Minimum Makespan Workload Dissemination in DTNs: Making Full Utilization of Computational Surplus Around

Sheng Zhang
State Key Lab. for Novel Software Technology
Nanjing University, China
ynifbs215@gmail.com

Jie Wu
Department of Computer and Information Sciences
Temple University, USA
jiewu@temple.edu

Sanglu Lu
State Key Lab. for Novel Software Technology
Nanjing University, China
sanglu@nju.edu.cn

ABSTRACT

This paper poses the following problem: given a task that originates at some node in a Delay Tolerant Network (DTN), how are we to disseminate the workload during pairwise contacts to minimize the makespan? We first investigate the scenario in which each node has access to an oracle that knows global and future knowledge of node mobility, and we propose a centralized polynomial-time optimal algorithm. We then develop a distributed dissemination protocol, $D2$, which maintains r-hop neighborhood information at individual nodes. D2 makes dissemination decisions based on the estimations of the potential computational capacities and the future workloads of DTN nodes. Using trace-driven simulations, we show that, D2 with only 1-hop information is already near-optimal in a wide variety of environments, and the performance gap becomes smaller as the amount of information maintained at individual nodes increases.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless Communication

Keywords

Crowd computing; delay tolerant networks; makespan; workload dissemination

1. INTRODUCTION

The last few years have witnessed an explosive proliferation of personal wireless devices, whose communication ranges are much smaller than their roaming regions. Due to the unpredictable node mobility, devices can only contact each other opportunistically. Hence, the network around us is in essence intermittently-connected, and is a type of Delay Tolerant Network (DTN) [5]. Much existing DTN work has been devoted to message routing [2], content distribution [6], and cellular traffic offloading [8], however, little attention has been given to utilizing computational resources in DTNs. Indeed, the computational capacity of a single device is generally quite small compared to the scale of tasks we have to handle at work or in scientific research; therefore, it would take a long time for us to complete a task. Inspired by crowd computing (e.g., the SETI@home [1] project exploits the massive idle computing resources across the Internet to analyze radio signal data), in this paper, we propose to disseminate the workload within the DTN around us, so that multiple devices can collaboratively finish a task, which may greatly reduce the makespan.

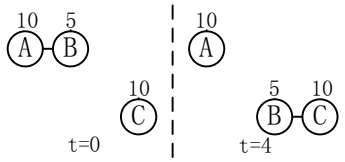


Figure 1: Two snapshots of a DTN with three nodes. The processing rate of each node is written next to the respective node that represents it.

Compared with offloading parts of the workload to remote clouds, disseminating workload around us has some desirable properties. First and foremost, users do not need to pay any service fee to cloud providers, and can achieve economic efficiency through opportunistically sharing computational resources. Second, it is ad-hoc and can be used in cases where there are no infrastructure-based services, e.g., in a disaster. Third, the Internet is relieved to some extent, since uploading tasks and related data to clouds may take up a large amount of bandwidth.

This paper poses the following question: given a task that originates at some node in a DTN, how are we to disseminate the workload during pairwise contacts, so as to minimize the makespan? Unlike the classical minimum makespan scheduling problem [7], 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. We use the example in Fig. 1 to illuminate the challenges in designing a distributed protocol. There are only two contacts in this DTN, i.e., A meets B at the beginning of slot 0, and B meets C at the beginning of slot 4. The processing rate of each node is written next to the respective node that represents it. For instance, A can finish 10 units of workload in one slot. Suppose that nodes A, B, and C have 300, 0, and 50 units of workload at the beginning of slot 0, respectively. We then show two possible dissemination schemes. Fig. 2(a) shows a naïve scheme, where the workload is split between two nodes in a contact

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MobiHoc'13, July 29–August 1, 2013, Bangalore, India.
Copyright 2013 ACM 978-1-4503-2193-8/13/07 ...$15.00.

node \ time	0	0′	4	4′	completion time
A	300	200	160	160	20
B	0	100	80	30	10
C	50	50	10	60	10

(a) Naïve scheme: the makespan is 20 slots.

node \ time	0	0′	4	4′	completion time
A	300	145	105	105	14.5
B	0	155	135	48.3	13.7
C	50	50	10	96.7	13.7

(b) D2 protocol: the makespan is 14.5 slots.

Figure 2: Comparison of two schemes. The numbers indicate the amount of workload in each node before workload splitting ($t = 0, 4$) and after workload splitting ($t = 0', 4'$).

based on the ratio of their processing rates. That is, node A transfers 100 units of workload to B since, in this way, A and B would finish their respective workloads by the same time. The makespan of this scheme is 20 slots. Fig. 2(b) shows the protocol developed in this paper, where the impact of the future contact between B and C is taken into account when B meets A. To measure the impact, B needs to know not only when the contact will emerge, but also how many units of workload that C has. In our protocol, node A transfers 155 units of workload to B at the beginning of slot 0, which reduces the makespan to 14.5 slots.

In this paper, to gain a better understanding of the problem, we first investigate the scenario where each node has 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 Dijkstra shortest path algorithm [4]. The proposed centralized algorithm is proven to be optimal and serves as the comparison benchmark in extensive simulations.

With the insights obtained from the oracle case, we then develop a distributed dissemination protocol, $D2$, 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, 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 slot. 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 a wide variety of environments, and the performance gap becomes smaller as the amount of information maintained at individual nodes increases.

2. MODEL AND PROBLEM

Task Model. This paper considers a type of task that has the following two properties. One is that the output of (part of) a task is small, which makes it possible for participating devices to send their respective output to the task source through long distance communications. Thus, this paper mainly concentrates on the workload disseminating process and does not care about the result gathering phase. The other property is that the workload of a task is fine-grained and can be parsed into arbitrarily small chunks. When two devices have a contact, their total workloads can be redistributed in any ratio between them.

These two properties are not made-up. For example, in the SETI@home [1] project, the observation data from the Arecibo radio telescope is divided into extremely small pieces, which are then assigned to the volunteers; a volunteer only has to report whether there are abnormal signals in a piece of data after analyzing it. Another example of this type of task could be an evaluation of the delivery ratio of a routing protocol on a data trace. To make the evaluation thoroughly, the routing protocol needs to be executed 100 times for each of the 1,000 different combinations of protocol parameters. In this example, we can see that the output (i.e., the delivery ratio) produced in a single execution is very small, and a single execution can be used as the smallest indivisible unit of workload.

Network Model. We model a delay tolerant network as a graph $G = (V, E)$. The vertex set V consists of all the mobile devices/users/nodes. 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, and is not affected by the execution of the proposed protocols developed in this paper. The edge set E represents the stochastic contacts between devices. The inter-contact time between i and j is assumed to be exponentially distributed with the contact rate λ_{ij}. 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 do not consider storage and bandwidth constraints in this paper.

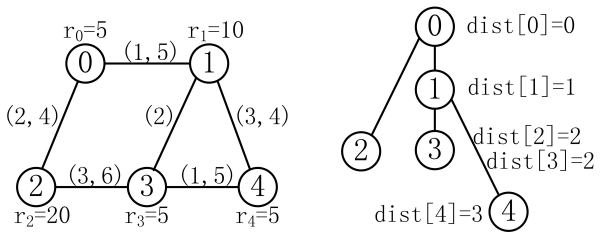
The Minimum Makespan Workload Dissemination Problem. As a starting point, this paper focuses on the problem of disseminating the workload from a single task in an empty DTN, where by "empty" we mean that: 1) when the single task originates at its source, all of the other nodes do not have any units of unfinished workload, and 2) before the single task is finished, there are no more tasks that would originate in the DTN.

To put it formally, the single task C consists of W units of workload; time is partitioned into slots of equal length. Denote the amount of workload in node i at the beginning of slot t (or slot t for short without causing confusion) as W_i^t. Without loss of generality, we assume that, the task C originates at its source $s \in V$ at slot 0. Since we consider the case where there is only one task, we have $W_s^0 = W$, and $\forall i \in V \setminus \{s\}$, $W_i^0 = 0$.

The completion time T, also called the makespan, is defined as the time difference between the origin time, i.e., slot 0, and the finish time, i.e., the time point when all nodes finish their respective workload that belongs to C. That is, $T = min\ t$, subject to $W_i^t = 0, \forall i \in V$. Our problem is how to disseminate the workload during pairwise contacts so as to minimize the makespan T.

3. OPT: CENTRALIZED POLYNOMIAL TIME OPTIMAL ALGORITHM

In this section, we introduce the centralized polynomial time optimal algorithm for the case that each node has global and future knowledge. The basic idea is to utilize the computational capacity of each node as early as possible and



(a) Discrete contact graph (b) Shortest delay tree

Figure 3: A numerical example of the OPT algorithm.

make sure that all of the participating nodes finish their respective workloads at the same time. Based on the Dijkstra shortest path algorithm, we can construct a shortest-delay tree, which encodes the shortest delays from the task source to all of the other nodes. Each node then makes workload dissemination decisions according to the tree. The details are omitted due to space limitations.

Fig. 3(a) shows a DTN that we will use throughout this paper. The processing rate of each node is labeled next to the respective node that represents it, and the contact opportunities of each pair of nodes are written next to the respective edge that represents it. For example, node 0 contacts node 1 at slots 1 and 5. Suppose that node 0 has 60 units of workload at slot 0, how are we to disseminate them?

As shown in Fig. 3(b), we first build the shortest delay tree; after simple calculations, we have $assignment[0] = 15$, $assignment[1] = 20$, $assignment[2] = 20$, $assignment[3] = 5$, and $assignment[4] = 0$. That is, at the beginning of slot 1, node 0 transfers 25 units of workload to node 1; at the beginning of slot 2, node 0 transfers 20 units of workload to node 2, and node 1 transfers 5 units of workload to node 3.

4. D2: DISTRIBUTED DISSEMINATION PROTOCOL

This section provides an overview of the D2 protocol. The details are omitted due to space limitations. Fig. 4 shows the architecture of the D2 protocol.

Workload queue. The workload queue manages operations on the actual workload in four aspects: 1) it accumulates the output of the finished workload and sends it to a source-specified server via long-distance communications at the proper time; 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.

The r-hop neighborhood information manager. Denote the set of nodes that are within r hop(s) of node i as N_i^r, where $r \in [1, R]$, and R is the network diameter that represents the hop count of the longest shortest path among any two nodes in a DTN. For each node $k \in N_i^r$, this manager is responsible for storing and updating the contact rate λ_{ik}, the workload $\Phi_k(t)$, and the opportunistic path P_{ik}. When node i comes in contact with j, for each node $k \in N_i^r \setminus N_j^{r-1}$, node i keeps P_{ik} unchanged; for each $k \in N_j^{r-1} \setminus N_i^r$, node i initializes P_{ik} by catenating i and P_{jk} (denoted as $i + P_{jk}$); for each node $k \in N_j^{r-1} \cap N_i^r$, node i updates P_{ik} to the path with the smaller weight among P_{ik} and $i + P_{jk}$.

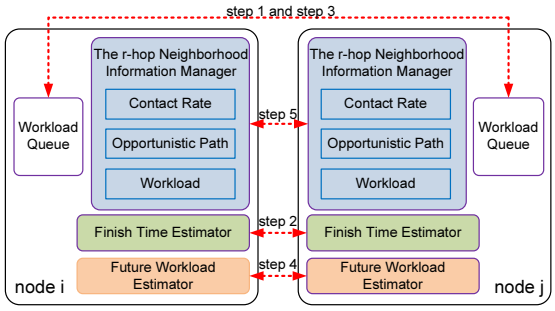


Figure 4: The architecture of D2.

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 node i to finish w units of workload. We note that it is non-trivial to obtain $\mathcal{T}_i(w)$, since the processing capacity and the amount of workload in all nodes that are within r hop(s) from node i should be taken into account. For example, suppose that node 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:

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

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

Take Fig. 4 for example; we illustrate how D2 works when nodes i and j meet at time slot t: (1) After neighbor discovery, i and j exchange the information about the amount of their respective workloads, i.e., W_i^t and W_j^t. (2) Node i generates $\mathcal{T}_i(w)$ based on the information (i.e., λ_{ik}, r_k, and $\Phi_k(t)$) of all nodes in $N_i^r \setminus \{j\}$; node j generates $\mathcal{T}_j(w)$ based on the information (i.e., λ_{jh}, r_h, and $\Phi_h(t)$) of all nodes in $N_j^r \setminus \{i\}$. Without loss of generality, suppose that $W_i^t < W_j^t$, then node j sends the function $\mathcal{T}_j(w)$ to i. (3) In order to locally minimize the makespan, their total workloads should be re-distributed in such a way that two nodes finish their separate parts by the same time. Specifically, denote the amounts of workload that nodes i and j would get after workload splitting by x_i and x_j, respectively; then, x_i and x_j should satisfy:

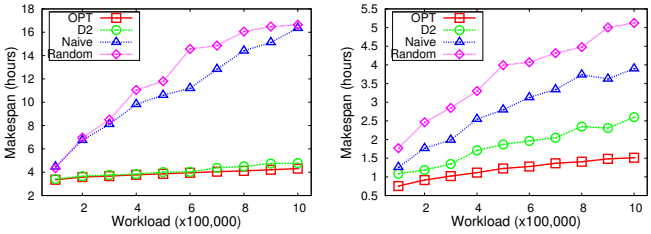
$$x_i + x_j = W_i^t + W_j^t \quad \text{and} \quad \mathcal{T}_i(x_i) = \mathcal{T}_j(x_j)$$

Node i returns the results to node j. The workload queues then complete the necessary workload transfers. (4) Nodes i and j generate $\Phi_i(t)$ and $\Phi_j(t)$, respectively. Node i updates its maintained copy $\Phi_j(t)$ to be the new one generated by j, and node j does the same in a similar way. (5) For each node $k \in N_j^{r-1}$, node i updates $\Phi_k(t)$ and P_{ik}; node j updates the corresponding information in a similar way.

Applying D2 to Fig. 3(a), the makespan is 3.34 slots, while the makespan of the Naïve scheme is 3.53 slots.

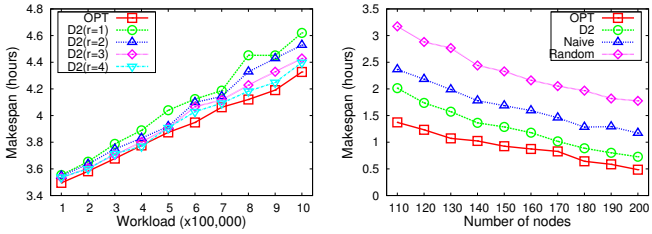
5. PERFORMANCE EVALUATION

This section evaluates the performance of D2, which is compared with the following dissemination algorithms: *OP-*



(a) Infocom06 trace (b) Synthetic trace

Figure 5: Performance comparison under different workloads while keeping $MaxCapacity = 10$.



(a) Impact of r (Infocom06) (b) Impact of N (Synthetic)

Figure 6: Sensitivity results.

T, the polynomial-time optimal algorithm that has access to global and future knowledge; *Naïve*, the total workload is split between two nodes in a contact based on the ratio of their processing rates; and *Random*, the total workload is randomly split between two nodes in a contact.

Simulation Setup. Our evaluations are conducted on three realistic traces and a synthetic trace. Due to space limitations, we only present results on the Infocom06 [3] and synthetic traces. In the Infocom06 trace, mobile users with Bluetooth-enabled devices periodically detect their peers nearby and record contacts over several days. After some preprocessing on the raw data, we find that there are few contacts made during the nighttime. In order to have a meaningful and usable DTN, we only use trace data that was collected during the daytime.

Since the scale of these traces is relatively small and cannot be flexibly tuned, we also generate a synthetic trace with N nodes: the number of 1-hop neighbors of a node is uniformly generated from the range $[AvgDeg-5, AvgDeg+5]$; the inter-contact time between two nodes follows exponential distribution, with λ being uniformly generated from the range $[1/2 \cdot AvgLambda, 3/2 \cdot AvgLambda]$. Based on the observations from the realistic traces, the defaults are set as $AvgDeg = 10$, $AvgLambda = 0.0001$, and $N = 100$. D2 maintains $r = 1$ hop information at individual nodes—unless otherwise noted. In each trace, the processing rates of mobile devices are uniformly generated from the range $[1, MaxCapacity]$, and W units of workload originate at a randomly selected node at the beginning of the partial trace that we use. The result is averaged over multiple running times for statistical convergence.

Simulation Results. Fig. 5 shows the comparison results under different workloads in two traces, while keeping $MaxCapacity = 10$. In general, D2 achieves a near-optimal performance and outperforms Naïve and Random. Specifically, the makespan in D2 is within 113% and 172% of that in OPT in the two traces, respectively. With various workloads, D2 maintains a 10%-70% performance advantage over Naïve and Random. We also notice that the advantage becomes greater when the amount of workload goes up. The reason behind this phenomenon is that D2 makes decisions by estimating the expected finish time and the future workload of each node, and the estimation becomes more accurate when the computation task lasts for a longer time.

We are also interested in evaluating the impact of the scope of network information maintained at individual nodes. Fig. 6(a) shows the simulation results. Generally, D2 performs better when more information is maintained at individual nodes. Surprisingly, we notice that D2 with 1-hop information has already received most of the performance gain under various workloads, and the marginal benefit of maintaining additional information is small. This is because each node does not maintain the information between its r-hop neighbors; when r increases, the estimation accuracy of D2 would be reduced. This insight can be used to trade-off between the performance and the overhead of D2.

Fig. 6(b) shows the impact of the number of nodes on the makespan, when there are more mobile nodes in a DTN, the makespan in all four algorithms goes down since more nodes provide more computational capacity.

In summary, our trace-driven 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 bringing us closer to guarantees of performance.

6. CONCLUSION

In this paper, we propose making full utilization of computational surplus around us, and we design a centralized optimal algorithm, OPT, and a distributed protocol, D2, for the minimum makespan workload dissemination problem in DTNs. Extensive simulations show that D2 achieves a near-optimal performance in a wide variety of settings.

Acknowledgements. This work was supported in part by NSFC (No. 61073028, No. 61202113, and No. 61021062), Key Project of Jiangsu Research Program (No. BE2010179), Jiangsu NSF(No. BK2011510), China 973 Program (No. 2009CB320705), Research and innovation project of Jiangsu (No. CXZZ12_0055), Program A for outstanding PhD candidate of Nanjing University, and US NSF (ECCS 1128209, CNS 1065444, CCF 1028167, CNS 0948184, and CCF 0830289).

7. REFERENCES

[1] SETI@home. *http://setiathome.berkeley.edu/*.
[2] A. Balasubramanian, B. Levine, and A. Venkataramani. DTN routing as a resource allocation problem. In *ACM SIGCOMM 2007*.
[3] A. Chaintreau, P. Hui, J. Scott, R. Gass, J. Crowcroft, and C. Diot. Impact of human mobility on opportunistic forwarding algorithms. *IEEE TMC 2007*.
[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2nd revised edition edition.
[5] K. Fall. A delay-tolerant network architecture for challenged internets. In *ACM SIGCOMM 2003*.
[6] W. Gao and G. Cao. User-centric data dissemination in disruption tolerant networks. In *IEEE INFOCOM 2011*.
[7] V. Vazirani. *Approximation algorithms*. Springer, 2004.
[8] X. Zhuo, W. Gao, G. Cao, and Y. Dai. Win-Coupon: An incentive framework for 3G traffic offloading. In *IEEE ICNP 2011*.