Computer Communications 32 (2009) 1687–1698

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

DBS-IC: An adaptive Data Bundling System for Intermittent Connectivity

Khaled A. Harras [a,*], Lara Deek [b], Caitlin Holman [b], Kevin C. Almeroth [b]

[a] Computer Science Department, Carnegie Mellon University, Qatar 5032 Forbes Avenue, Pittsburgh, PA 15289, United States
[b] Department of Computer Science, University of California, Santa Barbara, Santa Barbara, CA 93106, United States

ARTICLE INFO

Article history:
Available online 9 March 2009

Keywords:
Delay Tolerant Networks
Mobile networks
Intermittent connectivity

ABSTRACT

As mobile and wireless technologies become more pervasive in our society, people begin to depend on network connectivity, regardless of their location. Their mobility, however, implies a dynamic topology where routes to a destination cannot always be guaranteed. The intermittent connectivity that results from this lack of end-to-end connection is a dominant problem that leads to user frustration. Existing research to provide the mobile user with a facade of constant connectivity generally presents mechanisms to handle disconnections *when* they occur. In contrast, the system we propose in this paper provides ways to handle disconnections *before* they occur. We present a *Data Bundling System for Intermittent Connections (DBS-IC)* comprised of a *Stationary Agent (SA)* and a *Mobile Agent (MA)*. The SA *pro-actively* gathers data the user has previously specified, and opportunistically sends this data to the MA. The SA groups the user-requested data into one or more *data bundles*, which are then incrementally delivered to the MA during short periods of connectivity. We fully implement DBS-IC and evaluate its performance via live tests under varying network conditions. Results show that our system decreases data retrieval time by a factor of two in the average case and by a factor of 20 in the best case.

1. Introduction

As the Internet spreads throughout the world, demand for constant connectivity, regardless of location, is rising. The response to this demand has been the development of mobile applications and devices that can be used by in-motion users. However, as users move between connection points, they experience bursts of network connectivity interspersed with either weak or non-existent signals. A recent study finds that mobile devices can move at speeds of 75 mph and still experience periods of connectivity with high throughput and low loss [8]. However, most, if not all, current applications are not designed to take advantage of these short network connectivity bursts. An insufficient amount of data is exchanged before a disconnection occurs, and often must be re-gathered the next time a connection is present. Alternative, continuous Internet connections that are available, through current cell networks, for example, are either relatively expensive or slow, but are improving at a rapid pace. The intermittent connectivity in such scenarios leads to large latencies, user frustration, and possibly even complete application failure.

This user frustration is more fully appreciated in an example scenario. A mobile user takes a bus or a train to work every morning. This user tries to connect to available access points along her commute. The user opens a web browser and connects to http://www.cnn.com to read the morning news. She reads the blurb for the main story and clicks the link to read the full text. By this time, however, the bus has moved past the AP's signal range and the user receives a "Page Not Found" error. Although the user's laptop was likely connected long enough to receive a significant amount of data, part of this time expired before the user realized a connection existed, and more time was wasted as she read the main story's blurb. A system that quickly reacts to the acquisition of a signal, and utilizes the full connection period, would greatly enhance the user's experience in this case. If the system knows that the user enjoys reading the morning news on her way to work, it can pro-actively gather this data in the background whenever a connection is available. The full text of the main news story will then be waiting for the user when she wants to read it.

The intermittent connectivity that is the focus of the scenario above has been previously studied in various ways. Specifically, Delay Tolerant Networks (DTNs) and intermittent, or disconnected networks are research areas that address cases where an end-to-end connection does not exist [1,5,12,6]. With the generality of DTNs, and their focus on routing, many researchers have developed solutions that hide the interruptions of intermittent connectivity. The majority of these proposed solutions focus on reacting intelligently to disconnections after a request has been made [2,15,19]. Possible reactions include either caching requests [4,14,18] or maintaining high-level connections [17,20]. In all of these solutions, requests made during times of disconnection wait to be

* Corresponding author. Tel.: +1 805 403 0469.
E-mail addresses: kharras@cs.cmu.edu (K.A. Harras), cholman@cs.ucsb.edu (C. Holman), ldeek@cs.ucsb.edu (L. Deek), almeroth@cs.ucsb.edu (K.C. Almeroth).

0140-3664/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2009.02.014

serviced until connectivity returns. In addition, the mobile devices in these solutions are required to open separate connections to each application server the user wishes to contact. The system we present in this paper avoids both of these drawbacks.

We present and develop DBS-IC, a Data Bundling System for Intermittent Connectivity that takes advantage of short connection periods to enhance the experience of mobile users. A Stationary Agent (SA), located on a stationary device with a stable connection, collects data the user has specified will be needed in the future. This data can be heterogeneous: data from web servers, email servers, and other file servers. The SA then groups this data together into a single package, or *bundle*. Afterwards, the SA opportunistically sends this bundle to a Mobile Agent (MA), residing on a mobile device, whenever a connection is present. Once the bundle is successfully transferred to the MA, the user can view the data at any time, including times of disconnection. In this way, our system also hides the underlying instability of the connection, but unlike other systems, creates greater opportunities for users to be active during periods of disconnectivity.

DBS-IC efficiently utilizes available bandwidth using a combination of multiple techniques. First, our system forms a single connection between the MA and SA to send heterogeneous data, thereby eliminating application-specific connection and request times. In the scenario above, the user may want to check her email after reading the morning news. DBS-IC, therefore, sends the user's emails in the same bundle as the web data from CNN, relieving the user of the need to contact these servers separately. Second, after the first copy of a data bundle have been sent to the MA, DBS-IC bundles and sends only data *updates* in an effort to eliminate unnecessary re-transmissions. Referring to our scenario, after the email and web data has been sent to the user, only new emails and updated web content will be sent in the future.

With respect to bundling, we discuss various bundling schemes and address the transfer latency and data staleness that can arise from overly large bundles. For example, if the user's laptop is experiencing extremely short connections, the data may be out-of-date when the transfer finally completes. To counteract these problems, we introduce *mini-bundles* to expedite the transfer of data that is immediately viewable by the user and to keep data current. Instead of sending the user's email and the large amount of CNN web data all at once, we can send a chunk at a time so that the data is incrementally available. We examine different approaches for creating these mini-bundles, including data type, size, and priority.

We fully implement DBS-IC in order to evaluate its performance. We choose to implement DBS-IC, rather than simulate the system, in an attempt to obtain more realistic results under unpredictable wireless network conditions. We evaluate the performance of our implemented system in different intermittent connectivity scenarios, and compare the results to existing data retrieval methods. Results of live tests are excellent, showing that DBS-IC efficiently utilizes bandwidth to opportunistically deliver data to the user before disconnections occur. We find that mini-bundles further enhance our system, delivering viewable data to the mobile user significantly faster than traditional retrieval protocols such as the Hyper Text Transfer Protocol (HTTP).

The work presented in this paper is a significant extension of our previous work on DBS-IC [11]. This extension leads to a greater understanding of how the system actually works, where patches and modifications can be introduced to provide a more scalable system. There are three major additions in this paper. First, we refine the system architecture and operation to be more robust, and describe related algorithms in greater detail. Second, we present a more extensive set of evaluations to better assess the performance of our system. Third, we introduce and evaluate the idea of a simple adaptive predictor that dynamically determines mini-bundle sizes that should be delivered to the mobile agent. We show in our evaluation how this predictor helps the system increase the amount of viewable data at the end user, therefore, improving a user's experience.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 describes the components and operation of DBS-IC. Section 4 outlines our evaluation techniques and presents our results. Section 5 contains our concluding remarks.

2. Related work

In this section, we discuss related work, both in the general field of wireless networks and in more specific efforts to combat intermittent connectivity. We begin with a brief discussion of MANETs and progress into the area of disconnected and delay tolerant networks. We then examine two existing approaches to handle intermittent connectivity and we discuss how these approaches differ from our system.

A large focus of mobile research is on Mobile Ad hoc NETworks (MANETs), which consist of mobile wireless nodes, each acting as an end-point and a router. A main thrust in MANETs has been on routing [13,22–24]. However, since MANETs assume an end-to-end connection between the nodes in a network, this assumption immediately distinguishes the work in this area from our work on intermittent connectivity resulting from disconnected networks.

Other areas of research, such as disconnected mobile networks [16,26] and Delay Tolerant Networks (DTNs) [1,5], address networks where end-to-end connectivity cannot be assumed to exist. While our system focuses on delivering bundled data from a wired stationary device to a mobile node, the same bundling and opportunistic delivery concepts are applied in DTNs. However, existing research in DTNs mostly focuses on forwarding techniques [10,9], routing algorithms [12], and transport layer issues [7], which our system is not concerned with. In such challenged networks, where intermittent connectivity and variable delays are a dominant factor, a system like ours helps take advantage of every successfully routed message.

In the more specific area of intermittent connectivity, two main methods exist to counteract the detrimental effects of disconnections. The first approach involves maintaining session-level connections through disconnections [2,3]. Ott and Kutscher first examine the feasibility of mobile network traffic for in-motion users [19]. They introduce their Drive-thru Internet Architecture and examine a Connectivity-Loss Resilient Connection (CLRC) between a mobile client and a fixed proxy that maintains information regarding multiple TCP streams [20]. By maintaining the CLRC and splitting the connection at a proxy, transport-level connections remain open through disconnections. Mao et al. present a similar approach, maintaining session-level connections through disconnections [17]. The goal here is to allow the user to seamlessly resume applications upon return of connectivity. Comparably, Kulkarni et al. discuss methods to keep an unreliably connected mobile client synchronized with rapidly changing web page content [15]. Their solution uses a proxy that sits between the client and server, and caches requests during times of disconnection.

The second approach to intermittent connectivity does not try to maintain high-level connections, but simply delays delivery of data while the mobile device is disconnected. This approach includes solutions such as middleware to 'store-and-forward' client requests during times of disconnection or weak signal [14] and to synchronize data once connectivity returns [18]. Similarly, Chang et al. present an ARTour Express program which stores requests internally so the user can seek multiple pages without waiting for each to completely load [4]. In the Message Ferrying scheme, special nodes called ferries are used to forward messages between disconnected nodes [28,27].

These two approaches to intermittent connectivity differ from our system in the way they transfer data to the mobile device. The proxy servers in these implementations do not anticipate what data the user will need in the future, and react to disconnections as they occur. In contrast, we expand upon existing methods of pre-fetching in wired networks [21] to mask retrieval time of network data by *pro-actively* collecting and sending the data to the mobile device before it is needed. Furthermore, in these solutions, the mobile client must create a new transport-level connection each time the user requests a new piece of data. Our system avoids these multiple connections by bundling the requested data and sending this bundle over one connection set up between the Stationary Agent (SA) and the Mobile Agent (MA). Most importantly, neither of the above methods allow the user to view new data *during* times of disconnections, leading to an uneven user experience.

3. System architecture

In this section, we present an overview of our Data Bundling System for Intermittent Connectivity (DBS-IC) and its associated protocol, the Mobile Pro-active Transport Protocol (MPTP). We first explain the major components and overall operation of our system. We continue with a discussion of several critical issues such as authentication, bundling, and handling data updates. Next, we introduce mini-bundles, and show different methods by which they can be constructed. We also discuss the predictor component in our system, which dynamically determines an appropriate mini-bundle size that should be delivered. Finally, we conclude this section with a discussion of additional design considerations.

3.1. System components and operation

There are two major components that comprise our Data Bundling System for Intermittent Connections (DBS-IC). These components are a Stationary Agent (SA) and a Mobile Agent (MA). The SA is located on a stationary device that has a stable connection to the Internet, such as a user's desktop computer. The SA gathers various forms of data from different sources, such as web, email, and file servers. The MA is located on an in-motion mobile device which moves between wireless Access Points (APs) or mesh routers and therefore experiences intermittent connectivity. A Mobile Pro-active Transfer Protocol (MPTP) connection is created between the SA and MA whenever the MA enters connectivity range. We created MPTP to help provide authentication, lower connection overhead, and provide more bandwidth for data transfers. In general, MPTP borrows from the spirit of the bundling protocol specification in the sense that data from different applications are bundled together at the application layer [25]. However, we do not perform any protocol translation since we are operating over the same internet. Details of MPTP is more of an implementation issue and is not included due to space limitation, but can be provided upon request.

The DBS-IC architecture and a basic operation scenario are presented in Fig. 1. Operation begins with an initial configuration step in which the user interacts with the SA, specifying web, email, and file servers from which the agent should gather data (Step 1). The SA then contacts the specified application servers and gathers the user-requested data (Step 2). This step is periodically repeated, based on a user-configurable update frequency, to keep the data current. After gathering all the requested data, the SA then bundles these pages, emails, and files into a single group. The SA is then ready to send this bundle to the MA as soon as the SA is contacted (Step 3). An alternative to bundling all the data into one large bundle, called *mini-bundling*, can optionally be performed at this point. With mini-bundling, the gathered data is structured into multiple smaller bundles, which can each be autonomously sent to the MA (Step 4). Furthermore, a subcomponent we call the *predictor* will determine the best mini-bundle size to use in order to maximize the amount of viewable data at the MA. The methodology and benefits of mini-bundling and the predictor are discussed later.

The SA is now ready to transmit data to the MA. The MA contacts the SA when it comes within range of an access point. In response to this signal, the SA begins to send the data it has previously gathered and bundled (Step 5). However, in this example, the MA moves out of range and loses its connection in the middle of the transmission (Step 6). We employ heartbeat messages, sent from the MA to the SA at regular intervals, to help the SA recognize disconnections. When the MA loses connectivity, its heartbeat messages no longer reach the MA. As a result, the SA learns that a disconnection has occurred. The SA stops the transmission and waits until it is again contacted. When the MA moves back into connectivity range, it again contacts the SA. The SA then resumes sending the data from the last byte the MA received (Step 7). Once the whole bundle is sent, the MA has the opportunity to send configuration file updates. This cycle repeats indefinitely: as long as the SA is running, updates will be gathered and bundled, waiting to be sent to the MA.

Since we actually implemented the DBS-IC, the pseudo-code of our system is shown in Figs. 2–5. We present these figures to give a

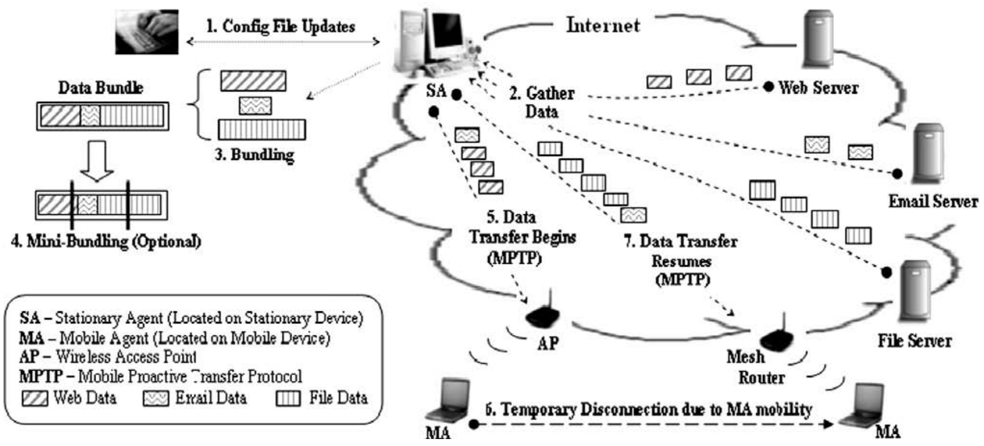
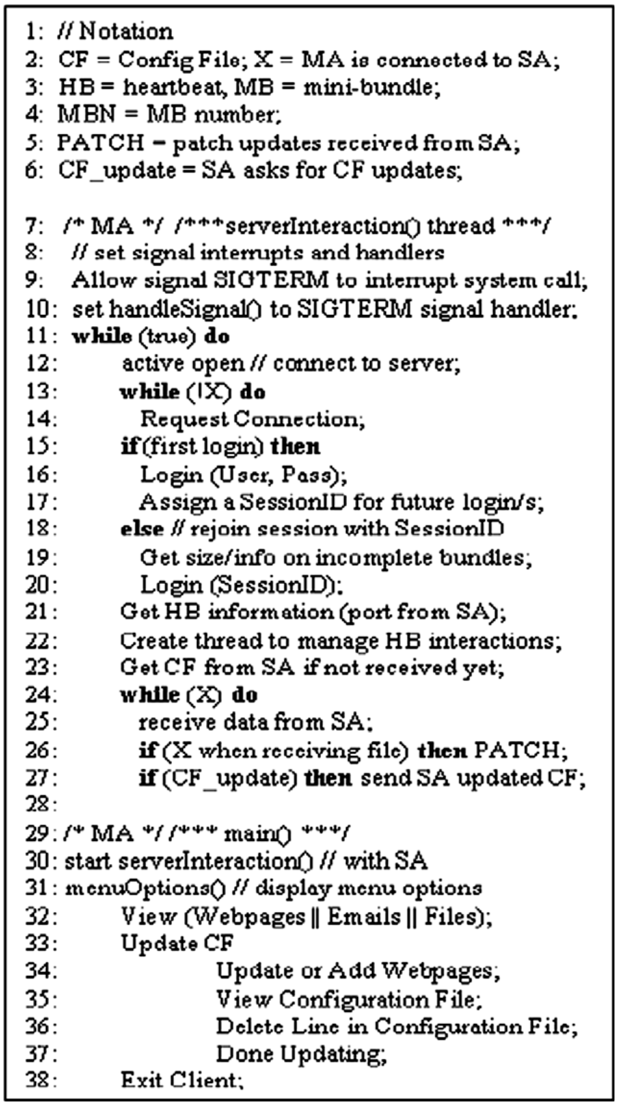

1. Config File Updates
Internet
Data Bundle
SA
2. Gather Data
Web Server
3. Bundling
Email Server
4. Mini-Bundling (Optional)
5. Data Transfer Begins (MPTP)
7. Data Transfer Resumes (MPTP)
File Server
AP
Mesh Router
SA – Stationary Agent (Located on Stationary Device)
MA – Mobile Agent (Located on Mobile Device)
AP – Wireless Access Point
MPTP – Mobile Proactive Transfer Protocol
▨ Web Data ▨ Email Data ▥ File Data
6. Temporary Disconnection due to MA mobility
MA
MA

Fig. 1. DBS-IC architecture & operation scenario.

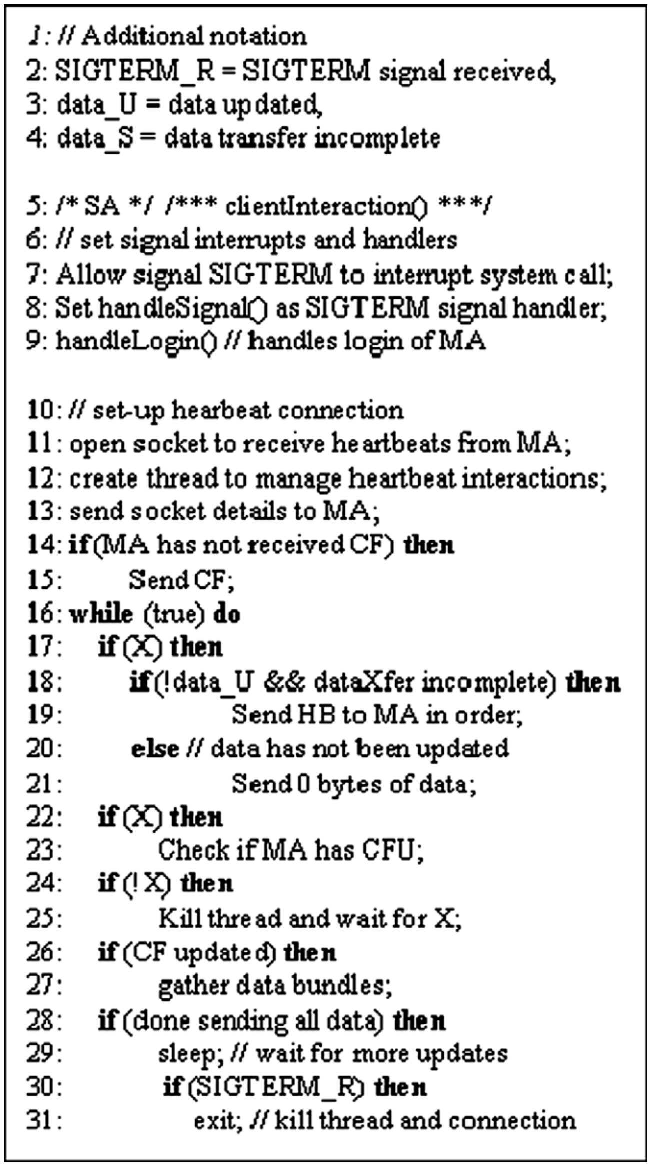


```
// Notation
CF = Config File; X = MA is connected to SA;
HB = heartbeat, MB = mini-bundle;
MBN = MB number;
PATCH = patch updates received from SA;
CF_update = SA asks for CF updates;

/* MA */ /***serverInteraction() thread ***/
  // set signal interrupts and handlers
  Allow signal SIGTERM to interrupt system call;
  set handleSignal() to SIGTERM signal handler;
while (true) do
    active open // connect to server;
    while (!X) do
      Request Connection;
    if(first login) then
      Login (User, Pass);
      Assign a SessionID for future login/s;
    else // rejoin session with SessionID
      Get size/info on incomplete bundles;
      Login (SessionID);
    Get HB information (port from SA);
    Create thread to manage HB interactions;
    Get CF from SA if not received yet;
    while (X) do
      receive data from SA;
      if (X when receiving file) then PATCH;
      if (CF_update) then send SA updated CF;

/* MA */ /*** main() ***/
start serverInteraction() // with SA
menuOptions() // display menu options
    View (Webpages || Emails || Files);
    Update CF
            Update or Add Webpages;
            View Configuration File;
            Delete Line in Configuration File;
            Done Updating;
    Exit Client;
```

Fig. 2. Pseudo-code for server interaction at the Mobile Agent (MA).



```
// Additional notation
SIGTERM_R = SIGTERM signal received,
data_U = data updated,
data_S = data transfer incomplete

/* SA */ /*** clientInteraction() ***/
// set signal interrupts and handlers
Allow signal SIGTERM to interrupt system call;
Set handleSignal() as SIGTERM signal handler;
handleLogin() // handles login of MA

// set-up hearbeat connection
open socket to receive heartbeats from MA;
create thread to manage heartbeat interactions;
send socket details to MA;
if(MA has not received CF) then
      Send CF;
while (true) do
   if (X) then
      if(!data_U && dataXfer incomplete) then
               Send HB to MA in order;
      else // data has not been updated
               Send 0 bytes of data;
   if (X) then
         Check if MA has CFU;
   if (! X) then
         Kill thread and wait for X;
   if (CF updated) then
         gather data bundles;
   if (done sending all data) then
         sleep; // wait for more updates
          if (SIGTERM_R) then
             exit; // kill thread and connection
```

Fig. 3. Pseudo-code for client interaction in the server agent (SA).

clearer picture regarding how all components fit together, and also to provide a more formal description of how DBS-IC operates. Fig. 2 describes the MA's ability to access and set the configuration file, and shows how its interactions with the server occurs. Since most of the complexity of DBS-IC is located at the Stationary Agent (SA), the other three figures represent major components in the SA. Fig. 3 shows the various parts built for basic interaction with the client MA. Fig. 4 describes handling client login, and how to either start or resume a session with any MA. Finally, Fig. 5 shows the operation of the predictor which dynamically determines the appropriate mini-bundle size that should be sent. We now talk in more details about each one of these components.

3.2. Authentication, bundling, and updating

We now discuss in more detail the features of DBS-IC that allow the system to efficiently utilize short connection periods to transmit data. Recent study findings show that even up to speeds of 75 mph, an intermittently connected mobile device experiences connection periods with high throughput and low loss [8]. The two main factors prohibiting meaningful data transfer at these speeds are (1) lengthy connection and authentication times and (2) multiple application-required request cycles (such as multiple HTTP GET requests). Our system solves both of these problems by providing a low-overhead authentication scheme, and by bundling data to avoid multiple connections from the mobile device.

In the basic DBS-IC operation scenario, presented above, there are some important features that solve the two problems discussed in the preceding study. First, authentication via a username and password is not needed the second time the MA contacts the SA. Instead, the MA will use a unique session identifier that the SA assigns to it. This simpler authentication technique lowers overhead and provides the system a larger percentage of connection time devoted for actual data transfer. Another important feature of our system is how it delivers data to the MA. Once the MA receives a complete copy of an initial bundle, the SA will send only updates to this data in the future. This technique will save bandwidth and unnecessary re-transmission of data the MA has already received.

While the SA continues to gather data updates even when the MA is disconnected, the SA does not apply these updates to any of the data that has been partially sent to the MA. The SA instead sends the MA a complete copy of the data before sending updates, even if this means sending stale data. Two other possible methods

```
// Additional Notation
configSent = configuration file sent to MA from SA;

/* SA */ /*** handleLogin() ***/
read header sent from MA;
if (header→ Login(User,Pass) then
    send initial default Mini BN;
    Confirm username and password;
if (header→ Login(SessionID) then
    send MB number; // could be changed by predictor
    if(MBN changed by predictor in SA) then
        tell client to receive new CF;
        read header sent by MA; // from Login(User,Pass)
    else
        tell client to resume normal activity;
        read header sent by MA; // from Login(SessionID)
    if(header→ size of information in header != 0) then
            read how many bytes received by MA;
            if(confirm session ID) then
                    if(header→ CF receive by MA) then
                            configSent = 1;
                    else
                            configSent = 0;
```

Fig. 4. Pseudo-code handling client logins at the SA.

```
// Additional Notation
new MBN = new Mini-Bundle Number;
maxMB = Maximum number of MB;
MB_s = Mini-Bundle size; B_s = total bundle size;

/* SA */ /*** Predictor- handleBN()***/
B_s = MB_s * MBN;
// calculate next appropriate MB size
for (count = 1→ maxMB) do
        MB_s = B_s / count;
        temp = MB_s % READ;
        if(temp < best approximation) then
                best approximation = temp;
                newMBN = count;
if(newMBN != MBN) then
        set parameter to inform client of change;
        MBN = newMBN;
        ConfigSent = 0;
        gather data after change in MBN;
```

Fig. 5. Pseudo-code for the dynamic predictor at the SA.

for handling updates are possible. First, if the MA is not currently connected, and an update is gathered, the SA could start sending from the beginning of this updated data when the MA re-establishes a connection. Second, the SA could update only those pages, emails, or files that have not yet been sent to the MA. The problem with the first alternative is that there is a risk the user will never receive a complete copy of the data, instead receiving only the beginning of multiple versions of data. This partial data cannot be unbundled and viewed by the user until all the data has been received. The second alternative is difficult because the SA does not know exactly how much information the MA received before being disconnected. In the current implementation, the SA only ascertains this information once the MA reconnects and sends a byte count to the SA. Hence, the SA cannot know exactly what data can safely be updated.

In addition to the stale data problem discussed above, large bundles can result in overly long data availability latencies. More specifically, our system can enter a situation in which there is a significant amount of data on the MA that cannot be viewed by the user. Consider the following scenario. The MA connects to the SA and receives half, or even 90%, of a large bundle of data before losing connectivity. The MA cannot re-connect to the SA for another hour. Because it has not received the entire bundle, the user cannot view any of the data already received. The entire bundle is needed before it can be unbundled, and, in the case of an update, patched with the data the MA already has. To solve this problem, we introduce mini-bundles.

3.3. Mini-bundles

Mini-bundles are pieces of the complete data bundle. By dividing the data bundle and transmitting smaller mini-bundles, there is a higher probability that the MA will receive the entire mini-bundle before experiencing a disconnection. And receiving the entire mini-bundle, as discussed earlier, is a prerequisite for the MA to view any data. However, due to the communication overhead that takes place after data is transferred, there is a chance mini-bundles could hurt system performance. A balance between the number of mini-bundles and the size of each bundle is needed. Mini-bundles can be created based on various criteria, such as the following:

(1) *Application Type:* Data of one type is bundled separately from data of another type. For example, one mini-bundle is composed entirely of email data, another contains only web data, and a third consists of file data.
(2) *Priority:* Mini-bundles are created based on user-specified priority values. One mini-bundle consists of the data of the highest priority, another of lesser priority, etc.
(3) *Size:* Regardless of the content of the bundle, the bundled data is divided into equally sized mini-bundles. This means that each mini-bundle could contain similar or different types of data.

When used alone, each of these techniques has benefits and drawbacks. We employ a merged technique in which mini-bundles are formed primarily based on size, secondarily on priority, and finally according to data type. This merged technique delivers the data the user wants most, while regulating the mini-bundles so they are approximately the same size. The size equality of mini-bundles ensures that one greedy mini-bundle does not monopolize the connection time, thereby defeating the purpose of mini-bundles. Regardless of the method that is used to create them, each mini-bundle is self-contained. More specifically, a piece of data that the user wants (for example, a web page or a set of emails from a single email server) will never be divided between mini-bundles. Therefore, after the transfer of a mini-bundle is accomplished, the user has a complete, viewable subset of the bundled data.

3.4. The predictor

The predictor component in the DBS-IC takes further advantage of each period of connection by predicting the expected intervals of connectivity and, accordingly, determining a more efficient mini-bundle number and size. As shown in the systems operation section, the predictor runs on the SA. When an MA first establishes a connection with the SA, the SA will randomly send a mini-bundle with a default size. Simultaneously, it will keep track of the length of the connection period with the MA before a disconnection occurred. The next time the MA is connected, the predictor will assume that the MA will remain connected for the time maintained by the earlier connection period and will send a mini-bundle size accordingly. The assumption is that if the MA is on a device that is in a car for example, the car will most likely experience similar connection periods since its average speed will likely be the same. The more the MA communicates with the SA, the more the predictor learns and is capable of picking more accurate mini-bundles.

Even though we described the operation of the predictor component in Fig. 5, we use Fig. 6 to fully appreciate its impact. Determining the size of mini-bundles is important for obtaining more efficient transmissions to have more viewable data during times of disconnection. Data is only viewable when an entire mini-bundle is received. As shown in the figure, if we were to divide the data bundle into two mini-bundles, and assume that the user reached an empty park after the second AP, only half the requested data can be viewed. If the mini-bundles were divided into three parts, the user would have two thirds of the data available while sitting at the park with no connection. Conversely, if we were to have too many mini-bundles, we will reduce the transfer efficiency of data due to the communication overhead of many mini-bundles. The point, however, is that choosing an efficient mini-bundle size is crucial in maintaining a balance between the pros and cons of mini-bundles. This observation is better shown in Section 4.2.

We quickly note that another approach we examined for the predictor was to gather connection availability history and store it in the MA. The predictor would then receive this information from the MA, and use historical statistics to give a prediction for future connectivity durations. The problem with this approach, however, is that it uses up precious bandwidth which cuts down the time that could be better used for data transfer. Also, while history based prediction may result in more accurate mini-bundle sizes, the gain compared to our simple approach is negligible.

3.5. Other design considerations

In our current implementation of DBS-IC, all MPTP connections are built on top of TCP. Because TCP provides reliability and in-order delivery, MPTP itself is not designed to provide these features. We justify using TCP with two reasons. First, reliability and in-order delivery are imperative for our system; every packet of a transmission must be received. We briefly attempted to develop a system using UDP connections, but without additional services built into MPTP, UDP fails. As soon as one packet is lost, the system breaks. To fix this fragility, we need to add more data acknowledgments and timeouts, both of which would slow the system.

Second, we justify our decision to use TCP based on the results of Gass et al.'s previous study, which finds that TCP bulk data transfer to an in-motion mobile device actually achieves much higher throughput than UDP [8]. Based on our experience with UDP, and the estimated overhead of making UDP reliable, all of the MPTP connections between the MA and SA are built on top of TCP. Possible future work in this area would be to examine the benefits of modifying TCP window size on the throughput of our system. However, we show in the evaluation section that our system still achieves high throughput without modifying TCP, even when only short connection periods can be guaranteed.

Another design consideration of DBS-IC is the initial configuration step in which the user specifies the data they want delivered to their mobile device. Instead of requiring the user to complete this step, the SA could incorporate a smart gathering agent which retrieves data based on recent user browsing trends. We chose not to focus on the smart gathering agent at this stage. Instead, we focus on bundling and sending data pro-actively to examine how user experience is affected. Adding a smart agent mainly reduces the burden on the user by automating the decision for which data needs to be gathered. This improvement is left for future work.

4. System evaluation

In this section, we present the evaluation of DBS-IC. We use our evaluation to accomplish two main goals. The first goal is to compare the performance of DBS-IC in situations of intermittent connectivity to the performance of traditional retrieval methods. Our second goal is to examine the impact of various parameters on the performance of DBS-IC. We discuss the evaluation setup and

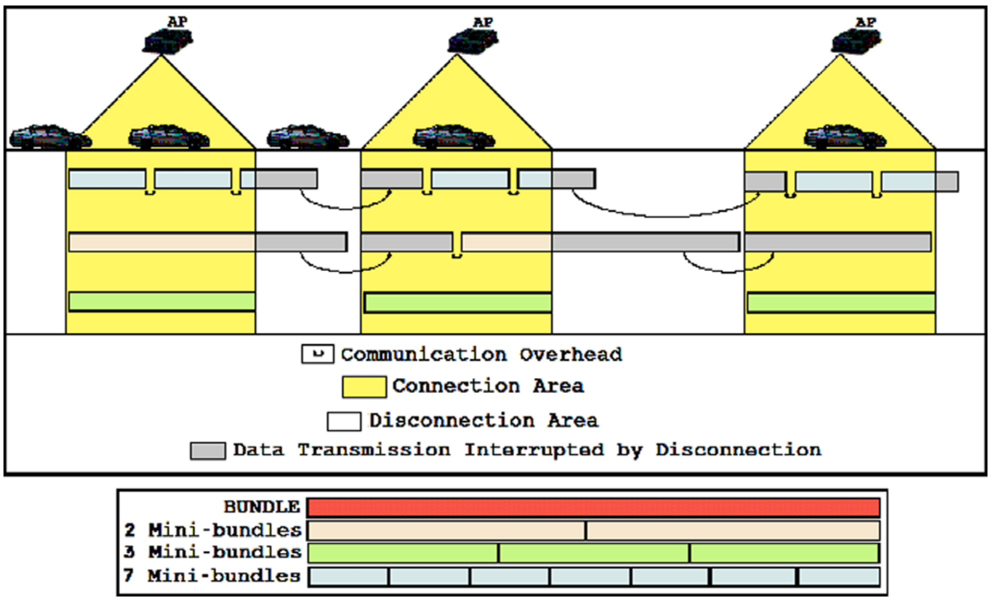


Fig. 6. The transfer of the same bundle using different mini-bundle numbers and sizes. In this scenario, dividing the bundle into three mini-bundles is the most efficient choice.

environment, followed by a set of results that fulfill our designated goals.

4.1. Evaluation setup and environment

We fully implement DBS-IC and test our implementation in various ways. We chose to implement our system, as opposed to simulating it, to obtain more realistic results. We perform our tests in the lab, using the results gathered in previous driving test studies [8,19] to accurately and realistically choose most of our testing parameters (Table 1). The SA is located on a lab machine that has a stable connection to the Internet, with an unloaded 100 Mbps full-duplex switched Ethernet connection. The MA is located on an intermittently connected laptop with an 802.11b network card. We run our tests over the same time of day or night to ensure relatively similar network load. Every point in our graphs is the average of 10 different runs.

The SA gathers and bundles three types of data in our tests: web pages, emails, and files. Throughout our evaluation, we examine different *intermittent connectivity models* to see how our system performs under varying mobile situations. Each model is characterized by a unique combination of connection and disconnection durations, as shown in Table 1. Gass et al. [8] show that a mobile device traveling at 5 mph experiences approximately 100 s of efficient connection time while passing a single access point, discounting the lossy entry and exit phases discussed by Ott and Kutscher [19]. Similarly, a mobile device experiences about 40 s of connectivity at 35 mph and 15 s at 75 mph. Using these numbers as a guide, we recognize the following four intermittent connectivity models:

1. *Downtown Walking Model:* This model is characterized by connections of 120 s and disconnections of 20 s. It simulates the experience of a mobile user walking past access points or mesh routers in an urban area. We do not focus heavily on this model since the lengthy connection periods are generally adequate to gather data using traditional retrieval methods like HTTP and FTP.
2. *Downtown Driving Model:* This model is characterized by connections of 40 s and disconnections of 15 s. It simulates the experience of a mobile user driving in slow traffic.
3. *Suburb Driving Model:* This model is characterized by connections of 20 s and disconnections of 10 s. It simulates the experience of a mobile user driving on surface streets without traffic.
4. *Highway Driving Model:* This model is characterized by connections of 15 s and disconnections of 30 s. It simulates the experience of a mobile user driving at 70 mph along a highway that passes periodic APs.

In addition to the intermittent connectivity model, we also vary characteristics about the data our system is transmitting throughout our tests. We define *bundle size* as the size of the compressed data that is physically transferred between the SA and the MA. This bundle is divided into a specific number of *mini-bundles*, which we vary from one to 10 to evaluate the improvement mini-bundles provide in delivery time vs. the trade-off of additional overhead. These mini-bundles are created based on the *mini-bundling technique*, which can be size, priority, data type, or the merged technique discussed earlier. The time it takes to make data available at the MA also depends on the percentage of *prefetched data*, the amount of data that was gathered by the SA before the MA connects to the SA. In the optimal case, the SA will have gathered 100% of the data before the MA connects, but we find that our system still performs well when this is not the case. Finally, we test our system with different *round trip times* between the MA and SA to see how this impacts DBS-IC.

Table 1
Evaluation parameters.

Parameter	Value range	Nominal value
Bundle size (MB)	1–50	20, 30
Connection duration (s)	15–100	20
Disconnection duration (s)	10–30	10
Number of mini-bundles	1 mini-bundle to 10 mini-bundles	3 mini-bundles
Mini-bundling technique	Size, priority, data type, Merged	Merged
Prefetched data (%)	0–100	100
Intermittent connectivity model	Walking, downtown, suburb, highway	Suburb, downtown
Round trip time (ms)	2–150	70

The metrics we use in our evaluation are the following.

User-Perceived Data Delivery Time: This is basically the delivery time for a given bundle of data. We add the term user-perceived to focus on the end user's impression when using the system. In other words, it is the time between when the user decides to view a piece of data and when that piece of data is viewable on the MA.

Data Throughput: The amount of data our system transfers between the SA and MA during connection periods. Our system is built to maximize this throughput by lowering connection overhead.

Data Staleness: The time difference between the latest version of a piece of data on the MA and the latest version on the SA. Our system hopes to deliver data updates in a way that will minimize data staleness.

4.2. Results

We divide our results section into several parts. Initially, we analyze how quickly data becomes available to the mobile user when using our system as compared to using existing retrieval protocols. We then take a closer look at the throughput our system achieves and follow this by testing the performance enhancements that mini-bundles provide to our system. We then evaluate how DBS-IC performs using different intermittent connectivity models. We also discuss the throughput of our system and show the impact of the round trip time between the SA and MA. Finally, we show the impact of using the predictor on the performance of DBS-IC.

4.2.1. Data gathering vs. data transfer

As stated earlier, the goal of DBS-IC is to opportunistically present the mobile user with data, so disconnections will have a less adverse effect on viewing data. Currently, protocols such as HTTP, FTP, and SMTP are used to gather web, file, and email data, respectively. These protocols were not designed with intermittent connectivity in mind; they have long connect, request, and timeout cycles. And in the case of HTTP, a new TCP connection must be created for each page requested from a new site. By gathering this data on a machine with a stable connection to the Internet, our system reduces the data transfer on the mobile device to an opportunistic bulk transfer of bundled data. Only one TCP connection is created when the mobile device gains connectivity, avoiding the connection overhead of each individual piece of data.

Our first test compares the time to gather data on the MA using traditional retrieval methods to the time to both gather the data on the SA and transfer it to the MA using our system. In both cases, the MA experiences intermittent connectivity based on the suburb driving model. We vary the total data size in this test, which means the compressed bundled data is smaller and varies depending on the type of data. In an attempt to keep the compressed data consistent, each bundle consists of 2/3 web data, 1/6 email data, and 1/6 file data.

Fig. 7 shows that in all cases, the user-perceived data delivery time is reduced when our system is used. On average, the time is reduced by a factor of two. The large gathering times experienced when the MA uses traditional retrieval methods are caused by two main factors. First, the MA is experiencing disconnections every 20 s, which slows web data retrieval significantly. When retrieving web pages on the MA, we set a 10 s timeout value. With this timeout value, a retrieval or connection is automatically re-tried if a response is not heard within 10 s. This timeout simulates a user hitting the refresh button after the page has been trying to load for 10 s. Since the MA is experiencing 10 s disconnections, this 10 s timeout is a lower bound on usefulness. A lower timeout value will have no beneficial effect on performance since the MA will still be disconnected when a connection is re-tried.

The second reason for the longer retrieval time on the MA is due to the fact that the MA is located on a wireless device with an 802.11b network card. It therefore has inherently less bandwidth and throughput than a wired device. In this test, we add the data retrieval time on the SA to the transfer time from the SA to the MA. This combination means that in the worst case, when the SA has prefetched 0% of the data, our system still performs relatively well. In many cases, however, the SA will have pro-actively gathered this data, before the MA ever connects to the SA. In this situation, the user-perceived data delivery time consists only of the transfer time between the SA and the MA, reducing data availability times, on average, by a factor of 12 as compared to traditional methods.

4.2.2. Mini-bundles

The previous test helps show that our system delivers data the user wants faster than if the mobile user gathers it themselves using HTTP or similar existing retrieval protocols. However, if the user is gathering data, such as web pages, each page will be viewable as soon as it loads. While the user will not have the complete data for a while, they will have part of the data to keep them occupied. We now empirically examine the speedup in user-perceived data delivery time that our system obtains by utilizing mini-bundles.

We first test mini-bundles by holding connection duration and bundle size constant, varying the mini-bundle count from 1 to 10. We again follow the suburb driving model. We note, however, that tests with the downtown driving model yielded very similar results. In this test, all mini-bundles are of equal size. More specifically, the SA gathers and bundles 20 MB of data, then sends this 20 MB of data to the MA in a varying number of mini-bundles. The SA first sends the data in one 20 MB mini-bundle, then two 10 MB mini-bundles, and so on.

Fig. 8 shows that as the number of mini-bundles increases, the total time to receive the full 20 MB of data increases. However, the advantage of mini-bundles in this case is that the user-perceived data delivery time of *some* data decreases. Compared to sending the data in one large bundle, the user can view half the data in half the time when using two mini-bundles. A less desirable effect of mini-bundles is the added overhead in terms of the total delivery time. There is extra communication costs with each additional mini-bundle, since the SA must prepare the MA for each mini-bundle it sends, and the MA must acknowledge every mini-bundle it receives. The MA also has the opportunity to send configuration file updates between each mini-bundle, in an attempt to keep the data as up to date as possible. However, although the total data delivery time does increase, mini-bundles present the mobile user with viewable data faster than sending the data in one bundle. And since mini-bundles can be arranged by priority, they are an efficient way to opportunistically deliver to the user their most important data more quickly.

4.2.3. Intermittent connectivity model

We next evaluate the performance of mini-bundles when our system experiences different connection durations. To compare mini-bundle overhead, we only look at total data transfer times in this test; we do not take into account the partial data delivery improvements provided by mini-bundles. We also hold disconnection time constant at 10 s in order to isolate the effect that different connection durations have on our system.

In the previous test, we saw that using additional mini-bundles resulted in a longer overall data transfer time. We observed that this time increase was due to the fact that the extra processing costs associated with more mini-bundles consumed a portion of the short connection periods available. In Fig. 9, we see that the same pattern holds, regardless of the connection duration. Each additional mini-bundle adds some processing overhead, increasing the user-perceived data delivery time for the whole data in all situations. Therefore, mini-bundles should only be used when some part of the data is more crucial to the mobile user than other parts. If the user needs to receive the entire data together, using only one bundle remains an efficient choice.

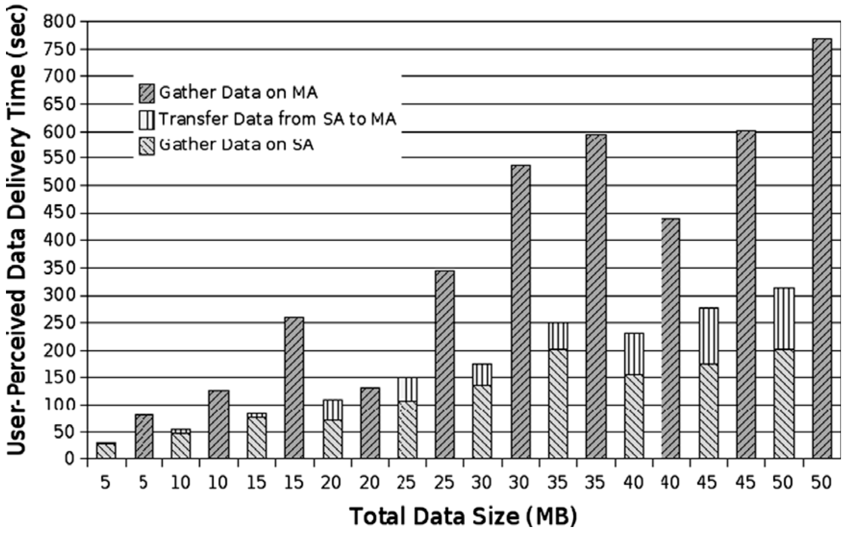


Fig. 7. Data gathering vs. data transfer.

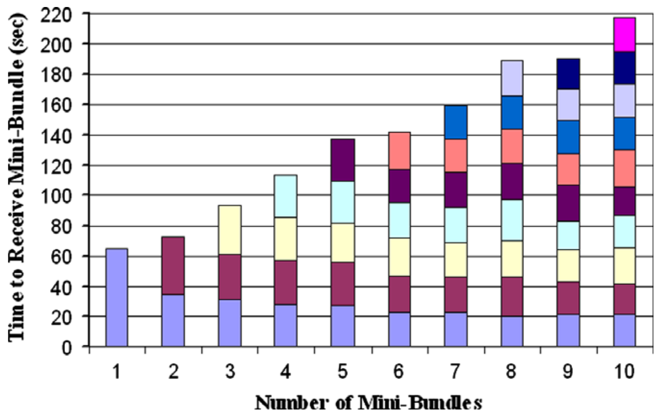


Fig. 8. Effect of mini-bundle quantity on data availability.

While holding disconnection periods constant in the above test is a nice way of isolating the effect of connection duration, it is not realistic. Fig. 10 plots the user-perceived data delivery time against bundle size for the downtown walking, downtown driving, suburb driving, and highway driving models. The trends for all models are very similar, with the highway driving model having the largest user-perceived data delivery time due to its short connection and long disconnection periods. The long disconnection periods in this model also lead to the highest levels of data staleness. However, with such short connections, traditional retrieval methods will suffer worse than our system suffers, making DBS-IC a viable solution in all intermittent connectivity models.

4.2.4. Data throughput

We have so far only vaguely examined data throughput between the SA and the MA. The transfer bandwidth that our system achieves warrants further examination. Therefore, we next examine the instantaneous throughput our system obtains while experiencing 15, 30, and 45 s connection periods. In this test, we hold bundle size constant at 30 MB and disconnection duration constant at 10 s. With no disconnections, our system transfers this 30 MB of compressed data in 59 s. We therefore do not examine the throughput obtained in situations where the MA experiences connection periods of greater than 45 s, since no disconnection would occur during data transfer. In these cases, our system remains beneficial if the 30 MB of compressed data can be transferred during a period of connectivity while the larger, uncompressed data cannot.

In Fig. 11, we see that our system achieves instantaneous throughput of up to 5.24 Mbps. As expected, the throughput drops to 0 when a disconnection occurs. We can further see that the longer the connection period, the less time it takes to transfer the 30 MB of data to the MA. Our system delivers the data to the MA in 82 s when the MA experiences 15 s connections; in 60 s with 30 s connections; and in 58 s with 45 s connections. Even in the worst case of 15 s connection periods, simulating a mobile device moving at 70 mph, our system averages a throughput of 2.92 Mbps. This throughput is the average over the entire data transfer, which includes two periods of disconnection. When discounting the disconnection periods, the throughput increases to an average of 4.14 Mbps. As a comparison, the MA needed 535 s to gather 30 MB of web, file, and email data in Fig. 7, resulting in an average throughput of 0.45 Mbps. By reducing the data transmission to a transfer of bundled data, our system achieves significantly more throughput than traditional retrieval protocols. This increased opportunistic data throughput means the mobile user will have viewable data faster, and that this data will remain viewable even during disconnections.

4.2.5. Prefetched data

In the above set of tests, with the exception of the first, we assumed that the data the user wishes to view has been completely prefetched at the SA. Therefore, we have been considering the transfer of the bundled data as the only factor keeping the mobile user from viewing it. However, there will be added overhead if the data has not been completely prefetched before the user wishes to view it. Fig. 7 showed that even when 0% of the data has been prefetched, our system transfers all the data to the mobile user faster than the user could gather it using traditional techniques. Our system will experience this case of 0% prefetched data when the user modifies the configuration file on the MA. After this change is made, the MA will send the updated configuration file to the SA, the SA will immediately gather and bundle any newly requested data, and send the bundled data back to the MA. Since our system performs well in this extreme case, we can easily modify Fig. 7 for different percentages of prefetched data, and to see that our system performs well in all cases.

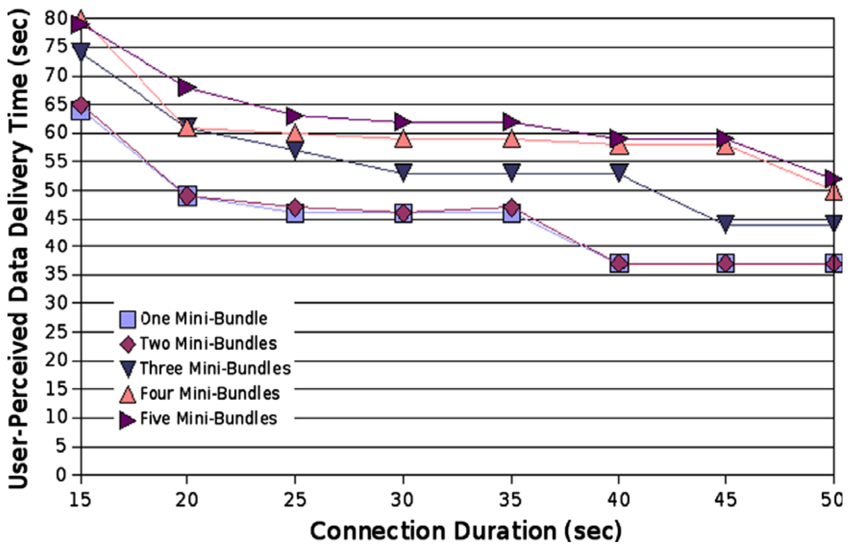


Fig. 9. Effect of connection duration on data availability.

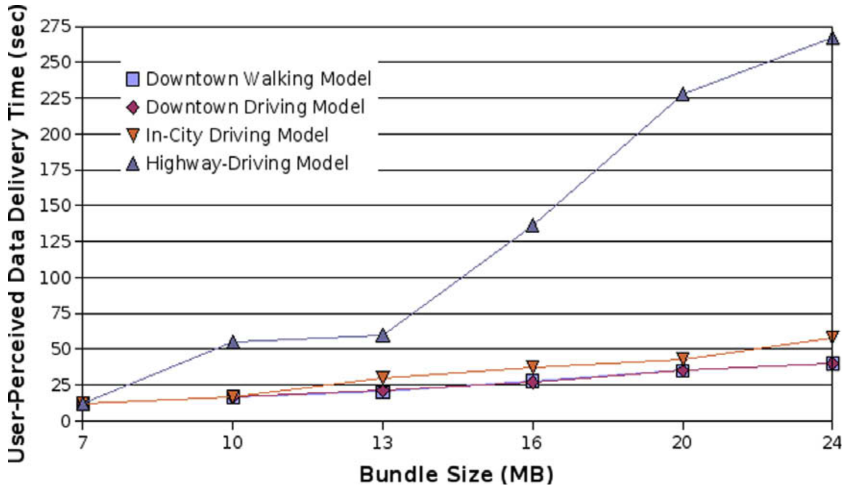


Fig. 10. Effect of connectivity model on data availability.

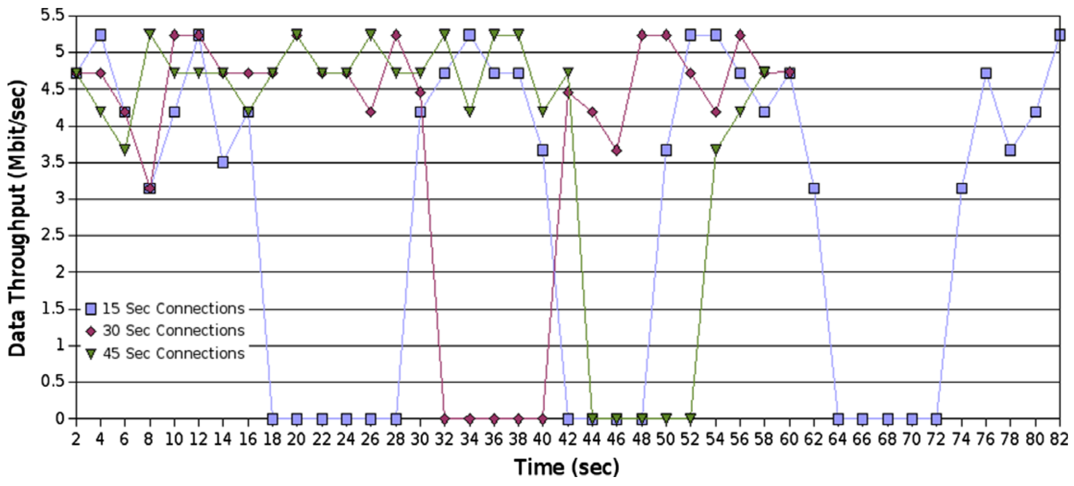


Fig. 11. Effect of connection duration on data throughput.

4.2.6. Round Trip Time (RTT)

The previous tests have been performed with an RTT between 40 ms and about 70 ms. This number is based on the assumption that the stationary agent can be placed on a server account that may be located in another state, or country. Contrary to the nominal value of RTT that we have been using, Fig. 12 shows the time to receive 20 MB data bundle using the downtown driving model, with an average round trip time of 3 ms.

In this experiment, the client or MA was placed within the same AS network as the SA. The round trip time was kept measured between 2 and 4 ms. The round trip time is almost negligible. The interesting message we receive from Fig. 12 is that by almost eliminating the factor of RTT, the linear increase between the number of mini-bundles and the time to receive each of them no longer holds. In other words, we lose the disadvantage of mini-bundles (takes longer to deliver the overall data bundle), while keeping its advantage (the ability to view more pieces of the data during disconnection times). The time to receive the bundles is almost constant regardless of how many mini-bundles the bundle is divided into. Therefore, in such scenarios, in order to improve the transmission of data that is immediately viewable to the user, we can divide the bundle into more mini-bundles.

Therefore, in smaller RTTs, we would worry less about the trade-off that the additional number of mini-bundles could bring in terms of the increase in transfer time. A system that is sensitive to the change in RTT would certainly be an interesting enhancement for future versions of DBS-IC.

4.2.7. Impact of the predictor

We had discussed the predictor component residing on SA in our architecture section. All our tests however, up to this point,

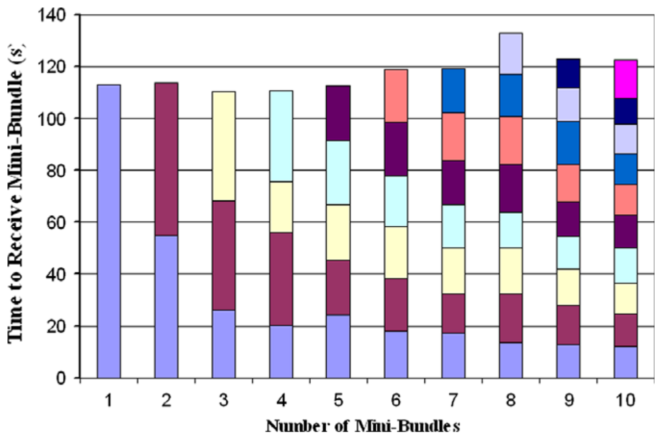


Fig. 12. Impact of a RTT of 3 ms while varying mini-bundles.

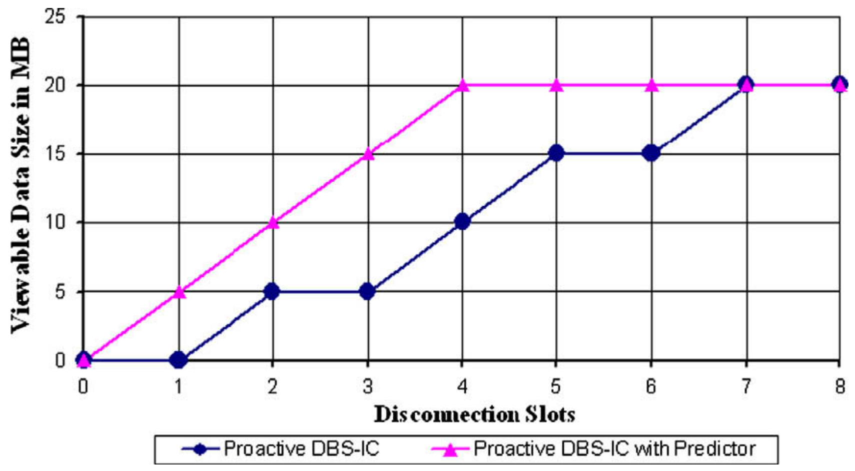


Fig. 13. Impact of the predictor on DBS-IC.

have had this component disabled. The reason behind this decision is that we wanted to isolate the impact of various parts of our system, in order to understand the impact of each part. We finally come to the point where we discuss the impact of the predictor.

The main goal of the predictor component is to dynamically pick a more suitable mini-bundle size based on the connectivity duration of the MA. This goal should ultimately enable the user to view more data during times of disconnections when compared to DBS-IC without the predictor.

In this evaluation we applied our nominal values for the various parameters, and ran our experiments with and without the predictor. The initial number of mini-bundles was set to a default value of 5 and the MA was running in a remote location. The round trip time was kept constant at 68 ms, as well as the bundle size at 20 MB, the situation where 100% of the data is prefetched, using the suburb driving model. Initially in his test, the SA gathers and bundles 20 MB of data and sends them to the MA divided among the default value of five mini-bundles each of size 4 MB.

Fig. 13 represents the values that were obtained from this test. We measure the amount of viewable data at the MA during a disconnection slot. The reason we use disconnection slots in the *X*-axis, is that we believe that disconnection durations are less predictable than connectivity duration. If a user is disconnected there could be a traffic jam, a long traffic light, or the user is simply sitting in an area with no access points. However, once the user moves, his speed is usually predictable (based on his location and method of mobility). When this user passes by an access point, it is easier to predict the worst case scenario of duration of connectivity.

Returning to Fig. 13, we observe that in the case of the DBS-IC with the predictor enabled, the entire bundle is received by the fourth disconnection slot. On the other hand, in the case of the DBS-IC with the predictor disabled, the entire bundle is received after the seventh disconnection slot. The other interesting aspect is that when the predictor is enabled, there is more viewable data ready for the user during most disconnection slots. We usually have double, or sometimes triple the amount of data available because the predictor was able to dynamically determine a more appropriate mini-bundle size to send to the MA. Overall, the results here reinforce the vision set by the predictor in Fig. 6.

5. Conclusions

In this paper we have presented a Data Bundling System for Intermittent Connections (DBS-IC), a system which deals with intermittent connectivity by pro-actively delivering data to the mobile user. DBS-IC is comprised of a Stationary Agent (SA), located on a machine with a stable connection to the Internet, and a Mobile Agent (MA), located on an intermittently connected mobile device. By confining the gathering of web, email, and file data to the SA, DBS-IC reduces the data transfer on the mobile device to a bulk TCP data transfer, which allows our system to utilize available bandwidth extremely well. We find that our system can make data available to the mobile user up to 20 times faster than if the data were gathered on the mobile device itself, even when the mobile device is only experiencing connection periods of 20 s at a time. This number could double when the predictor component in DBS-IC is enabled.

Even though current technologies and research offer alternative end-to-end connections on small mobile devices, the current trend is that these connections are expensive, and do not provide sufficiently high bandwidth. Our system takes advantage of the explosive deployment of access points where it can take advantage of any open access point that might be available.

We believe that DBS-IC is a solid step to improve mobile users' experience in the face of intermittent connectivity. However, there are several areas for future work. A valuable future improvement to our system would be the addition of an intelligent gathering and bundling agent. This agent would be located on the SA and would use past viewing trends to dynamically decide which data the user might need in the future. Our system could further be extended to handle interactive data, caching user requests during times of disconnection. This extension would be especially useful with interactive web pages that require user input, and with newly composed emails that the user wishes to send. Built on top of our work, these improvements would help make the in-motion mobile user's experience almost equal to that of a stationary user's.

References

[1] DTNRG, Delay Tolerant Networking Research Group. Available from: <http://www.dtnrg.org/>.
[2] The Drive-thru Internet Project. Available from: <http://www.drive-thru-internet.org/>.
[3] The DHARMA Project. Available from: <http://dharma.cis.upenn.edu/>.
[4] H. Chang, C. Tait, N. Cohen, M. Shapiro, S. Mastriann, R. Floyd, B. Housel, D. Lindquist, Web browsing in a wireless environment: disconnected and asynchronous operation in ARTour Web Express, in: ACM/IEEE International Conference on Mobile Computing and Networking, Budapest, Hungary, 1997.
[5] K. Fall, A delay-tolerant network architecture for challenged Internets, in: ACM SIGCOMM, Karlsruhe, Germany, August 2003.
[6] S. Farrell, V. Cahill, Delay- and Disruption-Tolerant Networking, Artech House, 2006.

[7] S. Farrell, V. Cahill, Evaluating LTP-T: a DTN-friendly transport protocol, in: International Workshop on Satellite and Space Communication (IWSSC), Salzburg, Austria, September 2007.

[8] R. Gass, J. Scott, C. Diot, Measurements of in-motion 802.11 networking, in: Workshop on Mobile Computing Systems and Applications (WMCSA), Semiahmoo Resort, WA, April 2006.

[9] K. Harras, K. Almeroth, Inter-regional messenger scheduling in delay tolerant mobile networks, in: IEEE World of Wireless, Mobile and Multimedia Networks (WoWMoM), Buffalo, NY, June 2006.

[10] K. Harras, K. Almeroth, E. Belding-Royer, Delay tolerant mobile networks (DTMNs): controlled flooding schemes in sparse mobile networks, in: IFIP Networking, Waterloo, Canada, May 2005.

[11] C. Holman, K. Harras, K. Almeroth, E. Belding, A proactive data bundling system for intermittent mobile connections, in: IEEE SECON, Reston, VA, September 2006.

[12] S. Jain, K. Fall, R. Patra, Routing in a delay tolerant network, in: ACM SIGCOMM, Portland, OR, August 2004.

[13] D. Johnson, D. Maltz, Dynamic Source Routing in Ad Hoc Wireless Networks, vol. 353, Kluwer Academic Publishers, 1996.

[14] M. Kaddour, L. Pautet, A middleware for supporting disconnections and multi-network access in mobile environments, in: PERCOM, Orlando, FL, March 2004.

[15] P. Kulkarni, P. Shenoy, K. Ramamritham, Handling client mobility and intermittent connectivity in mobile web accesses, in: MDM 2003, Melbourne, Australia, January 2003, pp. 401–407.

[16] Q. Li, D. Rus, Sending messages to mobile users in disconnected ad-hoc wireless networks, in: ACM MobiCom, Boston, MA, August 2000, pp. 44–55.

[17] Y. Mao, B. Knuttson, H. Lu, J.M. Smith, Dharma: distributed home agent for robust mobile access, in: IEEE INFOCOM, Miami, FL, March 2005.

[18] A.M. Keller, O. Densmore, W. Huang, B. Razavi, Zippering: managing intermittent connectivity in DIANA, Mobile Networks and Applications 2 (4) (1997) 357–364.

[19] J. Ott, D. Kutscher, Drive-thru Internet: IEEE 802.11b for automobile users, in: IEEE INFOCOM, Hong Kong, March 2004.

[20] J. Ott, D. Kutscher, A disconnection-tolerant transport for drive-thru Internet environments, in: IEEE INFOCOM, Miami, FL, March 2005.

[21] V.N. Padmanabhan, Using predictive prefetching to improve world wide latency, in: ACM SIGCOMM, Stanford, CA, July 1996, pp. 22–36.

[22] C. Perkins, Ad-hoc on-demand distance vector routing, in: IEEE Workshop on Mobile Computing Systems and Applications, New Orleans, LA, February 1999, pp. 90–100.

[23] C. Perkins, P. Bhagwat, Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers, in: ACM SIGCOMM, London, England, October 1994, pp. 234–244.

[24] E. Royer, C. Toh, A review of current routing protocols for ad-hoc mobile wireless networks, IEEE Personal Communications Magazine 6 (2) (1999) 46–55.

[25] K. Scott, S. Burleigh, Bundle Protocol Specification, IETF Internet Draft RFC-5050, November 2007.

[26] A. Vahdat, D. Becker, Epidemic routing for partially connected ad hoc networks, Technical Report CS-200006, Duke University, April 2000.

[27] W. Zhao, M. Ammar, E. Zegura, A message ferrying approach for data delivery in sparse mobile ad hoc networks, in: ACM MobiHoc, Tokyo, Japan, May 2004.

[28] W. Zhao, M. Ammar, E. Zegura, Controlling the mobility of multiple data transport ferries in a delay-tolerant network, in: IEEE INFOCOM, Miami, FL, March 2005.