# Optimizing Downloads over Random Duration Links in Mobile Networks

Amber Bhargava, Spencer Congero, Timothy Ferrell, Alex Jones,
Leo Linsky, Jayashree Mohan, and Bhaskar Krishnamachari
Ming Hsieh Department of Electrical Engineering and Department of Computer Science
Viterbi School of Engineering, University of Southern California, Los Angeles, CA 90089, USA
Contact: bkrishna@usc.edu

***Abstract*—Short range vehicle to vehicle and device to device communications are of growing interest due to their utility for vehicular safety and infotainment applications as well as for improving the capacity of cellular networks. These mobile systems are characterized by ephemeral, stochastic links. We consider a fundamental problem in this domain — how to maximize the amount of useful content downloaded by a client from a server over an encounter that lasts a random amount of time. We assume that the distribution of link duration is known or estimated *a priori* based on historical as well as real-time measurements. We present MERLIN (Maximum Expected download over Random LINks), a single-phase file request protocol that is provably optimal. We evaluate MERLIN comprehensively via simulations based on both ideal link duration distributions as well as empirical distributions obtained from real vehicular mobility traces from Taxis in Shanghai and Buses in Chicago. We also implement MERLIN on Contiki OS and evaluate it empirically using the Tmote Sky wireless embedded platform.***

## I. Introduction

In the near-future, all cars will be equipped with dedicated short range communication (DSRC) radios allowing them to talk to other cars on the road for safety as well as also potentially for various infotainment applications [1], [2]. The cellular industry is also exploring the design of device to device (D2D) communication schemes in order to improve network capacity [3], [4]. A common challenging problem in these domains is organizing efficient communication between the radio-equipped vehicles or devices which may encounter each other for a random duration.

We consider a fundamental problem pertaining to optimizing the link layer for such encounter-based communication systems – optimizing the amount of content downloaded by a client node from a server node. We show in this work that statistical knowledge of the random encounter duration can be exploited in the link layer protocol to carefully choose how requests are made, in such a way as to maximize the efficiency of information transfer.

One important assumption in this work is that the request from a client and its response from the server are carried out over a single phase. In other words, the client does not continue to make requests once the responses to its initial request are received. This single-phase assumption is motivated by short-encounter-duration scenarios in which there is a high switching cost for the client to make multiple requests. [1]

We also assume in modeling this problem that both the mobile client and server have a consistent estimate of the statistics of the encounter (namely, the distribution of the encounter duration). Though we treat the estimation of the distribution of encounter duration as out of the scope of this work, such an estimate could be obtained in practice based on historic measurements as well as sensor data concerning the location, speed and direction of the vehicles.

For many on-road applications the content to be disseminated or downloaded by vehicles can be assumed to be highly organized. For instance, consider recent traffic data about specific map "tiles", road condition data about a set of specific road stretches, or even a list of currently popular music files. In these cases the repository consists of discrete items (that we refer to as "files" in this paper) that can be numbered.

If the server itself is obtaining the content from the cloud through a process of intermittent downloads, then it may not possess the entire repository itself at the time of the encounter with the client. Similarly, the client is assumed to have previously downloaded a random subset of the repository from other encounters. At the start of the encounter, we assume that the link discovery protocol allows the client and server node to share information with each other about the current percentage of the repository that they hold.

Because the duration of the encounter is potentially quite short, it is important for the client to be able to communicate as quickly as possible the files it still needs to the server. To compress this information, we assume that the client sends information about ranges of missing files to the server, which responds with files from the requested ranges that are available at its end.

The crux of the problem we investigate in this paper is this: because of the limited encounter duration, it may not be possible or desirable for the client to make requests containing all its needs. We model this system mathematically, taking into account the distribution of the encounter duration, and

[1]For other scenarios where such switching costs are low, it would be possible to send requests in multiple-phases; optimizing multi-phase queries is a harder problem which we consider out of the scope in this study, but is a subject of our ongoing investigations.

derive the optimal request size for the client that maximizes the expected downloaded content.

To summarize, the following are the key contributions of this work:

- We mathematically formulate the problem of optimizing a client-server structured data download protocol for a random-link-duration communication system.
- Taking into account random availability of data at both server and client, as well as the distribution of the encounter duration, we show how the requests from the client should be structured and optimized in such a way as to maximize the expected amount of content transferred. This forms the basis of our novel protocol we call MERLIN (Maximum Expected download over Random LINks).
- Through numerical simulations, we investigate how optimal number of requests for MERLIN and its efficiency vary as relevant problem parameters such as the encounter distribution, the client and server availability percentages, file size are varied.
- We show that the proposed optimized link download protocol outperforms alternative baselines on empirically derived link duration distributions.
- We present two implementations of MERLIN on Contiki OS, one where the optimal number of requests is calculated locally on the client, and one where it is calculated on a remote sever, and evaluate them empirically on Tmote Sky wireless embedded devices.
- We identify more sophisticated settings for which the corresponding modeling and optimization solutions are posed as open problems to be considered in future work.

The rest of this paper is organized as follows. In section II, we model the problem mathematically and present the relevant notation. In section III, we present and prove relevant properties of the optimal solution. In section VI, we present numerical simulations. In section VII, we describe software implementations of MERLIN on ContikiOS and in section VIII we evaluate them empirically. We place our contributions in the context of related prior work in section IX, and finally, present concluding comments including suggestions for future work in section X.

## II. Model and Notation

We consider a single encounter between two mobile nodes, one a server with partial availability of content, the other a client. The encounter time between them is modeled as a discrete random variable $T$ with a known cumulative distribution function $F_T$.

There is a set of $N$ files, and the server has an arbitrary subset of these available, denoted by the binary vector $V_S$. Let the number of non-zero elements in this vector be denoted as $n_S$. The client also has an arbitrary subset available initially, denoted by the binary vector $V_C$, with $n_C$ non-zero elements.

Each file is of size $S_F$ bytes. For simplicity, we normalize the time steps to be such that 1 byte could be transferred in a unit time step (i.e. we have a normalized link rate).

A transmission takes the form of packets. Each packet has a header size of $S_H$ bytes, and a maximum data size of $S_D$. For some positive $\rho$, $S_F = \rho S_D$. If $\rho \leq 1$, then multiple files (namely $\lfloor \frac{1}{\rho} \rfloor$ files) fit into one packet. Otherwise, multiple packets (namely $\lceil \rho \rceil$ packets) are needed to transmit each file.

At the beginning of the encounter, we assume that the server advertises the number of files it has available (i.e., $n_S$). We initially assume that the exact set of files available is equally likely to be any of the possible $\binom{N}{n_S}$ configurations. We will refer to this as the Independent File Availability (IFA) assumption.

Let $V_C(t_{end})$ represent the vector of files available at the client at the end of the response, and $n_C(t_{end})$ the corresponding number of files.

In order to assign relative utility to each file, we associated with each file $i$ a non-negative weight $w_i$. If all $w_i$ are equal, this is the special case where all files are equally important. The client's goal is to choose a set of file ranges to request from the server that maximize the expected, weighted sum of all files in the client vector, which we call the total utility $V_C$. The weighted sum of a range's file weights $R$ is denoted $W(R)$.

As noted in the introduction, with a focus on short-encounter-duration scenarios with significant switching costs, we assume that the communications between client and server are restricted to take place over a *single phase*. The client is allowed one request period; after which, the server is allowed one response period. Each transmission may consist of one or more packets.

The client requests $R_b$ contiguous ranges of needed files ($R_b$ is a design variable to be optimized). These ranges are represented by two indices. Each index requires a certain number of bytes to represent, denoted $c$. Intuitively, $c = O(logN)$. The server responds by sending all available files in these $R_b$ ranges.

## III. Optimizing Requests

Intuitively, in this problem, there is a trade-off between the size of the request and the response. If the number of files requested is too few, then all of those requests may be satisfied, but there may be idle time left over in the encounter. Alternatively, if the number of files requested is too large, then there may not be enough time for the server to send the files in response.

Let $m_r$ be the number of request packets. If $m_r = \lceil \frac{2 \cdot R_b \cdot c}{S_D} \rceil$, then $max(m_r - 1, 0)$ of the packets will be fully packed, and the last packet will be of size $s_r$ which is $2 \cdot R_b \cdot c \pmod{S_D}$.

To map the number of ranges requested to the sum of file weights within those ranges, we use a utility function $U_{V_c} : \mathbb{R} \mapsto \mathbb{R}$. $U_{V_c}(R_b)$ outputs the summed weight per byte of the files in the top (ranked in order of size) $R_b$ contiguous file ranges in the client's file vector $V_c$. Note that this indicates $U_{V_c}$ is a monotonically increasing function in $R_b$.

We consider the two possible cases:

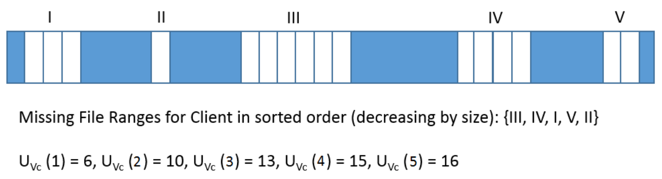

I
II
III
IV
V
Missing File Ranges for Client in sorted order (decreasing by size): {III, IV, I, V, II}
U_Vc (1) = 6, U_Vc (2) = 10, U_Vc (3) = 13, U_Vc (4) = 15, U_Vc (5) = 16

Fig. 1. Illustration of client file availability set $V_c$, with a set of missing file ranges, list of ranges sorted by size, and the corresponding utility function $U_{V_c}$

- Case 1: The client sends its request and receives all of the files that the server has within the $R_b$ requested ranges. Because we assume IFA, we can model the utility of the data received, $d_{r1}$, by the following equation. Note that this is an increasing function in $R_b$ because $U$ was shown to be monotonically increasing above.

$$d_{r1} = \frac{n_S}{N} \cdot U_{V_c}(R_b) \tag{1}$$

- Case 2: The client attempts to send its request and doesn't receive all $R_b$ requested ranges because the encounter ends either during the request or response phase. Let $F(x)$ be the top ranked $x$ files in our ranges. We do this because the server will respond with the file data in decreasing order of $w_i$. We consider the utility of received files, $d_{r2}$, in two sub-cases: (2A): $t_{request} <= t_e$ and no files are received, or (2B) $t_{request} > t_e$ and some of the files are received.

$$d_{r2} = \begin{cases} 0 & \text{(2A)} \\ W(F(\lfloor \frac{t_e - (S_H \cdot m_r + 2 \cdot R_b \cdot c)}{S_F} \rfloor)) & \text{(2B)} \end{cases} \tag{2}$$

We first present two simple claims, omitting their proofs for brevity.

### A. Claim 1:

Maximizing the expected total utility of the data downloaded is equivalent to maximizing total number of files requested within a given request period $t_{request}$, under the IFA assumption.

### B. Claim 2:

The way to request k file ranges so as to maximize their total utility within a given request period $t_{request}$ is to request the top k ranges in decreasing order of $W(\cdot)$.

**Proposition:** The maximum expected total utility of the data received from the server is given by the following equation:

$$\max_{R_b} E\left[\min(d_{r1}, d_{r2})\right] \tag{3}$$

**Proof:** Given an arbitrary value of $t_e$ from the sample space, case 1 gives the maximum expected total utility of data received. This is the expected ceiling for case 2, which calculates the utility of receiving some or none of the files requested. Because $t_e$ is a random variable, we must take the expectation of the expression in order to find the expected value. Finally, maximizing this expectation over all possible values of $R_b$ will give us the maximum expected utility. ■

## IV. The MERLIN Protocol

We propose a single-phase download request protocol, MERLIN. By utilizing the above solution in equation (3), it maximizes the expected downloaded data over the random duration link. Key inputs to the MERLIN algorithm are the prior distribution of the link encounter $F_T$, the number of available items at the server $n_S$ (which is assumed to be provided by the server to the client via a periodically broadcasted beacon message, which is used by prospective clients to detect the presence of the server in their vicinity), and $V_C$, the vector indicating which files are already available at the client and which ones are not.

We summarize MERLIN's operation in the following algorithm.

**Algorithm 1** MERLIN - single phase optimal download request protocol

```
Require: Inputs ($F_T$, $n_S$, $V_C$)
  1. Compute $R_b$ that maximizes equation (3)
  2. Client sends request packets containing $R_b$ ranges.
  3. Server responds with available files in requested ranges
```

## V. Mathematical Analysis of MERLIN

We further analyze the solution given in equation (3), which is at the core of the MERLIN protocol. Let $S(R_b, t_e)$ refer to the term inside the expectation of that expression, i.e. $S = \max(0, [\min(d_{r1}, d_{r2})])$. $S$ is in fact the downloaded data with $R_b$ requests if the encounter duration is $t_e$. Because of the randomness in the encounter duration, $S$ is a random variable. It can be expressed in simpler form (with some minor rounding approximations) as $max(0, min(\alpha \cdot U(R), t_e - \beta \cdot R))$, where $\alpha, \beta$ are simplified constants representing problem parameters ($\alpha = \frac{n_s \cdot S_F}{N}$, $\beta = 2c \cdot (1 + \frac{S_H}{S_D})$, $R = R_b$, $U = U_{V_C}$.

It can then be shown that the cumulative distribution function of $S$, $F_S(s)$ can be expressed in terms of the distribution of the encounter time $T$ as follows:

$$F_S(s) = \begin{cases} 0 & s \leq 0 \\ F_T(s + \beta \cdot R) & 0 \leq s \leq \alpha \cdot U(R) \\ 1 & s > \alpha \cdot U(R) \end{cases} \tag{4}$$

From this it can be inferred that the expected data downloaded with $R$ requests is:

$$E[S(R)] = \int_0^{\alpha \cdot U(R)} (1 - F_T(s + \beta R))ds \tag{5}$$

One special case in which equation (5) simplifies is that of an exponential encounter distribution, where $F_T(t) = 1 - e^{-\lambda t}$. In this case, we get:

$$E[S(R)] = \frac{e^{-\lambda \beta R} \cdot (1 - e^{-\lambda \alpha U(R)})}{\lambda} \tag{6}$$

Finally, in order to maximize the expected data downloaded, we can find the value of $R$ at which $E[S(R)]$ is maximized. Equation (6) is amenable to numerical calculations for this purpose, given a $U(R)$ function corresponding to some fixed client vector $V_C$, and given other problem parameters such as file size, mean encounter duration, server availability, repository size, etc. To gain further insight, it is possible to derive an (approximate) closed form expression for $E[S(R)]$ in terms of $R$, by fitting a tractable expression such as a quadratic polynomial to the utility function $U(R)$ [23].

## VI. Simulation-based Evaluation of MERLIN

We first evaluate MERLIN's performance with respect to various parameters. These include the probability distribution of the link duration, size of files, and the server and client availability ratios.

We then compare our optimized solution with some baselines using empirically-derived link duration distributions obtained from two real vehicular mobility traces: one from 632 taxis in Beijing over 24 hours, and another from 1608 buses in Chicago over 30 hours.

### A. Impact of Parameters on MERLIN Performance

We undertake a comprehensive set of simulations to evaluate MERLIN and understand how its performance varies with respect to various problem parameters.

In our simulations, the following are the default set of values (each particular experiment varies some of the parameters while keeping the others fixed at the default values):

- 500 files
- 95% server availability
- 30% client availability
- 200 mean encounter duration in time units, encounter distribution: exponential.
- 10 file size (in time units)
- 2 header size (time units)
- 2 time needed to describe each range

Figure 2 shows the percentage of time spent on different tasks as the number of requests is varied, for an Exponential encounter distribution. It can be seen that as the number of requests is increased the number of request packets increases and the amount of idle time decreases. The data transfer time initially increases then decreases indicating the presence of a particular number of requests (neither too small nor large) that is optimum.

Figure 3 shows how the optimal request size and optimal efficiency (the ratio of expected data transferred in time units to the total encounter duration) varies with the mean encounter duration for exponential distribution. As may be intuitively expected, as the mean duration of the encounter increases, there is more time for a greater number of requests as well as greater data transfer.

Figures 4 and 5 help us understand how the optimal number of requests and the corresponding efficiency vary as the percentage availability of files at the client and at the server are varied. As the number (percentage) of files already

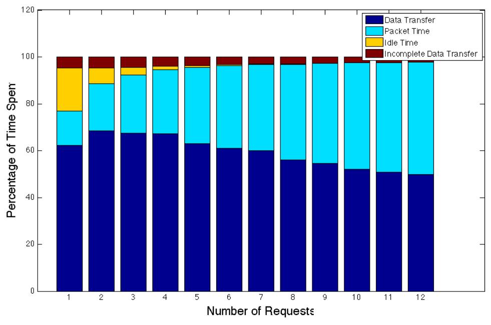


Fig. 2. Percentage of time spent on different tasks for exponential encounter distributions

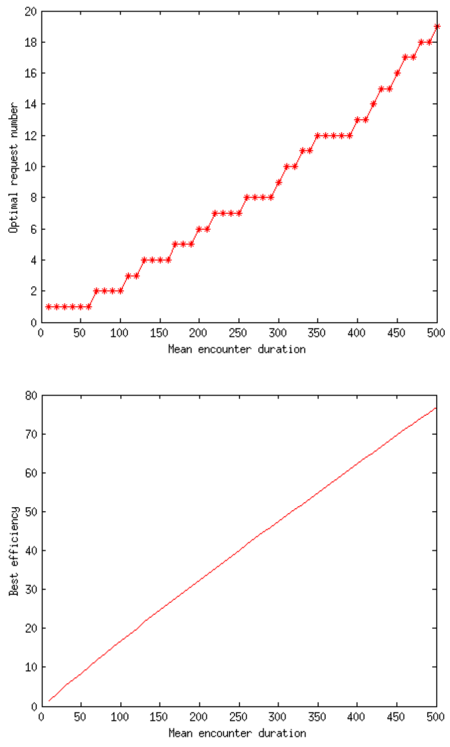


Fig. 3. Optimal request size and optimal performance (normalized expected data transferred) versus mean encounter duration for exponential distributions

available at the client increases, the requested ranges become more fragmented, more requests are needed to ask for the same number of files, incentivizing the client to send more requests, and at the same time, there is less time spent on productive data transfers. As the server availability increases, requests are more productive and efficient, and therefore fewer ranges need to be requested.

Figure 6 helps us understand how the optimal number of requests and the corresponding efficiency vary as the size of files is varied. As the file size increases, fewer file requests can be satisfied, and hence the number of requests is decreased. The efficiency initially increases due to more time spent in file transfer. Note, however, that it can eventually decrease as

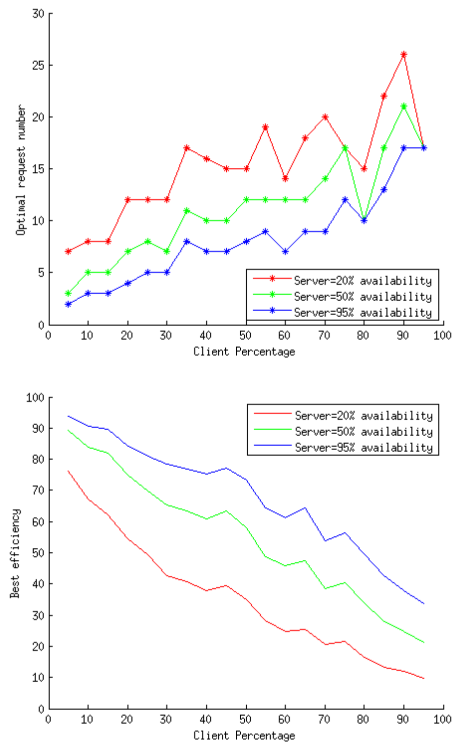


Fig. 4. Optimal request size versus client availability percentage and the optimal performance versus client availability percentage

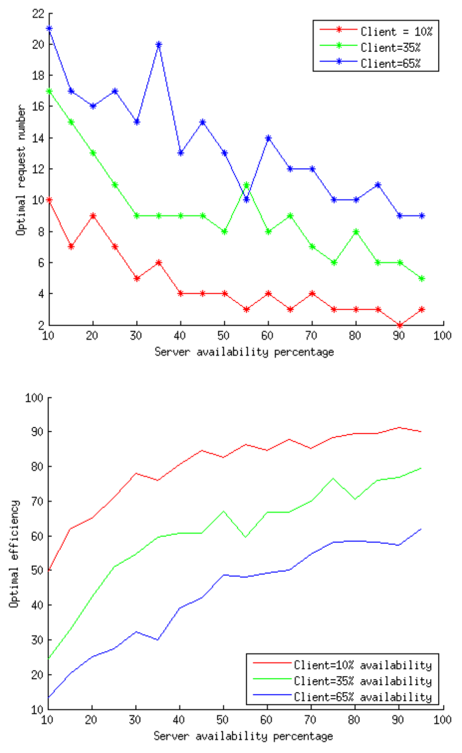


Fig. 5. Optimal request size versus server availability percentage and the optimal performance versus server availability percentage

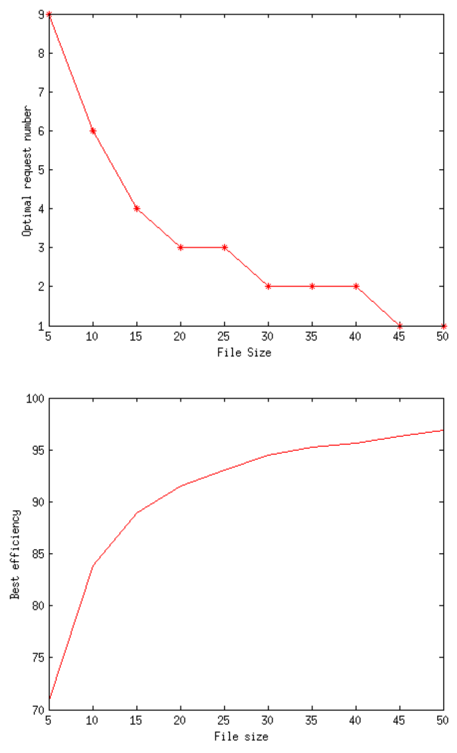


Fig. 6. Optimal request size and performance versus file size

the percentage of time downloading incomplete files increases with the file size.

## B. *Algorithms Used in Baseline Comparisons over Real Trace Distributions*

In our simulations, we compare MERLIN to two other baseline algorithms described below:

*1) Push:* In this algorithm the server sequentially sends each item that it has to the client without a request phase from the client.

*2) Deterministic OPT:* In this algorithm the shape of the contact time distribution is not taken into account, only the mean of the distribution. In other words, it is assumed that the encounter duration will be deterministically equal to the mean value, and the number of requests is chosen so as to optimize for such a deterministic encounter duration.

## C. *Structure of Real Trace Distributions*

As shown in figure 7, we look at two sets of empirical link duration distributions, one from Beijing taxis, the other from Chicago buses. In the simulations below, we compare MERLIN, Deterministic OPT and Push schemes for the distribution from the Chicago trace, and for the distribution from the Beijing trace.

## D. *Simulation: Baseline Comparisons over Real Trace Distributions*

In figures 8 and 9, we compare the efficiency of the three presented algorithms as the client availability percentage is

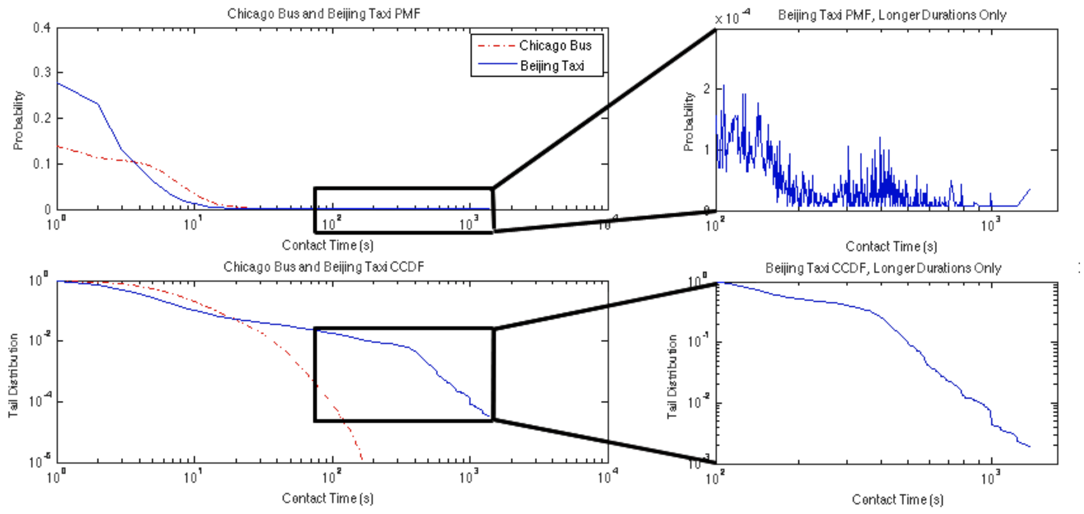
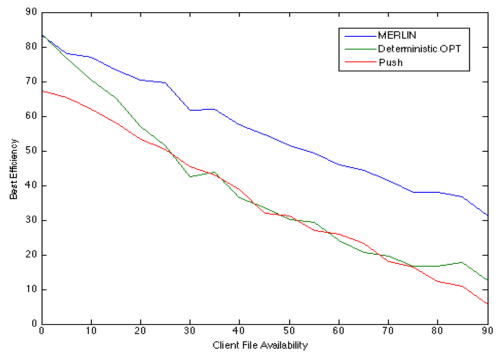
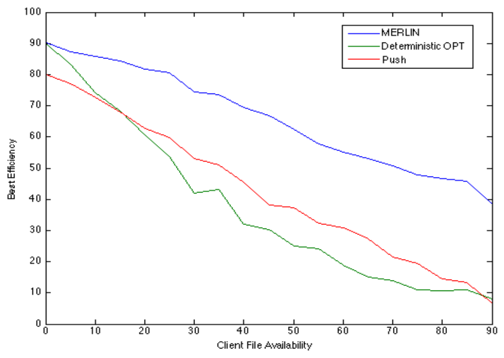


Fig. 7. Distribution of contact times for Beijing taxis and Chicago buses

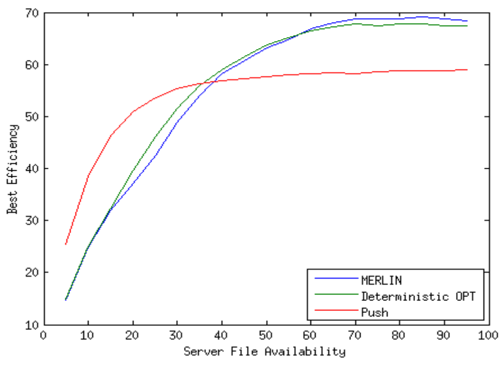


(a) Chicago

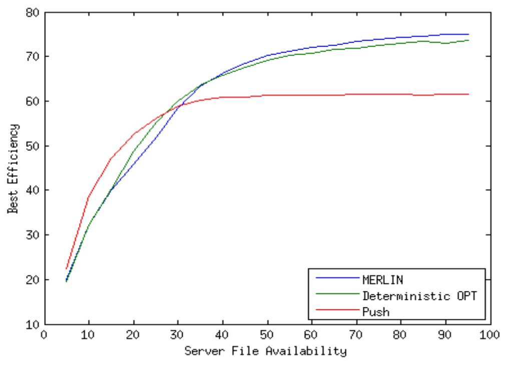


(b) Beijing - Long

Fig. 8. Comparison of efficiency of different algorithms as client availability percentage is varied, for the real trace distributions



(a) Chicago



(b) Beijing - Long

Fig. 9. Comparison of efficiency of different algorithms as server availability percentage is varied, for the real trace distributions

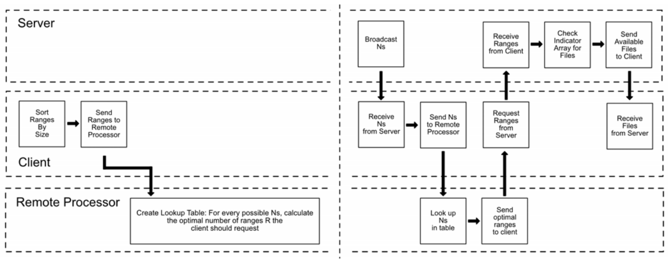

Server
Broadcast Ns
Receive Ranges from Client
Check Indicator Array for Files
Send Available Files to Client
Sort Ranges By Size
Send Ranges to Remote Processor
Client
Receive Ns from Server
Send Ns to Remote Processor
Request Ranges from Server
Receive Files from Server
Remote Processor
Create Lookup Table: For every possible Ns, calculate the optimal number of ranges R the client should request
Look up Ns in table
Send optimal ranges to client

Fig. 10. Implementation block diagram

varied, for the Chicago trace as well as for the Beijing trace (focusing on the long encounters which show the bimodal behavior). We observe that the pull-based approach adopted by MERLIN is better than the push approach except in some cases when the server availability percentage is low. This is because when the server availability is low, client pull requests by MERLIN are more likely to result in "misses" resulting in lower efficiency.

In both traces, the performance of MERLIN (which takes into account the full prior encounter distribution) appears slightly better than that of deterministic OPT particularly for larger server availability values, as MERLIN takes advantage of knowledge of the full encounter distribution.

## VII. Experimental Implementation

In order to conduct real-world experiments to test MERLIN, we implement the protocol using two Tmote Sky motes with Contiki OS, an operating system for low-power wireless devices. We are motivated to conduct these experiments to observe how the MERLIN protocol performs when computational time becomes a factor.

Firstly, in order to keep track of available and missing files, the client and server each use an indicator array of size $N$, where an entry of $1$ indicates the node possesses the file and a $0$ indicates that it does not. At the beginning of the interaction, once the client receives the server's number of available files, the client uses its file indicator array to determine all of its needed file ranges and sorts them in decreasing order. With these ranges and the number of available server files, the client can begin to use the MERLIN equation to calculate the expected number of files received based on the number of ranges it requests. Once the client has iterated over every possible number of ranges to request, it chooses the range that maximizes the expected value of files received. Finally, the client requests those ranges from the server in the form of start and end index pairs.

Once the server receives the range pairs from the client, the server uses its own indicator array to check if it has the files specified in the client's requested ranges. For any of the files it has within these ranges, the server transmits the files back to the client. In our experiment, the files did not contain actual information but were simply indexed ten-byte packets.

Then, as the client receives each file from the server, the client looks at the file index contained in the packet, checks to make sure its indicator array is actually missing that file, and updates its indicator array accordingly. Since we implement a single-phase interaction for the MERLIN protocol, the client simply collects as many files from the server as it can before the interaction duration expires, and there are no further communication events from the client whether the client receives all of the requested files or not.

Since these devices have low computational ability, we also implement the option of performing the MERLIN calculation remotely, instead of on the device itself, in order to increase the computation speed. As a further improvement, we implemented a lookup table on the remote processor as well. The block diagram for this implementation are shown in Figure 10. Before the client receives any communication from the server, it first sends its missing ranges to the remote processor, which is a piece of Python code that can read serial communication from the client through the USB port. Then, the remote processor uses these ranges and the MERLIN equation to very quickly calculate the optimal number of ranges the client should request for every possible value of $n_S$. After creating this lookup table, the remote processor simply waits.

At some later time, the client will eventually receive the actual $n_S$ from the server, which the client will then forward to the remote processor. Since the remote processor has already performed the MERLIN calculations for every possible value of $n_S$, the remote processor can use $n_S$ to retrieve the optimal value of $R$ in constant time and return this information to the client. After this step, the protocol continues in the same way as in the default implementation.

By using a lookup table on a remote processor, we test the increase in performance of the MERLIN protocol when the computation time is drastically reduced.

## VIII. Experiments

In our experiment we evaluate two different versions of the MERLIN protocol. First, we test the version where all of the calculations for the MERLIN equation are performed on the client itself. Second, we test the version where the the calculations for the MERLIN equation are performed on a remote processor and stored in a lookup table prior to server-client communication. Finally, we also compare these to the naive approach, where the client simply requests a single range of files (i.e., the largest range) during every encounter.

We perform each experiment three times, and average over the three trials for each version. When generating the graphs of our results we add a small randomized time delay between interactions in order to more accurately emulate a real-world situation where the client would only intermittently be within communication range of a server.

### A. Setup

To conduct the experiment we use two Tmote Sky devices installed with Contiki OS. One device, the server, is placed on a table and powered with batteries. Another device, the client,

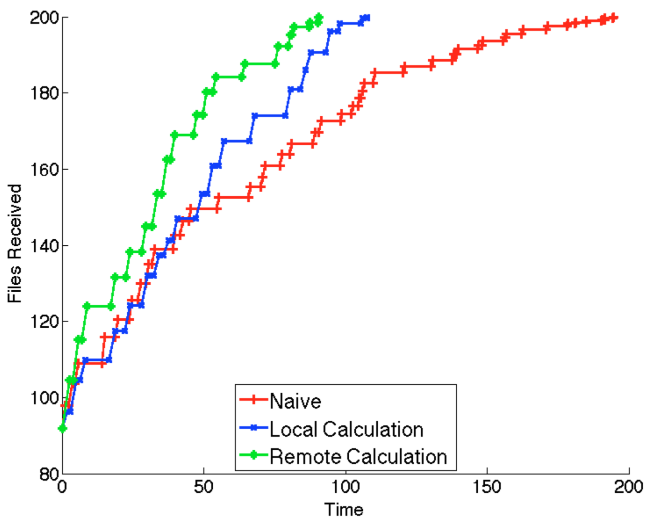


Fig. 11. Implementation evaluation results

is connected to a laptop so that we can monitor its activity during the experiment and collect our results.

Initially, the client possesses 46 percent of the total files, and the server possesses 96 percent. However, the server file vector is designed to have non-overlapping gaps with the client file vector in order to ensure that the client could eventually receive 100 percent of the files.

Once the experiment begins, we manually repeatedly move the client node in and out of the server's communication range in approximately two second intervals, which allows a single interaction to occur between the nodes. Once the client receives 100 percent of the files from the server, the experiment is concluded.

In all calculations we assume the encounter durations follow an exponential distribution with a mean of two seconds. Therefore, the rate parameter $\lambda$ for the exponential distribution and the MERLIN equation is set to 0.5.

### *B. Results*

Our results for testing the implementation of the MERLIN protocol are shown in Figure 11. The naive approach of requesting the single largest file range in every interaction has the poorest performance. In addition, we observe that the naive version becomes progressively worse as the experiment runs. This is because the size of the largest file range becomes smaller monotonically until almost every range contains a single file. Therefore, toward the end of the experiment, the client actually wastes communication by requesting a single file (one index) with a file range (two indices). Furthermore, in this experiment the server sends all of the client's requested files much before the interaction duration expired, which results in a large latter portion of the interaction remaining unused. In future work, implementing a multi-phase version of the MERLIN protocol could potentially make use of such unused time and improve the performance of the naive implementation.

While the performance results from the two MERLIN implementations are similar, the version where the calculations are performed remotely is observed to be superior. In the version where the calculations were performed by the client itself, the low processing speed of the device results in a loss of some usable time during the encounter, negatively affecting the overall performance. This suggests that future work into developing a less computationally demanding procedure for locally evaluating the MERLIN equation may be beneficial.

## IX. Related Works

Over the past decade and a half there has been considerable research on intermittently connected mobile networks [5]. An early work showed that carry-and-forward mechanisms in such mobile networks can significantly improve the capacity of wireless networks [6].

Many have explored the design of multi-hop encounter-based routing protocols for such networks [7], [8], [9] Several works have also explored various strategies for content/file dissemination for such networks including the use of network coding [15], coded storage [16], and making use of social interactions [17], [18]. All of these works implicitly assume a link layer that is able to transfer packets during the short, random encounters. However, the problem of designing an efficient link transfer mechanism has received relatively less attention. As such, our work is complementary to the above-mentioned works on content-dissemination and routing.

Compared to traditional approaches to wireless link layer protocols such as the point to point protocol, the reliable link protocol, and their enhancements [19], MERLIN is intended explicitly for random short-duration links with a known duration distribution, and for a structured client-server pull-based (download) application. As such it can be seen as an example of a cross-layer optimization [20].

One prior work that examines the design of a mechanism to maximize information transfer over a random encounter link is Encounter Transfer Protocol (ETP) [21]. ETP focuses on improving the link performance at the transport layer under a specific scenario whereby the mobile radio in a vehicle is separated from the processor running applications by a wired/wireless link. In contrast to our work, ETP does not consider a particular client-server communication scenario with structured content, and focuses primarily on rate adaptation for maximizing link utilization. Moreover, in contrast to our work, ETP does not assume the availability of a prior distribution on the link encounter duration.

Our MERLIN algorithm assumes that the encounter duration distribution is *a priori* known. This assumption is not entirely unreasonable, and has been made in other works before [10], [18]. Such distributions can be estimated for a given environment from real encounter time measurements (e.g., see [11]) or inferred from GPS traces (such as in our own trace-based evaluations), and can be enhanced (i.e., have their variance reduced, resulting in improved performance) by conditioning on specific parameters such as location, time of day, vehicle speed and direction.

## X. Conclusion and Future Work

We have introduced in this work a novel problem — how to design an efficient single-phase download request protocol for random short-duration communication links, which arise in the context of vehicular, D2D, and other intermittently connected mobile networks. As a solution, we have presented MERLIN, a protocol that is provably optimal in the case where clients make a request in the form of needed file-ranges that the server, whose file distribution is assumed to be independent and uniform, responds to in a single phase. We have shown how the optimal number of requests can be derived mathematically, and through simulations investigated how various parameters affect MERLIN performance.

We believe this work opens the door to a wide range of new and interesting investigations on the subject of optimizing data downloads over random duration encounters.

A direction for future work is to relax various assumptions made in the present model. For instance, if the availability of files is non-independent across files, then the ordering of request-ranges may be quite different, and also the client may "learn" from the response at each stage something about the file availability at the server; this could introduce a tradeoff between using the early requests to improve understanding of the server state and using them to maximize link efficiency (possibly leading the way to multi-armed bandit-type formulations which have a similar exploration-exploitation tradeoff).

We can imagine a scenario where multiple requests are generated by iterating over a sequence of optimally generated single-phase requests. While each stage of the iterative algorithm is optimized, it may be possible to improve the overall performance over multiple phases further by formulating the online problem as a stochastic dynamic program. We are currently working on this extension.

Another direction to be explored is to change the architecture of the download process from the purely "pull"-based scheme described to one that incorporates "push" from the server regarding its available files. Intuitively, the latter may be more efficient when the server availability is low. It may also be possible to develop hybrid schemes in which the choice of push/pull is determined at each phase depending on the availability percentages at the client/server respectively.

In dense environments where the client may have a number of possible servers to select from, there could also be enhancements to the protocol that consider the optimal selection of servers, possibly even dynamically switching between various servers over time.

Further, while we have currently assumed that the goal is simply to maximize the number of downloaded files, it may be beneficial in some settings to indicate different weights or priorities for the various files and try to maximize a suitable weighted objective function. For instance, this may be a way to optimize for some global objective function pertaining to dissemination of various files with different geographical demand distributions.

## References

[1] Hassnaa Moustafa, and Yan Zhang, *Vehicular networks: techniques, standards, and applications*, Auerbach publications, 2009.

[2] U. Lee, R. Cheung, and M. Gerla, "Emerging vehicular applications," *Vehicular Networks: From Theory to Practice*, Chapman and Hall/CRC, 2009.

[3] K. Doppler, M. Rinne, C. Wijting, C. Ribeiro, and K. Hugl, "Device-to-device communication as an underlay to LTE-advanced networks," IEEE Communications Magazine, 47(12), 42-49, 2009.

[4] A. Asadi, Q. Wang, V. Mancuso,"A Survey on Device-to-Device Communication in Cellular Networks," IEEE Communications Surveys and Tutorials, vol.16, no.4, pp.1801-1819, 2014

[5] Abbas Jamalipour, Yaozhou Ma, "Intermittently Connected Mobile Ad Hoc Networks: from Routing to Content Distribution", Springer, 2011.

[6] Matthias Grossglauser and David Tse, "Mobility increases the capacity of ad-hoc wireless networks." INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. Vol. 3. IEEE, 2001.

[7] Zhensheng, Zhang, "Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: overview and challenges." Communications Surveys and Tutorials, IEEE, (2006): 24-37.

[8] Thrasyvoulos Spyropoulos *et al.*, "Routing for disruption tolerant networks: taxonomy and design." Wireless networks 16.8 (2010): 2349-2370.

[9] Kevin C. Lee, Uichin Lee, and Mario Gerla. "Survey of routing protocols in vehicular ad hoc networks." Advances in vehicular ad-hoc networks: Developments and challenges (2010): 149-170.

[10] Gabriel Sandulescu and Simin Nadjm-Tehrani, "Opportunistic DTN routing with window-aware adaptive replication." Proceedings of the 4th Asian Conference on Internet Engineering. ACM, 2008.

[11] Jakob Eriksson, Hari Balakrishnan, and Samuel Madden, "Cabernet: vehicular content delivery using WiFi," 14th ACM international conference on Mobile computing and networking (MobiCom) 2008.

[12] Jing Zhao, and Guohong Cao, "VADD: Vehicle-assisted data delivery in vehicular ad hoc networks." Vehicular Technology, IEEE Transactions on 57.3 (2008): 1910-1922.

[13] Zhang, Yang, Jing Zhao, and Guohong Cao, "On scheduling vehicle-roadside data access." Proceedings of the fourth ACM international workshop on Vehicular ad hoc networks. ACM, 2007.

[14] Yang Zhang, Jing Zhao, and Guohong Cao, "Roadcast: a popularity aware content sharing scheme in vanets." ACM SIGMOBILE Mobile Computing and Communications Review 13.4 (2010): 1-14.

[15] Uichin Lee, J.S. Park, J. Yeh, G. Pau, M. Gerla, "Code torrent: content distribution using network coding in vanet." Proceedings of the 1st international workshop on Decentralized resource sharing in mobile computing and networking. ACM, 2006.

[16] M. Sathiamoorthy, A. Dimakis, B. Krishnamachari, and F. Bai, "Distributed storage codes reduce latency in vehicular networks," IEEE INFOCOM, 2012.

[17] W Gao, Q Li, B Zhao, G Cao, "Multicasting in delay tolerant networks: a social network perspective." Proceedings of the tenth ACM international symposium on Mobile ad hoc networking and computing. ACM, 2009.

[18] X. Zhuo, Q. Li, G. Cao, Y. Dai, B. Szymanski, T. La Porta, "Social-based cooperative caching in DTNs: a contact duration aware approach," IEEE 8th International Conference on Mobile Adhoc and Sensor Systems (MASS), 2011.

[19] R. Ludwig, B. Rathonyi, "Link Layer Enhancements for TCP/IP over GSM," IEEE INFOCOM 1999.

[20] Kota, S.L.; Hossain, E.; Fantacci, R.; Karmouch, A., "Cross-layer protocol engineering for wireless mobile networks: part 2," Communications Magazine, IEEE , vol.44, no.1, pp.83,84, Jan. 2006.

[21] B. Yu, and F. Bai, "ETP: Encounter Transfer Protocol for opportunistic vehicle communication." IEEE INFOCOM, 2011.

[22] A. Dunkels, B. Gronvall and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," Local Computer Networks, 2004. 29th Annual IEEE International Conference on, 2004, pp. 455-462.

[23] A. Bhargava *et al.*, "Optimizing Single-Phase Downloads over Random Duration Links in Mobile Networks," USC ANRG Technical Report, 2016. Online at http://anrg.usc.edu/www/publications/