Toward a Global Data Infrastructure

Nitesh Mor, Ben Zhang, John Kolb, Douglas S. Chan, Nikhil Goyal, Nicholas Sun,
Ken Lutz, Eric Allman, John Wawrzynek, Edward Lee, John Kubiatowicz
University of California, Berkeley

Abstract

The Internet of Things (IoT) represents a new class of applications that can benefit from cloud infrastructure. However, directly connecting smart devices to the cloud has a number of disadvantages and is unlikely to keep up with either the growing speed of the IoT or the diverse needs of IoT applications.

We explore these disadvantages and argue that fundamental properties of the IoT prevent the current approach from scaling. What is missing is a well-architected system extending functionality of the cloud and providing seamless interplay among the heterogeneous components in the IoT space. We argue that raising the level of abstraction to a data-centric design—focused around the distribution, preservation and protection of information—better matches the IoT. We present early work on such a distributed platform, called the Global Data Plane (GDP), and discuss how it addresses the problems with the cloud-centric architecture.

1 Introduction

The market has seen an explosion in the number of smart devices. These latest devices offer rich interactivity by connecting to computing platforms and services.[16] Fueled by the growth of *Internet* connectivity and the augmentation of everyday *things*, this shift is commonly referred to as the Internet of Things (IoT).[8]

On the "thing" side, we have seen a dizzying array of embedded platforms, from low-power microcontrollers to powerful computing units. They could be categorized into the following three types. First, low-power microcontrollers interface with sensors and actuators directly. Many of them come equipped with wireless radios to communicate but they are mainly targeted to run on batteries for years; therefore the computing power and the radio range are usually quite limited. The second category, smartphone devices, are mobile and moderately powerful, and can be used as sensing devices as well as gateways to connect other low-power devices to the Internet.[2] The third category consists of stationary but powerful mini PCs (Mac Mini, Intel NUC, Raspberry Pi, BeagleBone Black, etc.) that act as gateway devices for other low-power sensors.[5]

On the "Internet" side, the IoT industry has benefited tremendously from the economic model of the cloud as the central interconnection hub, computation resource and storage backend. With little investment in the infrastructure, even novice users can start collecting sensor data and stream it back to the cloud.[7] Riding on the popularity of the cloud, many of today's industrial[1,3] and academic[15,22] IoT solutions arise by connecting embedded platforms to the cloud.

At first glance, connecting devices to the cloud seems to be a natural architecture for IoT applications. However, several significant problems are revealed upon closer inspection, including issues with privacy, security, scalability, latency, bandwidth, availability and control over durability. While these problems are not new to typical web applications, they are exacerbated in the IoT space because of the fundamental differences between IoT and web services (see Sec. 2).

Our analysis suggests a need for a new layer of abstraction for the IoT—one that more naturally fits the requirements of IoT applications while exploiting the underlying computing platforms that enable the IoT (like the cloud, the Fog[10] and gateways[22]). Our proposed abstraction is centered around data. It is focused on the transport, replication, preservation, and integrity of streams of data while enabling transparent optimization for locality and quality of service. We call the resulting infrastructure the Global Data Plane (GDP). Its foundation is the concept of a single-writer append-only log coupled with location-independent routing, overlay multicast and higher level interfaces such as common access APIs (see Sec. 3).

In this article, we analyze the shortcomings of the existing architecture by explaining the fundamental differences between IoT applications and web services. We propose the design of a data-centric system, called the Global Data Plane (GDP), in order to address the issues in current IoT application development. Since this is an ongoing effort, we focus mainly on the design experience with GDP thus far.

2 Pitfalls with Today's Approach to IoT

Before discussing shortcomings of the current approach for IoT, we need to understand the application trends. IoT applications fall into two general categories:

Ambient data collection and analytics: These applications involve sensors installed in buildings,[12] in cities,[4] and on humans themselves.[2] Normally, data is not immediately inspected and the collected data is later processed for analytics. The magnitude of data collection is constantly growing and many researchers have predicted a new big-data problem.[23] One thing to note is that the collection of this sensor data may present privacy implications for personal health, operational security, *etc.*

Real-time applications with low-latency requirements: These applications could be reactive environments with humans in the loop. To provide a good user experience for human in the loop, an upper limit on latency is about 100 ms.[18] These applications could also be autonomous systems where humans are not involved (such as robots taking actions based on sensors). In this case, a tight control over latency is important for deterministic applications. Tight latency requirements are often incompatible with the unpredictable performance of cloud-based analytics or controllers.

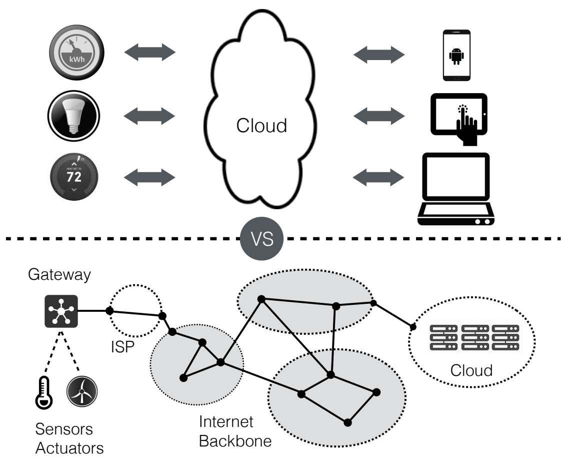

Figure 1: Although applications usually view the cloud as the center of all connected devices (*upper diagram*), in reality the cloud is usually on the edge of the Internet backbone, just like other devices (*lower diagram*).

Now given this broad classification, the current approach of connecting IoT devices directly to the cloud is incompatible with the evolving world of IoT application for the following reasons.

2.1 A Wrong Model

First of all, the architectural model of the cloud differs from reality. Application developers view the cloud as a component that interconnects smart devices. However, from a networking point of view, the cloud is on the edge of the network (see Fig. 1). Therefore, even simple IoT applications, such as automatically turning on lights when someone enters a room, will experience unpredictable latencies from sensing, wireless transmission, gateway processing, Internet delivery, and cloud processing. This model is also incompatible with the bandwidth distribution; typical consumer broadband networks have more downstream bandwidth than upstream bandwidth. IoT applications, however, generate data at the edges of the network, a pattern that could easily saturate the upstream link's bandwidth—especially at scale.

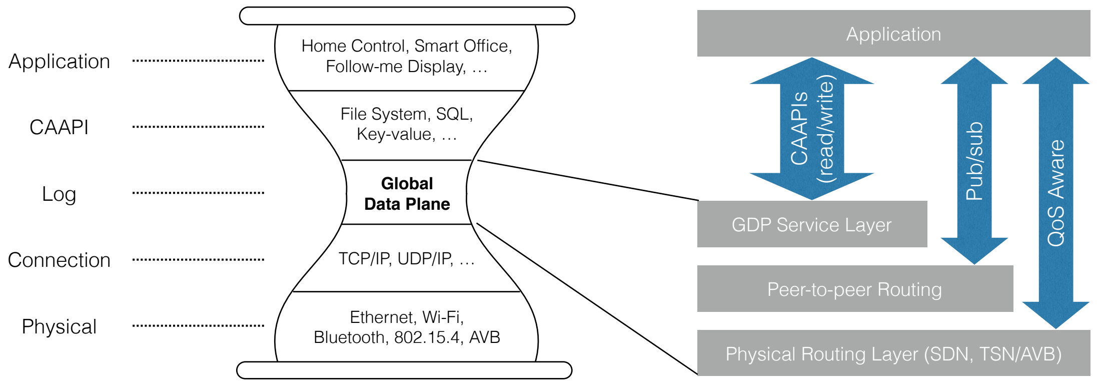
Application
CAAPI
Log
Connection
Physical
Home Control, Smart Office, Follow-me Display, ...
File System, SQL, Key-value, ...
Global Data Plane
TCP/IP, UDP/IP, ...
Ethernet, Wi-Fi, Bluetooth, 802.15.4, AVB
Application
CAAPIs (read/write)
Pub/sub
QoS Aware
GDP Service Layer
Peer-to-peer Routing
Physical Routing Layer (SDN, TSN/AVB)

Figure 2: The Global Data Plane (GDP) operates above the network level and offers Common Access APIs (CAAPIs) to applications rather than raw packet routing. We argue that this abstraction is appropriate for IoT applications both in the cloud and in distributed infrastructure.

Secondly, the usage model of the Internet differs from IoT applications. Tens of billions of devices[14] generating data will be interconnected in a few years. Techniques like caching popular items do not help in reducing the bisection bandwidth requirements for a centralized cloud solution, especially since most data acquired by IoT devices can or should be processed locally and immediately discarded.

2.2 Security and Privacy

Sensors implanted in our surrounding environment may collect extremely sensitive information. In a recent talk given by Wadlow,[21] he described the IoT as “hundreds of computers that are aware of me, can talk about me, and are out of my control.” This is a strong call for intrinsic security and privacy, echoed by others as well.[13,20] As a centralized resource out of users’ control, the cloud presents an ever-present opportunity to violate privacy, already a luxury,[6] and threatened further by the IoT.

2.3 Quality of Services

Guarantees on latency and availability are hard to realize. Web users tolerate variable latency and occasional loss of web services. In contrast, the temporary unavailability of sensors or actuators within IoT applications will directly impact the physical world. While significant engineering effort has been put into improving the availability and latency profile of the cloud (allowing Service Level Agreements), such efforts are stymied by operator error, software bugs, DDoS attacks, and normal packet-to-packet variations from wide-area routing. Further, the Internet connection to peoples’ homes is far from perfect even in developed world; this situation is worse in developing countries.

2.4 Durability Management

Some sensor data is ephemeral, while other data should be durable against global disasters. For ephemeral data, there is no effective way of verifying the data has been completely destroyed because the cloud is out of the user’s control. Moreover, the durability achieved by the cloud is typically done so without concern for application-specific privacy or export rules. Note that control over durability is closely related to control over data in general: making sure that users retain the control and ownership over their data rather than service providers.

3 GDP: A Data-Centric Proposal

In contrast to the existing cloud-centric model, we argue for the Global Data Plane (GDP), a data-centric abstraction focused around the distribution, preservation, and protection of information. It supports the same application model as the cloud, while better matching the needs and characteristics of the IoT by utilizing heterogeneous computing platforms such as small gateway devices, moderately powerful nodes in the environment and the cloud, in a distributed manner. The key mechanism for data storage and communication in GDP is the secure, single-writer log, which we describe in more detail later. As shown in Fig. 2, this log interface of the GDP provides a new “narrow waist” upon which applications are constructed.

3.1 System Overview

The concept of a log is central to the GDP. As the name suggests, a log is a time-series append-only data-structure addressed using a flat 256-bit identifier, called a *GDP-name*. Logs are lightweight, durable, potentially distributed over multiple physical machines, and don’t have a fixed location but rather are migrated as necessary to meet the locality, privacy, or QoS needs of applications. Logs are single-writer but support multiple simultaneous readers—either through random access (pull-based) or subscription (push-based).

Clients in the GDP are entities that read from or write to logs—this includes sensors that generate data, actuators that consume data, and various software entities in-between that process data by reading it from an input log and writing it to an output log (see Fig. 3). Logs are physically stored on *log-servers*—these log-servers can be small ubiquitous devices in homes, local servers in an enterprise, or powerful cloud based servers backed by existing cloud storage systems. Not only logs, but other entities in the GDP (clients, log-servers, etc) have flat 256-bit GDP-names.

GDP-routers are the routing elements that provide *location-independent routing* in this large, 256-bit address space. Our design is guided by the goal that a client should be able to operate without a single-point of trust in log-servers and GDP-routers. We hope to achieve this using a combination of cryptographic operations, trusted hardware and secure multi-party computation.

We also have a notion of a *Control Plane*—a set of services and applications that provide policy enforcement using mechanisms provided by the GDP. As an example, GDP makes sure that the logs are durable by ensuring replication across domains guided by a control plane replication service. The control plane and the GDP are closely integrated, yet have well specified boundaries.

We assume that devices (or a proxy gateway device) have cryptographic keys for signing and encryption. Signatures are used for verifying the origin, authenticity and integrity of data flow and control commands. Encryption is used wherever necessary to provide data secrecy. We are considering additional mechanisms for privacy that address concerns about leakage of information through timing or data format, such as combining data streams together with artificial noise.

3.2 Design Decisions

In this section, we elaborate on two key design choices: (1) log interface, and (2) flat Address space. We describe the rationale behind them and how they influence the rest of the system design.

3.2.1 The Log Interface

The majority of sensors and actuators in IoT can easily be represented by a stream of data, hence a queue-like interface for storing this streaming data seems to be the most obvious choice. However, the life-span of this data is application dependent. A log interface provides a wide range of options: logs could potentially be truncated for ephemeral data, or replicated widely for long lived data.

Properties of a Log: Logs are *append-only*; already existing data in a log is *read-only* and can be securely replicated and validated through cryptographic hashes and signatures. A *record* is the unit of read/write to a log; a log is essentially an ordered list of records. In addition, each log has immutable metadata created at the time of log-creation. For each log, our current design exposes *append*, *read* and *subscribe* APIs. Logs are single-writer, thus enabling serialization of records at client side. This implies that each sensor has its own log, however, aggregated logs representing more than one sensor can be created by reading multiple logs and writing back to the GDP. A single-writer log is minimal but complete interface that could be used to build richer interfaces.

A log is created by a client by issuing a signed *create-request*, which contains metadata including the public signature key of the designated writer. The create-request gets routed through a series of control plane services, the control plane checks whether the client is authorized to create a new log or not, allocates resources for this newly created log, sets up replication, etc.

Write access control is performed by the log-servers by validating signature on append operations against the designated writer's public signature key, while read access control is implemented by encrypting the payload and selectively sharing the decryption key. Since signatures remain with data, a malfunctioning or malicious log server is unable to fabricate data. More details on this are in Sec. 3.3.1.

In addition, a variety of basic control plane services could be used for making a log more functional. A replication service could set up multiple replicas of a log based on higher level policy decisions, such as level of durability, geographic span, etc. on a per log basis. A directory service could be used to associate human-readable names to 256-bit GDP-names on an organization level, or at a user level.

Benefits of a Log Interface: A log interface makes dumb sensors and actuators significantly more functional. Low-power sensors usually only generate data, but can not answer any queries. If data values are written to a log by such sensors, the log can be used as a proxy that supports a much richer set of queries, especially for historical data. A subscription to such a log provides latest sensor values in almost real time, thus virtualizing the sensor in some sense.

Actuators, on the other hand, usually need to maintain some kind of access control—by physical isolation, some authentication method, or a combination of both. Instead, if an actuator were to subscribe to an actuation log to read the actuation commands, access control can be implemented at the log level. This makes actuator design simpler and avoids the pitfalls of ad-hoc authentication mechanisms hastily put together by hardware vendors.

Representing sensors and actuators with logs separates policy decisions from mechanisms, enabling cleaner application designs. Applications can be built on top of GDP by interconnecting globally addressable log streams, rather than by addressing devices or services via IP addresses. Further, there is no need to expose the physical devices with potentially questionable standards of software security to the entire world, while still being able to connect things together. The "narrow waist" provided by such globally addressable logs avoids stove-piped solutions and provides for a heterogeneous hardware infrastructure.

Common Access APIs (CAAPIs): Although a log abstraction shelters developers from low-level machine and communication primitives, many applications are likely to need more common APIs or data structures. In fact, logs are sufficient to implement any convenient, mutable data storage repository. Thus, Fig. 2 shows a Common Access API (CAAPI) layer on top of the GDP. A CAAPI can present a key-value store, file system or database interface. Since logs serve as the ground truth, the benefit of consistency, durability, scalability and availability are carried over to CAAPIs for free.

3.2.2 Flat Address Space

As mentioned earlier, we use 256-bit long flat addresses for naming things in the GDP. This is true not only for logs, clients, log-servers, but also for control plane services and applications. In particular, logs are named with a 256-bit identifier which may be derived from a cryptographic hash of the owner's public key and meta-data.

This large address space allows us to employ *location-independent routing* that can better match the goals of flexible placement, controllable replication and mobility to optimize for latency, QoS, privacy and durability. Following a variety of placement and replication policies, GDP places logs within the infrastructure and advertises the location of these logs to the underlying routing layer.

GDP-routers take the burden of performing and optimizing this location-independent routing through an overlay network that uses a combination of Distributed Hash Table (DHT) technology and selective routing. DHT addresses the challenges of scalability with the sacrifice of an increased number of overlay hops. Important routes can be optimized by pushing routing entries into an underlying routing layer. Fig. 2 shows this layering.

Based on the interaction between GDP-routers and control-plane, logs could be migrated and routing topology altered dynamically. In addition, multicast trees can be built on top of the overlay network[9,25] to efficiently serve multiple subscribers.

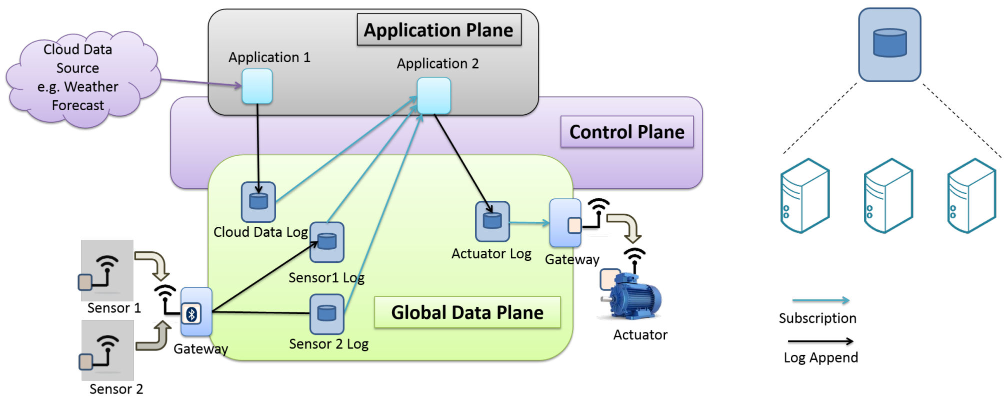

Figure 3: An IoT application uses GDP to combine heterogeneous data-streams from local environmental sensors and from a cloud source to actuate a device. Instead of direct communication with devices, the application uses the "narrow waist" (logs) provided by the GDP. Even though a log is represented as a single data-stream to an application, internally it can be distributed and replicated over multiple physical log servers to achieve locality and durability.

3.3 Challenges

In this section, we describe the major challenges that we faced in the design of GDP, and that are of concern to a general IoT framework.

3.3.1 Security and Privacy

As we have outlined in Sec. 2.2, data security and privacy is more important than ever, given the pervasive nature of devices and actuators. In the GDP, we design our security and privacy mechanisms and policies by focusing on the "narrow waist" provided by the logs.

Logs are stored on potentially untrusted log-servers. Hence it is important that we do not rely on a single log-server to provide data-integrity. An adversarial log-server could try to tamper with existing data in the logs it stores, or may not perform appropriate checks on access-control and accept writes from unauthorized writers, or maliciously re-order append operations received from a legitimate writer.

We propose to solve data-integrity and write access-control challenges by using signatures in our single-writer log model; a writer signs each append operation with a signature key and performs record-ordering on the client side by including a hash-pointer to the previous record in the signed content. The public signature key for the writer is included at the time of creation in the *create-request*, which itself is signed. All that a log-server has to do is to perform a signature-validation against this well-specified public key for any new append operation it receives. Any accidental or malicious behavior by a log-server results in invalid signatures or a broken chain of hash-pointers, and can be detected by a reader.

Globally addressable logs are a significant privacy concern if any unauthorized reader could read data at will. We envision encryption to be the mechanism for providing data secrecy. GDP does not assume any structure on the data being written to a log, enabling applications to encrypt data before handing it to GDP. This enforces the minimum trust philosophy by putting trust in cryptographic constructs rather than potentially buggy software running on untrusted servers. Read access-control is managed by the application by appropriate sharing of the decryption keys.

A secondary concern is exposing encrypted data to adversaries who may analyze timing or data size at will. To address these concerns, we propose a collaboration between policy (at the control plane) and routing (within the GDP) to help mitigate this problem by controlling the placement of data logs and path of updates.

3.3.2 Key-management

Since security and privacy in the GDP relies upon encryption, key management will be of paramount importance. Although still a work in progress, we propose a basic key management scheme as follows.

Each user maintains an encrypted wallet and an unencrypted public-key registry, both backed by logs. The user-supplied root key is used to decrypt the wallet, which contains the secret keys necessary to sign requests and secret or symmetric keys necessary to decrypt log entries.

Granting read access amounts to sending a bit-string over a tamper-proof channel (a log) to a remote entity; this bit string is the necessary decryption key that is in turn encrypted using the public key of the remote entity. As a very simple example: Alice wants to share a log L with a set of users. Alice creates the log L including the name of an "access control log" A in L's metadata. Alice then encrypts the contents of L with a symmetric key K and appends versions of K to A, each encrypted using the public key of the users who should have access.

This scheme works for simple and static data sharing scenarios. Slightly complicated but efficient hierarchical key management schemes can be created based on application requirements. In the extreme case, a compute service in a trusted environment could be designated as the only reader, with that service managing read access control lists in more traditional ways.

4 Related Work

Other efforts exist to address the challenges of IoT. Cisco's Fog Computing[10] provides computing resources closer to the edge of the network. We believe that our arguments strengthen the need for fog-like computing platforms and our proposed GDP architecture can leverage such resources. Also relevant are systems such as EdgeComputing from Akamai,[11] and Cloudlet.[19] In these architectures, the role of servers is to be intelligent gateways or proxies for data flowing into and from the cloud. Support for an entirely decentralized data storage and delivery platform is apparently absent.

Our data-centric design hails from Oceanstore[17] and shares a number of goals with Named Data Networking (NDN),[24] but our focus on the IoT application space leads to a number of important design differences. Among other things, push based communication—such as from sensors to logs and from logs to consumers—represents a communication style utilized extensively in the IoT space and deemphasized in NDN. A few of our design decisions are similar to Bolt:[15] single-writer time-series data, chunking for performance, efficient data sharing, policy-driven storage and data confidentiality/integrity. However, Bolt takes the cloud approach where the pitfalls in Sec. 2 are unavoidable.

5 Conclusions

A prototype version of the GDP has been depolyed within our own environment and has been running on a few servers since early 2015, however it is still a work in progress. Our design for the GDP is not yet bullet-proof and our initial implementation has not withstood the test of wide-scale deployment. Nonetheless, we believe that the core concepts of GDP overcome the pitfalls mentioned in Sec. 2 in the following way: the single-writer, append-only log models sensor data more accurately; integrity and authentication by design provides better privacy and security; the distributed nature with peer-to-peer technology makes scalability possible; explicit separation of policy from mechanism enables better control on level of durability for end users; and finally, latency, bandwidth and QoS guarantees are enabled by the integration of the cloud and the local infrastructure.

6 Acknowledgments

This work was supported in part by the Ubiquitous Swarm Lab at UC Berkeley. This work was also supported in part by TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

References

[1] Carriots. `https://www.carriots.com/`.

[2] Fitbit. `http://www.fitbit.com/`.

[3] Samsung SAMI. `https://developer.samsungsami.io/`.

[4] SFPark. `http://sfpark.org/`.

[5] SmartThings. `http://www.smartthings.com/`.

[6] ANGWIN, J. Has Privacy Become a Luxury Good? `http://www.nytimes.com/2014/03/04/opinion/has-privacy-become-a-luxury-good.html`, 2014.

[7] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al. A view of cloud computing. *Communications of the ACM 53*, 4 (2010), 50–58.

[8] Atzori, L., Iera, A., and Morabito, G. The internet of things: A survey. *Computer networks 54*, 15 (2010), 2787–2805.

[9] Ballardie, T., Francis, P., and Crowcroft, J. Core based trees (CBT). In *ACM SIGCOMM Computer Communication Review* (1993), vol. 23, ACM, pp. 85–95.

[10] Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing* (2012), ACM, pp. 13–16.

[11] Davis, A., Parikh, J., and Weihl, W. E. Edgecomputing: extending enterprise applications to the edge of the internet. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters* (2004), ACM, pp. 180–187.

[12] Dawson-Haggerty, S., Jiang, X., Tolle, G., Ortiz, J., and Culler, D. sMAP: a simple measurement and actuation profile for physical information. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems* (2010), ACM, pp. 197–210.

[13] Drozhzhin, A. Internet of Crappy Things. `http://blog.kaspersky.com/internet-of-crappy-things/`, 2015.

[14] Evans, D. The internet of things: How the next evolution of the internet is changing everything. *CISCO white paper 1* (2011).

[15] Gupta, T., Singh, R. P., and Mahajan, A. P. J. J. R. Bolt: Data management for connected homes. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)* (2014), pp. 243–256.

[16] Hartmann, B., and Wright, P. K. Designing bespoke interactive devices. *Computer*, 8 (2013), 85–89.

[17] Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., et al. Oceanstore: An architecture for global-scale persistent storage. *ACM Sigplan Notices 35*, 11 (2000), 190–201.

[18] Nielsen, J. *Usability engineering*. Elsevier, 1994.

[19] Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE 8*, 4 (2009), 14–23.

[20] Stanislav, M., and Lanier, Z. The Internet of Fails: Where IoT Has Gone Wrong and How We're Making It Right. `https://www.defcon.org/html/defcon-22/dc-22-speakers.html`, 2014.

[21] Wadlow, T. The Questions are the Same, but the Answers are Always Changing. `https://swarmlab.eecs.berkeley.edu/events/2014/11/18/5165/questions-are-same-answers-are-always-changing`, 2015.

[22] Zachariah, T., Klugman, N., Campbell, B., Adkins, J., Jackson, N., and Dutta, P. The Internet of Things Has a Gateway Problem. In *HotMobile'15* (2015), ACM, pp. 27–32.

[23] Zaslavsky, A., Perera, C., and Georgakopoulos, D. Sensing as a service and big data. *arXiv preprint arXiv:1301.0159* (2013).

[24] Zhang, L., Afanasyev, A., Burke, J., Jacobson, V., Crowley, P., Papadopoulos, C., Wang, L., Zhang, B., et al. Named Data Networking. *ACM SIGCOMM Computer Communication Review 44*, 3 (2014), 66–73.

[25] Zhao, B. Y., Kubiatowicz, J., Joseph, A. D., et al. Tapestry: An infrastructure for fault-tolerant wide-area location and routing.