Embracing Tag Collisions: Acquiring Bloom Filters across RFIDs in Physical Layer

Zhenlin An, Qiongzheng Lin, Lei Yang and Wei Lou
Department of Computing, The Hong Kong Polytechnic University, Hong Kong
Email: {an, lin, young}@tagsys.org, csweilou@comp.polyu.edu.hk

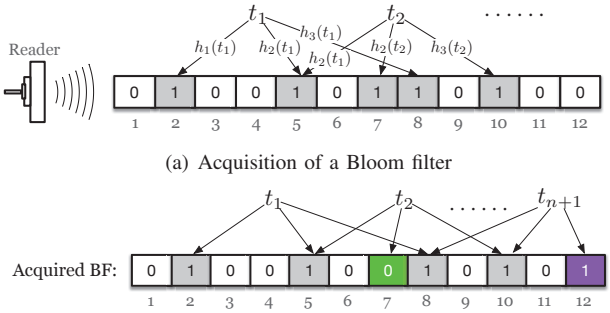
***Abstract*—Embedding Radio-Frequency IDentification (RFID) into everyday objects to construct ubiquitous networks has been a long-standing goal. However, a major problem that hinders the attainment of this goal is the current inefficient reading of RFID tags. To address issue, the research community introduces the technique of *Bloom Filter* (BF) to RFID systems. This work presents TagMap, a *practical* solution that acquires BFs across commercial off-the-shelf (COTS) RFID tags in the *physical layer*, enabling upper applications to boost their performance by orders of magnitude. The key idea is to treat all tags as if they were a single virtual sender, which hashes each tag into different *intercepted inventories*. Our approach does not require hardware nor firmware changes in commodity RFID tags - allows for rapid, zero-cost deployment in existing RFID tags. We design and implement TagMap reader with commodity device (e.g., USRP N210) platforms. Our comprehensive evaluation reveals that the overhead of TagMap is 66.22% lower than the state-of-the-art solution, with a bit error rate of 0.4%.**

I. Introduction

Today's largest and fastest growing market of the Internet of Things (IoT) by unit sale comes from the Radio Frequency IDentification (RFIDs) [1]. In RFID systems, a device called the *reader* transmits a continuous high-power RF signal. Nearby *tags* can modulate the reader's signal by changing the impedance match state of antenna to convey a message of zeros and ones back to the reader. Such communication allows tags to work without batteries; they operate solely by harvesting the energy from reader's RF signal [2].

A fundamental operation in RFID systems is to scan and read 96-bit or 128-bit Electronic Product Codes (EPCs, aka ID) from tags, wherein time efficiency is a crucial performance metric, especially when dealing with large volumes of RFID tags. The challenge is that tags can collide and cancel each other out, resulting in wastage of bandwidth and an increase in the total delay. Many existing work made efforts to design more efficient anti-collision inventory protocols [3]. However, the lack of traditional transceivers [2] prevents tags from utilizing a suitable channel to transmit additional bits per symbol and increasing the bandwidth efficiency. Other sophisticated communication mechanisms (e.g., CDMA or FDMA) are unsuitable due to their high energy demand. Worsely, tags merely rely on the reader to schedule their medium access with the framed-slotted ALOHA protocol because they cannot "hear" from each other. These limitations force the reader to go through a time-consuming inventory procedure.

To address the issue, the research community introduces the technique of *Bloom Filter* (BF) to RFID systems [4]. Unlike



(a) Acquisition of a Bloom filter

(b) Presence test with an acquired BF

Fig. 1: Acquisition of a Bloom filter in an RFID system. (a) Initially, the M-bit BF bitmap begins as an array of zeros. The reader divides the acquisition procedure into M time slots corresponding to the M bits. Each tag in the set $T = \{t_1, t_2, \ldots, t_n\}$ is hashed K times using the hash functions of $\{h_1, h_2, \ldots, h_K\}$ into K slots, in each of which the tag yields a short presence signal to show its presence. The reader sets a bit to 1 if any signal is detected in the corresponding slot. (b) To check if a tag is present, hash it K times and check the corresponding bits in the acquired BF. For example, the t_2 cannot be on the spot, since a '0' is found at one of the bits; a new tag (e.g., t_{n+1}) must arrive since unwanted '1' is found.

previous anti-collision protocols that *avoid* collisions, these works *embrace* collisions as informative feedbacks. A Bloom filter is a space- (or time-) efficient probabilistic data structure that is used to represent a set. It can tell whether an element is a member of the set that it represents. Fig. 1 demonstrates how the reader acquires a BF across tags and how the acquired BF is utilized to test the presence of a tag.

The upper layer can employ the acquired BF to explore many notable applications. For example, consider a warehouse that stores a large number of high-valued commercial products or a military base that stocks a large number of guns and ammunition packages [4]. How can a staff *immediately* determine if anything is missing? The BF naturally fits the demand of this application because it is an exact representation of the current tag set. Specifically, all tags are hashed K times and the corresponding bits in the acquired BF are verified. One tag is considered to have been definitely moved out or missing when a '0' is found; otherwise, it is assumed to be still present, although this assumption may be incorrect with a low probability.

Yet, RFID-oriented protocols or applications that appeal to BFs are actually NOT available to today's RFID systems, because translating the idea of acquiring BFs across RFIDs

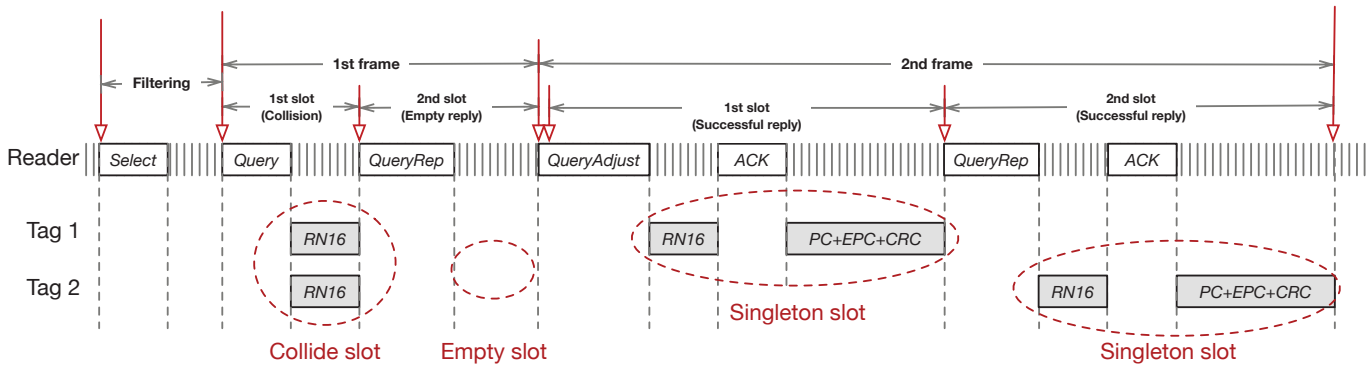


Fig. 2: Gen2 inventory procedure. The reader starts an inventory procedure with a `Select` command, which selects a group of tags to join. Each inventory is composed of many frames, which are initiated by either a `Query` or `QueryAdjust` command. Further, the reader divides a frame into many time slots with `QueryRep`. Tags (e.g., tag 1 and tag 2) randomly selects a time slot to reply. In particular, a short reply called `RN16` must be transmitted for collision avoidance before the transmission of a long reply.

into a practical system faces three major challenges:

- First, the primary challenge arises from the hardware constraint. The core of BF is the on-tag hash functionality, which allows each tag to select time slots according to the hash value of its `EPC`, making results predicable - exactly knowing which bits a desired tag is mapped into. Current pseudo-random generators of tags cannot meet the demand of predictability. One option is to supplement newly designed hash functions to tags [5]. However, they require new hardware (thousands of gate equivalents [6]) for RFID tags, making them significantly more expensive.
- Second, transmitting and receiving *presence signals* is non-compliant with today's RFID air protocols. These non-standard operations would leave out the billions of RFIDs already deployed in the today's world, some of which were even installed a decade ago.
- Third, the pioneering work named Tash [7] takes a step towards designing hash primitives in the application layer. Unfortunately, applications have no control of the physical-layer inventory procedure. Thus the application-layer Tash still requires the time-consuming anti-collision processing, without unleashing full efficiency potentials of embracing collisions.

In this work, we present TagMap, a *practical* solution that acquires BFs across commercial off-the-shelf (COTS) RFID tags in the *physical layer*, enabling upper applications to boost their performance by orders of magnitude. TagMap introduces multiple innovations that enable it to deal with the above challenges. First, TagMap shifts the computation loads for hashing from tag side to the reader side by an emulated hash function. When running, it treats a portion of the pre-stored hash values as a new hash value. Second, TagMap reader forces all tags that are hashed into this bit to transmit their `RN16` replies in the first single slot and regards a detected `RN16` reply as the presence evidence of a tag and skips the remaining anti-collision phase. Third, TagMap defines the `Modulo` operation, allowing to acquire arbitrarily sized BFs.

Summary of Results. TagMap offers a new and interesting tradeoff point on the spectrum between feasibility and effectiveness. Our experiments reveal the following findings:

- TagMap is compatible with the EPCglobal Gen2 protocol and can serve 97.5% types of commercial tags in today's market, even some were deployed 13 years ago.
- TagMap reduces the acquisition overhead by 74.73% and 66.22% compared with commercial readers (e.g., ImpinJ R420) and Tash [7] respectively. TagMap has a mean bit error rate of less than 0.4%.
- TagMap can accurately find out $97.1 \pm 0.4\%$ of missing events within 2 seconds when 20% of tags are removed, whereas a commercial reader requires more than 15 seconds to know these events.

Key Contributions. This work presents a system that can acquire BFs across COTS RFID tags in the physical layer. Our design presents three key innovations. First, it emulates on-tag hash and modulo functionality; second, it intercepts an inventory to test if any tags are hashed into the corresponding bit; finally, this work also presents a prototype implementation and reveals the utility of BFs in typical application scenarios.

II. Background

A. A Primer on Gen2 Air Protocol

The Gen2 RFID system adopts the reader-talks-first mode, in which the reader dominates the communication and all the tags follow its commands. Physical and MAC layers are designed with power asymmetry.

Physical Layer: The reader transmits a continuous high power RF signal (called *continuous wave*, or CW), which is used to supply energy to tags and carrys out query commands. Nearby RFID tags reply to the reader's query by reflecting the high power RF signal via ON-OFF keying. The tags transmit a bit by changing the impedance on their antennas.

MAC Layer: A framed ALOHA MAC protocol, called Q-adaptive, is employed in the Gen2 standard to avoid collisions. The reader divides time into several slots, which are further organized into frames. As shown in Fig. 2, the reader starts an inventory by broadcasting a `Select` command for choosing a group of tags to join in the incoming inventory (select all by default). Then the command `Query` or `QueryAdjust`

follows to start a new frame, in which each unidentified tag randomly selects a time slot to reply. The command QueryRep indicates a new beginning of a slot. During a slot, the tag transmits a 22-bit *short reply* (called RN16) firstly for collision detection. Then, either of the following two cases occurs on the reader side:

- RN16 is successfully decoded, which indicates that only one tag was transmitting (i.e., *singleton slot*). The reader then requests its 128-bit *long reply* (i.e., PC + EPC+ CRC) by issuing an ACK command.
- Otherwise, the situation indicates either a collision or non-reply (i.e., called *collision slot* and *empty slot*). In this case, the reader proceeds to the next slot without requesting any long reply. Thus, singleton slots are longer than other slots.

In the end of each frame, if any collision slot is detected or any long reply is corrupted, then the reader restarts a new frame using QueryAdjust for these tags. QueryAdjust is able to adjust the frame length based on the number of collisions in the previous frame. This procedure terminates till all tags are identified.

B. Comparison with the Sate-of-the-Art

This work is inspired by a pioneering work - Tash [7], which emulates a group of hash primitives in the *application layer*. However, our solution is more *efficient* and *practical* prior to Tash at least in two aspects.

- **Practicality**: As an application-layer emulation technique, Tash highly depends on a Gen2 command, i.e., Truncate, which forces tags to transmit truncated EPCs. These incomplete EPCs are viewed as the presence evidence of tags. Unfortunately, to our best knowledge and investigation, we have not found any Truncate-available tags on the market. Actually, this issue has been claimed in [7]. This is also the reason that the performance of Tash [7] related to this command is evaluated with simulations, instead of real tags. On the contrary, our solution uses RN16 reply as presence evidence. As a fundamental reply, the RN16 reply is completely serviceable in today's Gen2 tags.
- **Efficiency**: The anti-collision processing is actually unavoidable in Tash, because there is no way to control the inventory procedure in the application layer. No relevant APIs are provided by commercial readers so far. Our approach intercepts an inventory in the physical layer to skip the anti-collision processing.

In short, TagMap is highly advantageous due to its physical-layer performance boost and full compatibility with COTS tags.

III. System Design

In this section, we firstly sketch the overall solution and then progressively elaborate technique details.

A. Formulation of Bloom Filter

A BF represents a set $T = \{t_1, t_2, \ldots, t_n\}$ with n tags. It is described by an array of M bits, which are initially set to 0. We can use K hash functions, $\{h_1, h_2, \ldots, h_K\}$ to map each tag in the universe to a number within the range $\{1, \ldots, M\}$. For each tag $t \in T$, the bits of $\{h_1(t), \ldots, h_K(t)\}$ are set to 1. A bit can be set to 1 multiple times (e.g., collisions), but its value remains 1. To check if a tag t is in T, we check if all bits of $\{h_1(t), \ldots, h_K(t)\}$ are set to 1. If not, then t is not in the tag set. If all bits are set to 1, we assume that t is in T, although this assumption may be wrong with sufficiently low probability, namely, yielding a false positive. The false positive rate depends on parameter selection: using many hash functions equates to a high probability of finding '0' bits for each tag that is not in the set; using a few hash functions increases the fraction of '0' bits in the map. Here, we present two useful Lemmas as follows:

Lemma 1: The false negative rate is minimized when the number of hash functions $K = \lceil \ln 2 \cdot (M/n) \rceil$.

Lemma 2: Given a tolerance ε (i.e., upper bound of false positive rate), BF length $M \geq \lceil n \log_2 e \cdot \log_2(1/\varepsilon) \rceil$.

Their proofs are similar in nature to the standard analysis of the Bloom filter as shown in [8]. The number of tags n is assumed to be known already. It can be discovered through a preliminary inventory or stored in backend.

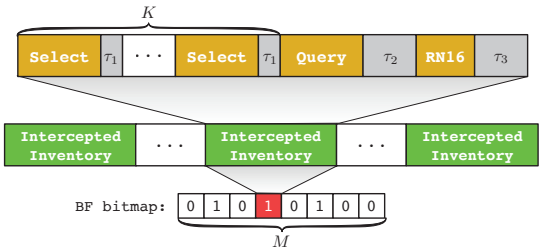


Fig. 3: Acquisition solution. To acquire an M-bit BF, TagMap reader initiates M intercepted inventories, each of which contains K Selects, one Query and a time window for RN16 reply.

B. Solution Sketch

The features and usage of the Bloom filter have been widely studied in various fields [8]. They are not our focus in this work. Instead, we more concern about the way to acquire a BF across RFID tags, which must be tightly integrated with the standard EPCglobal Gen2 RFID air protocol [9], making billions of tags serviceable. The whole solution consists of two phases as follows:

- In offline phase, we compute the hash value of each tag's EPC and store it inside the tag's *non-volatile* memory bank for future use. The storing can be accomplished by using the standard Gen2 command of Write. This phase is only initiated once. For example, it is initiated when the user writes EPCs to tags.
- In online phase, the reader acquires a BF bit by bit. The TagMap reader initiates an intercepted inventory for each bit. The inventory is used to answer the question of presence that if any tag is hashed into the corresponding bit. If answer is positive, the bit is set to one; otherwise, it is set to zero.

Fig. 3 shows the acquisition procedure from a high-level. To acquire an M-bit BF, the reader initiates M intercepted inventories, each of which corresponds to a bit in the BF. In

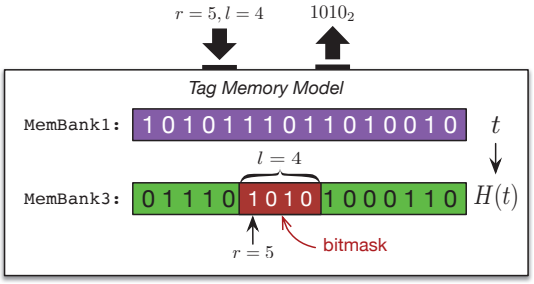


Fig. 4: Illustration of a hash function. The figure shows a tag's two memory banks: `MemBank1` and `MemBank3`, which store the tag's `EPC` t and the real hash value $H(t)$ respectively. Given an input of $r = 5$ and $l = 4$, the output of $h^{(4)}(t, 5)$ equals 1010_2, which exactly is the sub-bitstring of $H(t)$ within $[5, 9)$ bits stored in `MemBank3`.

each inventory, the reader broadcasts K `Select` commands and one `Query` command; the tags matching the hash rules are forced to transmit their `RN16` replies in the first slot. In the end of the intercepted inventory, the reader determines the value of the corresponding bit. The subsequent sections progressively introduce this acquisition approach by discussing the following issues:

- *Emulating hash function*. Commercially available tags at present do not support hash functions. Thus, our first task is to emulate a hash function that works as if it was embedded on each tag.
- *Answering the question of presence*. With the emulated hash functions, a TagMap reader initiates an intercepted inventory to suggest whether any tag that can output a given hash value is present or not.
- *Acquiring a Bloom filter*. Finally, we show how the reader builds a whole BF bit by bit through a series of intercepted inventories in three cases.

C. Emulating Hash Functions

A hash function, denoted by $h^{(l)}(t, r)$, outputs an l-bit hash value when given a tag's `EPC` t and a random number r. It takes three inputs: (1) an l_t-bit tag's `EPC` number t; (2) an l_r-bit random number r; and (3) the bit length of the hash value l. Formally,

$$h^{(l)}(t, r) : \{0, 1\}^{l_t} \times \{0, 1\}^{l_r} \rightarrow \{0, 1\}^{l} \qquad (1)$$

where l is also called the *dimension* of the hash function. The output is randomly distributed in $[0, 2^l)$. Following a common practice, we create K hash functions by shuffling the output using K different random numbers. The k^{th} hash function is denoted by $h_k^{(l)}(t, r_k)$. The superscript l is sometimes omitted, such as $h(t, r_k)$, $h_k(t)$ or h_k, for clarity unless it should be noted.

Memory Banks. Before introducing the hash design, we need to introduce some background about the memory bank of tag. The Gen2 air protocol specifies a simple tag memory model: each tag contains four types of non-volatile memory blocks (called *memory banks*). The first three banks are used to store the password, `EPC` number and `TID` number. The last memory bank (`MemBank3`) is reserved for user-defined data. Fig. 4 demonstrates the first and the third memory bank. The `Read` or `Write` commands are used to read or write data from or into these banks respectively.

Hash Function. Initially, we compute the *real hash value* (denoted by $H(t)$) of a tag's `EPC` by using `SHA-1` for tag t, and then store $H(t)$ in the tag's `MemBank3` for future use. Note both operations are performed once in the offline phase. We define a tag's sub-bitstring within $[r, r + l)$ in `MemBank3` as its l-bit hash value challenged by the random number r. In other words, our hash value $h^{(l)}(t, r)$ is a portion of $H(t)$ stored in `MemBank3`. Fig. 4 demonstrates the relation between the real hash value and our hash value, where the tag's 4-bit hash value $h^{(4)}(t, 5) = 1010_2$[1]. Clearly, our design does not requires a tag to equip a real hash function or gets its chip involved.

Discussion. A few points are worth noting:

- Evidently, $h^{(l)}(t, r)$ is random and its randomness is derived from $H(t)$, which is supposed to have good quality of randomness at each bit. If the upper application needs a stronger randomness, one could store a truly random bitstring in the `MemBank3` rather than the real hash value, meanwhile save the mapping between the `EPC` and the bitstrings in database for future query. In addition, to avoid the dependence of K hash values, one can tactfully choose $r_1, \ldots, r_K$ such that $[r_1, r_1 + l) \cap \cdots \cap [r_K, r_K + l) = \varnothing$.
- The computation of $H(t)$ is performed in a host computer rather than on tags, so its workload is almost neglected. Storing $H(t)$ is also performed once until the tag's `EPC` is changed, which is unlikely to happen often in practice. Although writing data into memory bank usually takes a longer time than reading them, the writing is performed in offline phase thereby its workload is not counted into the acquisition overhead.
- A possible concern is if today's RFID tags in the market can serve our design in terms of the operability of `MemBank3`. We investigate 40 tag chips from ImpinJ [10], Alien and NXP, whose total market share exceeds 80% [1]. The investigation result suggests that TagMap is completely compatible with the EPCglobal Gen2 air protocol and can serve 97.5% types of tags, except those that are *read-only*. More details refer to §IV.

D. Answering the Question of Presence

A simple way to obtain a tag's hash value is to `Read` the corresponding sub-bitstring from its `MemBank3`. However, keep in mind that our ultimate goal is to acquire a BF across all tags instead of their hash values. Here, we consider a question, named *question of presence*, that is, *is there any tag which can output a wanted hash value?* Formally, given the input pair (r, l) and the hash value v, we determine if

$$\left| \{t | h^{(l)}(t, r) = v\} \right| > 0$$

[1] $(\cdot)_2$ means a value in binary format.

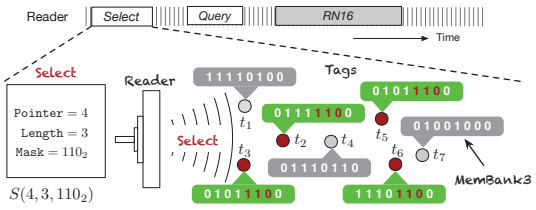


Fig. 5: Illustration of selective reading in Gen2 protocol. 7 tags are covered by a reader. The reader initiates a `Select` command $S(4, 3, 110_2)$. As a result, tags of t_2, t_3, t_5 and t_6 (highlighted in red) are only selected to join in the incoming inventory because their `MemBank3`s' sub-bitstring (highlighted in red) within $[4, 7)$ equals 110_2. Other tags do not meet the condition and remain silent.

for all $t \in T$, where $|\cdot|$ denotes the cardinality of a set. Since our hash value is a portion of the real one stored in a tag's `MemBank3`, the question can be converted to a new form as follows:

Problem 1: Is there any tag whose sub-bistring $\in [r, r+l)$ *in its* `MemBank3` *is equal to* v*?*

Selective Reading. As mentioned in §II-A, any inventory procedure must be started with a `Select` command. Gen2 protocol specifies that each tag maintains a flag variable `SL`. The reader can use the `Select` command to turn the `SL` flags of tags into `asserted` (i.e., true) or `deasserted` (i.e., false). Only the asserted tags will respond to a `Query` (or `QueryAdjust`) command, participating the inventory. The `Select` command is composed of 8 fields. For simplicity, we only introduce the four fields, which will be employed in our design: `MemBank`, `Pointer`, `Length` and `Mask`.

- `Mask`: it is a specific bitstring that should match the content of a specific position in a `MemBank`.
- `Length`: it defines the length of the mask in bits.
- `MemBank`: it specifies which memory bank the mask will be compared with. We consider `MemBank3` only by default in our scenario.
- `Pointer`: it specifies the starting position in the `MemBank` where the mask will be compared with.

These fields are combined to compose a *selection rule*. We formally denote the rule as follows:

$$S(\underbrace{p}_{\texttt{Pointer}}, \overbrace{l}^{\texttt{Length}}, \underbrace{m}_{\texttt{Mask}})$$

which implies that only tags whose `MemBank3`s' sub-bitstring $\in [p, p+l)$ is equal to m will be selected. Fig. 5 shows an example in which four tags are selected to join in the inventory. Note that multiple `Select`s are allowable (discussed later).

Intercepted Inventory. The function of selective reading offers an opportunity to select the tags based on a mask. If the reader broadcasts the following `Select`,

$$S(r, l, v)$$

, then only the tags whose sub-bitstring $\in [r, r+l)$ in its `MemBank3` is equal to v, are selected to participate in the incoming inventory. Interestingly, the set of selected tags

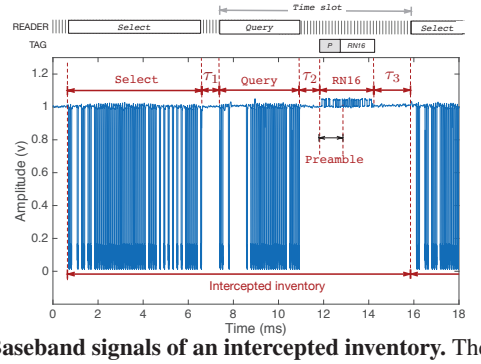
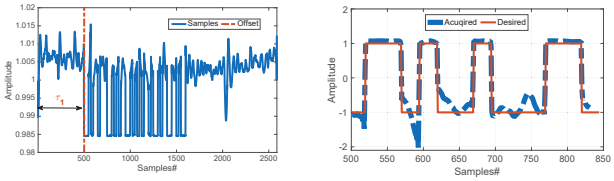


Fig. 6: Baseband signals of an intercepted inventory. The inventory contains a `Select`, a `Query` and an `RN16` reply. τ_1, τ_2 and τ_3 are key timing parameters, which are specified in the Gen2 protocol. The reader detects the presence of `RN16` reply to determine if any tag responds.



(a) `RN16` (b) Preamble

Fig. 7: Baseband signals of `RN16` reply. (a) shows the signals of a `RN16` reply, which is composed of a 5-bit preamble and 16 random bits. (b) shows zoomed-in preamble and its comparison with the desired template.

exactly fit the question of presence! To know if its cardinality is greater than zero, we let the reader afterwards broadcast a `Query` command to start a single-slot frame, by setting the field of frame length to 1. All selected tags are forced to reply in the following slot immediately. As aforementioned, a tag must transmit an `RN16` reply firstly before sending its `EPC` reply. Our trick here is that the `RN16` reply is regarded as the presence evidence of a tag, which has the given hash value. As long as the reader detects any `RN16` reply signal, the answer to the question of presence is negative. Finally, the reader ignores the request of long reply even if a single tag responds and terminates the inventory procedure. We call the above procedure *intercepted inventory*. The inventory is "intercepted" because an intact inventory is supposed to `ACK` the tag's `RN16` reply and request the long reply subsequently (see Fig. 2). In contrast to the entire inventory, the interval of an intercept inventory is kept constant regardless of how many tags participate the procedure. Fig. 6 shows the baseband signals of an intercepted inventory acquired by USRP N210.

Detecting Presence Signals. Each 22-bit `RN16` reply contains a 6-bit preamble and 16 random bits. The reader can detect an `RN16` reply by correlating its preamble template with the received signals. Considering that the 16-bit random signals are independent of the preamble, their correlation is nearly zero except when the preamble is perfectly aligned with the beginning of an `RN16`. One or multiple spikes appearing at the correlation result indicate the presence of one or multiple tags possessing the given hash value. Fig. 7(a) shows an example of `RN16` reply sent from a single tag. Fig. 7(b) zooms

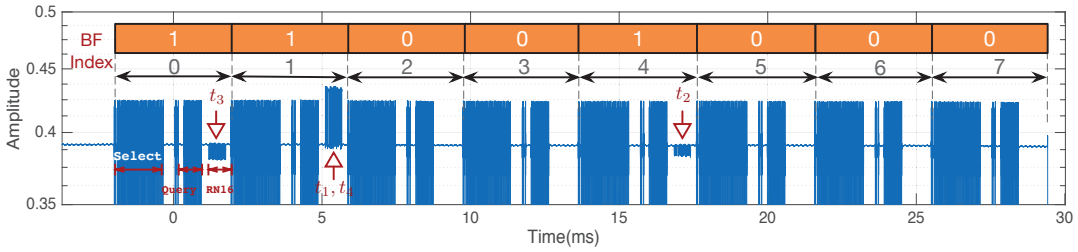


Fig. 8: Baseband signals of acquiring an 8-bit BF across 3 tags. The bitmap totally contains 8 bits, each of which corresponds to an intercepted inventory. The three tags reply in the 1^{st}, 2^{nd} and 5^{th} inventory respectively, so the final BF equals $[1, 1, 0, 0, 1, 0, 0, 0]$.

into its preamble compared with the template.

E. Acquiring a Bloom Filter

So far, we have discussed how the TagMap reader determines if any tag outputs a given hash value. Now, we start to acquire a whole BF by considering three cases progressively:

1) Case 1: Acquisition with a Single Hash: We firstly consider a simplified case, namely, acquiring a 2^l-bit BF using a single hash function: $h^{(l)}(t, r)$. Correspondingly, the bitmap size $M = 2^l$. The definition of hash bitmap indicates that: a tag t is hashed into the m^{th} bit of the bitmap if and only if $h^{(l)}(t, r) = m$ for $t \in T$, where $m = 0, \ldots, M-1$.

Namely, *the index of the bitmap actually implies the hash value*. We build a BF bit by bit, traversing the index from 0 to $M-1$. When building the m^{th} bit, we ask the question of presence like this: is there any tag whose hash value is equal to m? To do so, the reader initiates an intercepted inventory with the `Select` command of $S(r, l, m)$. If the answer is positive, the m^{th} bit of the BF is set to '1'; otherwise, '0'. This procedure continues until M bits are all traversed.

Fig. 8 shows the baseband signals of acquiring an 8-bit BF across 3 tags. From the figure, we can observe 8 intercepted inventories. Zooming into any one of them, three distinct segments - `Select`, `Query` and `RN16` - can be observed, exactly as same as a single intercepted inventory shown in Fig. 6.

2) Case 2: Acquisition with K Hashes: Secondly, we consider to acquire a 2^l-bit BF using K hash functions: $h^{(l)}(t, r_1), \ldots, h^{(l)}(t, r_K)$. In this case, the m^{th} bit is set to '1' when any one tag is hashed into the bit by any one of these K hash functions. Correspondingly, the question of presence turns to

$$\left|\{t|h^{(l)}(t, r_1) = m|| \ldots ||h^{(l)}(t, r_K) = m\}\right| > 0$$

for all $t \in T$. With regard to our hash function, the question can be expressed as follows:

Problem 2: Is there any tag whose sub-bitstring $\in [r_1, r_1 + l - 1)$, $\ldots$, *or* $\in [r_K, r_K + l - 1)$ *in its* `MemBank3` *is equal to* m*?*

Since each range corresponds to a `Select`, answering the above question requires K `Selects`.

Interestingly, Gen2 allows the reader to broadcast multiple `Selects` before the `Query`. The result of multiple `Select` commands is a union of the selected tags: if a tag's `SL` variable is asserted multiple times, then the final value remains asserted. The consequence is equivalent to combining the sets of selected tags. This is exactly the answer to the above question of presence. Therefore, for the m^{th} bit, the reader initiates an intercepted inventory, which contains K `Selects` as follows:

$$S(r_1, l, m), S(r_2, l, m), \ldots, S(r_K, l, m) \tag{2}$$

Similarly, if any `RN16` signal is detected after the `Query`, the corresponding bit is set to '1'.

3) Case 3: Acquisition with Arbitrary Size: The first two cases consider to generate BFs with size of $M = 2^l$ bit. The size M is a key parameter that directly determines acquisition overhead, and thereby should be well optimized. Lemma 1 implies the minimal (and optimal) size. Apparently, the minimal size of BF might not be exactly an exponent of 2. On the other hand, the hash values are stored in a binary format in the memory bank, so l must be an integer. As a result, the actual size M has to be set to 2^l and $l = \lceil \log_2 \widehat{M} \rceil$ suppose $\widehat{M}$ is the optimal size, such that $M \geq \widehat{M}$ and l is an integer. This setting is valid but might be extremely costly. For example, suppose the optimal size $\widehat{M} = 257$, we have to set actual size $M = 2^{\lceil \log_2 257 \rceil} = 512$, which is almost double than the optimal one.

To enable an arbitrary size, we define the operation of `modulo` to shrink the size, as used in other applications [8]. Apparently $M \geq \widehat{M}$. Traversing the m^{th} hash value where $m = 0, \ldots, M-1$, the reader forces tags hashed into m^{th} bits to reply at the

$$h^l(t, r) \bmod \widehat{M} \tag{3}$$

-th intercepted inventory. In other words, if a tag is supposed to be hashed into the bit greater than or equal to $\widehat{M}$, then the tag is moved to reply at $(m \bmod M)^{th}$ bit-inventory. As illustrated in Fig. 9, the `modulo` moves the replies at the bits equal to or greater than $\widehat{M}$ ahead to align with the first bits. Correspondingly, the question of presence turns to:

Problem 3: Is there any tag whose sub-bitstring $\in [r_1, r_1 + l)$, $\ldots$, *or* $\in [r_K, r_K + l)$ *in its* `MemBank3` *is equal to* m, $m + \widehat{M}$, $m + 2\widehat{M}$, $\ldots$, *or* $m + \lfloor \frac{M}{\widehat{M}} \rfloor \widehat{M}$*?*

Therefore, for the m^{th} bit, the reader initiates an intercepted inventory, which contains multiple `Selects` as follows:

$$S(r_1, l, m), S(r_2, l, m), \ldots, S(r_K, l, m)$$
$$\cdots\cdots$$
$$S(r_1, l, m + \lfloor \tfrac{M}{\widehat{M}} \rfloor \widehat{M}), S(r_2, l, m + \lfloor \tfrac{M}{\widehat{M}} \rfloor \widehat{M}), \ldots, S(r_K, l, m + \lfloor \tfrac{M}{\widehat{M}} \rfloor \widehat{M})$$

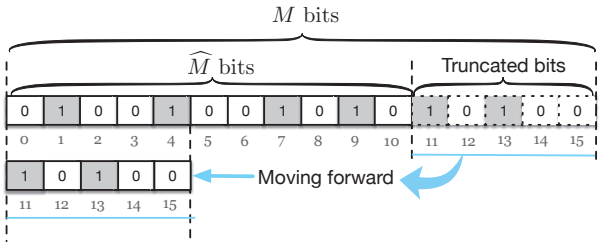


Fig. 9: The design of `modulo`. Suppose the optimal size of the BF is 11, the reader builds $2^{\lceil \log_2 11 \rceil} = 16$ bit BF and moves the last 5 bits to align with the first five.

It is worthing noting the `modulo` operation would increase the number of tags hashed into the head bits, breaking the randomness of BF. If the upper-layer application requires a strong randomness, a good way is to move truncated bit (i.e., indexes $\geq \widehat{M}$) forward to a random index less than $\widehat{M}$ while the reader labels the mapping between the tags and bits so the application can predicate the hashing.

IV. System Implementation

We implement a prototype of TagMap reader by using a USRP N210 software defined radio equipped with SBX daughterboard and two directional antennas. The implementation is based on [11] and integrates TagMap's design into the EPC Gen2 protocol. The TagMap reader consists of the following components: USRP Source, Gate, Tag Decoder, Reader Encoder and USRP Sink.

- *USRP Source:* The first block is to acquire samples from the USRP hardware. In our experiment, the ADC is set to 2MS/s, resulting in 50 samples for each tag bit.
- *Gate:* The reader always provides continuous signals to tags, thus is full duplex (i.e., transmitting and receiving simultaneously). This block is used to track the `Query` and `Ack` commands, and trigger the processing of tags' replies.
- *Tag Decoder:* This block is to detect tags' `RN16` reply signals like preamble correlation, synchronization and channel estimation.
- *Reader Encoder:* This block is to implement the reader commands. Depending on the output of the tag decoder block, the reader create the next commands. Although this work only requires the commands of `Select` and `Query`, we implement all other Gen2 commands (i.e., `QueryRep`, `QueryAdust`, `Ack`), as well as the Q-adaptive algorithm.
- *USRP Sink:* This block is to propagate the reader command to the USRP hardware. The USRP transmits at 30 dBm at a frequency during $902 \sim 928$ MHz. The ADC rate is configured to 2 MS/s.

V. Evaluation

Our experiments are performed in our office lounge. A total of 1000 tags are densely attached to plastic panels [2]. The office section has a size of $3 \times 3 \times 2.5m^3$, as shown in Fig. 10. Many sofas and tables are in the lounge. To fully cover numerous tags, we use a 12 dBi large antenna. The transmitting power is fixed at 30 dBm. The USRP reader is connected to a 64-bit MacBook with 2.8GHz Intel Core i7 for running GNURadio.

[2]The deployment mode (like grid) does not affect the performance as long as each tag can be identified.

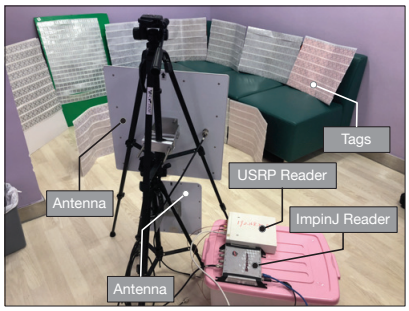


Fig. 10: Experimental scenario

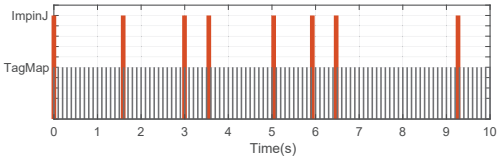


Fig. 11: Ten-second reading trace

Test Suite. We begin with a group of microbenchmark experiments to provide insights into the performance of TagMap with comparison with commercial readers. All `EPC`s are stored in database and known to upper applications. This assumption is reasonable and necessary because identifying if a tag is missing or under tracking without any prior knowledge of its existence is impossible. The tolerance of BF is set to 0.25 by default.

Baselines. To be fair to TagMap, we only focus on practical techniques without considering previous simulation-driven evaluation. We compare TagMap against two baseline schemes: a commercial reader (i.e., ImpinJ reader) [10] and Tash [7]. As the commercial reader collects all tags in the range, its inventory results are used as the ground truth.

A. Reading Rate

To better understand how TagMap reader improves the reading rate, Fig. 11 shows a 10-second reading trace of a tag. We totally deploy 140 tags in the surveillance region and randomly select one tag to show it reading trace. These tags are inventoried by an ImpinJ and a TagMap reader respectively. In the figure, the red (higher) and gray (lower) bars indicate the tag is identified by the ImpinJ and TagMap reader respectively. We can visually observe that the tag is much more frequently identified by TagMap reader than the ImpinJ reader. In particular, TagMap reader achieves a mean reading rate of 16Hz, outperforming ImpinJ (i.e., 0.8Hz) by $18\times$ higher. Such outperformance is achieved because TagMap reader removes the time-consuming processing for anti-collision. In addition, it can be seen that the tag is identified in a random manner by the ImpinJ reader but regularly by the TagMap reader. This is because ImpinJ reader's anti-collision processing uses a randomized algorithm (an ALOHA variant), while our approach is a deterministic algorithm (discuss later).

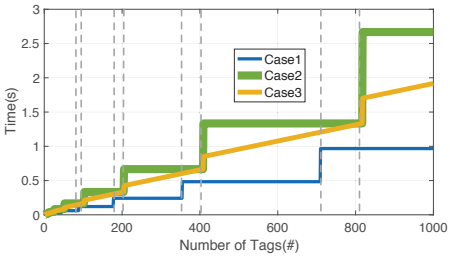


Fig. 12: Overheads across three cases

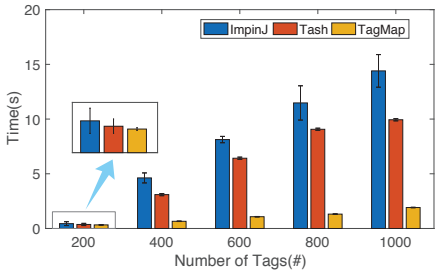


Fig. 13: Overheads across solutions

B. Acquisition Overhead

We are interested in evaluating the acquisition overhead as function of tag number (i.e., n). The overhead is defined as the time consumed in acquiring tags or a data structure like BF for representing them.

TagMap overhead across three cases. Although BF is a randomized data structure, the acquisition overhead is deterministic. The overhead depends on the bitmap size M and the number of hashes K. Let T_S, T_Q and T_{RN} denote the time consumed on transmitting the `Select`, `Query` and `RN16`. As shown in Fig. 3, each intercepted inventory takes $K\cdot(T_S+\tau_1)+T_Q+\tau_2+T_{RN}+\tau_3$ where τ_1, τ_2 and τ_3 are the constant spaces specified in Gen2. Thus, the total overhead C is given by

$$C = M\left(K\cdot(T_S+\tau_1)+T_Q+\tau_2+T_{RN}+\tau_3\right)$$

Thus, as long as two parameters are certain, the acquisition overhead of BF is a constant. This is why the tag is always regularly identified by TagMap reader in Fig. 11. Next, we measure the overhead with respect to the three cases discussed in §III-E. The results are shown in Fig. 12 and understood as follows:

• In the first case, the minimal size $\widehat{M} = \lceil n/\ln 2\rceil$ when keeping $K = 1$ (see Lemma 1) without using `modulo`. It seems that the overhead should be linearly proportional to n, i.e., $C \sim \lceil n/\ln 2\rceil$. Actually, the figure shows that the overhead increases step by step. This is because the hash value is represented in a binary form, we have to set the actual size $M = 2^{\lceil\log_2\widehat{M}\rceil} = 2^{\lceil\log_2(n/\ln 2)\rceil}$, making $C \sim 2^{\lceil\log_2(n/\ln 2)\rceil}$. We can exactly observe the turning points appearing at $n = \lceil 2\ln 2\rceil, \lceil 4\ln 2\rceil, \lceil 8\ln 2\rceil, \cdots = \ldots, 89, 178, 355, 719, \ldots$.

• The second case considers using multiple hash function to lower the false positives. The Lemmas suggest that $\widehat{M} = \lceil n\log_2 e\cdot\log_2(1/0.3)\rceil = \lceil 2.5n\rceil$ and $K = \lceil\ln 2\log_2(e)\log_2(1/0.25)\rceil = 2$. Thus, the actual size $M = 2^{\lceil\log_2(2.5n)\rceil}$. Similar to the first case, many steps are observed when $n = \lceil 2/2.5\rceil, \lceil 4/2.5\rceil, \lceil 8/2.5\rceil, \cdots = \ldots, 103, 205, 410, 820, \ldots$. Compared with the first case, this case consumes about double time. These the extra time is consumed on two `Selects` in each bit-driven intercepted inventory.

• The third case adopts `modulo` to fit the optimal size. Since this case adopts the same settings as that of the second case (i.e., $M = 2^{\lceil\log_2(2.5n)\rceil}$ and $K = 2$), the turning points are exactly observed as in the second case. However, we could observe that the time consumed in this case is far less than the second case. This is because the `modulo` moves the truncated bits ahead, reducing the total number of intercepted inventories. However, the overhead is still higher than the first case because two or four `Selects` must be broadcasted in each intercepted inventory. This case has a good linearity because it adopts the optimal size which is proportional to n. Hereafter, we use `modulo` by default unless it should be noted.

Overhead comparison to the-art-of-state. Next, we show the overhead of TagMap compared with ImpinJ reader and Tash. The results are shown in Fig. 13. From the figure, we obtain the following findings:

• Since no `Truncate`-supportable tags are available on the market, the improvement that Tash can make is very limited. Specifically, Tash can reduce the overhead by 20% on the average and achieve maximum overhead of 37% when $n = 1000$. Real outperformance is lower than that in the simulations shown in [7] due to the unavailability of the `Truncate` command.

• TagMap can reduce the overhead by 74.73% and 66.22% to ImpinJ and Tash on the average respectively. The reduction is almost linear to the number of tags. This is reasonable because more tags introduce more collisions. In particular, the outperformance is 86.69% and 83.96% when $n = 1000$. This superior outperformance is achieved mainly due to two unique techniques of TagMap: physical-layer acquisition (i.e., intercepted inventory) and being free of anti-collision. In addition, another distinct feature of TagMap is that its overhead is *unbiased*, i.e., the overhead is certain without fluctuation even facing numerous collisions. In contrast, both ImpinJ and Tash have about 10% dynamics in time because of the uncertain anti-collision processding.

VI. Case Study: Finding Missing Tags

Finally, we consider the effectiveness of TagMap in finding missing tags. We present the results of three cases, which are discussed in §III-E. We consider the application in view of the detection accuracy where the tolerance is set to 0.25. We wish to know the percentage of missing tags that are not found out, i.e., defined as *Error Rate*. Accuracy is evaluated as a function of the missing rate. In the experiment, a total of 1000 tags are placed in the surveillance region and 5%, 10%, 15% and 20% of tags are randomly taken away. We run 50 experimental trials and plot the detection accuracy in Fig. 14.

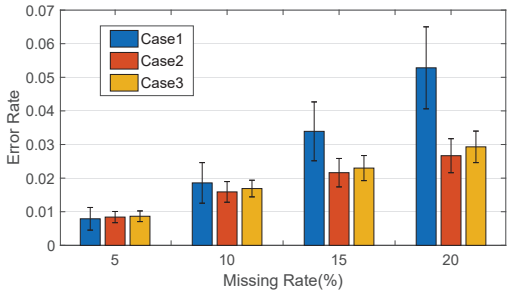


Fig. 14: Detection accuracy

- *Case 1: Using a single hash function*. From the figure, it is seen that $0.79\% \pm 0.3\%$ to $5.2\% \pm 1.2\%$ missing tags are falsely detected when the missing rate increases from 5% to 20%. All results are lower than the predefined tolerance of 25%. None of the false negatives are reported since the nature of the Bloom filter yields zero false negative.
- *Case 2: Using 3 hash functions*. Lemma 1 suggests the appropriate number of hash functions $k = 3$. In this case, we use 3 hash functions to create a 2^{12}-bit BF so that more '1's can be found in the bitmap to prove the presence of tags. This is confirmed by our tests: the error rate is reduced to $2.4\% \pm 0.5\%$ even the missing rate equals 20%, removing almost half (41.65%) false detections in Case 1.
- *Case 3: Using modulo operation.* We use the `modulo` operation to reduce the bitmap length from 4096 bits to 2396 bits. The accuracy for Case 3 is quite similar to that for Case 2 (e.g., $2.9\% \pm 0.4\%$ vs. $2.4\% \pm 0.5\%$ given missing rate of 20%) because both adopt same number of hash functions. The about 0.5% difference derives from the fact that the `modulo` operation disturbs the randomness of bitmap, i.e., moving trail bits to head bits.

Summary. Three acquisition overheads have been compared in Fig. 13 (i.e., the scenario when $n = 1000$), which implies that TagMap and an ImpinJ reader takes 1.9s and 15s to acquire a BF or identify 1000 tags respectively. In other words, TagMap can accurately find out $1 - (2.9\% \pm 0.4\%) = 97.1\% \pm 0.4\%$ of missing events within 2 seconds, whereas a commercial reader requires 15 seconds to know these events.

VII. Related Work

TagMap is related to previous work in three areas.

Design of Hash Function: Feldhofer and Rechberger [6] stated that current common hash functions (e.g., `MD5`, `SHA-1`, etc.), are not hardware friendly and are unsuitable for RFID tags, which possess a constrained computing capability. Such difficulty has spurred considerable research [6], [12]. Despite these optimized designs, the majority of them are still presented in theory and none is available for COTS RFID tags. Our work explores hash functions by leveraging selective reading to emulate equivalent hash functions.

Design of Communication Protocols. Plenty of work focus on designing communication protocols to improve the efficiency. Buzz [13] decodes tag collisions bit by bit. It assumes the linear combination of the *static* channel coefficients of reflecting tags independent of coexisting tags. BiGroup [14] recovers collisions without modifying COTS tags, but instead, changes the readers. However, these works either require modifying tags or work under specific conditions.

BF Applications: The traditional inventory interrogates all the tags, which makes it time-inefficient. These works [4], [15] proposed continuous reading protocols that can incrementally collect tags in each step by using a BF. TagMap provides a fundamental service to boost these works.

VIII. Conclusion

This work provides a fundamental service to today's commercial RFID Gen2 tags, which is dispensable for prior Bloom filter-based applications. A key innovation of this work is the physical-layer acquisition approach, which is a significant step in overcoming the performance bottleneck of current RFID systems. Our work provides new exciting avenues for exploration in RFID system and its applications.

Acknowledgments

The research is supported by NSFC General Program (NO. 61572282), UGC/ECS (NO. 25222917), Shenzhen Basic Research Schema (NO. JCYJ20170818104855702), and Alibaba Innovative Research Program.

References

[1] R. Das, "RFID Forecasts, Players and Opportunities 2018-2028: The complete analysis of the global RFID industry Brand new for 2018," https://www.idtechex.com/research/reports/rfid-forecasts-players-and-opportunities-2018-2028-000642.asp.

[2] D. M. Dobkin, *The RF in RFID: UHF RFID in Practice*. Newnes, 2012.

[3] D.-H. Shih, P.-L. Sun, D. C. Yen, and S.-M. Huang, "Taxonomy and survey of rfid anti-collision protocols," *Computer communications*, vol. 29, no. 11, pp. 2150–2166, 2006.

[4] B. Sheng, Q. Li, and W. Mao, "Efficient continuous scanning in rfid systems," in *Proc. of IEEE INFOCOM*, 2010.

[5] A. Poschmann, G. Leander, K. Schramm, and C. Paar, "New lightweight des variants suited for rfid applications," in *FSE*, vol. 4593, 2007, pp. 196–210.

[6] M. Feldhofer and C. Rechberger, "A case against currently used hash functions in rfid protocols," in *On the move to meaningful internet systems 2006: OTM 2006 workshops*. Springer, 2006, pp. 372–381.

[7] L. Yang, Q. Lin, C. Duan, and Z. An, "Analog on-tag hashing: Towards selective reading as hash primitives in gen2 rfid systems," in *MobiCom*. ACM, 2017, pp. 301–314.

[8] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2004.

[9] "EPCglobal Gen2 Specification," www.gs1.org/epcglobal, 2004.

[10] "ImpinJ, Inc," http://www.impinj.com/, 2017.

[11] N. Kargas, F. Mavromatis, and A. Bletsas, "Fully-coherent reader with commodity sdr for gen2 fm0 and computational rfid," *IEEE Wireless Communications Letters*, vol. 4, no. 6, pp. 617–620, 2015.

[12] H. Yoshida, D. Watanabe, K. Okeya, J. Kitahara, H. Wu, Ö. Küçük, and B. Preneel, "Mame: A compression function with reduced hardware requirements," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2007, pp. 148–165.

[13] J. Wang, H. Hassanieh, D. Katabi, and P. Indyk, "Efficient and reliable low-power backscatter networks," in *Proc. of ACM SIGCOM*. ACM, 2012, pp. 61–72.

[14] J. Ou, M. Li, and Y. Zheng, "Come and be served: Parallel decoding for cots rfid tags," in *MobiCom*. ACM, 2015, pp. 500–511.

[15] L. Xie, Q. Li, X. Chen, S. Lu, and D. Chen, "Continuous scanning with mobile reader in rfid systems: An experimental study," in *Proc. of ACM MobiHoc*. ACM, 2013, pp. 11–20.