

Real-Time Data Analyses in Secondary Schools Using a Block-Based Programming Language

This is an author draft.

The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-319-71483-7_17

Andreas Grillenberger and Ralf Romeike

Friedrich-Alexander-Universität Erlangen-Nürnberg
Computing Education Research Group
Martensstraße 3, 91058 Erlangen
{andreas.grillenberger,ralf.romeike}@fau.de

Abstract. Data management is central to many CS innovations: Smart home technologies and the Internet of Things, for example, are based on processing data with high velocity. One of the most interesting topics emphasizing several challenges in this field is real-time data analysis. In secondary CS education, it is only considered marginally. So far, there are no tools suitable for general-purpose real-time data analysis in school. In this paper, we discuss this topic from a secondary CS education perspective. Besides central concepts and differences to traditional data analysis using relational databases, we describe the development of a general-purpose **Snap/** extension that allows accessing and processing data from various sources. Thereby, students are enabled to conduct data analyses using, for example, sensor data or web APIs. With the example of a weather station, we outline how this tool can be used in school for analyzing sensor data generated in the classroom.

Keywords: Real-time data analysis · Data stream systems · Data management · Sensor data · Physical computing · Secondary CS education

1 Introduction

In previous years, several innovative topics of computer science became pervasive in our daily lives. The increasing possibilities when capturing, processing, analyzing and visualizing data are, for example, central to home automation (“smart home”) and when connecting various devices in the Internet of Things. Also, data analyses are often used for decision-finding, even in contexts in which this would hardly be expected in, such as autonomous cars or when addressing voters in election campaigns. The emerging field *data management* comprises several developments originating from the challenge to store and analyze so-called *big data*, i. e. large amounts of data that are generated and analyzed with high velocity and have strongly varying structures [10]. Despite its highly innovative character, various practices and principles of this field seem promising for general educative CS education. For example, partitioning and replication of data are increasingly relevant, not only when developing data management systems, but also when synchronizing data between different devices and services:

Understanding why data are sometimes duplicated or lost and how to prevent this is hardly possible without getting to know the underlying concepts, such as redundancy. One exemplary topic emphasizing several principles and practices of data management, is real-time data analysis. In practice, such analyses are often conducted using data stream systems. With these, immediate analysis of large amounts of data are possible. As today we are often confronted with the results and implications of real-time data analyses, getting to know their basic principles becomes increasingly relevant for understanding such results, for evaluating their quality and relevance, but also for making decisions based on such data [7]. For developing and fostering competencies related to capturing, storing and processing data and for estimating and evaluating the consequences and implications of these possibilities, real-time analyses are an ideal starting point. Hence, in this paper we will outline the functionality and central principles of real-time data analysis in general and of data stream systems in particular. Based on this, we will describe how central ideas of real-time data analysis can be included in secondary education using a general-purpose data stream system extension for the block-based programming language `Snap!` [8].

2 Related Work

Despite their high relevance, not only in computer science, neither the topic data management in general, nor real-time data analyses in particular, have been examined in detail as topics for secondary CS education yet. Although related contexts, such as the Internet of Things, have already been discussed as topic for CS education (cf. e. g. [11]), courses and projects presented are typically designed for higher education. As they pursue different goals, only few aspects can be transferred to our work, such as the practical orientation and the hands-on approach. Also in robotics, processing streams of sensor data and events is central from a technical perspective, but has not yet been discussed as a topic for CS education. As we have shown in a qualitative analysis of various curricula and educational standards, today most data management topics, including real-time data analysis, are typically not included in secondary CS teaching [5]. In consequence, as of today, there are no suitable tools for general-purpose real-time data analysis in school. To bring aspects of modern data analyses to school, we already developed and described a tool for analyzing the Twitter data stream [6], which has been presented to teachers and discussed with them at various opportunities. The advantages in comparison to data analyses using databases were convincing for most teachers. Yet, there was always one concern: To work with the Twitter API, students need to have an account on this platform. As a workaround, we offered the possibility to work offline using a cached data set, but this was obviously less motivating for the students than working with live data. Also, teachers were often concerned that privacy issues might arise and noted the restricted flexibility of the tool, as it only supported the Twitter data stream as a data source. Hence, for tackling these issues, in this paper we describe a new general-purpose approach which allows using various data sources and nearly all

possibilities of the continuous query language¹.

3 Real-Time Data Analyses

Today, real-time analyses are being used for several purposes: analyzing credit card transactions in order to prevent fraud (cf. [12]), reacting to temperature changes in smart home environments, or monitoring the environment of smart cars. Although in many use cases, providing results of data analysis immediately is not essential, in most modern use cases there are at least weak restrictions on the reaction times of a system: Typically, users want data to be available as soon as possible, hence “soft real-time”² becomes increasingly important. In other use cases, such as industrial robots, the restrictions on data analysis are even harder: they need to fulfill firm or hard real-time requirements³.

In order to meet a deadline even when analyzing large amounts of data, traditional data analyses are not sufficient. Instead, typical real-time data analyses are in several parts completely different. While traditional data sources are rather finite and discrete, for example when sensor data streams are analyzed, the analysis system needs to handle infinite and continuous data sets (cf. [13]). Also, the amount of data sources (in particular sensors) is growing, with wireless communication the data are available faster than years ago, and the data rate is increasing drastically [13]. Yet, when regarding current CS teaching, analyzing data in schools, if at all, typically takes place using databases: For example, in popular weather station projects, the sensors data are often stored in relational databases and queried using the query language SQL. While this approach is suitable for traditional data analyses, several challenges occur when trying to conduct real-time analysis that way: In particular when data are generated continuously, analysis jobs would need to be started frequently in order to ensure up-to-date results, as determining distinct points in time for starting the analysis is not possible when analyzing data streams⁴. In consequence, the database (and the analysis) gets overloaded or stuck because of the continuous and parallel read and write operations. Additionally, using databases for this purpose also results in enormous amounts of data being stored, because of the high rate of data generation and as it is not possible to delete outdated values, as information

¹ The “continuous query language” (CQL) is similar in syntax to SQL, but in particular allows using “sliding windows”, which makes it suitable for data stream analysis (cf. [1]).

² *Soft real-time* allow even frequent misses of the deadline, as only service quality is being influenced (cf. e.g. [3]).

³ *Hard real-time* strongly requires adherence to a deadline, as exceeding it results in a system failure. *Firm real-time* tolerates missing the given deadline infrequently, but the analysis results become irrelevant after the deadline and the quality of service is degraded.

⁴ “A data stream is a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items. It is impossible to control the order in which items arrive, nor is it feasible to locally store a stream in its entirety.” [4]

would get lost. Hence, although databases are a suitable tool for storing large amounts of rather static data for a long time and analyzing such data, they can hardly be used for real-time data analysis. Today, this problem can be solved by using data stream systems, which are designed as general-purpose systems for fast analysis of continuous data streams.

In several characteristics, these systems are the opposite of databases: In particular, data stream systems do not store data permanently, but instead process them on-the-fly by applying queries that have been previously defined [9]. Thus, data stream systems are appropriate when data are to be analyzed only once, immediately after being generated. This is the case in particular for sensor data: As every value is only up-to-date until the next value is generated, immediately analyzing them is important. In addition, in such use cases it is typically reasonable to drop values if they cannot be analyzed immediately and to continue with the next value which is then the most relevant one, as significant changes typically do not influence single values only. Data stream systems can, in particular, show their potential when “sliding windows” are used, i. e. when a defined number of recent values or all values from a defined time span are analyzed (cf. [2]). With databases, during every execution of a query, all data stored in the database would be re-evaluated (whether or not they were changed before), while using “sliding windows” allows to cache and evaluate only the relevant data, ensuring that they are deleted from the cache when out-of-date.

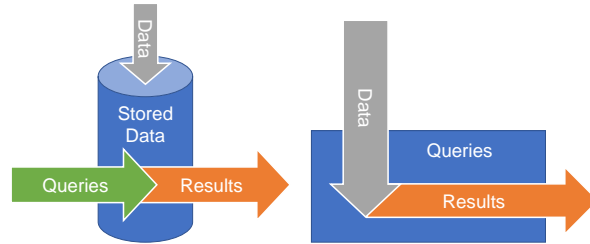


Fig. 1. Functional principles of databases and data stream systems.

The different functional principles of databases and data stream systems are visualized in Fig. 1: When using databases, storing and analyzing data are independent processes. In contrast, in data stream systems, data is directly analyzed after they were received by the system. Hence, many modern systems processing continuous data streams are based on characteristics of data stream systems rather than of databases: For example, when measuring the wind velocity in a smart home project for triggering the closure of windows, it is neither important to process every single value nor to store these data permanently. Hence, data stream systems are ideal for this purpose. To summarize, Table 1 shows a comparison of central characteristics of both, databases and data stream systems.

Table 1. Comparison of the main characteristics of databases and data stream systems.

characteristics	database system	data stream system
data storage	permanent storage, often relational but also other paradigms	no storage except caching of values needed for further processing
data processing	when executing a query	immediately after new data were received
typical data	data that should be stored permanently and that are relevant for long term; often used for multiple queries/purposes; often rather static data	continuous data streams consisting of data which quickly become outdated; typically only processed for one purpose
query language	typically descriptive, often SQL	typically descriptive, often CQL
popular implementations	Professional implementations: MySQL, PostgreSQL, MongoDB ...	Research implementations: STREAM, AURORA, TelegraphCQ, Odysseus ...
central concepts	integrity, consistency, redundancy	availability, concurrency

4 Real-Time Data Analyses in Current Secondary CS Teaching

When the topic “data” is addressed in secondary CS teaching, the focus is typically on (relational) databases [5]. This has not been changed since the early 1990s. Thus, because of the tremendous developments in recent years, several topics that are strongly related to the students’ daily life today are only considered marginally, such as data security, privacy, encryption or meta-data. Also, “real-time data analysis” is typically not considered in secondary CS teaching at all: Although such analyses are pervasive in our daily life, understanding their underlying principles and functioning is not a goal of typical CS education. Yet, to understand their relevance, opportunities and threats, getting to know basic aspects of real-time data analysis is essential. For example, it is hard to imagine how large amounts of data captured by CCTV may be analyzed in real-time without knowing the underlying analysis methods. But when they know about the restrictions and differences of real-time analysis in comparison to data analysis in general, students also become familiar with the involved opportunities and threats.

Recently, aspects of data management and real-time data analysis can be found in school teaching when data is acquired and evaluated in physical computing projects. For example, sensors are used for measuring and evaluating environmental influences. In simple projects, data is typically processed directly on the microcontroller, e. g. when just reacting on concrete values or measuring and displaying values such as the current temperature. But in more complex

projects, for example calculating average temperatures for defined time frames, analyzing the data directly on the microcontroller is often not possible due to memory and performance restrictions. Hence, data is often sent to a computer for further storage and analysis. This typically involves multiple tools, such as the programming environment for the microcontroller, one for controlling data storage and analysis, and a database such as SQLite for data storage.

Currently, a popular project in secondary schools is building weather stations. With several sensors, data is gathered continuously. Often, the goal is more complex than just showing the current values, as for example, average or extreme values are more interesting. When calculating such values for defined time frames (i. e. for the last 24 hours), the analysis becomes too complex for typical microcontrollers used in school. When trying to use traditional approaches for storing and analyzing the sensor data, all values in this time frame need to be stored in a suitable data storage, deleted from it as soon as they are outdated and replaced with other values. This would lead to numerous read/write operations. When using databases for this purpose, as it is common in such projects, the problems described before arise. Yet, professional data stream systems are hardly suitable for use within CS education because of their high complexity.

5 Challenges for Teaching Real-Time Data Analysis

5.1 Data Sources

As analyzing data in real-time is an especially suitable for dynamically changing data, data sources that are traditionally used in CS education are not suitable for this purpose. Instead, there are in particular two approaches for accessing “live data”:

Web APIs With several web applications providing free access to their application programming interfaces (APIs), lots of data can be acquired. For example, social media platforms such as Twitter and Facebook provide access to large parts of their data. There are two approaches for accessing the data: While Twitter uses a “push” method, i. e. the client subscribes to the stream and is notified by the server about new tweets (comparable to the observer pattern), most other APIs use a “pull” approach, i. e. the client frequently requests information. Although pulling produces more unnecessary communication and delays, it has an advantage for teaching: As the connections are not kept open all the time, typically multiple users can use the same credentials. Also, as most APIs are based on the REST principle and hence are accessed via HTTP calls, using them is relatively easy, particularly if they are based on a “pull” approach. Hence, these interfaces are ideal data sources for real-time data analysis, as the data are (depending on the application) highly dynamic and as access is typically only limited by rate limits of the APIs.

Sensor Data Another data source is sensor data. With physical computing projects gaining popularity in CS education, using sensors for measuring data from the environment of a system has already become common in CS teaching. This is also a suitable approach for gathering data for data management lessons: The main advantage of using sensor data as a basis for data analysis is that it is generated in real-time in the classroom environment, without hurdles of using web APIs, such as mandatory user accounts or privacy issues. Also, as nearly all common programming languages allow communication with microcontrollers, from traditional object-oriented languages such as Java right up to block-based programming environments designed for educational use such as Scratch or **Snap!**, finding a suitable tool for obtaining data is not a problem. Another advantage of using sensor data in data management education is the shift of focus from just processing and analyzing data, to the whole data life cycle from their acquisition and modeling, through processing and analysis, to visualization and (perhaps) deletion of data. This depicts the real usage of data and shows how different CS fields come together in innovative topics like the “Internet of Things”, which is based particularly on data management technologies and (interconnected) microcontrollers. As capturing sensor data can be done in various ways, there are also very different types of sensors that might be used, and hence also different ideas for data processing and analysis.

5.2 Development of a Real-Time Data Analysis Extension for **Snap!**











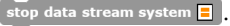
Design Decisions For accessing both, sensor data and REST APIs, several tools suitable for CS education already exist. In particular, the block-based programming language **Snap!** [8] has high potential for data management teaching: Not only does it provide blocks for sending HTTP requests that are suitable for accessing REST APIs, it can also communicate with Arduino (and compatible) microcontroller boards in the fork **Snap4Arduino**⁵. Hence, it is possible to access sensor data captured with such boards. It is also highly expandable, so that processing data is possible using the same tool with which they are captured, and it is easy-to-use for students, even without previous experiences using it. Thus, for conducting real-time data analysis in CS teaching, we implemented the central features of data stream systems and CQL [1] in the **Snap!** programming environment. Using **Snap!** as a basis clearly contributed to our main goals: The tool should be flexible enough to be used with various data sources, it should be easy-to-use in a school context and it should be easy-to-extend by teachers and (ideally) also by the students themselves. To allow the data stream system extension to be used in **Snap!** and all its forks, we only used functionalities that **Snap!** provides in its end-user interface and avoided to directly modify its source code. Hence, the extensions is completely based on primitive **Snap!** blocks and JavaScript functions (which can also be called in **Snap!** out-of-the-box by mapping them into blocks). In addition to extending **Snap!** for allowing data stream analyses, we also extended it for data visualization purposes in the same way.

⁵ <http://snap4arduino.org>

Implementation of the Data Stream Analysis Extension In contrast to our tool for analyzing the Twitter data, which was based on imperative queries, for the **Snap!** data stream system extension (“**Snap/DSS**”) we use declarative queries. This approach facilitates handling the queries as one unit in the programming environment and allows nesting queries like is possible, for example, in typical query languages such as SQL. Also, as professional data stream systems typically use declarative languages such as CQL, **Snap/DSS** conveys the function of professional data stream analysis tools better than SnapTwitter. For making the extension as flexible as possible, **Snap/DSS** offers the following functionalities:

- Creating a new data stream from any data source (“reporter block” in **Snap!**).
- Combining data from several sources in one stream.
- Running queries on data streams, using aggregate functions, projections, selections and sliding.
- Using analysis results as a data source for a new data stream (nesting).
- Continuous evaluation of queries in the background.

To implement these functions, we created a data structure which is represented in **Snap!** as an unevaluated block. This internal data structure is rather complex: The data stream system, data streams and queries are represented as lists with various fields (cf. Fig. 2). In particular, a data stream stores a list of its queries and information on its data source; each query stores its parameters as well as a cache of current values. When a data stream system is started, each query of all of its streams is executed in the background about once per second⁶. Hence, the values stored in the data stream system are updated permanently, as it is common in data stream systems. Despite its complex structure in the background, **Snap/DSS** can be easily used with the following blocks:

- **create data stream system** : Creates a new data stream system, to which streams can be added for continuous processing.
- **new data stream from**  : Creates a new data stream, which is based on the data given in an input field.
- **add stream**  **to data stream system** : Adds a data stream to a data stream system.
- **select**  **from stream**  **last**  **values** : Executes queries on the data streams. After the first use of a query, the system continuously captures the relevant data for the query while running. In a query, aggregate functions (average, minimum, maximum, sum) and windows may be used.
- **start data stream system** : Starts continuous data processing in the background.
- **stop data stream system** : Stops background data processing.

Besides allowing high flexibility and all relevant functionalities, our approach also gives the teacher the opportunity to simply adapt the tool for concrete lessons and thus to further reduce complexity (e.g. by hiding blocks or by creating new ones), without losing the flexibility for complex analyses.

⁶ We limited processing of queries to one time per second to prevent performance issues. Yet, this limit can easily be changed by modifying the block in **Snap!**.

6 Example Project: Weather Station

As mentioned before, a common example when capturing and processing sensor data in school, is building a weather station. This project allows students to capture lots of data with sensors using almost any typical physical computing platform⁸. In order to avoid the problems of using the typical database-based approach, in the following we will describe, how a weather station can be implemented using our **Snap/DSS** extension.

In this project, students can recognize the main functional principles and concepts of real-time data analysis. Hence, they can carry out their own data analysis and understand how easily several data sources can be combined for achieving better analysis results, e. g. by making recorded values more accessible through aggregation and by creating simple forecasts (e. g. using pressure differences as an indicator for weather changes). Also, by including API data, the results could be compared to professional analysis for evaluating them. For capturing and analyzing the data, **Snap4Arduino** on the software side (with **Snap/DSS** loaded) and an Arduino Uno microcontroller board on the hardware side can be used. We use a combined sensor for temperature, pressure and humidity, which we connected using the “grove shield”⁹. Yet, our extension does not restrict using other sensors, as long as their values can be read using reporter blocks in **Snap!**, as those are needed for using the data stream extension. Using our extension, the values read by the reporter blocks are interpreted as a data stream. After adding this stream to the data stream system and starting it, the sensor values are evaluated according to the queries defined. As no queries have been defined yet, nothing would happen when running the program; first, at least one query has to be defined (which can even happen on-the-fly while the system is running). As we are interested in the minimum, average and maximum values of both, pressure and temperature, we can define three queries, as it is depicted in Fig. 4 and Fig. 5 for calculating the average value of a light sensor.

The results of the queries can directly be shown, e. g. by clicking on a query block, but they can also be used as input for other blocks in **Snap!**. Hence, we are, for example, able to show them on a LCD display or to visualize them using the data visualization extension, as it was done in the example shown in Fig. 5. For displaying the data of one sensor and the corresponding average value, the code and the result is shown in Fig. 6. The visualizations may be changed without losing information, because the data from all sensors are analyzed in the background as defined in the queries after starting the system.

This example project is easier to understand and implement than the typical implementation in schools using databases, in particular as fewer different systems are involved. Due to the design of the tool, it is clearer and allows to be extended flexibly, and it can also be used for more complex analyses. The project can be realized with students with or without prior knowledge on data

⁸ An exemplary project was described by the Raspberry Pi Foundation: <https://www.raspberrypi.org/blog/school-weather-station-project/>

⁹ http://wiki.seeed.cc/Grove_Starter_Kit_v3/



Fig. 4. Analyzing a sensor data stream.

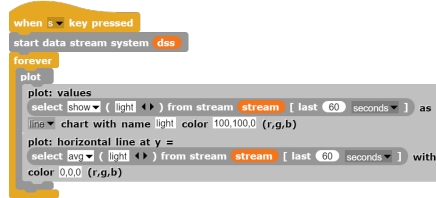


Fig. 5. Visualizing sensor data.

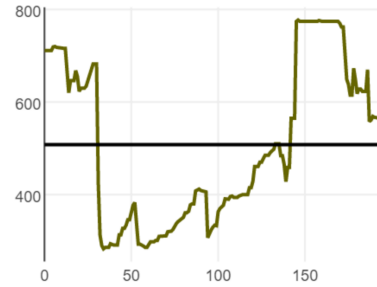


Fig. 6. The line chart represents the actual values, the horizontal line shows the average value.

management or physical computing, and with no additional material or systems in comparison to typical physical computing projects. The concept can be transferred to other projects, such as building a smoke detector (which could also take into account temperature increases, in order to prevent false alerts), realizing smart home projects (e.g. switching on the light when the environmental brightness becomes lower than the average in the last five minutes, ensuring that temporary peaks are neglected), or even for quantified self projects (such as realizing a sleep quality measurement device or for measuring heart rates).

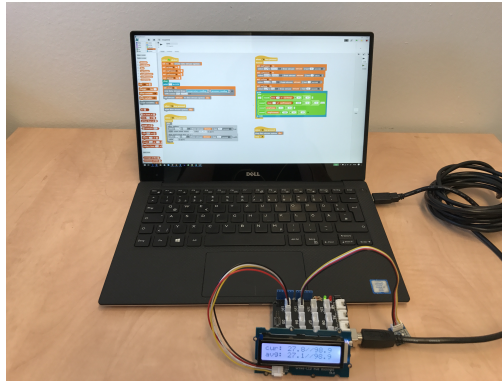


Fig. 7. Exemplary weather station project using a combined temperature and pressure sensor as well as a LCD attached to a Grove shield.

7 Summary

In this paper, we discussed central characteristics of real-time data analyses. Based on these, we proposed a tool, which is suitable for conducting general-purpose data stream analysis in secondary CS education. In an example, we have shown how this tool can be used in school in combination with physical

computing. Although the **Snap!**DSS extension reduces complexity, it preserves characteristics and functionalities of real-time data stream analyses. It is oriented at the syntax and semantics of CQL, allows its typical operations and implements central aspects and functionalities of general-purpose data stream systems. In combination with accessible hardware, students are enabled to pursue their own ideas and to conduct analyses on their own, which are otherwise only possible for professionals in the field. Thus, this approach opens up new opportunities for the students to benefit from the innovations in data management. Yet, by preserving the typical character of data stream systems and real-time data analyses, it also fosters a better understanding of several central concepts of data management.

References

1. Arasu, A., Babu, S., Widom, J.: The CQL Continuous Query Language: Semantic Foundations and Query Execution. *The VLDB Journal* 15(2), 121–142 (Jun 2006)
2. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: *Proceedings of the 21th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. pp. 1–16. ACM, New York, NY, USA (2002)
3. Edward A. Lee, S.A.S.: *Introduction to Embedded Systems*. MIT Press Ltd (2017)
4. Golab, L., Özsu, M.T.: Processing sliding window multi-joins in continuous queries over data streams. In: *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*. pp. 500–511. VLDB '03, VLDB Endowment (2003)
5. Grillenberger, A., Romeike, R.: A Comparison of the Field Data Management and its Representation in Secondary CS Curricula. In: *Proceedings of WiPSCE 2014*. ACM, Berlin (2014)
6. Grillenberger, A., Romeike, R.: Analyzing the twitter data stream using the snap! learning environment. In: *Proceedings of ISSEP 2015. Lecture Notes in Computer Science*, Springer International Publishing (2015)
7. Grillenberger, A., Romeike, R.: Teaching data management: Key competencies and opportunities. In: Brinda, T., Reynolds, N., Romeike, R. (eds.) *Proceedings of KEYCIT 2014*. Universitätsverlag Potsdam (2014)
8. Harvey, B., Mönig, J.: Snap! reference manual (2014), <http://snap.berkeley.edu/SnapManual.pdf>, last accessed 09/07/2017
9. Krämer, J.: *Continuous Queries over Data Streams - Semantics and Implementation*. Philipps-Universität Marburg (2007)
10. Laney, D.: 3D data management: Controlling data volume, velocity, and variety. Tech. rep., META Group (February 2001)
11. Mäenpää, H., Varjonen, S., Hellas, A., Tarkoma, S., Männistö, T.: Assessing iot projects in university education: A framework for problem-based learning. In: *Proceedings of the 39th International Conference on Software Engineering*. pp. 37–46. IEEE Press, Piscataway, NJ, USA (2017)
12. Mayer-Schönberger, V., Cukier, K.: *Big Data: A Revolution That Will Transform How We Live, Work, and Think*. Houghton Mifflin Harcourt (2013)
13. Nittel, S.: Real-time sensor data streams. *SIGSPATIAL Special* 7(2), 22–28 (2015)