Distributed Workload Dissemination for Makespan Minimization in Disruption Tolerant Networks

Sheng Zhang, *Member, IEEE*, Jie Wu, *Fellow, IEEE*, and Sanglu Lu, *Member, IEEE*

Abstract—Mobile devices are undergoing explosive proliferation today. Although they are gaining more and more capabilities, they still fall short to execute complex applications. One possible solution to alleviate this limitation is offloading tasks to remote clouds. However, it may require persistent connectivity to the Internet and thus is not always available or affordable. An alternative solution is taking advantage of pervasive mobile devices and their pairwise encounters. In this paradigm, complex tasks from mobile devices are processed in a distributed and collaborative fashion on all mobile devices that are loosely-connected. Working towards this vision, this paper studies the following problem: given a task that originates at some node in a Disruption Tolerant Network (DTN), how are we to disseminate the task's workload during the pairwise contacts among mobile devices to achieve makespan minimization? We first imagine access to an oracle that has global and future knowledge of node mobility, and we design a provably-optimal centralized polynomial-time solution as the benchmark for comparison. With the insights obtained from the centralized solution, we then develop a distributed dissemination algorithm, D2, which maintains certain neighborhood information at individual nodes. D2 makes dissemination decisions based on the estimations of the potential computational capacities and the future workloads of mobile nodes. Extensive trace-driven simulations confirm the effectiveness of D2.

Index Terms—Disruption tolerant networks (DTNs), minimum makespan scheduling, workload dissemination

1 Introduction

Mobile computing has experienced serious growth in recent years, attracting increasing attention from academic and industrial communities. Central to mobile computing, mobile devices are undergoing explosive proliferation today. Although these devices are gaining more and more capabilities, they still fall short to execute complex applications and services.

When we have to handle a task that requires complex processing, we usually have two choices. At one extreme is offloading the task to remote clouds, which may require persistent connectivity to the Internet and thus is not always available or affordable [1]. The other extreme is taking advantage of pervasive mobile devices and their pairwise encounters. In this paradigm, due to the unpredictable mobilities and the limited communication ranges of mobile devices, they can only contact each other opportunistically within their roaming regions, and are loosely-connected. In other words, these devices form a type of Disruption Tolerant Network (DTN) [2]. This paradigm requires computational tasks to permit partitioning of the workload into small pieces, which do not have dependency relations, and the computing output can be constructed by consolidating partial outputs from pieces of workload. Tasks that satisfy this requirement include large matrix-vector products [3], large file compression [4], etc. For these tasks, disseminating the workload in the network around us is not only suitable but also essential to complete them in time.

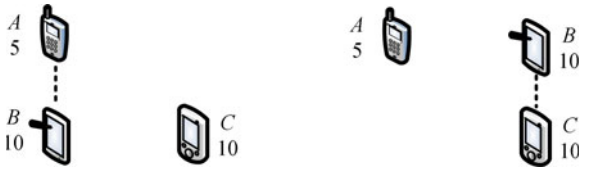
In this paper, we investigate the problem of disseminating the workload of a task among a set of loosely-connected mobile devices to minimize the *maximum completion time* (i.e., makespan), and focus on designing an efficient distributed dissemination algorithm. Unlike the classical minimum makespan scheduling problem [5], we do not have information about which devices would contribute to the completion of the task, or from which time a device begins to participate in the collaboration.

There are two general challenges, i.e., *no global knowledge* and *no future knowledge*, in designing efficient distributed disseminating algorithms. We use the example in Fig. 1 to illustrate these challenges. In Fig. 1, there are three mobile devices (A, B, and C) and two contacts, i.e., A meets B when the time t is 0.00 and B meets C when $t = 5.00$). The workload processing rate of each device is written next to the respective device, e.g., A can finish 5 units of workload in one time slot. Suppose that, A, B, and C have 600.00, 0.00, and 50.00 units of workload, respectively, when $t = 0.00$.

Fig. 1c shows the results of two different algorithms. With the Naïve algorithm, during each contact, the total workload is split between two devices based on the ratio of their processing rates. For example, when $t = 0.00$, after workload split, A keeps 200 units of workload for itself and transfers the other 400 units of workload to B. The makespan of this algorithm is 40 slots. With the D2 algorithm developed in this paper, the impact of the future contact

- *S. Zhang and S. Lu are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, P.R. China. E-mail: {sheng, sanglu}@nju.edu.cn.*
- *J. Wu is with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122. E-mail: jiewu@temple.edu.*

Manuscript received 4 Nov. 2014; revised 24 Apr. 2015; accepted 15 Sept. 2015. Date of publication 18 Sept. 2015; date of current version 1 June 2016. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TMC.2015.2480075



(a) $t = 0.00$ (b) $t = 5.00$

	Node	Split at $t = 0.00$		Split at $t = 5.00$		Completion Time
		Before	After	Before	After	
Naïve	A	600.00	200.00	175.00	175.00	40.00
	B	0.00	400.00	350.00	175.00	22.50
	C	50.00	50.00	0.00	175.00	22.50
D2	A	600.00	133.68	108.68	108.68	26.74
	B	0.00	466.32	416.32	208.16	25.82
	C	50.00	50.00	0.00	208.16	25.82

(c) Dissemination results.

Fig. 1. A motivational example. The processing rate of each device is written next to the respective device. (a) A meets B when the time t is 0.00. (b) B meets C when the time t is 5.00. (c) Results of applying two dissemination algorithms, Naïve and D2.

between B and C is taken into account even when A meets B. Based on Equ. (1), A transfers 466.32 units of workload to B and keeps the rest for itself. In this way, the makespan of D2 is 26.74 slots, which is much shorter than that of Naïve.

In this paper, to gain a better understanding of the problem, we first study the scenario where we have access to an *oracle* that knows global and future knowledge of node mobility, and we propose a centralized polynomial-time disseminating algorithm based on the shortest delay tree, which is derived from the Dijkstra's shortest path algorithm [6]. The proposed centralized algorithm is proven to be optimal, and later serves as the comparison benchmark in trace-driven simulations.

With the insights obtained from the oracle case, we then develop a distributed dissemination algorithm, D2, which enables each device to determine its disseminating strategy, such that all devices can collaboratively complete the task and achieve the minimal makespan. More specifically, in each individual node or device, D2 is comprised of four components: *the workload queue* manages operations (e.g., integration and splitting) on the actual workload; *the r-hop neighborhood information manager* stores and updates the contact rate, opportunistic path, and workload information for every r-hop neighbor of a node; *the finish time estimator* calculates the expected finish time of a given workload; and *the future workload estimator* predicts the expected workload in a node at a future time point.

We show through trace-driven simulations that the performance, in terms of makespan, of D2 with only 1-hop neighborhood information is already near-optimal in three realistic traces, and the performance gap between D2 and the optimal solution becomes smaller as the amount of information maintained at individual nodes increases. The contributions of this paper are summarized as follows:

1) To the best of our knowledge, we are the first to study the minimum makespan workload dissemination problem in disruption tolerant networks. We formulate the problem and identify the challenges in designing efficient distributed algorithms.
2) With the access to global and future mobility knowledge, we design an optimal centralized polynomial-time algorithm as performance benchmark.
3) We develop D2 that maintains limited information at individual devices and makes workload split decisions based on heuristic workload and finish time estimations.

The remainder of this paper is organized as follows. We motivate our work in Section 2. Section 3 provides the models and problem formulation. The optimal benchmark solution is presented in Section 4. Sections 5 and 6 present the overview and design details of D2, respectively. We provide several extensions in Section 7. Performance evaluations are introduced in Section 8. Before concluding this paper in Section 10, we go over related work in Section 9.

2 Motivation and Scenario

2.1 Motivation

In an era of mobile computing, we are surrounded by massive mobile devices. Although these devices are gaining more and more capabilities, they still fall short to execute complex applications and services. One solution to alleviate this limitation is mobile cloud computing [7], i.e., computationally expensive tasks are offloaded to the remote cloud infrastructure. However, this solution requires persistent connectivity to the cloud, which is not always *available* or *affordable* [8], [9].

An alternative solution is to pool nearby mobile devices together for resource sharing and forming an "ad hoc mobile cloud" [1], [10] or a "cloudlet" [9], [11]. In this paradigm, complex tasks from mobile devices are then processed in a distributed and collaborative fashion on all mobile devices that are loosely-connected. The feasibility of this idea has been demonstrated in Virtual Cloud Provider [8], which organizes neighboring mobile devices pursuing same interests or goals into a virtual cloud for distributed computational tasks.

Despite the obstacles mobile devices inevitably have to face compared to conventional information processing devices, e.g., resource limitations, and connectivity variability, there are several benefits to consider this paradigm [9], [10], [12]:

1) *In-place processing.* Mobile data (e.g., surveillance video clips, and environmental sensing data) originates at network edges and can be processed in-place or nearby devices. In doing so, we can avoid the expensive data transfer to remote clouds [10], and meanwhile, the Internet is relieved, since uploading tasks and related data to remote clouds may take up a large amount of bandwidth
2) *Infrastructure-free collaboration.* Utilizing neighboring devices requires no infrastructure, thus, can be a backup solution when a collection of devices are unable to access Internet-based clouds [12].
3) *Context-aware service.* Most mobile devices have sensing abilities, an ad hoc mobile cloud made up of these devices will be able to provide context-aware services [9].

2.2 Scenario

The ad hoc mobile cloud paradigm is attractive in many scenarios [9], [10], [11]. For example, mobile clouds can be used

to perform digital signal processing of radio telescope data from the Arecibo radio observatory [13], or search for evidence of continuous gravitational-wave sources [14]. In these applications, mobile clouds serve as a complementary solution to traditional Internet-based computing paradigms.

Besides, in some other scenarios, the ad hoc mobile cloud seems more appropriate. Here is an example adapted from [8]. Suppose an American Mike is visiting museums in China. In a museum, he becomes interested in an artwork from ancient times; though there is a description of the artwork, he cannot read it as the description is indistinct and in Chinese! He takes several pictures of the description and intends to use a pattern recognition-based software to recognize the texts, which can be translated into English later. However, his mobile phone cannot efficiently handle these pictures due to limited processing capability. As the cost of data roaming is extremely expensive, he cannot upload the pictures to remote clouds, either. What can he do? One possible way is to check for other nearby users that are also interested in understanding the texts, and collaboratively finish the recognition task.

In fact, this example is not unusual in place-based activities (e.g., museum visits, social meeting, and archeological expeditions in deserts). Collocation increases the probability of sharing common interests among users, which encourages them to collaboratively finish tasks through resource sharing.

2.3 Assumptions

In this paper, we consider parallel tasks which satisfy the following two properties. First, *workload of a task is fine-grained and permits arbitrary partitioning*. Second, *the computing output can be constructed by consolidating partial outputs from pieces of workload*. Many tasks in reality have these two properties, e.g., large matrix-vector products, and large file compression. With these two properties, when two devices encounter each other, the total workload on them can be freely re-distributed in any ratio, and the final output can be derived by gathering partial outputs from each piece of workload.

There are two phases in finishing a task: *workload dissemination* and *output collection*. For the second phase, it is merely a routing problem in DTNs, and there are plenty of routing protocols. Hence, when a device finishes the workload allocated to it, the device can send the output to the task source with some DTN routing protocol (e.g., Spray and Wait [15], Delegation forwarding [16], and RAPID [17]). Therefore, in this paper, *we focus on the workload dissemination phase*.

As there are hundreds of (or even thousands of) gigabytes of storage in modern devices; files of several megabytes are no problem. Hence, we do not consider storage constraints in this work. Given the high transmission speed of proximity-based communication technologies, e.g., the typical rate of Bluetooth v1.2 is 1 MB per second [18], we also assume that all necessary data transfers can be completed in any contact, and the transmission time is negligible compared to the relatively long task processing time at individual devices.

3 Problem Formulation

In this section, we introduce the network and task models, followed by the problem formulation.

3.1 Models

We model the underlying loosely-connected network as a graph $G = (V, E)$. The vertex set V represents mobile devices. Each device $i \in V$ can process r_i units of workload in one time slot. We assume that the processing rate is constant for each node. This is reasonable, since a mobile device can decide how much computing capability it contributes to our system, based on its battery and load status. Note that, there are already some mechanisms (e.g., [19]), which motivate mobile users to participate in crowdsourcing or volunteer computing applications. We will not discuss the design of such mechanisms, as it is orthogonal to the focus of this paper.

The edge set E represents the stochastic contacts between devices. Two devices i and j can communicate with each other via wireless interfaces only when they are within the communication range of each other. The inter-contact time between i and j is assumed to be exponentially distributed with contact rate λ_{ij}. In other words, the contacts between two devices i and j follow Possion distribution with contact rate λ_{ij}. This assumption fits well with realistic traces, as shown in many prior theoretical and experimental works [17], [20]. Each node is also assumed to contact its neighbors one by one, since a node does not frequently contact multiple neighbors at the same time. We note that, though the inter-contact time is assumed to follow exponential distribution, our dissemination algorithm can also be extended to scenarios where inter-contact times follow other distributions, as shown later in Section 7.

3.2 Problem

We are interested in examining the research decisions and design tradeoffs that may arise when making full use of the computing power of pervasive mobile devices. To initiate a tractable study, this paper narrows the scope of this problem to a manageable extent—we investigate how to disseminate the workload from a *single* task in an *empty* DTN, where by "empty" we mean that: 1) when the single task originates at its source, all of the other devices do not have any unfinished workload, and 2) before the single task is finished, there are no more tasks that would originate in the DTN. This simplification lets us focus on the main problem; we also provide remarks on how to support multiple simultaneous tasks in our algorithm in Section 7. We hope that our work can provide some insights for future studies.

In our scenario, we consider a task C that contains W units of workload. Denote the amount of workload in device i at time t as W_i^t. Without loss of generality, we assume that, the task C originates at its source $s \in V$ at $t = 0$. Since we consider the case where there is only one task, we have

$$W_i^0 = \begin{cases} W & \text{if } i = s, \\ 0 & \text{otherwise.} \end{cases}$$

The completion time T, also called the makespan [5], is defined as the time difference between the origin time, i.e., $t = 0$, and the finish time, i.e., the time point when all participating devices finish their respective workloads that belong to C. Different from the classical minimum makespan scheduling problem [5], which finds an assignment of multiple jobs to multiple identical machines so that the

Notations	Definitions
V	the set of devices in a DTN
E	the set of stochastic edges in a DTN
r_i	the processing rate of device i
λ_{ij}	contact rate between devices i and j
W	the amount of workload
T	the task completion time
P_{ij}	opportunistic path between devices i and j
W_i^t	the amount of workload in device i at time t
N_i^r	the set of devices within r hops from device i
$\Phi_i(t)$	the future workload estimator of device i
$\mathcal{T}_i(w)$	the finish time estimator of device i

Fig. 2. Main notations for quick reference.

makespan is minimized, we concentrate on designing a distributed solution with respect to unpredictable communication delays between mobile devices. Main notations are summarized in Fig. 2 for quick reference. Our problem can be stated as follows:

Problem: (Minimum Makespan Workload Dissemination, MMWD) Given a DTN $G = (V, E)$, where the processing rate of node $i \in V$ is r_i, and the inter-contact time between i and j follows exponential distribution with contact rate λ_{ij}. For a task C that consists of W units of workload and originates at node $s \in V$ when $t = 0$, how are we to disseminate the workload during pairwise contacts to minimize its makespan T?

4 The Optimal Solution with Global and Future Information

In this section, based on the access to the oracle that has global and future knowledge of device contacts, we present a provably-optimal centralized polynomial-time solution, which will serve as the benchmark in performance evaluation. We also provide an example to better explain our solution.

4.1 Discrete Contact Graph

For a DTN $G = (V, E)$, if we have the global and future knowledge of node contacts, the edge between device $i \in V$ and $j \in V$ can be represented by $(t_{ij}^1, t_{ij}^2, \ldots, t_{ij}^k, \ldots)$, where t_{ij}^k indicates the time point of the kth contact between devices i and j over a period of time. Fig. 4a shows an example that we will use throughout this paper. There are five mobile devices in the DTN, and the processing rate of each device is written next to the respective circle that represents it, and the contact opportunities of each pair of devices are written next to the respective edge that represents it.. Take the edge between devices 2 and 3 for example, "(3,6)" means that, devices 2 and 3 could communicate with each other when the time $t = 3$ and $t = 6$.

```
1: for each 1-hop neighbor v of u do
2:     alt ← min t_uv^k, s.t., t_uv^k ≥ d[u]
3:     if alt < d[v] then
4:         d[v] ← alt, previous[v] ← u
5:     end if
6: end for
```

Fig. 3. The relaxation part of SDTA.

4.2 Shortest Delay Tree-Based Algorithm (SDTA)

In this subsection, we present the Shortest Delay Tree-based Algorithm, a provably optimal centralized polynomial-time solution to the MMWD problem when we have the global and future knowledge of node contacts. The main idea of SDTA is to make sure that each device participates in the dissemination as early as possible, and all participating devices finish their respective workloads at the same time. In doing so, the makespan is minimized, since the computing power of each device is utilized as early as possible.

SDTA consists of four main steps. First, we derive *the shortest delay* from the source device to all of the other devices based on the Dijkstra's shortest path algorithm [6]; then, we find out *the maximum amount of workload* $W(t)$ that originates at the source and can be finished with exactly t time slots. Third, we determine *the minimum makespan* based on the shortest delay tree and $W(t)$. Finally, we compute *the dissemination plan* according to the shortest delay tree and the minimum makespan.

Without loss of generality, we assume that the task originates at its source device s. Denote the shortest communication delay that some workload could be transferred from s to another device i as $d[i]$. With the discrete contact graph, we can obtain the the shortest delays using Dijkstra's shortest path algorithm with a few modifications—the only difference is the relaxation part, which is shown in Fig. 3. Similar to Dijkstra's algorithm, we use $previous[i]$ to represent the direct previous device of i along the shortest path from s to i. Connecting each device i to $previous[i]$, we obtain the shortest delay tree. Fig. 4b shows the shortest delay tree of the discrete contact graph in Fig. 4a with $s = 0$. For example, the shortest delay $d[3]$ from device 0 to device 3 is 2, and $previous[3] = 1$.

We denote by $W(t)$ the maximum amount of workload that originates at source s and can be finished within exactly

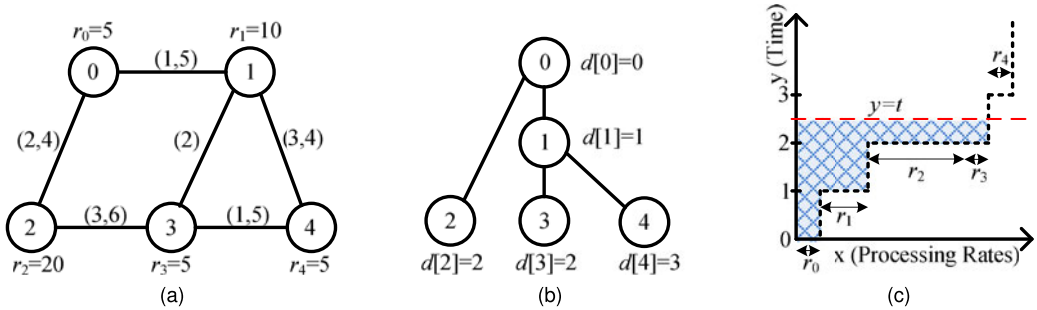


Fig. 4. An example of SDTA. (a) Discrete contact graph. (b) Shortest delay tree. (c) $W(t)$ is the size of the plane bounded by $y = 0$, y-axis, $y = t$, and the dashed polyline.

t time slots. Without loss of generality, we assume that $s = 0$ and $0 = d[0] < ... < d[i] < ... < d[|V|-1]$ (if $d[i] = d[j]$, we consider them as a whole). We have the following piecewise function:

$$W(t) = \begin{cases} \sum_{k=0}^{i} r_k \cdot (t - d[k]) & \text{if } d[i] \le t < d[i+1] \text{ for some } 0 \le i < |V|-1 \\ \sum_{k=0}^{|V|-1} r_k \cdot (t - d[k]) & \text{if } d[|V|-1] \le t. \end{cases}$$

In Fig. 4, we have $0 \le t \le 1$, $W(t) = 5t$; when $1 < t \le 2$, $W(t) = 15t - 10$; when $2 < t \le 3$, $W(t) = 40t - 60$; when $3 < t$, $W(t) = 45t - 75$. In fact, as shown in Fig. 4c, $W(t)$ is the size of the plane bounded by $y = 0$, y-axis, $y = t$, and the dashed polyline.

The minimum makespan T can be determined by solving $W(T) = W$. In Fig. 4a, suppose that a task with 28 units of workload originates at its source 0. Based on the above results on $W(t)$, we have $T = 2.2$.

Denote the amount of workload that device i should finish as $assign[i]$. To achieve the minimum makespan, every device contributes its computing power as early as possible, thus, it is not hard to see that,

$$assign[i] = \begin{cases} r_i \cdot (T - d[i]) & \text{if } d[i] < T, \\ 0 & \text{otherwise.} \end{cases}$$

In our example, we have $assing[0] = 11$, $assing[1] = 12$, $assing[2] = 4$, $assing[3] = 1$, and $assing[4] = 0$.

With the shortest delay tree and the workload assignments, we can determine the dissemination plan as follows: When two devices i and j have a contact, if $previous[j] = i$, node i has some unfinished workload, and device i has not transferred any workload to j, then device i transfers to j all of the workload assignments that belong to the subtree rooted at j. In Fig. 4, when two devices 0 and 1 meet at the beginning of the first time slot, device 0 transfers $assign[1] + assign[3] = 13$ units of workload to device 1; when two devices 0 and 2 meet at the beginning of the second time slot, device 0 transfers $assign[2] = 4$ units of workload to device 2; when two devices 1 and 3 meet at the beginning of the second time slot, device 1 transfers $assign[3] = 1$ unit of workload to device 3.

4.3 Summary

Since SDTA utilizes the computing power of every device as early as possible, and makes sure that all of the participating devices finish their respective workloads at the same time, it is straightforward to see its optimality.

Theorem 1. *For the MMWD problem with an oracle that knows global and future information of pairwise contacts,* SDTA *is optimal in terms of makespan.*

Finding the shortest delays in SDTA takes $O(|V|^2)$ time; sorting these delays takes $O(|V|log|V|)$ time; searching in step 3 takes $O(log|V|)$ time; assigning in step 4 takes $O(|V|)$ time; therefore, the overall running time is dominated by $O(|V|^2)$.

5 D2 IN A NUT SHELL

D2 is a distributed dissemination algorithm for the MMWD problem. In this section, we provide a brief overview of D2 to help the reader grasp the main principle behind our design. We first introduce its main components, then we present the workload split rule and information exchange procedure during a contact.

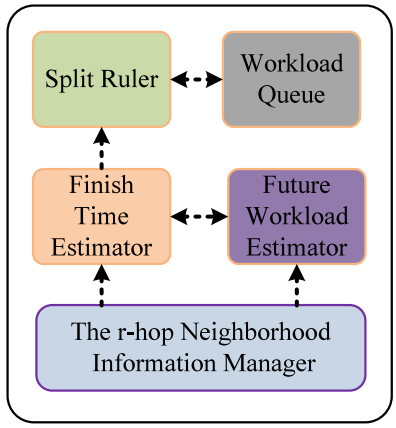


Fig. 5. Main components of D2.

5.1 Main Components

The global network status and future node contacts are key to minimizing makespan. However, in DTN environments, they are hard to know from the perspective of each individual device. D2 makes dissemination decisions based on estimations of the potential computing power and the future workload of each device. Fig. 5 shows the main components in the proposed algorithm.

The r-hop neighborhood information manager. Device i is a 1-hop neighbor of device j if and only if their contact rate λ_{ij} is positive. Recursively, we can define r-hop neighbor: i is a r-hop neighbor of j if and only if, i is 1-hop neighbor of another node, which is a $(r-1)$-hop neighbor of j. For example, in Fig. 4a, nodes 1 and 2 are 1-hop neighbors of 0; nodes 3 and 4 are 2-hop neighbors of 0. Denote the set of devices that are within r hop(s) from device i by N_i^r. For each device k that is within r hop(s) from device i, this manager is responsible for storing and updating the contact rate λ_{ik}, the opportunistic path P_{ik}, and the workload $\Phi_k(t)$, which are used by the other components for estimations. As we shall see in our simulations, when more neighborhood information is maintained at individual nodes, D2 performs better. Section 6.1 provides more details regarding this manager.

Finish time estimator. Based on the information maintained by the r-hop manager, this component generates and updates a function $\mathcal{T}_i(w)$, which returns the expected time for device i and its neighbors to finish w units of workload. We note that it is non-trivial to obtain $\mathcal{T}_i(w)$, since the computing power and the amount of workload in all r-hop neighbors of device i should be taken into consideration. Section 6.2 provides more details.

Future workload estimator. This estimator generates and updates a function $\Phi_i(t)$, which returns the expected workload in device i in a future time point t, i.e., the domain of this function is $\{t|t \ge t_0\}$, where t_0 is the generating time of this function. Since device i may transfer/receive some workload to/from other devices during future contacts, it is also non-trivial to obtain this estimator. We present more details in Section 6.3.

Workload queue. This component manages operations on the actual workload in four aspects: 1) it consolidates partial outputs from the finished workload to construct an intermediate result and sends it to a task source at proper time points; 2) it stores the unfinished workload and is responsible for updating W_i^t; 3) when another node j transfers some workload to node i, it integrates the new workload into its own; 4) when node i goes to transfer a certain part of its workload to another node j, it splits the workload into two corresponding parts.

Split ruler. When one node encounters another one, from their viewpoints, in order to minimize the makespan, their total combined workloads should be re-distributed in such a way that *they finish their separate parts by the same time*. We note that, the functions $\mathcal{T}_i(w)$ and $\mathcal{T}_j(w)$ have already incorporated the impact of the computing power and the amount of workload in their respective r-hop neighborhoods, as we shall see in Section 6.2. Suppose that, device i with W_i^t units of workload meets another device j with W_j^t units of workload at time t. Denote by x_i and x_j the amounts of workload of i and j after workload split, respectively. Then, x_i and x_j should satisfy:

$$\begin{cases} x_i + x_j = W_i^t + W_j^t, \\ \mathcal{T}_i(x_i) = \mathcal{T}_j(x_j). \end{cases} \tag{1}$$

We note that, in D2, some pair of devices may need to re-distribute the workload between them whenever there is a contact, and a device may receive workloads from multiple devices, while in SDTA, there is no need for a pair of devices to re-distribute the workload multiple times. This is because, in DTN environments, a device cannot have accurate global and future information of node contacts, and thus, every communication opportunity should be exploited to locally improve the workload dissemination performance.

5.2 Information Exchange Procedure

We now present how two devices i and j exchange information in a contact at time t. Generally speaking, the goal of information exchange is to facilitate the construction of the finish time estimator and future workload estimator. Specifically, after neighbor discovery:

1) Two devices i and j exchange the amounts of their respective workloads, i.e., W_i^t and W_j^t.
2) Device i generates $\mathcal{T}_i(w)$ based on λ_{ik}, r_k, and $\Phi_k(t)$ of each $k \in N_i^r \setminus \{j\}$; device j generates $\mathcal{T}_j(w)$ based on λ_{jh}, r_h, and $\Phi_h(t)$ of each $h \in N_j^r \setminus \{i\}$. Here, "$\setminus$" denotes the set minus operation. Without loss of generality, j sends the function $\mathcal{T}_j(w)$ to i.
3) Device i decides how to split the total workload among them according to the aforementioned split rule, and returns the result to device j. The workload queues then complete the necessary workload transfers between two devices.
4) Devices i generates a new version of $\Phi_i(t)$ and sends it to j, then, j updates its maintained version of $\Phi_i(t)$ to be the new one; similarly, j generates a new version of $\Phi_j(t)$ and sends it to i, then, i updates its maintained version of $\Phi_j(t)$ to be the new one.

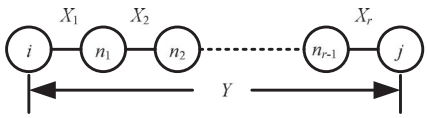


Fig. 6. Opportunistic path and its weight.

5) Device i updates $\Phi_k(t)$ and P_{ik} for each device $k \in N_j^{r-1}$, as stated in Section 6.1; device j updates its corresponding information in a similar way.

6 Design Details of D2

In this section, we present the design details of main components in D2. We also provide a concrete example that helps the reader to better understand our design.

6.1 The r-Hop Information Manager

This component, i.e., the r-hop manager of a device i maintains the following information for each $j \in N_i^r$:

Contact rate λ_{ij}. The r-hop manager of a device i maintains two variables, i.e., F_{ij} and f_{ij}, for calculating λ_{ij} in a time-average manner in real time. F_{ij} represents the number of time slots elapsed since D2 took effect; f_{ij} represents the number of contacts between i and j. Thus, $\lambda_{ij} = f_{ij}/F_{ij}$. F_{ij} and f_{ij} are updated whenever i meets j. For example, we start D2 at slot t_0, and i meets j at slots $t_1, t_2, \ldots, t_{m-1}$; when they meet at slot t_m, F_{ij} and f_{ij} are updated to $(t_m - t_0)$ and m, respectively.

Workload $\Phi_j(t)$. The r-hop manager of a device i maintains a version of the function $\Phi_j(t)$, which represents the amount of workload in device j at slot t. We note that, device i may get this function from device j or another device $k \in N_i^r$. As we mentioned before, when i meets j, device i replaces its version of $\Phi_j(t)$ with the new one generated by j; in addition, for each device $h \in N_j^{r-1}$, if the version of $\Phi_h(t)$ in device j is more recent than that in device i, then device i updates it. Device j will also update information similar to that of device i.

Opportunistic path P_{ij}. Opportunistic path and its weight are defined as follows [21]: An r-hop opportunistic path P_{ij} between i and j is a sequence of devices $i = n_0, n_1, \ldots, n_{r-1}, n_r = j$, where the contact rate λ_k $(1 \le k \le r)$ between n_{k-1} and n_k is positive. Path weight $\omega(P_{ij})$ is defined as the expected delay from i to j.

We use Fig. 6 to illustrate how we compute $\omega(P_{ij})$. Denote the inter-contact time between n_{k-1} and n_k as X_k, the delay from i to j as Y, and then we have $Y = \sum_{k=1}^{r} X_k$. As we mentioned before, X_k follows exponential distribution with parameter λ_k, i.e., its probability density function (PDF) is $p_{X_k}(x) = \lambda_k e^{-\lambda_k x}$. Therefore, Y follows hypoexponential distribution [22], and the PDF of Y is:

$$p_Y(y) = \sum_{k=1}^{r} \left(\prod_{h=1, h\neq k}^{r} \frac{\lambda_h}{\lambda_h - \lambda_k} \right) p_{X_k}(y). \tag{2}$$

Then, the weight of P_{ij} is:

$$\omega(P_{ij}) = \int_o^\infty p_Y(y) y dy = \sum_{k=1}^{r} \frac{1}{\lambda_k}. \tag{3}$$

For each device $j \in N_i^r$, device i maintains a path P_{ij} with the smallest weight. Such kinds of information are updated

as follows: When i meets j, for all $k \in N_i^r \setminus N_j^{r-1}$, i keeps P_{ik} unchanged; for all $k \in N_j^{r-1} \setminus N_i^r$, i initializes P_{ik} by catenating i and P_{jk} (denoted as $i + P_{jk}$); for all $k \in N_j^{r-1} \cap N_i^r$, if the weight of P_{ik} is larger than $i + P_{jk}$, i replaces P_{ik} with $i + P_{jk}$.

We make two remarks. First, denote by R the network diameter that represents the hop count of the longest shortest path among any two devices in the network. We note that, even if D2 maintains R-hop information at individual devices, it does not imply that each device knows global network status, since D2 does not maintain the information between any two neighbors of a device.

Second, D2 updates network status information via pairwise communication. There are several reasons for this choice. First, even if we leverage long-distance communication to collect global network status and push it to all participating devices, information about future contacts between mobile devices remains unknown. Besides, the underlying DTN environment could be highly dynamic and very large in terms of number of devices. Whenever any device arrives or leaves, the network information should be updated, which may cause too frequent long-distance communication. Third, D2 aims to provide a low-infrastructure service. With its current design, D2 assumes no infrastructure; otherwise, D2 would require all participating devices to keep their long-distance communication interfaces on all the time.

6.2 Finish Time Estimator

In this section, we present how device i generates its finish time estimator. We first consider two special cases, then extend the results to the general case.

Special case 1: $r = 1$, *and* N_i^1 *contains only one device.* Denote the only 1-hop neighbor of i as j, and the current time as t. Suppose that device i has w units of workload; based on its r-hop manager, device i knows λ_{ij}, r_j, and $\Phi_j(t)$. If $\Phi_j(t)/r_j > w/r_i$, device j would be helpless, thus, $\mathcal{T}_i(w) = w/r_i$. Otherwise, we have the following theorem (the proof can be found in the supplemental material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TMC.2015.2480075):

Theorem 2. *When $r = 1$, device i has w units of workload and has only one 1-hop neighbor j; if $\Phi_j(t)/r_j < w/r_i$, then we have:*

$$\mathcal{T}_i(w) = \frac{1}{r_i + r_j}\left(w + \Phi_j(t) + \frac{r_j}{\lambda_{ij}}\left(e^{-\frac{\lambda_{ij}}{r_j}\Phi_j(t)} - e^{-\frac{\lambda_{ij}}{r_i}w}\right)\right).$$

Special case 2: $r = 1$, *and* N_i^1 *contains only two devices, say j and k.* Similarly, we have:

Theorem 3. *When $r = 1$, device i has w units of workload and only has two 1-hop neighbors, j and k; if $\Phi_j(t)/r_j < w/r_i$ and $\Phi_k(t)/r_k < w/r_i$, then we have:*

$$\begin{aligned}\mathcal{T}_i(w) = \frac{1}{r_i + r_j + r_k}\Bigg(w + \Phi_j(t) + \frac{r_j}{\lambda_{ij}}\Bigg(e^{-\frac{\lambda_{ij}(r_j+r_k)}{r_j \cdot r_j}\Phi_j(t)} \\ - e^{-\frac{\lambda_{ij}(r_j+r_k)}{r_i \cdot r_j}w}\Bigg) + \Phi_k(t) + \frac{r_k}{\lambda_{ik}}\Bigg(e^{-\frac{\lambda_{ik}(r_j+r_k)}{r_k \cdot r_k}\Phi_k(t)} \\ - e^{-\frac{\lambda_{ik}(r_j+r_k)}{r_i \cdot r_k}w}\Bigg)\Bigg) + \Delta(i,j,k).\end{aligned}$$

Here, $\Delta(i,j,k)$ denotes a complicated function with parameters related to i, j, and k. It is negligible compared with the other terms in $\mathcal{T}_i(w)$, thus, it is ignored in D2. As we shall see shortly in trace-driven evaluations, D2 with this approximation already achieves a near-optimal performance in a variety of settings.

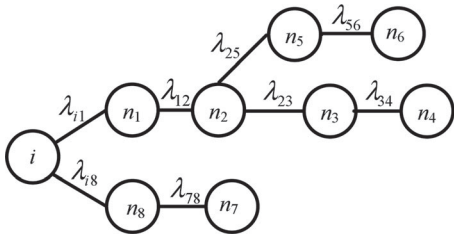


Fig. 7. An example of the general multi-hop case.

We then extend Theorem 3 to the scenario of multiple 1-hop neighbors:

Corollary 1. *When $r = 1$, device i has w units of workload and has n 1-hop neighbors, say $i_1, i_2, \ldots, i_n$; if $\Phi_{i_k}(t)/r_{i_k} < w/r_i$, $\forall 1 \le k \le n$, then we have:*

$$\begin{aligned}\mathcal{T}_i(w) = \frac{1}{r_i + \sum_{k=1}^n r_{i_k}}\Bigg(w + \sum_{k=1}^n \Phi_{i_k}(t) + \sum_{k=1}^n \Bigg(\frac{r_{i_k}}{\lambda_{ii_k}} \\ \Bigg(e^{-\frac{\lambda_{ii_k}\left(\sum_{k=1}^n r_{i_k}\right)}{r_{i_k} \cdot r_{i_k}}\Phi_{i_k}(t)} - e^{-\frac{\lambda_{ii_k}\left(\sum_{k=1}^n r_{i_k}\right)}{r_i \cdot r_{i_k}}w}\Bigg)\Bigg) + \Delta(i, i_1, \ldots, i_n).\end{aligned}$$

The general case. Theoretically, we can obtain $\mathcal{T}_i(w)$ for the general case with similar analyses. However, the delay from device i to another device $h \in N_i^r \setminus N_i^1$ follows hypoexponential distribution (see Equ. (2)), which greatly complicates the computation of $\mathcal{T}_i(w)$. Therefore, we resort to some heuristic solution.

Fortunately, the mean of the hypoexponential distribution in Equ. (2) is $\sum_{k=1}^r 1/\lambda_k$ (as indicated in Equ. (3)), which motivates us to approximates this hypoexponential distribution with an exponential distribution with parameter

$$\lambda_{ih} = 1/\omega(P_{ih}),$$

and treats each device $h \in N_i^r \setminus N_i^1$ as a 1-hop neighbor of i. Then, we can apply Corollary 1 to this case.

It is better to explain the general case with an example. In Fig. 7, suppose that D2 maintains 4-hop information at individual nodes. We note that, as we mentioned in Section 6.1, for each node $j \in N_i^4$, node i maintains a path P_{ij} with the smallest weight. Therefore, node i and its 4-hop neighbors form a tree rooted at i.

Take n_2 for example. Since the delay between i and n_2 follows hypoexponential distribution, which is very complicated in deriving the finish time estimator. As we have said above, we use an exponential distribution with contact rate λ_{i2} to approximate the actual distribution, where λ_{i2} defined as follows:

$$\lambda_{i2} = \frac{1}{\frac{1}{\lambda_{i1}} + \frac{1}{\lambda_{12}}}.$$

Similarly, we get the contact rate parameters for the other seven 2-hop, 3-hop, and 4-hop neighbors. After that, we can treat all these eight neighbors as 1-hop neighbors of i, and apply Corollary 1.

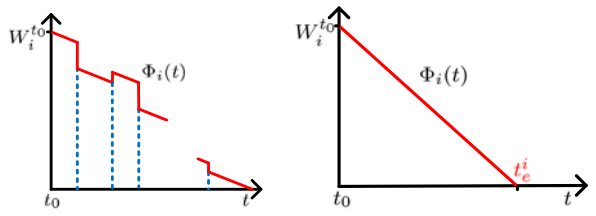


Fig. 8. Future workload estimator. (a) $\Phi_i(t) \sim t$. (b) $\Phi_i(t) = (t_e^i - t)/(t_e^i - t_0) \cdot W_i^{t_0}$.

6.3 Future Workload Estimator

Suppose that device i meets another device at slot t_0; after workload split, i has $W_i^{t_0}$ units of workload. We now present how to generate $\Phi_i(t)$ from the viewpoint of device i.

$\Phi_i(t)$ changes over time due to the following reasons: (1) i processes the workload at rate r_i, which is indicated by the slope of the oblique line in Fig. 8a; (2) i transfers some of its workload to another device in a contact, i.e., the sudden drop in Fig. 8a; (3) i receives some workload from another device in a contact, i.e., the sudden rise in Fig. 8a. Since device i cannot know the occurrence time and sequence of these contacts, or the amount of workload that is transferred in these contacts, it is extremely hard for i to accurately predict $\Phi_i(t)$. To be practical, D2 uses a straight line, as shown in Fig. 8b, to approximate the workload curve in Fig. 8a. Since we already know $\Phi_i(t_0) = W_i^{t_0}$, to represent this line, We need to estimate when the line intersects with the x-axis, i.e., t_e^i in Fig. 8b.

To locally minimize the makespan, after workload split, two devices should finish their respective workload by the same time. Therefore, if there are sufficient communication opportunities, all of the devices within r hops from i would finish their respective workload by the same time. For each device $j \in N_i^r$, since i maintains a version of $\Phi_j(t)$, by letting $\Phi_j(t_e^j) = 0$, i could estimate the finish time of j. Then, t_e^i is estimated as the average of t_e^j over all $j \in N_i^r$, that is, $t_e^i = (\sum_{j \in N_i^r} t_e^j)/|N_i^r|$. Therefore, the future workload estimator in Fig. 8b can be represented by

$$\Phi_i(t) = \frac{t_e^i - t}{t_e^i - t_0} W_i^{t_0}.$$

6.4 A Concrete Example

Fig. 9 shows the details of applying D2 to the example in Fig. 4. Without loss of generality, we assume that D2 took effect at slot -43, and a task arrives at slot 0. D2 maintains only 1-hop information at individual devices.

In Fig. 9a, $F_{01} = 40$, $f_{01} = 10$, so $\lambda_{01} = 1/4$; $F_{02} = 42$, $f_{02} = 21$, so $\lambda_{02} = 1/2$; $F_{13} = 39$, $f_{13} = 13$, so $\lambda_{13} = 1/3$; $F_{14} = 42$, $f_{14} = 21$, so $\lambda_{14} = 1/2$; $F_{23} = 40$, $f_{23} = 10$, so $\lambda_{23} = 1/4$; $F_{34} = 38$, $f_{34} = 19$, so $\lambda_{34} = 1/2$. Each device has no workload from the viewpoint of any other device. A task with 60 units of workload arrives at device 0.

In Fig. 9b, during the contact between devices 0 and 1, device 0 takes device 2 into consideration when generating $\mathcal{T}_0(w)$, where $\Phi_2(t) = 0$ from the perspective of device 0. Based on Theorem 2, we have

$$\begin{aligned}\mathcal{T}_0(x_0) &= \frac{1}{5+20}\left(x_0 + 0 + \frac{20}{1/2}\left(e^{-\frac{0}{20}} - e^{-\frac{x_0}{10}}\right)\right) \\ &= \frac{1}{25}\left(x_0 + 40\left(1 - e^{-\frac{x_0}{10}}\right)\right).\end{aligned}$$

Similarly, device 1 takes devices 3 and 4 into account when generating $\mathcal{T}_1(w)$, where $\Phi_3(t) = \Phi_4(t) = 0$ from the perspective of device 1. Based on Theorem 3, we have

$$\begin{aligned}\mathcal{T}_1(x_1) = &\frac{1}{10+5+5}\left(x_0 + 0 + \frac{5}{1/3}\left(e^{-\frac{10\times 0}{75}} - e^{-\frac{x_1}{15}}\right)\right. \\ &\left. + 0 + \frac{5}{1/2}\left(e^{-\frac{10\times 0}{75}} - e^{-\frac{x_1}{10}}\right)\right) \\ = &\frac{1}{20}\left(x_0 + 25 - 15e^{-\frac{x_1}{15}} - 10e^{-\frac{x_1}{10}}\right).\end{aligned}$$

Based on the workload split rule (see Equ. (1)), we have $x_0 + x_1 = 55$ and $\mathcal{T}_0(x_0) = \mathcal{T}_1(x_1)$. After solving these equations, we get $W_0^1 = x_0 = 26.5$ and $W_1^1 = x_1 = 28.5$. Since the last contact between them occurs at slot -3, λ_{01} is updated to $(10+1)/(40+4) = 1/4$. Also, during the contact between devices 3 and 4, since the last contact between them occurs at slot -5, λ_{34} is updated to $(19+1)/(38+6) = 5/11$.

In Fig. 9c, during the contact between devices 0 and 2, since device 1 cannot help device 0, we have $\mathcal{T}_0(w) = w/5.0$. Device 2 takes device 3 into account when generating $\mathcal{T}_2(w)$. We then have $W_0^2 = 4.2$ and $W_2^2 = 17.3$ after workload split. Since the last contact between them occurs at slot -1, λ_{02} is updated to $22/45$. During the contact between devices 1 and 3, by similar analyses, we get $W_1^2 = 13.5$ and $W_3^2 = 5.0$ after workload split. Since the last contact between them occurs at slot -4, λ_{13} is updated to $14/45$.

Fig. 9d shows the workload split results at slot 3. The makespan of D2 is 3.24 time slots. For comparison, Fig. 10 presents the results of applying the Naïve scheme to the

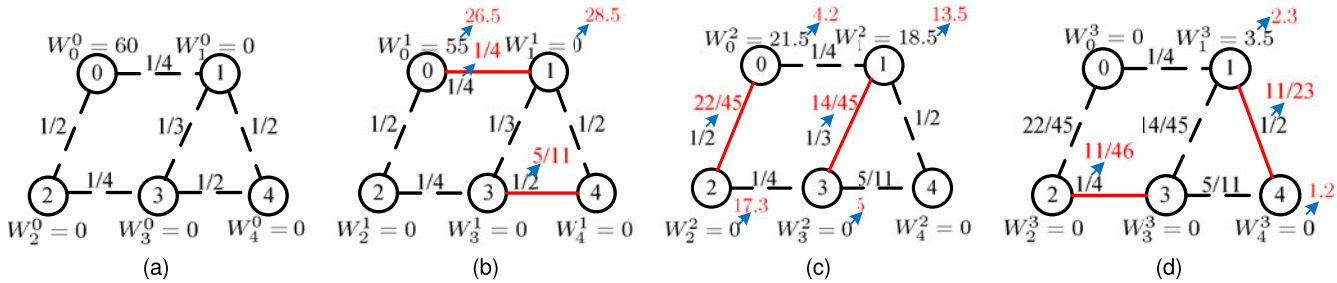


Fig. 9. The makespan of D2 is 3.24 slots. Each solid line represents a contact. Each arrow shows the change of workload or contact rate. (a) $t = 0.00$. (b) $t = 1.00$. (c) $t = 2.00$. (d) $t = 3.00$.

	Split at $t = 1.00s$		Split at $t = 2.00s$		Split at $t = 3.00s$		Comp. Time
	Before	After	Before	After	Before	After	
0	55.00	18.30	13.30	2.66	0.00	0.00	2.53
1	0.00	36.70	26.70	17.80	7.80	5.20	3.52
2	0.00	0.00	0.00	10.64	0.00	3.12	3.16
3	0.00	0.00	0.00	8.90	3.90	0.78	3.16
4	0.00	0.00	0.00	0.00	0.00	2.60	3.52

Fig. 10. The makespan of applying the Naïve scheme to the example in Fig. 9 is 3.52 slots.

same example. As we mentioned before, the Naïve scheme splits the workload between two devices, based on the ratio of their processing rates. For example, during the contact between devices 0 and 1 at slot 1, since $r_0 = 5$ and $r_1 = 10$, devices 0 and 1 get 18.3 and 36.7 units of workload, respectively. The makespan of this scheme is 3.52 slots, which is about 108.6 percent of that achieved by D2.

7 Discussion

In this section, we discuss several extensions of D2.

Incorporating redundant computing into D2. As we mentioned before, the underlying network environment below D2 is often highly dynamic and intermittently-connected. There is no centralized server that performs admission control. Therefore, it is possible for malicious devices to participate in the system. Malicious devices may claim some workload but do not send the corresponding output back. To mitigate the effect of malicious devices, we can incorporate redundant computing into D2: for each piece of workload, D2 employs multiple devices to compute it and uses the majority of the outputs returned by these devices.

Multiple simultaneous tasks. It is not hard to support multiple simultaneous tasks in D2. We can partition the computing capability of a device into K parts, which can be achieved by temporal resource partitions [23], and allocate each part to a task, where K is the maximum number of simultaneous tasks a node can handle. In doing so, the mobile devices that would like to join a task form a "virtual" DTN, in which we can run D2 for that task. For a mobile node i, if there are more than K tasks that are interesting to i, how should we choose K tasks for i? Although we could strategically select K tasks for i to maximize the overall performance, we want to expose this flexibility to the node i, and the node i can prioritize any K tasks based on its interests.

Extending D2 *to scenarios where the inter-contact time does not follow an exponential distribution.* In the following, we show how to modify D2 to deal with the case where the inter-contact time between two devices i and j follows a continuous uniform distribution with the minimum and maximum values being 0 and u_{ij}, respectively. D2 can be easily extended to the other scenarios via similar analyses, as follows.

Remember that, the key idea of D2 is to estimate the potential computing power of a device and its r-hop neighbors, which is taken into account when workload is split between two devices in a pairwise contact. For different inter-contact time distributions, we only have to get the weight of an opportunistic path and the finish time estimator. For the opportunistic path, its weight can be derived from the fact that the sum of multiple uniform distributions follows the Irwin Hall distribution [24].

Trace Name	No. of Devices	No. of Internal Contacts	Duration (Hours)
Intel [26]	9	1,364	~100
Cambridge [27]	12	4,229	~126
Infocom06 [28]	78	128,979	~93

Fig. 11. Brief summary of three realistic traces.

For the finish time estimator, we present how to deal with the first special case as in Section 6.2: $r = 1$, *and* N_i^1 *contains only one device.* We have the following theorem:

Theorem 4. *When the inter-contact time follows an uniform distribution* $[0, u_{ij}]$, $r = 1$, *device* i *has* w *units of workload and has only one 1-hop neighbor* j; *if* $\Phi_j(t)/r_j < w/r_i$, *then we have:*

$$\mathcal{T}_i(w) = \begin{cases} \frac{1}{r_i+r_j}\left(w + \frac{r_j w}{r_i} + \frac{\Phi_j^2(t)}{2r_j u_{ij}} - \frac{r_j w}{2r_j^2 u_{ij}}\right) & \text{if } \frac{w}{r_i} \le u_{ij}, \\ \frac{1}{r_i+r_j}\left(w + \frac{r_j u_{ij}}{2} + \frac{\Phi_j^2(t)}{2r_j u_{ij}}\right) & \text{if } \frac{w}{r_i} > u_{ij}. \end{cases}$$

The proof can be found in the supplemental material, available online. For the finish time estimator in the general case, we can obtain it with similar analyses.

8 Performance Evaluation

In this section, we conduct extensive trace-driven simulations to evaluate D2 under different settings and reveal insights of the proposed design performance.

We introduce four algorithms for comparison:

- RAN: in each pairwise contact, the total workload is randomly split between two devices.
- Naïve: in each pairwise contact, the total workload is split between two devices based on the ratio of their processing rates.
- mNaïve: in each pairwise contact, the total workload is split between two devices based on the ratio of the sums of their r-hop neighborhood processing rates.
- SDTA: the polynomial-time optimal algorithm that has access to global and future knowledge.

We also discuss the performance of D2 under different settings with respect to the total number of devices, the average number of 1-hop neighbors, the average contact rate, and the amount of the network information maintained at individual devices. We also show how D2 performs in scenarios where the inter-contact time follows a uniform distribution.

8.1 Experiment Setup

Our evaluations are conducted on three realistic traces and two synthetic traces. Fig. 11 summarizes the realistic traces, where mobile users with Bluetooth-enabled devices periodically detect their peers nearby, and record contacts over several days. The reason for choosing these three traces is that, the movement of devices in them follows exponential distributions, as evidenced by previous studies [25] and [21], respectively. After some preprocessing on the raw data, we find that there are few contacts during the nighttime. In order to have a meaningful and usable underlying network, we only use trace data that was collected during the daytime.

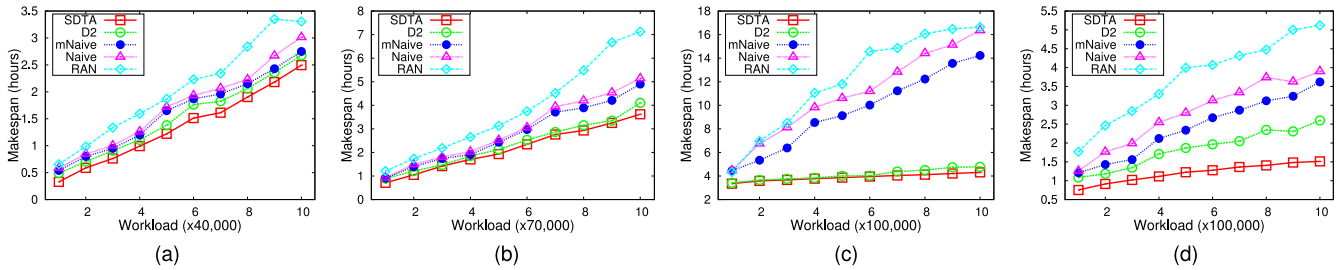


Fig. 12. Dissemination with different workloads while keeping $MaxCap = 10$. (a) Intel trace. (b) Cambridge trace. (c) Infocom06 trace. (d) Synthetic-Exp trace.

We also generate two other synthetic traces, which are denoted by Synthetic-Exp and Synthetic-Uni, respectively. In the Synthetic-Exp trace, there are N mobile devices; the inter-contact time between two devices follows an exponential distribution, with λ being uniformly generated from the range $[\frac{1}{2}AvgL, \frac{3}{2}AvgL]$; the number of 1-hop neighbors of each device is uniformly generated from the range $[AvgD - 5, AvgD + 5]$. Most of the settings in the Synthetic-Uni trace are the same as those in the Synthetic-Exp trace, except that the inter-contact time in the Synthetic-Uni trace follows a uniform distribution $[0, U]$. The defaults are set as $AvgD = 10$, $AvgL = 0.0001$, $U = 5$ hours, and $N = 100$. Here, if a device j is said to be a "1-hop neighbor" of another device i, we mean the inter-contact rate between them is positive (see Section 5.1). Therefore, the number of "1-hop neighbors" of i in a trace is also the total number of devices that i encounters for at least one time in that trace.

In our evaluations, the first 20 percent of each trace is the warmup period for mobile devices to collect and accumulate network information. Both of mNaïve and D2 maintain $r = 1$ hop information at individual nodes—unless otherwise noted. W units of workload originate at a random node in the remaining part of each trace. The processing rates of mobile devices are uniformly generated in the range from 1 to $MaxCap$. The result is averaged over multiple executions for convergence.

8.2 Performance Comparison

Impact of workload. Fig. 12 shows the comparison results with different workloads while keeping $MaxCap = 10$ in four traces. As we have mentioned in Section 4, SDTA is the optimal solution for the offline problem, and is used as the optimal benchmark in our simulations. In general, D2 achieves the second best performance in all four figures; mNaïve outperforms Naïve and RAN, since mNaïve utilizes 1-hop neighborhood information in deciding how to split the workload, while neither of Naïve and RAN does; and RAN has the worst performance.

In four figures, the makespan of D2 is at most 149, 125, 113, and 172 percent of that of SDTA, respectively, and the proposed algorithm always maintains a 10-70 percent performance advantage over the other three algorithms, i.e., mNaïve, Naïve and RAN. This is because, in D2, a mobile node exploits the distribution of the inter-contact time to predict the potential help from its r-hop neighbors; thus, the workload splitting decision is made more reasonably.

We note that the performance gap between D2 and the other four algorithms except SDTA increases when the amount of workload increases. The reason behind this phenomenon is that, both the finish time estimator and future workload estimator would be more accurate when the computation task exists longer than before.

Impact of processing rate. Fig. 13 shows the impact of processing rate. In four traces, not surprisingly, D2 achieves the second-best makespan in four traces. When the processing rates of the mobile devices increase, the makepan of every algorithm becomes smaller.

We also observe that the decrease of makespan in Figs. 12a and 12b is greater than that in 12c and 12d. For example, when $MaxCap$ increases from 5 to 50, the makespan of D2 decreases by 78 and 16 percent in the Intel and Infocom06 traces, respectively. This is due to the relatively small scale of the Intel trace: all devices in this trace probably participate in finishing the task, even when $MaxCap = 10$, while only a part of the devices in the Infocom06 trace take part in processing the task, and the number of participating devices becomes smaller when $MaxCap$ increases. From this viewpoint, the impact of $MaxCap$ on the Intel trace is greater than that on the Infocom06 trace.

8.3 Sensitivity Analyses

Fig. 14 shows the impact of the scope of network information maintained at individual devices in the infocom06 and

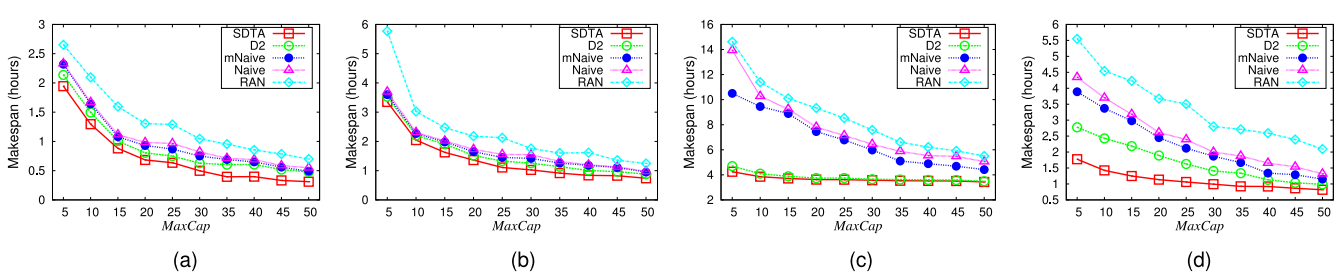


Fig. 13. Dissemination with different processing capacities. (a) Intel trace ($W = 200,000$). (b) Cambridge trace ($W = 350,000$). (c) Infocom06 trace ($W = 500,000$). (d) Synthetic-Exp trace ($W = 500,000$).

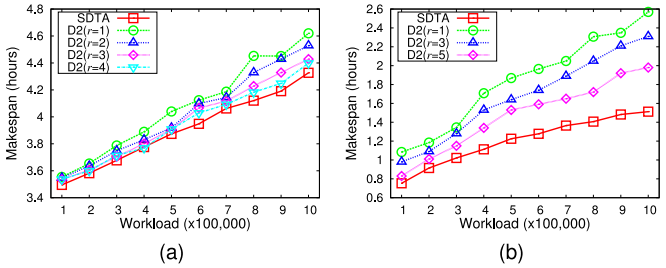


Fig. 14. Impact of the amount of network information maintained at individual nodes. (a) In Infocom06 trace. (b) In Synthetic-Exp trace.

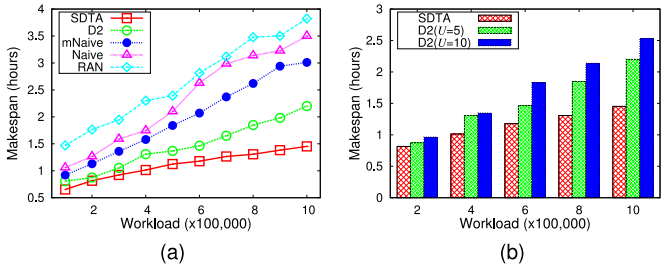


Fig. 16. Workload dissemination in the Synthetic-Uni trace. (a) Impact of W. (b) Impact of U.

Synthetic-Exp traces. We see that, D2 performs better when more information is maintained at individual devices in both traces. When the amount of workload is 1,000,000, for example, in Fig. 14a, the makespans achieved by D2$(r = 1)$ and D2$(r = 4)$ are about 106.9 and 101.6 percent of the makespan obtained by SDTA; while in in Fig. 14b, the makespans achieved by D2$(r = 1)$ and D2$(r = 5)$ are about 170.2 and 131.1 percent of the makespan obtained by SDTA. It seems the gap between D2$(r = 1)$ and SDTA is smaller in Fig. 14a than in Fig. 14b. This is because, the network scale of Fig. 14b is larger than that in Fig. 14a, which implies, 1-hop neighborhood information exposes a smaller part of network status in Fig. 14b than that in Fig. 14a.

We also find that, the marginal benefit of maintaining more information is small. This is because each device does not maintain the pairwise information among its r-hop neighbors. When r increases, the estimation accuracy of D2 would reduce. The observation could be used to strike a balance between the performance and the overhead of D2.

Figs. 15a, 15b, and 15c present the evaluation results of the impact of the number of devices (N), average node degree ($AvgD$), and average contact rate ($AvgL$), respectively. In these figures, we use the Synthetic-Exp trace, since in this way, we can flexibly change the parameters and observe its impact. In Fig. 15a, we notice that, when the number of mobile devices increase, the makespan of every algorithm decreases, as more devices mean more computing power. We notice in Fig. 15b that, when the number of nodes that a device encounters for at least one time in the trace increases, the makespan of every algorithm decreases; this is because the workload can be spread more quickly as each device has more neighbors on average. In Fig. 15c, when the average contact rate increases, i.e., the average inter-contact time decreases, there are more transfer opportunities for mobile devices to locally improve workload distribution, which is beneficial for makespan minimization.

Fig. 15d shows the impact of random prediction error. In D2, we assume that the computational capacity of each mobile node is constant over time. However, in practise, this assumption may not hold. To see how D2 responds to inconstant processing capacities, we conducted the following evaluations: during each run of D2, we make p percent of all mobile nodes have variable capacities, i.e., for a mobile node with a variable processing capacity r, in each discrete time slot, r switches to a different value between 1 and $MaxCap$. Fig. 15d shows the impact of the percentage p on the task makespan. Of course, when the percentage increases, more nodes have variable capacities, thus, the estimations in D2 become less accurate, which finally translates into the makespan increase. However, fortunately, even when there are 10 percent nodes with inconstant capacities, the makespan achieved by D2 is still within 115.9 percent of the makespan obtained when there are no inconstant capacities. This suggests that D2 is robust in settings with variable capacities.

8.4 Uniformly Distributed Inter-Contact Time

We have mentioned in Section 7 that, we only have to modify a part of the r-hop manager and the finish time estimator in extending D2 in respect to the uniformly distributed inter-contact rates.

Fig. 16a shows how the aforementioned five algorithms perform in the Synthetic-Uni trace. We find similar observations as in Fig. 12, except that the average makespan in this trace is smaller than that in Fig. 12d. The main reason is that, in the Synthetic-Uni trace, the inter-contact time between two mobile nodes follows a uniform distribution $[0, U]$; in other words, the inter-contact time between any two nodes is at most U. This upper bound would make the estimations in D2 not too far away from the true values.

We also plot the impact of the upper bound U in Fig. 16b. When U increases, the inter-contact time would increase as well; therefore, for a mobile node, the potential help from

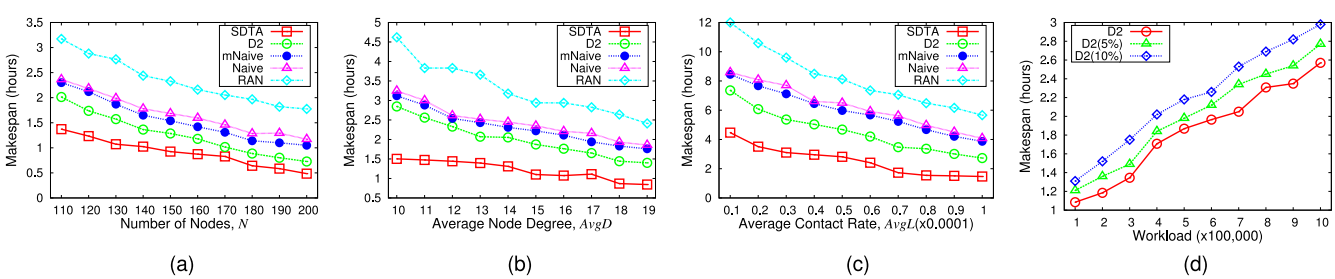


Fig. 15. Sensitivity results. (a) Impact of N (Synthetic-Exp). (b) Impact of $AvgD$ (Synthetic-Exp). (c) Impact of $AvgL$ (Synthetic-Exp). (d) Impact of the percentage of nodes with inconstant computational capacities (Infocom06).

its neighbors would get less, which finally translates into the increase in the makespan.

In summary, our simulations show that D2 performs well in a variety of settings. In future work, we believe a more sophisticated estimation of the potential computational capacity and the future workload of each node will improve our results, and perhaps bring us closer to a guaranteed performance.

9 Related Work

Our work is inspired by some previous studies on job scheduling in grid computing [29], [30]. Rossi et al. [31] proposed a meta-heuristic for scheduling a fixed set of jobs such that the difference between the completion time and the submission time of each job is less than or equal to a given threshold. Singh et al. [32] presented a multi-objective genetic algorithm for minimizing resource provision cost and optimizing application performance. Kim et al. [33] focused on designing a decentralized load balancing algorithm for a content-addressable network to match incoming jobs to available system resources. In these studies, each task has a fixed size and a fixed completion deadline; different from them, we assume that a task is fine-grained and permits arbitrary partitioning. Besides, these prior studies focused task scheduling atop organizationally-owned resources for various objectives; different from these studies, our paper extends the underlying environments to weakly-connected networks composed of mobile devices.

Similar to our work, some other studies focused on parallelism-based divisible workloads [3], i.e., workload is arbitrarily divisible. Cheng and Robertazzi [34] and Kim et al. [35] investigated the optimal workload scheduling for makespan minimization in linear and tree networks, respectively. Drozdowski and Głazek [36] provided analytical results for a similar problem in a three-dimensional mesh of processors. Different from these works, which assumed static networks with known network latencies, we study the minimum makespan scheduling problem in a highly dynamic environment where computing devices are intermittently connected.

Flooding-based epidemic routing [37] was proposed to cope with the intermittent connectivity; however, this incurs an extremely high forwarding cost. Some recent work [15], [16] reduces this cost through intelligent relay selection. Intentional routing [17] translates an administrator-specified routing metric into per-packet utilities, and replicates packets to locally maximize the marginal utility. Multicasting in DTNs is considered in [20], [38], and most of them focus on tailoring unicast routing protocols to multicasting scenarios. In comparison, while these routing protocols mainly focus on message delivery ratio and delay, our work seeks to minimize the computation makespan by making full utilization of the computing power around us, which also helps us achieve economic efficiency and relieve the congested Internet.

Previous research on data dissemination and packet routing in DTNs also informed our study. Data dissemination via flooding is implemented in [39]. The publish/subscribe-based dissemination is investigated in [40]. User interests and preferences are considered in [21] and [41], respectively. Most of them focus on maximizing the number of users that receive the target data; in contrast, this paper differs from them primarily in its goal—to design a distributed workload dissemination algorithm that minimizes the makespan. Besides, the workload can be locally "consumed," while data packets must be forwarded.

10 Conclusions

In this paper, we advocate taking advantage of the computing power around us to cope with the resource-constrained nature of mobile devices. We study the problem of minimum makespan workload dissemination over an weakly-connected network, for which we propose a distributed dissemination protocol, D2. With D2, each node maintains limited neighborhood information and updates and propagates it whenever there is an opportunity for communication. This kind of information is used to predict the potential computing capacities and future workloads of nodes, based on which workload is strategically split between two nodes in a contact, in an effort to minimize makesapn. Extensive simulations confirm the effectiveness of D2.

Acknowledgments

The authors thank the reviewers for their insightful suggestions. This work was supported in part by NSFC (61502224, 61472181, 61321491, and 61202113), China Postdoctor Science Fund (2015M570434), Key Project of Jiangsu Research Program (BE2013116), and Collaborative Innovation Center of Novel Software Technology and Industrialization.

References

[1] F. Liu, P. Shu, H. Jin, L. Ding, J. Yu, D. Niu, and B. Li, "Gearing resource-poor mobile devices with powerful clouds: Architectures, challenges, and applications," *IEEE Wireless Commun.*, vol. 20, no. 3, pp. 14–22, Jun. 2013.

[2] K. Fall, "A delay-tolerant network architecture for challenged internets," in *Proc. Conf. Appl. Technol. Archit. Protocols Comput. Commun.*, 2003, pp. 27–34.

[3] V. Bharadwaj, D. Ghose, and T. G. Robertazzi, "Divisible load theory: A new paradigm for load scheduling in distributed systems," *Cluster Comput.*, vol. 6, no. 1, pp. 7–17, 2003.

[4] M. Drozdowski and P. Wolniewicz, " Experiments with scheduling divisible tasks in clusters of workstations," in *Proc. 6th Int. Euro-Par Conf. Parallel Process.*, 2000, vol. 1900, pp. 311–319.

[5] V. Vazirani, *Approximation Algorithms*. New York, NY, USA: Springer, 2004.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA, USA: MIT Press.

[7] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless Commun. Mobile Comput.*, vol. 13, no. 18, pp. 1587–1611, 2013.

[8] G. Huerta-Canepa and D. Lee, "A virtual cloud computing provider for mobile devices," in *Proc. ACM Workshop Mobile Cloud Comput. Serv.: Soc. Netw. Beyond*, 2010, pp. 6–11.

[9] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Cloudlets: Bringing the cloud to the mobile user," in *Proc. 3rd ACM Workshop Mobile Cloud Comput. Serv.* 2012, pp. 29–36.

[10] E. E. Marinelli, "Hyrax: Cloud computing on mobile devices using mapreduce," Master Thesis, Comput. Sci. Dept., CMU, Pittsburgh, PA, USA, Tech. Rep. CMU-CS-09-164, 2009.

[11] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct.–Dec. 2009.

[12] W. Li, Y. Zhao, S. Lu, and D. Chen, "Mechanisms and challenges on mobility-augmented service provisioning for mobile cloud computing," *IEEE Commun. Mag.*, vol. 53, no. 3, pp. 89–97, Mar. 2015.
[13] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "Seti@home: an experiment in public-resource computing," *Commun. ACM*, vol. 45, no. 11, pp. 56–61, 2002.
[14] B. Knispel, B. Allen, J. Cordes, J. Deneva, D. Anderson, C. Aulbert, N. Bhat, O. Bock, S. Bogdanov, A. Brazier, et al., "Pulsar discovery by global volunteer computing," *Science*, vol. 329, no. 5997, pp. 1305–1305, 2010.
[15] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, "Spray and wait: An efficient routing scheme for intermittently connected mobile networks," in *Proc. ACM SIGCOMM Workshop Delay-Tolerant Netw.*, 2005, pp. 252–259.
[16] V. Erramilli, M. Crovella, A. Chaintreau, and C. Diot, "Delegation forwarding," in *Proc. ACM MobiHoc*, 2008, pp. 251–260.
[17] A. Balasubramanian, B. Levine, and A. Venkataramani, "DTN routing as a resource allocation problem," in *Proc. Conf. Appl. Technol. Architectures Protocols Comput. Commun.*, 2007, pp. 373–384.
[18] J. Haartsen, M. Naghshineh, J. Inouye, O. J. Joeressen, and W. Allen, "Bluetooth: Vision, goals, and architecture," *ACM SIGMOBILE Mobile Comput. Commun. Rev.*, vol. 2, no. 4, pp. 38–45, 1998.
[19] D. Yang, G. Xue, X. Fang, and J. Tang, "Crowdsourcing to smartphones: Incentive mechanism design for mobile phone sensing," in *Proc.18th Annu. Int. Conf. Mobile Comput. Netw.*, 2012, pp. 173–184.
[20] W. Gao, Q. Li, B. Zhao, and G. Cao, "Social-aware multicast in disruption-tolerant networks," *IEEE/ACM Trans. Netw.*, vol. 20, no. 5, pp. 1553–1566, Oct. 2012.
[21] W. Gao and G. Cao, "User-centric data dissemination in disruption tolerant networks," in *Proc. IEEE INFOCOM* 2011, pp. 3119–3127.
[22] S. M. Ross, *Introduction to Probability Models*. New York, NY, USA: Academic, 2006.
[23] A. K. Mok, X. Feng, and D. Chen, "Resource partition for real-time systems," in *Proc. IEEE 7th Real-Time Technol. Appl. Symp.*, 2001, pp. 75–84.
[24] N. L. Johnson and S. Kotz, *Distributions in Statistics: Continuous Univariate Distributions: Vol.: 2*. Boston, MA, USA: Houghton Mifflin, 1970.
[25] T. Karagiannis, J.-Y. Le Boudec, and M. Vojnovic, "Power law and exponential decay of intercontact times between mobile devices," *IEEE Trans. Mobile Comput.*, vol. 9, no. 10, pp. 1377–1390, Oct. 2010.
[26] J. Scott, R. Gass, J. Crowcroft, P. Hui, C. Diot, and A. Chaintreau, Intel trace [Online]. Available: http://crawdad.cs.dartmouth.edu/cambridge/haggle/imote/intel, Aug. 2009.
[27] J. Scott, R. Gass, J. Crowcroft, P. Hui, C. Diot, and A. Chaintreau. Cambridge trace from [Online]. Available: http://crawdad.cs.dartmouth.edu/cambridge/haggle/imote/cambridge, Aug. 2009.
[28] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott, "Impact of human mobility on opportunistic forwarding algorithms," *IEEE Trans. Mobile Comput.*, vol. 6, no. 6, pp. 606–620, Jun. 2007.
[29] F. Berman, G. Fox, and A. J. Hey, *Grid Computing: Making the Global Infrastructure a Reality*, vol. 2. Hoboken, NJ, USA: Wiley, 2003.
[30] D. Anderson, "BOINC: A system for public-resource computing and storage," in *Proc. IEEE/ACM Int. Workshop Grid Comput.* Nov. 2004, pp. 4–10.
[31] A. Rossi, A. Singh, and M. Sevaux, "A metaheuristic for the fixed job scheduling problem under spread time constraints," *Comput. Oper. Res.*, vol. 37, no. 6, pp. 1045–1054, 2010.
[32] G. Singh, C. Kesselman, and E. Deelman, "A provisioning model and its comparison with best-effort for performance-cost optimization in grids," in *Proc. 16th Int. Symp. High Perform. Distrib. Comput.*, 2007, pp. 117–126.
[33] J.-S. Kim, P. Keleher, M. Marsh, B. Bhattacharjee, and A. Sussman, "Using content-addressable networks for load balancing in desktop grids," in *Proc.16th Int. Symp. High Perform. Distrib. Comput.* 2007, pp. 189–198.
[34] Y.-C. Cheng and T. G. Robertazzi, "Distributed computation with communication delay (distributed intelligent sensor networks)," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 24, no. 6, pp. 700–712, Nov. 1988.
[35] H. J. Kim, G.-I. Jee, and J. G. Lee, "Optimal load distribution for tree network processors," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 32, no. 2, pp. 607–612, Apr. 1996.
[36] M. Drozdowski and W. Głazek, "Scheduling divisible loads in a three-dimensional mesh of processors," *Parallel Comput.*, vol. 25, no. 4, pp. 381–404, 1999.
[37] A. Vahdat and D. Becker, "Epidemic routing for partially-connected ad hoc networks," Duke Univ., Durham, NC, USA, Tech. Rep. CS-200006, 2000.
[38] W. Zhao, M. Ammar, and E. Zegura, "Multicasting in delay tolerant networks: Semantic models and routing algorithms," in *Proc. ACM SIGCOMM Workshop Delay-Tolerant Netw.*, 2005, pp. 268–275.
[39] G. Karlsson, V. Lenders, and M. May, "Delay-tolerant broadcasting," in *Proc. ACM SIGCOMM Chants*, 2006, pp. 369–381.
[40] W. Gao, G. Cao, A. Iyengar, and M. Srivatsa, "Cooperative caching for efficient data access in disruption tolerant networks," *IEEE Trans. Mobile Comput.*, vol. 13, no. 3, pp. 611–625, Mar. 2014.
[41] K.-J. Lin, C.-W. Chen, and C.-F. Chou, "Preference-aware content dissemination in opportunistic mobile social networks," in *Proc. IEEE INFOCOM* 2012, pp. 1960–1968.

Sheng Zhang received the BS and PhD degrees from Nanjing University in 2008 and 2014, respectively. He is currently an assistant professor in the Department of Computer Science and Technology, Nanjing University. He is also a member of the State Key Lab. for Novel Software Technology. His research interests include cloud computing and mobile networks. To date, he has published more than 20 papers, including those that have appeared in the *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *Computer Networks*, ACM MobiHoc, and IEEE INFOCOM. He received the Best Paper Runner-Up Award from IEEE MASS 2012. He is a member of the IEEE.

Jie Wu (F'09) is the chair and a Laura H. Carnell professor in the Department of Computer and Information Sciences, Temple University. He is also an Intellectual Ventures endowed visiting chair professor at the National Laboratory for Information Science and Technology, Tsinghua University. Prior to joining Temple University, he was a program director at the US National Science Foundation and was a distinguished professor at Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. He regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including the *IEEE Transactions on Service Computing* and the *Journal of Parallel and Distributed Computing*. He was a general co-chair/chair for IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, and ACM MobiHoc 2014, as well as a program co-chair for IEEE INFOCOM 2011 and CCF CNCC 2013. He was an IEEE Computer Society Distinguished visitor, ACM Distinguished speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). He received the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award. He is a CCF Distinguished speaker and a fellow of the IEEE.

Sanglu Lu received the BS, MS, and PhD degrees from Nanjing University in 1992, 1995, and 1997, respectively, all in computer science. She is currently a professor in the Department of Computer Science and Technology and the State Key Laboratory for Novel Software Technology. Her research interests include distributed computing, wireless networks, and pervasive computing. She has published more than 80 papers in referred journals and conferences in the above areas. She is a member of the IEEE.