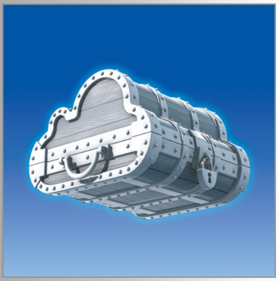

Toward a Global Data Infrastructure

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 multiple disadvantages and is unlikely to keep up with the growing speed and diverse needs of IoT devices/applications. Here, the authors argue that fundamental properties of the IoT prevent the current approach from scaling. What's missing is a well-architected system extending cloud functionality and providing seamless interplay among heterogeneous components closer to the edge in the IoT space. To address such problems, they present their distributed platform, the Global Data Plane.

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

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. 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). With the ever-growing proliferation of cloud computing and storage services, even novice users can start deploying sensors and actuators with minimal investment in infrastructure. However, issues of privacy, security, scalability, latency, and bandwidth availability — which already are a challenge for Web applications — are exacerbated in the IoT space because of the fundamental differences between IoT and Web services.

In this article, we analyze the shortcomings of existing architectures by explaining the fundamental differences between IoT applications and Web services. Our analysis suggests a need for a new abstraction layer for the IoT — one that more naturally fits the requirements of IoT applications while exploiting the underlying computing platforms that enable the IoT (such as the cloud, the fog,[1] and gateways). Although influenced by the needs of IoT, there's no reason this layer of abstraction can't be used for other scenarios.

Our new 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 (QoS). 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

1089-7801/16/$33.00 © 2016 IEEE

Related Systems and Architectures in the IoT Space

Other efforts exist to address the challenges of the Internet of Things (IoT). Ericsson's Capillary networks[1] and Cisco's Fog Computing[2] provide computing resources closer to the edge of the network. We believe that our arguments strengthen the need for fog-like computing platforms and our Global Data Plane (GDP) architecture can both leverage and simultaneously enable such platforms. Also relevant are systems such as EdgeComputing from Akamai,[3] and Cloudlet.[4] In these architectures, servers act as intelligent gateways or proxies for data flowing into and out of the cloud. Support for an entirely decentralized data storage and delivery platform is apparently absent.

Our data-centric design hails from OceanStore[5] and shares a number of goals with Named Data Networking (NDN)[6] and MobilityFirst,[7] 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 used extensively in the IoT space and deemphasized in NDN. MobilityFirst shares the fundamental design principle of a flat-address (a globally unique identifier, or GUID); however, the emphasis is primarily on communication. In contrast, GDP provides a higher-level log abstraction supported by underlying communication primitives.

A few of our design decisions are similar to Bolt,[8] an approach using single-writer, time-series data, chunking for performance, efficient data sharing, policy-driven storage, and data confidentiality and integrity. However, Bolt takes the cloud approach where pitfalls are unavoidable.

References

1. J. Sachs et al., "Capillary Networks — A Smart Way to Get Things Connected." *Ericsson Rev.*, 9 Sept. 2014; www.ericsson.com/res/thecompany/docs/publications/ericsson_review/2014/er-capillary-networks.pdf.
2. F. Bonomi et al., "Fog Computing and Its Role in the Internet of Things," *Proc. 1st Edition of the MCC Workshop on Mobile Cloud Computing*, 2012, pp. 13–16.
3. A. Davis, J. Parikh, and W.E. Weihl, "EdgeComputing: Extending Enterprise Applications to the Edge of the Internet," *Proc. 13th Int'l World Wide Web Conf. Alternate Track Papers and Posters*, 2004, pp. 180–187.
4. M. Satyanarayanan et al., "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, no. 4, 2009, pp. 14–23.
5. S.C. Rhea et al., "Pond: The OceanStore Prototype," *Proc. 2nd Usenix Conf. File and Storage Technologies*, 2003; www.usenix.org/legacy/events/fast03/tech/rhea/rhea.pdf.
6. L. Zhang et al., "Named Data Networking," *ACM Sigcomm Computer Comm. Rev.*, vol. 44, no. 3, 2014, pp. 66–73.
7. I. Seskar et al., "MobilityFirst Future Internet Architecture Project," *Proc. 7th Asian Internet Eng. Conf.*, 2011; doi:10.1145/2089016.2089017.
8. T. Gupta, R.P. Singh, and A.P.J.J.R. Mahajan, "Bolt: Data Management for Connected Homes," *Proc. 11th Usenix Symp. Networked Systems Design and Implementation*, 2014, pp. 243–256.

common access APIs. (For others' work in this area, see the related sidebar.) We also discuss how a communication and storage platform can enable better security by reducing the attack surface of potentially vulnerable end devices. Because this is an ongoing effort, here we focus mainly on the design experience with GDP thus far.

Pitfalls with Today's Approach to IoT

Looking at recent trends, IoT applications fall into two general categories:

- *Ambient data collection and analytics*. These applications involve sensors installed in buildings, in cities, and on humans themselves. Normally, data aren't immediately inspected and the collected data are later processed for analytics.
- *Real-time applications with low-latency requirements*. These applications could be either autonomous systems with tight control loops (for example, robots taking actions based on sensors), or reactive environments with humans in the loop. Unpredictable latencies of cloud-based solutions make this challenging.

It's not unusual to use the same infrastructure for both categories of IoT applications. Performing actuation in real time using local sensor data, but with global knowledge of long-term trends to optimize on various parameters, is a common practice. With this context, relying entirely on the cloud for IoT applications raises concerns about resource optimality, security and privacy, QoS, and durability management.

The suboptimal cloud. Application developers view the cloud as an interconnection hub for smart devices. However, from a networking point of view, the cloud is on the edge of the network, leading to unpredictable latencies (see Figure 1). For Web applications, a centralized hub enables amortization of resources and enables economies

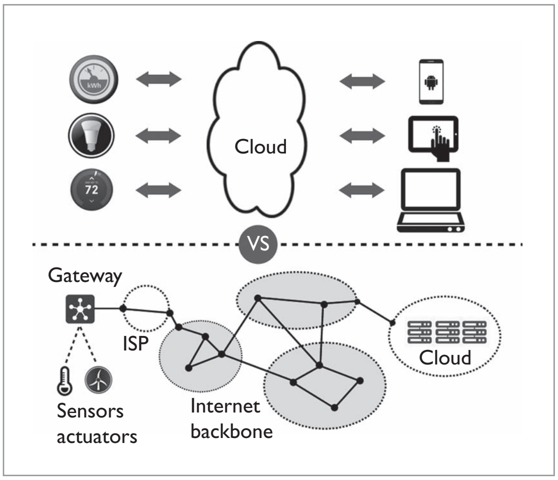

Figure 1. Although applications usually view the cloud as the center of all connected devices (upper diagram), in reality the cloud is usually on the edge of the Internet backbone, just like other devices (lower diagram).

of scale (by caching popular resources, reduced management overhead, and so on), and occasional latencies are often ignored in favor of such cost benefits. IoT applications, on the other hand, are highly specific in their execution pattern and don't necessarily benefit from such centralization.

Security and privacy. Sensors implanted in our surrounding environment might collect extremely sensitive information. As a centralized resource out of users' control, the cloud presents an ever-present opportunity to violate privacy, which is already a luxury and is threatened further by the IoT. With appropriate encryption techniques, data stored in the cloud can be protected from unauthorized use. However, an end-to-end secure data flow is difficult to achieve because most applications can't process encrypted data. Further, there's information leakage from side-channels based on access patterns. Even with the state of the art (homomorphic encryption, secure hardware containers, and so on), absolute end-to-end security is a challenging technical problem. Giving users some control over where applications execute is a more general solution, especially when low latency requirements also require such flexibility of execution.

QoS requirements. Guarantees on latency and availability are difficult to realize. Web users tolerate variable latency and the occasional loss of Web services. In contrast, the temporary unavailability of sensors or actuators within IoT applications can directly impact the physical world. While significant engineering effort has been put into improving the cloud's availability and latency profile, such efforts are stymied by operator error, software bugs, distributed denial of service (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 the developed world; this situation is worse in developing countries.

Durability management. Some sensor data is ephemeral, while other data should be durable against global disasters. For ephemeral data, there's no effective way of verifying that the data has been completely destroyed, because the cloud is out of the user's control. Control over durability is closely related to control over data in general, for example, users retaining the control and ownership over their data rather than service providers.

GDP: A Data-Centric Proposal

In contrast to the existing cloud-centric model, we argue for the GDP, a data-centric abstraction focused around the distribution, preservation, and protection of information. GDP 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 Figure 2 shows, this log interface of the GDP provides a new "narrow waist" upon which applications are constructed.

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*. A log is an *authenticated data structure*[2] stored on potentially untrusted infrastructure. 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

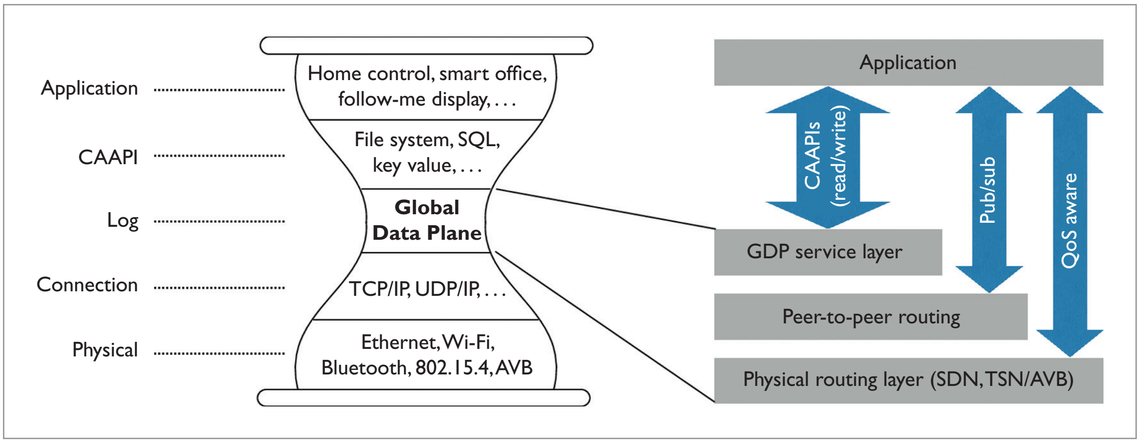
Application
CAAPI
Log
Connection
Physical
Home control, smart office, follow-me display, . . .
File system, SQL, key value, . . .
Global Data Plane
TCP/IP, UDP/IP, . . .
Ethernet, Wi-Fi, Bluetooth, 802.15.4, AVB
Application
CAAPIs (read/write)
Pub/sub
QoS aware
GDP service layer
Peer-to-peer routing
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 Internet of Things (IoT) applications, both in the cloud and in a distributed infrastructure.

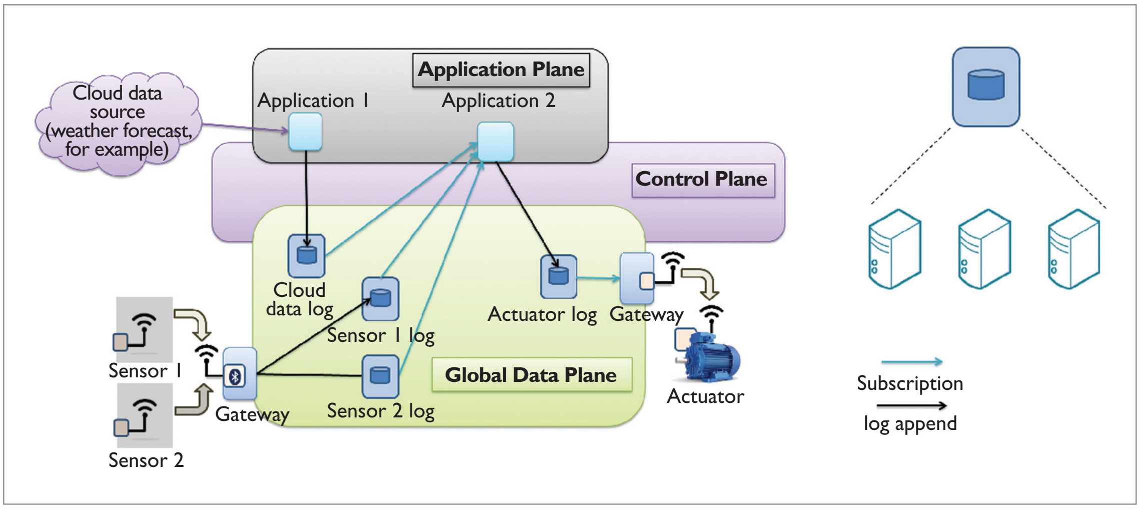

Figure 3. An IoT application uses GDP to combine heterogeneous data streams from local environmental sensors and from a cloud source to actuate a device. Instead of direct communication with devices, the application uses the "narrow waist" (logs) provided by the GDP. Even though a log is represented 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.

needs of applications. Logs are single-writer but support multiple simultaneous readers – either through random access (pull-based) or subscription (push-based). Not only logs, but other entities in the GDP (clients, log-servers, and so on) have flat 256-bit GDP-names.

Clients in the GDP are entities that read from or write to logs – this includes sensors that generate data, actuators that consume data, gateway devices, and various software entities in-between that process data by reading it from an input log and writing it to an output log (see Figure 3). We provide an event-driven communication library for interaction with logs. Moderately powerful gateway devices (smartphones, Raspberry Pi's, and so on) are a particularly important case: they connect to the GDP on behalf of sensors and actuators limited in computation or communication capabilities, and enable wide adoption with minimal changes in end devices. We assume that clients have cryptographic capabilities, including key storage (for encryption or decryption and signatures). 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 describe this in more detail later.

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. *GDP-routers* are the routing elements that provide *location-independent routing* in this large, 256-bit address space using an overlay network. We have a preliminary version of the GDP-router implemented in Click[3] on top of TCP/IP.

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 multiparty 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.

The Log Interface

The majority of sensors and actuators in IoT easily can 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 lifespan 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 the 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 another log. A single-writer log is a minimal-but-complete interface that could be used to build richer interfaces.

Even though a log is append-only, log-truncation policies can be specified on a per-log basis. Log-truncation marks the data older than a specific threshold *safe to delete* by the infrastructure. Guaranteed destruction of data by a remote entity is practically impossible. However, because the GDP allows clients to specify where data are stored, it's much easier to control the longevity of data than in an exclusively cloud-based approach. Further, a client can encrypt a range of data with a unique key, and destroy the decryption key for data older than a certain threshold while maintaining data integrity-related invariants of the log data structure.

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, allocates resources for this newly created log, sets up replication, and so on.

Write access control is performed by the log-servers by validating the 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. Because signatures remain with data, a malfunctioning or malicious log-server is unable to fabricate data. We'll provide more details on this later.

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 the level of durability, geographic span, and log-truncation policies — on a per-log basis. A directory service could be used to associate human-readable names to 256-bit GDP-names on an organization or 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't 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 the 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 could 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.

Further, there's 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. This is especially important because it takes the burden of implementing security off the device vendors' shoulders. All a device manufacturer has to do is publish data to a log (in case of a sensor), or subscribe to a log (in case of an actuator). This greatly reduces the attack surface, because GDP as a platform implements security best practices.

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, with applications running inside containers (Docker, Unikernels, Intel SGX enclaves, and so on), forcing data-flows in and out of the container through logs enables any filtering at the log level (for example, access control).

Last but not least, the narrow waist provided by globally addressable logs avoids stove-piped solutions and provides for a heterogeneous hardware infrastructure.

Beyond a log interface: 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. Figure 2 shows a CAAPI layer on top of the GDP. A CAAPI can present a key-value store, file system, or database interface. Because logs serve as the ground truth, the benefit of consistency, durability, scalability, and availability are carried over to CAAPIs for free.

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, and log-servers, but also for Control Plane services and applications. In particular, logs are named with a 256-bit identifier, which might be derived from a cryptographic hash of the owner's public key and metadata.

This large address space lets us 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, possibly using software-defined networking (SDN) routers when available. Figure 2 shows this layering.

Based on the interaction between GDP-routers and the Control Plane, logs could be migrated and routing topology altered dynamically. In addition, multicast trees can be built on top of the overlay network[4,5] to efficiently serve multiple subscribers. Not only migration, but location-independent flat-addressing also enables replication; a log (or service) can have multiple read-only copies spread throughout the network. Single-writer, append-only design makes a log mostly read-only (except for the active head), which makes for easy replication with simpler concurrency issues. These read-only replicas act much like a content delivery network (CDN) and provide for fault tolerance. In case of network events such as flash crowds or targeted bandwidth-saturation attacks, the traffic is distributed over multiple replicas rather than a single point of failure. In addition, because low-power devices (sensors or actuators) are exposed to the world as a virtual device (in the form of a log), any potential bandwidth saturation is limited to the GDP and doesn't turn into a battery-exhaustion attack.

Challenges

Next, we describe the major challenges that we faced in GDP's design, especially those of concern to a general IoT framework.

Security and privacy. As we previously outlined, data security and privacy are 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's important that we 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 might not perform appropriate checks on access control and accept writes from unauthorized writers, or maliciously reorder append operations received from a legitimate writer.

We address data integrity and write access control challenges using a combination of signatures and Merkle trees 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 must do is 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, detectable 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 doesn't assume any structure on the data being written to a log, enabling applications to encrypt data before handing it to the 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 appropriately sharing decryption keys.

A secondary concern is exposing encrypted data to adversaries who might analyze timing or data size at will. To address these concerns, we envision 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 the path of updates.

Key management. Because security and privacy in the GDP relies upon encryption, key management will be of paramount importance. Although advanced schemes for key management are still under consideration, we support 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 simple example: Alice wants to share a log L with a set of users. Alice creates the log L, with 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 computing service in a trusted environment could be designated as the only reader, with that service managing read access control lists in more traditional ways.

Routing overhead. A flat address space offers several benefits, but a naive overlay implementation can severely affect the performance, especially roundtrip latencies. However, reasonably dense, locality-aware distributing routing frameworks limit relative delay penalty (the ratio between the distance traveled between two points using an overlay and the minimum distance between the two points) of using a DHT to 2–3.[5] We propose using locality to overcome the overlay performance penalty; for example, roundtrip latency to a close-by or local node via overlay (10–20 ms) is still better than roundtrip latency to a cloud datacenter (> 50 ms). Further, with support for optimizing flows from the underlying networks (SDN, audio-video bridging, and so on), long-term communication to and from logs can achieve better performance.

A prototype version of the GDP has been deployed within our own environment and has been running on a few servers since early 2015; however, it's still a work in progress. Our design for the GDP is not yet bulletproof and our initial implementation hasn't withstood the test of wide-scale deployment. Nonetheless, we believe that the core concepts of GDP overcome the pitfalls of a purely cloud-based approach in the following ways: the single-writer, append-only log models sensor data more accurately; integrity and authentication by design provide 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 the level of durability for end users; and finally, latency, bandwidth, and QoS guarantees are enabled by integrating the cloud and local infrastructures.

Acknowledgments

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

References

1. F. Bonomi et al., "Fog Computing and Its Role in the Internet of Things," *Proc. 1st Edition of the MCC Workshop on Mobile Cloud Computing*, 2012, pp. 13–16.
2. R. Tamassia, "Authenticated Data Structures," *Algorithms-ESA 2003*, Springer, 2003, pp. 2–5.
3. R. Morris et al., "The Click Modular Router," *ACM Sigops Operating Systems Rev.*, 1999, vol. 33, pp. 217–231.
4. T. Ballardie, P. Francis, and J. Crowcroft, "Core-Based Trees (CBT)," *ACM Sigcomm Computer Comm. Rev.*, vol. 23, 1993, pp. 85–95.
5. B.Y. Zhao et al., *Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing*, tech. report UCB/CSD-01-1141, Computer Science (EECS), Univ. of California, Berkeley, 2001; https://oceanstore.cs.berkeley.edu/publications/papers/pdf/tapestry_sigcomm_tr.pdf.

Nitesh Mor is a graduate student in the Electrical Engineering and Computer Science (EECS) Department and a member of the Ubiquitous Swarm Lab at the University of California, Berkeley. His research interests include computer security, privacy, and distributed systems. Mor has an MS in computer science from the University of California, Berkeley. Contact him at mor@eecs.berkeley.edu.

Ben Zhang is a graduate student in the EECS Department at the University of California, Berkeley, and a member of the TerraSwarm Research Center. His research interests include mobile computing, networking, and distributed systems. Zhang has a BE in electronic engineering from Tsinghua University. Contact him at benzh@berkeley.edu.

John Kolb is a PhD student in the EECS Department at the University of California, Berkeley. His research interests include distributed and embedded systems. Kolb has a BS in computer science from the University of Minnesota. Contact him at jkolb@cs.berkeley.edu.

Douglas S. Chan is a visiting researcher scholar at the Ubiquitous Swarm Lab and Berkeley Wireless Research Center at the University of California, Berkeley. His research interests include wireless LANs, fog computing, data analytics, and the IoT. Chan has a PhD in electrical and computer engineering with a minor in applied mathematics from Cornell University. Contact him at douglas.chan@ieee.org.

Nikhil Goyal is a software engineer in the Exadata group at Oracle, America. His research interests include distributed systems, databases, and computer networks. Goyal has an MS in computer science from the University of California, Berkeley. Contact him at ngoyal@berkeley.edu.

Nicholas Sun is an undergraduate at the University of California, Berkeley. His research interests include distributed systems and virtualization. Contact him at nlsun@berkeley.edu.

Ken Lutz is a research engineer and executive director at the Ubiquitous Swarm Lab at the University of California, Berkeley. His interests include the Industrial IoT and distributed energy storage systems. Lutz has a BS in electrical engineering from the University of California, Davis. Contact him at lutz@berkeley.edu.

Eric Allman is a research software engineer at the Ubiquitous Swarm Lab at the University of California, Berkeley. His research interests include distributed systems and databases. Allman has an MS in computer science from the University of California, Berkeley. Contact him at eric@cs.berkeley.edu.

John Wawrzynek is a professor in the EECS Department at the University of California, Berkeley, and a director of the Berkeley Wireless Research Center. His research interests include computer architecture, reconfigurable computing, and wireless systems. Wawrzynek has a PhD in computer science from the California Institute of Technology. He's a member of ACM and IEEE. Contact him at johnw@berkeley.edu.

Edward A. Lee is the Robert S. Pepper Distinguished Professor in the EECS Department at the University of California, Berkeley. His research interests include design, modeling, and analysis of embedded, real-time computational systems. Lee has a PhD in electrical engineering and computer science from the University of California, Berkeley. Contact him at eal@berkeley.edu.

John Kubiatowicz is a professor in the EECS Department at the University of California, Berkeley, and a director of the Ubiquitous Swarm Lab. His research interests include Internet-scale storage, networking, computer architecture, and quantum computing. Kubiatowicz has a PhD in electrical engineering and computer science from the Massachusetts Institute of Technology. He's a member of ACM and IEEE. Contact him at kubitron@cs.berkeley.edu.

cn *Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.*