Data or Index: A Trade-Off in Mobile Delay Tolerant Networks

Hong Yao†, Han Zhang†, Jie Wu*, Huanyang Zheng*, Changkai Zhang†, Deze Zeng†
†School of Computer Science, China University of Geoscience, Wuhan, China
*Department of Computer and Information Sciences, Temple University, USA
Email: {dr.hyao, sylarjohn, cug09evan, dazzae}@gmail.com, {huanyang.zheng, jiewu}@temple.edu

***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 are 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.**

***Index Terms*—Mobile networks, delay tolerant networks, throwbox, index.**

I. Introduction

The rapid growth of all kinds of mobile devices (high-end smart phones, tablets, vehicles equipped with networking devices, etc.) leads to a mobile data explosion. According to Cisco forecasts [1], we are now facing the "mobile data apocalypse"; by 2018, mobile data traffic will increase by 13 times, and will climb to 15 exabytes per month. 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, data dissemination. And, the authors in [2] indicate that delayed transmissions can achieve substantial gains, especially when the tolerant deadline is longer than 1h (up to a 29% traffic gain increase). In [3], the authors further analyze 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.

Delay Tolerant Networks (DTN) [4] performs the so-called store-carry-forward paradigm to deliver messages in an end-to-end fashion, although a continuous end-to-end communication path may never exist between sender and destination devices. However, 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* [5], and to locate them at some strategic geographical positions so as to help the mobile nodes in exchanging data. In other words, throwboxes are stationary wireless nodes with significantly improved storage and energy capabilities that simply act as fixed relays. Differ from the previous delayed-offloading schemes, throwboxes use the data cached in each mobile node as the data recourses. Once a data request is received, throwboxes will try to fetch the data from other mobile nodes instead of downloading it from the Internet. Traditional offloading strategy and the deadline-driven mechanism make mobile nodes always try to wait fetching data from throwboxes until the deadline. Obviously, the scheme mentioned above could offload the maximum data traffic. But the disadvantage is quite clear, either. 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. We notice that the mobility of real mobile users follows some social characteristics rather than a random mobility model. Many mobility models [7]-[9] capture these characteristics from several real traces. This important discovery proves 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 minimized total cost, under different network conditions, is the objective of this work.

In order to quantify the influence that different proportions

of data space and index space have on the system, we build a utility function to make a comprehensive evaluation about the system performance under different situations. Then, based on the theoretical analysis, we propose an *index-based buffer allocation mechanism*. This allocation mechanism gives throwboxes the ability to adapt to some different network environments and to automatically choose the best strategy to allocate the buffer storage. Thus, it can achieve a better performance than the existing buffer management mechanism and some simple approaches that add an index into the buffer.

The key contributions of this paper are summarized as follows:

- We add an index file to the throwboxes, making throwboxes able to not only store data items, but also to 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 minimize 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 the paper is organized as follows. We introduce the system model in Section II, and then we introduce the data encounter prediction approach in Section III. The buffer space allocation mechanism is presented in Section IV. Simulation results which are presented in Section V prove our theory. Finally, we review the related work in Section VI and conclude the paper in Section VII.

II. System Model

In this section, we introduce the system model, including network model and delivery model. Before that, let's see a typical example of vast throwbox-equipped DTN applications. When mobile nodes request a data, it will try to get it from other mobile nodes that carrying the data through throwboxes' help. The mobile node will wait throwboxes to fetch the data from other mobile nodes until the deadline of the requested data. If the requested data wasn't fetched till deadline, mobile node will fetch it from Internet via cellular networks. This mode was widely used because it could minimize the transmission cost. So we consider our system model this way either.

A. 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

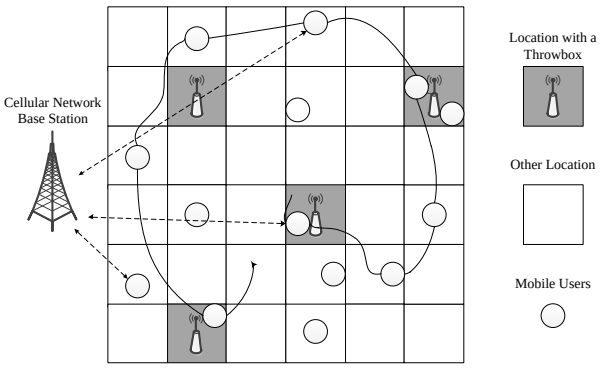


Fig. 1. The network model

mobile nodes and throwboxes form a data set 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 s represent the size of the index file, then $B - s$ represents the size of buffer space stored data.

Mobile nodes have two choices to fetch the requested data, (i) fetch the data through cellular network with transmission cost c_c at any location; (ii) 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 Fig. 1.

B. Utility Model

Based on the basic network model, we present the utility model as follows. Each success 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 $\overline{W}$. Simulations in Section V show that we can use the mean value of W to estimate the buffer management strategy. 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:

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

The utility decreases monotonously with respect to the delay. 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 maximize the utility. For each data request, the trade-off between delay and transmission cost must be considered, which will guide the node to make a proper choice to fetch the data. To do so, we apply a buffer space allocation algorithm and a novel data contact prediction algorithm on the throwboxes, which are presented in following sections.

C. 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. If the deadline is too short to encounter a throwbox before the deadline, fetching the data via cellular network will be a good choice. Otherwise, mobile nodes can choose to wait and fetch the data via DTN. If so, they will send the request to a throwbox when they enter the communication range of the throwbox. 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:

1) Direct Mode: If any throwbox holds the requested data, the mobile node will be replied with the requested data immediately.

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. If the mobile node chooses to fetch the data via throwbox, the throwbox then disseminates a data fetching command to all throwboxes. Then, the throwboxes encounter a mobile node with the requested data will hold the data. After throwboxes hold the requested data, the mobile node can get the data later, when it will encounter a throwbox next time. If the estimation is longer than the deadline, mobile node can choose to download the data from remote center via cellular networks immediately, avoiding the meaningless waiting.

The estimations replied by the throwboxes are calculated based on the data contact prediction, which give mobile nodes suggestions to make a proper choice to get the requested data. To get the data in a lower price, and in a waiting time not too long. We will present the details of data contact prediction in the next section.

III. Data Contact Prediction

In this section, we introduce the data contact prediction algorithm. Differ from other contact prediction schemes, we take the contact between a data item and the throwboxes as the major prediction object. The data contact prediction algorithm only pays attention on when and which data contact the throwboxes and it doesnt need to know which mobile user carried what data. The reason why we record data contact events rather than node contact events is that the key point which mobile users care about is fetching the requested data before the deadline, they dont care where the data is. Due to this characteristic, the data contact pattern is not only affected by the mobile nodes mobility pattern but also affected by other elements such as data distribution and so on. Thus, traditional future contact prediction algorithms dont suit this well and we will give our solution below. The data contact prediction includes two parts: history record collecting and future contact prediction.

TABLE I
Contact History Records

Data Items	**Contact Times**			
	1st	2nd	$\cdots$	kth
Data 1	t_{11}	t_{12}	$\cdots$	t_{1k}
Data 2	t_{21}	t_{22}	$\cdots$	t_{2k}
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
Data i	t_{i1}	t_{i2}	$\cdots$	t_{ik}
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
Data M	t_{M1}	t_{M2}	$\cdots$	t_{Mk}

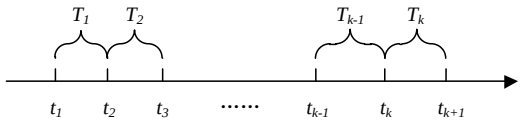


Fig. 2. Data contact timeline

A. 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. If the mobile node has two or more data items, Throwboxes will record the information of all data. 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. The time span between any two neighboring time spot is the *off-line time*. Table I gives an example. Then, the time span in contact table can be used as input to predict the time of the next data contact. Using the time stamp in the contact table as input, throwboxes can predict the time of the next data contact as below.

B. Future Contact Prediction

We introduce a novel algorithm to predict future data contact: time-window based predict algorithm. Fig. 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 kth data contact. The off-line time is denoted as T. Time-window based prediction uses the former $k-1$ off-line times to predict the kth off-line time T_k. $J = \{T_1, T_2, ..., T_i, ..., T_{k-1}\}$ represents the set of one data item's off-line time from the first data contact to the kth. We consider the maximum off-line 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 $K = \{\Delta t, 2\Delta t, ..., i\Delta t, ..., k\Delta t\}$.Every item in K is a candidate predicted off-line time. By comparing

the candidate predicted off-line time with every historical off-line 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 off-line time:

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

where a_j is an indicator and equals to 1 only when the candidate is larger than the off-line 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:

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

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 off-line time T_i. We take the $i\Delta t$ whose reliability is R_{min} as the final predicted off-line time PT_k:

$$PT_k = \{i\Delta t|R(i\Delta t) = R_{min}\} \quad (6)$$

In reality, the data encounter frequency can change dynamically because the mobile nodes don't 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' off-line time also was divided into small sets:$\{T_1, T_2, ..., T_{\frac{k}{s}}\}$,$\{T_{\frac{k}{s}+1}, T_{\frac{k}{s}+2}, ..., T_{\frac{(m+1)k}{s}}\}$,...,$\{T_{\frac{(s-1)k}{s}+1}, T_{\frac{(s-1)k}{s}+2}, ..., T_k\}$, and the accuracy calculated of the mth time window is expressed as:

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

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

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

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 mth time window the weight of:

$$\omega_m = \frac{m}{\sum_{i=1}^{s} i} \quad (9)$$

Lots of prediction algorithms' accuracy rate grows extremely slow 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 V). So we define the index file that only record one data item's 40 latest contact information as the *data index meta*, suppose the size of data index meta is k, then the size of the index file stored all data's index is Mk.

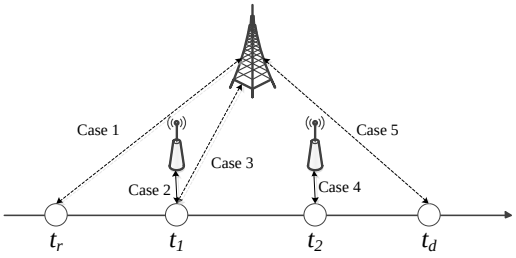


Fig. 3. Data fetching timeline

TABLE II
SYSTEM PARAMETERS

parameters	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 for the first time
t_2	the time of users fetch a data from throwbox
t_d	the deadline of a data request

IV. BUFFER SPACE ALLOCATION

Helping mobile nodes downloading data via DTN can reduce the total transmission cost. On the other hand, a complete data historical record means a large index file. Because of the limited buffer size, a large index file means less space to store data item. 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 use the total cost utility function introduced in Section II to make a comprehensive evaluation about the system total cost under different situations. Then we propose a buffer space allocation mechanism to manage the buffer space in throwboxes.

A. Data Fetching Process

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

- Case 1: Due to some reasons (traffic jam, too short deadline, etc.), mobile node couldn't encounter any throwbox before the deadline, which means its impossible to fetch the data via DTN. Under this circumstance, spending time to wait is meaningless. 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 doesn't 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 make it could't 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 $A_5 = t_d\zeta + c_c$.

B. Utility Function Optimization

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

Theorem 1: Let s denote the size of the buffer space that stores the index, then its optimal configuration $\overline{s}$, can be solved to maximize the data utility.

Proof: 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:

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

where j represents the jth 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 minimizing 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:

$$a_i = P_0A_1 + (1-P_0)(P_1A_2 + (1-P_1)A_x) \tag{12}$$

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:

$$A_x = P_2A_y + (1-P_2)A_3 \tag{13}$$

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:

$$A_y = P_3(P_4A_3 + (1-P_4)A_4) + (1-P_3)A_5 \tag{14}$$

where P_3 is the accuracy of the predicted off-line time and P_4 is the probability that predicted data contact delay is within the request deadline. Simultaneously considering (12) (13) (14), the average total cost of one successful data downloading can be presented as:

$$\begin{aligned} a_i =& P_0A_1 + (1-P_0)(P_1A_2 + (1-P_1)\{P_2\{P_3(P_4A_3 \\ &+ (1-P_4)A_4) + (1-P_3)A_5\} + (1-P_2)A_3\}) \end{aligned} \tag{15}$$

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

$$\begin{cases} t_r = m_1c_d, t_1 = m_2c_d, t_2 = m_3c_d, t_d = m_4c_d \\ c_c = m_0c_d \end{cases} \tag{16}$$

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:

$$\begin{cases} A_1 = m_0c_d \\ A_2 = (m_2+1)c_d \\ A_3 = (m_2+m_0)c_d \\ A_4 = (m_3+1)c_d \\ A_5 = (m_4+m_0)c_d \end{cases} \tag{17}$$

Simultaneous (15) and (17), we can change the problem into calculating the minimized coefficient of c_d. The size of data index meta is k (as we explained in Section III) 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^* represents the maximum storage space that index can use, and $s^* = min\{B-1, Mk\}$ 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 (15):

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

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

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

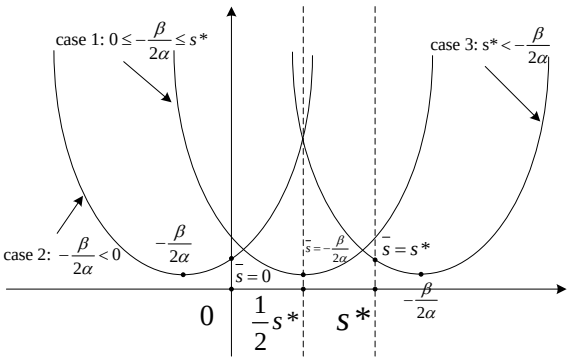


Fig. 4. Different optimal $\overline{s}$ when $\alpha \geq 0$

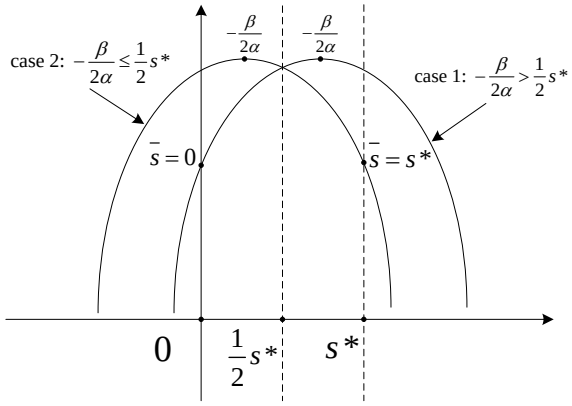


Fig. 5. Different optimal $\overline{s}$ when $\alpha < 0$

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 $\overline{s}$ satisfies:

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

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

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

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

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

All the cases are shown in Fig. 4 and Fig. 5. ■

With all the parameters mentioned above, the utility function can evaluate the system total utility, and give the optimal index file's size $\overline{s}$. By analyzing (22)(23)(25), we can find out that the system will be more likely to give index more buffer space if the requested data has a longer deadline, the throwboxes have a larger buffer space or the size of index meta is smaller. In other words, the optimal size of index $\overline{s}$ is mainly affected by the probability that predicted data contact delay is smaller than the request deadline P_4, the throwbox buffer size B and the index meta size k. P_4 is affected by the data's initial benefit. Thus, the system will prefer to store a larger index if the mean initial benefit is higher. Meanwhile, if the k becomes smaller, the system also should enlarge the index to achieve the maximize remaining benefit. The trade-off between data and index is determined by different network conditions.

In mobile networks, the system runs in a distributed and self-organized way. So, at the very beginning of operation, the system will have a warm up phrase to recognize the networks' characteristic, including the number of data M, the probability that request deadline comes before mobile nodes encountering the first throwbox P_0, the accuracy of the predicted off-line time P_3 and the P_4 mentioned above. Other parameters, the throwbox buffer size B and the index meta size k are determined by the throwbox. And the weight coefficient $m_0, ..., m_4$ are determined by system's setting. In order to make sure that throwboxes can allocate the buffer space precisely to achieve the minimum system cost, we further design a buffer space management mechanism.

C. 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*.

1) Strategy-Choosing Phrase:

At the very beginning of the system's operation, throwboxes must determine the storage strategy. Steps 1-2 calculate the s using (22) according to the corresponding parameters, then comparing the s and s^* to choose the optimal $\overline{s}$ which achieves the maximum system benefit, and use the optimal $\overline{s}$ to guide the later phrase.

2) Fill-Up Phrase:

After choosing the strategy of buffer space allocation, the throwboxes' buffer needs to be filled up as soon as possible. Steps 4-9 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 $\overline{s} = s^*$ or $\overline{s} = -\frac{\beta}{2\alpha}$, 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.

3) Adjustment Phrase:

Algorithm 1 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 (22);
2: According to different α, compare the s and s^* to determine the optimal $\overline{s}$;
3: **for** each data i contact event **do**
4: **if** optimal $\overline{s} = 0$ **then**
5: **if** $b_{data} < B$ **then**
6: **if** data $i \notin b_{data}$ **then**
7: fetch data i into the buffer;
8: **else**
9: break;
10: **else**
11: **if** $b_{data} + s < B$ **then**
12: record this contact information into index file;
13: **if** $i \notin b_{data}$ **then**
14: fetch data i into the buffer;
15: **else**
16: **if** $s < \overline{s}$ **then**
17: delete the data item with the lowest initial benefit and record this contact information;
18: **else**
19: delete the oldest contact information and store this new one;

Once the buffer is full, system using steps 16-19 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. As the time goes by, more and more data contacts were recorded into the index file and the file become larger and larger. This phrase will end as long as the index file reaches the size of optimal $\overline{s}$ and system will go to the static phrase.

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.

Notice: In this paper, we use a closed network model to evaluate the index-based buffer space allocation algorithm, but it doesn't mean the index-based buffer allocation mechanism only apply the closed network model. If the network condition is dynamical, the mechanism can be easily modified to adapt by doing the observation continuously and revise the optimal $\overline{s}$. Then the fourth phrase, *static phrase*, is no longer last forever but turns to the *strategy-choosing phrase* when the observed parameters are changed. With this design, the mechanism was endowed a high portability and usability.

V. 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.

TABLE III
Evaluation Settings

parameter name	default	range
number of mobile nodes N_n	100	100-120
number of data M	300	300-400
Deadline	20000	5000-45000
throwboxes buffer size B	75	60-100
index meta size k	0.05	0.01-0.1

A. Simulation Settings and Metrics

To study the performance of our approach, we use two types of traces to conduct our simulations. The first trace is generated by the ONE simulator [19]. We deploy 100 mobile nodes in a small area of a real city: Helsinki, Finland. Mobile nodes perform shortest path mapbased 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 [20]. The traces span duration of a month, from February 2014 to March 2014. There are average 100-120 taxies are in the street per day. The trace of a taxi is a sequence of positions (in longitude and latitude) tagged with timestamp. In the simulation, the virtual throwboxes are deployed in the street in every two kilometer. Note that the deployment of throwboxes may influence the delivery performance but it is out of the scope of this paper. Each mobile node has a buffer to store 5 data items, and generate a request of a random data item from the set M. After fetching a data item, mobile nodes will generate another data request immediately. At the initiate stage of the system, mobile nodes store 5 data items randomly selected from the set M and throwboxes have a short time to warm up recognizing the basic information of the whole network.

Through theoretical analysis in section IV, we can find out that system performance is affected by some key parameters: M, B, k and P_4, where P_4 is determined by data request deadline. The number of data M affects the amount of data items' copies, a larger M means there are less data copies in the network, which will reduce the hit rate of each data item. Based on the scale of mobile nodes, the network can only contain up to 500 data items (100 nodes $\times$ 5 data per node), so 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 $\overline{W} = 20000$. In order to simplify the simulation, we set different request deadlines to represent the initial benefit. Considering the size of an index file, 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 $B = 60$ to $B = 100$ because of the full connection. $m_0, ..., m_4$ only reflect the proportional relation between time consumption and transmission cost, we won't discuss it in this paper. All of the evaluation variables are shown in Table II.

In order to evaluate the effects of the index-based buffer space management algorithm, we also implement a traditional

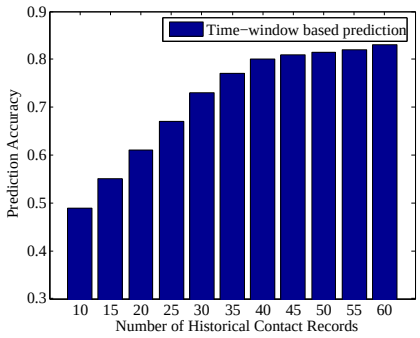


Fig. 6. Prediction accuracy under different numbers of historical records

buffer management mechanism called *data-only* buffer space management, where throwboxes use all storage space to store the data item. Once a data request is delivered to the throwboxes, throwboxes will record it and try to fetch the data from other mobile nodes. If the buffer is full, the oldest data item will be replaced by the new one.

B. Results and Discussion

Firstly, to evaluate the accuracy of the time-window based prediction, we run some tests and results are shown as Fig. 6. 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. Fig. 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, the convergence accuracy of the time-window based prediction is 80% and the convergence size of the index is 40. So, in the following simulations, throwboxes' index only records 40 latest contact information.

Then we conduct three groups of simulations to evaluate the performance of the two buffer space allocation algorithms (index-based, data-only) on the offloading ratio and data fetching delay. The offloading ratio is the proportion of successful data fetching via DTNs in all the data requests. In the first group of simulations, we set different mean initial benefits to see the effect of the initial benefit's value. Then, we vary the size of the throwbox buffer in the second group. Finally, we modify the size of index meta in the third group. In all of the simulations, we record the data fetching tpye (via DTNs or cellular networks) and delay of each accomplished data request for a certain time period and calculate an average value.

1) Effects of the Throwbox Buffer Size:

Fig. 7(a) and Fig. 9(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. The results demonstrate that about 20% of all the data requests were offloaded by mobile nodes. As the size of the throwbox buffer become larger, more data can be stored in the buffer, which means a higher hit rate of the requested data, a higher hit rate reduce chances to fetch data via cellular networks. 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. Fig. 8(a) and Fig .10(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. A lot of data requests with lower initial benefit were found, and these requests which can hardly be offloaded are finished via cellular networks immediately. Thus, the average delay reduces a lot compare to the data-only algorithm.

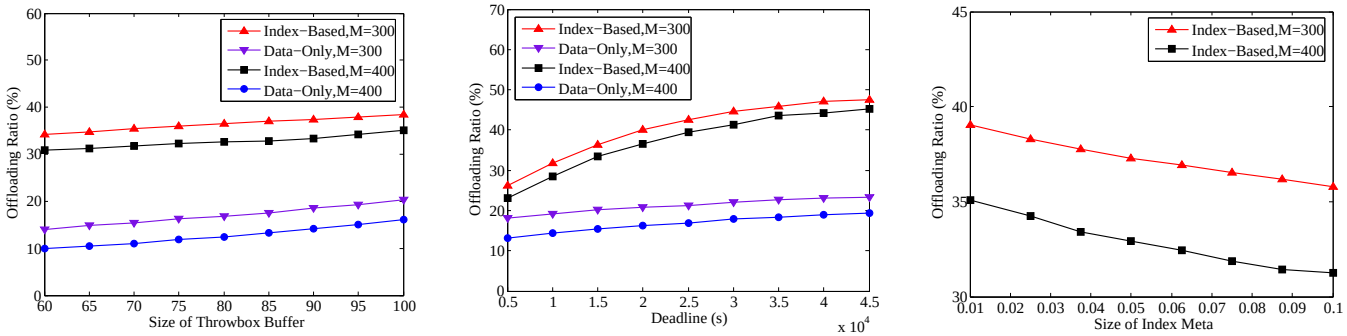
2) Effects of the Dedline:

Fig. 7(b) and Fig. 9(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. We can see that when the deadline increases, the offloading ratios of the two algorithms are all increased. But 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. This is due to the difference of the delivery model of the two algorithms. To the data-only algorithm, due to lack of network global information, data items in throwboxes buffer doesn't change since the buffer is full. A longer deadline only enlarges the possible encounter chances between mobile nodes and throwboxes. It can increase the hit rate in throwboxes buffer but the efficiency is quite low. Differ from the data-only algorithm, a longer deadline gives the throwboxes more opportunities to help requester to fetch the data from other mobile modes, which can raise the offloading ratio efficiently especially when the deadline is short. 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 Fig .8(b) and Fig 10.(b). As we can see, when the deadline is short, the gap between the two algorithms is quit 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 can't give the data-only algorithm the same benefits, so the gap becomes wider.

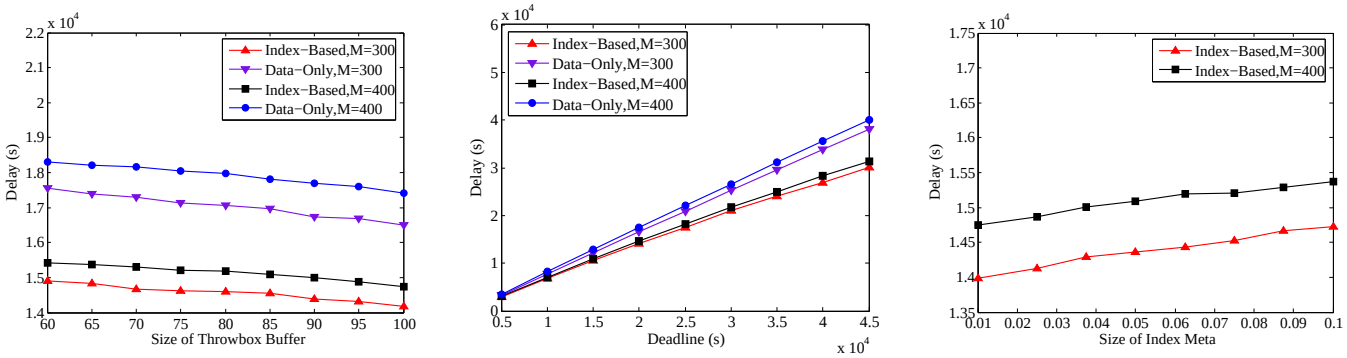
3) Effects of the Index Meta Size:

The number of data index that can add by replacing a data item is determined by the size of the index meta. Fig. 7(c) and Fig. 9(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 3 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



(a) Offloading Ratio vs. Size of Throwbox Buffer (b) Offloading Ratio vs. Deadline (c) Offloading Ratio vs. Size of Index Meta

Fig. 7. System average offloading ratio under generated traces.



(a) Delay vs. Size of Throwbox Buffer (b) Delay vs. Deadline (c) Delay vs. Size of Size of Index Meta

Fig. 8. System average delay under generated traces.

index is still useful but not that significantly. The average delay showed in Fig .8(c) and Fig .10(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. The results also give us a hint that finding a good way to minimize the size of the index meta can improve the performance of the index-based algorithms.

VI. Related Work

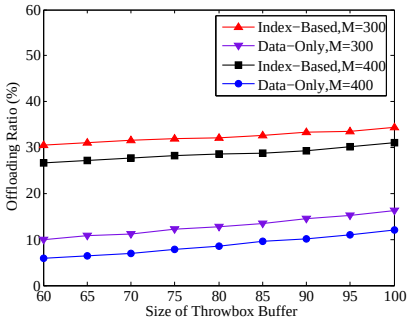
This section reviews the related work in the literature and highlights the differences among them. We mainly pay attention on two aspects of related work, including throwbox-equipped DTN and future events prediction.

A. Throwbox-Equipped DTN

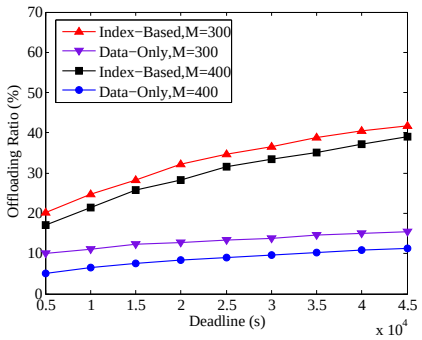
In the mobile DTN with throwboxes equipped, the existing works mainly focus on the capacity and delivery delay of the routing algorithm. Throwboxes based DTN are first proposed in [5] where the gain on the network throughput of deploying throwboxes is studied. This work use throwboxes in mobile DTN to create a greater number of encounter opportunities, consequently improving the performance of the network. In the later works [9][10], 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 [11] and designing/evaluating routing strategies [12] for throwbox-based DTN are also presented. Meanwhile, Banerjee et al. [13] consider the problem about energy efficiency of each throwbox node for throwbox-based DTN. However, all the works mentioned above treated throwboxes as fixed data buffers. 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 and this significantly reduces the total system cost.

B. Future Events Prediction

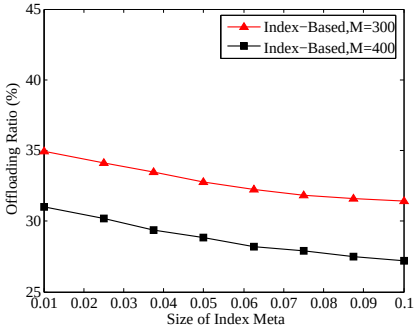
In existing prediction-based schemes, mobile nodes' mobility and contact is estimated based on a history of observations. A representative case is using utility-routing [14] [15], where each node maintains a utility value for every other node that is updated using the time between contacts. Mobile nodes consider the utility value as the predictor of two nodes' future likelihood of encounter. LeBrun et al. [16] 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. In [17], Burns et al. propose a prediction scheme that uses past frequencies



(a) Offloading Ratio vs. Size of Throwbox Buffer

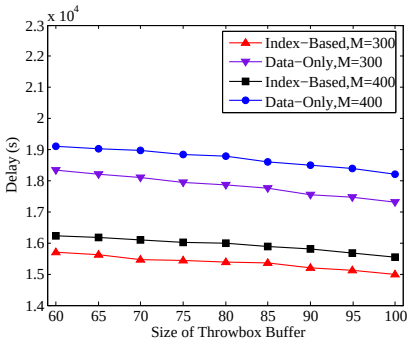


(b) Offloading Ratio vs. Deadline

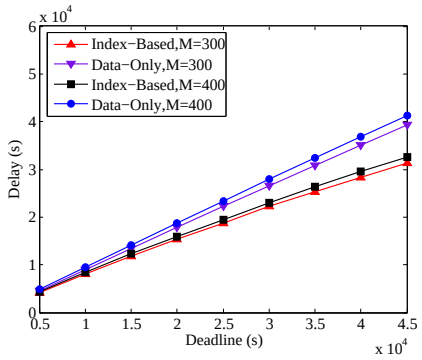


(c) Offloading Ratio vs. Size of Index Meta

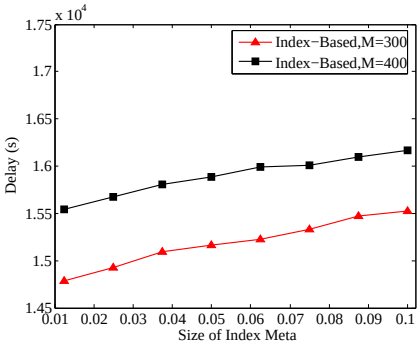
Fig. 9. System average offloading ratio under real traces.



(a) Delay vs. Size of Throwbox Buffer



(b) Delay vs. Deadline



(c) Delay vs. Size of Size of Index Meta

Fig. 10. System average delay under real traces.

of contacts, as well as the past contacts. Another prediction-based generic algorithm for DTN routing is MobySpace [18], which uses a high-dimensional Euclidean space constructed upon nodes mobility patterns. The frequency of visits of nodes to each possible location is recorded as the basis of the future distance calculation in the Euclidean space. Most of these prediction schemes focus on the contact or the geographical positions of mobile nodes. The major difference between our approach and previous works is we take the data as our target of prediction. In mobile data downloading scenario, mobile nodes don't care where the requested data is. The major concern is, when I can get it. The prediction methods mentioned above cannot effectively address this concern.

VII. Conclusion

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 aspe cts. 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.

References

[1] "Cisco visual networking index: Global mobile data traffic forecast update, 2013-2018," Feb. 2014.

[2] K. Lee, J. Lee, Y. Yi, I. Rhee and S. Chong, "Mobile data offloading: How much can WiFi deliver?," *IEEE/ACM Transactions on Networking*, vol. 21, no. 2, pp. 536-550, 2013.

[3] F. Mehmeti and T. Spyropoulos, "Is it Worth to be Patient? Analysis and Optimization of Delayed Mobile Data Offloading," in *IEEE INFOCOM*, 2014.

[4] K. Fall, "A Delay Tolerant Network Architecture for Challenged Internets," in *ACM SIGCOMM*, 2003.

[5] W. Zhao, Y. Chen, M. Ammar, M. D. Comer, B. N. Levine, and E. Zegura, "Capacity Enhancement using Throwboxes in DTNs," in *IEEE MASS*, 2006.

[6] T. Henderson, D. Kotz, and I. Abyzov, "The changing usage of a mature campus-wide wireless network," in *ACM MobiCom*, 2004

[7] L. Jeremie, F. Timur, and C. Vania, "Evaluating mobility pattern space routing for dtns," in *IEEE INFOCOM*, 2006.

[8] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, "Perfor-

mance analysis of mobility-assisted routing," in *ACM MobiHoc*, 2006.

[9] M. Ibrahim, A. Al Hanbaliand and P. Nain, "Delay and resource analysis in MANETs in presence of throwboxes," *Performance Evaluation*, vol. 64, no. 9-12, pp. 933-947, 2007.

[10] M. Ibrahim, P. Nain, and I. Carreras, "Analysis of relay protocols for throwbox-equipped dtns," in *WiOPT*, 2009.

[11] B. Gu, X. Hong, P. Wang, and R. Borie, "Latency analysis for thrown box based message dissemination," in *IEEE Globecom*, 2010.

[12] B. Gu and X. Hong, "Capacity-aware routing using throwboxes," in *IEEE Globecom*, 2011.

[13] N. Banerjee, M. D. Corner, and B. N. Levine, "Design and field experientation of an energy-efficient architecture for DTN throwboxes," *IEEE/ACM Tranactionss on Networking*, vol. 18, no. 2, pp. 554-567, 2010.

[14] A. Lindgren, A. Doria, and O. Schelen, "Probabilistic routing in intermittently connected networks," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 7, no. 3, pp. 19-20, 2003.

[15] X. Zhang, G. Neglia, J. Kurose, and D. Towsley, "Performance modeling of epidemic routing," *Computer Networks*, vol. 51, no. 10, pp. 2867-2891, 2007.

[16] J. LeBrun, C. Chuah, and D. Ghosal, "Knowledge based opportunistic forwarding in vehicular wireless ad hoc networks," *IEEE VTC*, vol. 4, pp. 2289-2293, 2005.

[17] B. Burns, O. Brock, and B. N. Levine, "Mv routing and capacity building in disruption tolerant networks," in *IEEE INFOCOM*, 2005.

[18] J. Leguay, T. Friedman, and V. Conan, "Evaluating mobility pattern space routing," in *IEEE INFOCOM*, 2006.

[19] Keranen, Ari, Jorg Ott, and T. Karkkainen, "The ONE simulator for DTN protocol evaluation," in *Simutools*, 2009.

[20] Raul Amici, Marco Bonola, Lorenzo Bracciale, Antonello Rabuffi, Pierpaolo Loreti and Giuseppe Bianchi, "Performance Assessment of an Epidemic Protocol in VANET Using Real Traces," in *Mobile and Wireless Networking*, 2014.