# MADServer: A Server Architecture for Mobile Advanced Delivery

Agoston Petz[1], Anders Lindgren[2], Pan Hui[3], and Christine Julien[1]
[1]The University of Texas at Austin, [2]Swedish Institute of Computer Science,
[3]Deutsche Telekom Labs

agoston@utexas.edu, andersl@sics.se, Pan.Hui@telekom.de,
c.julien@mail.utexas.edu

TR-ARiSE-2012-007




ARiSE
Advanced Research in Software Engineering

# MADServer: A Server Architecture for Mobile Advanced Delivery

Agoston Petz[1], Anders Lindgren[2], Pan Hui[3], and Christine Julien[1]
[1]The University of Texas at Austin, [2]Swedish Institute of Computer Science, [3]Deutsche Telekom Labs
agoston@utexas.edu, andersl@sics.se, Pan.Hui@telekom.de,
c.julien@mail.utexas.edu

## ABSTRACT

Rapid increases in cellular data traffic demand creative alternative delivery vectors for data. Despite the conceptual attractiveness of mobile data offloading, no concrete web server architectures integrate intelligent offloading in a production-ready and easily deployable manner without relying on vast infrastructural changes to carriers' networks. Delay-tolerant networking technology offers the means to do just this. We introduce MADServer, a novel DTN-based architecture for mobile data offloading that splits web content among multiple independent delivery vectors based on user and data context. It enables intelligent data offloading, caching, and querying solutions which can be incorporated in a manner that still satisfies user expectations for timely delivery. At the same time, it allows for users who have poor or expensive connections to the cellular network to leverage multi-hop opportunistic routing to send and receive data. We also present a preliminary implementation of MADServer and provide real-world performance evaluations.

## Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*wireless communication, network communications, store and forward networks*

## General Terms

Design

## Keywords

Cellular data offloading, Mobile advanced delivery, Delay tolerant cache

## 1. INTRODUCTION

The recent explosion in cellular data traffic (due largely to the popularity of smartphones and flat-rate data subscriptions) is generating capacity problems for operators, both in the wireless spectrum and in cellular access networks. We develop a novel DTN-based web server architecture that alleviates these problems and provides a better end-user experience. Our architecture will: (*i*) offload "heavy" content transfers from the cellular network; (*ii*) make use of client mobility patterns and predictions to find suitable network resources and enable advanced delivery of content; (*iii*) provide a novel framework for using user context and data properties in making offloading decisions; and (*iv*) provide a simple way for developers to prioritize content to show which can be easily offloaded and which is time-critical. In this way, we extend DTN-concepts to cellular networks in which nodes are generally considered well-connected, but in which mobile nodes can benefit greatly from opportunistic content sharing based on DTN principles.

Recent work has explored the potential of data offloading from cellular networks to, for example WiFi hotspots, and offloading techniques can address capacity problems plaguing cellular networks [16, 3]. If the data is not time-critical, further savings can be had through deferred transmissions and the use of delay tolerant networking techniques [17]. Advances in content-centric networking [2] allow for the caching of popular content at base stations and in mobile nodes. This can reduce the load on shared access networks, which can often be the bottleneck [10]. However, while data offloading has been viewed from theoretical and conceptual perspectives, there is little existing work on supporting these techniques in real networks. Existing network architectures are not sufficiently aware of network and data context to make intelligent offloading decisions, or demand infrastructure changes in the carriers' networks which can be expensive and time-consuming to accomplish. MADSever builds upon existing work and enables intelligent content offoading without relying on infrastructure changes.

MADServer is aware of the mobility of content consumers and the context of the network and data content. Specifically, our architecture makes it possible to send different pieces of web content over different network technologies, enabling offloading of "heavy" content from the cellular networks, in particular if such content is not time critical. Mobile devices can recombine the pieces before providing them to the user. This decoupling of web content from user requests allows for *advanced delivery* of content to places where a user will be instead of where a user currently is. In today's highly mobile environment, this pre-caching takes full advantage of offloading opportunities. By integrating this with a DTN query mechanis, the needs of multiple users can potentially be served by a single transfer over the access


*CHANTS'12,* August 22, 2012, Istanbul, Turkey.

network and users can also collaboratively share content locally to satisfy requests. The following are some motivating example situations.

**Mobile Video-on-Demand.** Services such as YouTube and those offered by local TV channels have become immensely popular. With the advent of smartphones, users also want to use these services on their mobile devices, straining cellular networks. Potential local similarities in requests for this content (commuters on the same train all wanting to catch up on last night's episode of a popular TV series or watch the latest viral video) generate great potential gains for caching and data offloading.

**Events with large crowds.** Some events entail crowds in areas where networks are dimensioned for fewer people. Such events include big outdoor sporting events (such as marathons, which are spread over a large area, making it hard to deploy extra capacity at a particular location), or the recent royal weddings in Sweden and the UK, where large crowds gathered to see the newlyweds but still wanted to be able to watch the wedding on their mobile devices. Many in the crowd will have similar interests and request the same data; local caching and opportunistic exchange of data has great potential benefits.

## 2. MADSERVER ARCHITECTURE

MADServer (Mobile Advanced Delivery Server) enables context-based data offloading without requiring new software at cell towers and with minimal changes at clients and servers; this allows for quick adoption of data offloading and eases the burden on network programmers. We split web content into two conceptual pieces: large content (pictures, streaming video, music, etc.), and the rest of the HTML frame, which itself can contain smaller content items (news tickers, feeds, etc.). We also distinguish two delivery vectors, *3G* (which can really be any cellular communication technology) and the *content offloading vector*, which can take many forms, although it will require some local content-cache and will generally rely on WiFi for its "last-hop" delivery.

We transfer small content over 3G and offload large content when beneficial and within delay constraints. Active sessions need not be terminated and restarted when switching technologies, which disrupts user experience and leads to re-transferring partially received pages. Instead, when an alternate delivery vector is available, the web server can offload the bulk of communication but still provide a seamless session with minimal 3G usage.

Modern web architectures typically consist of a combination of *a)* a high-performance web server written in C/C++ with *b)* web applications often written in object-oriented languages such as Python. Our architecture places a lightweight "middleware" layer between these components to allow for expressive content offloading based on user and data context; no additional software is needed at the cell towers, and web developers need not contend with an unfamiliar and complex new system. Figure 1 shows the high-level architecture. Client requests are sent over 3G, although large requests like file uploads could be offloaded. The server response is split into two, *Response′* and *Response″* to be sent over 3G and the offloading vector, respectively. *Response′* contains HTML frames to be served in the traditional way with the content tags re-written to point to future offloaded locations; *Response″* contains the content with its meta-data. The decision of when to offload content and which

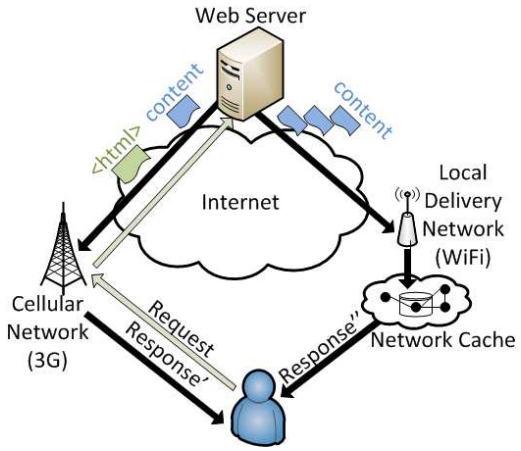


**Figure 1: High-Level MADServer Architecture**

delivery vector to use depends on the context of the user and the context of the data.

### 2.1 Context and its Uses

With respect to acquiring and using context for intelligent data offloading, there are two key issues: *(i)* what context to acquire and *(ii)* how to distribute and respond to context. There are many mechanisms for sensing, storing, and aggregating context; we do not develop new mechanisms but instead rely on existing work. The key question asks what types of context information are particularly useful for this scenario. For example, when a network cache relies on opportunistic node contacts, context may include knowledge about contact patterns (e.g., duration of contacts or number of unique contacts). Alternatively, when applications attempt to use offloading to benefit from multiple users requesting the same content, context may include aggregate activities of local groups of nodes. We give additional concrete examples in Table 1. To enable a node to acquire its context, we rely on existing services and techniques [21, 23, 26, 25] that can acquire information about a device, its local environment, and its network links; to enable nodes to acquire aggregate context information, we rely on recent work in assessing the context of a dynamic group [14] that can reason about similarities among users, the network neighborhood, and social graphs.

In addition to these types of context that give information about users' situations, we can also use *data context*, which is independent of the requester and the network. This includes the data's creation time, it's time-to-live (e.g., stock quotes need to be delivered quickly before they become irrelevant), the size of the data, priority labels, etc. Data context must be provided by the web application developer (or inferred by the middleware). Generally speaking, it is associated directly with the chunks of data as meta-data.

It is up to the client to determine which context to acquire and share with the server. This will largely depend on the client's context acquisition capabilities, the requirements of a request. The client has compelete control over what context it shares with a given web service, and what it does not. This is a significant benefit since it ensures that the system does not violate a user's privacy requirements. The

| Type of Context | Examples | Usage |
|---|---|---|
| System Context | battery level, charging status, CPU load, free memory | selectively enable/disable a client's participation in mobile caching and ad-hoc content sharing |
| Network Context | network type, roaming status, calling status, WiFi state | can influence sharing patterns; enables content *prediction*, which is required for advanced delivery |
| Location Context | GPS location, speed, heading | can provide common mobility patterns for prediction; correlated with cache availability |
| Aggregate Context | common activities, social connections | servers can learn popular locations for caches and popular data to cache |
| Data Context | creation time, time-to-live, data size, priority labels | influence which data can be off-loaded depending on the network cache capabilities |

**Table 1: Potentially Useful Concrete Context Metrics**

use of context is not entirely new in web services; existing web services that provide tailored services such as streaming bitrate adaptation have a similar reliance on context [5]. As in all similar uses of context, the client and server must agree *a priori* on the language used to describe context.

The context that a user acquires about her situation must be sent to the web server to enable intelligent data offloading. A client's context can be piggybacked on the client's (HTML) request. Our architecture leverages existing approaches for succinctly summarizing context [14] to prevent the transmission of context from overburdening 3G connections. Aggregated context information about a group of nodes can be similarly shared. Alternatively, context can be shared through the network cache and back to the server (through the reverse of the process of delivering *Response''*). Again, we do not construct new context acquisition and distribution mechanisms but instead integrate existing approaches with our novel architecture for jointly delivering content directly and through offloading.

## 2.2 Content Offloading Vector

In our architecture, time-critical content is delivered in the traditional manner across the cellular network so the user experience in not degraded, but less critical content can selectively be pushed across an alternative delivery mechanism, especially when the associated delivery delay is within tolerable bounds [3]. Our vision combines *delay-tolerant* networking principles with *content-based* networking, and relies on asynchronous, opportunistic communication.

**Content-Based Networking.** In content-based networking, messages are addressed by application-specified *content* instead of unique addresses [15, 29, 2], increasing expressiveness and flexibility, decreasing the time to discover desired content, and leveraging locality of content consumers. These approaches have been explored in traditional Internet scenarios [7, 11] and in mobile computing [1, 12, 15]. Our architecture can rely on content-based networking to locate and share "heavy" web content through a network cache that spans a local delivery network containing WiFi access points and mobile nodes.

**Publish/Subscribe with BPQ.** In publish-subscribe systems, senders *publish* messages with topic labels, and these are distributed through the network according to *subscriptions*. In general, subscriptions are distributed to the entire network to form a routing structure; however, maintaining this routing structure in the face of topological changes [6, 13, 20, 28] is not always advisable in a mobile network.

The Bundle Protocol [27] is the de facto standard session protocol for DTNs, and the Bundle Protocol Query (BPQ) extension block [8] allows for intelligent in-network content caching, in essence providing a publish/subscribe system over DTNs. BPQ queries are sent towards the original content publisher, who responds to it. BPQ-enabled nodes on the path back to the requestor will, depending on space availability and local policies, cache the content. If another node makes a query that can be satisfied by the same content, the request can be served by these intermediate caching sites directly instead of forwarding the request all the way to the publisher.

**Global Virtual Data Structures.** A useful abstraction for mobile users represents dynamic distributed data as though it is accessible through a locally available data structure; such *global virtual data structures* fill in communication as needed below the data structure abstraction [24]. A canonical example is the distributed tuple space [9, 4], which has enjoyed significant use in parallel, distributed, and mobile computing. Similar database abstractions have been extended to a variety of domains, including MANETs [18] and sensor networks [19].

Although the MADServer architecture is itself independent of the particular content offloading vector, our implementation uses the DTN2 with the BPQ extension.

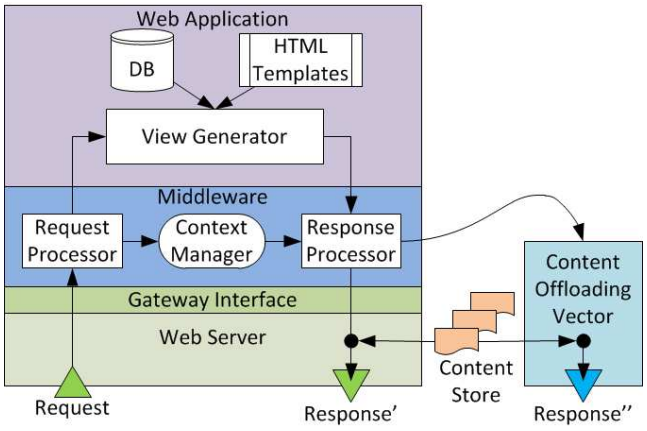


**Figure 2: Server-Side Architecture**

## 2.3 Server-Side Architecture

Our server-side architecture, shown in Figure 2, has three main components, a *Request Processor*, a *Context Manager*, and a *Response Processor*, all of which live in a middleware layer directly below the web application. Many production-capable web application frameworks such as Django[1] already implement middleware layers well-suited for this architecture. The client inserts its context into its HTML requests. The *Request Processor* looks for specific context tags, strips them from the request, and sends the context information to the *Context Manager*, which tracks each user by his 3G IP address. Otherwise, requests proceed as normal, with no changes required to the web application itself. Once the HTML response is generated, the *Response Processor* rewrites the response according to the pre-defined rules and the user context provided by the *Context Manager*, possibly removing some of the content and content links and sending that data over the offloading delivery vector. Consider the following two content items:

1. <a href="http://**server-location.url**/media/largeVideoFile.flv" id=embedStreamPlayer>
2. <a href="http://**server-location.url**/images/largeImageFile.jpg">

Normally, both content items would be served by the web server over the 3G network. The *Response Processor* could rewrite the above response to:

1. <a href="http://**localhost/tmp-cache/**largeVideoFile.flv" id= embedLocalStreamPlayer>
2. <a href="http://**localhost/tmp-cache/**largeImageFile.jpg">
3. < LOOKUP CONTENT "dtn://*/largeImageFile.jpg", "dtn://*/largeVideoFile.flv" USING DTN_BPQ>

The largeVideoFile and largeImageFile urls now point to the local client cache, and the embedded video player changed to the local streaming service. The client looks for both content items in the local DTN network cache using the BPQ extension, and, once the content arrives, the client can stream the video from its own local cache.

## 2.4 Client-Side Architecture

In MADServer, clients must provide context to the server. This requires two elements on clients, a *context aggregator*, and a *MADServer browser plugin*. The *context aggregator* enumerates available resources and services, tracks offloading delivery vector availability, and calculates future offloading possibilities. The resulting context is then sent to the servers via a browser plugin that automatically detects servers that have data offloading capabilities and inserts context information from the *context aggregator* into HTTP requests. In our implementation we simply offload data using the bundle protocol when a WiFi connection is available; this is naïve and could lead to performance degradation [3], but it is sufficient for a proof of concept. More centralized context aggregation and processing might be desired, for example provided by the cellular carriers. The MADServer architecture is still valid since such context can easily be sent to the clients, which can then send it on to the servers. Data privacy is a natural advantage of client-controlled context—the user retains full control over what context she shares and with whom.

[1] `https://www.djangoproject.com/`

# 3. PROTOTYPE AND EVALUATION

This section describes our MADServer proof-of-concept implementation and the evaluations performed using it.

**Web Server and Interface.** We use Apache[2] (since it is the open-source standard for deployment web servers) interfaced with our web application using Python Web Server Gateway Interface (WSGI).[3]

**Middleware.** We implemented the middleware layer and context manager in the Django Web Framework.[4] In our preliminary implementation of the *Context Manager*, the client encodes three pieces of context in each HTTP request:

$$\{\textbf{Offload}?, \textbf{DestIPAddr}, \textbf{DTNEndpointID}\}.$$

When the context manager on the client determines that mobile advanced delivery of content is beneficial, it asks the server to start offloading the content and provides a destination IP address where the content should be sent. Note that this need only be a destination running a bundle protocol router that can accept and forward bundles, and can thus act as an in-network content cache. The client also provides its globally unique bundle protocol endpoint identifier. The server encapsulates the requested content items in their own bundle, which is addressed to the client's DTNEndpointID and forwarded over the Internet to the provided IPAddr. Once the bundle is transferred to the destination, hop-to-hop opportunistic routing will deliver the content. We do not implement a predictive location context service on the client; instead for the evaluation we loaded this context *a priori*. However, recent work in energy efficient cellular phone-based predictive location awareness could provide this information easily [22].

**Web Application.** For the application, we used a fully-functional social networking website built in the Pinax rapid web application development framework[5] with a MySQL database back-end and static file system for the content (pictures, video, etc.).

**Content Offloading Service.** We used the DTN2 Reference Implementation[6] of the Bundle Protocol to implement the data offloading delivery vector. This allows us to address content independent of a user's current IP address (instead using its globally unique endpoint ID) and to pre-cache content in places where a node might visit as long as there is at least one host there who implements the bundle protocol and can accept and forward bundles. If there is a network of such nodes, then content is disseminated to all nodes according to the the DTN2 forwarding rules. When the user eventually comes into contact with a node caching its content, the content is delivered.

## 3.1 3G-Only vs. Offloading

Content offloading not only frees expensive cellular bandwidth, it can deliver content faster even without pre-caching. In this first experiment, we issued 50 requests for three different web pages with minimal HTML frames containing images of sizes {512 KB, 1 MB, and 5 MB}. The client is a linux-based laptop with a USB 3G Modem and WiFi card located in Europe, and the server is located in the central

[2] `http://apache.org/`
[3] `http://wsgi.org/wsgi/`
[4] `https://www.djangoproject.com/`
[5] `http://pinaxproject.com/`
[6] `http://www.dtnrg.org`

United States. We measured average delivery latency using only 3G connectivity and with content offloading using our bundle protocol based offloading vector—the last hop link of which was over 802.11b via a WiFi access point. Figure 3 shows the latency results with their standard deviation for two different WiFi access points, one with a relatively poor signal strength and many users and one with good signal strength and very few users. WiFi is not necessarily faster than 3G [3], although this strengthens the argument for user context-based data offloading: if the cellular connection is currently expensive (e.g., the user is roaming), the extra latency may well be worth money saved (of course, not all WiFi access might be free of charge either, but it is likely that it will be cheaper than 3G access, and any cost associated with WiFi usage is another context parameter to take into consideration). Conversely, if the data is of high priority, using 3G (depending on the available WiFi bandwidth) may be better.

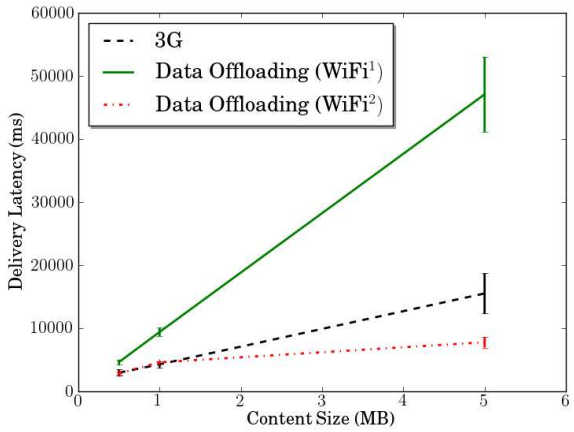


**Figure 3: Impact of offloading on response time**

## 3.2 3G Bandwidth Savings

Figure 4 shows the 3G bandwidth usage during regular operation and with data offloading. The client makes three requests for a page containing a 16MB video file. In the 3G-only case, the video is streamed over the 3G connection. In the data-offloading case, the video is bundled and sent to the client using our DTN2 content delivery vector; the final hop is over 802.11g from a WiFi access point. The client receives the bundle, inserts it into its local cache, and the video is "streamed" locally. As the results show, the 3G savings are significant (three orders of magnitude). Furthermore, the latency of the WiFi connection setup and bundle protocol client registration handshake is only a few seconds and is not detrimental to the user experience. The local video file stream can be started as soon as the video file starts arriving on the client; there is no need to wait for the whole file to arrive in order to start streaming from the file descriptor. The combined request-to-video-start latency is thus only a few seconds more than when using only 3G.

## 3.3 Advanced Delivery

Our implementation also enables mobile advanced delivery of content. If the client's context manager determines that a node is about to connect to a content cache, it can

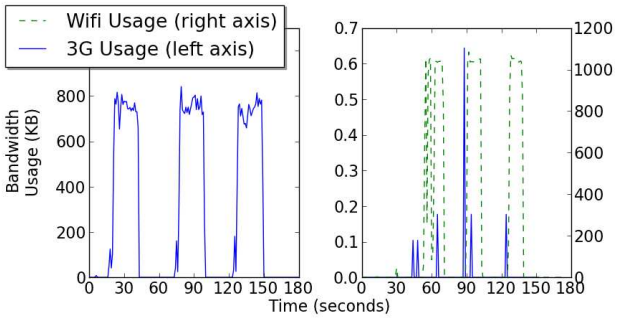


**Figure 4: Bandwidth (3G-only vs. 3G + Offloading)**

request that current and future content requests be serviced by the cache instead of downloaded over 3G. Figure 5 shows the results of an experiment in which a user makes 20 independent requests for web pages each including a 5MB content item. The left-hand graph plots the 3G usage over time if the content is requested and delivered via 3G only. The right hand plot shows a scenario where after the first five requests, the user determines that it will soon connect to a mobile advanced delivery cache (implemented by DTN2), at which point all content should be forwarded to the cache. The server responds to two of the requests over 3G regardless because the data is *high priority* according to its meta-data tags. The rest is forwarded to the content-cache and served to the client when it connects over 802.11b. When the client disconnects from the content cache, it sends a context update, and the remaining five requests are serviced over 3G. In this experiment, the *user context* was predetermined and provided ahead of the experiment. Not only does the MADServer architecture save 3G bandwidth, in this case reducing the 3G load from 108 MB to 60.7 MB given only a 51 second connection to a content-cache, but it is able to deliver all twenty content items in almost half the time.

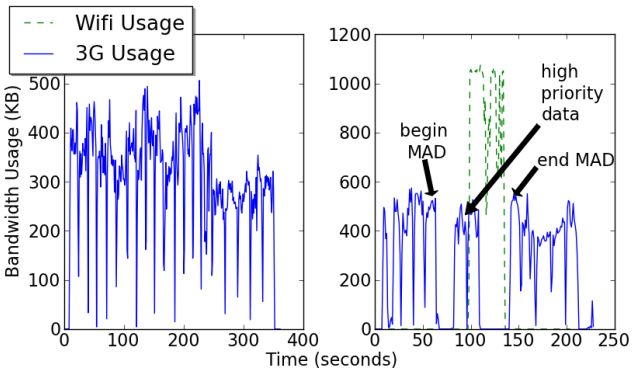


**Figure 5: 3G-Only vs. Mobile Advanced Delivery**

# 4. CONCLUSION

MADServer, a web server architecture for mobile advanced delivery, performs adaptive content offloading by splitting

web responses into pieces based on user and data context, and delivering them to the client using different *delivery vectors*. Content offloading addresses critical capacity problems in cellular data networks but must be done in context sensitive ways so as not to deteriorate the user experience. Although there are many challenges in aggregating, interpreting and responding to context, our architecture is a first step in integrating context metrics with data offloading decisions, and our implementation provides tangible results of the benefits.

Prior work in data offloading has been mostly theoretical or conceptual, and this work paves the way for integrating adaptive offloading with production web services without the need for drastic systems changes.

MADServer is general enough to be independent of the particular *content offloading vector* to be used, but we have demonstrated one possibility based on delay-tolerant session and routing protocols. It is also flexible enough to incorporate research in network context and content delivery networks without the need for web developers to change their applications to utilize them. MADServer is a novel web architecture approach that brings together fundamental recent advances in content-based communication and context-awareness to provide revolutionary capabilities for web content delivered to mobile clients. The initial work in this paper lays the groundwork for MADServer and exposes key research questions in applying context in this domain and appropriately selecting expressive content delivery vectors.

## 5. REFERENCES


[1] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *Proc. of SOSP*, 1999.

[2] B. Ahlgren and V. Vercellone. Networking of information âĂŞ an information-centric approach to the network of the future. In *ETSI Future Network Technologies Workshop*, Mar. 2010.

[3] A. Balasubramanian, R. Mahajan, and A. Venkataramani. Augmenting mobile 3g using wifi. In *Proceedings of MobiSys*, 2010.

[4] D. Balzarotti, P. Costa, and G. Picco. The LighTS tuple space framework and its customization for context-aware applications. *J. or Web Intelligence and Agent Systems*, 5(2):215–231, 2007.

[5] A. Begen, T. Akgul, and M. Baugher. Watching video over the web: Part 1: Streaming protocols. *IEEE Internet Computing*, 15:54–63, 2011.

[6] M. Caparuscio, A. Carzaniga, and A. Wolf. Design and evaluation of a support service for mobile, wireless publish/subscribe applications. *IEEE Trans. on Software Eng.*, 29(12):1059–1071, Dec. 2003.

[7] A. Carzaniga, M. J. Rutherford, and A. L. Wolf. A routing scheme for content-based networking. In *Proc. of INFOCOM*, 2004.

[8] S. Farrell, A. Lynch, D. Kutscher, and A. Lindgren. Bundle protocol query extension block. Technical Report draft-farrell-dtnrg-bpq-01, IRTF, Mar. 2012.

[9] D. Gelernter. Generative communication in Linda. *ACM Trans. on Programming Languages and Systems*, 7(1):80–112, 1985.

[10] D. Giustiniano, E. Goma, A. Lopez Toledo, I. Dangerfield, J. Morillo, and P. Rodriguez. Fair wlan backhaul aggregation. In *Proc. of MobiCom*, 2010.

[11] M. Gritter and D. R. Cheriton. An architecture for content routing support in the internet. In *Proc. of USITS*, 2001.

[12] I. Gruber, R. Schollmeier, and W. Kellerer. Performance evaluation of the mobile peer-to-peer service. In *Proc. of CCGrid*, 2004.

[13] Y. Huang and H. Garcia-Molina. Publish/subscribe in a mobile environment. *Wireless Networks*, 10(6):643–652, Nov. 2004.

[14] C. Julien. The context of coordinating groups in dynamic mobile networks. In *Proc. of COORDINATION*, 2011.

[15] C. Julien and M. Venkataraman. Cross-layer discovery and routing in reconfigurable wireless networks. In *Proc. of MASS*, 2006.

[16] K. Lee, I. Rhee, J. Lee, S. Chong, and Y. Yi. Mobile data offloading: how much can wifi deliver? In *Proceedings of Co-NEXT '10*, 2010.

[17] A. Lindgren. Efficient content-distribution in a hybrid opportunistic network. In *CCNC*, 2011.

[18] J. Luo, P. Eugster, and J.-P. Hubaux. Pilot: Probabilistic lightweight group communication system for ad hoc networks. *IEEE Trans. on Mobile Computing*, 3(2):164–179, 2004.

[19] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. Tinydb: An acquisitional query processing system for sensor networks. volume 30, pages 122–173. ACM, 2005.

[20] L. Mottola, G. Cugola, and G. Picco. A self-repairing tree overlay enabling content-based routing in mobile ad hoc networks. *IEEE Trans. on Mobile Computing*, 7(8):946–960, 2008.

[21] A. Nicholson and B. Noble. BreadCrumbs: Forecasting mobile connectivity. In *Proc. of MobiCom*, 2008.

[22] M. Papandrea. A smartphone-based energy efficient and intelligent multi-technology system for localization and movement prediction. In *PerCom Workshops*, pages 554–555, 2012.

[23] A. Petz, T. Jun, N. Roy, C.-L. Fok, and C. Julien. Passive network-awareness for dynamic resource-constrained networks. In *Proc. of DAIS*, 2011.

[24] G. Picco, A. Murphy, and G.-C. Roman. On global virtual data structures. In *Process Coordination and Ubiquitous Computing*, pages 11–29, 2002.

[25] A. Rahmati and L. Zhong. Context-for-wireless: context-sensitive energy-efficient wireless data transfer. In *Proceedings of MobiSys*, 2007.

[26] D. Salber, A. K. Dey, and G. D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *CHI*, 1999.

[27] K. L. Scott and S. Burleigh. Bundle protocol specification. Technical Report RFC5050, IRTF, Nov. 2007.

[28] E. Yoneki and J. Bacon. An adaptive approach to content-based subscription in mobile ad hoc networks. In *Proc. of MP2P*, 2004.

[29] H. Zhou and S. Singh. Content based multicast (CBM) in ad hoc networks. In *Proc. of Mobicom*, 2000.