Bundling the Web: HTTP over DTN

Jörg Ott
Helsinki University of Technology
Networking Laboratory
jo@netlab.tkk.fi

Dirk Kutscher
Universität Bremen
Technologie-Zentrum Informatik
dku@tzi.org

ABSTRACT

Mobile users traveling by car, bus, or train usually experience varying connectivity characteristics while they are on their way, including unpredictable (loss of) network access, changes in a data rate, and the like. In short, they operate in a challenged networking environment that is not well suited for many Internet applications. While different approaches have been developed to specifically mitigate (short-term) disconnections while at least partly preserving the end-to-end notion of such applications, Delay-tolerant Networking (DTN) takes a different approach by relying exclusively on asynchronous communications. While this fits well with the mobile user connectivity, particularly interactive applications (such as web access) are affected by the lack of synchronous end-to-end interaction. In this paper, we present protocol mechanisms for running *HTTP-over-DTN* as well as a system architecture for incremental deployment, and we report on findings from measurements performed with our prototype implementation.

Categories and Subject Descriptors

C.2.2 [**Computer Communication Networks**]: Network Protocols; C.2.6 [**Computer Communication Networks**]: Internetworking

General Terms

Design, Experimentation, Algorithms, Performance

Keywords

Delay-tolerant Networking, DTN, Mobility, HTTP, WWW

1. INTRODUCTION

Mobile communication has undoubtedly become a key ingredient of everyday life for many people. While mobile telephony and messaging are ready to use for everyone, other applications, particularly those not running on mobile phones but rather on laptops or PDAs, still do not work well with the mobile environment and its characteristics. Limiting factors for mobile (data) communications include the risk of signal (and thus connection) loss and variation in access link quality both of which may lead to disruptions in the regular interaction of applications.[1]

Numerous suggestions are focusing on mitigating the risk of connectivity loss and maximizing the connection quality to keep users *always best connected* [1] [2]. Thereby, the focus remains on networking technologies and their capabilities, largely ignoring related issues of potential disconnection: cost associated with staying connected (which may vary tremendously across different link layer technologies and service providers); legal or social requirements leading to connectivity interruptions (e.g., during take-off and landing of an aircraft); and aspects concerning the user (who may want or need to turn off a mobile device) and the device itself (e.g., running out of battery); among others. In particular, however, this does not consider the respective application needs.

Some of the typical applications of mobile or nomadic users—such as email, calendar synchronization, and database access—may basically operate asynchronously [3] and support for this is often provided by the application software. But even interactive applications such as web access (retrieving web pages and accessing shared web-based information spaces or editors) and (instant) messaging do not require permanent connectivity: the user is busy most of the time reading, thinking, or typing, leaving a connection unused during this time—and, in principle, occasional message exchanges would suffice to update or synchronize the mobile user with her (fixed) peer(s). While such information exchanges are basically possible even in mobile scenarios with intermittent connectivity, the main issue preventing today's web applications from functioning smoothly in a dynamic mobile environment are that (1) connectivity may not be available when the application wants to communicate and (2) an end-to-end path between the two application peers may not exist at all. The former (1) means that the application transaction (and thus the user) may experience unexpected delays of unpredictable duration which needs to be concealed. Depending on the nature of connectivity, highly interactive application protocols such as HTTP (where sequences of requests are used to obtain a web page), are not well suited for environments requiring delay-tolerant communications. The latter (2) implies that applications can no longer rely on end-to-end properties inherited, e.g., from the transport layer (such as secure end-to-end communications as provided by TLS).

In this paper, we address the former issue of enhancing HTTP to become workable in a delay-tolerant networking (DTN) environment. Building on our earlier proof-of-concept work on this subject [4], we focus on the details for running HTTP in intermittently

[1]Both factors also inhibit voice communications but, with voice, the necessary recovery actions are taken—more or less efficiently—by the human users.

WNEPT 2006, August 10, 2006, Waterloo, Ontario, Canada.

connected environments. In particular, we specify the *HTTP-over-DTN* protocol operation and describe a system architecture suitable for incremental deployment in today's Internet.

In section 2, we briefly review the Drive-thru Internet scenario for opportunistic mobile Internet access. We present related work concerning the core aspects of mobile web access in section 3 and summarize different resulting architectures for (mobile) web access in section 4. We present our design for HTTP-over-DTN and deployment-related aspects in section 5 and describe our ongoing implementation efforts, measurements, and findings in section 6. Section 7 concludes this paper with a short summary and hints at future work.

2. DRIVE-THRU INTERNET AND DTN

In the Drive-thru Internet project [5], we have been investigating mobile Internet access for users in vehicular environments, e.g., aboard a car, bus, or train. Figure 1 shows a mobile user in a vehicle passing through wireless LAN hot-spots—connectivity islands—located at the roadside. The hot-spots may be dedicated for travelers, e.g., at gas stations or restaurants in rest areas, or may be additionally available for use in city areas from cafés or other locations. Hot-spots may comprise one or more access points and may be operated by different WISPs as is shown on the left hand side of the figure [6].

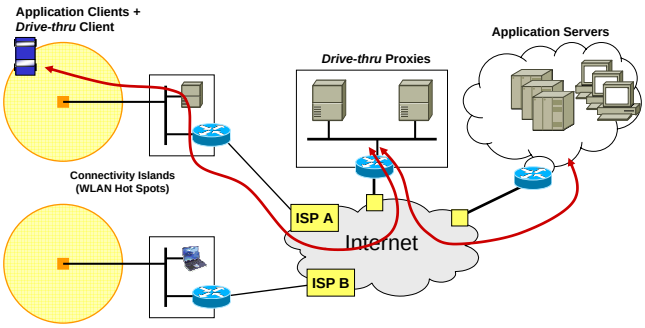

Figure 1: Example for Mobile Internet Access on the Road

When a mobile user travels, she will experience periods with and without connectivity: The beginning of a connectivity period may often not be predictable, their respective end may be anticipated from signal strength readings. Due to traffic conditions on the road and obstacles (buildings, vegetation, bridges), the reach of a connectivity island may vary as will the quality of the radio channel (and hence transmission and error rates). These also heavily depend on the radio equipment installed in the vehicle or the mobile device and used in the hot-spot. Our measurements have shown that, with proper equipment, mobile users may, e.g., obtain more than 60 s of connectivity at 120 km/h (up to 2 km) and may transfer some 30–70 MB of data to/from a peer located in the hot-spot in a single pass [7]. A mobile user shares the bandwidth reasonably fairly with fixed and other mobile users. Automatic authentication with wireless ISPs and auto-configuration can enable users to effectively use most of their connectivity window for data transfer [8, 9].

To support existing Internet applications in such an environment, we have applied connection-splitting between the mobile and the fixed endpoints, realized in two dedicated components: the *Drive-thru client* running locally to or on the mobile device and the *Drive-thru proxy* somewhere well-connected in the fixed network. Our *Persistent Connection Management Protocol (PCMP)* follows a session layer approach towards achieving continued application sessions in spite of connectivity interruptions, changing IP addresses, NATs/firewalls, and highly variable performance characteristics. Application-specific modules implemented as plug-ins on the Drive-thru client and proxy perform additional functions to prevent application layer timeouts, provide user feedback, and allow for asynchronous processing of messages (exchanged with the application peers) while the user is disconnected—thereby also offering performance enhancement to maximize the utilization of the short connectivity period [10].

In [4], we have introduced the general concept of treating a mobile user and her applications as node in a *delay-tolerant network (DTN)* and have shown that interactive applications such as HTTP can operate in this environment, albeit at suboptimal performance. In this paper, we formalize HTTP over DTN regarding the protocol operation and discuss an architecture to enable incremental deployment.

3. RELATED WORK

We combine aspects of mobile and vehicular communications with HTTP-specific approaches for optimization and offline operation and apply the concept of delay-tolerant networking to enable usable mobile web access.

3.1 Mobile and Vehicular Networking

IP communications on the road has independently been studied by the FleetNet [11] and Networks on Wheels [12] projects, albeit with a different focus and slightly different goals: both primarily target inter-vehicle communications in wireless ad hoc networks for traffic-related control information and data sharing across vehicles. Despite Internet access was only considered a secondary objective, a proxy-based architecture was developed [13]. Network access for vehicles via multiple (complementary) networks to keep users always best connected has also been addressed in various projects, including OverDRIVE [14], IPonAir [15], and the Mobile Access Router [16]. A mobile router dealing with temporary connectivity loss is also studied in the eMotion project [17]. The latter two implement dedicated mobile routers to provide wireless network access, addressing multi-provider support at the IP layer and temporary connectivity interruptions at the transport-layer, respectively. The DHARMA project targets agent-supported mobility in general [18] as does the Tetherless computing architecture (TCA) [19]. All of these projects have devised their own infrastructure support to deal with intermittent connectivity. Finally, a vast body of research has studied performance improvements for reliable (TCP-based) communication across wireless links, which we acknowledge but for which we do not provide references in this paper as these techniques are not directly related to our focus.

3.2 Mobile and Offline Web Access

HTTP-based access to web pages from mobile users and challenged environments has been discussed for about a decade and various approaches have been pursued in research or in deployed products. We can roughly distinguish three different ideas that are somewhat related: offline operation, cache prefill, and on-demand (predictive) prefetching [20, 21] in addition to TCP acceleration.

Tools such as `wget`[2], `puf`[3], or `wwwoffle`[4] allow replicating contents of a web server to some local storage which can then be

[2]http://www.gnu.org/software/wget/
[3]http://puf.sourceforge.net/
[4]http://www.gedanken.demon.co.uk/wwwoffle/

accessed when offline. Similar functions have been built into web browsers that support some offline mode of operation. These tools often provide means for manual content synchronization (e.g., invoked by the user), however, contents not available from the local storage cannot be accessed while offline.

In contrast, cache prefill aims at improving the web access performance by populating caches with information that will likely be requested by the user in the not too distant future (before its expiry). Cache prefill can be push-based (initiated by content providers) or pull-based (initiated by the cache), and contents to be pushed or pulled (also: *prefetched*) will typically be selected following some access heuristics. While push is usually based upon some dedicated distribution or replication protocol, pull often employs HTTP, possibly augmented by inter-cache protocols such as ICP, to retrieve the desired resources. This type of cache prefill usually operates asynchronously to actual user requests and it is hence unknown whether the requested resource will be used at all [22].

A special case we refer to as *on-demand prefetching* is used to improve user experience across challenged links (e.g., low bit rate and high error rate for mobile users or high latency for geo satellite users). A user request may trigger prefetching of resources related to the requested one (such as objects embedded in a web page) to be delivered to the user along with the requested resource and thus save uplink bandwidth and reduce latency. But this comes at the expense of potentially unnecessarily prefetched objects—the probability of which is lower than with cache prefill because prefetching is invoked by an actual user request. While not perfect due to the increasing complexity of web resources, significant performance improvements can be achieved [23].

Combinations of HTTP acceleration and cache prefill are used in public transportation systems such as trains and airplanes for passenger entertainment. Such solutions are suitable for public transportation environments in which networking infrastructure (such as the satellite links in *Connexion by Boeing*[5]) are available. Even then, disconnections may occur (e.g., if buses or trains pass through longer tunnels) or communication performance of the instantly available links (such as GRPS) may not be sufficient. This applies even more to personal transportation (e.g., a mobile user in her car) or when no supportive infrastructure is available on public transport—which calls for delay- and disruption-tolerance for mobile web applications as we have presented in section 2.

3.3 Delay-tolerant Networking

Applying delay-tolerant networking (DTN) principles to (mobile) communications removes the need for an end-to-end path and therefore is attractive to a set of mobile applications that do not inherently require immediate interaction. DTN uses asynchronous communications (modeled after email): Rather than sending (small) packets end-to-end, DTN endpoints exchange messages of arbitrary size (*bundles*) forwarded by DTN routers (also referred to as *bundle routers, BRs*) hop-by-hop from the source to the destination. Conceptually, DTN operates above the transport layer and may interconnect different internets with arbitrary underlying protocol stacks [24]. A *custody transfer* mode allows ensuring reliable delivery while delegating the responsibility to the next capable DTN router. Status reports may convey information about the delivery progress of a bundle and include forwarding and custody notifications from intermediate routers as well as receipts from the receiver to achieve end-to-end semantics [25]. DTN endpoints and applications are identified by *endpoint identifiers, EIDs* specified in a URI-style format: *scheme:scheme-specific-part*. The *scheme* defines the scope within which the *scheme-specific part*, i.e., the actual address, is interpreted.

DTN routing and forwarding offers particular challenges due to the delay tolerance of communications. While (Internet) and ad-hoc routing protocols determine an end-to-end path when they have to forward a packet (and may report an error if none is found), DTN routing must also account for potential or known future paths so that the complexity of forwarding decisions increases [26]. Furthermore, if paths are only available intermittently, no continuous exchange of routing information is possible. And, for links that become available only opportunistically, like with ad-hoc networking, predictions are difficult. For mobile ad-hoc networking environments, DTN routers rely on various kinds of information replication for forwarding bundles to maximize delivery probability and minimize transit time. Such forwarding schemes, for example, follow the concept of *epidemic routing* [27], with numerous variants having been developed such as [28, 29, 30].

4. HTTP SCENARIOS

HTTP is designed to perform end-to-end interactions between a client and an origin server but also supports the introduction of intermediaries (web caches, proxies). The latter are usually separate physical entities but may also be integrated or co-located with the client (e.g., internal web caches). In all cases, however, permanently available connectivity is assumed so that resources can always be retrieved from the origin server. This "regular" mode of operation governing today's Internet is depicted as scenario 1 in figure 2.[6] *Performance Enhancing Proxies (PEPs)* [31], explicitly configured or transparently inserted, are often used to deal with challenged networks as shown in scenario 2. While they are typically meant to improve performance for connected users (using, e.g., mobile or satellite links), approaches such as the Drive-thru Internet architecture described in section 2 or the DHARMA project [18] make use of proxies to conceal temporary complete loss of connectivity. Scenario 3 shows an alternative class of approaches in which the functionality to maintain sessions across disruptions is built into the end points [32, 33], supported through appropriate mechanisms provided by the operating system.

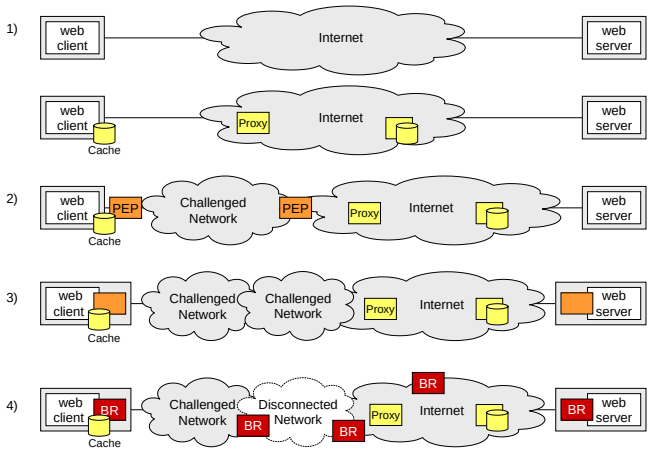

Figure 2: Options for HTTP operation

[5]http://www.connexionbyboeing.com/

[6]Note that proxies and caches may be introduced by service providers without the user noticing, e.g., to improve performance, augment content, or perform content adaptation.

The end-to-end approach has the clear advantage that, besides the obvious avoidance of additional points of failure (fate sharing), it is not necessary to identify the location of the challenged links and more than one challenged link is implicitly dealt with. The downside of end-to-end approaches operating at the transport or a session layer is that an end-to-end path has to exist in the first place. We have suggested that Delay-tolerant Networking can be a suitable communication platform for mobile web access [4] as depicted in scenario 4: the resulting complete decoupling of client and server allows a maximum of flexibility when dealing with intermittent connectivity, at the expense of losing interactivity. Nevertheless, less interactive communication may often be preferable to not communicating at all.

5. HTTP OVER DTN

Running HTTP across a DTN-based infrastructure needs to be addressed in two dimensions: (1) From a plain transport perspective, the bundle protocol can be used as a substitute for TCP in which case a suitable protocol mapping needs to define how to carry HTTP payloads in bundles, how to map addresses, and how to achieve reliability. (2) From an application perspective, the impact of disruptions needs to be addressed since, otherwise, delays or disconnections occurring during HTTP-style interactive retrieval of the resources belonging to a web page would render the user experience unacceptable [10, 3, 4].

While pursuing (1) alone will be insufficient in most cases, simply substituting TCP by the DTN bundle protocol in environments with permanent end-to-end connectivity allows benchmarking the additional overhead from introducing DTN for a single HTTP transaction. This may also serve as the baseline for interoperability if we assume a migration from TCP to a DTN transport if functionality according to (2) is not available or not desirable as we will discuss below.

5.1 HTTP over DTN Transport

As introduced in section 3.3, the bundle protocol supports the delivery of arbitrarily sized bundles. The natural approach is to perform a one-to-one mapping of HTTP requests and responses onto bundles.

In an HTTP context, addressing bundles is basically straightforward using HTTP URIs: Just as with regular HTTP, *request* bundles are either addressed to the origin server at which a request is targeted or to a configured intermediary while *response* bundles are addressed to the originator of the corresponding request. Demultiplexing from other applications takes place using the scheme identifier (http) and we assume the host part of the scheme-specific part to be used for routing.

5.2 Application Layer Bundling: Content and Message Aggregation

When intermediaries are used several requests or responses may be *bundled* to reduce overhead and improve efficiency. This requires an aggregation scheme for independent requests and responses (including HTTP headers and the respective message bodies) and a way to match responses to requests. One option is to use Multipart MIME [34] paired with a suitable content type for HTTP messages ("message/http") for aggregation and to perform matching requests and responses by means of the `Content-Location` MIME header. As some environments may demand a more compact representation, alternatively, a general bundle-in-bundle encapsulation is conceivable.

Since web pages typically consist of many objects, a request for a particular page will be followed by numerous additional ones for the "embedded" objects. With delay-tolerant communication, one-by-one retrieval may not be workable at all or may simply take too long for an acceptable user experience. Instead, all the resources required to display a web page could be retrieved upon a single request and returned in *bundled* format, ideally a single response. One suitable format to return associated resources in response to a request is MHTML [35] that defines a MIME-based aggregation scheme for HTML and other contents. Alternatively, a file system-based format such as the directory tree structure created by *wget* or *puf* bundled in an archive format (such as *tar*) is also workable. Both options are orthogonal to and can be combined with bundles containing multiple requests.

5.3 Application Layer Bundling: Content Retrieval

With the HTTP encapsulation format in place, we discuss several conceivable scenarios how and where to create the content bundle in response to a request as shown in figure 3:

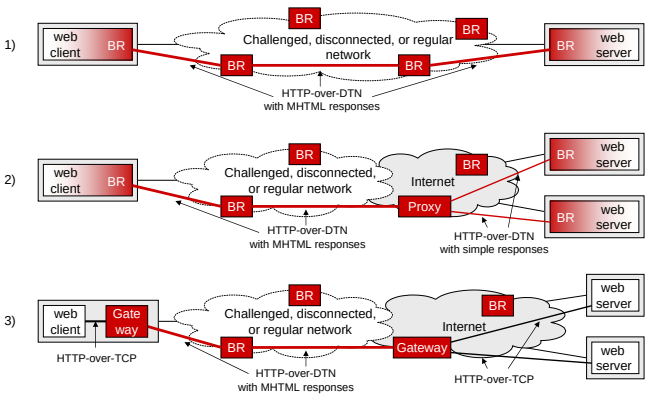

Figure 3: Alternatives for web access in challenged networks

5.3.1 *End-to-end HTTP-over-DTN*

Ideally, both client and server (as well as potential intermediaries) implement HTTP-over-DTN so that bundles can be sent directly to the respective (origin) server. A DTN-aware server is expected to have locally stored knowledge—statically supplied by the web page creator or dynamically created per request—that identifies all the associated objects for each resource so that all objects making up a web page can be returned in a single request. For resources for which such information is not available, the web server reverts back to the plain HTTP-over-DTN transport operation described above.

Web browsers inherently supporting HTTP-over-DTN formulate requests as (aggregated) bundles and interpret bundled contents from responses. Such DTN-aware browsers should be able to provide adequate feedback to users: this may include, besides indicating the progress of a page retrieval, showing whether the user is presently connected or not and distinguishing server non-reachability due to temporary disconnection from unknown or non-operational servers. Even the estimated time of arrival for web resource might be predicted.

5.3.2 *Proxy-based HTTP-over-DTN*

Introducing proxies into an HTTP-over-DTN environment may support a mobile node in *content aggregation* from one or more origin servers. As web pages often contain objects from multiple

origin servers (e.g., advertisements), a client requesting a single resource will thus have to receive multiple responses without knowing so a priori. Therefore, either a two-step retrieval is required in which the client first receives all the objects the origin server of the requested resource can provide and then analyzes these in a second step to retrieve the missing pieces—which one may want to avoid in challenged environments. Or, a proxy "closer" (i.e., better connected) to the origin servers gathers the necessary resources on the client's behalf and then returns the bundled content to the client in a single response. This intermediary may also be the initially contacted origin server itself.

But even when retrieving resources only from a single origin server, charging this server with the task of collecting all local resources associated with a single web page requested by a client may incur significant load both in computation and subsequently required data transmission capacity—and this creates an easy to exploit target for DoS attacks.[7] To remedy this risk, proxies trusted by the users may be introduced: after authenticating a user, a proxy acts on their behalf and performs the resource collection for each request, thus maintaining anonymity towards the servers while distributing the load incurred by a single request. Proxies collecting resources from one or more origin servers interact with the latter via HTTP-over-DTN or HTTP-over-TCP (see next subsection). They gather the resources and then return them as a single bundle to the initiating web client. To improve performance, they may implement local caching.

An enhanced origin server may support third party resource collection by providing an explicit list of resources required by a web page. This information should be made available as a set of HTTP headers—`Embedded-Resource`—when responding and include both local resources required as well as pointers to external resources, all of which are identified by their respective URIs to a request to avoid extra round-trips.[8] Typical HTTP response headers are usually a few hundred bytes in size (147–502 bytes for accessing the web pages investigated in section 6) followed by the message body containing the initial resource of some kilobytes (3–120 KB with a mean of some 26 KB for these web pages). Each additional `Embedded-Resource` header adds another 21 bytes (header name, blank, and CRLF) plus the URI size to the response. The investigated web pages comprised 6–115 resources (some 38 on average) with a mean URI length of 60 bytes so that some 2–3 KB of header fields would be added on average to the response containing the initially retrieved resource, or roughly 2–95 %. To avoid this overhead if not needed, the additional headers are only returned if the proxy includes an `Embedded-Resource-Request` header in which it lists which resources it would like to obtain: `none` does not request any resources, `embedded` indicates that all embedded objects are asked for, and `linked` may hint that pointers to the referenced web pages are also desired.[9]

5.3.3 Gatewaying HTTP-over-DTN

Finally, the sheer number and variety of web clients and servers and their independent administration makes anything but incremental deployment unrealistic. With web clients and web servers speaking only HTTP-over-TCP, another intermediary is required to convert between HTTP-over-DTN (using one of the formats introduced above) and HTTP-over-TCP operation. Such a gateway must also perform the necessary aggregation (on the server-side) and deaggregation (towards the client) functions so that existing web browsers and servers may be used.

Even an unmodified server may support resource collection for DTN-based HTTP access: as (static) HTTP resources are typically stored in files (e.g., *resource.html*), for each resource with dependencies on other resources, an additional *dependency file* may be placed on the server following a simple naming convention (e.g., *resource.html.dep*), either (manually) by the creator of the web page or automatically by a web design tool or some content management system. A client, proxy, or gateway simply attempts to retrieve this dependency file using a regular HTTP GET request. It needs to do so only if the resource type indicates the possibility for embedded resources—which is typically identified by the filename suffix or subsequently by the `Content-Type` header of the response containing the resource—so that the additional overhead is fairly low. An empty dependency file indicates that no dependencies exist while a `404 Not Found` error code shows that the corresponding server has no dependency information associated with this particular resource. In the latter case, a gateway has to revert to predictive prefetching to collect resources for the mobile user.

Towards a web client, similar to Drive-thru Internet as described in section 2, a (local) gateway maintains the transport connections and may generate temporary pages until the sought resources are retrieved [36] where it also may indicate further information (such as the expected availability).

5.4 A Gateway Resource Retrieval Algorithm

With the aforementioned variety of operational options available, it is easy to foresee that some if not all of them will coexist. In particular, when beginning to introduce DTN-based web access, this will have to rely on gateways being available for an extended period of time. In the following, we describe an algorithm that to be executed by DTN-over-HTTP gateway located "in the fixed network" to fetch resources on behalf of a mobile user and forward the collected resources as bundles.[10] The algorithm stepwise explores all the aforementioned options for retrieving web resources as efficiently as possible. We only describe the interaction with the server and assume that the contents will be returned to the client as a bundle; whether the user's web browser natively supports HTTP-over-DTN or whether a co-located personal gateway performs the necessary conversion for a web browser is not relevant here.

The algorithm is shown in pseudo code in figure 4. It maintains some local state to remember whether a particular server supported DTN before. For the first contact with a given server S, the gateway simply contacts the server assuming that in a well-connected Internet environment, non-support will be detected quickly so that a short timeout can be used.[11] A response to a DTN-based request will either return all resources associated with the web page or just the requested resource itself. In the latter case, subsequent (DTN-based) interactions with this and possibly other servers are necessary to retrieve the missing pieces (using additional information supplied by the web server).

If DTN is not supported, the gateway reverts to plain HTTP-over-TCP to request the resource but includes a hint that it supports

[7] Unless all the client requests are authenticated which, however, would counteract anonymous web access.

[8] Embedded resources could also be listed in an additional meta element in the HEAD of an HTML document; this would, however, make the solution dependent on the document type and would require the proxy to parse document contents.

[9] The latter could ultimately support recursive retrieval of parts of a web site up to a given depth, similar to *wget*, to allow offline navigation.

[10] A variant could also be run on the mobile client or as part of a web browser.

[11] Alternatively, a DTN-capable origin server could include a `DTN-Supported` header in each response and the gateway could issue the first request using plain HTTP to learn about this capability.

Gateway algorithm: **Retrieve (URI)**

```
receive request bundle request for a URI on server S
if (S supports DTN || nothing known about S) {
  // try HTTP-over-DTN first
  send bundle request to S and wait for response
  if (response received) {
    if (error || response body contains all resources) {
      forward response bundle towards originator
      return
    } else {
      analyze bundle response and fetch embedded objects
      bundle responses and forward to originator
      return
    }
  } else {
    // timeout -> note that server S does not support DTN
  }
  // Try plain HTTP
  create HTTP request with Embedded-Resource-Request header
  send HTTP request to S and wait for response
  if (response received && no error) {
    if (response contains Embedded-Resource: headers) {
      parse Embedded-Resource: headers
      recurse to retrieve all indicated resources via HTTP
      aggregate the the collected resources in a bundle
      send the response bundle towards the originator
    } else {
      // server does not support additional HTTP headers
      create HTTP request for request URI dependency file
      send HTTP request to S and wait for response
      if (response received && no error) {
// dependency file received
        parse returned dependency file
        recurse to retrieve contained resources via HTTP
        aggregate the the collected resources in a bundle
        send the response bundle towards the originator
      } else {
        // no dependency file -> revert to prefetching
        execute predictive prefetching for request URI
        aggregate the the collected resources in a bundle
        send the response bundle towards the originator
      }
    }
  } else {
    forward error response to originator as bundle
  }
}
```

Figure 4: DTN-HTTP gateway algorithm for resource retrieval

information about embedded objects. If the response includes a list of objects, the gateways subsequently retrieves these. Otherwise, the gateway additionally requests a potential dependency file from the server. If such a file is available, the gateway extracts the additional URIs and requests the missing resources similar to the previous step. If no further information is available, the gateway uses predictive prefetching as last resort to obtain the necessary resources. In any case, the collected resources are aggregated in a bundle and forwarded to the originator via DTN.

6. MEASUREMENTS

Considering the mobile Internet access scenario discussed in section 2, we are interested in the performance of our alternative retrieval schemes for web pages using DTN compared to the performance of plain HTTP. We have used our prototype DTN application proxy for plain TCP connections and HTTP, *dtntcp* [4] and enhanced it to allow us simulating the availability of dependency lists offered by the server. *dtntcp* has been developed for Linux 2.4, is written in plain C, and is based on the DTN reference implementation version 2.2.0 [37], using a snapshot from the public cvs repository from May 2006. We have used the *dtnd* included in the distribution as DTN bundle routers.

While *dtntcp* is designed in a symmetric fashion, command line parameters can be used to configure one instance as the (mobile) client and another one as the (fixed) server. On the client side, *dtntcp* accepts incoming connections on a configured transport address and forwards the received data as bundles to a configured peer (identified by its DTN EID). On the server side, *dtntcp* accepts incoming bundles, opens a new TCP connection to the target, and then forwards the data via TCP. After connection setup, data forwarding operates largely symmetrically and if one TCP connection is torn down a corresponding event is relayed via DTN to the peer. *dtntcp* may be configured to interpret incoming data as HTTP requests and responses and react accordingly which we use to implement three modes of operation:

1) Simple HTTP forwarding: HTTP requests and responses are parsed and forwarded as bundles to the corresponding peers. This mode of operation is analogous to plain HTTP with TCP being replaced by the bundle protocol.

2) HTTP prefetching: The server-side *dtntcp* accepts an incoming HTTP requests and then forks and executes a the *Parallel URI Fetcher (puf)*[12] for parallel prefetching of the resources belonging to a web page. The retrieved resources are collected in a directory in the local file system with the web server's DNS names as top-level directories under which each resource is stored, associated with the respective HTTP response headers in a separate file. *dtntcp* uses a compact binary representation[13] of the collected tree containing the absolute URI, resource size, and its contents which is then returned as a response bundle. The client unpacks the contents and stores the retrieved resources in its local cache from where they are used to satisfy subsequent requests by the web browser.

3) *HTTP resource lists* would require support by the server. Rather than replicating the web servers locally, we have added a specific resource list extension to the server-side *dtntcp*: For each requested URI, we have supplied a list of embedded resources (that we have determined immediately before the measurements by accessing the web page before using the we debugging tool *Charles*[14]) as local files from which the respective URI list is extracted on demand. While this saves one or two round-trips to the server, this minimal systematic error is considered acceptable for our evaluation below. The resulting URI list is fed into an instance of *puf* for efficient retrieval[15]; the results are stored, bundled, and forwarded as described above.

The main difference between modes (2) and (3) is that (2) operates in a probabilistic mode because not all the objects are known a-priori and prefetching may miss some resources (so that subsequent bundle requests are required) while (3) is deterministic as complete resource lists supplied for the origin server. We have not yet implemented true end-to-end interaction with HTTP-over-DTN integrated into a client and a server; this is subject to further study.

[12]Version 1.0.0, available from http://puf.sourceforge.net/. We use *puf* because our earlier experiments with *wget* did not provide satisfactory results due to its sequential operation. *puf* is invoked as *puf -ns -v -nc -pr++ -F -lc 4 -nb -xg -xd -xh * -U ""* followed by all headers received from the client (-xH) and the request URI.

[13]For simplicity of the implementation, we do not yet support the MIME-based MHTML.

[14]http://www.xk72.com/charles. This tool also provides lists of all resources accessed including statistics about HTTP request and response headers.

[15]Using *puf -ns -v -nc -F -lc 4 -nb -xd -xh * -U "" -i urilist-file*, again followed by the headers received from the client prefixed by -xH each.

6.1 Setup and Scenarios

We have built a measurement environment for the gateway-based approach (figure 3.3) as this is the expected initial situation to be dealt with. The setup was located at Universität Bremen and comprises three Intel 80686 PCs (A, B, and C) running Debian Linux 2.6.16.11, interconnected by a Gigabit Ethernet switch and connected to the Internet via a 1 Gbit/s access link. Host A runs Mozilla Firefox as the web browser used to initiate all requests, Hosts B and C each optionally run a *squid* web cache in proxy mode (i.e., with caching deactivated) as well as an instance of *dtnd* and *dtntcp*. Host B runs *tcpdump* to monitor the link between hosts A and B to determine the web retrieval times from the initial web request to the retrieval of the last resource associated with a web page. To measure the end user experience we have additionally used the Firefox plug-in *Fasterfox*[16] that also captures rendering times of the web browser and excludes retrieval time for resources not needed to display a web page (such as the icon file *favicon.ico*.[17] Host C runs *tcpdump* to monitor the interaction with remote web servers.

Figure 5 depicts the four measurement setups we have used to characterize the impact of introducing DTN. The focus is on the performance impact under ideal conditions, i.e., no loss and no congestion between a "mobile" node and the fixed Internet for two different access link delays. Analyzing this scenario allows to determine whether HTTP-over-DTN would be suitable as a baseline mode of operation for all future web interactions—as opposed to being chosen only when a user is mobile and finds herself in a challenged environment [4]. Measurement (0) provides the reference setup of plain HTTP over TCP; we have included two *squid* proxies to compensate for the connection splitting introduced by the other scenarios. (1) measures plain HTTP-over-TCP with each HTTP request and response encapsulated in a single DTN bundle. (2) is the prefetching approach where we use *puf* for recursive retrieval of the requested resources that are then pushed towards the *dtntcp* client and cached to satisfy subsequent requests. If *puf* fails to determine all resources, subsequent HTTP request bundles will be generated by the *dtntcp* client. Finally, (3) emulates the interactive retrieval of a web page's resources using hints supplied by the origin server (embedded in HTTP headers or by means of a separate dependency files) using a locally available file containing all relevant URIs (since modifications to the origin servers are not possible). Again, all retrieved resources are pushed from the *dtntcp* server to the client but, in this case, there are no missing resources expected and thus no subsequent retrieval request bundles will be necessary.

6.2 Measurement Results

We have carried out the above measurements three times for each web page and report on the mean retrieval values. The retrieval time is calculated from the SYN packet for the browser's request for the initial resource of the web page until the last TCP segment carrying data belonging to the respective web page and the page having been rendered.[18] The number of resources associated with each web page ("Obj") is counted based up on the number of GET requests as captured and reported in a separate run by *Charles*. The tool *ping* has been used on host A to determine the RTT between host A and each remote web server. In order to assess the influence of delay in communication paths, e.g., caused by congestion or short disconnections, we have also performed all measurements with an additional round-trip delay of 300 ms.

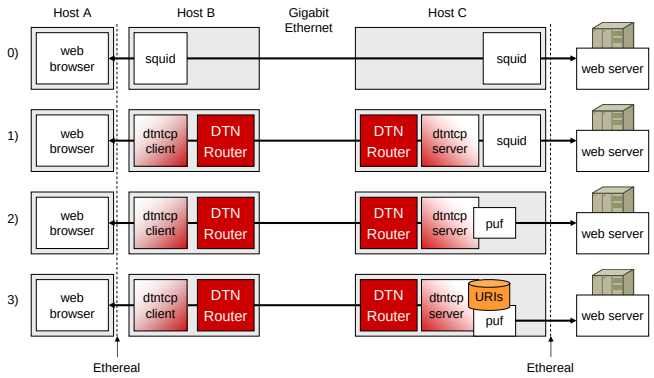

Figure 5: Measurement Scenarios

We have used a set of selected websites with different complexity (with respect to the number of embedded objects) and different geographic and network topological locations (Germany, US, Japan, Australia). From our twelve websites, we present those eight that have worked properly in all four scenarios during our repeated measurements; they are shown in table 1.[19]

Id	URL	Obj	RTT
TZI	http://www.dmn.tzi.org/	12	<1 ms
TS	http://www.tagesschau.de/	94	5 ms
WID	http://www.wide.ad.jp/	32	295 ms
APC	http://cocoon.apache.org/	29	28 ms
IETF	http://www.ietf.org/	8	107 ms
W3C	http://www.w3.org/	13	53 ms
News	http://www.theaustralian.news.com.au/	69	5 ms
ADL	http://www.adelaide.edu.au/	24	374 ms

Table 1: Overview of web pages and their characteristics

Table 2 summarizes the delays for the four aforementioned scenarios. Due to the small number of measurements and the variable environmental conditions, we can only provide general trends observed.[20] First of all, we observe that the retrieval latency for web pages between HTTP over TCP (0) and plain HTTP over DTN (1) is roughly in the same order of magnitude and, in most case, DTN only introduces a latency factor up to about six, with the exception of the TZI that exhibits an extremely short RTT so that any DTN-introduced latency gains particular weight. We expect this factor to be lower in a real-world environment with less perfect connectivity and when using optimized rather than prototype DTN and gateway implementations.

The results also show that prefetching (2) may provide probabilistic improvements, i.e., reduce retrieval latency, in cases where

[16] http://fasterfox.mozdev.org/

[17] Note that, due to varying local system load, the rendering time for the same web page may vary by up to 30total retrieval time; however, since we have observed varying response times due to server and network load this local deviation is considered within reasonable error bounds of our measurements.

[18] Using *Ethereal* and *Fasterfox* to support our evaluation.

[19] The contents of the other four websites could not be retrieved consistently using either prefetching or our URI list emulation; for those measurement results we obtained with these websites, they exhibited largely similar behavior to what we present below.

[20] While results obviously will vary between different websites due to their characteristics, even the evolution of a single website over time can lead to variations (as we observed in measurements of these twelve web sites over time). Therefore, any assessment has to be conservative at this point.

Id	(0)	(1)	(2)	(3)	(2):(0)	(3):(0)
TZI	0.5 s	41.9 s	4.1 s	3.7 s	7.9	7.1
TS	1.4 s	9.6 s	13.3 s	12.4 s	9.2	8.6
WID	4.1 s	8.3 s	8.0 s	5.8 s	2.0	1.4
APC	1.1 s	2.7 s	3.8 s	2.9 s	3.5	2.7
IETF	0.7 s	1.3 s	1.5 s	1.1 s	2.1	1.6
W3C	1.3 s	2.8 s	3.8 s	2.5 s	3.0	2.0
News	7.5 s	9.9 s	31.1 s	17.7 s	4.2	2.4
ADL	7.7 s	37.1 s	12.8 s	15.9 s	1.7	2.1

Table 2: Retrieval time for web pages with no artificial latency

Id	(0)	(1)	(2)	(3)	(2):(0)	(3):(0)
TZI	4.0 s	50.6 s	8.5 s	7.2 s	2.1	1.8
TS	19.4 s	43.8 s	36.1 s	57.1 s	1.9	2.9
WID	8.4 s	15.2 s	10.6 s	11.5 s	1.3	1.4
APC	9.9 s	16.7 s	19.0 s	11.1 s	1.9	1.1
IETF	1.8 s	3.6 s	3.8 s	3.6 s	2.1	2.0
W3C	4.9 s	10.7 s	11.5 s	8.0 s	2.4	1.7
News	17.1 s	29.3 s	51.7 s	34.0 s	3.0	2.0
ADL	18.8 s	52.9 s	29.0 s	23.4 s	1.6	1.3

Table 3: Retrieval time for web pages with 300 ms delay

prefetching succeeds in predicting most embedded objects (which is the case for at least 80 % of the embedded objects with most sites except for APC, TS, and News). We attribute variations for the other web pages primarily to varying server load and congestion effects. If prefetching does not succeed, the retrieval time naturally increases as repeated complex operations are required including forking *puf* for each resource not prefetched and collecting data from the file system afterwards; this explains the increased retrieval times for APC, TS, and News.

Using URI lists (3) provides this improvement almost deterministically across all web pages because no heuristics are needed to determine embedded links. Again, variable system load and congestion lead to a variation of the results (for example, the performance for WID has improved from (2) to (3) for the measurements without delay but has not in the 300 ms delay setup).

When comparing the measurements without delay (table 2) with those with 300 ms additional delay for the (wireless) access link (table 3), we observe that the relative performance of (2) prefetching and (3) URI lists compared to plain HTTP-over-TCP improves significantly with increasing delay. We observe that the relative latency—i.e., the latency ratio of scenario (2) to (0) and (3) to (0), columns "(2):(0)" and "(3):(0)" in the above tables—decreases from a maximum of nine down to only three when introducing artificial delay between the fixed and the mobile node. Most web pages take only about twice as long to retrieve despite the introduction of the non-optimized DTN infrastructure so that the performance "penalty" for using DTN becomes acceptable.

Overall, particularly deterministic advance retrieval of web page contents (3) as suggested above allows for complete collection of embedded resources and performs mostly only marginally worse (remaining in the same order of magnitude) compared to today's HTTP over TCP, in individual measurements even better, while improving robustness for mobile web access and enabling communication in particularly challenged environments in the first place.

However, as pointed out already in [4], the user experience suffers if bundling is applied because of the increased initial delay before any web contents can be displayed. While this is perfectly acceptable in disconnected environments, HTTP-over-DTN may benefit from a two-stage delivery process: returning the initially requested resource right away and then bundling all the embedded ones into one subsequent response—allowing for providing a quick first impression of the web page while preserving disconnection tolerance and efficiency.

7. CONCLUSION

In this paper, we have presented a protocol design and a system architecture for delay-tolerant access to web pages by running a slightly enhanced variant of HTTP on top of the DTNRG bundle protocol which enables web access for intermittently connected users. We have presented different complementary architectural options for incremental deployment and have shown that the achievable performance for HTTP-over-DTN given perfect connectivity is not much worse compared to plain HTTP over TCP operation. We believe that even the user experience may be maintained in connected environments if appropriate mechanisms are applied. This is important because it allows mobile users to choose HTTP-over-DTN as regular mode of operation rather than switching modes between mobile and fixed environments [4].

While these results look promising, more encompassing measurements (ideally using controlled environmental conditions) are needed to establish more statistically significant results. Furthermore, we seek to provide a complete DTN-over-HTTP gateway and proxy implementation in one component following the specification above and provide a minimal operational infrastructure for access to (selected) web sites. Our next steps are towards enabling end-to-end support for DTN, particularly the integration with a standard web browser so that co-located mobile gateways can be avoided. On the server side, we want to enable least partial support for resource collection within the realm of a single origin server (with the necessary consideration of DoS risks), while cross-server collection will remain up to the client or proxies on the path for the time being.

The most important conceptual issue to address in the future is a security architecture for web access via DTN that allows clients to also access secure contents on the web and thus substitute the reliance on TLS by an appropriate end-to-end mechanism at the application layer. This is essential as, with a constantly growing awareness of and need for security in communications, otherwise the most important (and an increasing number of) applications will not be accessible by mobile users.

8. ACKNOWLEDGMENTS

The authors would like to thank Christoph Dwertmann for helping with the measurements described in this paper.

9. REFERENCES

[1] Gosta Leijonhufvud, "Multi access networks and Always Best Connected, ABC," November 2001, MMC Workshop.

[2] Eva Gustafsson and Annika Jonsson, "Always Best Connected," *IEEE Wireless Communications*, vol. 10, no. 1, pp. 49–55, February 2003.

[3] Jörg Ott and Dirk Kutscher, "Why Seamless? Towards Exploiting WLAN-based Intermittent Connectivity on the Road," in *Proceedings of the TERENA Networking Conference, TNC 2004, Rhodes*, June 2004.

[4] Jörg Ott and Dirk Kutscher, "Applying DTN to Mobile Internet Access: An Experiment with HTTP," Technical Report TR-TZI-050701, Universität Bremen, July 2005.

[5] "Drive-thru Internet project," http://www.drive-thru-internet.org/.

[6] Jörg Ott and Dirk Kutscher, "Drive-thru Internet: IEEE 802.11b for „Automobile" Users," in *Proceedings of the IEEE Infocom 2004 Conference, Hong Kong*, March 2004.

[7] Jörg Ott and Dirk Kutscher, "The "Drive-thru" Architecture: WLAN-based Internet Access on the Road," in *Proceedings of the IEEE Semiannual Vehicular Technology Conference May 2004, Milan*, May 2004.

[8] Jörg Ott and Dirk Kutscher, "Exploiting Regular Hot-Spots for Drive-thru Internet," in *Proceedings of KiVS 2005, Kaiserslautern, Germany*, March 2005.

[9] Jörg Ott, Dirk Kutscher, and Mark Koch, "Towards Automated Authentication for Mobile Users in WLAN Hot-Spots," in *Proceedings of VTC Fall 2005*, September 2005.

[10] Jörg Ott and Dirk Kutscher, "A Disconnection-Tolerant Transport for Drive-thru Internet Environments," in *Proceedings of the IEEE Infocom 2005 Conference, Miami*, March 2005.

[11] "FleetNet project," http://www.et2.tu-harburg.de/fleetnet/.

[12] "Network on Wheels project," http://www.network-on-wheels.de/.

[13] Marc Bechler, Walter J. Franz, and Lars Wolf, "Mobile Internet Access in FleetNet," in *13. Fachtagung Kommunikation in verteilten Systemen, Leipzig, Germany*, April 2003.

[14] "IST OverDRIVE project," http://www.ist-overdrive.org/.

[15] M. Zitterbart, K. Weniger, O. Stanze, S. Aust, M. Frank, M. Gerharz, R. Gloger, C. Görg, I. Gruber, S. Hischke, P. James, H. Li, C. Pampu, C. de Waal, W. Weiß, D. Westhoff, J. Wu, D. Yu, and X. Xu, "IPonAir – Drahtloses Internet des nächsten Generation," PIK, Vol 26, No 4, October 2003.

[16] Pablo Rodriguez, Rajiv Chakravorty, Julian Chesterfield, Ian Pratty, and Suman Banerjee, "MAR: A Commuter Router Infrastructure for the Mobile Internet," in *Proceedings of the ACM Mobile Systems, Applications and Services Conference (ACM Mobisys 2004)*, June 2004.

[17] Adeel Baig, Mahbub Hassan, and Lavy Libman, "Prediction-based Recovery from Link Outages in On-Board Mobile Communication Networks," in *Proceeding of IEEE Globecom 2004*, December 2004.

[18] Yun Mao, Björn Knutsson, Honghui Lu, and Jonathan Smith, "DHARMA: Distributed Home Agent for Robust Mobile Access," in *Proceedings of the IEEE Infocom 2005 Conference, Miami*, March 2005.

[19] "Mindstream project," http://mindstream.watsmore.net/.

[20] Venkata N. Padmanabhan and Jeffrey C. Mogul, "Using Predictive Prefetching to Improve World-Wide Web Latency," Proceedings of the ACM SIGCOMM '96 Conference, 1996.

[21] Zhimei Jiang and Leonard Kleinrock, "Prefetching links on the WWW," in *ICC (1)*, 1997, pp. 483–489.

[22] B. Davison, "Assertion: Prefetching with GET is not good," 2001.

[23] Nils Seifert, "A Pragmatic Approach for "d-burdened" Internet Services," in *Presentation at the Dagstuhl Seminar on Disruption Tolerant Networking*, April 2005.

[24] Kevin Fall, "A Delay-Tolerant Network Architecture for Challenged Internets," Proceedings of ACM SIGCOMM 2003, Computer Communications Review, Vol 33, No 4, August 2003.

[25] Keith L. Scott and Scott C. Burleigh, "Bundle Protocol Specification," Internet Draft draft-irtf-dtnrg-bundle-spec-05.txt, Work in progress, May 2006.

[26] Sushant Jain, Kevin Fall, and Rabin Patra, "Routing in Delay Tolerant Networks," Proceedings of the ACM SIGCOMM 2004 Conference, Portland, OR, USA, 2004.

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

[28] Anders Lindgren, Avri Doria, and Olov Schelen, "Probabilistic routing in intermittently connected networks," in *The First International Workshop on Service Assurance with Partial and Intermittent Resources (SAPIR)*, 2004.

[29] Jörg Widmer and Jean-Yves Le Boudec, "Network Coding for Efficient Communication in Extreme Networks," in *ACM SIGCOMM Workshop on Delay-Tolerant Networking (WDTN)*, 2005.

[30] Yong Wang, Sushant Jain, Margaret Martonosi, and Kevin Fall, "Erasure Coding Based Routing for Opportunistic Networks," in *ACM SIGCOMM Workshop on Delay-Tolerant Networking (WDTN)*, 2005.

[31] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations," RFC 3135, June 2001.

[32] Alex C. Snoeren, "A Session-based Architecture for Internet Mobility," PhD Dissertation, MIT, 2002.

[33] Simon Schütz, Lars Eggert, Stefan Schmid, and Marcus Brunner, "Protocol enhancements for intermittently connected hosts," *ACM Computer Communications Review*, vol. 35, no. 3, pp. 5–18, July 2005.

[34] N. Borenstein and N. Freed, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies.," RFC 1521, September 1993.

[35] J. Palme, A. Hopmann, and N. Shelness, "MIME Encapsulation of Aggregate Documents, such as HTML (MHTML)," RFC 2557, March 1999.

[36] Henry Chang, Carl Tait, Norman Cohen, Moshe Shapiro, Steve Mastrianni, Rick Floyd, Barron Housel, and David Lindquist, "Web browsing in a wireless environment: Disconnected and asynchronous operation in artour web express," Proceedings of ACM MOBICOM 97, Budapest, Hungary, 1997.

[37] "DTN Reference Implementation," http://www.dtnrg.org/wiki/Code, May 2006.