Some of the existing sensors determine or are concerned with vehicle battery voltage and ignition, internal device temperature, RFID, 1-wire, proximity, 3D acceleration, and internal tamper. The tracking device has one communication modules that provide for transmission of information about position, sensor values, events, etc. to the central tracking server or directly to the driver's mobile phone. Alternatively, some other means of communication are also supported. Low-range Bluetooth can be used to relay different information to the driver's mobile phone, while driving or to download all the tracking data from the device in cases where either GPRS is not available or is missing from the device. Using Wi-Fi communication can provide periodic synchronization or firmware updates when the vehicle is in a garage. It also provides a power management mode that enables a low power mode of the device necessary to save battery power when the vehicle is not operating.

3. Method for Data Representation, Storage, and Transmission

Based on our main goal of *near-real-time remote vehicle monitoring using a very low cost microcontroller-based telemetry device* that provides for *efficient and effective data management*, our system had to fulfill the following data management requirements:

- Dealing with *a large number of mobile devices*;
- Ability *to store and manipulate various data* like: device identifier, GPS data (position, speed, latitude, longitude, altitude etc.), GSM data (cell information, error rate, etc.), vehicle data (battery voltage, error sensors, etc.), sequence number, and so on;
- Capability to store and transfer *near-real time data at very small time intervals (one second), for a very long time*;
- *The time needed to process the current data* (store, transmit, retrieve, retransmit) *is minimal*, such that the device is available to process the data from the next time stamp;
- *Minimization of data to be transferred*, which resulted in three more low-level requirements: *data pre-processing* at the device level; *reducing the communication overhead* for data as the device identifier, source and destination addresses, and other network and Internet related communication data; *reducing the security/privacy overhead*, required by encryption and authentication.

As most existing solutions for embedded data storage and management require some sort of lightweight file system or scaled-down database management system, the main drawbacks of using them consist of adding to the complexity of the storage component, increasing the time needed to store/retrieve the data, having a non-deterministic response time, and, potentially, generating data inconsistencies (e.g. in case of a power loss). Since our device need to store the data temporarily, without processing it further and due to its very critical time constraints, such increased complexity is not justified. The solution currently in place in Gipix implements a *data storage and representation method* that provides for:

- *Reducing the data size* by compression and/or applying an efficient encoding scheme (i.e. not storing/transferring redundant or unnecessary data, using more compact representations and using smarter sensors);
- *Delaying the data transmission*, i.e. the data from the sensors is not transferred immediately after reading it;
- *Transferring data in bulks* to reduce communication overhead;
- *Storing the data temporarily on the device*'s internal storage memory;
- Transferring data while keeping track of packet sequence numbers for *easy management of retransmission;*
- *Using a minimal number of bits to encode the data without losing any information*.

Further, we present detailed information about the data representation, storage, and transfer between the embedded device and the main server of Gipix. All the tracking data generated on the device is, at first, stored on its internal storage, a relatively large capacity microSD flash memory card (2 GBytes). If the GPRS connection is available, the data is further transmitted to the central storage server. In cases when the connection is not available or any transmission errors occur, the data is re-transmitted at a later time (see Fig. 3).

According to the above method, the application running on the device includes two tasks related to the data management: one for saving the current data and one for retrieving the stored data. The first one creates the data packet to be stored on the local storage (the flash memory) and to be further transmitted to the main server. If the transmission fails due to various circumstances, for example an area with no mobile data communication coverage or an error in transmission, the data can be later retrieved from the local flash memory and then re-transmitted to the server (see Fig. 4). Each data packet is tailored to fit in one physical sector of the flash memory for dual reasons: minimization of the data saving/retrieving time and easiness of data retrieval. Additionally, each data packet is given a time stamp (sequence number or packet number) to facilitate both data retrieving and identifying of missing packets.

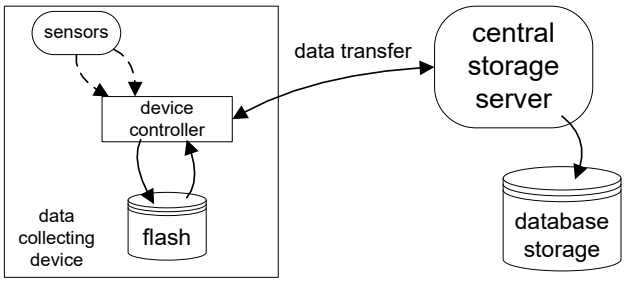


Fig. 3. Gipix–bird's eye view: data storage and transmission

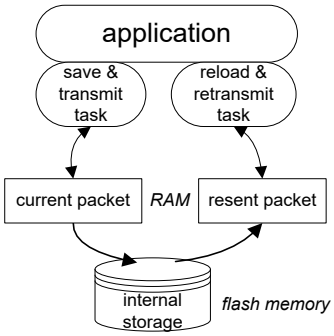


Fig. 4. Application tasks for data saving/retrieving on/from the device

The second task is responsible with data retrieving and retransmitting. Two scenarios are possible. The first one is when a certain data segment was not transmitted at all, in which case this task will try to resend it when a new connection to the server is established. This situation mainly occurs after a period of time when there was no network coverage and no active connection with the server could be established. Then the data sequences not marked as sent are retrieved from the local memory and resent to the server. The second scenario happens when even if some data was already sent, the server specifically asks the device to resend a specific set of data sequences. This might happen because of either communication errors occur or data is lost during transmission or data is received by the server with errors. The use of appropriate data management techniques (to be presented in the next section) allows data storing for a very long time. Depending on the sensory data and on the vehicle's operating time intervals for which the data is recorded and stored, the device's internal flash memory can store all recorded data for *up to 10 years*, thus acting like a very long term data recording device. In special cases, all the stored data on the device can also be directly downloaded by using a wired serial/USB connection between the device and a computer.

4. Compact Data Representation

We illustrate further on how the data is encoded and stored on the embedded device and how it is transmitted between to the main server, exemplifying with particular data sets. The main heuristics used and some lessons learned are also shown. The following information types are of interest and, therefore, they are stored and also transferred from the device to the server: device identification data (device ID or IMEI - the mobile equipment identification); time stamp data: date and time; position data: latitude, longitude, altitude; movement data: speed, heading, position error, number of satellites; other sensor data: battery voltage, temperature, etc.; control checksum for data integrity verification; other data needed for the particular applications, e.g. other sensor or fusion data. Further on, we consider the following smaller set of data transferred between the device and the server: *device identification (device id), time stamp (tstamp), latitude, longitude, altitude (position), heading (direction of movement), speed, positioning error (hdop - horizontal dilution of precision), number of satellites in view (sats), and a valid fix for the position, if available (fix)*. This set could also include data from additional sensors. Formatted as *a text string*, the sample data set for an interval of 3 seconds (3 measurements) could look like in Table 1 (the first line is a comment describing the data fields). Each line represents one set of measurements taken at the specified time stamp. The length of each line can be different, depending on the representation of values as text strings (larger numerical values require more digits to represent) and can vary approximately between 68 and 78 bytes per line. In case of a better *standard binary data representation* for the same data, we would have something similar to the representation in Table 2. Each data set will describe the position for exactly one second (identified by the time stamp). If knowing more than one position at the server is necessary, storing and sending this amount of data for each of the required positions is needed.

Table 1. Data with text representation

```
;type,device,date,time,latitude,longi
tude,altitude,speed,heading,err,sat
$DATA,1234578,2013-09-20,
13:40:20,40.123456,25.123456,345.0,55
,12,10.4,7
$DATA,1234578,2013-09-20,
13:40:21,40.124468,25.123589,346.0,50
,119,0.4,8
$DATA,1234578,2013-09-20,
13:40:22,40.124600,25.123600,345.0,45
,115,0.4,7
      Data length: 3*≈72 bytes
```

Table 2. Data with binary representation

descr.	type	dev_id	tstamp	latitude	longit	altit	head	speed	hdop	sats	fix
bytes	1	4	8	4	4	2	2	1	2	1	1
type	byte	integer	long int	float	float	sh int	sh int	byte	sh int	byte	byte
value1	1	12345678	1379684420	40.123456	25.123456	345	120	55	10.4	7	3D
value2	1	12345678	1379684421	40.124468	25.123589	346	119	50	9.3	8	3D
value3	1	12345678	1379684422	40.124600	25.123600	345	115	45	9.9	7	3D

Length = 3***30 bytes**

In addition, each transferred data set has an additional communication overhead necessary to be able to transmit that data to the specific server on the Internet (IP address, port number). There are two types of Internet Protocols (IP): TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). TCP is connection oriented meaning that once a connection is established data can be sent in both ways, while UDP is a connectionless protocol. TCP is better suited for applications that require high reliability and transmission time is relatively less critical, whereas UDP is suitable for applications that need fast and efficient transmission. UDP's stateless nature is also useful for servers that answer small queries from huge numbers of clients. One of the UDP's downsides is that it has no inherent order as all packets are independent of each other. Dealing with packet failures, packet re-ordering, or multiple sources is made at the application level both on the mobile device and on the data server. TCP is slower that UDP because in UDP error recovery is not attempted at protocol level, only simple error checking and discarding. UDP is a small transport layer designed on top of IP in which there is neither ordering of messages nor tracking connections. The choice of using UDP instead of TCP is mainly because UDP is much more lightweight, the retransmission of lost packets being handled at application level with better control. IP header is 20 bytes, without options. TCP header is 20 bytes without options and UDP header is 8 bytes (Fig. 5.).

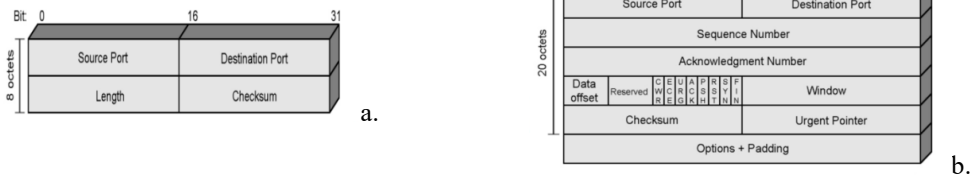


Fig. 5. Header format and size for a) UDP and b) TCP packets

In our case, UDP is used for transferring the data, so this overhead amounts to a total of 28 bytes. In the case of a mobile vehicle, the data for each second is required to be able to accurately describe the status, movement, and behavior of the vehicle (speed, acceleration, sensory data, etc.). Let us consider an amount of 30 bytes of data for each second. Then the total number of bytes required for transmitting N data units using the binary representation is (D_{comm} is the communication overhead of 28 bytes and $D_{data.bin}$ is the actual data: 30 bytes):

$$T_{bin}(N) = N*(D_{comm}+D_{data.bin}) \quad (1)$$

In such a case, the amount of data to be transferred would be very large – e.g. for a one second resolution data, one needs a total of 2.6 million data sets per month (30 days*24 hours*3600 seconds). For each data set, one needs 58 bytes (30 bytes for the actual data and 28 bytes for the overhead), which is equivalent to a total of about 150 MegaBytes of data transferred per month, i.e. T_{bin}(1 month) = 150 Mbytes. This amount corresponds just to the transfer from the device to the server, but the confirmation packet backwards, for each packet received, is also necessary. Having tens of thousands of devices sending data to the server, a storage space in the orders of many TerraBytes would be necessary just to store the raw data.

Further on, we present *a more compact data representation* in accord with the method in the previous section. One prerequisite is that the precision required by each data type will dictate the number of bits necessary for each particular data. For example, considering that the

timestamp for each packet has a resolution of one second and represents the date and time of the event, one do not necessarily needs to represent it as seconds from January 1st, 1970 (Unix time), but as seconds since one particular version of the system is running (e.g. January 1st, 2006) – this could potentially save a few bits in representing the time. For representing latitude information with a precision of six digits after the decimal point (e. g. 45.123456) results in a worst case positioning error (depending on the position on Earth) equivalent to 0.1 meters. The full range of latitudes is -90.000000 ... +90.000000 that is equivalent to the range 0...180000000 (if one deletes the decimal point and converts to positive values). In this approach, only 28 bits are necessary (2^28=268435456). Similarly, for representing longitudes ranging from -180.000000 ... +180.000000, 29 bits are needed (2^29=536870 912). The maximum ranges and compact encoding for the other data fields are shown in Table 3.

Table 3. Storage requirements for each data type

	data representation				data variation - delta (per second)			
	range	min	max	bits	range	min	max	bits
tstamp	100 years	0	4294967295	32	30 seconds	0	31	5
latit.	-90.000000 … +90.000000	0	268,435,455	28	-0.000500 … +0.000500	0	1023	10
longit.	-180.000000 … +180.000000	0	536,870,911	29	-0.000750 … +0.000750	0	2047	11
altit.	8000 meters	0	8,191	13	-32 … +32 meters	0	63	6
head	0 … 360 degrees	0	511	9	not necessary			-
speed	200 km/h	0	2,047	11	not necessary			-
hdop	0.0 … 25.0 meters	0	255	8	0.0 … 25.0 meters	0	255	8
sats	0 .. 31 satelites	0	31	5	not necessary			-
fix	yes/no	0	1	1	not necessary			-

The data to be stored (the measured values at time t) are first compacted as shown above and this is the fixed part of the storage and transfer unit (*fixed value stored* in Table 4.). Instead of sending the same data amount for each subsequent time stamp, only variations of individual data fields are sent (*value var1 stored*, *value var 2 stored*, … in Table 4). The rationale behind this is heuristic being based on the observation that for slow changing data one needs lower data amount to represent the variations between adjacent values than to represent the data itself. Therefore, only the delta variations from one time stamp to the next are computed and a lower amount of bits to represent them is used. This is done differently for each data field depending on both their meaning and possible ranges in variation. The corresponding *compact binary data representation* for the previous 3 second interval data example and the corresponding data transmission sequences are illustrated in Table 4 and, respectively, Fig. 6.

Table 4. Data with compact binary representation

descr.	pack_no	dev_id	tstamp	latit	longit	altit	head	speed	hdop	sats	fix
bits	32	32	32	28	29	13	9	11	8	5	1
fixed value repres.	102030	12345678	1379684420	40.123456	25.123456	345	120	55.4	10.4	7	2D
fixed value stored	**102030**	**12345678**	**1379684420**	**130123456**	**205123456**	**345**	**120**	**554**	**104**	**7**	**1**

F = Length (fixed) = **25 bytes**

descr.	delta tstamp	delta latitude	delta longitude	delta altitude	hdop
bits	5	10	11	6	8
value var1 repres.	1	0.001012	0.000133	1	9.3
value var1	**1**	**1512**	**1113**	**33**	**93**
value var2	1	0.000132	0.000121	-1	9.9
value var2	**1**	**632**	**1121**	**31**	**99**

V = Length (variation) = 2***5 bytes**

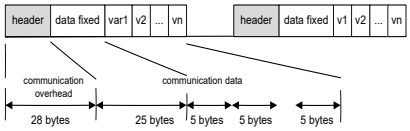


Fig. 6. Data transmitted in the compact format

5. Evaluation of the method

In the case of the compact binary data representation, for example, a total of 28+25+9*5 bytes=98 bytes needs to be transferred to send the equivalent of 10 consecutive positions at 1 second intervals (28 bytes for the communication overhead, 25 bytes for the fixed length data for the first second, and 9*5=45 bytes for the variable length data of the remaining 9 seconds). If we would send the data as independent packets, the total amount of data transferred would be 10*(28+30) bytes=580 bytes in the previous standard binary data representation (28 bytes for the communication overhead and 30 bytes for each data sent). As it can be seen, an almost *sixfold reduction* of the amount of the data transferred (580bytes/98bytes=5.92) can be

obtained just by using a *more compact data representation*. The total number of bytes required for transmitting N data units using the compact binary data representation is:

$$T_{comp}(N)=D_{comm}+D_{data.fix}+(N-1)*D_{data.var} \quad (2)$$

where D_{comm} is the communication overhead (28 bytes), $D_{data.fix}$ is the fixed data for the first second (25 bytes), $D_{data.var}$ is the variable data for each extra second (5 bytes). The plot in Fig. 7 represents a comparison of the total data amount transferred for each of the two binary representations, i.e. $T_{bin}(N)$ and $T_{comp}(N)$. In the case of standard binary representation, for a 60 second interval, one would need to transfer a total of 3480 bytes (=60 seconds*58 bytes), compared to only 348 bytes (= 28bytes + 25bytes + 59seconds * 5bytes) required for the binary compact representation, therefore ***a reduction of 90% is possible***.

For each time unit (*one second* in our case), the amount of data to be transferred (corresponding to the same information) is significantly reduced in case of the compact representation compared to the standard one as it can be seen below. In Fig. 8, one can see that in case of the standard binary representation, exactly 58 bytes for each data unit are needed, whereas for the compact binary representation, one would need a decreasing number of bytes per unit, ranging from 29 bytes (when sending data for only two time units) to 5.8 bytes (when sending data for a total of 60 seconds). For even larger data units, for example 240 seconds (4 minutes), this will be further reduced to around 5.2 bytes per data unit (the total data sent for 240 time units consist of 1248 bytes). The downside of this approach is that all the data will be available to the server only after the time required to collect and send the whole data, in our cases above of at least 60 seconds or 240 seconds. This means that the data arrives at the server in bulk, at fixed time intervals, according to how much data is sent in one packet. In the second case, the total amount of data transferred in one month would be:

$$\begin{gathered} T_{comp}(1mo) = 30days * 24hrs * (15*4) * 60secs = \\ 30days * 24hrs * 15 * T_{comp}(240s) = 10{,}800 * 1248 \text{ bytes} \approx 13 \text{ Mbytes} \end{gathered} \quad (3)$$

For each of the two representations, the plots in Fig. 8 correspond to the next equations:

$$\begin{gathered} T_{bin}(N)/N = (D_{comm} + D_{data.bin}) \text{ and} \\ T_{comp}(N)/N = (D_{comm}+D_{data.fix}-D_{data.var})/N + D_{data.var} \end{gathered} \quad (4)$$

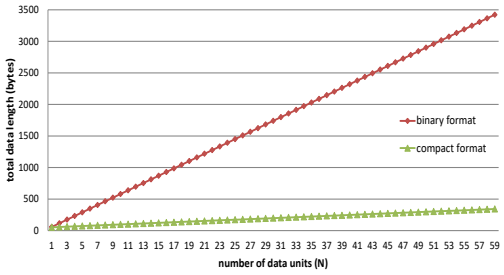


Fig. 7. Total amount of data transferred - standard and compact representation

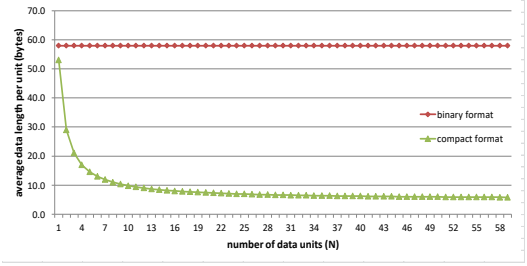


Fig. 8. Average size of data per time unit (second) - standard and compact representation

The compression ratio between the standard and the compact format is shown in Fig. 9. One can see, for example, that for the data required to transfer 10 time units (10 seconds), the compression ratio is 5.92, meaning that we need to transfer almost six time less data in the compact binary form compared to the standard binary format. This ratio is 10 for the data necessary for 60 time units (60 seconds). The compression ratio is:

$$R_{compress}(N)=T_{bin}(N)/T_{comp}(N) =(D_{comm}+D_{data.bin})*N/(D_{comm}+D_{data.fix}-D_{data.var} +N*D_{data.var}) =1/(D_{data.var}/(D_{comm}+D_{data.bin})+1/N*(D_{comm}+D_{data.fix}-D_{data.var})/(D_{comm}+D_{data.bin})) \quad (5)$$

and *for our example*, $R_{compress}(N) = 58/(5+48/N)$, with $R_{compress}(10) = 58/(5+48/10) = 5.92$ *for a 10 sec interval* and $R_{compress}(60) = 58/(5+48/60) = 10.0$ *for a 60 sec interval*. This means that the larger the time interval for which the data is transmitted the better is the compression ratio. However, this cannot be applied ignoring the maximum transmission unit, as it can be seen further here. In computer networking, the Maximum Transmission Unit (MTU) is the

largest size data unit that can be transferred in a single transaction. For larger sizes, the packet needs to be divided into smaller pieces (of at most MTU size) that are sent one after the other, with additional communication overhead. For Ethernet, the MTU size is 1500 bytes, but for mobile networks, including GPRS, it can be smaller, usually 1476 bytes. This means that in order to send, for example, a total of 2000 bytes (28bytes overhead included), one would need to send two data packets, with two overheads, one of 1476 bytes (28bytes overhead + 1448bytes data) and one with the rest of 552 bytes (28bytes overhead + 524bytes data), i.e. each data set larger than the MTU will add more overhead to the transmission. Therefore, in spite of the deduction above showing that the larger the time interval for which the data is transmitted the better is the compression ratio, the time interval needs to be tailored so that the number of packages to be transmitted is kept as low as possible. Furthering this reasoning, in case of different data sets corresponding to different sensors the compact representation has different values for both the fixed part (*fixed value stored* – $F=D_{data.fix}$) and the variations part (*value var stored* – $V=D_{data.var}$) and, therefore, the compression ratio is different. In Fig. 10, the compression ratio for such different data sets is shown (higher values mean better compression). One can see that reducing the variations part for each time unit could result in higher compression rates. This means that in order to improve the communication and reduce the amount of data transferred, one has to minimize the size of the variations part.

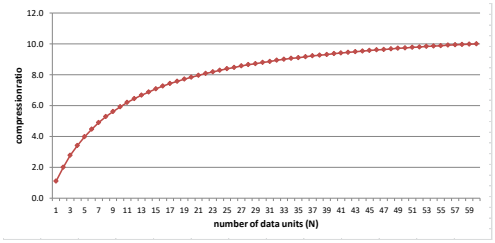


Fig. 9. Compression ratio for data transferred

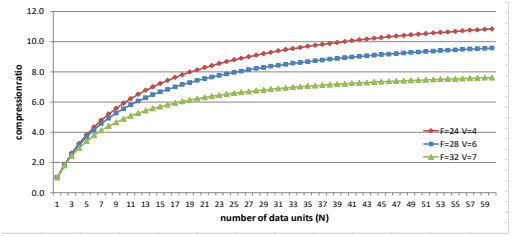


Fig. 10. Compression ratio for fix/variable data

Further on, we present the real traffic data for 3 particular devices (Table 5), which have been collecting data for time intervals between 11 and 13 months, such as the total distance, travel time, number of tracks and total number of packets. A more detailed traffic data including the number of packets of different sizes, average packet size, packets per month, the total data transferred, and monthly data average is presented in Table 6. All the devices are configured to send data during driving at intervals of 20 seconds, so that their near-real-time status is available to the user. In case of vehicles being stopped the devices send periodic status data only once per hour. In Table 7, a comparison between uncompressed and compressed data is shown. The compression ratio is varying between 1:6.0 and 1:6.6.

Table 5. Real traffic data for 3 devices

Device ID	No. months	Distance (km)	Travel time (hours)	No. of tracks	Total pkt. no.
109915	12	2,070	90	375	52,100
660867	11	19,170	134	495	458,700
659549	13	17,136	330	1,400	191,950

Table 6. Detailed data traffic for 3 devices

Device ID	Total no. of packets	No. of packets 208 bytes	No. of packets 80 bytes	No. of packets other size	Avg bytes / packet	Average no. pck. / month	Avg data / month Mbytes	Total compress Mbytes
109915	52,100	25,650	21,336	5,114	144	4,342	0.6	7.5
660867	458,700	205,218	221,677	31,805	133	41,700	5.5	61.0
659459	191,950	108,513	63,337	20,100	151	14,765	2.2	29.0

Table 7. Real data traffic: compression ratio between uncompressed and compressed data

Device ID	Compressed (Mbytes)	Uncompressed (Mbytes)	Compression ratio
109915	7.5	45.5	1:6.1
660867	61.0	367.1	1:6.0
659459	29.0	190.3	1:6.6

As mentioned before, our system is in place for more than 10 years now, monitoring over 500 cars. During this time, a total of about 100 million data packets (about 80 Gigabytes) have been transferred to the data center. The equivalent data in text format would have been around 6 TerraBytes, respectively around 700 Gigabytes in binary uncompressed format. Thus, the system provides significant savings with regard to data transmission's time. This contributes also to important reductions in transmission costs and power consumption.

6. Conclusion and Future Work

The evolution of embedded systems and IoT towards their next-generation depends, to some extent, on what the associated technologies will have to offer. However, the challenge of how to implement applications that perform effectively and efficiently, on limited resources, while fulfilling a rich variety of requirements both functional and non-functional, will remain on, mainly because of their complexity and due to the "delicacy of touch" with which they need to operate within our environment. In such systems, data management is critical from at least two points of view. First, they have to operate with very constrained resources and low power requirements and to make the most of what these resources have to offer, while providing for basic database functionality (storage management, transactions, query processing, or recovery) that is optimized in various directions (such as energy consumption, memory use, etc.). Second, the presence of some sort of data management system is necessary to ensure robustness, flexibility, timeliness, reduced costs, reliability, performance, but also safety of the systems, their users, and the environment they operate within. However, complex compressions algorithms requiring large memory footprint cannot be used due to their limitations, so simpler but still efficient and effective methods must be devised.

In this paper, we introduced a *general method for data representation, storage, and transmission for embedded devices* based on a compact representation scheme and some heuristics. The core idea is to transform the data so that *a minimal number of bits are necessary to encode it without losing any information*. The method can be easily adapted, with the suitable heuristics, for any other type of low cost and resource constrained embedded or IoT device and this is our main future work direction. The same is true for the compact representation scheme, which is well suited for any kind of data that includes time and various sensor readings. Slow variations of data from sensors imply small variations per time unit, which, in turn, allows a more compact data representation by keeping only the variations, which results in higher compression ratio. Such data types are very common to embedded and IoT devices, where time, position, and different sensor values are sent periodically over the Internet for monitoring purposes.

Using a more compact data representation, as the one introduced in this paper, reduces the necessary for memory, processing power, and time required for data transmission, thus reducing effectively the time-to-emit (generally very power-consuming, especially for wireless communication). So, this method contributes to improving the use of both memory and processing capabilities, but also to a significant increase of the battery life time, while decreasing the costs of the mobile data transfers. Future work includes further optimizations of data storing and transferring by using other statistical compression techniques.

References

1. Ashton, K.: That 'Internet of Things' Thing. RFID Journal 22 97-114 (2009)
2. Broy, M: Challenges in automotive software engineering. In: Proceedings of the 28th International Conference on Software Engineering, pp. 33-42. ACM (2006)
3. Buttazzo, G.: Research trends in real-time computing for embedded systems. ACM SIGBED Rev. 3(3), 1-10 (2006)
4. Catthoor, F., Wuytack, S., de Greef, G. E., Banica, F., Nachtergaele, L., Vandecappelle, A.: Custom memory management methodology. Kluwer Academic Publishers Norwell, MA, USA (1998)

5. Chang, L. P., Kuo, T. W.: An Adaptive Striping Architecture for Flash Memory Storage Systems of Embedded Systems. In: Proceedings of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 187-196. IEEE Computer Society (2002)
6. Chui, M., Löffler, M., Roberts, R.: The Internet of Things. Mckinsey Quarterly 2010, http://www.mckinsey.com/insights/high_tech_telecoms_internet/the_internet_ of_things Accessed July, 3, 2017
7. Constantinescu, Z., Vladoiu, M.:Challenges in Safety, Security, and Privacy of Vehicle Tracking Systems. In: Proceedings of Int. Workshop on Systems, Safety and Security for Automotive, Passengers and Good Protection (IWSSS'2013) (2013)
8. Constantinescu, Z., Marinoiu, C., Vladoiu, C: Driving Style Analysis Using Data Mining Techniques. Int. J. of Comp., Comm. Control (IJCCC), 5(5), 654-663 (2010)
9. Culler, D., Estrin, D., Srivastava, M.: Guest Editors' Introduction: Overview of Sensor Networks. Computer, 37(8), 41-49 (2004)
10. Ebling, M.R.: Pervasive Computing and the Internet of Things. IEEE Pervasive Computing. 15(1), 2-4 (2016)
11. Gerla, M., Lee, E-K., Pau, G., Lee, U: Internet of vehicles: from intelligent grid to autonomous cars and vehicular clouds. In: Proceedings of the IEEE World Forum Internet of Things (WF-IoT 2014), pp. 241-246. IEEE (2014)
12. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Comp. Syst., 29(7), 1645-1660 (2013)
13. Hansmann U., Merk L., Nicklous M. S. M., Stober T.: Pervasive Computing Handbook, 2nd edition, Springer Verlag Berlin Heidelberg (2003)
14. Henzinger, T. A., Sifakis, J.: The Discipline of Embedded Systems Design. Computer, 40(10), 32-40 (2007)
15. Kim, G. J., Baek, S. C., Lee, H. S., Lee, H. D., Joe, M. J.: LGeDBMS: a small DBMS for embedded system with flash memory. In: Proceedings of the 32nd Int. Conf. on Very Large Data Bases, pp. 1255-1258. VLDB Endowment (2006)
16. Noergaard, T.: Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers, 2nd edn., Newnes-Elsevier, Waltham, MA (2012)
17. Rosenmüller, M., Siegmund, N., Schirmeier, H., Sincero, J., Apel, S., Leich, T., Spinczyk, O., Saake, G.: FAME-DBMS: tailor-made data management solutions for embedded systems. In: Proceedings of EDBT workshop on Software engineering for tailor-made data management, pp. 1-6 (2008)
18. Saake, G., Rosenmüller, M., Siegmund, N., Kästner, C., Leich, T.: Downsizing Data Management for Embedded Systems. Egyptian Computer Science Journal, 31(1), 1-13 (2009)
19. Saha, D., Mukherjee, A.: Pervasive computing: a paradigm for the 21st century. Computer 36(3) 25-31 (2003)
20. Schulze, S., Pukall, M., Saake, G., Hoppe, T., Dittmann, J.: On the Need of Data Management in Automotive Systems. In: Proceedings of Datenbanksysteme in Business, Technologie und Web, pp. 217-226 (2009)
21. Top 50 Internet of Things Applications, 2014, http://www.libelium.com/top_50_iot _sensor_applications_ranking/ Accessed May, 3, 2018
22. Vasilescu, I., Kotay, K., Rus, D., Dunbabin, M., Corke, P.: Data collection, storage, and retrieval with an underwater sensor network. In: Proceedings of the 3rd ACM Conference on Embedded Networked Sensor Systems, pp. 154-165 (2005)
23. Vladoiu, M., Cassens, J., Constantinescu, Z: FACE – A Knowledge-Intensive Case-Based Architecture for Context-Aware Services. Networked Digital Technologies, Zavoral, F. et al. (eds) CCIS, 88. pp. 533-544, Springer (2010)
24. Vladoiu, M., Constantinescu, Z.: u-Learning within a Context-Aware Multiagent Environment. Int. J. Comp. Networks & Comm. (IJCNC), 3(1), 1-15, January (2011)
25. Weiser, M.: The Computer for the Twenty-First Century. Scientific American. 2, 94-100 (1991)