Towards Secure and Efficient Communication for the Internet of Things

Xin Li, *Student member, IEEE* and Minmei Wang, *Student member, IEEE* and Huazhe Wang, *Student member, IEEE* and Ye Yu, *Student member, IEEE* and Chen Qian, *Member, IEEE*

***Abstract*—Internet of Things has been widely applied in everyday life, ranging from transportation and healthcare, to smart homes. As most IoT devices carry constrained resources and limited storage capacity, sensing data need to be transmitted to and stored at resource-rich platforms, such as a cloud. IoT applications need to retrieve sensing data from the cloud for analysis and decision-making purposes. Ensuring the authenticity and integrity of the sensing data is essential for the correctness and safety of IoT applications. We summarize the new challenges of the IoT data communication with authenticity and integrity and argue that existing solutions cannot be easily adopted to resource-constraint IoT devices. We present two solutions, called Dynamic Tree Chaining (DTC) and Geometric Star Chaining (GSC) that provide efficient and secure communication for the Internet of Things. Extensive simulations and prototype emulation experiments driven by real IoT data show that the proposed system is more efficient than alternative solutions in terms of time and space.**

***Index Terms*—IoT, Cloud, Authentication, Partial sample-rate data retrieval**

I. Introduction

Internet of Things (IoT) is being widely applied in a great number of everyday applications such as healthcare [2], [3], transportation [4], [5], smart home [6]–[8], and surveillance systems [9], [10]. IoT devices usually generate a large amount of sensing data to reflect physical environments or conditions of objects and human beings. As most IoT devices carry constrained resources and limited storage capacity, sensing data need to be transmitted to and stored at resource-rich platforms, such as a cloud. On the other hand, analyzing historical sensing data is essential for decision-making in various IoT applications [6] [11]. To this end, both state-of-art IoT proposals [8], [12] and industrial IoT practices [13] adopt the centralized data store residing in the cloud, aiming that the sensing data can be stored economically and retrieved effectively for analysis, as depicted in Fig. 1.

In this paper, we present the design of an IoT data communication involving the three key entities: *sensing devices*, *cloud*, and *data applications*. We summarize the following key requirements or challenges of the IoT data communication, which distinguish it from traditional data collection and management methods.

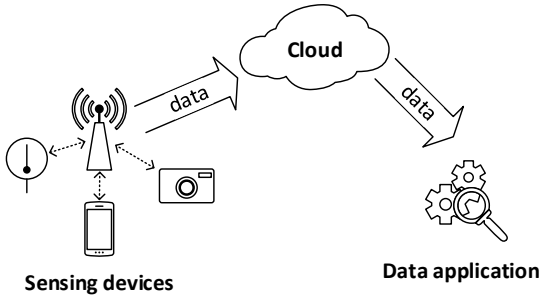


Fig. 1. Overview of the IoT data communication

1) **Authenticity and integrity.** Since the sensing data are stored in a third-party cloud, data authenticity and integrity, which guarantee that data are from these sensing devices and have not been modified, are important for trustworthy IoT applications [14]. However the data could be corrupted by outside attackers [15]–[17], malicious cloud employees [18], transmission failures, or storage loss [19]. Without data authenticity and integrity, IoT applications may make wrong decisions and cause economic and human-life losses. Authenticity and integrity should be *verifiable* by data applications.

2) **Data random sampling.** A common but critical problem shared by state-of-art IoT designs is that the resources for transmitting and storing data (e.g. network bandwidth, storage quota) are limited in presence of massive IoT data. By 2022, the IoT data is expected to constitute 45% traffic in the Internet [20]. Cloud providers charge users for storage, retrieval and transferring of data [21]. It is desired to have *predictable cost* for both users (in finance) and the cloud (in resource) [22]. Hence only a fixed resource budget can be allocated to the sensing data over a time period, called an *epoch*. For example, the cloud can only keep 100 data records from any device collected during every minute. Random sampling is widely used in IoT [23]. To guarantee representative samples, every event should have an equal probability to be sampled, which is called the *uniformity* property. Uniformity is essential for unbiased statistics estimation such as histogram [24], median [25] and average [26].

3) **Flexible application requirements.** Different applications may have different requirements on sending data granularity. For example, applications like self-driving cars need fine-grained road information, while other applications like road-traffic estimation only need a few sampled data. Even if the cloud can store up to 100 records, some applications

Xin Li, Minmei Wang, Huazhe Wang and Chen Qian are with the Department of Computer Science and Engineering, University California Santa Cruz, Santa Cruz, CA 95064. E-mail: xinli@ucsc.edu, mwang107@ucsc.edu, huazhe.wang@ucsc.edu, cqian12@ucsc.edu. Ye Yu is with Google Inc., Mountain View, CA 94043. E-mail: ye.yu@uky.edu.

The preliminary version of this paper [1] appeared at the ACM/IEEE International Conference on Internet of Things Design and Implementation (IoTDI), 2017.

The authors were supported by National Science Foundation grants CNS-1701681 and CNS-1717948.

only retrieve part of them, e.g., 10 records, due to bandwidth limit, memory limit, or application requirements. In addition, these 10 records should have verifiable authenticity, integrity, and uniformity. We call this feature *partial sample-rate data retrieval*.

We summarize our contribution in this paper as follows.

1) Our first effort is to extend a well-known digital signature, Merkle tree [27], [28], to Dynamic Tree Chaining (DTC). DTC supports verifiable authenticity and integrity with low space overhead at the IoT devices. DTC enables the receiver to verify any single message and thus features partial sample-rate data retrieval.

2) Our another contribution is an efficient digital signature method, Geometric Star Chaining (GSC), which is designed for the IoT data communication. GSC allows each sensing device to sign only once for all data records in an epoch and provides verifiable authenticity, integrity, and uniformity for partial sample-rate data retrieval. GSC is preferable to DTC when single message verification is not needed because GSC could achieve better performance with less space overhead.

3) Efficiency and uniformity of data sampling that come along with DTC and GSC are the highlight and distinguish this work from existing solutions.

4) We investigate the problem of shared budget constraint for a group of sensing devices and extend DTC as well as GSC to resolve this problem.

The rest of the paper is organized as follows. Some preliminaries to understand this paper is first described in Sec. II. We present the problem statement in Sec. III. We describe the system design details and extend it to incorporate budget limit in Sec. IV and Sec. V respectively. Security and performance analyses are presented in Sec. VI and Sec. VII. We conduct extensive simulation and prototype emulation in Sec. VIII. Sec. IX presents related work. Some practical issues are discussed in Sec. X. Finally, we conclude this work in Sec. XI.

II. Preliminaries of Digital Signature

There have been mature solutions for any individual requirement that has been studied by researchers for over decades. However, no existing solution collectively resolves all requirements of emerging IoT applications. The difficulty stems from the combination of seemly conflicting requirements. *Digital signature* is a widely used method to protect data authenticity and integrity: The sender first computes a message digest D by hashing its original message m using a cryptographic hash function H, $D = H(m)$. H is also called message digest function. Note the length of D is significantly shorter than that of m. Then the sender uses its private key k to encrypt D and attaches the *signature* $E_k[D]$ to the original message. When the receiver gets m and $E_k[D]$, it decrypts $E_k[D]$ using the public key of the sender and verifies whether $D = H(m)$. However, applying the digital signature to every sensing record, called the *Sign-each* method, is not practical, because public-key encryption/decryption is considered slow and expensive, especially for sensing devices with limited resources. We test the performance of some mostly used cryptographic operations on M3, one mainstream IoT hardware platform available from one public testbed [29]. This hardware platform features one 32-bit ARM Cortex-M3 CPU@72MHz. The result shows the average time and energy to encrypt (RSA/DSA) or to compute hash (MD5/SHA1/SHA256) over a 10-byte string. The result is presented in TABLE I. Even though there is great advances in hardware performance compared to prior platforms [30], directly applying RSA/DSA is still not suitable for resource-constraint IoT devices, especially for those powered by batteries. A more efficient method, *concatenate*, is to compute the message digest D for a large number of records and sign once on D. This approach requires each sensing device to cache all records and has the all-or-nothing feature: if some applications only require part of the records, the signature cannot be verified. A well-known method to sign a data stream is hash chaining [31]. However, it does not fit the IoT communication model either, because sampling and partial sample-rate data retrieval will break the chain and hence make the signature unverifiable.

TABLE I
Performance of some cryptographic operations

Metric	RSA	DSA	MD5	SHA1	SHA256
Time (ms)	203.6	192.8	0.041	0.051	0.098
Energy (mJouel)	37.6	36.2	0.007	0.009	0.017

TABLE II
Overall comparison of different signature schemes.

Signature Scheme	Computation efficiency	Partial data retrieval	Constant space cost	Sampling uniformity
Sign-each	X	✓	✓	X
Concatenate	✓	X	X	X
Hash chaining	✓	X	✓	X
DTC	✓	✓	X	✓
GSC	✓	✓	✓	✓

We qualitatively compare existing digital signature schemes and the main contributions of this paper, namely DTC and GSC, in TABLE II.

III. Problem Statement

A. Network Model

We demonstrate the life cycle of IoT sensing data in Fig. 1. Three different kinds of entities are identified as follows.

1) **IoT devices** are resource-constraint devices that generate sensing data. IoT devices are usually limited in computation, memory, and power resources. 2) **Cloud** is an ISP or a third-party cloud provider who has rich resources and expertise in operating cloud computing services. It charges clients for data storage and data access. 3) **Data applications** are software systems that may request to retrieve the sensing data for analysis purposes. Different data applications may have varied data granularity requirements. An application may fetch all or a fraction of data from the cloud of an epoch to conduct post-processing based on their requirements.

B. Data Model

IoT sensing data can be classified into two types: time series data and event data [32]. Time series data are generated by

each device for every fixed time period, such as 1 second. They are used to conduct continuous monitoring tasks such as temperature reports. Event data are generated whenever a certain type of events occurs, such as a vehicle appearing in a smart camera. They are used to monitor discrete events. **Note time series data can be viewed as a special case of event-based data driven by clocks. Moreover, event-based data is more difficult to handle. In the case of time series data, the data volume generated within one unit time is predictable and therefore the resources would be allocated with priori knowledge. For example, challenges of the sampling protocol design in Sec. V derive from the combination of the distributed setting and the unpredictability of data streams.** *Hence this paper focuses on finding a solution for event-based data. The proposed methods are also applicable to time series data.*

We assume IoT devices transmit sensing data to the cloud at a fixed time interval called *epoch*. IoT devices do not require perfect synchronization but we do assume one synchronization protocol available to loosely synchronize clocks on different IoT devices with bounded drift.

C. Threat Model

We assume only IoT devices and data applications are trustworthy. Cloud is **NOT** trustworthy, which may return incorrect query results to the data application. We consider that the adversary cloud may launch the following attacks.

1) Message forgery attack: The cloud forges data that have never been sent by the IoT device. Specially, the adversary cloud modifying the meta-data (*e.g.* data source or epoch) or content falls into this category. 2) Biased sampling attack: This attack violates the uniformity property.

The goal of this paper is to allow IoT applications to have the capability to verify the authenticity and integrity of the stored sensing data. *Note that we do not address the issue of data confidentiality and privacy in this paper.* They are orthogonal to the problem we study int this paper.

Each IoT device hosts and uses its own private key in case of device compromises. We assume that there is a well-functioning PKI which manages the distribution of the public keys. We also assume that no special hardware is leveraged to use physical-layer information to boost secure communication [33], [34].

IV. System Design

We first assume that there is no budget constraint as it is a common scenario in current IoT settings. We will relax this assumption and incorporate budget constraints in Sec. V.

A. Existing Signature Schemes

Digital signature is widely used to ensure data authenticity and integrity. However, none of existing signature schemes are appropriate for the IoT setting.

First, the straightforward Sign-each method causes expensive computational cost on both the signer and the verifier owe to excessive public-key encryption/decryption operations.

TABLE III
Important Notations.

Notation	Definition
D_i	Message digest of sample block i or event i
D_{ij}	Message digest summarizing ith till jth events
$H(\cdot)$	Message digest function
pk	Public key
pk^{-1}	Private key
$\{\cdot\}_{pk^{-1}}$	Encrypt using private key pk^{-1}
$\{\cdot\}_{pk}$	Decrypt using public key pk
m	Number of sampled events
n	Number of monitored events
K	Number of sensing devices
S_a	Numerical interval between 2^{-a-1} and 2^{-a}
$h(\cdot)$	Hashing function whose range is between 0 and 1
B	Budget limit

Since data selection is completely executed in the cloud, if the cloud selects event samples or the partial data with bias, IoT applications are unaware of it. Furthermore, the Sign-each method may not be able to detect data loss if an event record is entirely disappeared.

The *concatenate* signature scheme can amortize the signing and verification cost to multiple messages, but it is not suitable for sensing devices which may be lack of buffer space to accommodate all messages. In addition, it does not support partial sample-rate data retrieval.

Hash chaining [31] reduces the buffer space complexity from $O(m)$ to $O(1)$ for both the signer and verifier, where m is the number of messages buffered in the sensing device to be jointly signed. In the hash chaining signature scheme, only the first message is signed and each message carries the one-time signature for the succeeding message. However, hash chaining fails when some events are dropped due to sampling or partial sample-rate data retrieval.

To address the aforementioned problems, this paper presents two novel signature schemes.

B. Dynamic Tree Chaining (DTC)

We start from the Tree chaining designed by Wong and Lam [28], one variation of Merkle tree [27]. The digest of each event report is one leaf node in binary *authentication tree* presented in Fig. 2. The value of the internal node is computed as the hashing of the concatenation of its two children. Take the authentication tree in Fig. 2 as an example. D_{12} is the parent of D_1 and D_2 and $D_{12} = H(D_1||D_2)$, where $H(\cdot)$ is the message digest function, such as SHA-1 [35] or MD5 [36], used for tree chaining. Likewise, $D_{14} = H(D_{12}||D_{34})$ and $D_{18} = H(D_{14}||D_{58})$. As a result, the root summarizes all the leaf nodes. The root node is regarded as the block digest. The block digest is appended with *epochID* and then signed by the private key to create the block signature. EpochID is used to identify which epoch the data are generated; otherwise, the cloud returns events from other epochs without being detected.

The verification process is on a per-event basis. In order to verify the integrity/authenticity of an event e, the verifier requires the block signature, the position of event e in the authentication tree and the sibling nodes in the path to the root, which are all appended to event e. As a result, the overhead to

transmit this metadata is $O(\log n)$, where n denotes the number of events.

Basically, the verification algorithm is to replay the process to build the authentication tree and to verify the nodes in the path to the root. Imagine the receiver begins to verify event e_3 which is represented as the dashed circle in Fig. 2. First, the receiver computes $D'_3 = H(e_3)$ and then its ancestors in order: $D'_{34} = H(D'_3||D_4)$, $D'_{14} = H(D_{12}||D'_{34})$, $D'_{18} = H(D'_{14}||H_{48})$. Event e_3 is verified if the decrypted block signature equal D'_{18}, that is to say $\left\{\{D_{18}\}_{pk^{-1}}\right\}_{pk} = D'_{18}$, where $\{\cdot\}_{pk^{-1}}$ denotes singing using private key whereas $\{\cdot\}_{pk}$ is the function to decrypt signature with public key. In this case, all the nodes in the path, as well as their siblings, are verified and they could be cached to accelerate the verification process. Suppose the 4th event e_4 arrives after e_3 has been verified. Event e_4 is verified directly if $H(e_4) = D_4$.

Note that the expensive encryption operation is amortized to all events in one authentication tree and thus tree chaining is computationally efficient. More importantly, since every single event is verifiable in tree chaining, it is fully compatible with partial sample-rate data retrieval without resource waste. The most severe issue that impedes the adoption of the original tree chaining in IoT environment is that all events should be buffered in the IoT device before the building of authentication tree, since each event ought to be appended with auxiliary authentication information from the authentication tree.

Introducing cloud can greatly reduce the memory footprint at IoT devices. The IoT device only maintains the message digest of each event and stores all events to the cloud directly without caching. At the end of each epoch, with all leaf nodes available, the IoT device builds the authentication tree, which is then sent to the cloud. The cloud in turn attaches essential authentication information to each event received in the current epoch. The memory footprint can be further optimized if the authentication tree grows in an online fashion: The IoT device transmits to the cloud internal nodes no longer needed for calculating the rest of authentication tree. An internal node is generated when its two children are available. In the meantime, these two children are transmitted to the cloud. We reuse Fig. 2 to illustrate the online authentication tree building process. $D_1 - D_8$ represent the message digests of the events in timely order. D_{12} is calculated immediately when D_2 comes into play. In the meantime, D_1 and D_2 cached in the sensing device are transmitted to the cloud. Likewise, when D_4 is available, D_{34} is computed, which in turn immediately contributes to the calculation of D_{14}. As a result, at that time D_3, D_4, D_{12} and D_{34} are dismissed from the sensing device. It is not hard to imply that this optimization reduces the space complexity in the sensing device to host nodes of authentication tree from $O(n)$ to $O(\log n)$, where n denotes the number of events monitored in one epoch.

For DTC, data selection is completely executed in the cloud when the data application retrieves partial sample-rate data. Without additional mechanisms, if the cloud selects event samples or the partial data with bias, IoT applications are unaware of it. The plausible solution is to allow the data application to specify the sequences of the interesting events. It is the application's own responsibility to guarantee uniformity. Apparently, this straightforward method is not scalable. To this end, we propose to optimize this solution by expressing the sequence of requested events in a succinct way. A sequence number is appended to each event to indicate its position in the authentication tree. The data application sends a number m which specifies the number of events requested as well as a seed s which determines one random permutation. The cloud returns the events specified by the top m elements in the random permutation. The data application checks whether these events sequence numbers are consistent with the random permutation. In this proposal, the uniformity is guaranteed by the random permutation. In appendix **??**, we demonstrate the pseudocode of random permutation algorithm in Algorithm **??**, which is an implementation of Fisher-Yates shuffle [37]. The random permutation algorithm ensures that the m randomly sampled events conform to uniformity. The detailed analysis is provided in Sec. VI.

Furthermore, using different seeds enables *re-sampling*, which means that the receiver can request for different sets of uniformly drawn events from the cloud. **This feature is a useful tool for various purposes, including estimating the bias and standard error of a statistic [38], cross-validation [39] and ensemble learning [40], *etc.* For example, each weak learner of ensemble learning requires one instance of training dataset generated by one round of re-sampling from the whole dataset. More references on the usage of re-sampling could be found in [41].**

Nevertheless, the number of generated events is unpredictable and may be unbounded. Once the buffer in the sensing device is full, the root node in the authentication tree is signed and the remaining nodes are flushed to the cloud to spare space for upcoming events. In this case, one IoT device may apply digital signature more than once in one single epoch. The verifier also requires additional space to cache the verified nodes. The verifier stops caching new verified nodes when the buffer is full. As a result, the buffer space constrains the performance of DTC, which is a particularly severe problem in IoT environment where most devices possess little buffer space. DTC can be also extended to k-degree Merkle tree. We will have performance analysis in Sec. VII and show that the binary DTC presented in this section is the most space efficient.

Even though DTC is a variant of Tree chaining designed by Wong and Lam [28] (which itself is derived from Merkle tree [27]), there are several differences and we summarize as follows. 1) DTC and Tree chaining target different scenarios. Tree chaining is used for signing broadcast messages and therefore uniformity is not supported in the Tree chaining original paper. **2) The most significant feature of DTC is its low space cost and the accompany improvement in performance. The evaluation results reveal in Fig. 9 indicate the that DTC outperforms tree chaining when the memory space on the IoT device is constraint.**

C. Geometric Star Chaining (GSC)

We propose a more efficient and secure data communication in this paper, called Geometric Star Chaining (GSC). GSC

still support partial sample-rate data retrieval. Even though GSC does not enable the data consumer to verify every single message alone as DTC can, most IoT applications do not have such requirement.

The basic idea of GSC is inspired by one observation that any arbitrary fraction value can be represented or closely approximated by a few number of binary digits. For instance, $5/8 = (0.101)_2$. Thus, partial data with sample rate p, where $p = \sum 2^{-b_i}$ and b_i is the position of the i-th 1 in the binary expression of p, is equivalent to the union of multiple data blocks each corresponds to one set bit in the binary representation. The data block is called *sample blocks* in this paper. For instance, to retrieve a sampled data with sampling rate $5/8$, the cloud can send the data application two blocks containing (approximately) $1/2$ and $1/8$ of the samples respectively.

The events included in the sample blocks are in *geometric distribution*. Each sample block should draw events uniformly from the IoT data stream. In order to ease the presentation of how sample blocks form, we define a set of successive numerical intervals $\{S_i\}$ where $S_i \triangleq \{x \in \mathbb{R} : 2^{-i-1} < x \leq 2^{-i}, i \in \mathbb{N}\}$, which are visually represented as rectangles in Fig. 3. On receiving a new event e, the sensing device computes which numeric interval in $\{S_i\}$ that $h(e)$ falls in and event e is inserted into the corresponding sample block, where $h(\cdot)$ is a non-cryptographic uniform random hashing function and $\forall x : 0 \leq h(x) \leq 1$.

Note that events in the same data block are either completely retrieved or not retrieved at all. Thus we can view each of such data block as an atomic "giant event". GSC computes one message digest for every block and concatenates these digests to a single digest for digital signature, as is depicted in Fig. 4. The digest of one sample block is computed in an online fashion. One variable D_i is allocated to each sample block to capture the newest value of message digest. Suppose a new event e observed at the device which belongs to the ith sample block. The message digest updates as $D_i = h(h(e)||D_i)$, which is also referred as Merkle-Damgård Construction [42]. This online updating proceeds until the end of the epoch. At this time, concatenate approach is applied to all the message digests $\{D_i\}$. The result summarizes all events generated in one epoch. Note the value i, which indicates the sampling rate of each block, should also be stored and hashed with the block. In this way, the application that receives the block can verify the sampling rate.

In fact, any random function can be used to implement the geometric distribution for GSC, such as continuous coin-tossing, but using a uniform random hash is convenient. One practical issue about hashing is that the raw output of hashing functions is one finite-length bit sequence. Computing which numerical interval in $\{S_i\}$ that $h(e)$ falls in is equivalent to counting leading zeros (CLZ) in that bit sequence, which is intrinsically supported in many hardware platforms including X86 and ARM. Therefore, $|\{S_i\}|$ and hence $|\{D_i\}|$ are bounded by the size of the bit sequence. For the case of xxHash64 [43], this function produces 64-bit hash values and thus $|\{S_i\}_{0 \leq i \leq 64}| = 65$ and $|\{D_i\}_{0 \leq i \leq 64}| = 65$. It is evident that space cost for this signature scheme at the sensing device is constant.

D. Data Retrieval and Verification of GSC

A sampled fraction of sensing data is usually sufficient for most IoT applications [44]. In the network model presented in Sec. III-A, an application requests for a certain fraction of events observed at a particular sensing device from the cloud. GSC provides verifiable authenticity, integrity, and uniformity for partial data retrieval with an arbitrary sampling rate.

Based on the application requirement, a data application first determines the maximum number of events of each sensing device for an epoch it wants to receive, called a portion number. It then sends all portion numbers to the cloud. For each portion number, the cloud converts it to a sampling rate p and constructs the binary expression of p, such that $p = \sum 2^{-b_i}$ where b_i is the position of the i-th 1 in the binary expression of p. Then the cloud sends the corresponding sample blocks to the application.

For the received sample blocks, the application first computes their digests as the final digest used for the signature. It then compares the final digest and the decrypted signature. This step verifies the following properties. 1) The received blocks were not modified or partially dropped and 2) The data were indeed uniformly sampled based on the given sampling rates and the uniform random hash function.

Compared to DTC, GSC requires smaller buffer size on each sensing device. It also provides verifiable uniformity. **Moreover, GSC is more performant than DTC as demonstrated by Fig. 14 which shows the throughput comparison between GSC and DTC. The improvement in performance is not attributed to the underlying star chaining structure, since K-degree DTC, of which star chaining could be viewed as an extreme case, is even slower than normal 2-degree DTC in terms of tree building and packet generation, as indicated by TABLE VI and Fig. 11.** GSC does not support random permutation as in DTC because GSC by design ensures verifiable uniformity which is the motivation for random permutation; otherwise, random permutation would offset most of the performance gain of GSC.

V. Incorporating Budget Limit

With ever-growing volume of IoT data, storing all raw IoT data in the cloud poses a heavy monetary burden on the users. In the previous section, we have discussed the system design without restricted budget limits. We relax this assumption and incorporate the budget limit in this section. The solution presented in this section to address the issue of budget limit is compatible with DTC and GSC.

A. Sampling Protocol Design

In this section, we describe a distributed sampling protocol taking the budget limit into consideration. This sampling protocol introduces a new entity, called *coordinator*, in the network model. One coordinator is a software working as a sampler which sits between the sensing devices and the cloud. A coordinator can be installed on an IoT hub or a server at the edge of the Internet. It maintains communications with all

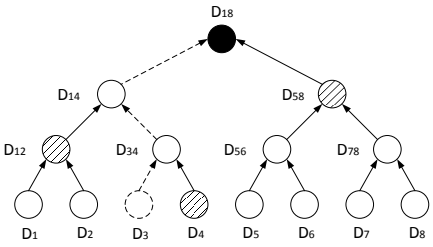


Fig. 2. Illustration of tree chaining. Verifying D_3 requires sibling nodes in the path to the root (D_4, D_{12}, D_{58}), signature of the root ($\{D_{18}\}_{pk}$) and the position of e in the tree (3).

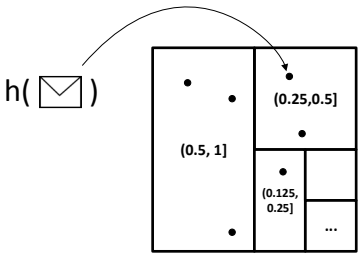


Fig. 3. Visual representation of numerical intervals.

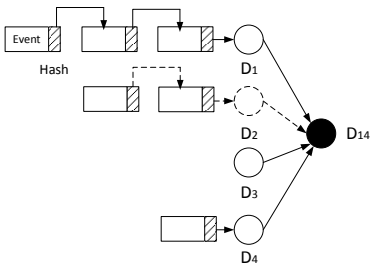


Fig. 4. Illustration of GSC. Verifying the second sample block requires the events in it, D_1, D_3 and D_4.

sensing devices on behalf of the cloud and temporarily buffers IoT data samples.

We focus on one single epoch in the discussion since at the beginning of each epoch, the *sampling protocol* (SP) is reset to the initial state. At the end of each epoch, the coordinator signals all sensing devices to advance to the next epoch. The straightforward solution is to buffer all the events in the coordinator and uniformly sample them based on the budget limit. However, the number of these events could possibly be very large, and therefore the storage capacity of the coordinator may be not enough to accommodate them all. Thus, a sampling protocol with space bound for both the sensing device and the coordinator is desired. The challenge of such sampling protocol design derives from the combination of the distributed setting and the unpredictability of data streams. If only one stream of data is considered, the problem is regressed to classic *reservoir sampling* [45], which has been studied extensively in the literature. Also, as long as the number of elements in each stream of data is known in advance, the central coordinator can decide how many samples are allocated to different sensing devices, each of which runs an instance of reservoir sampling.

To this end, we utilize the recent study in distributed streams [22] and design an efficient sampling protocol based on it. The basic idea of this sampling protocol is to dynamically maintain events with the smallest hash values on the coordinator, also known as bottom-k sampling [46]. Suppose B is the sampling budget per epoch. For the simplest implementation, all IoT devices upload generated events to the coordinator directly. The coordinator only maintains the B events with smallest hashing value and discards others in an online fashion. As long as the hashing is uniform, the events maintained in the coordinator are drawn uniformly from all events already observed from the epoch.

In order to reduce network bandwidth consumption, the coordinator could broadcast to all sensing devices current global B-th smallest hash value, denoted as τ, so that the IoT devices could discard the events locally whose hash value is greater than τ. Let σ denotes the total number of events sent to the coordinator and K is the number of sensing devices. One straw-man sampling protocol is that the coordinator broadcasts the new value of τ every time it changes. Since τ changes $O(B\log\sigma)$ times, the communication cost between the coordinator and the sensing devices is $O(KB\log\sigma)$. Cormode et al. proposed a distributed sampling algorithm [22], which is proved to be optimal in terms of communication cost, which is $O(K\log_{K/B}\sigma + B\log\sigma)$ with high probability. We tailor it to fit our proposed signatures, DTC and GSC. Basically, the coordinator accepts any event from all IoT devices until the budget limit is exceeded. The coordinator discards half of the received events (*i.e.* the coordinator halves the sample rate) to accommodate new events. The coordinator repeats this process until the end of the epoch and then uploads the stored events to the cloud. The detailed distributed sampling protocol is presented as follows.

The sampling protocol executes in multiple rounds. The coordinator as well as the sensing devices maintain a variable which represents which round the sampling protocol is in, and the coordinator ensures that all devices are kept up to date with this information. Initially, the sampling protocol begins at round 0. Suppose the sampling protocol is at round j. As we will see, round j indicates a sample rate of 2^{-j}. This protocol involves two algorithms at the sensing device and the coordinator respectively. The pseudocode for the sensing device and the coordinator is presented in Algorithm 1 and Algorithm 2 respectively.

Sensing device: On receiving a new event e, the sensing device first computes which numeric interval in $\{S_i\}$ that $h(e)$ falls in, and updates the *local counter* associated with this set, where $h(\cdot)$ is a uniform random hashing function and $\forall x: 0 \le h(x) \le 1$. Computing the numeric interval can still be visually interpreted by Fig. 3 where the result presented the $i-th$ largest rectangle. Let l_i^k be the local counter for S_i at device k. Each sensing device and the coordinator maintain their own local counters. The local counters at devices are used for auditing the coordinator. The detail will be discussed in Sec. VI and the sampling protocol still works correctly without these counters. It is worth mentioning that all sensing devices and the coordinator use the same hashing function. Suppose $h(e) \in S_i$. If $i \ge j$, which implies $h(e) \le 2^{-j}$ (sample rate), the device instantly forwards event e to the coordinator; otherwise, the event is discarded locally. At the end of each epoch, the sensing device signs both sampled events and all counters it maintains. Note that none events are buffered at the device in any case.

Coordinator: The coordinator maintains queues $\{Q_i^k\}$, each

Algorithm 1: SP at sensing device k in round j

```
foreach event e do
    i ← min{x ∈ ℕ : h(e) ≥ 2^{-x-1}};
    l_i^k ← l_i^k + 1;
    if i ≥ j then
        Forward e to the coordinator;
    else
        Discard e;
    end
end
```

Algorithm 2: SP at the coordinator in round j

```
foreach event e do
    i ← min{x ∈ ℕ : h(e) ≥ 2^{-x-1}};
    k ← e.source;
    if i ≥ j then
        Q_i^k.add(e);
        l'_i ← l'_i + 1;
        g ← g + 1;
        while g > B do
            Discard queues {∀k̂, Q_j^k̂};
            g ← g − l'_j;
            j ← j + 1;
            Broadcast j to all sensing devices;
        end
    else
        Discard e;
    end
end
```

of which corresponds to one numerical interval in $\{S_i\}$ of each sensing device. Upon receiving an event e, the coordinator first computes i, such that $h(e) \in S_i$, followed by comparing the value of i and j. In the case of $i < j$, event e is discarded; otherwise, it is buffered at queue Q_i^k (suppose the event is from kth sensing device) followed by updating both the counter associated with numerical interval S_i and the global counter g, which records the total number of events buffered at the coordinator. At this moment, as long as the value of the global counter g exceeds the budget limit B, all event queues associated with S_i are discarded, the global counter is updated accordingly and the sampling protocol advances to the next round (i.e. $j \leftarrow j+1$). The coordinator then signals all sensing devices to promote to the newest round j. It is evident that coordinator buffers at most $B+1$ events all the time. Hash chaining cannot coexist with the sampling protocol, because the coordinator is allowed to discard events that are essential for the verifier to validate the received data. DTC and GSC, on the other hand, do not bear the same problem. Algorithm 2 is the pseudo-code for the coordinator part of this sampling protocol.

B. Data Retrieval

The sampling protocol is compatible with DTC and GSC. It is natural for this budget-based sampling mechanism to be compatible with GSC since the sampling algorithm discarding the events half at each round which is essentially removing the existing largest GSC sampling block. As a result, the remaining buffered sample blocks correspond to successive numerical intervals. The data application can still fetch any fraction of data that is stored in the cloud. DTC requires a minor modification to support verifiable uniformity when the sampling protocol is performed. Recall how DTC leverages random permutation to guarantee uniformity in Sec. IV-B. The sampling algorithm may discard the events specified by the random permutation. We argue that the receiver can still draw events uniformly from the cloud if the receiver can check the existence of every event. Suppose the receiver would like to fetch n events from the cloud. Instead of sending to the receiver the first n events specified by the random permutation, the cloud should reply with the first n **existing events** sorted by the random permutation. We enable the receiver to check the existence of an event locally by proposing that the hash function $h(\cdot)$ in the sampling algorithm is based on the event sequence number and epochID, but not the content. If the hash value falls outside the discarded numerical intervals, the receiver instantly knows the existence of the event.

VI. Security Analysis

We use digital signatures to verify data integrity and authenticity. Any inconsistency in the verification procedure indicates data in the cloud untrusted. In the sampling protocol, each sensing device maintains a counter to record the number of events that fall in a certain sample block.

A. Defending against Message Forgery Attacks

Both DTC and GSC follow classic Hash-and-Sign Signature paradigm to provide the desirable property of existential unforgeability [47] to defend against message forgery attack. Hash-and-Sign Signature paradigm requires the hash function to be collision resistant. **For a collision-resistant hashing function, if the output length of the hash function is l bits, the probability for the adversary to forge a signature by finding the collision of a hash value is 2^{-l}. In the context of this paper, performing the hash operation means to compute the root of the tree chain (star chain) for DTC (GSC). DTC does not modify the hash computation of the underlying tree chaining. GSC leverages Merkle-Damgård Construction (Hash Chaining) [42], which has been proved to be collision resistant, to compute each node of start chain.**

We prove in Appendix **??** that Tree Chaining (used in DTC) and Star Chaining (used in GSC) are collision resistant if the underlying hash function is collision resistant. Therefore, both DTC and GSC are resistant to message forgery attacks.

B. Defending against Biased Sampling Attacks

For DTC, a random permutation is leveraged to sample data from the cloud. We will prove that the property of uniformity is

preserved by the random permutation. Random permutation is an implemented of Fisher-Yates shuffle [37]. In the paper [37], it has been proved that after random permutation any element can be placed at any position with equal probability. Since our implementation outputs the first m elements of random permutation, the uniformity is preserved.

For GSC, which sample block events belongs to is uniquely determined by a non-cryptographic uniform random hashing function. As a result, any sample block contains uniformly drawn events.

C. Defending against Dishonest Coordinators

The sampling protocol also defends against dishonest coordinators which do not execute the protocol in a correct way. For example, a dishonest coordinator may stop monitoring sensing devices by intentionally setting a negligible sample rate. The most difficult part is to check the final round that sampling protocol terminates at. Appendix **??** proves how the local counters from IoT devices are used for audit correctness of the sampling protocol.

VII. Performance Analysis

In Sec. IV-B, we present the Merkle tree implemented by one binary tree. Merkle tree can be constructed in the form of k-degree tree as well. For the extreme case, Merkle tree authentication is degraded to star chaining when k exceeds the number of events. In this section, we first analyze more generic k-degree DTC in terms of time and space complexities. After that, we comprehensively compare k-degree DTC with GSC. Finally, the performance analysis, especially space cost is presented in Sec. VII-C.

A. K-degree Dynamic Tree Chaining

Suppose the IoT device detects n events in one epoch. Therefore, the height of the k-degree Merkle tree is $O(\log_k n)$. The sibling nodes in the path to the root are needed to compute the root hash value for signature verification. As a result, $O(k\log_k n)$ hash values are attached to every event. For the same reason, the signer maintains at least $O(k\log_k n)$ nodes to dynamically update the authentication tree.

The time to sign a set of events consists of three parts: *authentication tree building time*, *root signing time* and *packet generation time*. The time to build the authentication tree is proportional to the number of nodes in the tree. In the k-degree Merkle tree summarizing n events, there are $\frac{1}{k}n+\frac{1}{k^2}n+\ldots = O(\frac{1}{k-1}n)$ internal nodes in total. The value of each internal node is computed by taking the hashing of the concatenation of all its children. Suppose the time complexity of the hashing function is $O(l)$, where l is the length of the input string. (This is the case for most hash functions such as MD5 [36] and SHA-1 [35].) In this case, computing one internal node takes $O(k*O(1))=O(k)$ time units. With $O(\frac{1}{k-1}n)$ internal nodes, the time complexity to compute all internal node values is $O(n)$. For the leaf node, its value equals the hashing of raw data. If we denote l_i as the data length of event e_i, the time complexity of computing leaf nodes is $O(\sum_{i=1}^{n} l_i)$. The overall authentication construction time is thus $O(n+\sum_{i=1}^{n} l_i)$.

Signing the root is one public-key encryption operation, which is the same for any degree of Merkle tree.

Packet generation is to append necessary verification information to each event. In DTC, we offload packet generation to the cloud. Following what discussed above, $O(k\log_k n)$ time units are required to append all hash values to the event for verification.

If the receiver is granted enough space to cache all the internal nodes, the time complexity to build the authentication tree in the receiver side is identical to that in the sender side. Accordingly, one public-key decryption is applied to the signature of the authentication tree root.

B. Geometric Star Chaining

Following the notation in the last subsection, we still present the number of events detected in one epoch as n. Recall that hash value of sample block is updated as $D=h(h(e_i)||D)$ upon a new event e_i. The time complexity to update the sample block is thus $O(l+O(1))$. In GSC, an individual event does not go through packet generation phase. Instead, one whole sample block is only associated with one piece of verification information which significantly reduces the packet signing time consumption. In conclusion, the time to sign n events in one epoch for GSC is $O(n+\sum_{i=1}^{n} l_i)$ plus one public-key encryption.

The main advantage of GSC over DTC is its constant space complexity. The memory footprint is only bounded by the number of sample blocks.

C. Sampling Protocol

Both the sensing device and the coordinator are sensitive to space consumption. Since the space consumption for the events themselves is the same, we concentrate on the space used for different signature schemes. We go into details on the space complexity of different signature schemes, which are compatible with the sampling protocol, at the sensing device and the coordinator respectively, as shown in TABLE. IV, where n is the number of events monitored at one device and n' denotes the maximum value of n among all sensing devices. The space complexity of DTC at the coordinator is $\mathrm{O}(B\log n')$ because there are $\mathrm{O}(B)$ events buffered at the coordinator and in the worse case each event is appended with $\mathrm{O}(\log n')$ hash values for verification. Following the same line of reasoning in Sec. IV-B, the lack of buffer space in the coordinator may significantly degrade the performance of DTC. GSC requires $O(K)$ space at the coordinator because the coordinator maintains the sample block digests for all K devices. Note that space complexity discussed above is for the verification metadata and the space complexity of the raw data is always $\mathrm{O}(B)$ for all signature schemes.

The communication cost of the sampling protocol is the same as the protocol proposed by Cormode et al. [22], which is proved to be optimal in terms of communication cost, which is $O(K\log_{K/B}\sigma+B\log\sigma)$ with high probability.

TABLE IV
SPACE COMPLEXITY OF DIFFERENT SIGNATURE SCHEMES.

Signature Scheme	Device	Coordinator
Sign-each	O(1)	O(B)
DTC	O($\log n$)	O($B \log n'$)
GSC	O(1)	O(K)

VIII. EVALUATION

We conduct extensive trace-driven simulation and prototype experiments using real dataset. We use two encryption algorithms, RSA-1024 [48] and DSA-512 [49]. MD5 [36], SHA-1 or SHA-256 [35] are leveraged as the message digest function. We implement DTC and GSC as well as another three alternative signature schemes for comparison.

A. Experiment Setup and Methodology

Dataset. The dataset [50] includes a wide variety of 90-day sensing data collected from sensing devices at three homes. We select 7 sources event data from the dataset to represent the event reports generated at sensing devices: environmental information (including temperature and humidity, etc.) about homeA, homeB and homeC respectively; electrical data from dimmable and non-dimmable switches for homeA; two sets of operational data on door and furnace on/off for homeA; the data from the motion detector located at homeA. Each record is encoded into 4 bytes: 2 bytes for timestamp and 2 bytes for sensor reading.

Hardware configuration. The prototype emulation experiments are conducted on a quadcore@3.40GHz Linux desktop with 32GB memory and a Raspberry Pi 3 Model B with a quad-core 64-bit ARM Cortex A53@1.2GHz. For all prototype experiments, only one core is used.

Methodology. The simulation experiment takes the budget limit into consideration. Conducting the simulation experiment serves two purposes: 1) Investigate the sampling protocol. 2) More importantly, the data trace of the simulation experiment will be fed the prototype experiment. For example, the simulation experiment computes which events should be sent to the coordinator and which events should be discarded locally. Only the events sent to the coordinator are fed to the prototype to evaluate signature generation through/speed at full speed. The prototype experiments are conducted on both the Linux desktop and the Raspberry Pi board to represent IoT device. Unless otherwise explicitly expressed, the default hardware platform to conduct prototype experiment is the Linux desktop. We do not test the throughput/speed in real distributed settings, because the data trace (*e.g.* sparse event data) may not be able to stimulate the IoT device to run at full speed. The prototype experiment is not conducted on one M3 board (We test the cryptographic operation performances on this IoT platform) because we cannot replay trace on it. We simulate the sampling protocol driven by the 7 sets of data listed above. We vary the budget limit and evaluate its impact on the simulation results. We implement the signature scheme prototype for performance comparison against alternative solutions. The prototype experiment first tests the signing and verifying performance without sampling protocol involved under varied parameters. We next conduct prototype experiment in a setting where sampling protocol is involved. The second prototype experiment focuses more on the potential maximal throughput of tested signature schemes, since other impacting factors are explored in the prior prototype experiment.

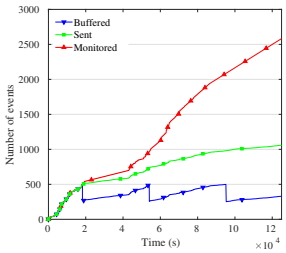


Fig. 5. One-day micro-scale exp.

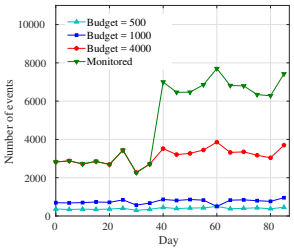
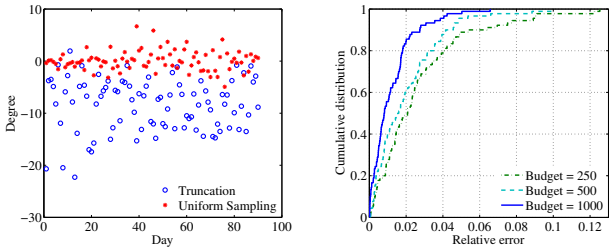


Fig. 6. #events saved in the cloud.

B. Simulation Result

We first conduct one micro-scale experiment to illustrate how the sampling protocol proceeds when new events arrive, as depicted in Fig. 5. We fix the budget limit to 500 events in this micro-scale experiment. The three lines in Fig. 5 represent the number of events buffered at the coordinator, sent to the coordinator by all the 7 sensing devices and monitored at all sensing devices, respectively. The three lines vary against time in one day (May 1st, 2012). Initially, the number of events is the same for the three lines until the number of buffered events at the coordinator reaches the budget limit. At this time, approximately half buffered events are discarded, illustrated as the first vertical drop in Fig. 5. Events at the coordinator then are accumulated over time until the next sharp decrease. This process repeats down to the end of this epoch. The space used at the coordinator never exceeds the budget limit. It is worth mentioning that the total number of events sent to the coordinator grows slower with the time, which is a desirable property since the communication cost stays low even if much more events are monitored. This simulation experiment, to some extent, validates the theoretical analysis on the communication cost in which the communication cost only grow logarithmically. From Fig. 5, totally 1057 events were sent to the coordinator on May 1st, 2012. On the other hand, there were totally 2572 events monitored on that day.

Next, we investigate how different values of budget limit impact the number of events eventually saved to the cloud. We present the number of events saved at the cloud each day from May 1st, 2012 to July 31st, 2012 with different values of budget limit in Fig. 6. Fig. 6 shows that this sampling protocol utilizes approximately 75% of the budget on average for different budget values. In Fig. 6, we also demonstrate that this sampling protocol works correctly in the presence of drastic changes, as the number of events monitored soars at the 40th day. In this case, the sampling protocol does not violate the budget constraints. On condition that the total number of monitored events is smaller than the budget limit, the sampling protocol saves all of them in the cloud, as is exemplified by the case where $budget = 4000$ in Fig. 6.



(a) Deviation from the ground truth (b) Impact of budget limit

Fig. 7. Computing average temperature from data saved in the cloud.

The underlying foundation of our sampling protocol is that uniformly sampling is ensured. We will see the importance of uniformity in one real application. The temperature sensor periodically measures the environmental temperature and sends the sensing data to the cloud for archiving purpose. We calculate the average temperature outside homeA each day based on the sampled data saved at the cloud. The ground truth is the mean of all temperate sensing data from the temperature sensor. In order to illustrate the need for uniformity, we calculate the average temperature by the first 40 truncated sensing data (which is greater than the number of saved data in the cloud under most circumstances in this simulation experiment). Fig. 7(a) demonstrates how estimated average temperature deviates from the true one with respect to using our proposed sampling protocol and using naive truncation, when the budget limit is fixed to 500. It is obvious that the average temperatures calculated by uniformly sampled data are much more useful in reflecting the real data. In this example, the truncated data are measured in the morning. Thus, the average temperatures calculated by truncated data are smaller than real average temperatures in nearly all days (only one day is an exception). We then evaluate how the value of budget limit affects the accuracy of estimated average temperature. As expected, a greater value of budget limit yields more accurate results, as illustrated in Fig. 7(b).

C. Prototype Emulation Experiment Without Budget Limit

We conduct extensive prototype emulation experiments in this subsection. The efficiency of the signature scheme used greatly impacts the adoption of sensing devices, since most sensing devices are resource-constraint. As an indirect measurement of power consumption, we evaluate the speed of signing under different parameter settings. We also evaluate the performance at the verifying phase. The data applications may fetch data from hundreds or thousands of devices, the verifying speed is also critical for a scalable application.

The 7 data sources each divided into 90 epochs are the input to the singing phase of the signature scheme, whose output feeds the verifying phase afterwards. No budget constraints are involved and therefore all signed events are stored in the cloud. The parameter space consists of the space available at the signer/verifier as well as the sampling rate of the data application. The space cost at both the signer and the verifier to host the events themselves is orthogonal to the choice of signature scheme. Thus, the space cost in this subsection is in particular referred to the memory footprint of the signature scheme. It can be implied from the algorithm descriptions in Sec. IV that the memory usage for all signature schemes mentioned in this paper is a multiple of the length of the message digest function. In order to simplify the presentation, we refer one unit of space cost as the memory space used for storing one message digest. DSA is applied to the public encryption/decryption and MD5 is utilized as the message digest function in this subsection.

TABLE V
QUANTITATIVE COMPARISON OF DIFFERENT SIGNATURE SCHEMES.

Signature Scheme	Encryption time	Sender comm. cost	Receiver comm. cost
Sign-each	2.82 ms	132 bytes	132 bytes
Concatenate	0.59 ms	4.43 bytes	8.89 bytes
Hash chaining	0.57 ms	4.43 bytes	8.89 bytes
DTC	0.55 ms	4.45 bytes	4.52 bytes
GSC	0.41 ms	4.43 bytes	4.43 bytes

First of all, we quantitatively compare the performance of the signature schemes in TABLE II. In this set of experiments, no limits are placed onto the signer/receiver space usage and we set the application's sampling rate to be 50%. The encryption algorithm is RSA-1024 and the hashing function to compute digest is MD5. We measure the amortized cost for encryption time, sender and receiver communication cost. Especially, the amortized receiver communication cost is computed as the $\frac{\#received\ bytes}{\#records\ of\ interest}$. The results are listed in TABLE V. As expected, the amortized time to sign one record much larger for Sign-each method. Since the signature length of RSA-1024 is 128 bytes, the amortized sender/receiver communication cost is $128+4=132$ bytes. For all other signature schemes, their amortized encryption time is similar. For Concatenate and Hash chaining, the receiver's communication cost is approximately twice as large as the sender's because only half of the received records are of interest.

We measure the performance of two signature schemes with given space in the signer. To focus on the impact of the space issues at the signer side, we allocate enough free space to the verification process and the receiver takes all data stored in the cloud. If DTC is used, once the buffer in the signer is full, the root node in the authentication tree is signed and the remaining nodes are flushed to the cloud to spare space for upcoming events. Thus, lacking space in the signer may lead to multiple expensive encryption operations in one epoch. Furthermore, the same number of decryption operations are also needed at the verifier side. On the other hand, the available space affects the resolution of the sample blocks, rather than signing speed. Only one encryption operation is performed in a single epoch. Fig. 8 illustrates the signing/verifying performance comparison between GSC and DTC under varied space available at the signer. It is obvious that both signing and verifying performance of DTC are capped by available memory at the signer, whereas GSC runs at full speed all the time.

The sampling rate at the receiver side also affects the verifying performance for both GSC and DTC, because it directly determines the number of events to share the cost of

encryption/decryption, as depicted in Fig. 10, where higher sampling rate yields better verifying throughput. Another observation from Fig. 10 is that sampling rate also impacts GSC in terms of the space needed in the signer to achieve the maximal verifying throughput. Recall that the available space in the signer defines the resolution of sample blocks. The unused but verified events decrease as the resolution of the sample block improves. Suppose the receiver asks for 10% data in the cloud. In the case where there are 2 units of space in the signer, the receiver fetches and verifies 30% unused data because the finest sample block contains 50% data. If the available space increases to 3 units, the smallest sample block consists 25% data and thus the unused data shrinks to 15%. The time wasted for unused data becomes increasingly prominent when the sampling rate decreases. Therefore, smaller the sampling rate, more space required at the signer. The good news is that as small as 7 units of space are enough to support maximal verifying throughput when the sampling rate is 1%.

Moreover, the verifying throughput varies with the space allocated to cache verified nodes in the authentication tree for DSC. In our current prototype implementation, the verifier stops caching new verified nodes if the buffer is full. As expected, the performance acceleration is more evident with more cached verified nodes, as illustrated in Fig. 9. It is interesting to note that before any of the three lines in Fig. 9 reaches full speed, for a given fixed space at the verifier, the verifying throughput is higher when there is less space available in the signer. This is because the locality of the cache nodes favors higher refreshing frequency. When smaller space is available at the signer, the number of jointly signed events is less and thus the cached nodes refresh quicker.

We also measure the time to build the Merkle tree and to generate packets for K-degree DTC. In this set of experiments, we synthesize different sizes of data traces to feed the machine running K-degree DTC. MD5 is used as the cryptographic hash function. For each trace, we conduct the experiment for 10 times and we depict the medians of per-packet amortized time consumption for Merkle tree building and packet generation in Fig. 11. In terms of per-packet time to build the Merkle tree of a certain degree, it remains stable with the varying number of events, which conforms the time complexity analysis in Sec. VII-A. In Fig. 11, the lines lightly decline because more events share the initial setup overhead. Another interesting finding is that it is faster to build the K-degree Merkle tree when $K = 4$ than when $K = 2$ and $K = 3$, even though they all share one asymptotic time complexity. The performance gap is mainly due to the asymptotic time complexity of hash functions cannot describe the time consumption in K-degree Merkle tree building accurately. In Sec. VII-A, we assume the time complexity of the cryptographic hashing function is $O(l)$, where l is the length of the input string. However, the actual time consumption is proportional to the number blocks the input data are chopped into. For MD5 that we use as the hash function, the block size is 512 bits which can host 4 digests. As a result, computing an internal node of a 4-degree Merkle tree is **not** more expensive than computing a 2-degree Merkle tree internal node in terms of performing the hash function. Building a 4-degree Merkle tree requires less hashing operations compared to the other two, hence smaller per-packet amortized time to build the Merkle tree. The per-packet generation time grows with more events as illustrated in Fig. 11. This is expected from the theoretical analysis which indicates that the time complexity is $O(\log_k n)$, where n is the number of events signed under a same Merkle tree. From Fig. 11, 4-degree Merkle tree is more efficient in generating the packet due to the fact that appending multiple hashes from the same level in the Merkle tree in batch is efficient. We evaluate the per-packet generation time with fixed height $h = 3$ to validate its asymptotic time complexity. As shown in TABLE VI, the packet generation overhead is more prominent with larger tree degree.

TABLE VI
PER-PACKET GENERATION TIME WITH FIXED HEIGHT $h = 3$

Degree (K)	500	600	700	800	900	1000
DTC (μs)	4.219	5.035	5.933	7.078	7.925	8.800

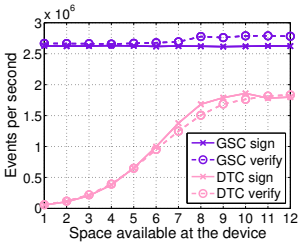


Fig. 8. Throughput comparison

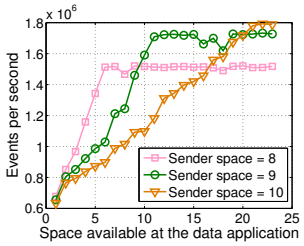


Fig. 9. DTC receiving throughput

D. Prototype Experiment in Raspberry Pi

We conduct experiments on a Raspberry Pi 3 Model B board, which is one of the most popular IoT device platforms. We compare the power and time consumption of the two signature schemes we propose in this paper, namely GSC and DTC. The two metrics are especially important for resource-constrained IoT devices. The experiment setup is exactly the same as the PC experiment which has been described in Sec. VIII-C, except that the program is running in one Raspberry Pi instead of a more powerful PC. We utilize an inline power meter to measure the power consumption overhead to finish the signing process. The voltage remains at 5.1V all the time but the current jumps from 0.23A in the idle state to 0.37A

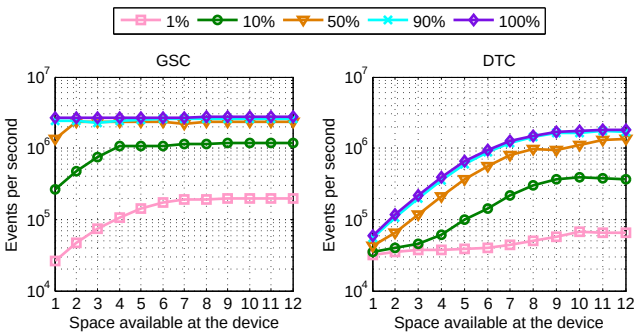
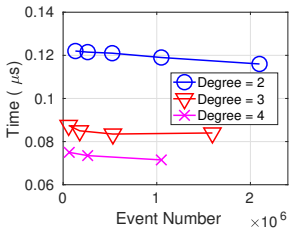
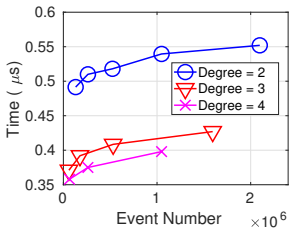


Fig. 10. Verifying throughput comparison with different sample rate.



(a) Amortized building time



(b) Amortized pkt. gen. time

Fig. 11. K-degree DTC Merkle tree exp.

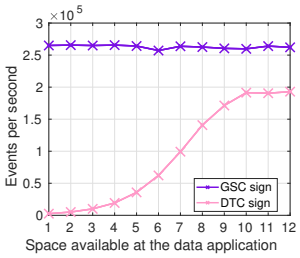


Fig. 12. Signing thrpt. at RPi

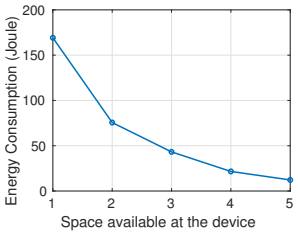


Fig. 13. DTC energy comsumption

when the signing program starts. In our experiment, the energy consumption due to the signing program is calculate as $E_{sign} = E_{total} - P_{idle} \times time$, where we can read E_{total} and $time$ from the power meter. P_{idle} is computed as $5.1V \times 0.23A = 1.18W$. As we have shown in Sec. VIII-C, the signing speed for DTC is limited by the available memory space. The situation is also true on Raspberry Pi where the signing speeds for both GSC and DTC are about 10 times slower than those in the PC experiment. We illustrate the signing speed in Fig. 12. When the program is running, the voltage and current remain stable, hence the power. Therefore, the power consumption is proportional to the program running time. For DTC, the limited memory space elongates the programming running time, leading to higher power consumption, as depicted in Fig. 13.

E. Prototype Experiment with Sampling Protocol

From the prototype experiment without budget limit, it seems that the space requirement, $\log n$ units, at the signer is trivial, where n is the number of event reports generated in one sensing device. If the sampling protocol is utilized, the signing/verifying performance is likely to be limited by the space available at the coordinator. The spacial cost to host auxiliary authentication information is $B \log n$, where B could be very large. Suppose the space available at the coordinator is C. It is equivalent to the situation where there are $\frac{C}{B}$ units of space in the signer. Since we have already illustrated the impact of space in one single signer in Fig. 8, how signing/verifying throughput changes with varied available space in the coordinator is not shown for brevity.

We focus more on the potential maximal throughput of tested signature schemes. We suppose there is enough space at both the signer and verifier sides. The events sent to the coordinator are used for signing performance evaluation whereas the verification algorithm is fed by the events saved at the cloud. The number of events involved is listed in TABLE. VIII-E.

TABLE VII
SIMULATION RESULTS REGARDS THE NUMBER OF EVENTS.

Budget	500	1000
Signature	115821	186619
Verify	35038	70296

Fig. 14 shows the throughput comparison between GSC and DTC. We do not put results for sign-each approach in this figure because its throughput is much slower than the other two and we focus more on the visualization of more comparable results. From all performance evaluation experiments, sign-each approach is more than 50X slower than the other two. Since each day is one epoch and there are only 7 sensing devices, there are only $90 \times 7 = 630$ encryption/decryption operations for both GSC and DTC. For all the experiments conducted in this subsection, GSC is faster than DTC in terms of both signing and verifying. This is because processing an event in GSC is simpler than tree traversal in DTC, but not more hashing operations. We can verify this conjecture by analyzing the performance comparison in Fig. 14. We can see that performance gap between GSC and DTC becomes more prominent when quicker message digest function is applied. For example, the throughput gap increases from 0.35M events per second to 0.7M events per second if MD5 replaces SHA256 in the prototype emulation experiment using DSA signature and the budget limit is 500. The throughput decreases when the value of budget limit is reduced as implicated in Fig. 14, because the same number of encryption/decryption operations are amortized to fewer events.

IX. RELATED WORK

Many lightweight signature schemes are developed over the years which are especially attractive to IoT devices.

TESLA [51] and its variants [52], [53] achieve message authenticity while retaining their computational efficiency by delayed symmetric key disclosure. These methods require tight synchronization of the sender and the receiver. Therefore, they are not applicable to the network communication model described in this paper, where the IoT device does not send messages to the data consumer directly.

Another category is based on one-time signature (OTS), including BiBa [54] and HORS [55]. The high computation efficiency is achieved by one-way functions without trapdoors. However, OTS-based methods require pre-distribution and retransmission of large size public keys, which pose a heavy burden on IoT devices, especially for those deployed in the wild without wired connections.

To shift the expensive signature generation to the offline phase is also one direction towards efficient signature schemes. Yavuz *et al.* [56] proposed to synthesize digital signatures from pre-generated templates. This method however is limited for highly structured data. Furthermore, this method generates a large space overhead at the sender side.

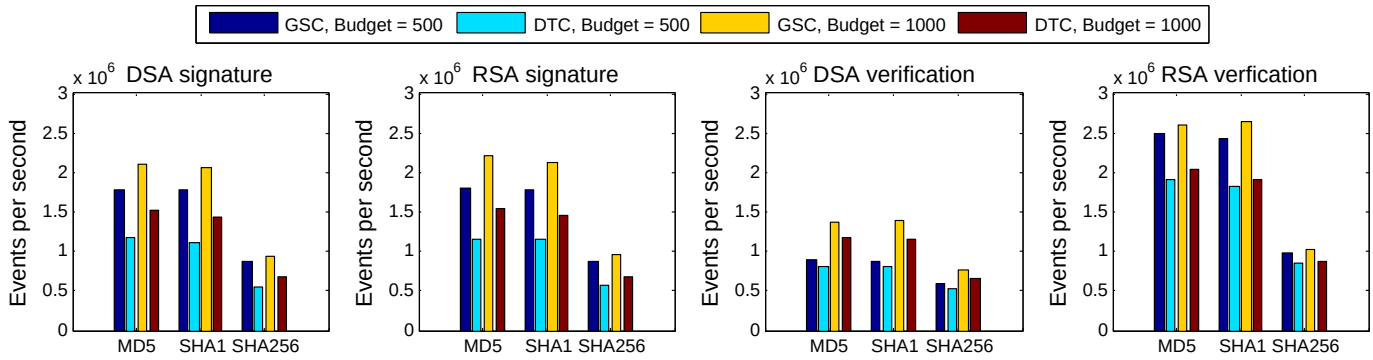


Fig. 14. Throughput comparison with different parameter settings.

A large population of works rely on amortization of one signature cost over multiple messages. Both DTC and GSC fall into this category. However, existing works [28], [57], [58] do not consider the issue of uniformity and consume more space than the solutions proposed in this paper.

X. Discussion and Future Work

A plausible problem of the evaluation discussion is that we completely ignore the space cost to store events themselves. In fact, these events can be saved in the disk whereas the space discussed in Sec. VIII must resides in the memory.

Different sensing devices may send generated data to the coordinator at vastly different speed. The video surveillance system [10] continuously generates tons of data whereas human-motion detector [59] sends much less data occasionally. To avoid starvation of devices, the sampling protocol discussed in this paper can be easily generalized to allow weighted items. How to automatically set weights for different devices with little human intervention would be our future work.

Network latency attributed to propagation and processing is inevitable. Network latency causes lagged round promotion which in turn results in substantial network bandwidth waste due to the transmission of events that should be discarded at devices locally. In our future work, we plan to design a queuing principle which prioritizes the coordinating messages to favor the devices sending more events thus to reduce the network bandwidth waste.

Our sampling and signature scheme can be also applied to other areas beyond IoT data storage. It is increasingly important to monitor networks at geographic locations in a scalable way [60]. Our sampling protocol and signature scheme provide the opportunity to relieve the burden of the network, where the local collector acts as the coordinator and periodically transmits the sampled packets to the global traffic analytic. Since the sampled packets may be transmitted over the Internet at 10/40 **Gbps, the memory usage of switches becomes critical. GSC with constant space overhead is especially suitable in this case.**

XI. Conclusion

We summarize the new challenges of the IoT data communication with authenticity and integrity and argue that existing solutions cannot be easily adopted. We design a system aimed to address these challenges. This system is able to uniformly sample data from sensing devices and then securely store the data in the cloud while respecting resource budget constraint. The sub-systems in our paper symbiotically operate together and this system is efficient in terms of space and time, as is validated by extensive simulation and prototype emulation experiments.

References

[1] X. Li, H. Wang, Y. Yu, and C. Qian, "An iot data communication framework for authenticity and integrity," in *Proc. of ACM/IEEE IoTDI*, 2017.

[2] "eHealth," http://www.who.int/topics/ehealth/en/.

[3] World Health Organization, "mHealth: New horizons for health through mobile technologies," http://www.who.int/goe/publications/goe_mhealth_web.pdf.

[4] M. Gerla, E. Lee, G. Pau, and U. Lee, "Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds," in *Proc. of IEEE WF-IoT*, 2014.

[5] G. Wang, J. Han, C. Qian, H. Xi, W.and Ding, Z. Jiang, and J. Zhao, "Verifiable smart packaging with passive rfid," *IEEE TMC*, 2018.

[6] "Nest," https://nest.com.

[7] "HVAC Monitoring System," https://www.sensaphone.com/industries/hvac.php.

[8] T. Gupta, R. P. Singh, A. Phanishayee, J. Jung, and R. Mahajan, "Bolt: Data management for connected homes," in *Proc. of USEIX NSDI*, 2014.

[9] Y. Kim, J. Kang, D. Kim, E. Kim, P. K. Chong, and S. Seo, "Design of a fence surveillance system based on wireless sensor networks," in *Proc. of the Autonomics*, 2008.

[10] A. J. Brush, J. Jung, R. Mahajan, and F. Martinez, "Digital neighborhood watch: Investigating the sharing of camera data amongst neighbors," in *Proc. of ACM CSCW*, 2013.

[11] J. Scott, B. B., J. Krumm, B. Meyers, M. Hazas, S. Hodges, and N. Villar, "Preheat: controlling home heating using occupancy prediction," in *Proc. of ACM Ubicomp*, 2011.

[12] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, 2013.

[13] "OpenSensors," https://www.opensensors.io/.

[14] C. Wang, Q. Wang, K. Ren, and W. Lou, "Ensuring data storage security in cloud computing," in *Proc. of IEEE IWQoS*, 2009.

[15] "AWS Cloud Hacked by Bitcoin Miners," https://www.enterprisetech.com/2017/10/09/aws-cloud-hacked-bitcoin-miners/.

[16] "AWS Cloud Hacked by Bitcoin Miners ," https://www.enterprisetech.com/2017/10/09/aws-cloud-hacked-bitcoin-miners/.

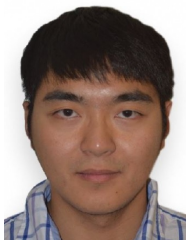
[17] "Computer hackers take to the cloud," https://www.sciencenewsforstudents.org/article/computer-hackers-take-cloud.

[18] P. Mahajan, S. Setty, S. Lee, A. Clement, L. Alvisi, M. Dahlin, and M. Walfish, "Depot: Cloud Storage with Minimal Trust," *ACM Trans. Comput. Syst.*, vol. 29, 2011.

[19] "The 10 Biggest Cloud Outages Of 2015," https://tinyurl.com/y7sxjf8m.

[20] D. Evan, "The Internet of Things, Cisco White Paper," https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf.

[21] "Amazon S3 Pricing," https://aws.amazon.com/s3/pricing/.
[22] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang, "Continuous sampling from distributed streams," *JACM*, vol. 59, no. 2, 2012.
[23] B. A. Bash, J. W. Byers, and J. Considine, "Approximately uniform random sampling in sensor networks," in *Proc. of ACM DMSN*, 2004.
[24] S. Chaudhuri, R. Motwani, and V. Narasayya, "Random sampling for histogram construction: How much is enough?" in *Proc. of ACM SIGMOD*, 1998.
[25] G. S. Manku, S. Rajagopalan, and B. G. Lindsay, "Approximate medians and other quantiles in one pass and with limited memory," in *Proc. of ACM SIGMOD*, 1998.
[26] S. Nath, P. B. Gibbons, S. Seshan, and Z. Anderson, "Synopsis diffusion for robust aggregation in sensor networks," *ACM TOSN*, vol. 4, no. 2, 2008.
[27] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Proc. of CRYPTO*, 1987.
[28] C. K. Wong and S. S. Lam, "Digital signatures for flows and multicasts," in *Proc. of IEEE ICNP*, 1998.
[29] "FIT IOT-Lab," https://www.iot-lab.info.
[30] K. Piotrowski, P. Langendoerfer, and S. Peter, "How public key cryptography influences wireless sensor node lifetime," in *Proc of ACM SASN*, 2006.
[31] R. Gennaro and P. Rohatgi, "How to sign digital streams," in *Crypto*, 1997.
[32] Y. Zhang, L. Duan, and J. L. Chen, "Event-Driven SOA for IoT Services," in *Proc. of IEEE SCC*, 2014.
[33] G. Wang, H. Cai, C. Qian, J. Han, X. Li, H. Ding, and J. Zhao, "Towards replay-resilient rfid authentication," in *Proc. of ACM MobiCom*, 2018.
[34] W. Xi, C. Qian, J. Han, K. Zhao, S. Zhong, X. Li, and J. Zhao, "Instant and robust authentication and key agreement among mobile devices," in *Proc. of ACM CCS*, 2016.
[35] "SHA-1," http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf.
[36] "The MD5 Message-Digest Algorithm," https://tools.ietf.org/html/rfc1321.
[37] S. Sattolo, "An algorithm to generate a random cyclic permutation," *Information processing letters*, vol. 22, no. 6, 1986.
[38] J. Tukey, "Bias and confidence in not-quite large sample," *Annals of Mathematical Statistics*, 1958.
[39] D. Verbyla, "Potential prediction bias in regression and discriminant analysis," *Canadian Journal of Forest Research*, vol. 16, no. 6, 1986.
[40] Z. Zhou, "Ensemble learning," *Encyclopedia of biometrics*, pp. 411–416, 2015.
[41] B. Efron, *The jackknife, the bootstrap, and other resampling plans*. Siam, 1982, vol. 38.
[42] I. B. Damgård, "A design principle for hash functions," in *Proc. of Crypto*, 1989.
[43] "xxHash," http://www.xxhash.com/.
[44] "Sampling for Big Data," www.kdd.org/kdd2014/tutorials/t10_part1.pptx.
[45] J. S. Vitter, "Random sampling with a reservoir," *ACM Transactions on Mathematical Software*, vol. 11, no. 1, 1985.
[46] E. Cohen and H. Kaplan, "Summarizing data using bottom-k sketches," in *Proc. of ACM PODC*, 2007.
[47] D. Boneh, E. Shen, and B. Waters, "Strongly unforgeable signatures based on computational diffie-hellman," in *Proc. of PKC*, 2006.
[48] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *CACM*, vol. 21, no. 2, 1978.
[49] "DSA," http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.
[50] "Smart Data Set for Sustainability," http://traces.cs.umass.edu/index.php/Smart/Smart.
[51] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "Efficient authentication and signing of multicast streams over lossy channels," in *Proc. of IEEE S&P*, 2000.
[52] D. Liu, P. Ning, S. Zhu, and S. Jajodia, "Practical broadcast authentication in sensor networks," in *Proc. of IEEE MobiQuitous*, 2005.
[53] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler, "Spins: Security protocols for sensor networks," *Wireless networks*, vol. 8, no. 5, 2002.
[54] A. Perrig, "The biba one-time signature and broadcast authentication protocol," in *Proc. of ACM CCS*, 2001.
[55] L. Reyzin and N. Reyzin, "Better than biba: Short one-time signatures with fast signing and verifying," in *Australasian Conference on Information Security and Privacy*, 2002.
[56] A. A. Yavuz, "An efficient real-time broadcast authentication scheme for command and control messages," *IEEE TIFS*, vol. 9, no. 10, 2014.
[57] A. Lysyanskaya, R. Tamassia, and N. Triandopoulos, "Multicast authentication in fully adversarial networks," in *Proc. of IEEE S&P*, 2004.
[58] S. Miner and J. Staddon, "Graph-based authentication of digital streams," in *Proc. of IEEE S&P*, 2001.
[59] T. Yamada, Y. Hayamizu, Y. Yamamoto, Y. Yomogida, A. Izadi-Najafabadi, D. N. Futaba, and K. Hata, "A stretchable carbon nanotube strain sensor for human-motion detection," *Nature nanotechnology*, vol. 6, no. 5, 2011.
[60] L. Elsen, F. Kohn, C. Decker, and R. Wattenhofer, "goProbe: a scalable distributed network monitoring solution," in *Proc. of IEEE P2P*, 2015.



Xin Li (M'15) is a Ph.D. candidate at the Department of Computer Science and Engineering, UC Santa Cruz. He received the B.Eng. degree in Communication Engineering from University of Electronic Science and Technology of China and M.S. degree in Electrical Engineering from University of California Riverside. His research interests include network security, software defined networking, network functions virtualization and Internet of Things.

Minmei Wang (M'17) received the B.E. degree from Nanjing University of Posts and Telecommunications in 2014, and the M.Sc. degree from the Nanjing University in 2017. She is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, University of California at Santa Cruz. Her research interests include Internet of Things and network security. She is a student member of the IEEE.

Huazhe Wang (M'15) received the B.Sc. degree from Bejing Jiaotong University in 2011, and the M.Sc. degree from the Beijing University of Posts and Telecommunications in 2014. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, University of California at Santa Cruz. His research interests include softwaredefined networking and network security. He is a Student Member of the ACM.

Ye Yu (M'13) is currently a software engineering at Google Inc. He completed his Ph.D in Computer Science from University of Kentucky at 2018. He received the BSc degree from Beihang. His research interests including data center networks and software defined networking.

Chen Qian (M'08) is an Assistant Professor at the Department of Computer Science and Engineering, UC Santa Cruz. He received the B.Sc. degree from Nanjing University in 2006, the M.Phil. degree from the Hong Kong University of Science and Technology in 2008, and the Ph.D. degree from the University of Texas at Austin in 2013, all in Computer Science. His research interests include computer networking, data-center networks and cloud computing, Internet of Things, and software defined networks. He has published more than 60 research papers in a number of top conferences and journals including *ACM SIGMETRICS*, *IEEE ICNP*, *IEEE ICDCS*, *IEEE INFOCOM*, *IEEE PerCom*, *ACM UBICOMP*, *ACM CCS*, *IEEE/ACM Transactions on Networking*, and *IEEE Transactions on Parallel and Distributed Systems*. He is a member of IEEE and ACM.