End-to-End Data Architecture Considerations for IoT

Alokika Dash

1 Introduction

Advances in shrinking form factor of embedded systems, ubiquity of networking, and low cost of sensors and processors has led to the proliferation of IoT (Internet Of Things) devices. An IoT system is often deployed to gather data and make decision about a system—at a fine granularity that is was not possible earlier. There has been significant effort towards the challenges of building and deploying IoT devices. Specifically, these challenges arise due to resource constraints that an environment may pose. For example, power consumption, form factor, cost per device, security, systems software (OS), and application software challenges need to be addressed before deploying an IoT. While the aforementioned are important, and there has been significant research work [1–5], an end-to-end view of the entire system in terms of data flow needs to be addressed as a first class architecture consideration *before* deployment. For example, in certain class of IoT devices, reacting in real-time to an anomaly in sensor value could be of utmost importance. In this case, *compute* and *latency* are the primary metrics of data i.e., the ability to compute and detect an anomaly and subsequently react. Further, this has to be done close to the IoT device (edge/fog) as opposed to the cloud or the datacenter due to strict latency requirements. In another class of IoT devices, analyzing months of data to learn about patterns could be of utmost importance. In this case, the *data storage format* is of utmost importance. In a third scenario, there could be a pipeline of systems that act on data where the first system in the pipeline acts on streams of data (i.e., near real-time) and the subsequent pipeline needs to run throughput oriented (batch, map-reduce) jobs. In all of the above cases, it is important to understand and architect the flow of data such that the SLA (service level agreements)—be it latency, throughput, and/or query processing rate, can be met. Therefore, an end-to-end data

A. Dash (✉)
Irvine, CA 92618, USA
e-mail: adash@alumni.uci.edu

S. Kumar Pani and M. Pandey (eds.), *Internet of Things: Enabling Technologies, Security and Social Implications*, Services and Business Process Reengineering,
https://doi.org/10.1007/978-981-15-8621-7_2

oriented design consideration can lead to requirements that can be placed upon the architecture i.e., network considerations, processing power, RAM on each node etc. In the rest of the paper, we present the challenges and techniques to approach towards and "end-to-end" design of the system.

The rest of the paper is divided into two sections. In Sect. 2, a model for the end-to-end IoT system is defined and data considerations at each stage of this model is quantified. Such a model can help define the system specs at each stage. In Sect. 3, once the system is defined, design considerations for data are outlined that one must be aware of. Designing an IoT end-to-end system with such an approach can result in a predictable system and avoid problems due to either under or over engineering.

2 Modeling

Figure 1 shows a typical IoT architecture. The following are the various components of this architecture:

1. *IoT Device* This is the singleton unit of the system—a single senor/device that is capable of operating independently with the ability to detect (sense) and respond (transmit) to stimuli.
2. *IoT Deployment* A single IoT device in itself is not of much use. Typically, an entire group of IoT devices are deployed to sense and send their data to the next hop in the system.
3. *IoT Gateway* This is the first hop in the system that can gather data, perform limited computation on the gathered data, and the has the ability to interact with external servers. Examples of these systems could be anywhere from a local (on-prem) computer that gathers data to a cell phone tower servers. These systems are capable of limited compute on data such as finding aggregates, running inference models for AI, etc.
4. *Cloud/Datacenter* This is the last hop in an IoT architecture. The cloud and/or datacenter has the ability to provide virtually unlimited compute and data. The tradeoff being latency in response. In this paper, this is also referred to as "Backend system".

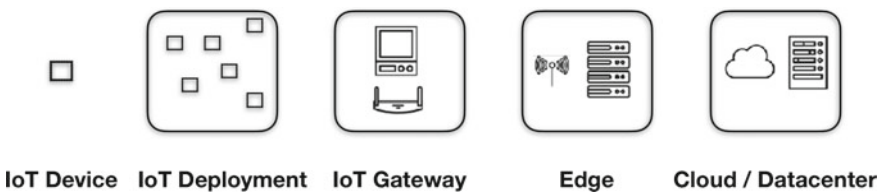

Fig. 1 IoT architecture overview

3 Data Formats

In this section, the data format at each step of IoT architecture shown in Fig. 1 is discussed. Specifically, a quantitative model is presented. While an exact quantitative model is very specific to an implementation, the goals of the data model presented in this section are to provide a first order approximation and bounds (upper limit) on data rate and format so that it can help guide the design of an actual (final) system.

3.1 IoT Device

The data format for IoT devices is typically in raw (binary) format. Given the power, cpu, and memory constraints, it is imperative that the raw sensor data is transmitted in the most efficient format. Therefore, transmitting in a format such as delta encoding [6] is an attractive solution. The reason for a delta encoded format are follows:

1. Most of the time the change in sensor value will be limited (i.e., within the low standard deviation). Given this assumption, the difference in subsequent sensor values can be encoded in few bits/bytes compared to the actual sensor value.
2. Delta encoding involves simple addition/subtraction (low cpu overhead) and maintaining minimal state in RAM (low memory overhead).
3. Delta encoding implies that the network overhead is minimal (low upload bandwidth) which is one of the main contributors to power.

If there are a total of N samples and the delta encoding is such that there are M intervals, and in each interval, there are m_i samples that sum up to N i.e., $\sum_{i=1}^{M} m_i = N$. The gain of using delta encoding is:

$$g = \frac{b \times N}{\left(b + \sum_{i=1}^{M-1} d_i \times m_i\right)} \tag{1}$$

where, b is the bit width for the sensor value. For example, $b = 8$ for 8-bit values, or $b = 64$ for unsigned long values. d is the number of bits used to encode the difference in subsequent values. For example, $d = 3$ if we know that the difference in sensor values is within 8 i.e., 2^3. Note that the delta encoding is efficient when $d \ll b$, and $N \gg M$ i.e., the changes in the sensor values are relatively small. These assumptions are not far from reality due to the fact that most of the times a cyber physical or IoT deployment works in a "normal" state where there are no anomalies that can cause a large delta between subsequent values.

3.2 IoT Deployment

An IoT deployment is a group of IoT devices that report and get response from a fixed set or a single IoT gateway. Since the IoT deployment is a group of IoT devices, the aggregate data that is generated due to the individual IoT device is of importance (i.e., the bandwidth or rate of data generation). If each IoT device, i, generates data every r_i^{th} second, and there are N IoT devices, then the edge gateway receives every j seconds, where: j is the GCD of the N IoT devices:

$$j = gcd(r_0, gcd(r_1, gcd(r_2, \ldots, gcd(r_{N-2}, r_{N-1})))) \tag{2}$$

Therefore, for an IoT gateway, the bandwidth the data processing requirements are as follows:

1. The IoT gateway must be able to receive data at a rate of every j seconds.

2. The amount of data received every j seconds is equal to:

$$\sum_{i=1}^{N} \left(D += \begin{cases} d_i, & \text{if } r_i \bmod = 0 \\ 0, & \text{otherwise} \end{cases} \right) \tag{3}$$

where, D is the amount of data (in bytes) received every jth second. Therefore, the data considerations will be to make sure that the network bandwidth on the edge has enough bandwidth to receive this data and the storage and buffering (RAM) has enough space to accommodate the accumulated data.

In summary, the data considerations put the following minimum requirements on the IoT deployment:

1. The ability to receive data at the peak rate without loss i.e., enough network bandwidth.
2. Enough buffering (RAM) to accommodate this data while it is being computed upon.
3. Enough storage space to persist this data if it is so desired.
4. Based on the data transformation requirements, enough network transmit bandwidth to push this data to the next level which is an IoT edge.

3.3 IoT Edge

The edge is the first hop in the flow of data which has dedicated compute resources for processing and making decision on data. However, the resources on an edge are still limited compared to a data-center or a cloud where there are virtually infinite compute and storage resources available. The flow of data between a gateway and the IoT edge can be modeled using queue theory.

One of the main reasons for having an IoT edge between a gateway and a server is that of reduced latency. The IoT edge can provide response to data in the order of high microseconds or few milliseconds due to the network proximity to the IoT gateway. Therefore, any algorithm or data-based decision made at the IoT edge will have to process streaming data in near real time. A few examples of such data-based decision algorithms are:

1. *Aggregates*: Statistical operations such as average, moving average, min-max etc. The streaming (incoming) data will have to be de-serialized to a standard format.
2. *Inference*: The de-serialized data can be passed into an neural-network inference model in order to classify and/or predict.

In both of the above cases, the data considerations are:

1. De-serialize data to a time-series data format.
2. Run stream-processing. For example using Apache flink, Spark, Kafka to name a few.
3. Optionally use other specialized processing such as Tensorflow if an inference is required.
4. The ability to—(i) optionally store this data on the server, and (ii) push the data to the backend server, and (iii) discard data based on a time-window if it is already stored.

In terms of data architecture considerations, the IoT edge plays the role of both a "backend" for IoT gateway and "frontend" for the backend-server (i.e., Cloud/Datacenter). The flow of data in this stage can be modeled using queueing theory, where the arrival rate is dictated by the IoT gateway and the departure rate is modeled based on the backend server. Figure 2 shows a queueing model that can be used to determine the system characteristics of an IoT edge.

Specifically, the following explain how a queueing model can be used to come up with a system specification for an IoT edge node.

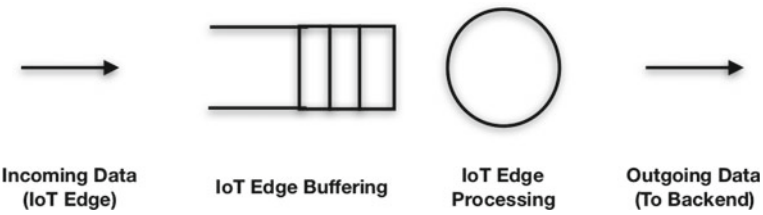

Fig. 2 IoT edge data queueing model

3.4 Incoming Data

This is the rate at which data arrives from one or more IoT gateways. This is driven by Eq. (3). The implications here are that the IoT edge must have sufficient network bandwidth to receive data at this rate. For example, if the IoT edge has a full duplex 1 GbE ethernet connection, then the receive rate for the IoT edge is capped at around 100 MB/s theoretically and practically around 80 MB/s given the IP layer overhead.

3.5 IoT Edge Buffering

Using Little's law $L = \lambda \times W$, one can determine the amount of RAM and processing required for the IoT edge node. In the IoT edge case, λ is the arrival rate of the incoming data. W is the rate at which data is processed which correlates to the computation (algorithm) required to process the data. Note that this computation can be in many different forms such as—(i) time required to run inference model on the data, (ii) time required to compute average and standard deviation, (iii) time required to transform the incoming data and write it to local storage. (Note that this involves matching CPU and IO bandwidth, which itself can be another queueing model). (iv) time required to transform data and push it to the backend server. (Note that this involves matching CPU and network upload bandwidth). Finally, L is the buffering required which in the IoT edge case is the amount of RAM (memory) available for application buffering. Note that this is just the memory required for application buffering and does not include memory requirements of the system itself such as operating system and other services (logging, daemons, ...) running on the IoT edge.

Since λ is fixed and known, one can perform design space exploration based on different values of RAM (i.e, L) and CPU (i.e., W). These can be based on power, cost, etc. Also, if the data needs to persist, this can further drive the requirements of the storage system i.e., choice of flash memory (SSD) and/or HDD. If the data storage requirements are known, Little's law can be applied further in order to calculate W i..e, $W = W_{cpu} + W_{storage}$ i.e., W is the sum of rates at which data is transformed by the (computation) and the rate at which data can be persisted.

3.6 Backend Servers

This is the last hop of the data flow. When designing an IoT system, one can consider the backend servers to be an on-prem data center or the cloud. The data storage here can be transformed from one format into another depending on the analytic needs. For example, the incoming data can be stored as a time series data base. This data for analytical purposes might need to be converted into a format that applications such as

Apache Spark, Cassandra, etc. can use to run analytical, map-reduce jobs. Therefore, the data may need transformation from time series data base into a NoSQL format. While not a full fledged ETL, the choice of data format is important.

4 Software Considerations for Data

Having established model for an end-to-end IoT system in the previous section, the table below summarizes the flow of data across each stage of an IoT system and describes the data considerations at each stage (Table 1).

Specifically, the considerations are as follows:

1. *Data Type* This is the layout of data. As we go from the IoT sensor towards the cloud, the nature of data turns from being bytes (raw/binary) to having a structure (Eg: SQL schema). As we move from left to right in Fig. 1, the data gathers meaning and form.
2. *Data Persistence* It is important to have a checklist at each stage on data persistence. The persistence further can be divided into short term and long term. Short term persistence implies less storage but needs guarantee that the short term storage has been processed before it can be overwritten by the next batch of data. For long term persistence, storage space is important and therefore, considerations such as compression becomes important which may not be necessary for short term persistence because the data is eventually ephemeral.
3. *Data Size* The data size along with persistence drives the combination of system RAM and storage and also helps in answering questions such as does one need in-memory database ? The data size in the cloud must consider questions as to how important the data is—does it need to be replicated, backed up, should one use RAID or not.
4. *Data Operations* The choice here is important because it drives what kind of software frameworks will run on the edge and on the cloud. Note that at each step, there can be multiple data processing and software frameworks and therefore it is best to prototype this ahead of time. For example, there are multiple software

Table 1 Data characteristics

Stage	Data type	Data persistence	Data size	Data operations
IoT device	Raw (binary)	Volatile	Bytes	Encoding
IoT deployment	Encoded	Volatile	KiBs to MiBs	Encoded, serialized
Edge	Streaming	Streaming and persistent	KiBs to MiBs	Aggregates, inference
Cloud (on-prem/public)	Schema and/or NoSQL	Non volatile	GiBs to TiBs	Map-reduce, query, AI

frameworks available for streaming—Spark streaming [7], Flink [8], Kafka [9], etc. Similarly, there are multiple framework available for machine learning—Tensorflow [10], Caffe [11], Pyspark [12] etc.

4.1 Data Interoperability

As data flows in an IoT system, the question of data interoperability i.e., as the output data from one software framework is passed to another software framework, does the data format needs a change ? If so, what is the cost of transforming data from one format to another. Specifically, what is the compute, memory, and storage cost of transforming data. This is especially an important issue given the wide matrix of software and open source frameworks available at each stage of data processing. One needs to answer the following questions:

(a) *Interoperability* How efficient it is to build a pipeline using two different frameworks. Questions such as—do they work together, is it required to write a lot of code for interoperability, etc.
(b) *Data Standard* Does the data need to be transformed between these software stacks or do they operate on a single standard data format.

5 Conclusion

In conclusion, when designing an IoT system, it is important to take a step back and enumerate the flow of data from sensor all the way to data at rest. Further, while enumerating, quantifying the rate and size of data flow can help answer design considerations such as minimum resource (CPU, memory, storage, and networking) required at every stage. Further, it is important to consider—(i) the data formats, and (ii) the data interoperability when making a choice for software framework used on data. Such a data and quantitative based approach to designing IoT can help guide the "end-to-end" design of an IoT system and alleviate hardware resource limitation issues that typically arise post deployment and are therefore, much harder to resolve.

References

1. S. Chen, H. Xu, D. Liu, B. Hu, H. Wang, A vision of IoT: applications, challenges, and opportunities with China perspective. IEEE Int. Things J. **1**(4), 349–359 (2014)
2. D. Blaauw, IoT design space challenges: Circuits and systems, *Symposium on VLSI Technology (VLSI-Technology): Digest of Technical Papers* (Honolulu, HI **2014**, 2014), pp. 1–2
3. J.B. Xu, Wendt, M. Potkonjak, Security of IoT systems: design challenges and opportunities, in *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD '14)* (IEEE Press, Piscataway, NJ, USA, 2014), pp. 417–423

4. D.G. KorzunSergey, I. BalandinAndrei, V. Gurtov, Deployment of Smart Spaces in Internet of Things: Overview of the Design Challenges: Internet of Things, Smart Spaces, and Next Generation Networking pp. 48–59 (2013)
5. S.C. Mukhopadhyay, N.K. Suryadevara, Internet of things: challenges and opportunities, in *Internet of Things. Smart Sensors, Measurement and Instrumentation*, ed. by S. Mukhopadhyay , vol 9 (Springer, Cham 2014)
6. Delta Encoding. https://en.wikipedia.org/wiki/Delta_encoding
7. M. Zaharia et al., Apache Spark: a unified engine for big data processing. Commun. ACM CACM Homepage **59**(11), 56–65 (2016)
8. E. Friedman, K. Tzoumas, *Introduction to Apache Flink: Stream Processing for Real Time and Beyond* (O'Reilly Media, Inc. 2016)
9. Apache Kafka. https://kafka.apache.org/
10. TensorFlow. https://www.tensorflow.org
11. Caffe. http://caffe.berkeleyvision.org/
12. PySpark. http://spark.apache.org/docs/2.2.0/api/python/pyspark.html/