CEDO: Content-Centric Dissemination Algorithm for Delay-Tolerant Networks

Francisco Neves dos Santos*, Benjamin Ertl†, Chadi Barakat‡, Thrasyvoulos Spyropoulos§, Thierry Turletti‡
*École Polytechnique Fédérale de Lausanne, Switzerland
†Université de Nice Sophia-Antipolis, France
‡INRIA Sophia-Antipolis, France
§Eurecom - Communications Mobiles, Sophia-Antipolis, France
Emails: Francisco.Santos@epfl.ch, Ertl@polytech.unice.fr, {Chadi.Barakat, Thierry.Turletti}@inria.fr,
Thrasyvoulos.Spyropoulos@eurecom.fr

Abstract—**Emerging challenged networks require new protocols and strategies to cope with a high degree of mobility, high delays and unknown, possibly non-existing routes within the network. Researchers have proposed different *store-carry-and-forward* protocols for data delivery in challenged networks. These have been complemented with appropriate drop and scheduling policies that deal with the limitations of the nodes' buffers and the limited duration of opportunistic encounters in these networks. Nevertheless, the vast majority of these protocols and strategies are designed for end-to-end transmissions. Yet, a paradigm shift from the traditional way of addressing the endpoints in the network has been occurring towards content-centric networking. To this end, we present CEDO, a content-centric dissemination algorithm for challenged networks. CEDO aims at maximizing the total delivery-rate of distributed content in a setting where a range of contents of different popularity may be requested and stored, but nodes have limited resources. It achieves this by maintaining a delivery-rate utility per content that is proportional to the content miss rate and that is used by the nodes to make appropriate drop and scheduling decisions. This delivery-rate utility can be estimated locally by each node using unbiased estimators fed by sampled information on the mobile network obtained by gossiping. Both simulations and theory suggest that CEDO achieves its set goal, and outperforms a baseline LRU-based policy by 72%, even in relatively small scenarios. The framework followed by CEDO is general enough to be applied to other global performance objectives as well.**

I. Introduction

Challenged networks like tactical networks, exotic media networks (e.g. underwater, interplanetary), sensor networks, and pocket-switched networks require new protocols and strategies to handle a high degree of mobility, high delays and unknown, possibly non-existing routes between nodes in these networks [4]. Many researchers have proposed different protocols to overcome these emerging challenges within the area of delay tolerant networking (DTN) [23]. Most of the propositions imply a paradigm shift from the traditional network architecture of addressing the endpoints in the network with the current TCP/IP model towards a *"bundle"* overlay or named content network architecture. However, these new architectures also pose new challenges for example in the field of routing, security or buffer management.

To deal with the high degree of mobility and possibly unknown or non-existing paths between the source node(s) and the target node(s) in these networks, researchers have proposed routing protocols based on the *store-carry-and-forward* routing principle [9][10][18][22]. Following this principle, nodes store messages or contents in their buffers and carry them until an opportunity to forward or replicate them arises. These routing protocols disseminate data in an epidemic, probabilistic, or utility-based manner, with recent protocols trying to exploit known and measured patterns in human mobility in a sophisticated manner [25].

Nevertheless, such long-term storage and heavy replication impose a heavy toll on the resources of the nodes involved. While the focus of existing routing protocols has been on the selection of the right next hop(s), the performance of these protocols is tightly coupled to the storage capacity of the nodes in the network as well as the amount of bandwidth available during a contact with another node. It is thus key to implement efficient buffer management policies for dropping data upon nodes' buffers saturation and scheduling it for transmission upon nodes' encounters [13]. To this end, researchers have studied the effect of simple policies such as *Drop Last*, *Drop Front*, *Drop Oldest*, etc. [24]. Furthermore, a theoretically optimal buffer management policy, along with an efficient distributed implementation for it have been proposed in [14].

However, in the majority of these works, the buffer management and scheduling problem is formulated and solved in the context of *end-to-end* traffic sent by one device to another well defined device. Yet, an increase in user demand for content has been observed, due to multimedia streaming and data sharing. This has led the Internet community to shift towards content-centric architectures that care less about addressing specific endpoints and more about the content and data themselves (e.g. CCN [9][23]). Wireless and opportunistic networks are no exception to this trend, with cellular traffic being dominated by content related to streaming and social applications [15]. It becomes of prime importance to be able to inject new content and rely on network architecture to forward it to all nodes that request it. Yet, given the increasing abundance and size of content available, and the limited, in comparison, resources of handheld nodes, interesting questions arise in this context, especially when the number of requesting nodes varies over time and from one content to another: *How can one optimize (or even define) the global performance of such a network? How can one serve content requests in a "fair" manner?* While such questions are not new, when it comes to content-centric approaches, answering them in the context of DTNs poses interesting new challenges. As a result, developing efficient drop and scheduling policies for content-centric delay-tolerant networks is an important research objective [7][8][20][21][25].

To this end, in this paper we formulate and propose efficient content-delivery strategies for Delay Tolerant Networks (DTN). We consider a setting where different contents of varying popularity exist in the network, and where nodes can store copies for only a small subset of them due to storage limitations. This turns the scheduling and drop problem into a storage allocation problem. Our main contribution in this paper is to study this problem in depth, in the context of DTN, proposing both optimal and practical (i.e. implementable) solutions. Specifically, our contributions can be summarized as follows:

- We formulate the centralized optimization version of the problem and derive the *theoretically* optimal allocation that maximizes network-wide content hit rate as a function of content popularity distribution, number of nodes, rate of mobility, and nodes' buffer capacity.
- We use the above formulation to derive appropriate utilities per content, that can be used *locally* by each node to make drop and scheduling decisions; this distributed implementation of the optimal policy, based on content utilties, can be shown to converge to the globally optimal one.
- In order to obtain global quantities (e.g. popularity, measured as request rate) needed to calculate a content's utility, we propose appropriate (*unbiased*) estimators. This leads to our practical policy for the problem in hand, where nodes: (a) estimate key quantities (per content), (b) use them to derive the utility of a content (i.e. marginal gain from keeping or replicating it), and (c) use the utilities to choose which content to drop or forward, when encountering other nodes. This sequence of actions, while not provably optimal at all times, is designed to closely resemble the optimal actions *in the average sense*.
- Finally, we have implemented our distributed solution, called CEDO, within the CCN architecture [9], [3] and evaluated its performance using different simulation scenarios in NS-3 [6]. CCN is a promising architecture for content-centric networking and provides a set of building blocks that allow a smooth support of CEDO. Our first results suggest that, even in small networks, CEDO outperforms baseline policies, as for example $60-72\%$ higher throughput than with LRU (Least Recently Used), in different mobility scenarios.

In the following section, we describe our solution CEDO and the storage allocation problem behind. Section III discusses the implementation of CEDO and explains the details of its integration with the CCNx architecture. Simulation results over synthetic mobility traces are presented in Section IV. Section V concludes the paper with some perspectives on our future research.

II. Content-Centric Dissemination Algorithm (CEDO)

We cast the problem of scheduling and drop as a network-wide content allocation problem where the question is *how nodes should replicate contents so that the network throughput is maximized.* We formulate the problem as follows. Suppose there are L nodes in the network, each of which has a buffer of capacity B, which can be used to store contents. We assume that each node generates at random times requests for random contents, and that each such request has some "deadline" (denoted as TTL) after which it disappears from the requesting node and the network. Whenever a node E comes within transmission range of node F, and is interested in a content available in the buffer of F, E can retrieve the content. If E successfully retrieves the content before the deadline (i.e. less than TTL time units since it issued the request), we say that *the request is satisfied*. If not, then there is a *miss*.

It is clear from the above discussion that, the more nodes store a copy of a given content, the faster on average can this content be found by an interested node (with the actual statistics depending on the mobility model). However, if we assume that many different contents exist in the network, then each node can use its buffer space to store only a small subset of them. Hence, having more copies for content X in the network implies fewer copies for some other content Y, which creates a tradeoff behind the allocation problem we are interested in.

Furthermore, we assume that different contents might have different request rates, as some might be more popular than others. This is highly realistic as skewed popularity distributions have commonly been observed in a variety of contexts [2][5][19]. In this case, intuition suggests that more popular content should be allocated more buffer space. But how much? To what extent and under which network conditions can popular contents "kill" less popular ones?

Next, we first elaborate on our assumptions and formulate an optimization problem that aims at maximizing the long-term rate of satisfied requests (referred to as "delivery rate" hereafter), in the setting described above (Section II-A). We then use this formulation to come up with a distributed solution for the problem: this solution is based on the notion of *utility per content* that each node can calculate locally and use to rank contents it stores. In this case, the drop strategy is simply to remove contents with minimum utilities and the scheduling strategy is to forward contents to encountered nodes by decreasing order of utilities (Section II-B). Nevertheless, these utilities require the knowledge of content-related metrics (e.g. popularity), which are not available locally. Thus, as a final step, we show how each node can estimate these quantities locally so as to derive reliable estimates of the optimal utilities (Section II-C).

A. Formulation of the Optimal Content Allocation Problem

For tractability reasons, we consider that nodes' inter-meeting times are IID, following an exponential distribution with a constant parameter λ; in other words, if node E meets node F, it expects to meet F again after $1/\lambda$ seconds. This assumption is commonly made in related work, and has also received some support (as a useful approximation) by recent trace analysis [11]. To further validate our policies, we will use in Section IV a simulation scenario that greatly departs from this assumption. With the above assumption, the probability that a receiver obtains a replica of content i can be easily derived as (see e.g. [13]):

Lemma 2.1: Given the node meeting rate λ, the average lifetime for a content request TTL, and the number of replicas of content i, $n^{(i)}$, the probability that a node obtains a copy of content i is: $p^{(i)} = 1 - exp\{-\lambda TTL \cdot n^{(i)}\}$.

Hence, by duplicating content i (i.e., increasing $n^{(i)}$), relaxing the request deadline TTL, or augmenting the meeting rate λ, we improve the likelihood $p^{(i)}$ that a device obtains a copy of content i.

The above probability refers to the success of a *single* request for content i. If we look at *all* requests for content i, we can define its delivery rate $DR^{(i)}$ that measures the number of replicas of this content successfully delivered to the intended destinations per unit of time:

Definition 2.2: Given the request rate for content i, $q^{(i)}$, and the probability of receiving a replica of content i, $p^{(i)}$, we define the delivery rate for content i as $DR^{(i)} = q^{(i)} \cdot p^{(i)}$.

Our framework, CEDO, attempts to maximize the total delivery rate, which is the sum of the delivery rates of all contents, subject to the following constraints: (i) the total number of replicas of all content cannot exceed the amount of storage of all nodes, (ii) a node cannot store duplicate copies of a content i, and (iii) there must exist at least one replica of each content i in the network. This corresponds to the following optimization problem:

$$\begin{aligned} \underset{n^{(1)},\dots,n^{(k)}}{\text{maximize}} \quad & f(n^{(1)},\dots,n^{(k)}) = \textstyle\sum_{i=1}^{k} DR^{(i)} \\ \text{subject to} \quad & \textstyle\sum_{i=1}^{k} n^{(i)} - L.B \leq 0, \\ & n^{(i)} - L \leq 0 \quad \text{for all content } i, \\ & n^{(i)} \geq 1 \qquad\quad \text{for all content } i. \end{aligned} \tag{1}$$

k is the number of contents, L is the number of mobile nodes and B is the buffer capacity of each node.

Given that the constraints in (1) are linear [13], that the objective function is concave, and assuming that $n^{(1)}, \dots, n^{(k)}$ are real random variables, we get a convex optimization problem that is tractable and has a unique solution in terms of the number of replicas per content [13]. A key difference compared to the problem in [13] (end-to-end case) is that, in addition to the success probability of a request, the delivery rate per content is now also directly proportional to its popularity value. This will push the optimal algorithm towards an unbalanced allocation of resources ($n^{(i)}$) to contents, favoring popular contents (as expected). Yet, the concave nature of the delivery probability implies *diminishing returns* when further and further turning the balance in favor of more popular contents. The optimal allocation is thus the tradeoff between these two "forces".

B. Deriving per Content Utilities

While the above problem leads to the optimal allocation, in the context of DTNs, we cannot access all the buffers of all nodes in parallel and in one shot, choosing exactly $n^{(i)}$ nodes to store content i, $n^{(j)}$ nodes to store content j, etc. Instead, each node is faced with a *local* decision, whenever it encounters another node: it has to decide which contents to forward and which contents to drop, if its buffer is full and the encountered node has new contents to offer.

Since the above problem is convex, it can be generally solved with a *gradient ascent* strategy. CEDO allows for a distributed implementation inspired by [13]. When a given allocation exists in the network, taking the gradient of the objective function w.r.t the number of replicas, $n^{(i)}$, leads to a sum of the marginal gains in the total delivery rate with respect to increasing (or reducing) the number of replicas, $n^{(i)}$, for each content i. Hence, this marginal gain is the *utility of content* i with respect to the problem of maximizing the delivery rate. Each node can then use these utilities to *locally* compare different contents between which it must make a drop or transmission decision. This leads to a distributed implementation of the global algorithm that can be proven to converge to the optimal solution.

Definition 2.3: Given the delivery rates for all available contents, $DR^{(1)}, \dots, DR^{(k)}$, the delivery-rate utility for a content i is defined as:

$$U^{(i)} = \frac{\partial}{\partial n^{(i)}} \sum_{j=1}^{k} DR^{(j)}. \tag{2}$$

Using this definition of the utility and the expression of the delivery rate given above, we can express $U^{(i)}$ in a more convenient form, as explained next.

Lemma 2.4: The delivery-rate utility is proportional to the request-rate for content i, $q^{(i)}$, and decays exponentially with the number of replicas $n^{(i)}$:

$$U^{(i)} = q^{(i)}(\lambda TTL) exp\{-\lambda TTL \cdot n^{(i)}\}. \tag{3}$$

The above result follows easily by taking the derivative of the expression for $DR^{(i)}$ given by Definition 2.2 and Lemma 2.1. In other words, as nodes replicate a particular content i, the marginal gain in the delivery rate of i is gradually smaller; as $n^{(i)}$ tends to infinity, $U^{(i)}$ tends to zero. This effect allows less popular contents to not "starve", provided there is some minimum buffer space available. We will return to the issue of starvation and fairness later.

With these utilities in hand, the *optimal policy* for each node is simply (a) to drop the contents with the minimum utility, when its buffer is full, and (b) when it encounters another node, to replicate the contents with the maximum utility (among the ones that the latter node does not already have). Consequently, the optimal policy essentially attempts to equalize the utilities of all contents: according to Eq. (3), action (a) reduces the number of replicas $n^{(i)}$ of content i thus increasing its utility; action (b) increases the number of replicas $n^{(i)}$ of content i thus decreasing its utility.

The following lemma converts the per-content utility into a different form that: (a) provides us with additional insight as to what the optimal allocation is really aiming at, and (b) will allow us to derive efficient estimators for these utilities (in the next section).

Lemma 2.5: The delivery-rate utility for content i, $U^{(i)}$, can alternatively be expressed as:

$$U^{(i)} = \lambda TTL(q^{(i)} - DR^{(i)}). \tag{4}$$

Proof. We can rewrite (3) as follows:

$$\begin{aligned} U^{(i)} &= q^{(i)}(\lambda TTL)exp\{-\lambda TTL \cdot n^{(i)}\}, \\ &= q^{(i)}(\lambda TTL)(1-(1-exp\{-\lambda TTL \cdot n^{(i)}\})), \\ &= \lambda TTL(q^{(i)} - q^{(i)} \cdot p^{(i)}), \\ &= \lambda TTL(q^{(i)} - DR^{(i)}). \end{aligned}$$

Eq. (4) suggests that a content's utility is in fact proportional to the miss rate, defined as the number of unsatisfied requests for content i per second: $MR^{(i)} = q^{(i)} - DR^{(i)}$. Since the optimal policy, as we saw, attempts to equalize utilities in order to solve the optimal allocation problem, this is in fact equivalent to equalizing the miss rate of contents, irrespective of their request rate or popularity.

This latter interpretation allows to have a practical implementation of the optimal policy, that we call CEDO. CEDO essentially estimates miss rates of contents and manages buffer and scheduling accordingly. When a device's buffer is full, CEDO discards the content replica with the smallest rate of unsatisfied requests; by definition, this replica has the smallest impact on the total delivery rate. When a node meets another node, it sends first the content with the highest rate of unsatisfied requests; by replicating this item, the device will increase the probability that a receiver obtains a copy of this content. This gradient ascent strategy, illustrated in Figure 1, shall converge to the number of replicas per content that maximizes the objective function f in (1), provided the iteration step is such that CEDO reaches the efficient allocation before the network changes substantially [13]. Note that the iteration step is decided by the speed at which nodes move. For a proper functioning of CEDO, there should be enough node encounters before the traffic demand changes.

Although we now have a distributed implementation of the optimal policy, miss rates are *global* quantities (one needs to know every request made and whether this was successful), but need to be known locally by each node. Hence, miss rates must be estimated, in an efficient manner, by each node in order to calculate (or rather estimate) utilities and apply the above algorithm. This brings us to the final, and important, component of CEDO.

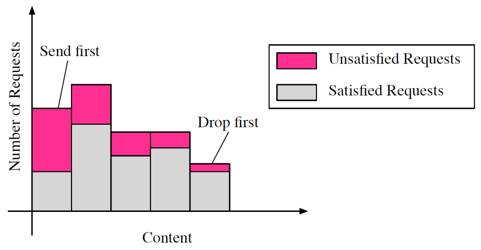


Fig. 1: CEDO strategy for scheduling and drop as a function of content popularity and content miss rate.

C. Estimating the delivery-rate utility

To calculate the miss rate for a content i (as explained earlier the utility is proportional to the miss rate), a node must estimate the corresponding request and delivery rate. CEDO performs this estimate by requiring each node to record the time when the application running on it requests content i and when the corresponding request is fulfilled. This provides a first local estimation. To refine this local estimation, the node merges its own rate estimate with those received from other encountered nodes. Using the two approximate values for the request and delivery rate, the node computes the utility of content i by applying (4). Next, we define our unbiased estimators for $q^{(i)}$ and $p^{(i)}$. Without loss of generality, we consider in the sequel their normalized values with respect to the network size.

Definition 4 *Let $X_n^{(i)}$ be a random variable denoting the n-th measurement of the number of requests for content i per node, issued by the n-th encountered node in the past W seconds, such that $E[X_n^{(i)}] = q^{(i)}W$. Then our estimator for the request rate for content i per node, $q^{(i)}$, is given by*

$$\hat{q}_n^{(i)} = \alpha \hat{q}_{n-1}^{(i)} + (1-\alpha)X_n^{(i)}/W, \tag{5}$$

where $\hat{q}_n^{(i)}$ is the value of the estimator $\hat{q}^{(i)}$ after considering the n-th node meeting and $\alpha \in (0,1)$ is the weight given to the old estimate of the request-rate for content i, $\hat{q}_{n-1}^{(i)}$.

In other words, the estimator for the request rate of content i is computed as the exponentially weighted moving average (EWMA) of the observed samples $X_0^{(i)}, X_1^{(i)}, \ldots, X_n^{(i)}$, applying a weight of $(1-\alpha)$ to the most recent sample, $X_n^{(i)}$. This allows on one hand to average over different measurements, and on the other hand to forget about the very past ones for reactivity reasons. One can easily show that this estimator is consistent and unbiased, i.e., $E[\hat{q}^{(i)}] = q^{(i)}$, meaning that its average over all nodes approaches the real popularity and it moves towards the real popularity as long as we add more samples.

In a similar way, we define the estimator for the delivery rate of content i per node, $DR^{(i)}$.

Definition 5 *Let $Y_n^{(i)}$ be a random variable denoting the n-th measurement of the number of satisfied requests for content i issued in the past t seconds, where $t \in [TTL, W+TTL]$, and such that $E[Y^{(i)}] = p^{(i)}q^{(i)}W$. Then our estimator for* *the delivery rate of content i, $\widehat{DR}^{(i)}$, is given by*

$$\widehat{DR}_n^{(i)} = \beta \widehat{DR}_{n-1}^{(i)} + (1-\beta) Y_n^{(i)}/W, \tag{6}$$

where $\widehat{DR}_n^{(i)}$ is the value of the estimator after considering the n-th node meeting and $\beta \in (0,1)$ is the weight given to the previous estimate of $DR^{(i)}$.

In other words, the estimator for the delivery rate of content i is computed as the exponentially weighted moving average of the observed samples $Y_0^{(i)}, Y_1^{(i)}, \ldots, Y_n^{(i)}$, applying a weight of $(1-\beta)$ to the most recent sample, $Y_n^{(i)}$. Here we shift back the time window by TTL seconds to allow the content requests issued in the last TTL seconds to receive an answer or, if none is received, to be classified as "unsatisfied". As the previous estimator for request rate, this one is also consistent and unbiased, i.e., $E[\widehat{DR}^{(i)}] = DR^{(i)}$.

In addition to refining the estimators for request and delivery rates using local measurements at other encountered nodes, a mobile node might also have the possibility to combine its estimators with those at other encountered nodes, if available. This allows local estimates to permeate the network. If two estimates for the same content i are available at two nodes that meet, CEDO combines them as a weighted sum with the weight of each estimator being inversely proportional to its variance [12][13].

III. Algorithm implementation

We present hereafter details on the implementation of our algorithm and the main functions and building blocks required for its support. We start by a general description of the implementation, supported by pseudo code for the different modules (Modules 1-5), then move into a specification to the CCNx architecture [3][9]. CCNx is a promising solution for content-centric networking that provides a set of functions allowing a smooth support of CEDO.

A. Outlining the algorithm

Our content-centric dissemination algorithm resides between the application and the network layer: it receives interest messages from the application and attempts to find matching content cached in the mobile device's buffer or available from a neighbouring node. When CEDO fails to find matching content in the local buffer, it writes the query to the pending interest table of the underlying CCNx protocol [3] until it can satisfy the request. Periodically, each mobile device issues a hello message (HM) that signals its presence. When CEDO captures a hello message, it prepares a message with all its pending queries, known as the pending interests message (PIM). A mobile node within range attempts to satisfy the pending requests by searching its local storage and sending all matching content. CEDO relies on a node's ability to estimate the request and delivery rate, $\hat{q}^{(i)}$ and $\widehat{DR}^{(i)}$, for each content i, in order to compute the delivery-rate utility for this content, $U^{(i)}$, as stated in the previous sections. To reduce the load on nodes and avoid them storing and continuously updating estimators for different contents, we assume that each content i carries in its header the estimates for the two rates, which are initialized by the content origin server and updated by intermediate nodes thanks to gossiping. When a mobile device's buffer is full, CEDO uses the content utility to erase the least useful content from the memory. Similarly, when exchanging content with a neighbour, it prioritizes transmissions by decreasing utility. This ensures that the most useful content reaches the other node in case the connection stops before all requested content is transmitted. In the following, we will present the different steps performed by the algorithm.

Module 1 Processing interest messages received by the application at a device

```
Require: interest for content $i$ from application
1: record time of request for $i$ in statistics table
2: if buffer contains $i$ then
3:     deliver content $i$ to application
4:     record time when request for $i$ satisfied in statistics table
5: else
6:     write interest to pending interest table
7: end if
```

Module 2 Processing hello messages

```
Require: hello message from network
1: if a hello message is waiting to be sent then
2:     delay transmission of hello message
3: end if
4: create new pending interests message
5: for each interest in the pending interest table do
6:     write interest to pending interests message
7: end for
8: schedule transmission of pending interests message
```

Module 3 Processing pending interests messages

```
Require: pending interests message from network
1: if a pending interests message is waiting to be sent then
2:     delay transmission of pending interests message
3: end if
4: for each interest in the pending interests message do
5:     if exists matching content $i$ in buffer then
6:         write content $i$ to list
7:     end if
8: end for
9: sort list in order of decreasing DR utility
10: for each content $i$ in list do
11:     write request and delivery rate to content $i$
12:     schedule transmission of content $i$
13: end for
```

1) Processing application queries: When the application issues a content query, expressed as an interest message (IM), CEDO records the time at which it receives the interest in the statistics table and searches the mobile device's buffer for matching content. The statistics table is a data structure, independent of the content buffer, that stores the time when requests are issued and when they are satisfied, from which CEDO calculates the request and delivery rate of each content seen. If the buffer contains the desired content, CEDO delivers this data immediately to the application; otherwise, the device stores the query in the pending interest table. The pending interest table is a data structure that stores all the unsatisfied queries issued by the application. Note that the interest remains in the pending interest table until CEDO retrieves this data from an encountered node. Periodically, CEDO iterates through the statistics table to update the request and delivery rate of all content seen using Equations (5) and (6).

A mobile device cannot satisfy all content requests with its cache alone; in such a case, the device stores the query in the pending interest table while waiting for a response. Periodically, a device emits a hello message, which serves to alert other nodes of its presence; the delay for transmitting the next hello message, d_{hello}, is calculated as follows:

$$d_{\text{hello}} = d_{\text{hello}}^{\min}(b_{\text{hello}} + r), \tag{7}$$

where $d_{\text{hello}}^{\min}$ is the minimum delay between hello messages, $b_{\text{hello}} \in [0, b_{\text{hello}}^{\max}]$ is the hello backoff-counter, $b_{\text{hello}}^{\max}$ is the maximum hello backoff, and $r \in [0,1]$ is a uniform random value. By randomizing the transmission delay of the hello message, CEDO reduces the probability of interfering with ongoing but yet undetected transmissions. When a mobile device receives a hello message, it delays the transmission of its own message by increasing the backoff counter, b_{hello}, as shown in lines 1-3 of Module 2; this avoids congesting the network with an excessive amount of hello messages. As a response to a hello message, CEDO prepares a pending interests message, comprising requests in its pending interest table, and schedules the transmission of the message to all devices within range; see lines 4-8 of Module 2.

Module 4 Processing incoming content

```
Require: datagram with content $i$ from network
 1: if a pending interest exists for content $i$ then
 2:     deliver content $i$ to application
 3:     record time when request for $i$ satisfied in statistics table
 4: end if
 5: average request-rate in statistics table and rate in datagram
 6: average delivery-rate in statistics table and rate in datagram
 7: if buffer contains $i$ then
 8:     discard content $i$
 9: else
10:     store-content(i)
11: end if
```

CEDO computes the transmission delay of the pending interests message, d_{PIM}, as following:

$$d_{\text{PIM}} = d_{\text{PIM}}^{\min}(b_{\text{PIM}} + r), \tag{8}$$

where $d_{\text{PIM}}^{\min}$ is the minimum waiting time before transmitting the pending interests message (PIM), $b_{\text{PIM}} \in [0, b_{\text{PIM}}^{\max}]$ is the PIM backoff-counter, bounded by 0 and the maximum backoff $b_{\text{PIM}}^{\max}$, and $r \in [0,1]$ is a random value. When a node receives a pending interests message, it defers the transmission of a possibly pending PIM by increasing the backoff-counter, b_{PIM}, see lines 1-3 of Module 3; this enables a mobile device to satisfy a part or all of its own pending interests by overhearing the upcoming data exchange, thus reducing the number of entries in its next PIM. To process an incoming PIM, a node retrieves from its buffer all the content items that satisfy the interests expressed in the message, see lines 4-8 of Module 3. It then orders the sought content by decreasing utility order and updates their estimated request and delivery rates, see lines 9-13 of Module 3.

Module 5 Storing content in the buffer

```
 1: function store-content($i$)
 2: if buffer not full then
 3:     write content $i$ to buffer
 4:     return true
 5: end if
 6: let $m \leftarrow$ content with lowest DR utility in buffer
 7: if $i$'s utility $>$ $m$'s utility then
 8:     replace replica $m$ with $i$ in buffer
 9:     return true
10: end if
11: discard content $i$
12: return false
13: end function
```

2) Processing incoming content: Whenever a mobile device receives a content object, it checks the pending interests table for interests corresponding to this data; in case the content object matches an interest in the table, CEDO delivers the data to the application and records the time at which the interest was satisfied, see lines 1-4 of Module 4. As each mobile device may have its own estimate of the request and delivery rates, CEDO computes the arithmetic mean of the two estimates and updates the statistics table, as shown in lines 5-6 of Module 4. Finally, CEDO caches the replica of content i if no such copy exists in the device's buffer, see line 10 of Module 4.

The function **store-content()** implements CEDO's buffer management strategy, see Module 5. It accepts to store all unique content published by the user and replicas of other content received from the network as long as there is enough storage capacity, see lines 2-5 of Module 5. Whenever the buffer is full, the function selects the replica with the lowest delivery-rate utility from memory, say m, and compares it with the utility of replica i; it keeps only the replica with the highest utility from the pair $\{i, m\}$, see lines 6-12. By definition, the replica with the lowest utility has the lowest impact on the total content delivery-rate, if dropped.

B. Integration within CCNx

We implemented CEDO on the PARC CCNx implementation [3], version 0.6.0, which is based upon the Content-Centric Networking (CCN) architecture [9]. This open source software reference implementation is available under the GNU General Public License version 2. It is divided into two main components: (i) the client, which acts as a mediator between the server and the application and provides an interface for the application to issue interests and obtain content objects from the network, and (ii) the server, which propagates interests to the network and obtains content objects from other servers. The main changes required to deploy CEDO over CCNx consist in modifying the header of the content object to include the request and delivery rate estimates, disabling the propagation of interest messages to network peers, and introducing two control packets that implement the *hello* and *pending interests* messages.

A content object contains four units: `Signature`, `Name`, `SignedInfo` and `Content`, as illustrated in Figure 2. The `Signature` contains a digest of the `Name`, `SignedInfo` and `Content` components, computed using a signature algorithm [3]; the `Name` unit holds the hierarchical CCNx name of the content; the `SignedInfo` component specifies the signer of the message, the time when the object was signed, the type of content conveyed and the message's expiration time.

The original CCNx server stores each content object received from the network in a hash table. For each content replica received, the server divides the content object into two parts: (i) the first part comprises the `Signature` and `Name` units and (ii) the second consists of the remaining components of the content object. The server computes a hash value of the first part of the content object to index each replica stored in the buffer. We introduce the estimates of the request and delivery rates as two integer fields in the `SignedInfo` section of the message. Because the `SignedInfo` section belongs to the second half of the content object, recomputing the two rate estimates for each stared replica does not affect its hash value; hence, the server can still locate content in the buffer after updating the rate estimates of the stored replicas.

A message in the CCNx protocol is encoded using the CCNx binary (CCNB) format [3]. Each section of a message is labelled by a `DTAG`, which is an unique identifier drawn from an internal dictionary. The parser or decoder outputs the byte number at which each message element starts and ends. We encode each estimate as a four-byte `BLOB`, representing an integer value, enclose each estimate with a `DTAG` and modify the content-object parser so as to locate the boundaries of the two values. By using `BLOB`s of fixed length, we can recompute the estimates for the two rates, encode them using the CCNB format and replace the old values in the content object without generating a new message-header. Figure 2 illustrates the components of the content object, including the section where we encode the two estimated rates.

To compute the value of the two rates, we added a statistics table to the server that records the history of content requests; by storing content requests as a separate data structure, we ensure that its lifespan is independent of other server events, such as the removal of stale content from the buffer. Each table entry contains a list of all requests issued and satisfied in the preceding $W + TTL$ seconds for a specific content, where W denotes the measurement period and TTL is the maximum lifetime of an information request; each entry is indexed by the CCNx name of the content being tracked. To calculate the value of the request-rate estimator, $\hat{q}^{(i)}$, we apply (5) and measure $X_n^{(i)}$ by counting the number of requests sent in the preceding W seconds recorded in the content-requests table. We estimate the delivery rate, $\widehat{DR}^{(i)}$, using (6), and evaluate $Y_n^{(i)}$ by tallying all the satisfied requests transmitted in the preceding $[TTL, W+TTL]$ seconds. Knowing the two estimates, $\hat{q}^{(i)}$ and $\widehat{DR}^{(i)}$, we determine the utility of content i using (4) and record this value alongside each replica stored in the buffer. Whenever the buffer exceeds a storage threshold, which is controlled by the parameter `CCND_CAP` [3], we sort the buffer in order of increasing utility and discard the least-useful replicas until the buffer size is below the storage limit.

Finally, we implement CEDO's hello and pending interests messages as content objects. The hello message consists of a content object with `Name` set to `/CEDO/hello`, the `type` field set to `CCN_CONTENT_HELLO`, and having an empty `Content` section. A pending interests message has a CCNx name of `/CEDO/pit`, a `type` field set to `CCN_CONTENT_PIT` and the `Content` section of the message holds a list of the content-names requested by the application and stored in the source node's pending interests table; this message is generated as a response to a hello message. Upon receiving a pending interests message, CEDO reads each name entry in the pending interests message's payload and locates matching content in the device's memory; the algorithm sorts all matching content by decreasing utility before scheduling each transmission. We use CCNx's built-in scheduler to implement the transmission delay equations (7) and (8) for the pending interests message and the hello message, respectively. Additionally, we rely on CCNx's procedure for sending data that introduces an additional random delay before each transmission; this helps reducing the likelihood of collision with ongoing communications.

The CCNx's forwarding-information base comprises a list of IP addresses for dispatching interest messages for which there is no matching content in the local content-store. Because we use the hello and pending interests messages for a similar purpose, we disabled the interest-forwarding procedure found in CCNx. However, we use the forwarding-information base table to store a default entry with CCNx name "/" that points to CEDO's multicast address; a device always uses this address to send data to other devices.

IV. Evaluation

In this section, we aim to verify if CEDO exhibits the behavior predicted in Section II for the optimal policy. The practical implementation of CEDO is based on estimated

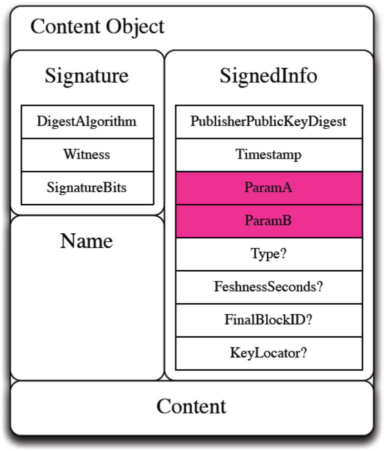


Fig. 2: Structure of the CCNx content object [3][9]. Fields with ”?” are optional. Red fields, param-A and param-B, carry the request and delivery rates.

values of the content utilities (rather than actual ones), and also involves a number of additional implementation steps. It is thus important to examine the extent to which these factors affect the desired behavior. To this end, we first check whether CEDO succeeds in equalizing the delivery-rate utility of disseminated content, which, as explained in Section II, is what an optimal policy should be doing. We then compare our policy to a baseline policy, to evaluate the potential performance benefits.

A. Simulation Setting

We simulate a mobile delay-tolerant scenario using the network simulator NS-3 [6] and use the direct-code execution (DCE) feature [16] to deploy the same version of CEDO conceived for real mobile-devices on the simulated nodes. Although this simulation method poses a significant performance penalty (in terms of simulation speed), we believe it increases the credibility of our results and facilitate a future deployment of CEDO on real devices.

We have based our simulations on two mobility models: (i) the simple *Random Direction model (RD)*, which has approximately exponentially-distributed inter-meeting times [13], consistent with our assumptions, and (ii) the *Self-Similar Least-Action Human Walk (SLAW)* synthetic mobility model, whose inter-contact times follow a truncated power-law [17]. This latter model thus departs from our assumption, but has been shown to better capture inter-contact times and contact durations observed in a large number of real mobility traces. The mobility trace file from each model was generated with the *BonnMotion* mobility scenario generation tool according to the simulation parameters in [1]. Finally, all mobile devices communicate using IEEE 802.11b radios with a PHY data rate of 1 Mbps and a transmission range of 50 to 100m.

We further assume that each node publishes one or more content items and requests one or more content items published by other nodes (the exact numbers are random) with a given randomized rate for requests, to avoid synchronous queries. We model the popularity of content according to a Zipf distribution with parameter $\alpha = 0.8$ (this is consistent with estimated values from measurements reported for web content popularity [2][5][19]). Hence, each new request issued by a node will be for a specific content i with a probability that is defined according to this distribution. We also assume that published content does not expire (only requests for it do), and that there always exists at least one replica for each content in the network (at the node publishing it). For each content, we collect the number of requests and the number of satisfied requests and average the data gathered over independent trials.

TABLE I: Simulation parameters.

Parameter	Value
Simulation time	4,000s
Network size	50 nodes
Deployment field	500 m x 500 m
Mobility model	random-direction, SLAW
Node speed	2 m/s (constant)
Node travel direction	changes every 2s
Content published per node	1 content item
Content requested per node	15
Content request lifetime	30s
Radio	IEEE 802.11b, PHY rate = 1 Mbps
Transmission range	50 m, 100 m
Data collected	number of requests per content
Number of Trials	10 independent test runs

Finally, regarding the actual implementation, each node runs an instance of the CCNx server process which contains the implementation of CEDO. We model the behaviour of the user’s application by generating events from within the simulation script. Initially, each application calls the CCNx client process to create a default entry in the forwarding-information base, named ”/”, that points to the CCNx multicast address `255.1.2.4` and multicast port `9695` through which all inter-device communication takes place. Because content does not expire, each application publishes content once, at the beginning of the simulation, and stores it permanently in the node's buffer. The application requests content by generating an interest message with the name of the desired content. All processes interchange data using Unix sockets and all nodes transmit data using the UDP transport protocol running over an IP network.

All simulation parameters are listed in Table I. The source code of CEDO along with the simulation scripts are also available in the public domain[1] so as to be able to reproduce the figures presented in this paper.

B. Results

In our first scenario, captured in Figure 3, we consider a mobility generated by the Random Direction model and a wireless range of 100m. The figure shows the total number of requests (blue), the number of satisfied requests (green) and the number of unsatisfied requests or miss rate (red) for each content, listed in order of decreasing popularity. The first interesting observation is that there is no correlation between the number of unsatisfied requests per content, and the content popularity. This is the desired behaviour that maximizes the total delivery rate, as suggested by our previous analysis. The second observation is that more requests are satisfied for more popular contents, as also expected. While this is indeed the behavior consistent with optimizing the *total* delivery rate, this also implies that, when resources are quite limited in relation to the amount of different contents, low popularity contents might suffer significantly when it comes to the percentage of their satisfied requests. This is better captured in Figure 3b, where we plot the relative number of unsatisfied requests over the total number of requests for a given content, in other words the probability of satisfying a request for content i.

Digressing slightly, we could modify the objective function f in Equation (1), to achieve fairness in relative terms:

$$f(n^{(i)}, \dots, n^{(k)}) = \sum_{i=1}^{k} \frac{DR^{(i)}}{q^{(i)}}, \tag{9}$$

and derive the utility function for each content i as

$$U^{(i)}(DR/q) = \frac{\partial}{\partial n^{(i)}} f(n^{(i)}, \dots, n^{(k)}). \tag{10}$$

In this case, we would expect the plot in Figure 3b to exhibit a constant line, implying the same success probability for each content, regardless of its popularity. This is a completely different objective than our original one aiming to maximize the network throughput. Clearly, a number of intermediate policies in between these two extremes could be derived, by manipulating the objective function between Eq. (1) and (9). We defer this investigation to future work.

In Figure 4, we turn our attention to a network which is more "challenged", where we reduce the transmission range to 50m. With a simulation time of $4,000s$, this results in nodes having fewer encounters and of shorter duration, which makes the need for good scheduling decisions (and thus CEDO) during these encounters more pressing. We also consider now two mobility scenarios, RD and SLAW. As is evident by this figure, CEDO achieves an almost constant miss rate for all content regardless of popularity, but higher than that of Figure 3, for both mobility scenarios. The effect of limited resources is even more evident for the random direction scenario, where the resources are so limited that requests are only satisfied for the 15-20 most popular contents. Consequently, as predicted by our analysis, when there are enough resources, the optimal policy will favor popular contents but will still allow requests for less popular contents to obtain a good chance of success (Figure 3). However, when resources become too limited, the optimal policy introduces a "cutoff" below which less popular contents are starved (i.e. they will only be served directly by their sources in this case).

[1]see URL http://planete.inria.fr/Software/CEDO/

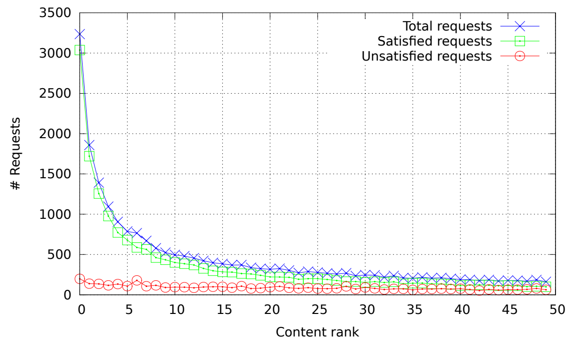
TABLE II: Requests' statistics w/o CEDO.

Policy	Mobility	Range	Total Requests	Satisfied (%)	Unsatisfied (%)
LRU	RD	100m	26,493	14,322 (54%)	12,171 (46%)
CEDO	RD	100m	22,629	18,125 (80%)	4,504 (20%)
LRU	SLAW	100m	25,821	14,951 (58%)	10,870 (42%)
CEDO	SLAW	100m	22,261	18,499 (83%)	3,762 (17%)
LRU	SLAW	50m	28,367	12,315 (43%)	16,052 (57%)
CEDO	SLAW	50m	27,056	13,722 (51%)	13,334 (49%)

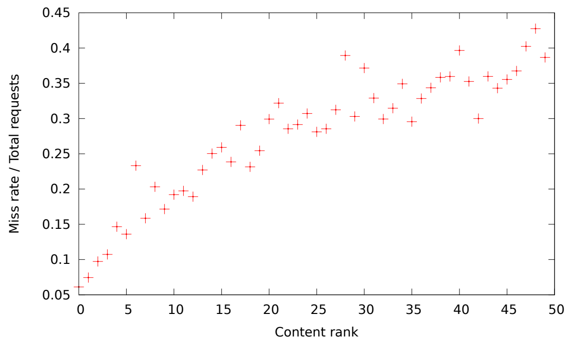
In our final scenario, we compare CEDO to a reasonable baseline policy where nodes implement *Least Recently Used (LRU)* as drop policy in a non-collaborative way (this is often the default policy in CCN), and that schedule content upon encounters according to a simple First-In First-Out policy. Results for both mobility scenarios are shown in Table II and Figure 5. As can be seen there, CEDO achieves a 72% improvement in terms of satisfied requests for the random direction model and a 60% improvement for the SLAW mobility trace with a transmission range of $100m$. Even with a reduced transmission range of $50m$, CEDO still achieves a 19% improvement in terms of satisfied requests compared to LRU as indicated.

V. Conclusion

We designed CEDO, a content delivery algorithm for disseminating named data over a delay-tolerant network. We showed that the delivery rate is maximal when the delivery-rate utility of all contents equals a constant. We also showed that for this particular global objective function, the delivery rate utility for a content is no other than its miss rate. Hence, CEDO must strive to equalize the miss rates of all contents in order to optimize the network throughput. Compared to a *Least Recently Used* baseline drop policy, CEDO achieves up to 72% better performance on delivered content in the network as claimed by our simulation results. Although the initial results are promising, we must perform additional tests with larger networks, comprising hundreds of nodes that publish and subscribe a greater amount of content and measure the convergence time of the utility estimators. We also need to extend CEDO to account for other global performance objectives than network throughput.

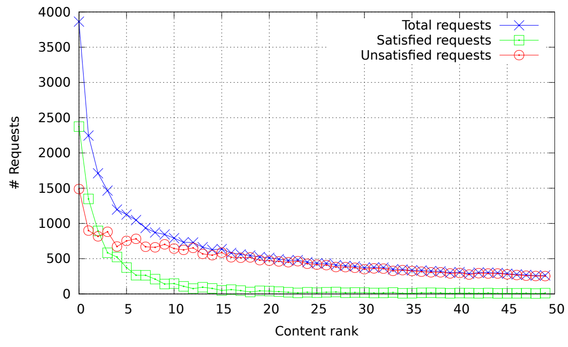


(a) Constant Miss Rate.

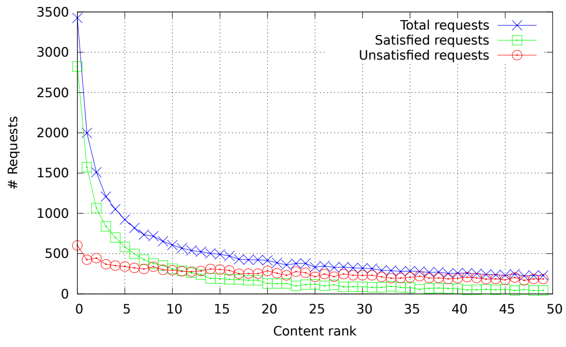


(b) Increasing $\frac{MissRate}{TotalRequestsRate}$.

Fig. 3: CEDO maximizes the total delivery-rate by equalizing the number of unsatisfied requests for all content. As expected, the plot in Figure 3a evidences a constant number of unsatisfied requests for all content, for ranked on the x-axis by decreasing popularity. CEDO does not attempt to equalize the MR/q value, hence the proportion of unsatisfied requests per number of requests increases as the popularity of content decreases; this phenomenon is illustrated in Figure 3b.



(a) Random direction.



(b) Self-Similar Least-Action Human Walk (SLAW).

Fig. 4: The number of total requests, satisfied and unsatisfied, for all content ranked on the x-axis by decreasing popularity. The figures correspond to a challenged scenario with a reduced transmission range of $50m$ and therefore less number of encounters within the transmission range. For both random mobility, Figure 4a, and SLAW mobility, Figure 4b, CEDO can still achieve an almost constant miss rate for all content.

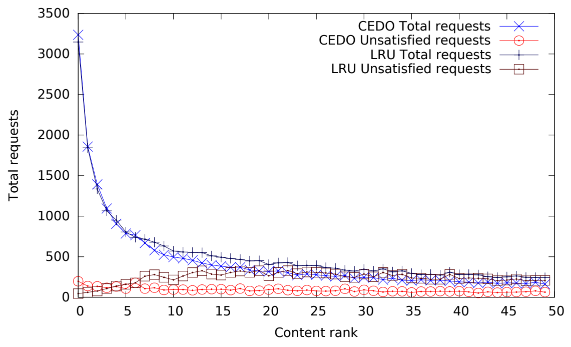


Fig. 5: CEDO shows a constant number of unsatisfied requests, regardless of content popularity. LRU from its side shows a high satisfaction rate for very popular content, which means that unlike CEDO, it can not satisfy requests for unpopular content in favour of very popular content.

VI. Acknowledgments

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

References

[1] BonnMotion, A mobility scenario generation and analysis tool. URL http://net.cs.uni-bonn.de/wg/cs/applications/bonnmotion/
[2] Lee Breslau, Pei Cue, Pei Cao, Li Fan, Graham Phillips, Scott Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications.In *Infocom*. March 1999.
[3] *Project CCNx*. URL http://ccnx.org/.
[4] Kevin Fall. A Delay-Tolerant Network Architecture for Challenged Internets. In *ACM Sigcomm*. August 2003.
[5] Christine Fricker, Philippe Robert, James Roberts, Nada Sbihi. Impact of Traffic Mix on Caching Performance in a Content-Centric Network. *Technical report, 1202.0108v1 INRIA*. February 2012.
[6] Thomas R. Henderson, Mathieu Lacage, George F. Riley. Network Simulations with the ns-3 Simulator. In *ACM Sigcomm (demo session)*. August 2008.
[7] Liang Hu, Jean-Yves Le Boudec, and Milan Vojnovic. Optimal channel choice for collaborative ad-hoc dissemination. In *IEEE Infocom*. 2010.
[8] Stratis Ioannidis, Laurent Massoulie, Augustin Chaintreau. Distributed caching over heterogeneous mobile networks. In *ACM Sigmetrics*. 2010.
[9] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, Rebecca L. Braynard. Networking Named Content. In *ACM CoNEXT*. December 2009.
[10] Sushant Jain, Kevin Fall, Rabin Patra. Routing in Delay Tolerant Network. In *ACM Sigcomm*. August 2004.
[11] Thomas Karagiannis, Jean-Yves Le Boudec EPFL, Milan Vojnovic. Power law and exponential decay of inter contact times between mobile devices. In *ACM Mobicom*. New York 2007.
[12] Timothy Keller, Ingram Olkin. Combining Correlated Unbiased Estimators of the Mean of a Normal Distribution. *Technical report, Stanford*. 2002.
[13] Amir Krifa, Chadi Barakat, Thrasyvoulos Spyropoulos. Message Drop and Scheduling in DTNs: Theory and Practice. in *IEEE Transactions on Mobile Computing,* vol. 11, no. 9, 2012.
[14] Amir Krifa, Chadi Barakat, Thrasyvoulos Spyropoulos. Optimal Buffer Management Policies for Delay Tolerant Networks. In *IEEE SECON*. June 2008.
[15] Amir Krifa, Chadi Barakat, Thrasyvoulos Spyropoulos. MobiTrade: Trading Content in Disruption Tolerant Networks. In *ACM Chants*. September 2011.
[16] M. Lacage. Outils d'expérimentation pour la recherche en réseaux. *Ph.D. thesis, Université de Nice Sophia-Antipolis*. 2010.
[17] Kyunghan Lee, Seongik Hong, Seong Joon Kim, Injong Rhee, Song Chong. SLAW: Self-Similar Least-Action Human Walk. In *IEEE/ACM Transactions on Networking*. July 2011.
[18] Anders Lindgren, Avri Doria, Olov Schelén. Probabilistic routing in intermittently connected networks. In *ACM Sigmobile*. July 2003.
[19] Anirban Mahanti, Carey Williamson, Derek Eager. Traffic analysis of a web proxy caching hierarchy. In *IEEE Network*, Vol. 14, June 2000.
[20] Joshua Reich, Augustin Chaintreau. The age of impatience: optimal replication schemes for opportunistic networks. In *ACM Conext*. 2009.
[21] Giuseppe Sollazzo, Mirco Musolesi, Cecilia Mascolo. TACO-DTN: a time-aware content-based dissemination system for delay tolerant networks. In *MobiOpp Workshop*. 2007.
[22] Amin Vahdat, David Becker. Epidemic Routing for Partially-Connected Ad Hoc Networks. *Technical report, CS-200006*. April 2000.
[23] Athanasios V. Vasilakos, Yan Zhang, Thrasyvoulos Spyropoulos. Delay Tolerant Networks Protocols and Applications. *CRC Press*. 2012.
[24] Xiaolan Zhang, Giovanni Neglia, Jim Kurose, Don Towsley. Performance modelling of epidemic routing. In *IFIP Networking*. May 2006.
[25] Xuejun Zhuo, Qinghua Li, Guohong Cao, Yiqi Dai, Boleslaw Szymanski, Tom La Porta. Social-Based Cooperative Caching in DTNs: A Contact Duration Aware Approach. In *IEEE MASS*. 2011.