Data or index: a trade-off in mobile delay tolerant networks

Hong Yao, Han Zhang, Changkai Zhang and Deze Zeng

Hubei Key Laboratory of Intelligent Geo-Information Processing,
School of Computer Science,
China University of Geoscience,
Wuhan, Hubei, 430074, China
Email: yaohong@cug.edu.cn
Email: sylarjohn@gmail.com
Email: cug09evan@gmail.com
Email: dazzae@gmail.com

Jie Wu* and Huanyang Zheng

Department of Computer and Information Sciences,
Temple University,
Philadelphia, PA 19122, USA
Email: jiewu@temple.edu
Email: huanyang.zheng@temple.edu
*Corresponding author

Abstract: Acquiring content through mobile networks is a basic and general topic. Mobile nodes have two different ways of obtaining data. The first method is to download data quickly through 3G/4G networks, which is expensive. The second way is to get data from other nodes by means of delay tolerant networks (DTN), which are much cheaper, but are time-consuming. Throwboxes deployed in DTN act as fixed ferry nodes. The index records the historical encounter information, in order to give the mobile nodes predictive abilities regarding future encounter events. We try to compare the effectiveness when we replace some space for the data to index. We bring forward an index-based buffer space management mechanism for throwboxes, by which mobile nodes can have the chance to fetch data at a lower total cost. Preliminary simulations demonstrate that the buffer space allocation strategy is affected by some system parameters, and that replacing some space for data with an index can lower the system total cost significantly in most cases. Simulation results also show that the index-based buffer space management mechanism outperforms other mechanisms which only store data items or hold an index of static size.

Keywords: mobile networks; delay tolerant networks; DTN; throwbox; index.

Reference to this paper should be made as follows: Yao, H., Zhang, H., Zhang, C., Zeng, D., Wu, J. and Zheng, H. (2017) 'Data or index: a trade-off in mobile delay tolerant networks', *Int. J. Computational Science and Engineering*, Vol. 14, No. 4, pp.330–340.

Biographical notes: Hong Yao received his PhD in Computer Architecture from the Huazhong University of Science and Technology, Wuhan, Hubei, China in 2010. He is currently an Associate Professor with the School of Computer Science, China University of Geosciences, Wuhan 430074, China. He is also with Hubei Key Laboratory of Intelligent Geo-Information Processing, China University of Geosciences, Wuhan 430074, China. His current research interests include computer networks and distributed systems.

Han Zhang received his Master degree in School of Computer Science, China University of Geosciences, Wuhan 430074, China in 2015. His current research interests include computer networks and distributed systems.

Changkai Zhang is pursing his MS in School of Computer Science at China University of Geosciences, Wuhan 430074, China. His current research interests include computer networks, and distributed systems.

Copyright © 2017 Inderscience Enterprises Ltd.

Deze Zeng received his PhD and MS in Computer Science from the University of Aizu, Aizu-Wakamatsu, Japan, in 2013 and 2009, respectively. He received his BS degree from the School of Computer Science and Technology, Huazhong University of Science and Technology, China in 2007. He is currently an Associate Professor with the School of Computer Science, China University of Geosciences, China. He is also with Hubei Key Laboratory of Intelligent Geo-Information Processing, China University of Geosciences, Wuhan 430074, China. His current research interests include: cloud computing, software-defined sensor networks, data centre networking, networking protocol design and analysis.

Jie Wu is the Chair and Laura H. Carnell Professor in the Department of Computer and Information Sciences at Temple University. Prior to joining Temple University, he was a Program Director at the National Science Foundation and Distinguished Professor at the Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. He regularly publishes in scholarly journals, conference proceedings and books.

Huanyang Zheng received his BEng in Telecommunication Engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2012. He is currently a PhD candidate in the Department of Computer and Information Sciences, Temple University, Philadelphia, Pennsylvania, USA. His current research focuses on mobile networks, social networks, and cloud systems.

1 Introduction

The rapid growth of all kinds of mobile devices leads to a mobile data explosion. According to Cisco Visual Networking Index (Cisco VNI) (http://www.cisco.com/), mobile data traffic will increase ten-fold between 2014 and 2019. Mobile data traffic will grow at a CAGR of 57% between 2014 and 2019, reaching 24.3 exabytes per month by 2019. Mobile data offloading seems to be the most promising solution at this moment. On the other hand, a lot of mobile data flows are not delay-sensitive, e.g., messaging, file transfer, and data dissemination. And, Lee et al. (2013) indicate that delayed transmissions can achieve substantial gains. In Mehmeti and Spyropoulos (2014), the authors further analyse the problem and give some expressions to choose the optimal deadline. In this paper, we explore a new way to offload the mobile data by using DTN as a collaborative entirety.

In Fall (2003), *delay tolerant networks* (DTN) performs the so-called store-carry-forward paradigm to deliver messages in an end-to-end fashion, since the mobile node has a high node mobility, a low cache capability, and a limited energy, sharing data between nodes may not be efficient enough. An alternative approach is to equip the DTN with dedicated fixed nodes, called *throwboxes* in Zhao et al. (2006), which are stationary wireless nodes with significantly improved storage and energy capabilities that simply act as fixed relays. Traditional offloading strategy and the deadline-driven mechanism make mobile nodes always try to wait fetching data from throwboxes until the deadline. For some data request which can hardly be fetched before deadline, mobile nodes also have to wait till the deadline. A lot of waiting time is unnecessary and wasted.

To address this issue, we bring index into throwboxes. Index is a table file recording the historical contact information between mobile nodes and throwboxes. Throwboxes can use this knowledge to predict future contact event and give mobile nodes prediction about whether they can fetch the data from throwboxes. Many mobility models in Jeremie et al. (2006), Spyropoulos et al. (2006) and Ibrahim et al. (2007) prove that the contact event between throwboxes and mobile nodes is predictable. With the prediction, mobile nodes can make a wise choice to avoid the meaningless waiting. However, the added index file shares the limited buffer with the data. Some data space must be sacrificed for storing the index file. So, here comes the problem: is it worthy to add the index file into throwboxes although it may reduce the hit rate of users fetching data from throwboxes? Our initial motivation is to find out the effect of replacing some of the data space with the index file. We define that the *total cost* of the data fetching is formed by the time consumption and transmission cost. And how to balance data space and index file space, to achieve the minimised total cost, under different network conditions, is the objective of this work.

The key contributions of this paper are summarised as follows:

- We add an index file to the throwboxes, making throwboxes able to not only store data items, but also give mobile nodes suggestions about how to fetch the requested data in a min-cost way.
- We propose a novel future event prediction algorithm. Differing from the traditional approach, we set the data contact event as the prediction target instead of the mobile node's contact event.
- We present a index-based buffer allocation mechanism to balance the data file space and index file space to achieve the minimise total cost.
- We conduct extensive simulations to evaluate the index-based mechanism. The results clearly show that the index-based buffer allocation mechanism

significantly outperforms the traditional data-only mechanism.

The remainder of this paper is organised as follows. We introduce the system model in Section 2, and then we introduce the data encounter prediction approach in Section 3. The buffer space allocation mechanism is presented in Section 4. Simulation results which are presented in Section 5 prove our theory. Finally, we review the related work in Section 6 and conclude the paper in Section 7.

2 System model

In this section, we introduce the system model, including network model, utility model and delivery model.

2.1 Network model

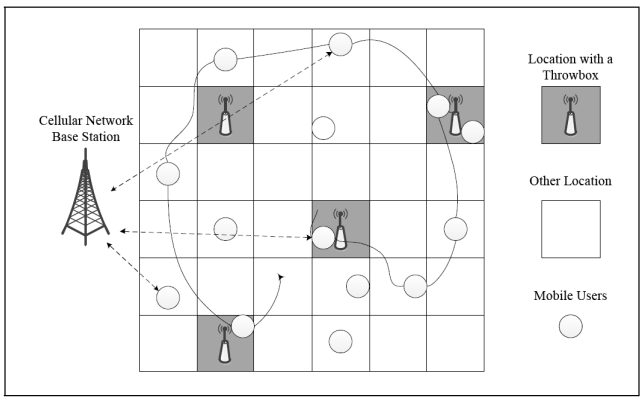
We consider a mobile network with a node set $N = N_t \cup N_n$, where N_t and N_n donate the set of throwboxes and mobile nodes, respectively. All the mobile nodes independently and randomly move on a two-dimensional plane. Throwboxes are distributed on some spots of the plane. We assume that all the throwboxes are fully connected. All the data stored in both mobile nodes and throwboxes form a dataset M. Each data is with equal size and can be completely delivered within one encounter. Mobile nodes randomly request data $i \in M$. The data is stored in throwboxes and mobile nodes. Each throwbox has a limited buffer. We consider all distributed buffers form into an uniform one, because of they are fully connected. The size of the big buffer is denoted as B and we define that b_{data} and b_{index} represent the size of buffer space stored data and index file, respectively.

Mobile nodes have two choices to fetch the requested data:

1 fetch the data through cellular network with transmission cost cc at any location

2 fetch the data from throwboxes via Wi-Fi with transmission cost c_d, when mobile nodes are in a location near the throwbox, as shown in Figure 1.

Figure 1 The network model



2.2 Utility model

Based on the basic network model, we present the utility model as follows. Each successful data fetching contains a *benefit*, denoted by $W(t)$. The benefit decreases linearly as time t elapses. The initial benefit of a data is denoted by W, while the initial benefits of different data are different. The distribution for the initial benefits of different data follows the truncated normal distribution, the mean value of which is denoted by $\bar{W}$. The decreased benefit value within each unit time interval is defined as the benefit decay coefficient, denoted by ζ. Formally, the benefit satisfies the following formula:

$$W(t) = W - t \cdot \zeta \tag{1}$$

The *utility* is defined as the benefit minus the transmission cost, denoted by $U(t)$. Let c denote the total cost incurred by message forwarding until time t, then the utility satisfies:

$$U(t) = W(t) - c \tag{2}$$

and (2) can be changed into:

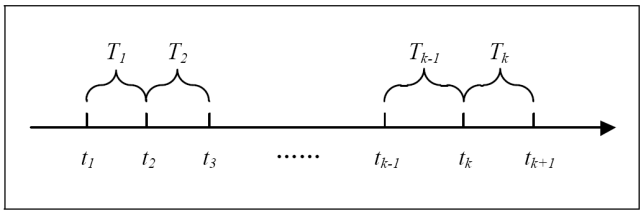
$$U(t) = W - (t\zeta + c) \tag{3}$$

We define the t_d, which makes the utility equal zero, is the *deadline* of a data fetching. Assume that the total cost of a data fetching including the time consumption and transmission cost is denoted as: $a = t\zeta + c$. Our objective is to maximise the utility.

2.3 Delivery model

A mobile node generates different data requests, each request with the deadline t_d. Once a data request is generated, according to the comparison of deadline and estimated encounter time, mobile nodes now need to decide whether to fetch the data in the cellular network. Depending on whether the throwbox stores the requested data or not, there are two different modes for the mobile nodes to get the data from throwbox:

Figure 2 Data contact timeline



2.3.1 Direct mode

If any throwbox holds the requested data, the mobile node will reply with the requested data immediately.

2.3.2 Indirect mode

Otherwise, the throwbox replies an estimation about how long it will take to get the requested data from DTN, and the

probability. Then, the node can make a decision whether to fetch the data via cellular network right now.

3 Data contact prediction

In this section, we introduce the data contact prediction algorithm. The data contact prediction includes two parts: history record collecting and future contact prediction.

Table 1 Contact history records

Data items	*Contact times*			
	1st	*2nd*	*...*	*kth*
Data 1	t_{11}	t_{12}	···	t_{1k}
Data 2	T_{21}	t_{22}	···	t_{2k}
...	...	...	...	...
Data *i*	t_{i1}	t_{i2}	···	t_{ik}
...	...	...	...	...
Data *M*	t_{M1}	t_{M2}	···	t_{Mk}

3.1 History records collecting

When a mobile node carrying data item i contacts with a throwbox, we consider this contact event as a *data contact* of item i. Throwboxes will record the name of the data and the current time. Then, throwboxes use these records to build a data contact table. The data contact table is stored as an index file in the big buffer. Table 1 gives an example. Then, the time span in contact table can be used as input to predict the time of the next data contact.

3.2 Future contact prediction

We introduce a novel algorithm to predict future data contact: time-window-based predict algorithm. Figure 2 shows an example about one data item's contact timeline (a row in the data contact table), where the time spot is denoted as t and t_k represents the time spot of the k^{th} data contact. The offline time is denoted as T. Time-window-based prediction uses the former $k-1$ offline times to predict the k^{th} offline time T_k. $J = \{T_1, T_2, \ldots, T_i, \ldots, T_{k-1}\}$ represents the set of one data item's offline time from the first data contact to the k^{th}. We consider the maximum offline time in set J is T_{Max}, $T_{Max} > \forall T_i \in J$ and we take Δt as the minimum decrease meta, where $\Delta t = \frac{T_{Max}}{k}$. Then we use Δt to build an arithmetic progression $K = \{\Delta t, 2\Delta t, \ldots, i\Delta t, \ldots, k\Delta t\}$. Every item in K is a candidate predicted offline time. By comparing the candidate predicted offline time with every historical offline time, we can find a most reliable value. (4) uses these candidates of K as an input to calculate the reliability $R(i\Delta t)$ of each candidate predicted offline time:

$$R(i\Delta t) = \frac{\sum_{j=1}^{k} a_j}{k},\ a_j = \begin{cases} 0 & i\Delta t \le T_j \\ 1 & i\Delta t > T_j \end{cases},\quad i \in [1, k] \tag{4}$$

where a_j is an indicator and equals to 1 only when the candidate is larger than the offline time T_j. Then, we compare $R(i\Delta t)$ with a threshold R_{th}, and take the minimum $R(i\Delta t)$ of all $R(i\Delta t)$s that larger than the threshold as R_m:

$$R_{\min} = \min\{R(i\Delta t) \mid R(i\Delta t) > R_{th}\},\quad i \in [1, k] \tag{5}$$

The threshold R_{th} can be used to control the preferred $R(i\Delta t)$, we take the $R_{th} = 0.5$ here. Under this setting, the calculated $i\Delta t$ is close to the median value of all the offline time T_i. We take the $i\Delta t$ whose reliability is $R_{\min}$ as the final predicted offline time PT_k:

$$PT_k = \{i\Delta t \mid R(i\Delta t) = R_{\min}\} \tag{6}$$

In reality, the data encounter frequency can change dynamically because the mobile nodes do not hold the data items all the time, the total copies of one data item is not static. To quickly adapt to such dynamic factors, we improve the basic method by a time-window technique. Its principle is to segment the whole timeline into a series of smaller time windows and to place the highest emphasis on the most recent records while gradually decreasing the emphasis on the preceding ones. Suppose the timeline was divided into s time windows, and the set of the data items' offline time also was divided into small sets: $\{T_1, T_2, \ldots, T_{\frac{k}{s}}\}, \{T_{\frac{k}{s}+1} + T_{\frac{k}{s}+2}, \ldots, T_{\frac{(m+1)k}{s}+1}\}, \ldots, \{T_{\frac{(s-1)k}{s}+1}, T_{\frac{(s-1)k}{s}+2}, \ldots, T_k\}$, and the accuracy calculated of the m^{th} time window is expressed as:

$$R_m(i\Delta t) = \frac{\sum_{j=\frac{mk}{s}+1}^{\frac{(m+1)k}{s}} a_j}{k},\ a_j = \begin{cases} 0 & i\Delta t \le T_j \\ 1 & i\Delta t > T_j \end{cases} \tag{7}$$

Now, (4) can be improved and expressed as a weighted average of $R_m(i\Delta t)$ over s time windows:

$$\begin{aligned} R(i\Delta t) &= \omega_1 R_1(i\Delta t) + \omega_2 R_2(i\Delta t) + \ldots + \omega_s R_s(i\Delta t) \\ &= \sum_{m=1}^{s} \omega_m R_m(i\Delta t) \end{aligned} \tag{8}$$

where ω_m is the weight of the time window. Different weight-selection methods (e.g., linearly or exponentially decreasing weights) would discard the history data at different rate. Here, we simply give the m^{th} time window the weight of:

$$\omega_m = \frac{m}{\sum_{i=1}^{s} i} \tag{9}$$

Lots of prediction algorithms' accuracy rate grows extremely slowly after they have enough historical records, which means there is a convergence state. Through test and analysis, we find that the convergence accuracy of the time-window-based prediction is 80% and the convergence size of the index is 40 (more details will be shown in Section 5).

Figure 3 Data fetching timeline

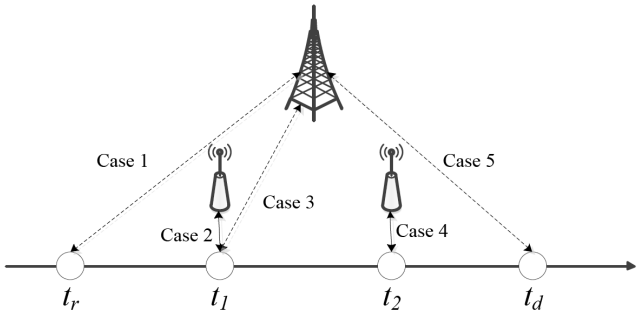


Table 2 System parameters

Para.	*Explanation*
c_c	Transmission cost of cellular network
c_d	Transmission cost of DTN
t_r	The time of users generate a data request
t_1	The time of users meet a throwbox
t_2	The time of users fetch a data from throwbox
t_d	The deadline of a data request

4 Buffer space allocation

Because of the limited buffer size, a large index file means less space to store data items. In order to achieve a better system performance, an efficient buffer space allocation strategy is needed imminently to consider the trade-off. In this section, we propose a buffer space allocation mechanism to manage the buffer space in throwboxes.

4.1 Data fetching process

Complete process of one success data fetching is shown in Figure 3. All the parameters are listed in Table 2. Based on the network and delivery model, a complete process of data fetching could be any one of the following five situations:

- Case 1: If mobile node could not encounter any throwbox before the deadline, which means its impossible to fetch the data via DTN. The best choice is to download the data via cellular networks immediately. So, the total cost in this case is $A_1 = t_r\zeta + c_c$.
- Case 2: If mobile node chooses not to fetch the data via cellular networks at t_r, then encounter a throwbox before the request deadline at t_1. Once the throwbox's buffer stores the requested data, mobile node can complete this transmission by downloading it from the throwbox. So, the total cost in this case is $A_2 = t_1\zeta + c_d$.
- Case 3: If mobile node meets a throwbox at t_1 but the throwbox does not have the data. Then, throwbox will reply a time prediction which is t_2, and let mobile node to choose waiting for throwboxes to fetch it, or fetching it via cellular networks immediately. If mobile node chooses not to wait, then the total cost of this data fetching is $A_3 = t_1\zeta + c_c$.
- Case 4: After meeting the throwbox for the first time, the mobile node could wait and continue moving. At time t_2, mobile node fetches the data via DTN from another throwbox in indirect mode successfully. Then, the total cost is $A_4 = t_2\zeta + c_d$.
- Case 5: Similar to case 4, mobile node chooses to wait to fetch the data in indirect mode. However, a wrong time prediction makes it could not fetch the data via DTN before the deadline. Mobile node has to fetch the data via cellular networks at deadline t_d with the total cost $A_5 = t_d\zeta + c_c$.

4.2 Utility function optimisation

By analysing the data fetching progress, we present the optimal size of the index file to achieve the maximum utility.

Theorem 1: Suppose the size of the index buffer space is denoted as s, the optimal size $\overline{s}$ achieved the maximum utility can be found and can be changed under different networks conditions.

To a request of data i, the utility is the remaining benefit when the transmission is finished. Thus, the system total benefit of m success data fetching for a certain time can be presented as:

$$U_{total} = \sum_{j=1}^{m} \left(W_j - a_j \right) \tag{10}$$

where j represents the j^{th} data fetching. Since mobile nodes request the data randomly, the access probability of each data $i \in M$ is equal. Due to the distribution for the initial benefits of different data follows the truncated normal distribution with a mean value $\overline{W}$, the sum of m success data fetching's initial benefits can be calculated as $\sum_{j=1}^{m} W_j = m \cdot \overline{W}$. Then (4) can be changed into:

$$U_{total} = m \cdot \overline{W} - \sum_{j=1}^{m} a_j \tag{11}$$

So, our objective can be converted to minimising the total cost $\sum_{j=1}^{m} a_j$. Based on the five data fetching cases, the total cost of one success fetching process a random data i can be presented as:

$$a_j = P_0 A_1 + (1 - P_0)\left(P_1 A_2 + (1 - P_1) A_x \right) \tag{12}$$

where P_0 is the probability that the data request deadline comes before mobile nodes encountering a throwbox, and P_1 is the probability that requested data is existing in the buffer of throwboxes. If requested data is not in the data buffer, then the cost A_x can be presented as:

$$A_x = P_2 A_y + (1-P_2) A_3 \tag{13}$$

where P_2 is the probability that the requested data is existing in the index file of throwboxes, in other words, a predicted t_i can be delivered. A_y is the cost after the predicted t_i is given and A_y can be presented as:

$$A_y = P_3 \left(P_4 A_3 + (1-P_4) A_4 \right) + (1-P_3) A_5 \tag{14}$$

where P_3 is the accuracy of the predicted offline time and P_4 is the probability that predicted data contact delay is within the request deadline. Simultaneously considering (14), (15) and (16), the average total cost of one successful data downloading can be presented as:

$$\begin{cases} a_j = P_0 A_1 + (1-P_0)(P_1 A_2 \\ \quad + (1-P_1)\{P_2 \{P_3 \left(P_4 A_3 + (1-P_4) A_4 \right) \\ \quad + (1-P_3) A_5 + (1-P_2) A_3\}\} \end{cases} \tag{15}$$

In order to calculate the time cost and transmission cost synthetically, we transform all the cost into the form of c_d. Suppose that:

$$\begin{cases} t_r = m_1 c_d, t_1 = m_2 c_d, t_2 = m_3 c_d, t_4 = m_4 c_d \\ c_c = m_0 c_d \end{cases} \tag{16}$$

where m_0, …, m_4 are controllable parameters, by changing the value of m_0, …, m_4, the system can modify the weight of time and transmission cost in the total cost and adapt the delay-sensitive or the transmission-cost-sensitive environment. Then, we can get:

$$\begin{cases} A_1 = m_0 c_d \\ A_2 = (m_2 + 1) c_d \\ A_3 = (m_2 + m_0) c_d \\ A_4 = (m_3 + 1) c_d \\ A_5 = (m_4 + m_0) c_d \end{cases} \tag{17}$$

Simultaneous (17) and (19), we can change the problem into calculating the minimised coefficient of c_d. The size of data index meta is k (as we explained in Section 3) and suppose the size of the index buffer space are denoted as s. So, P_1 and P_2 can be presented as:

$$P_1 = \frac{B-s}{M}, P_2 = \frac{s}{Mk}, s \in (0, s^*) \tag{18}$$

where $s^* = \min\{B - 1, Mk\}$ and it ensures that there is at least one space to store the data item. P_0, P_3 and P_4 are known constant. Suppose that $F(s)$ represents the coefficient function of (9):

$$F(s) = \frac{a_i}{c_d} \tag{19}$$

Then after merging similar terms, $F(s)$ can be presented as:

$$F(s) = \alpha s^2 + \beta s + \gamma \tag{20}$$

where α, β, γ are formed by parameters B, M, k, P_0, P_3, P_4, m_0, m_1, m_2, m_3, m_4. The derivative of $F(s)$ can be presented as:

$$F'(s) = 2\alpha s + \beta \tag{21}$$

So, when $\alpha \geq 0$, $F(s)$ achieve the minimum if:

$$s = -\frac{\beta}{2\alpha} \tag{22}$$

so we can know that the optimal $\bar{s}$ satisfies:

$$\bar{s} = \begin{cases} -\frac{\beta}{2\alpha}, & 0 < -\frac{\beta}{2\alpha} < s^* \\ 0, & 0 > -\frac{\beta}{2\alpha} \\ s^*, & -\frac{\beta}{2\alpha} > s^* \end{cases} \tag{23}$$

and when $\alpha < 0$, $F(s)$ achieve the maximum if:

$$s = -\frac{\beta}{2\alpha} \tag{24}$$

so we can know that the optimal $\bar{s}$ satisfies:

$$\bar{s} = \begin{cases} 0, & -\frac{\beta}{2\alpha} > \frac{1}{2} s^* \\ s^*, & -\frac{\beta}{2\alpha} \leq \frac{1}{2} s^* \end{cases} \tag{25}$$

□

With all the parameters mentioned above, the utility function can evaluate the system total utility, and give the optimal index file's size $\bar{s}$.

4.3 Buffer space management algorithm

Base on the theoretical analysis above, we develop the *index-based* buffer space management algorithm, as shown in Algorithm 1. The index-based buffer space management guarantees that throwboxes will choose the best way to allocate the buffer space to make sure the system total cost is the minimum. Index-based buffer space management includes four phrases: *strategy-choosing phrase*, *fill-up phrase*, *adjustment phrase* and *static phrase*.

Buffer space management algorithm:

Input: System parameters: M, B, k, P_0, P_3, P_4, m_0, …, m_4;

1: With all the parameters, calculate the s using (16);

2: According to different α, compare the s and s^* to determine the optimal $\bar{s}$;

3: ***for*** *each data i contact event* ***do***

4: ***if*** *optimal s = 0* ***then***

5: ***if*** *$b_{data} < B$* ***then***

6: ***if*** *data i ∉ b_{data}* ***then***

7: *fetch data i into the buffer;*

8: ***end if***

```
9:          else
10:             break;
11:         end if
12:     else
13:         if b_data + s < B then
14:             record this contact information into index file;
15:             if i ∉ b_data then
16:                 fetch data i into the buffer;
17:             end if
18:         else
19:             if s < s̄ then
20:                 delete the data item with the lowest initial
                    benefit and record this contact information;
21:             else
22:                 delete the oldest contact information and
                    store this new one;
23:             end if
24:         end if
25:     end if
26: end for
```

4.3.1 Strategy-choosing phrase

At the very beginning of the system's operation, throwboxes must determine the storage strategy. Steps 1 and 2 calculate the s using (24) according to the corresponding parameters, then comparing the s and s^* to choose the optimal $\bar{s}$ which achieves the maximum system benefit, and use the optimal $\bar{s}$ to guide the later phrase.

4.3.2 Fill-up phrase

Steps 4 to 9 show that if the optimal $s = 0$, then throwboxes will store the data item into the buffer until it is full, instead of recording any contact information. After the buffer is full of data, the system goes to the static phrase. If the optimal $\bar{s} = s^*$ or $\bar{s} = -\frac{\beta}{2\alpha}$, then whenever a mobile node holding some data items encounter a throwbox, throwbox will record the data contact information into index file, and fetch the data items into the buffer if these items have no copies in the buffer (steps 10–14). This procedure will continue until the buffer is full, after that, system goes to the next phrase.

4.3.3 Adjustment phrase

Once the buffer is full, system using steps 16 to 19 to delete one data item and use the empty space to store the upcoming data contact information. After a while, the empty space will be used up and another data item will be deleted as before. This phrase will end as long as the index file reaches the size of optimal $\bar{s}$ and system will go to the static phrase.

4.3.4 Static phrase

In this phrase, the proportion of data buffer and the index file will not change anymore. The replacement strategy of new data contact records and new data items are same as the strategy in adjustment phrase.

5 Evaluation and discussion

In this section, we present our simulation to evaluate the performance of buffer space allocation algorithm under various settings. The evaluation methods, settings, and results are presented as follows.

5.1 Simulation settings and metrics

We use two types of traces to conduct our simulations. The first trace is generated by the ONE simulator in Keranen et al. (2009). We deploy 100 mobile nodes in a small area of a real city: Helsinki, Finland. Mobile nodes perform shortest path map-based movement patterns on the roads. The second trace is a large-scale dataset of real GPS traces from around 320 taxies operational in the urban area in Rome, Italy. In the simulation, the virtual throwboxes are deployed in the street in every two kilometres. Each mobile node has a buffer to store five data items, and generate a request of a random data item from the set M. At the initiate stage of the system, mobile nodes store five data items randomly selected from the set M and throwboxes have a short time to warm up recognising the basic information of the whole network.

We run the simulations under two different number of data, $M = 300, 400$. Then we take P_4, B and k as our main objects of observation. Deadline is determined by the initial benefit, the initial benefit W follows the truncated normal distribution with the mean value $\bar{W} = 20{,}000$. In order to simplify the simulation, we set different request deadlines to represent the initial benefit. We set the default index meta size as 0.05 while the size of a data item is set as 1. We combine all throwboxes buffer as one big buffer sized from $B = 55$ to $B = 100$ because of the full connection.

In order to evaluate the effects of the index-based buffer space management algorithm, we also implement a traditional buffer management mechanism called *data-only* buffer space management, where throwboxes use all storage space to store the data item.

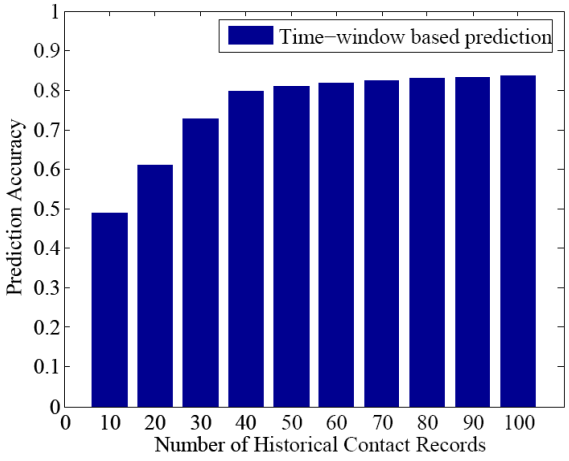
Table 3 Evaluation settings

Parameter name	*Default*	*Range*
Number of mobile nodes N_n	100	100–120
Number of data M	300	300–400
Deadline	20,000	5,000–45,000
Throwboxes buffer size B	75	60–100
Index meta size k	0.05	0.01–0.1

5.2 Results and discussion

Firstly, to evaluate the accuracy of the time-window-based prediction, we run some tests and results are shown as Figure 4. We record every encounter time of a random data item i and repeat the simulation with different number of contact records for 100 times each. Figure 4 shows that the prediction accuracy is increasing as the historical records grow up until the index records about 40 times contact information. Then, the prediction accuracy stays around the 80%. So, in the following simulations, throwboxes' index only records 40 latest contact informations.

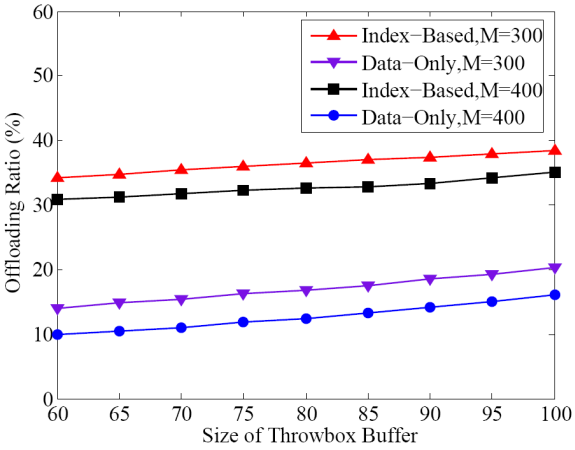
Figure 4 Prediction accuracy under different numbers of historical records (see online version for colours)



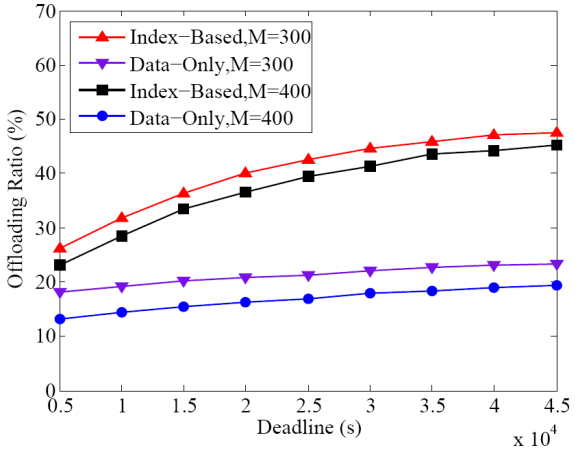
5.2.1 Effects of the throwbox buffer size

Figure 5(a) and Figure 7(a) show the offloading ratio under different throwbox buffer size. We can see that when the size of the throwbox buffer increases, the offloading ratio of two algorithms will increase and the offloading ratio of index-based algorithm is always higher than the data-only algorithm. And we also notice that the system performance is different under different number of data items $M = 300$ and $M = 400$. More data items means more diversity data requests, which will reduce the request hit rate in the throwbox buffer. Figure 6(a) and Figure 8(a) give the average delay of all the data fetching. As we had expected, the average data fetching delay of index-based algorithm is much lower than the data-only algorithm. This is due to the future contact prediction algorithm.

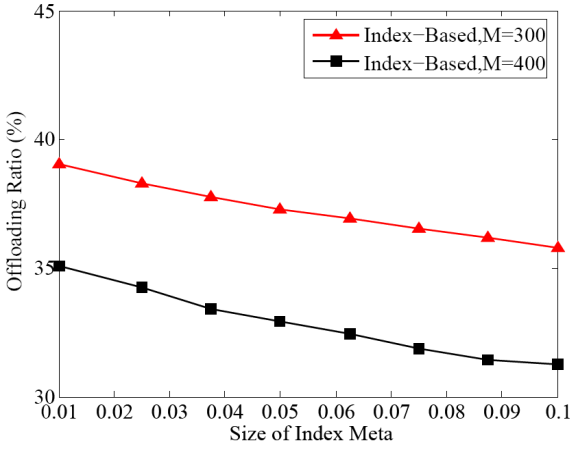
Figure 5 System average offloading ratio under generated traces, (a) offloading ratio vs. size of throwbox buffer (b) offloading ratio vs. deadline (c) offloading ratio vs. size of index meta (see online version for colours)



(a)

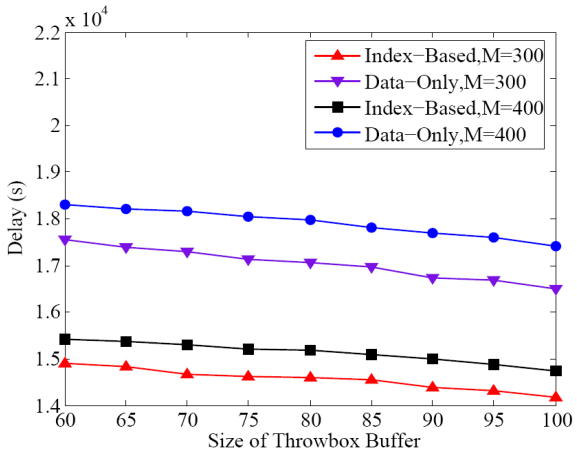


(b)

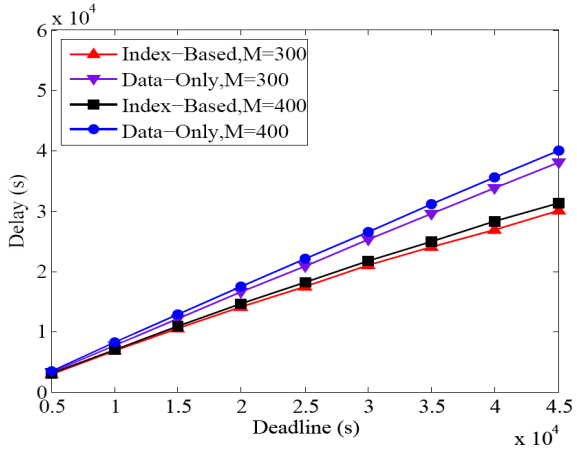


(c)

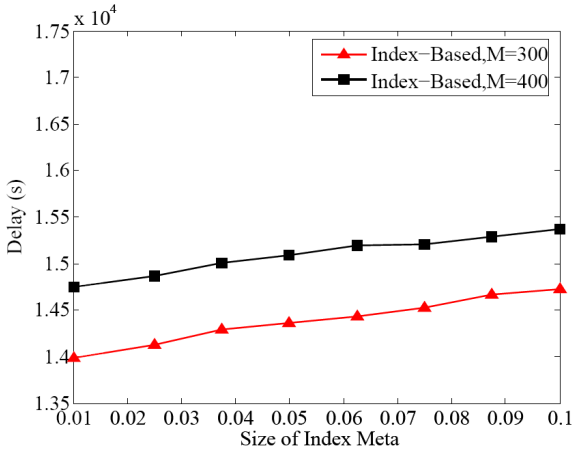
Figure 6 System average delay under generated traces, (a) delay vs. size of throwbox buffer (b) delay vs. deadline (c) delay vs. size of size of index meta (see online version for colours)



(a)

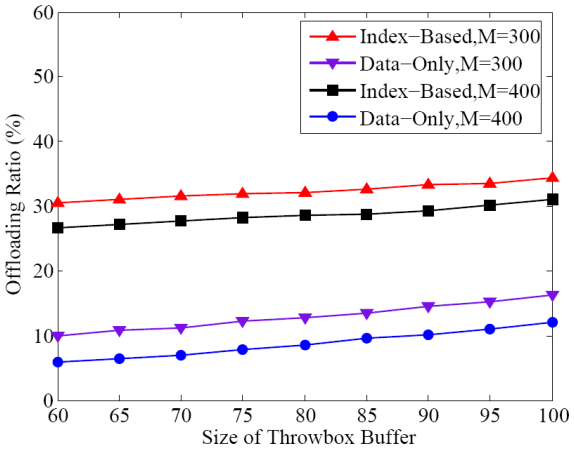


(b)

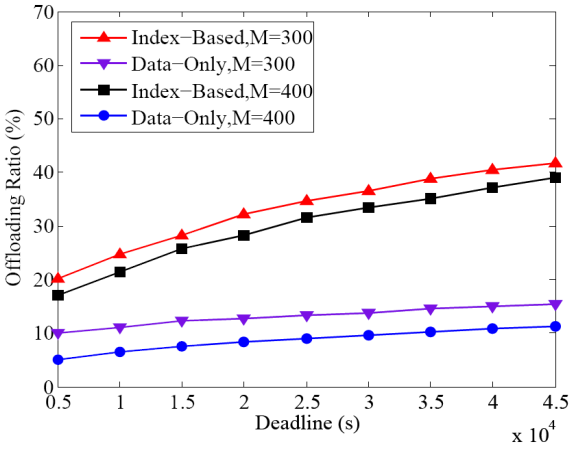


(c)

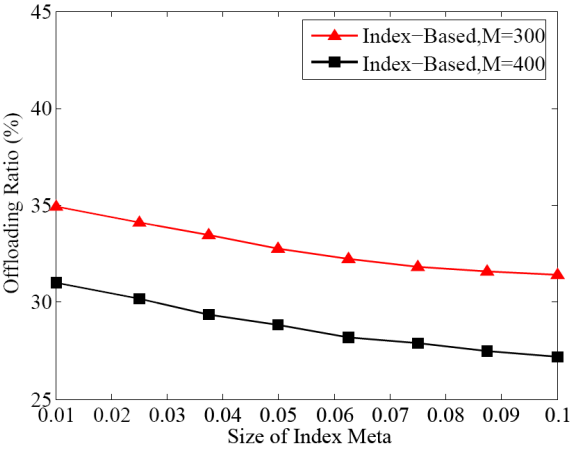
Figure 7 System average offloading ratio under real traces, (a) offloading ratio vs. size of throwbox buffer (b) offloadingratio vs. deadline (c) offloading ratio vs. size of index meta (see online version for colours)



(a)

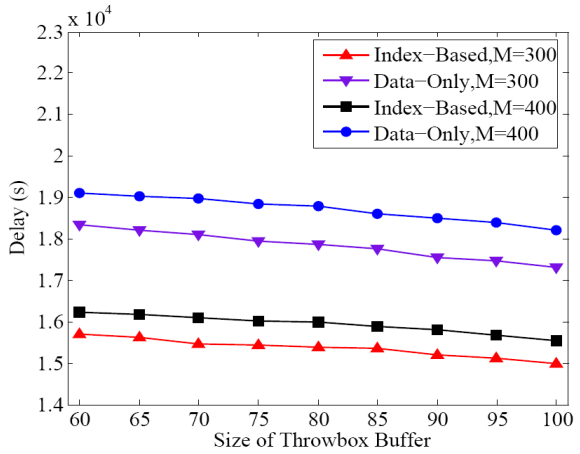


(b)

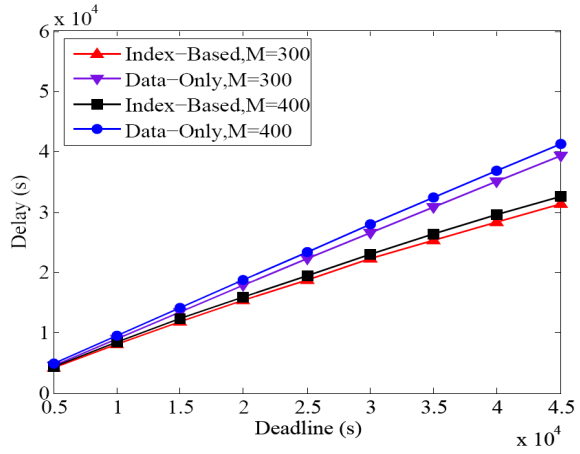


(c)

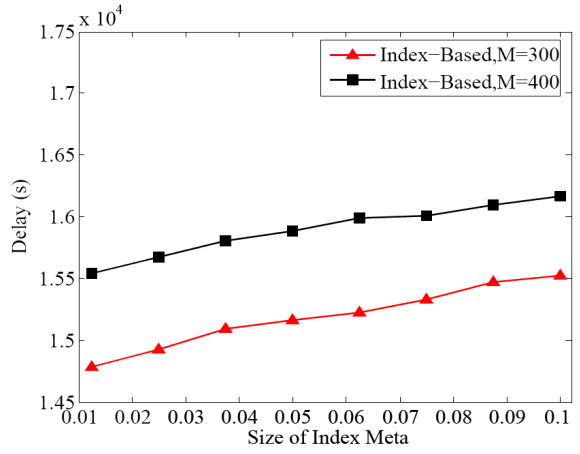
Figure 8 System average delay under real traces, (a) delay vs. size of throwbox buffer (b) delay vs. deadline (c) delay vs. size of size of index meta (see online version for colours)



(a)



(b)



(c)

5.2.2 *Effects of the deadline*

Figure 5(b) and Figure 7(b) show the offloading ratio under different initial benefit. Initial benefits determined the deadline of each data request, so we choose to change the deadline to observe the system performance. The increasing rate of the data-only algorithm is very low and increasing rate of the index-based algorithm is much higher at the beginning and gradually low down. To the data-only algorithm, due to lack of network global information, data items in throwboxes buffer do not change since the buffer is full. It can increase the hit rate in throwboxes buffer but the efficiency is quite low. When the deadline is long enough for most data requests can be fetched from other mobile nodes, the increasing rate reduces.

The initial benefit's influence to average delay is represented in Figure 6(b) and Figure 8(b). As we can see, when the deadline is short, the gap between the two algorithms is quite narrow. But as the deadline increased, the gap is wider and wider. The reason is that when the deadline is short, mobile nodes can hardly have chance to meet the throwboxes more than once, so the index-based algorithm can barely help mobile nodes to fetch data. But when the deadline enlarges, the advantage of index-based algorithm shows up and lots of requests were offloaded before the deadline coming. However, a longer deadline cannot give the data-only algorithm the same benefits, so the gap becomes wider.

5.2.3 *Effects of the index meta size*

Figure 5(c) and Figure 7(c) show that as the size of the index meta become larger, the offloading ratio is reducing. This is because when the size of index meta is small (e.g., k = 0.01), replacing a data item can store 100 more data index into the buffer and the space of three data items can store all the data index if the number of data is 300, this kind of replacing is very efficient. When the size of index meta is increasing, replacing data items with index is still useful but not that significantly. The average delay showed in Figure 6(c) and Figure 8(c) also give the same results. A smaller index meta size gives throwboxes more space to store data items and it can bring a shorter delay.

6 Related work

Throwboxes-based DTN are first proposed in Zhao et al. (2006). In the later works in Ibrahim et al. (2007, 2009), simulation results and real deployments have demonstrated that importing a number of throwboxes into the DTN can indeed improve the routing performances and overall throughput. Besides, some other studies focusing on analytical models for delay distribution in Gu et al. (2010) and designing/evaluating routing strategies in Gu et al. (2010) for throwbox-based DTN are also presented. Meanwhile, Banerjee et al. (2010) consider the problem about energy efficiency of each throwbox node for throwbox-based DTN. The main difference between our work and previous work is that we implement a contact prediction mechanism on throwboxes, by sacrificing some data storage, and treat throwboxes as both data buffers and forecast equipment. To the best of our knowledge, this is the first work that makes throwboxes become multifunctional.

In existing prediction-based schemes, mobile nodes' mobility and contact is estimated based on a history of observations. A representative case is using utility-routing in Lindgren et al. (2003) and Zhang et al. (2007), where mobile nodes consider the utility value as the predictor of two nodes' future likelihood of encounter. Deng and Chang (2014) propose a multicast routing scheme based on social difference (SDMR), which considers the social differences between nodes, including both the similarity and the centrality differences. LeBrun et al. (2005) propose a routing algorithm for VANET that use the current position and trajectories of nodes to predict their future position and calculate the distance to the destination. Yeh et al. (2014) reveal a system performance prediction and analysis method for multi-core system by adopting electronic system-level (ESL) design methodology. In Burns et al. (2005), they propose a prediction scheme that uses past frequencies of contacts, as well as the past contacts. Another prediction-based generic algorithm for DTN routing is MobySpace in Leguay et al. (2006), which uses a high-dimensional Euclidean space constructed upon nodes mobility patterns. The major difference between our approach and previous works is we take the data as our target of prediction.

7 Conclusions

In this paper, we introduce a novel throwbox design by adding an index file into the buffer, which modifies the throwbox from a pure data buffer into a data transfer helper with future prediction. Aiming at the trade-off between data and index, we propose a utility function to evaluate the system performance under different combinations of variables. Theoretical analysis shows that replacing some data items with an index file in the buffer can reduce the total cost effectively in most cases. Simulations results also prove that the index-based prediction plays an important role in reducing the transmission cost of data fetching. Besides, simulations further show that the index-based buffer space allocation mechanism outperforms the simple index-added mechanisms. Our future work will mainly focus on two aspects. The first is to extend current system model to enable data transmission among mobile nodes. Secondly, we will bring in real-world trace into simulations to evaluate the system performance.

Acknowledgements

This research was supported by the NSF of China (Grant Nos. 61402425, 61272470, 61305087, 61440060), the China Postdoctoral Science Foundation funded project (Grant No. 2014M562086), the Fundamental Research Funds for National University, China University of Geosciences, Wuhan (Grant No. CUG14065, CUGL150829), the Provincial Natural Science Foundation of Hubei (Grant No. 2015CFA065).

References

Banerjee, N., Corner, M.D. and Levine, B.N. (2010) 'Design and field experientation of an energy-efficient architecture for DTN throwboxes', *IEEE/ACM Transactions on Networking*, Vol. 18, No. 2, pp.554–567.

Burns, B., Brock, O. and Levine, B.N. (2005) 'MV routing and capacity building in disruption tolerant networks', in *IEEE INFOCOM*.

Cisco Visual Networking Index (Cisco VNI) *Forecast and Methodology*, 2014–2019 White Paper [online] http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html.

Deng, X. and Chang, L. (2014) 'A time-considered multicast routing scheme based on social differences in delay-tolerant networks', in *International Journal of Embedded Systems*, Vol. 6, No. 1, pp.50–60.

Fall, K. (2003) 'A delay tolerant network architecture for challenged internets', in *ACM SIGCOMM*.

Gu, B., Hong, X., Wang, P. and Borie, R. (2010) 'Latency analysis for thrown box based message dissemination', in *IEEE Globecom*.

Ibrahim, M., Al Hanbaliand, A. and Nain, P. (2007) 'Delay and resource analysis in MANETs in presence of throwboxes', *Performance Evaluation*, Vol. 64, Nos. 9–12, pp.933–947.

Ibrahim, M., Nain, P. and Carreras, I. (2009) 'Analysis of relay protocols for throwbox-equipped DTNs', in *WiOPT*.

Jeremie, L., Timur, F. and Vania, C. (2006) 'Evaluating mobility pattern space routing for DTNs', in *IEEE INFOCOM*.

Keranen, A., Ott, J. and Karkkainen, T. (2009) 'The ONE simulator for DTN protocol evaluation', in *Simutools*.

LeBrun, J., Chuah, C. and Ghosal, D. (2005) 'Knowledge based opportunistic forwarding in vehicular wireless ad hoc networks', *IEEE VTC*, Vol. 4, pp.2289–2293.

Lee, K., Lee, J., Yi, Y., Rhee, I. and Chong, S. (2013) 'Mobile data offloading: how much can WiFi deliver?', *IEEE/ACM Transactions on Networking*, Vol. 21, No. 2, pp.536–550.

Leguay, J., Friedman, T. and Conan, V. (2006) 'Evaluating mobility pattern space routing', in *IEEE INFOCOM*.

Lindgren, A., Doria, A. and Schelen, O. (2003) 'Probabilistic routing in intermittently connected networks', *ACM SIGMOBILE Mobile Computing and Communications Review*, Vol. 7, No. 3, pp.19–20.

Mehmeti, F. and Spyropoulos, T. (2014) 'Is it worth to be patient? Analysis and optimization of delayed mobile data offloading', in *IEEE INFOCOM*.

Spyropoulos, T., Psounis, K. and Raghavendra, C.S. (2006) 'Performance analysis of mobility-assisted routing', in *ACM MobiHoc*.

Yeh, J.C., Lin, C.H. and Liu, C.N. (2014) 'Multi-core system performance prediction and analysis at the ESL', in *International Journal of Computational Science and Engineering*, Vol. 9, Nos. 1–2, pp.86–94.

Zhang, X., Neglia, G., Kurose, J. and Towsley, D. (2007) 'Performance modeling of epidemic routing', *Computer Networks*, Vol. 51, No. 10, pp.2867–2891.

Zhao, W., Chen, Y., Ammar, M., Comer, M.D., Levine, B.N. and Zegura, E. (2006) 'Capacity enhancement using throwboxes in DTNs', in *IEEE MASS*.