Modeling Object Flows from Distributed and Federated RFID Data Streams for Efficient Tracking and Tracing

Yanbo Wu, Quan Z. Sheng, *Member, IEEE,* Hong Shen, and Sherali Zeadally

Abstract—In the emerging environment of the Internet of Things (IoT), through the connection of billions of radio frequency identification (RFID) tags and sensors to the Internet, applications will generate an unprecedented number of transactions and amount of data that require novel approaches in RFID data stream processing and management. Unfortunately, it is difficult to maintain a distributed model without a shared directory or structured index. In this paper, we propose a fully distributed model for federated RFID data streams. This model combines two techniques namely, *Tilted Time Frame* and *Histogram* to represent the patterns of object flows. Our model is efficient in space and can be stored in main memory. The model is built on top of an unstructured P2P overlay. To reduce the overhead of distributed data acquisition, we further propose several algorithms that use a statistically minimum number of network calls to maintain the model. The scalability and efficiency of the proposed model are demonstrated through an extensive set of experiments.

Index Terms—Radio frequency identification, RFID data streams, Internet of Things, object flow pattern, scalability.

1 Introduction

Recent advances in wireless sensors, radio frequency identification (RFID) technologies, and Web services have led to the emergence of the "Internet of Things" (IoT), a global network where everyday objects such as buildings, sidewalks, and commodities are identifiable, readable, addressable, and even controllable via the Internet [12], [18], [23]. Such a ubiquitous network offers the capability of integrating the information from both the physical world and the virtual one, which not only affects the way how we live, but also creates tremendous business opportunities such as efficient supply chains, independent living of elderly persons, and improved environmental monitoring. For example, in supply chain management, RFID tags are affixed to individual product items, and companies install RFID readers at various locations on their premises to capture tag reading events. As the tagged items are transported across companies, trails of tag reading events are left behind. Products' movements can thus be precisely recorded and tracked.

The most important feature underpinning these opportunities is *traceability*, the ability to find the current and historical states of RFID-tagged objects [24]. For instance, we might want to find the source where a bottle of problematic medicine comes from. In addition, it is also helpful for the managers to understand the patterns of product flows (e.g., "how often does hospital $\mathcal{A}$ order medicine $\mathcal{B}$ from company $\mathcal{C}$?") in order to make appropriate decisions on stock management.

Through the connection of billions of tags and sensors to the Internet, applications will generate an unprecedented number of transactions and amounts of data that require novel approaches in RFID data stream processing and management [6], [16], [23]. Many models and techniques have been proposed recently [9], [13], [22], [23]. Unfortunately, most of them require a centralized processing unit (e.g., data warehouse) that has severe drawbacks such as scalability.

IBM's Theseos [1] is one of the first few architectures which addressed this challenge by enabling traceability applications to process complex queries across organizations in a completely distributed setting. It relies on a distributed data model for object movements by introducing two attributes, namely *sentTo* and *receivedFrom* to maintain the information on the movement path of an object. With this information, it is possible to minimize the number of nodes to be visited without flooding queries to all nodes in the network. Unfortunately, to obtain this information, Theseos requires a high synchronization with other enterprise data (e.g., billing or accounting information). This is impractical for many applications in IoT where such enterprise data may not be available. In a very recent work [26], [25], Wu et al. propose a generic approach to solve this problem. The approach is built on top of the DHT (Distributed Hash Table) overlay network [3]. In particular, an object and its latest status are indexed at its *gateway node* in the P2P network. Every time when the object moves from a source node to a destination node, the gateway node updates the object's status at the source and the destination nodes on its movement, thereby establishing the information on

- *Y. Wu is with Beijing Jiaotong University. E-mail: ybwu@bjtu.edu.cn*
- *Q.Z. Sheng and H. Shen are with the University of Adelaide.*
- *S. Zeadally is with the University of the District of Columbia.*

the movement path of the object. This approach however requires extra storage for indices, and is expensive in bandwidth usage.

Instead of maintaining the exact movements of objects, we propose in this paper a probabilistic model that maintains the *object flow patterns*. An object flow pattern is a function of time, which describes the volume/frequency of object movements at a specific time. Our goal is to extract and model the patterns of object flows from high-volume, highly dynamic RFID data streams in autonomous network environments such as IoT. With these patterns, the efficiency of distributed data processing and mining can be significantly improved. The requirements of large-scale applications such as frequent data updates, row level security, and data archiving can be nicely satisfied. It should be noted that these are challenging tasks because the movement of an object is implicit. It is impossible to acquire such information without querying other nodes in the network. Ideally, the model should be established with a limited number of network calls without flooding the whole network. Our contributions are summarized as follows:

- We propose a new model called *Tilted tIme Series of Histograms* (TISH) that combines two techniques, namely *Histogram* and *Tilted Time Frame* [10]. Essentially, TISH is a synopsis of the object flow. It represents the patterns of object flow between two nodes for a long history using limited memory. TISH does not require any kind of indices. It is highly efficient in storage and bandwidth.
- We develop a Peer-to-Peer (P2P) architecture and a set of algorithms to establish and maintain the TISH model. To avoid long delays and extra bandwidth usage caused by network queries, we further develop an algorithm to choose the neighbors which are most possible to have the quested information as the target of query rewriting to avoid unnecessary network traffic. To avoid data migration when the underlying P2P layer changes (nodes leave or join), we introduce a simple but effective data structure for maintaining network topology. Based on the TISH model and the P2P architecture, our proposed algorithm on processing item-level tracking and tracing queries, statistically keeps the number of network calls to a minimum.
- We validate our approach through extensive experiments on large datasets. Our results demonstrate the efficiency and scalability of the proposed model, the architecture and the algorithms.

The rest of this paper is organized as follows. We formally define the problems in Section 2. In Section 3, we introduce the architecture of a distributed RFID system which is built based on our proposed model. In Section 4 and Section 5, we describe the TISH model, and its related maintenance algorithms. Experimental results are presented in Section 6 and the related works are discussed in Section 7. Finally, Section 8 provides some concluding remarks.

2 Problem Definition

An RFID network is formed by a set of nodes $\mathcal{N}$: $n_1, n_2, \ldots, n_m$. A node is a place under consideration for tracking and other interactions between objects and the networked system. For example, in a supply chain management system, each participating party (e.g., manufacturer, distribution center, and retailer) can be treated as a node. Objects move along these nodes in the network. In this paper, we define that a connection exists from node n_i to n_j in the network as "there exist objects moved from n_i to n_j", where n_i and n_j are the *source* node and the *destination* node of the connection, respectively. An *active* connection is defined as the connection where "there exist objects moving in the current time window (cycle)". Each node may have zero, one or many inbound or outbound connections.

Each object has one and only one original node where it is first discovered in the network. For example, in an RFID-enabled supply chain network, the original node of an object in its movement path is normally the node where the object is manufactured. When an object o is scanned at a node n_j at time t_k, it leaves a record of (o, n_j, t_k) at n_j. Since RFID readers continuously scan the object, a stream of tuples for the same object are generated with increasing timestamps. Without loss of information[1], we simplify the stream at each individual node to a single record of (o, n_j, t_s, t_e), where t_s and t_e represents the time when object o is first and last seen at node n_j, respectively. The basic schema for an RFID record is $(Object, Node, Start, End)$. It is worth noting that, this is different than that of other trajectory data, such as GPS records, which is normally represented as $(Object, Time, Coordinates)$. The major difference is that the values in space dimension in RFID networks are discrete.

Since in our approach, RFID data is kept in the place where it is collected and each node only governs its own data, i.e., for the record set $\mathcal{R}$ at node n_i, $\forall r \in \mathcal{R}$, r.Node = n_i. The $Node$ column can be omitted in the raw schema, i.e., the schema of data stored at each node is simplified to $(Object, Start, End)$.

Tracking and tracing are two fundamental queries in traceability applications in IoT, which are defined as follows. The difference between tracking and tracing is that tracking finds latest status of an object, while tracing finds part of or the full history of an object. They are defined with SQL language as follows.

Definition 1 (*Tracking*). *Given an object o, tracking o means*:
SELECT r.Node, r.End FROM Record r
WHERE r.Object = o AND r.Start =
(SELECT MAX(Start) FROM Record WHERE Object = o); □

Definition 2 (*Tracing*). *Given an object o, and a time range $tstart$ and $tend$ tracing o means*:

1. It should be noted that tracing queries are not sensitive to the internal states of an object at a particular node.

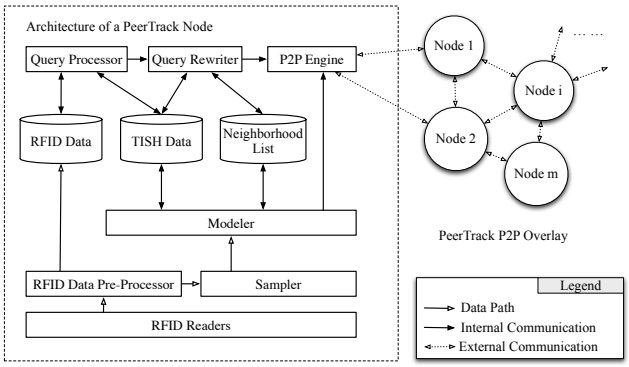


Fig. 1: PeerTrack Node Architecture

SELECT r.Node, r.Start, r.End FROM Record r
WHERE r.Object = o ORDER BY r.Start
AND r.Start >= tstart AND r.Start <= tend; □

The tracking queries aim to find the most recent location of the given object o. "Most recent" is defined as the record with the latest observation timestamp ($r.Start$). Tracing queries are interested in the full lifetime history of the given object, or the history in a particular time range.

Our goal is to build a distributed model which can be used as a middleware in a federated system. It does not require any centralized server for coordination, nor full access to data at other partners for its establishment and maintenance. The model is expected to be able to represent the pattern defined above for any time-and-node pair, and to be used to expedite distributed tracing and tracking query processing.

3 ARCHITECTURE

PeerTrack[2] is a federated system focusing on distributed RFID data processing and management. In PeerTrack, data is kept where it is collected and each node only governs its own data. Model establishment and query processing are done in a unstructured Peer-to-Peer (P2P) fashion.

The architecture of a PeerTrack node is depicted in the left part of Figure 1. The *Neighborhood List* contains the nodes to (or from) whom the data flow patterns are maintained by the local model. The neighborhood list consists of two sets of nodes: the source, and the destination nodes of n_i. For each source (resp., destination) node n_j, we maintain the pattern of object flow from (resp., to) n_j to (resp., from) n_i using the *Tilted tIme Series of Histogram* (TISH) model. The details of the TISH model and its maintenance will be described in Section 4 and Section 5.

The *TISH* model is updated repeatedly for every event cycle by the *Modeler*. The modeler takes a small random sampled data, which is generated by the *Sampler*, out of the large volume of RFID data as input. It analyzes the sample by querying the neighbors in the *Neighborhood List*, which is also maintained by the *Modeler*.

The *Query Processor* is responsible for answering queries from either local or remote users. If the query cannot be answered by the *RFID data* stored locally, the *Query Processor* exploits the information from the *TISH* model to find candidate nodes so that the query can be rewritten to query the corresponding RFID data stored at these nodes. The rewritten queries are then sent to other nodes through the *P2P Engine*, which is responsible for communicating with other nodes. With the *P2P Engine*, the nodes form an unstructured overlay.

In particular, when a tracing or tracking query is issued to the local PeerTrack node, for example, "where has object o been?", the query processor analyzes the query and recognizes it as a tracing query. It first contacts the local RFID database (*RFID Data* in Figure 1) to see whether object o has been observed locally. When it is, the query processor asks TISH model about the object flow patterns at the time when o is observed and the query rewriter will rewrite the query to nodes (neighbors) that are ranked top in the patterns. The rewritten queries will be sent to neighbors via the P2P engine. Since we have the IP of neighbors stored locally, the P2P engine will communicate with the neighbors directly, without going through the P2P overlay. The rewritten queries will be answered by related nodes and the result will be sent back. The query processor then aggregates the results and returns the trace of o to the querier.

This model is built by exploiting the fact that movements of objects are likely to be continuous and bulky in both time and space, so a connection which is active in the previous window is likely to be active in the current window. Based on this observation, when looking for the source node of an object, it is more efficient to first query the source nodes in previously active connections, which can be obtained by searching the neighbors. It is worth mentioning that the unstructured P2P query is much more expensive. We only use it when an object is from a node which is not in the neighborhood list.

This architecture adapts well with the characteristics discussed in Section 1 of the supplemental material:

- *Frequent Updates*. We sample the input and use only a small portion of incoming data to maintain the model. Thus the system scales well with frequent updates. Also, using the tilted time frame model, we can store a long history of object flow patterns in the main memory.
- *Row Level Security Requirement*. Since data is never stored at central servers, each node can have its own security schema. Each node owns the data physically and fully controls who can access which portion of its data. This model is strictly private, because there is no super user who can access data from every node.
- *Archiving*. Partners can archive their data whenever they deem appropriate, with flexible strategies. The archiving process is fast, because the records are stored in the order of time. To archive the records before a particular date, we only need to find the first record that is younger than the given criteria using a binary search, and move all the pages before it to the archive media.

2. http://cs.adelaide.edu.au/peertrack

- *Mining Efficiency*. The model maintained at each node contains the patterns for the object flow. It can be used as a starting point for online aggregation, materialized data cube and data visualization. While the TISH model at each node is limited to the local and recent neighbors, it is easy to gather the information from distributed nodes for higher level knowledge discovery, via P2P queries. It should be noted that distributed mining is one of our future research direction.

4 THE TISH MODEL

4.1 Basic Idea

In RFID applications, the object flow among nodes is determined by business actions and follows certain patterns. For example, a supplier sends products to a supermarket on a regular basis, such as once a week or once a month. Different products may (usually) have different patterns, so do the same products from different suppliers. If the patterns are known, we can use them to find out which supplier is most likely to be a source node from which a given object comes.

The problem of modeling these patterns can be categorized as *time-series data mining* [10]. A popular method is *regression analysis* in modeling time-series data and finding trends. However, we do not use this method because regression analysis often requires reading data for multiple times, which is not possible with RFID streams.

Our model is based on two important techniques in data mining: *Logarithmic Tilted Time Frame Model (LTTF)* and *Histogram*. As illustrated in Figure 2, the first (right most) slot represents the data for the time range of $\mathcal{T}_0$, while the i^{th} slot s_i represents a range of $2^i * \mathcal{T}_0$. Each time unit (slot) in LTTF occupies the same amount of memory, but the most recent time slot provides the statistics with the finest granularity, while the older ones are with coarser granularities.

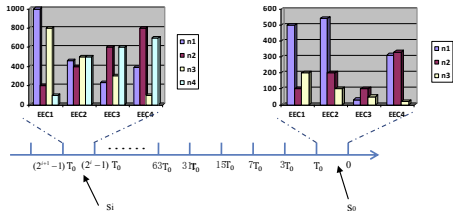


Fig. 2: TISH Model

We combine the two techniques and propose a new structure, namely *Tilted tIme Series of Histograms* (TISH). By combining the two techniques, it is possible to model the dynamics of RFID streams in both time and spatial dimensions. Tilted time series capture the changes of streams at different times, while histograms measure the distributions of streams from different nodes.

The basic idea of our model is that within each slot (i.e., $[(2^i - 1) * \mathcal{T}_0, (2^{i+1} - 1) * \mathcal{T}_0)$) in the tilted time frame model, histograms are used to summarize the object flow pattern for each business neighbor for the period of time represented by the slot. The height of each bar in the histogram represents the volume of object flow from/to a specific node. Using this model, we can calculate the probability of an object being from/to a specific neighbor at a given time, based on the statistics.

Symbol	Description
w_e	The width of an event cycle
s_i	The i^{th} slot in the tilted time frame
b_i	The i^{th} neighbor
h_{ij}	The histogram for b_j in slot s_i
f_i	The frequency of object flow for neighbor b_i in a new event cycle
w_s	The maximum number of Exponential Event Cycles (EECs) in the slots. It is a constant.
m	The size of the reservoir sample
n	The number of business neighbors
l	The number of slots in the model

Fig. 3: Symbols

The symbols used in the following discussion are summarized in Figure 3.

We use source (resp., destination) LTTF to record histories for the source (resp., destination) nodes. The time series are split into *Event Cycles* (also known as the *Sliding Windows*) of fixed time interval. All the slots manage a number of (at most w_s) units, which we name as *Exponential Event Cycles (EEC)*, because the unit in the i_{th} slot (s_i) manages the compressed data for 2^i event cycles (an example of a slot can be found in Section 2 of supplemental material). The most recent slot s_0 maintains the uncompressed event cycles, while the others maintain compressed ones. We denote the j_{th} EEC in the slot s_i as EEC_{ij}. It is easily inferred that $\mathcal{T}_0 = w_s * w_e$ and the time covered by slot s_i is $\mathcal{T}_i = 2^i * \mathcal{T}_0 = 2^i * w_s * w_e$. How to determine w_s and w_e can be found in Section 5 of the supplemental material.

4.2 Update for the Current Slot

At node n_i, we only have the local information of the object sent to n_i, such as arrival time and departure time. To get the information about the object's moving path, it is necessary to query its neighbors. In the worst case, the underlying unstructured P2P overlay is used to locate the object. Because P2P queries are more expensive than direct communications, we need to avoid such a case as much as possible. In addition, due to the large volumes of RFID data in the stream, we cannot afford building the model using all the data. Instead, we use a sample of the original data in each event cycle for the stream. The reservoir sampling algorithm [21] makes only one pass over the data set without knowing its size beforehand. So it is well suited for data streaming sampling. Its output is a uniform sample of the given data set.

After the data, which are collected during an event cycle, have been preprocessed and sampled, the sample is sent to the modeler for local update. We use the P2P tracking and tracing algorithm which we will introduce in Section 5. We

Algorithm 1 : Update the Model: *update*

Input: Neighbor set $\mathcal{B} = \{b_1, b_2, \ldots, b_n\}$
Corresponding frequency set $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$
Output: The refined model $\mathcal{M}$

1: **for** b_i in $\mathcal{B}$
2: Gets its histogram $h_{0i} \leftarrow s_0[b_i]$
3: **if** h_{0i} is **nil**
4: $H_{0i} \leftarrow$ new array with size w_s, $s_0[b_i] \leftarrow h_{0i}$
5: **end if**
6: $h_{0i}[s_0.size + 1] = f_i$
7: **end for**
8: $s_0.size \leftarrow s_0.size + 1$
9: **if** s_0 is full
10: merge($s_0, \mathcal{M}$)
11: clear s_0
12: **end if**

Fig. 4: Algorithm to Update the LTTF Model

call the information of source and destination nodes for all the objects in the sample as *Flow Synopsis*. This synopsis is then added to the most recent slot (s_0). If the slot is full, it is moved to the slot before it (s_1). Otherwise, s_1 is summarized and merged into s_2, and then s_0 is compressed and stored in s_1. If s_2 is also full, the summarization and merging process repeats until we find a non-full slot, or we reach the end of the LTTF. In the latter case, a new slot is created and appended to the tail. Note that s_i is only merged when it is full thus not all the boundaries in LTTF are updated. This does not introduce any problem because s_i is still the summary for the time frame of $[(2^i - 1) * \mathcal{T}_0, (2^{i+1} - 1) * \mathcal{T}_0)$ backwards from now.

The algorithm for updating the model is described in Figure 4. First we add the synopsis from the new event cycle to the slot (line 1–7). If a new neighbor joins, a new entry is inserted into the hash table (line 4). If an existing neighbor did not send anything, it is set to zero (this is not shown in the figure).

If the new event cycle fills the most recent slot s_0, all slots s_i are merged if necessary (the *merge* function in line 10, which is introduced in Section 3 of the supplemental material). s_0 is then cleared to be ready for the coming event cycles (line 11).

5 P2P Tracking and Tracing

The TISH model and its maintenance have been introduced in Section 4. In this section, we answer the unresolved question "how the model finds the source and destination node of an object?" by introducing the P2P tracking and tracing algorithm.

5.1 Tracing and Tracking Objects

With a centralized setting, answering tracing queries is easy. However due to privacy and performance issues, it is impractical to use in real applications. In a fully distributed, federated environment, our model avoids using Discovery-Service-like index or flooding the whole network. The idea is to utilize the history maintained in the model to rewrite the query to the node which has the most possibility to be the source of the requested object. The tracing algorithm is

Algorithm 3: Trace an Object: $trace$

Input: The object to trace o
The query initiating node n
Output: A list of nodes that o has been, sorted by time

1. locate any node that has had o using P2P overlay
2: $t_{start} \leftarrow$ select *Start* from Record where Id=o
3: **if** t_{start} is nil, return
4: $s \leftarrow$ the index of slot which covers t_{start}
5: $\mathcal{B} \leftarrow$ the set the neighbors in slot s and adjacent slots
// adjacent slots : c_1 to the recent and c_2 to the distant
6: $\mathcal{P} \leftarrow$ an array of size $\mathcal{B}$
7: **for** each neighbor b_i in $\mathcal{B}$
8: $\mathcal{P}[i] \leftarrow p_1(b_i, s)$ // Equation 1
9: **end for**
10: $sort(\mathcal{P}, \mathcal{B})$ // sort $\mathcal{B}$ in descending order according to $\mathcal{P}$ values
11: sequentially ask all the nodes in $\mathcal{B}$ that whether it is source or destination node of o
12: upon receiving the query, b_i repeats 2–11 on itself if it had o otherwise, return a negative result.
13: b_i then sends message to n confirming the appearance of o with extra information including arrival/leaving timestamps, and application-specific data

Fig. 5: Algorithm to Trace an Object

defined in Figure 5. The tracking algorithm is almost the same except the direction is reversed to tracing, and instead of retrieving all the nodes on the object's moving path, only the latest one is retrieved.

The key idea is to query the neighbors about the object(s) in the order of the probability calculated according to Equation 1 where $p_1(b_j, s)$ is the probability of the given object coming from neighbor b_j, s is the slot covering the given time t, n is number of neighbors, c_1 and c_2 are two constants that we use to control the range of past and future data under consideration, respectively. It should be noted that using more history data (i.e., larger c_1 and c_2) does not increase the accuracy. This is because the flow patterns are unknown and may frequently vary, and as a result using more history data may add more bias.

$$p_1(b_j, s) = \frac{\sum_{i=min(s-c_1,0)}^{max(s+c_2,l)} \left(\frac{1}{2^{|i-s|}} * \frac{\sum_{k=1}^{w_s} h_{i,j}[k]}{\sum_{l=1}^{n}\sum_{k=1}^{w_s} h_{i,l}[k]}\right)}{\sum_{i=min(s-c_1,0)}^{max(s+c_2,l)} 2^{|i-s|}} \quad (1)$$

It is easy to get s from the requested time t, because

$$s_0.size + w_s * \sum_{i=1}^{s-1} 2^{i-1} \leq t < s_0.size + w_s * \sum_{i=1}^{s} 2^{i-1} \quad (2)$$

Thus,

$$s = \lfloor \log_2 \frac{t - s_0.size}{w_s} + 1 \rfloor \quad (3)$$

The calculation of p_1 only involves a very small portion of the data, thus it is efficient. We will also prove with extensive experiments in Section 6 that it is accurate.

As shown in Figure 5, the neighbors are sorted (line 10) by the probabilities calculated using Equation 1. Then the query is redirected to the neighbor with the highest probability (line 11). If the neighbor does not return the positive result, the second possible neighbor is queried, and so on. Note the query contains the timestamp returned from

previous queries, and the neighbor only returns positive result if the timestamp is earlier than the one in the query. Otherwise, an infinite loop may happen if the object visited a node more than once.

5.2 Building the Flow Synopsis

After the data within an event cycle is sampled, we need to ask the neighbors to get the source and destination nodes. Instead of flooding the network, the history of object flow (i.e., the histogram) is used to find the most possible neighbors who may be the source nodes. Assuming that the object flow pattern changes smoothly, we choose the recent slots in the model to compute the probability of a neighbor being the source node for the objects. Instead of only using the most recent slot, we introduce a zipf-weighted method to compute the probability with several recent slots. This mechanism is introduced to smooth the data flow in case that there are some peak moments for a neighbor. Equation 4 shows how the probability is computed. It is easy to see that $p_2(b_j)$ is a variant of $p_1(b_j, s)$ that we introduced in Section 5.1.

$$p_2(b_j) = \frac{1}{\sum_{i=0}^{c} 2^i} * \sum_{i=0}^{c} (\frac{1}{2^i} * \frac{\sum_{k=1}^{w_s} h_{i,j}[k]}{\sum_{l=1}^{n} \sum_{k=1}^{w_s} h_{i,l}[k]}) \quad (4)$$

Where c is a configurable constant (similar to c_2), representing the number of slots under consideration, and n is the number of business neighbors. The factor $\sum_{i=0}^{c} 2^i$ is for normalization. $\sum_{l=1}^{n} \sum_{k=1}^{w_s} h_{i,l}[k]$ calculates the total number of objects for a slot, while $\sum_{k=1}^{w_s} h_{i,j}[k]$ is the number of objects from b_j. The factor $\frac{1}{2^i}$ assigns weights to different slots. The most recent slot has the highest weight of 1, and the weight decrease exponentially. In implementation, the sum of frequencies for all nodes in a slot ($\sum_{l=1}^{n} \sum_{k=1}^{w_s} h_{i,l}[k]$) can be calculated and cached.

Essentially, flow synopsis building is a simplified version of tracing. The differences between the two are:

1) Flow synopsis building happens on a node which must have the objects, whereas tracing does not have this advantage so it has to invoke the underlying P2P layer to first locate a node having had the object being traced.
2) Flow synopsis building happens when objects have arrived at a new node. So there is no future information. When calculating the probability, there are no future slots. In contrast, when answering tracing queries, future data has to be included.
3) There is no time or slot parameter for p_2, because p_2 is used to calculate the probability of a neighbor being the source node for objects in the *most recent* slot, i.e., the implicit time parameter is "now".

With Equation 4, the neighbors can be sorted by probability p_2. We can first try to contact the neighbor with the highest probability. If there are objects without source node after this query, the neighbor ranked the second is queried. The process repeats until we find the source nodes for all objects in the sample, or we have tried all the neighbors.

Algorithm 4 : Building Flow Synopsis

Input: A set of sample objects $\mathcal{O} = \{o_1, o_2, \ldots, o_m\}$
A set of neighbors $\mathcal{B} = \{b_1, b_2, \ldots, b_n\}$
Number of objects in the unsampled event cycle: x
Output: The updated sender LTTF model

```
1:  P ← an array of size n
2:  for each neighbor b_i in B
3:      P[i] ← p_2(b_i) // Equation 4
4:  sort(P, B) // sort B in descending order according to P
5:  F ← a map from b_i to its frequency
6:  for each neighbor b_i in B
7:      result set R ← query(b_i, O)
8:      F(b_i) ← R.size/m * x
9:      O ← O − R
10:     if O = Φ,        break
11: end for
12: if O ≠ Φ
13:     for each object o_i in O
14:       b ← P2POverlay.locate(o_i)
15:       if b exists in M, F(b) ← F(b) + x/m
16:       else F(b) = x/m;  B.append(b)
17:     end for
18: end if
19: update(B, F)
```

Fig. 6: Algorithm to Gather the Source Node Information

For the latter case, if there are still objects without source node, we will have to rely on the P2P overlay to locate the object.

Figure 6 shows the details of the algorithm to gather the source node information for the TISH model. First the probabilities $\mathcal{P}$ for all the neighbors are calculated and the neighborhood list is sorted according to the probabilities (line 1–4). Then we query the neighbors for the list of objects with unknown sources, in descending order of sorted probabilities (line 6–11). After querying the neighbors, if there are still objects with unknown sources, we have to use underlying P2P overlay to find the sources of the objects (line 12–18). Finally, the neighbors and the corresponding frequencies are used to update the TISH model (line 19).

In real applications, nodes may leave the network without notifying neighbors. This may cause some problems. We discuss in detail on how to deal with these problems in Section 4 of the supplemental material.

6 Experimental Evaluation

We conducted extensive experiments to evaluate the performance of the proposed approach[3]. This section focuses on reporting six experimental results i) to demonstrate the accuracy of modeling the object flow, ii) to evaluate the performance of the model maintenance, and iii) to prove the scalability of the system built on top of the TISH model. Additional experiments can be found in Section 7 of the supplemental material.

6.1 Experimental Setup

Experiments were conducted on a Core 2 Quad 2.40GHz machine with 4GB RAM. We simulated a network with 1,000 nodes. This network is built with the following

3. The performance of our approach has been analyzed theoretically in Section 6 of the supplemental material.

Parameter	Default Value
Fanout	10
Number of Slots	10
Value of V, c and w_s	1000, 2, 10
Size of Sample (m)	50

Fig. 7: Default Settings

characteristics: i) the network overlay is a connected uni-directional graph; ii) there are no hot or cold spots in the network; and iii) the fanout of the nodes follows normal distribution with a given average (see Figure 7) and variance (0.01). The first characteristic guarantees that every node is involved in the experiments, so there are no outliers. The last two help to keep the variance of calculating averages low. The edges in the graph represent a business connection, along which objects move.

All nodes in the network have $\mathbf{V}$ objects of their own at the beginning of the experiments. Each connection is associated with a time-varying pattern from a pre-defined pattern set (see Table 8). All the nodes send objects to neighbors with the amount determined by the associated pattern. These patterns vary in the volume of object flow for time t (i.e., $g(t)$) where $\mathbf{V}$ is a constant coefficient and $random()$ returns a number which is within $(0, 1]$. If there are less objects than $g(t)$, the node generates enough objects to make up for the objects. Parameters a and b in the patterns are random numbers between 0 and 1 (inclusive). They are chosen before the experiments and remain constant during the experiments.

For each epoch, a node randomly chooses half of its connections to send objects. The connections with/without objects moving on inside an epoch are called *active connections/idle connections*, respectively.

An RFID object generator has been implemented to generate RFID objects for each pattern. The default settings for the experiments are listed in Figure 7.

Pattern Name	Definition ($g(t)$)
Constant	$a * \mathbf{V}$ (a is chosen randomly)
Random	$\mathbf{V} * random()$
Segmentary	$a * \mathbf{V}$, if $2 * seg \leq t < 2 * seg + 1$ $b * \mathbf{V}$, if $2 * seg + 1 \leq t < (2 + 1) * seg$ seg is 100 Event Cycles a, b are chosen randomly
Sinusoidal	$\lvert\mathbf{V} * \sin(t)\rvert$

Fig. 8: Patterns

6.2 The Accuracy of Modeling

This experiment evaluated the accuracy of TISH. Since the TISH model keeps more information on recent data, we expect that the accuracy decreases for the distant data. However, the accuracy loss should not be significant. The error for the TISH model in a given time frame is defined as the difference between the real and the modeled distribution of objects' source nodes. To accurately represent this, we first calculated the average of δ (described in Section 6 of the supplemental material) as $\bar{\delta}$ for all neighbors at each node. We then calculated the error of the model as the average of $\bar{\delta}$ for all nodes in the system.

We ran the simulation using the default settings in Figure 7 for 1000 event cycles. During each cycle (called an *epoch*), several objects were sent from one node to another according to the pattern associated with this connection, if it is active. The objects are randomly chosen from local objects, including the ones initially assigned to a node and those sent by its neighbors. The experiments were done for each pattern separately (i.e., all connections are associated with the same pattern) and all patterns together (i.e., each connection is associated with a randomly chosen pattern). In this experiment, after the 1000 event cycles finished, we calculated the error ϵ, for each epoch.

The result is shown in Figure 9 and Figure 10. The time axis represents the epochs from the most recent ($1st$) to the most distant ($1000th$).

Figure 9 shows the error of the TISH model in experiments that only a single pattern is chosen for all nodes in the network[4]. We note that the error is very low. Although it increases for distant epochs, the increasing rate is not high. In theory, "Constant Pattern" should not generate any error because the distribution of objects are never changed. In practice, the sampling process and idle epochs add randomness to the system. For "Sinusoidal Pattern", although the volume changes, the distribution does not. So the orders of $\mathcal{B}$ and $\mathcal{B}'$ are not changed. This explains why it shows almost exactly same results with the "Constant Pattern". "Segmentary Pattern" shows an periodic pattern where the error increases every 100 epochs. This is caused by the change of patterns described in Figure 8. However, after the change finished, the error quickly decreases to almost the same with "Constant Pattern".

The impact of randomness on the accuracy of the TISH model is important. We ran the simulation with "Random Pattern" (all connections are associated with "Random Pattern"), reusing the settings in the experiments without "Random Pattern". We also ran the simulations for the scenarios of "Mixed" (each node randomly picked a pattern from Figure 8 individually) and "Mixed Without Random" (each node randomly picked a pattern from Figure 8 without "Random Pattern"). For "Mixed" or "Mixed Without Random", connections are associated with different patterns. Figure 10 also includes results for "Random Pattern", "Mixed Without Random" and "Mixed". The "Mixed Without Random" experiment shows that even when the nodes in the network choose different patterns for different connections, the TISH model is still able to describe them accurately. Compared to the "Constant Pattern", the extra error for the "Mixed Without Random" experiment is caused by the mixing "Segmentary Pattern"

4. "Random Pattern" is not included, as "Random" is actually not a pattern at all.

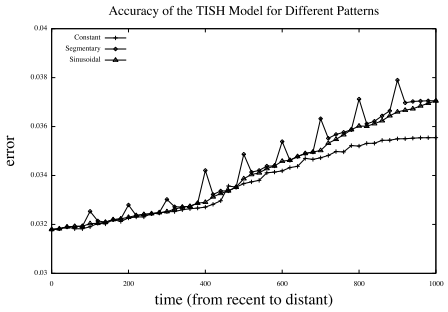


Fig. 9: Accuracy of the Model for Different Patterns

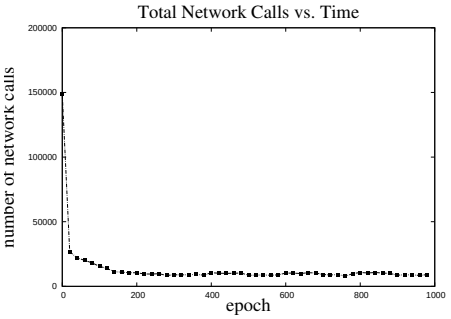


Fig. 11: Number of Network Calls vs. Time

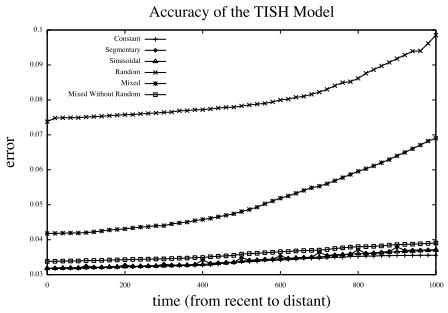


Fig. 10: Accuracy of the Model with Mixed and Random Pattern

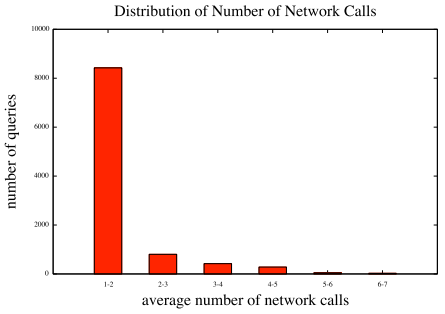


Fig. 12: Distribution of Number of Network Calls for Query Processing

and "Sinusoidal Pattern". Since the change in "Sinusoidal Pattern" is continuous, at some point (when $\sin(t) = a$ or $\sin(t) = b$), the order of $\mathcal{B}$ is changed.

"Random Pattern" generates a high error because it is not a pattern and is unpredictable. Mixing it with other patterns also increases the average error.

6.3 Network Traffic Cost

The main performance bottleneck in our model is caused by the procedure of querying neighbors for new incoming objects. It is also possible that when the new patterns are being established, more network calls are used due to the use of the underlying P2P overlay. However, as discussed in Section 5, our model is sensitive to these kinds of network changes and adapts quickly with the changes.

In this experiment, we verified the adaptation of the model by counting the number of network calls at different time points. The system setting is the same as the "Mixed without Random". Figure 11 shows the result. We can see that during the time of system bootstrap, the network traffic is higher. This is because at that time there was no history information and all the objects were found by P2P calls. However, after the TISH model has been established, only a few network calls are used and the number of network calls stays stable.

6.4 Query Processing Cost

In this experiment, we executed 10,000 trace queries on an established model (object movements were stopped after 1,000 epochs) to see how efficient the query processing is, using the "Mixed without Random" setting. All traces were initiated from a node on the moving path of the queried object. We calculated the average number of network calls for the discovery of each path segment, by dividing the total number of network calls used for each query by the length of the moving path for the object being queried.

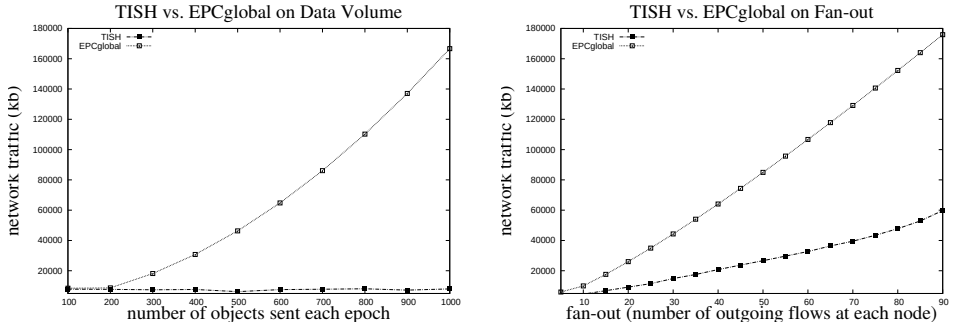
The distribution of the average number of network calls used for each path segment is shown in Figure 12. We note that most queries used 1 to 2 network calls on average for each path segment. The average number of network calls used for each path segment for all the queries is 1.27. We can therefore conclude that the model is efficient in supporting tracing queries.

6.5 The Scalability

In this experiment, we want to see how scalable our TISH model is against i) variation on volumes of object flows and ii) variation on network topologies. We exploited the number of network queries used for maintaining the model as the measurement of scalability. We compare TISH with our implementation of EPCglobal architecture[5] under the same experimental settings. EPCglobal developed a *Discovery Service* (DS) standard which is used to trace individual items. To enable the traceability, partners have to register all the objects to the service.

Data Volume: In this test, we examined the impact of the data volume on the performance of the model by comparing the total size of network traffic with the EPCglobal

5. http://www.epcglobalinc.org



(a) Scalability against Data Volume (b) Scalability against Network Topology

Fig. 13: Network Traffic Cost and Scalability

architecture implementation. We controlled the volume of object flows by varying the value of $\mathbf{V}$ (Figure 7) from 100 to 1000. The result is shown in Figure 13a. As we can see, in the case of the TISH model, the network traffic is fairly constant compared to the cost of EPCglobal architecture which increases quickly with the number of objects sent in each epoch. This is because the TISH model samples the input and only queries the objects in the sampled set. Nodes in the EPCglobal architecture need to send information about each object to the Discovery Service.

Network Topology: In this test, we investigated the performance of our model on different network topologies. Using default settings, we varied the maximum fanout to change the connectivity of the network. The maximum fanout varies from 5 to 90. Figure 13b shows that the cost increases for both the TISH model and the EPCglobal architecture when the fanout increases. However, the TISH model increases much slower than the EPCglobal architecture. More fan-outs cause more business neighbors at each node, and according to our experiment settings, more objects are sent and received at each node. However, the cost of the TISH model increases slowly because it samples the input and chooses the most possible neighbors first to query. In contrast, the EPCglobal architecture sends the information about all the objects to the DS.

7 Related Work

In [6] and [8], the fundamental problems in RFID data management and query processing are discussed. One of the important topics lies in how to develop an efficient model to infer the implicit business knowledge from large volumes and distributed RFID data streams. In this section, we review the major techniques that are most closely related to our proposed approach.

EPCglobal[6] is an organization focused on developing standards to support RFID in information rich trading networks. It has developed a *Discovery Service* standard which is used to trace individual items. To enable the traceability, partners have to register all the objects to the service. This architecture is not fully distributed and scalable.

6. http://www.epcglobalinc.org

The authors of [5] extend their work on SPIRE [15] to adapt to large-scale RFID networks. The location and containment relationships are inferred in a distributed way. [1] proposes a pure distributed RFID data model. Two attributes *sentTo* and *receivedFrom* are associated with each object. The distributed path is formed by records in correlated nodes. However, this work does not solve the problem on how to acquire these attributes. Sheng et al. solve this problem by using a DHT-based architecture [17]. This work requires every item to be indexed in the network which makes the approach costly. This same effort is further extended in [26] by introducing a model which indexes the objects in a structured P2P network and algorithms to maintain the model. However, this model supports item-level and aggregation traceability queries at the cost of indexing spaces.

Query processing in a P2P environment is essentially searching for the proper resource to answer the query. The general P2P architectures, such as [14], [4] and [19], can be applied. However, RFID records have implicit knowledge about the distribution of objects, which can direct the search in a more efficient fashion than the general methods. In [11], Jinoh et al. propose the notation of accessibility to capture both availability and performance as a measurement in node selection. However, this work is still too generic to consider the data itself as a reference. Most existing *Content-Aware* P2P systems, such as [7] and [20], focus on efficient replication of the data to increase its availability based on the content of data. They are not feasible in processing RFID data streams because the partners require sovereignty of the data. In addition, compared to texts or multimedia resources, replicating RFID data is often unnecessary since only a very small portion of them is going to be queried. In a very recent work in [2], the authors propose a framework for RFID-based inter-organizational cooperation. This work includes a cooperative, complex event processing method, which is based on event notification services.

8 Conclusion

Recent advances in technologies such as radio-frequency identification (RFID) make automatic tracking and tracing possible in a wide range of applications. Unfortunately,

realizing traceability applications in large-scale, distributed environments such as the emerging Internet of Things (IoT) presents significant challenges due to their unique characteristics such as large volume of data and sovereignty of the participants. In this paper, we have introduced a distributed model for sovereign RFID data streams by combining the techniques of *Titled Time Frame* and *Histogram*. We developed distributed algorithms to establish and maintain the model. Our proposed model and algorithms are scalable and efficient. We demonstrated the usefulness of this model in processing tracking and tracing queries. Extensive experimental results showed the viability, efficiency, and scalability of our proposed techniques.

Ongoing work includes further performance evaluation with real data from a large-scale supply chain management system. In this paper, we assume that the preprocessed RFID data is clean and complete. In reality, this is hardly true. The noisy, incomplete data introduces *uncertainties* into the model. This is another challenging research problem for our future work.

ACKNOWLEDGMENT

Yanbo Wu's work has been supported by AFSI Scholarship from the University of Adelaide and Google Top-Up PhD Scholarship from Google. Quan Z. Sheng's work has been partially supported by Australian Research Council (ARC) Discovery Grant DP0878917 and Linkage Project LP100200114. This research was partially supported by National Science Foundation of China under its General Projects funding #61170232, Fundamental Research Funds for the Central Universities #2012JBZ017, National Key Laboratory Research Funds RCS2011ZT009. The authors would like to thank the anonymous reviewers for their valuable feedback on this work.

REFERENCES

[1] R. Agrawal, A. Cheung, K. Kailing, and S. Schönauer. Towards Traceability Across Sovereign, Distributed RFID Databases. In *Proceedings of the 10th Intl. Database Engineering and Applications Symposium (IDEAS'06)*, Delhi, India, December 2006.

[2] L. A. Amaral and F. Hessel. Cooperative CEP-based RFID Framework: A Notification Approach for Sharing Complex Business Events Among Organizations. In *Proceedings of the 5th IEEE International Conference on RFID (RFID'11)*, Orlando, Florida, April 2011.

[3] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking up Data in P2P Systems. *Communications of the ACM*, 46(2):43–48, 2003.

[4] J. D. K. Ben Y. Zhao and A. D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical report, Berkeley, CA, USA, 2001.

[5] Z. Cao, C. Sutton, Y. Diao, and P. Shenoy. Distributed Inference and Query Processing for RFID Tracking and Monitoring. *VLDB Endowment*, 4, February 2011.

[6] S. S. Chawathe, V. Krishnamurthy, S. Ramachandran, and S. Sarma. Managing RFID Data. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB'04)*, Toronto, Canada, September 2004.

[7] H. Chen, H. Jin, X. Luo, Y. Liu, T. Gu, K. Chen, and L. Ni. BloomCast: Efficient and Effective Full-Text Retrieval in Unstructured P2P Networks. *IEEE Transactions on Parallel and Distributed Systems*, 23(2):232–241, 2012.

[8] R. Derakhshan, M. Orlowska, and X. Li. RFID Data Management: Challenges and Opportunities. In *Proceedings of the 1st IEEE International Conference on RFID (RFID'07)*, Vienna, Austria, March 2007.

[9] H. Gonzalez, J. Han, X. Li, and D. Klabjan. Warehousing and Analyzing Massive RFID Data Sets. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*, Atlanta, Georgia, USA, April 2006.

[10] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Elsevier, 2006.

[11] J. Kim, A. Chandra, and J. Weissman. Using Data Accessibility for Resource Selection in Large-Scale Distributed Systems. *IEEE Transactions on Parallel and Distributed Systems*, 20(6):788 – 801, June 2009.

[12] N. Koshizuka and K. Sakamura. Ubiquitous ID: Standards for Ubiquitous Computing and the Internet of Things. *IEEE Pervasive Computing*, 9(4):98–101, 2010.

[13] C.-H. Lee and C.-W. Chung. Efficient Storage Scheme and Query Processing for Supply Chain Management Using RFID. In *Proceedings of the 2008 ACM International Conference on Management of Data (SIGMOD'08)*, Vancouver, Canada, 2008.

[14] Y. Liu, L. Xiao, and L. Ni. Building a Scalable Bipartite P2P Overlay Network. *IEEE Transactions on Parallel and Distributed Systems*, 18(9):1296–1306, 2007.

[15] Y. Nie, R. Cocci, Z. Cao, Y. Diao, and P. Shenoy. SPIRE: Efficient Data Interpretation and Compression over RFID Streams. *IEEE Transactions on Knowledge and Data Engineering*, 24(1):141–155, 2012.

[16] Q. Z. Sheng, X. Li, and S. Zeadally. Enabling Next-Generation RFID Applications: Solutions and Challenges. *IEEE Computer*, 41(9):21–28, September 2008.

[17] Q. Z. Sheng, Y. Wu, and D. Ranasinghe. Enabling Scalable RFID Traceability Networks. In *Proceedings of the 24th International Conference on Advanced Information Networking and Applications (AINA'10)*, Perth, Australia, April 2010.

[18] Q. Z. Sheng, S. Zeadally, Z. Luo, J.-Y. Chung, and Z. Maamar. Ubiquitous RFID: Where are We? *Information Systems Frontier*, 12(5):485–490, 2010.

[19] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, New York, NY, USA, 2001.

[20] J. M. Tirado, D. Higuero, F. Isaila, J. Carretero, and A. Iamnitchi. Affinity P2P: A Self-organizing Content-based Locality-aware Collaborative Peer-to-peer Network. *Computer Networks*, 54(12):2056 – 2070, 2010.

[21] J. S. Vitter. Random Sampling with a Reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, 1985.

[22] F. Wang, S. Liu, and P. Liu. A Temporal RFID Data Model for Querying Physical Objects. *Pervasive and Mobile Computing*, 6(3):382–397, 2010.

[23] E. Welbourne, L. Battle, G. Cole, K. Gould, K. Rector, S. Raymer, M. Balazinska, and G. Borriello. Building the Internet of Things Using RFID: The RFID Ecosystem Experience. *IEEE Internet Computing*, 13(3):48–55, May/June 2009.

[24] Y. Wu, D. Ranasinghe, Q. Z. Sheng, S. Zeadally, and J. Yu. RFID Traceability Networks: A Survey. *Distributed and Parallel Databases*, 29(5):397–443, 2011.

[25] Y. Wu, Q. Z. Sheng, and D. Ranasinghe. Facilitating Efficient Object Tracking in Large-Scale Traceability Networks. *The Computer Journal (Oxford)*, 54(12):2053–2071, 2011.

[26] Y. Wu, Q. Z. Sheng, and D. Ranasinghe. Peer-to-Peer Objects Tracking in the Internet of Things. In *Proceedings of the 40th International Conference on Parallel Processing (ICPP'11)*, Taipei, Taiwan, 2011.

Yanbo Wu received the PhD degree in computer science from the University of Adelaide, Australia. He is a lecturer in the School of Computer Science and Information Technology at Beijing Jiaotong University, China. His research interests include Internet of Things, distributed database and mobile computing. He has published papers in various peer-reviewed journals, including Distributed and Parallel Databases and The Computer Journal (Oxford). He is the recipient of Google PhD Top-Up Grant in 2011.

Quan Z. Sheng received the PhD degree in computer science from the University of New South Wales, Sydney, Australia. He is a senior lecturer in the School of Computer Science at the University of Adelaide. His research interests include service-oriented architectures, web of things, distributed computing, and pervasive computing. He was the recipient of the 2011 Chris Wallace Award for Outstanding Research Contribution and the 2003 Microsoft Research Fellowship. He is the author of more than 100 publications. He is a member of the IEEE and the ACM.

Hong Shen is Professor (Chair) of Computer Science in the University of Adelaide, Australia. He was Professor and Chair of the Computer Networks Laboratory in Japan Advanced Institute of Science and Technology during 2001-2006, and Chair of Compute Science at Griffith University, Australia, where he taught 9 years since 1992. He has published more than 300 papers including over 100 papers in international journals such as a variety of IEEE and ACM transactions, on parallel and distributed computing, networks, algorithms, data mining and privacy preserving computing. He received many honours/awards and served on the editorial boards of numerous journals.

Sherali Zeadally received his Bachelor degree in Computer Science from the University of Cambridge, England, and the Doctoral degree in Computer Science from University of Buckingham, England, in 1996. He is currently an Associate Professor at the University of the District of Columbia. He currently serves on the Editorial Boards of 15 peer-reviewed international journals. He has served as a Guest Editor for over a dozen special issues of various peer-reviewed scholarly journals. He is a Fellow of the British Computer Society and a Fellow of the Institution of Engineering Technology, UK. His research interests include computer networks including wired and wireless networks, network/system/cyber security, mobile computing, ubiquitous computing, RFID, multimedia, and performance evaluation of systems and networks.