# A Policy-Oriented Architecture for Opportunistic Communication on Multiple Wireless Networks

Aaditeshwar Seth[1], Matei Zaharia[1], Srinivasan Keshav[1], and Supratik Bhattacharya[2]

[1]School of Computer Science
University of Waterloo, ON, Canada

[2]Sprint Advanced Technology Lab
Burlingame, CA, USA

***Abstract*— Today's mobile devices are already equipped with multiple wireless interfaces that differ in data rates, power consumption, monetary cost, and coverage areas. Previous research has shown that intelligent policy-based switching between wireless interfaces can obtain better performance than using a single interface [29]. We build upon this pioneering work to add the notion of user-defined policies that operate at the *session layer*, unlike the network- and transport-layer policies of past work. Session-layer policy definitions allow us to include a delay component (such as a deadline for data transfer) at a timescale of minutes and hours. This helps us distinguish between online and offline applications, and allows us a high degree of flexibility in network selection. For example, we can defer data transfer on currently available networks if they are too costly, and wait until a low cost network (such as a free WiFi hotsopt) becomes available for opportunistic use in the future.**

**However, actually using multiple interfaces, which requires sessions to be maintained across interface switches, disconnections, and device shutdowns, is complex. Consequently, to be useful in practice, we believe that application writers need to be shielded from these details, while still exercising fine-grained control over network usage policies. In this paper, we describe a system that allows applications running on a mobile device to seamlessly exploit multiple heterogeneous wireless networks based on user-defined session-layer policies. We have designed, implemented, and evaluated the performance of an Opportunistic Connection Management Protocol (OCMP) that allows such applications to opportunistically communicate on multiple network interfaces, switch across interfaces, remain disconnected or powered off for arbitrarily long periods, and interoperate with legacy applications and servers. The implementation is in J2ME so that it can be run on any Java-based mobile device. Extensive policy control for interface selection is also provided to applications, along with a simple API for application developers. We explain the design, architecture, and implementation of OCMP, and illustrate its benefits through field experiments. Our results are encouraging and suggest that policy support for multi-network opportunistic communication is both achievable in current systems, and of significant practical value.**

## I. Introduction

### A. The opportunity of multi-interface devices

The past few years have seen an explosive growth in the number of mobile devices such as cellphones, PDAs, and laptop computers. These devices have multiple NICs (network interfaces) and can use a variety of wireless access technologies ranging from wide-area technologies such as GPRS, EDGE, CDMA 1xRTT, EV-DO, and WiMax, to local-area technologies such as 802.11 and short-range technologies such as Bluetooth and UWB. These wireless technologies differ from each other on a number of parameters, summarized in Table 1 [1], [2], [43].

In general, short-distance wireless technologies are better than long-distance technologies on parameters of data rate, power consumption per bit, and monetary cost:

- *Data rate*: Any wireless access technology must make a difficult tradeoff between the coverage of an access point and the capacity available to a user in that access point's coverage area. Thus, long distance technologies such as EDGE have a range of a couple of kilometers, but support a lower data rate than short distance technologies such as WiFi, that have coverage areas of only a couple of hundred meters.
- *Power consumption*: The average power consumption of a cellular radio such as EvDO is of the same order as that of a WiFi radio, despite the larger communication distances, because cellular technologies use protocols for fine-grained open- and closed-loop power control that enable efficient channel management. However, due to the lower data rate of EvDO, the power consumption per bit of WiFi is much lower than EvDO.
- *Monetary cost*: The monetary cost of using an access network depends on the pricing policy of the network provider. It is practically zero in the case of private Bluetooth networks or free public access WiFi hot-spots operating on unlicensed spectrum, but can be very expensive (up to $25/MB) for cellular data access plans.

Despite their superior performance, short distance technologies are very limited in their coverage. For example, only a few CDMA base stations are sufficient to provide coverage to a large geographical area, but almost a hundred times the number of WiFi access points are required to cover the same area: this greatly increases the management overhead. Unfortunately, besides having poor performance, wide area technologies cannot provide ubiquitous coverage either, and coverage can be decidedly spotty inside enclosed areas such as buildings. Furthermore, strict

TABLE I
Comparison of wireless technologies*

| | Blue-tooth | WiFi | GPRS | EvDO |
|---|---|---|---|---|
| Downlink data-rate (Mbps) | 2.1 | 54 | 80Kbps | 1.8 |
| Coverage-radius (m) | $\sim 10$ | $\sim 100$ | $\sim 1000$ | $\sim 1000$ |
| Tx-power (mW) | 130 | 1400 | 1250 | 3500 |
| Rx-power (mW) | 100 | 1150 | 600 | 1500 |
| Energy(mJ)/Megabit** | 47.6 | 21.3 | 7500 | 833 |
| Cost/month | 0 | $< \$20$ | $30 | $80 |

* These are representative values only. Actual values can differ based on the coding scheme used, distance from the base station, etc.
** Energy/Mbits in mW/Mbps = mJ/Megabit.

call-admission control is imposed on the number of simultaneous users in a given geographic area.

To sum up, we think that no single wireless access technology can be expected to provide *ubiquitous* high-bandwidth, power-efficient, and low cost coverage. This points to the need for intelligent switching among multiple wireless interfaces, such that users can opportunistically use one or more wireless networks to increase their overall communication capacity, power efficiency, and cost effectiveness [29]–[31].

### B. Delay tolerant applications

In parallel, we observe that for many non-interactive applications, users can tolerate data delivery delays on the order of minutes, hours, or even days. Examples include non time-critical applications such as personal email, mobile blog uploads, download of media clips, etc. This flexibility in data transfer delays can be used to optimize communication overhead in novel ways. For example, data transfer can be intentionally delayed such that cellular networks are not utilized, and instead free WiFi or Bluetooth networks are opportunistically used whenever they become available. This can reduce the monetary cost per bit. It can also reduce the battery consumption on mobile devices because faster WiFi networks used for short bursts of time consume lesser power than slower cellular networks to transfer the same amount of data. Note that the decision to delay data transfer is a policy-based decision because it involves user preferences for delay and cost.

Data transfer may be delayed not just on mobile devices, but also within the infrastructure. This becomes useful in situations when the wireless link supports data rates much higher than the backhaul connection from a WiFi or WiMax access point. This is a typical scenario because most backhaul connections use DSL that provides up to 5 Mbps, whereas 802.11g wireless links can support up to 54 Mbps. In such cases, WiFi access points can be enhanced with large local storage buffers. This will allow data transfer on the fast wireless link to proceed at the full data rate of the wireless technology: the data is buffered at an access points and transmitted in batches over the slower backhaul link.

### C. Policy design for opportunistic communication

Summarizing this discussion, consider for example a user who has email with photo attachments to send from her mobile device but prefers not to use the credit available on her expensive cellular plan. The user may, however, be willing to wait for a few hours in the hope of finding a free public hot-spot for sending this email. We would like to design a system that allows such a user to express interface usage preferences (e.g., minimize overall cost but tolerate transaction delays of no more than 3 hours). The rest of the process should be automated - the emails should be held back on the mobile until a free WiFi hot-spot is found. If none is found within three hours, then they should be sent out anyway over the cellular interface. The user should still be able to use a legacy email client application and should not have to be concerned with the intricacies of interface switching or the scheduling of data transmission.

Generalizing from the above scenario, our goal is to develop an architecture for policy controlled opportunistic communication on multiple wireless networks. This should allow users to specify policies at the applications-level; for example, can a certain application tolerate delays, and if so, then what is the maximum tolerable deadline for data delivery. Users should also be able to define policies in terms of bounds on the monetary cost associated with data delivery for different applications, or preferences to operate only over a certain type of network, etc. Similarly, users should be able to specify preferences on whether to minimize overall power consumption on the mobile device, or to minimize the cost of data transfer, etc. While providing this flexibility, our solution should also shield users and application writers from the dreary details of interface switching and session maintenance across disconnections.

We would like to mention that the notion of user-defined policies is well-known in the areas of autonomic computing, user interface design, and access control. The Policy SIG defines policies as "*a set of rules governing choices in the behavior of the system*" [33]. The same concept clearly applies to policy-based network selection. Although this has been looked into in the past [29], the novelty of our work lies in highlighting the importance of delay-based user preferences for the selection of wireless networks on mobile devices.

We note in passing that the same factors of monetary cost, data rates, and other user-defined policies that influence network selection on mobile devices, are equally relevant for network selection in rural Internet kiosks. Rural kiosks in developing countries provide a variety of services such as birth certificates, bill collection, email, land records, and consulting on medical and agricultural problems. These kiosks are unproductive without reliable Internet connectivity. Today, kiosks connect using dialup lines, Very Small Aperture (satellite) Terminals, EDGE and GPRS PCI/ PCMCIA cards, long-range WiFi, or through mechanical backhaul such as cars and buses that ferry data to and from villages and cities [39]. Each of these connectivity options differ from each other based of parameters of monetary cost, data rates, delay, and reliability. It is common for kiosks to have multiple options for Internet connectivity, and the selection of an appropriate network for different applications at different times can be complex. Interestingly, the same architecture we develop for network selection on mobile devices is also applicable for network selection in rural Internet kiosks. For ease of exposition, however, the focus of this paper is in the context of network selection on mobile devices.

### D. Road map

We describe a detailed use case for opportunistic communication in Section II, and use it to derive design goals for our system in Section III. Based on these goals, we give an overview of our system design in Section IV, and present the detailed system architecture in Section V. Evaluations of the implementation of our system is given in Section VI. This is followed by a description of future work in Section VII, related work in Section VIII, and a final discussion in Section IX.

## II. Detailed use case

In addition to the policy directed use of wireless NICs, a practical solution to policy-oriented opportunistic communication has to deal with several other issues. These issues are illustrated in more detail through the email-with-photo-attachments example from Section I-C, in the scenario of Fig. 1.

Here, a set of proxy (email cache) servers (marked 'P'), located in different data centers in the Internet, are accessed by the mobile

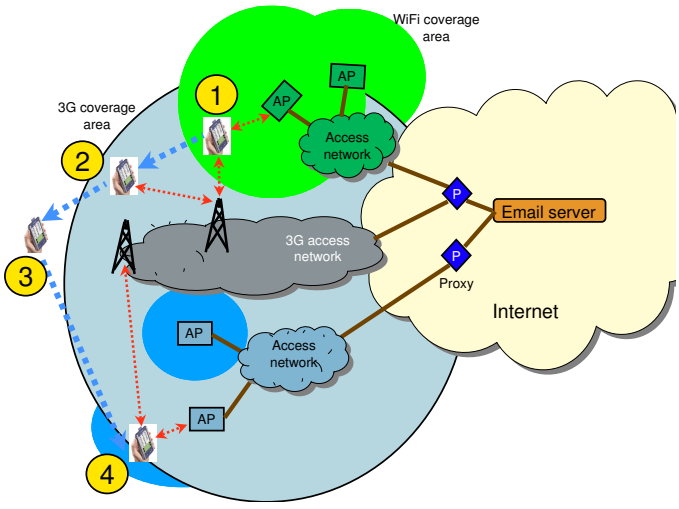


Fig. 1. Opportunistic communication

device. The device is initially at location (1) in overlapping 3G and WiFi wireless coverage areas. Because it can access the Internet using more than one wireless network, it needs to decide which network to use. The choice is complex, dictated by the dollar cost to use a wireless network, the power cost per bit, and the data rate, while simultaneously satisfying application requirements such as a 3 hour deadline for data delivery. How should these choices be resolved? Existing work on NIC selection only considers network properties such as cost and data rate to hierarchically rank networks with respect to each other [28], [29], and does not take delay based user preferences into account.

The device then moves to location (2), where it has only 3G coverage. How does it even know that it has left the WiFi coverage area? And how does it decide whether to use 3G, or to just wait for the next WiFi network? Should the user be involved in making this decision? This choice is equally complex, dictated by the user requirements, mobility schedule, and other network characteristics. Further, suppose the mobile decides to switch from WiFi to 3G because it predicts that it will not run into any WiFi network before the deadline. How can we hide this switch from the applications running on the mobile? Prior work to handle NIC switching or disconnections has only looked at small timescales at the network and transport layer [27], [30], [38]. Although systems such as [15], [22] do consider large timescale operations handled at the session layer, policy-based network selection using these systems has only considered the problem from a routing perspective, with a goal to minimize overall delays for data delivery [10]. To the best of our knowledge, design of a general user-defined policy framework operating at the session layer has not been studied in the past.

Now suppose the device moves to location (3), where it has no coverage. At this location, the applications on the device have to deal with disconnection. How should this be hidden from applications, most of which are not disconnection-tolerant? Other systems handle this by constructing application specific plugins to hide disconnections from legacy applications [22], and we adopt a similar approach.

The device now moves to location (4), where it can access the Internet from the second WiFi access network. The device now has to decide whether this network is safe to use, and, if not, then use application-layer security mechanisms to protect privacy. Moreover, applications need to recover session state established in location (1) or (2) so that they don't start from scratch each time the mobile moves to a new location. Such scenarios have not been explore in existing work.

This use-case above elicits the requirements that we enumerate in the next Section.

## III. Design goals

1) *User-directed use of multiple networks*: Today's mobile devices can detect the availability of one or more access networks, but the interface selection is completely manual. This is clearly not suitable for opportunistic access. We'd like a mobile device to have knowledge of factors such as the bandwidth and congestion state of the available wireless networks, energy-efficiency of the radios, and monetary costs of the networks. The device should then be able to use this knowledge to decide connectivity choices that satisfy the user requirements, and automatically connect to the appropriate networks. Thus, users should be able to specify application-specific policies such as data delivery deadlines, or bounds on monetary costs, or network preferences. Similarly, users should be able to specify application-independent preferences such as minimization of cost or power consumption. Whenever the mobile device is in the presence of one or more networks, it should be able to automatically select and connect to a suitable network in accordance with the user-defined policies.
2) *Support for legacy servers*: It is unlikely that a solution that requires changes at a server will ever be deployed in practice. Consequently, we would like a solution that interoperates with existing servers.
3) *Application session persistence across disconnections*: Consider a mobile device that has established a connection to a server using one of several network interfaces. Suppose the mobile decides to power itself off or switch to a different interface; then reconnects to the same server with a different IP address. With existing systems the server would be unable to recognize that the two connections correspond to a single ongoing data transfer session, and therefore would not be able to migrate persistent application state from the old connection to the new. For network access to be truly opportunistic and seamless, an application should be able to exchange data with a server when changing network interfaces or even when faced with intermittent loss of connectivity. This requires the maintenance of persistent application state at both the server and the client so that data transfer can resume from the point where it stopped once connectivity is restored.
4) *Optimized network switching*: Consider a user who is walking or driving in a car, and whose mobile device opportunistically connects to WiFi networks. Disconnections are likely to be unclean in such a scenario, and connections may terminate without an appropriate handshake that clears data in transit, or data sitting in local transport and link layer buffers. Although session layer persistence [3], [22], [27], [38] can ensure correctness that such data is not permanently lost, these protocols operate at timescales of the order of minutes (TCP timeouts); this

can delay retransmission of data and result in large re-sequencing buffers at the receiver. It can also lead to the maintenance of redundant connection state at the proxy, because the proxy may wait for lengthy transport layer timeouts before tearing down the connections even though the mobile is already disconnected. Thus, we would like to have alternative mechanisms to better deal with network disconnections.

Detecting and selecting a 'good' network from among multiple networks also has room for optimization. Suppose the device is confronted with a choice of five to ten publicly accessible WiFi networks at the same time, which can be quite common in some urban settings. If the device connects to a few networks and probes the network quality to find a good network, valuable connection time can be wasted [32]. Again, optimization is required so that the mobile device can make quick decisions.

5) *Ease of application design and implementation*: Application designers who are familiar with the socket-bind-connect approach to writing distributed applications cannot deal well with systems where connections may fail arbitrarily, be resumed arbitrarily, and exhibit large variations in bandwidth depending on the currently available network. We would like to insulate application developers from these problems and provide them with a simple and intuitive communication interface.
6) *Support for buffered access points*: Access points enhanced with local persistent storage make better use of the faster wireless links because data is not bottle-necked at a slow backhaul link from the AP to an ISP. We would like to support such APs.

   This design goal is also motivated from the desire to support rural kiosks connected to the Internet via mechanical backhaul [25]. In such a system, cars and buses that regularly travel between villages and cities, carry a WiFi access point having local storage. This 'mobile' AP ferries data between rural kiosks and an Internet gateway. The rural kiosks and the gateway also possess a similar AP to hold the data locally until it is picked up by a mobile AP. APs communicate with each other over short distances using WiFi, and data is transferred whenever a mobile AP comes in range with a stationary AP at a kiosk or gateway. Since cellular connections are typically unavailable in remote rural areas, users possessing mobile devices can continue to upload or download data via these WiFi based APs at the kiosks. Therefore, we would like solutions that can support buffer APs that may or may not be connected directly to the Internet.

We address these goals by means of our system architecture and Opportunistic Connection Management Protocol (OCMP). We present an overview of the system design in the next Section, deferring details of OCMP to Section V.

Note that efficient opportunistic communication also requires solutions to issues such as 802.11 association delays, wireless losses due to interference and mobility, appropriate MAC rate adjustment, impact of lower layers on TCP and other transport layer implementations, etc. We defer an analysis of such factors to future work, and focus only on the design of a policy-based architecture for opportunistic communication.

## IV. Design overview

Fig. 1 also presents an overview of the system architecture. The main components are the content host, the proxies – which run OCMP servers, and the mobile host – which runs the OCMP client. We describe each component next.

### A. Content host

At the right of Fig. 1 is the content host, a server that either provides content such as video or stored voice to a mobile device, or receives uploads and content requests from the mobile. This represents popular web sites like youtube.com and yahoo.com, or media servers that provide audio and video content. Content hosts reside in a data center at the core of the Internet. These servers are connected to a wired, high-capacity, and global IP core backbone. Existing content servers run legacy applications and do not support disconnection resilience or parallel transport connections over multiple networks for a single application session. We would like to provide a feasible path for supporting opportunistic communication *without* requiring modifications to legacy servers. We achieve this via the deployment of network-based proxies which are described next.

### B. Proxy servers

Proxy servers (marked 'P') allow interworking between legacy servers and our protocols [4], [15], [22]. A proxy is located in the communication path between a mobile device and a content host. It serves as the termination point for the transport connections opened by the mobile host over multiple network interfaces. The proxy server hides multiple connections and disconnections from the content host. It can also provide fine-grained and application-specific connection management, as will be described in Section V.

The proxy can either be provided by an Internet Service Provider, or by an enterprise on behalf of its employees. Proxies should be placed so that the round trip time from the proxy to the bulk of the mobile devices is as low as possible [23]. For example, cellular providers could keep the proxies adjacent to the PDSNs in CDMA or the GGSNs in GPRS networks, or on the backhaul point to the Internet core [24]. On the other hand, if a third party provides a proxy as a value-added service, it should place the proxy in a well-connected data center. We only require that the proxies have one or more globally-reachable public IP addresses, or dynamic DNS registrations.

The proxy acts as a store-and-forward agent for data downloads to a mobile device. A download starts with a mobile application initiating a data transfer request, e.g., an HTTP GET request. This request is intercepted by the OCMP client on the device and forwarded to the proxy, as in PCMP [22]. The OCMP server on the proxy supports an *application plug-in* that allows it to understand how to process application-specific data transfer requests. If the request is from an application supported by the proxy, the proxy processes the request and then uses legacy protocols to contact the content host on behalf of the mobile device. Thus the content host is completely shielded from all details of communication between the mobile and the proxy.

Once data is downloaded from the content host to the proxy, the proxy caches the data and looks for available connections to the mobile device. No such connection may be available at that time if the mobile device is temporarily disconnected from all

access networks. If so, the proxy holds the downloaded data in persistent storage until the device reconnects. Alternatively, the proxy can use an out-of-band mechanism (e.g., an SMS message if the mobile device is a smart phone) to inform the mobile device about the availability of data. When the mobile device reconnects over one or more wireless networks, the proxy segments the application data into *bundles* (which are the message units in Delay Tolerant Networking [15]) and routes the bundles over these connections. The policy-based algorithm for determining which bundles to transmit on what connections is negotiated in advance between the OCMP peers on the mobile client and the proxy.

When data is being uploaded from the mobile device to a content host (e.g., blog or picture uploads), the proxy receives bundles from a single application, potentially over multiple transport connections from the mobile, reassembles them into a single stream, opens a connection to the content host and forwards the data using application-specific protocols. Thus the OCMP client on the mobile host and the proxy implement analogous functions for segmentation and reassembly of application data as well as policy-based routing of application data segments.

We envision that the multi-connection state between a mobile and a proxy can be packaged and moved to a different proxy to allow a mobile to always use a 'nearby' proxy, greatly improving performance. Similarly, the state on a mobile device can be retained persistently across arbitrary periods of disconnection or power loss. The state can even be transferred to a different endpoint like a home or office desktop, and unpackaged to recreate an operating state identical to the state on the mobile prior to disconnections. This can be used to provide semantics similar to that provided by Internet Suspend and Resume [18]. Note that we have not implemented a multiple-proxy system as yet. We describe it briefly as part of future work in Section VII.

### C. OCMP

The proxy and a mobile may be connected by multiple heterogeneous wireless networks that differ in coverage, capacity, pricing, and availability. An OCMP server-side protocol running on the proxy coordinates access on these networks with an OCMP client-side protocol running on each mobile. OCMP defines a message format for encapsulating application data segments for session-level data reassembly. OCMP also supports control messages, i.e., messages that consist of only an OCMP header and an empty body. For example, control messages are exchanged between the OCMP client and the proxy to coordinate policies regarding selection of network connections as described in Section V-H.

Unlike past work [27], [38], OCMP does not depend on TCP semantics of the underlying connections, as long as the transport layer provides end-to-end reliability. An underlying connection can be a standard or modified TCP/IP connection [8], [20] or can be a transport protocol optimized for wireless networks, such as erasure-coded UDP [6], [7]. OCMP can therefore exploit systems that compress and transcode data on wireless links to optimize bandwidth use [44].

Using OCMP, data can be striped on multiple connections in parallel, under fine-grained application control. Essentially, OCMP clients and servers choose the connection to be used for each application-level data unit based on pre-specified policies. Moreover, if a connection abruptly terminates, or even if *all* the connections terminate, the OCMP client and server gracefully recover from the failure, providing applications the illusion of seamless connectivity.

Besides working with the server-side, the OCMP client-side also has the additional responsibility of detecting network connections and disconnections. It uses application-specific policies to decide whether it should initiate a connection to the server side when a connection opportunity arises. It also has a notification mechanism to inform an application if there is any data that has arrived for it.

Connections between an OCMP client and OCMP server are always initiated by the client. A new transport layer connection is created each time the device connects on a new network and torn down when the device disconnects. Each network interface is associated with a single transport connection that is shared by all application data units assigned to that interface.

### D. Mobile

The proxy identifies a mobile device (and all connections originating from it) by a globally unique identifier (which could be an IMSI or phone number). The GUID for a mobile device is common to all its transport connections, and serves several purposes. It decouples device addressing (a GUID) from routing (in terms of IP addresses), as in HIP [21]. This solves the problem of IP address changes due to mobility and/or disconnections. It also enables the proxy to maintain persistent data transfer state even when a mobile application uses parallel transport connections over different access networks.

The OCMP client on a mobile device provides two application interfaces. The first is meant for legacy applications that are designed in the socket-bind-connect paradigm. For such applications, application-specific data download requests are intercepted by the OCMP client and sent on a control connection to the proxy. The OCMP server layer in the proxy, acting on behalf of the client, initiates a connection to the server and downloads the data. It then transmits the data to the mobile device. The OCMP client layer on the mobile device reassembles the data before delivering it to the application. For data uploads, the proxy reassembles data received from the device and transmits it to the server.

We have also built a new application interface for disconnection- and delay-tolerant applications. It takes the form of a 'communication directory', which is a standard directory in the file system. An application writer drops a file into this directory, and is guaranteed that the file will appear at a corresponding directory on the destination at some point in the future. Policies can be associated with each directory, and can be defined through a configuration file for different classes of applications. This is described in more detail in Section V-A. We have found that this API is both robust and easily understood by application developers.

We have developed a Java-based prototype implementation for the system described above and evaluated its performance on laptops with diverse wireless access technologies such as 802.11b/g, CDMA 1xRTT, and GPRS EDGE. A detailed performance evaluation of our prototype is presented in Section VI.

### E. Buffered Access Points

Whereas our solution will work with off-the-shelf WiFi or WiMax access points, we recommend enhancing these access

points with store-and-forward infrastructure. This allows for better utilization of the wireless link capacities, which are typically much higher than backhaul bandwidths for public access networks. We have developed such prototype access points using single-board computers from Soekris Corporation [42] fitted with 40 GB harddisks. These access points run the DTNRG Delay Tolerant Networking software [9]. They interface with OCMP running on the mobile devices on the wireless network, and OCMP running on the proxies in the Internet on the wired network. Note that the same DTNRG software can also be used for mechanical backhaul because it supports multiple hops of disconnections. We have suitably extended it to make it usable as a practical mechanical backhaul solution for rural kiosk communication, as well as to support data transfer for mobile devices via rural kiosks [25].

Such buffered access points are ideal for data uploads from mobile devices. Data can be opportunistically transferred to the intermediate access points at the highest data rates possible, and session level persistence ensures that the data is correctly reassembled and conveyed to the proxy server.

Data downloads are more difficult because the proxy servers need to be aware of the next access-point that the mobile will connect to pre-cache downloaded data, and this depends on the mobility pattern of the users. We envision using network coding for redundancy and to compensate for incorrect mobility prediction by replicating data on multiple access points, such that if the mobile device is able to pick up a certain minimum amount of data, it can decode and reassemble the rest of the data. Although we have not addressed the problems of mobility prediction and optimal network coding in this paper, our implementation is flexible to accommodate such features in the future.

### F. Control channel

We observe that cellular networks are nearly ubiquitous, even though they may not support high data rates. We use this insight to assign a special role in OCMP to the cellular network, other than its regular use for data transfer. The cellular network in OCMP provides a *control plane*. For example, when mobile devices are confronted with a choice of multiple WiFi networks to choose from, they can use the control channel to query a centralized GIS (Geographic Information System) for the best-performing network at that time [40], or report back the performance status of different networks to keep the database updated in real time. Similarly, the control channel can be used for DHCP negotiations and pre-authentication to achieve small WiFi drive-thru association latencies. The control channel can also be used to send disconnection notifications for more efficient handling of in-transit or buffered data that is lost during unclean disconnections, and reduce the amount of redundant connection state being maintained. Applications can use the control channel in many other ingenious ways too [26].

### G. Policy control

We now turn our attention to the main focus of this paper, that is, policy control of communication over multiple wireless NICs. OCMP allows policy definitions at the application-specific and application-independent levels, and provides a framework in which algorithms can be implemented to exercise these policies. Any configuration parameters required by policies are assumed to reside in a policy module in OCMP, and can be queried to enforce policy at two key decision points.

1) *Which network to connect to*: When faced with a choice of multiple networks which may be of different types, decisions can be made based on the policy specifications.
2) *Which application data unit to schedule for data transmission*: When multiple applications compete for a common network resource, decisions have to be made based on the policy specifications.

OCMP is designed such that policy enforcement algorithms and policy configuration can be distributed among different entities. For example, policy configuration data may reside on the mobile device, but proxies may remotely query the policy module on the mobile device. If the enforcement algorithms are complex, then the algorithm evaluation can be performed in the infrastructure instead of using valuable battery and computation resources on the mobile device. The results of the algorithm can then be conveyed to the mobile device and enforced locally using a control channel.

In the current implementation, the policy enforcement algorithm to be used is specified as a startup parameter in OCMP. This algorithm has two methods: (a) $connect$, which is called each time a new network is detected, and returns $true$ or $false$ to indicate whether to connect to this network or not, and (b) $getBundle$, which is called whenever the transport queue of an active network is vacant, and returns a $bundle$ to be dispatched on this network, or returns $null$. We have implemented only a simple policy so far, described later in Section V-H. However, any other kind of a policy enforcement algorithm can be used.

1) Policies can be modeled as utility functions, where the utility gained is greatest when application data is delivered immediately, and the utility decays with time, eventually going to zero after the deadline. An example utility function can be $U = a - bt$, where $a$ and $b$ are constants, and $t$ is the time difference between the time of data delivery and the time when the application started. Algorithms can now be designed to schedule applications such that the overall utility is maximized. This is the approach followed in [35], although the focus is on real-time applications where utilities can be based on round-trip latencies or loss rates or cost.
   Application-independent utility functions can also be designed, so as to minimize power consumption and maximize the total application-specific utility simultaneously.
   Note that the performance of such a scheduling algorithm may depend on whether or not it can successfully predict the expected mobility schedule of a user.
2) Policies can also be modeled as rules; for example, a policy may dictate to always use WiFi instead of EDGE, unless application deadlines are only an hour away. Such policy enforcement algorithms may require to implement a rules-engine, where rules can be specified in a policy definition language [34], along with suitable conflict resolution procedures.

Design of fast and optimal policy algorithms is an area of future work. This paper describes the framework necessary to implement different policies.

## V. System architecture

This section describes the OCMP client and server protocols in greater detail.

### A. Client-side communication API

OCMP interacts with applications on the mobile client either by means of a 'communication directory' or by intercepting socket calls made by legacy applications.

A communication directory is simply a directory in the file system that contains application data. To send data, an application creates a new file in the directory, or modifies an existing file. A 'watcher' process periodically looks for modifications to the last modified time of the directory. If the modification time is more recent than the last time the directory was checked, the newly created or modified files are sent to the OCMP stack using the OCMP API.

The watcher also registers itself as a default plugin with OCMP, much like inetd. When called, it writes data to a file in the appropriate communication directory, and invokes an application-specific script to notify the application when its data has been received. The application can simply read this file to get its incoming data.

Each communication directory has two special files. The *config* file has application-specific configuration parameters. For example, for the blog-upload application, this is the username and password for the user. The config file also contains application-specific policies to control the interface(s) used for transferring data for that application. These policies are passed to OCMP by the watcher. The other special file is the *status* file. This file has one entry for each file in the communications directory and contains the status of that file. The status of a file can be, for example, 'ready to send', 'partially sent', or 'sent'. An application that wants to know the status of a file's transfer can read the status file. This can be used, for example, to display progress in a user-friendly GUI, shown later in Fig. 8.

The use of a communication directory simplifies application development. An application writer has to only create a send and receive directory and the associated config and status files. After that, all communication is achieved by writing files or reading files from the directory.

In addition to the communication directory, we support legacy Java applications by intercepting 'socket' calls in the Java API. These calls are instead handled by OCMP, specifically by the application plugin associated with that application. OCMP infers the plugin associated with a socket call by looking at the destination port number as well as the first few bytes of the written data. We describe this in more detail in Section V-D. After the interception, the remainder of the processing is identical as with the communication directory.

### B. OCMP protocol stack

The OCMP client and server stacks that run on a mobile and on a proxy respectively are shown in Fig. 2. On the client side, OCMP-aware applications interact with OCMP through a communication directory monitored by a 'directory watcher'. Also, socket calls made by legacy applications are intercepted by OCMP, which redirects them to application-specific plugins.

We assume that applications or their associated plugins can categorize their communications into either a control or one or more data *streams*. For example, an email application may create a data stream for email bodies, and another stream for email attachments, where the delivery deadlines for attachments may be more relaxed than the deadlines for email bodies. The application control stream (shown by 'App Ctrl' in Fig. 2) provides an explicit control channel between the application plugin peers running on the mobile and proxy. For example, it is used to tell a receiver about the length of the bulk data sent on a data stream, or application parameters required by a peer plugin. It can also convey to the mobile the status of the data transfer between the plugin on the proxy and legacy servers.

Each application data stream is assigned to a *Storage manager* that segments/reassembles the data into/from multiple bundles. It also stores the data in persistent local storage to deal with power loss on the device. Each data stream has an associated policy that is registered with the OCMP *policy module*. The policy module maintains a collection of preferences for data streams belonging to different applications. It also contains user-defined preferences. This set of user and application preferences are used by the OCMP *scheduler* to run policy enforcement algorithms to schedule application bundles on different interfaces.

The scheduler contains a *connection pool* object that maintains a list of active transport layer connections, one on each interface. Each connection is encapsulated in a *connection object*. The scheduler maintains a pre-fetch buffer for each data stream to minimize latencies in fetching data from the disk. It then selects bundles and sends them to one of the connection objects depending upon network availability and the user-specified policy. The scheduler can also decide what kind of a transport layer to use on each interface. The scheduler also may choose to asscociate on a network when a *link detection* module notifies it of new networks in range.

At a proxy, incoming bundles are processed by a symmetric stack and eventually handed to an application-specific plugin. These plugins can be loaded into OCMP dynamically on packet arrival. The plugin can then take application-specific actions to transfer the data to a legacy server. The plugin can also fetch data from a legacy server on behalf of an application and store it in data streams on the proxy. When a mobile opportunistically connects with the proxy, this data is sent to the mobile.

Note that the scheduler can be implemented either on the mobile or at the proxy. Computationally-efficient devices can run scheduling algorithms on the device itself and remotely set light-weight scheduling rules on the proxy. On the other hand, computationally-starved devices can shift schedule computation to the proxy, and rely on simple rules for bundle scheduling on the device. This flexibility is possible because the policy module that contains various user preferences can be queried remotely over the cellular control channel. Data on network status or mobility pattern can similarly be exchanged over the control channel to provide timely information to the scheduling algorithm. Note that in an area with no coverage, where the control channel is absent, no scheduling decisions are needed in the first place.

### C. Session-level reliability

Due to the presence of a send buffer in the network stack, write calls that enqueue data into a non-empty send buffer return successfully, making an application think that the data was reliably delivered to the receiver, even though it might not be delivered at all if the connection closes prematurely! In this case, the data in the send buffer is actually lost after a connection termination is announced to the application. Similarly, on the receive side, bundles that have been acked by TCP, are not necessarily passed to the OCMP agent on an unclean disconnection. Hence, with

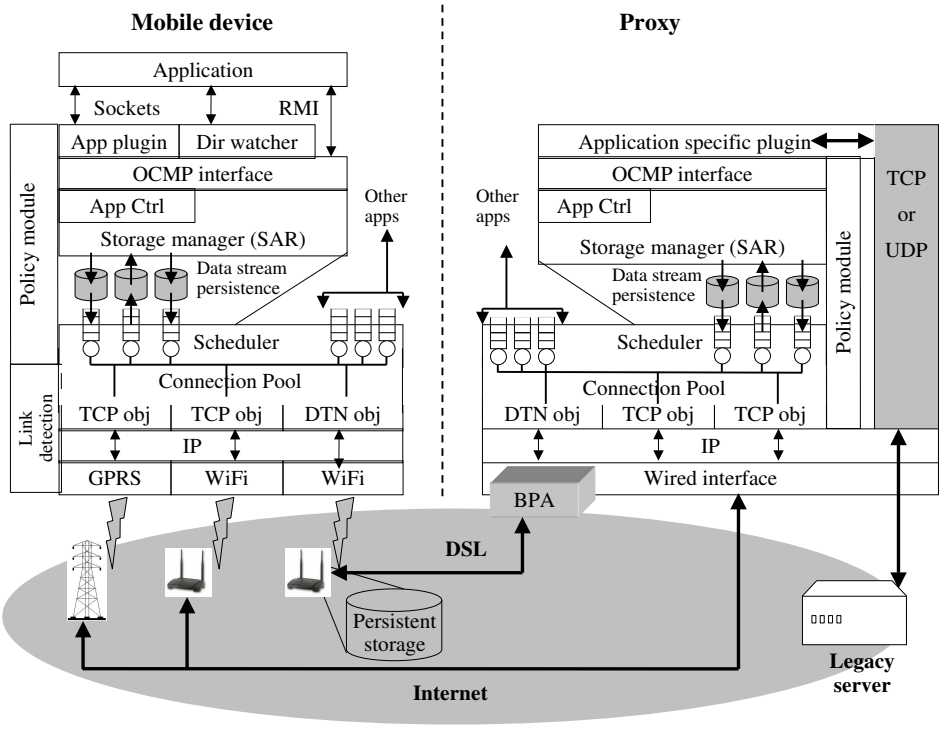


Fig. 2. OCMP stack

any buffered protocol stack, there is always a possibility that data equal to the sum of the send and receive buffers is actually lost, even though the sending side believes the data to have been delivered successfully. For this reason, transport layer semantics are insufficient for reliable delivery, and a session level reliable data transfer protocol is needed to recover from lost data.

To avoid the overhead of a per-bundle ack or nak protocol, an OCMP sender keeps track of the order in which it transmitted data on each network interface, and retains, in persistent store, all data that might possibly get lost in transit. When a connection closure is detected, the receiver uses the control channel to inform the sender of the last sequence number bundle it successfully received on that connection. This allows the sender to infer the set of bundles that were not successfully received. The sender therefore queues them for transmission on a working interface, or marks them as undelivered for subsequent retransmission.

The ability for one end of a connection to inform the other of unclean connection termination on an alternate interface is a useful feature of OCMP. This is because we have found that in practice, one of the ends knows about a disconnection far sooner than the other. This technique allows both ends to reason correctly about the disconnection and to take corrective action. Typically, disconnections are due to wireless failures, which the mobile device finds out about much faster than the proxy. The mobile then sends a disconnection notification, along with the last sequence number it received on the WiFi interface, on the control channel. If the proxy was sending some data to the mobile on the WiFi interface, it can immediately retransmit the data sent on the failed interface after the last sequence number received by the mobile. The proxy responds to a disconnection message with a reply that carries the last sequence number it received on the failed interface. In case the mobile was uploading data, it can now retransmit everything it sent after the last sequence number that was received by the proxy. This allows us to quickly recover from a broken connection. We evaluate the performance of this technique in Section VI.

The discussion is illustrated in a sample scenario shown in Fig. 3 for a mobile device that encounters intermittent WiFi connectivity and uses both WiFi and EDGE for data transfer. The protocol begins when the OCMP proxy notifies the mobile device that it has data waiting to be picked up by the device. We assume that these notifications can be sent through an out-of-band mechanism, such as SMS. When the mobile receives this notification, it asks the link detection module to raise an event whenever the device connects to a new network. Thus, when the mobile connects to a WiFi hotspot, the OCMP control layer decides to use TCP as a transport layer on WiFi to connect to the proxy. The connection is initiated through a control message, which first instantiates an OCMP *connection* entity for the mobile on the proxy if it did not exist already. The connection is then added into the connection pool. If the WiFi connection breaks uncleanly, the EDGE connection is used to send control messages to the proxy so that the proxy does not have to wait until a TCP timeout to detect the connection failure.

### D. Application-specific plugins

Both the OCMP client and the server support application-specific *plugins*. These short-lived code modules are invoked to carry out application-specific actions for each client-server interaction. All applications need a plugin at the proxy, and legacy

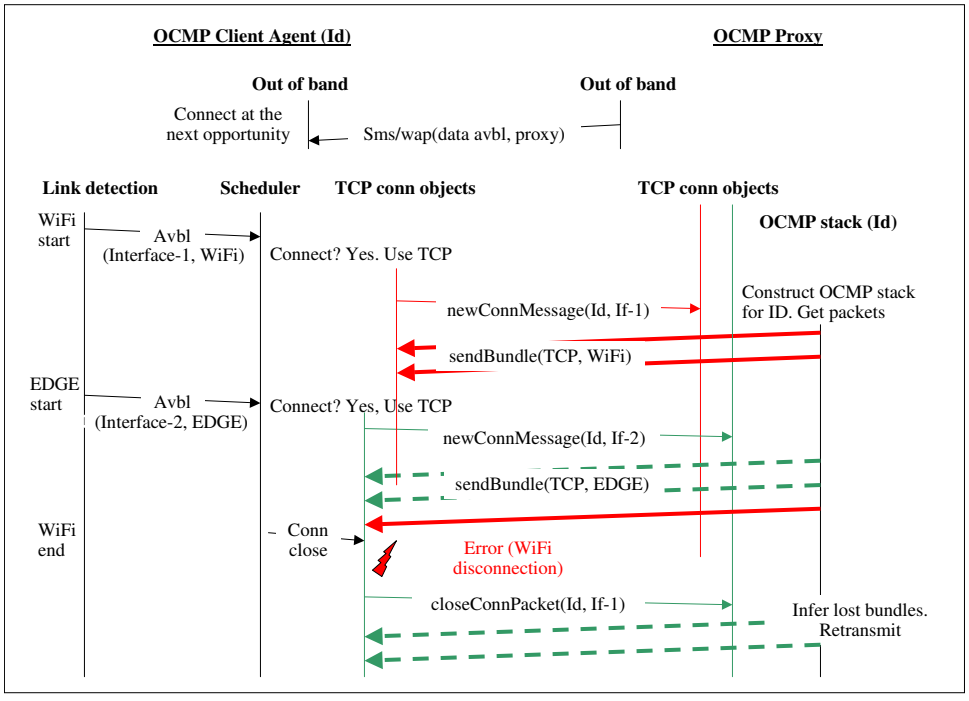


Fig. 3. Control flow sequence diagram

applications need a plugin at the client end as well. For example, a legacy web browser request on a mobile is associated with an instance of a HTTP plugin both on the client on the proxy that initiates an HTTP GET on its behalf. The proxy-side plugin stores the results in persistent storage and communicates them to the client over opportunistic links. Other examples are a blog plugin to support upload from a mobile device to Blogger, and a Flickr plugin to upload a photograph to Flickr. Application plugins attempt to mask a mobile's disconnections from legacy applications either on the mobile or at the content host. Of course, long disconnections that last for hours or days cannot be masked, particularly from interactive applications. However, opportunistic communication is ideal for delay-tolerant applications, such as music downloads and email.

An instance of a plugin is created on the mobile if OCMP intercepts a socket call made by legacy applications. The destination port number or the first few bytes written into the socket are used to disambiguate applications, and a corresponding plugin object is created to handle the connections. Whenever a new plugin is created, or a new file is dropped into the 'communication directory', an application control message is also sent to the proxy to ask it to dynamically instantiate a peer plugin on the proxy. Application control messages have an application ID and application type field to uniquely identify the correct plugin and the type of the plugin. The plugin then collects *one-time* data from the legacy server, hands it to a SAR (Segmentation and Reassembly) agent, and finally destroys itself. A distinct plugin object is therefore associated with each client interaction with the server.

Persistent application daemons can also be created at the proxy that either monitor legacy servers for updates, or receive 'push'-style updates from the servers. The data from these updates is then handed to an application plugin, which hands over the data to SAR agents in the usual way. If the mobile is already connected to the proxy over a control channel, an application control message is sent to the mobile to notify it about pending data lying at the proxy. The mobile now either downloads the data into the 'communications directory', or instantiates the appropriate application plugin to handle the incoming data. If the mobile is not connected to the proxy, which could happen if the mobile is temporarily unable to access data services on cellular networks, an out-of-band SMS message can be sent to the mobile to indicate pending data. The client OCMP running on the mobile receives this SMS and tries to connect to the proxy whenever connection opportunities arise.

### *E. OCMP identifiers*

As described earlier, OCMP identifies each mobile device by a unique GUID such as its IMSI [36]. The proxy uses this ID to demultiplex bundles belonging to different users. A different class of identifiers is needed for some applications. Consider a proxy that registers itself as the email server for a set of mobile users using a DNS MX record. When receiving incoming email, the proxy needs to find the user's OCMP-GUID. Therefore, the proxy needs to maintain a mapping from the user's application-specific address, such as an email address, to the user's GUID so that when the user connects to the proxy, it can send data to the correct user.

We have defined a framework on the proxy to support translation from application-IDs to OCMP-GUIDs. A registered appli-

cation can create a daemon on the proxy that maintains mappings from application identifiers to the OCMP identifiers for all users of that application. This daemon is also registered to receive content from legacy servers. So, when a content server pushes data to the application daemon, it can instantiate an application specific plugin with the correct OCMP-GUID for the user, and redirect the incoming data to the plugin. The plugin caches the data in the usual way and delivers it to the mobile whenever it connects.

Note that each bundle carries a *GUID* to distinguish bundles belonging to different users, an *application identifier* so that bundles can be routed to the correct application, a *SAR agent identifier* for each data stream, and a *sequence number*. A concatenation of the first three identifiers defines a unique *session identifier*.

### F. DTN support

DTN is modeled as a network interface for OCMP that implements its own transport layer protocol. Whenever a mobile device detects a WiFi network, OCMP examines the SSID to determine whether the network belongs to a DTN access-point, or it is a third party WiFi network providing a direct connection to the Internet (other methods can also be used, such a UDP broadcast on a particular port, or querying a GIS over the control channel for more information about the WiFi network). A different type of connection is instantiated depending upon whether the access-point provides a store-and-forward facility or not. We have implemented a new *Connection* object for DTN that talks to a DTN Bundle-Protocol-Agent (BPA) running on the DTN access-point. This is shown in Fig. 2, and works as follows:

- *OCMP on mobile host*: A BPA does not run locally on the mobile host because the DTNRG reference implementation is not in Java. Instead, OCMP connects wirelessly to a BPA stub on the DTN access-point, and transfers data to it using TCP over the stub's RPC interface. Custody transfer acknowledgments are relayed back from the BPA to the DTN *Connection* object in OCMP. Thus, OCMP is made to believe that it is actually talking to a proxy in the Internet, but the BPA successfully masks the absence of an end-to-end route to the Internet.
- *DTN on access point*: The BPA receives bundles from OCMP, and stores them locally for subsequent forwarding to the OCMP proxy over DTN.
- *OCMP on proxy*: The proxy has a corresponding BPA running locally that receives DTN bundles from the access-points. OCMP bundles are extracted from the DTN bundles, and passed on to application plugins just like other OCMP bundles.

  For data to be sent to a mobile device through a DTN access-point, the scheduler on the proxy first checks whether the mobile device is registered with a DTN access-point, or it is likely to be in the presence of one. This step is crucial for the proxy to decide whether to retain the data in OCMP or to push it into the DTN overlay by routing it to an appropriate access-point. In the former case OCMP layers itself on TCP/IP as usual, but for the latter case, the proxy instantiates a connection to the BPA running locally and dispatches all the data to this BPA for eventual delivery to the user's custodian.

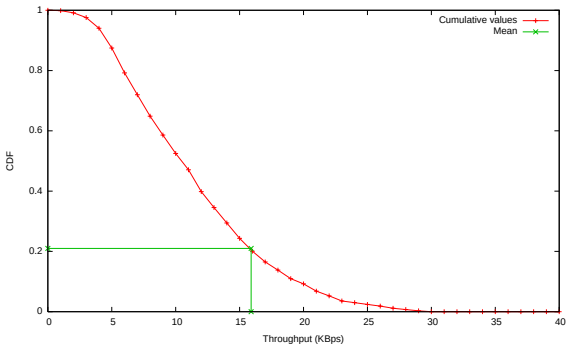


Fig. 4. CCDF of EDGE application layer throughput

### G. Control channel

Based on the discussions above, we summarize the uses of the control channel:

1) Send disconnection notifications when links terminate uncleanly, so as to reduce the amount of redundant state being maintained, and to exchange session state between the end points to reduce the size of the resequencing buffers.
2) Query the policy module on the mobile device for information about user preferences.
3) Exchange data required for policy enforcement algorithms. This data may include network performance status, or remaining battery life on the mobile device, or network cost information, etc.
4) Transfer application control data to instantiate plugins.

However, availability of the cellular network is not essential for correctness in OCMP. It is meant only as an optimization mechanism. We quantify some benefits of this approach in Section VI.

### H. Policy control

The focus of this paper is on an *architecture* that supports policy definitions. To exercise the architecture, we evaluate a simple policy described next (we are studying more sophisticated policies in current work).

We have implemented a simple policy enforcement algorithm for messages with deadlines, which works as follows. We connect to WiFi networks in preference to cellular networks, assuming lower usage costs for WiFi. We assume having prior knowledge of the average throughput provided by the cellular network. This assumption is easily justified by an experiment, where an EDGE network was regularly probed over the duration of one day, and found to have a throughput of 16 KBps or more approximately 80% of the time, shown in Fig. 4.

Each application data stream registers itself with the policy module on the mobile device, and specifies the size of its data stream, a delivery deadline, and direction of data transfer(uplink or downlink). The scheduler then back-computes an approximate commencement time for each data stream, ordering the streams from the furthest deadline to the nearest deadline. This is done by assuming the worst case scenario, i.e. the device may not run into any WiFi network and the cellular network will have to be used

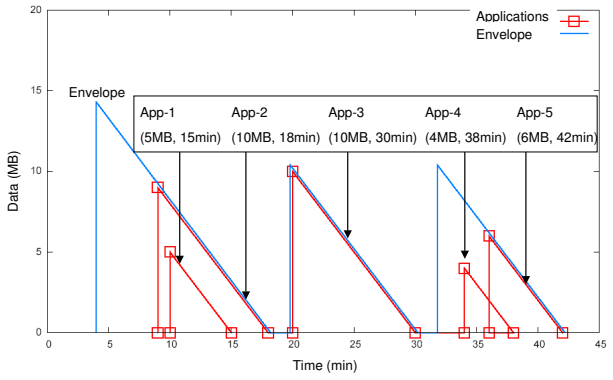


Fig. 5. Simple network selection algorithm

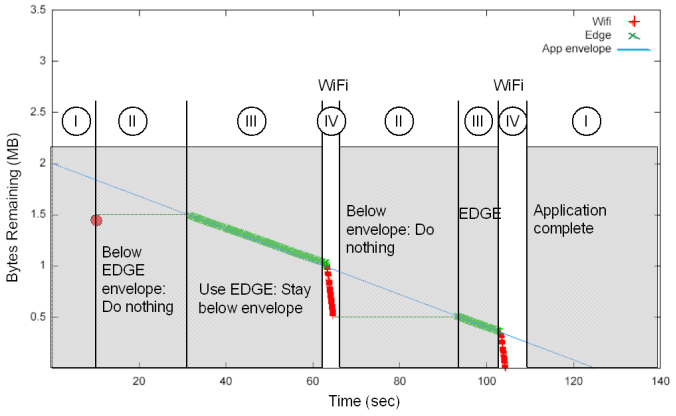


Fig. 6. Single application, easy deadline

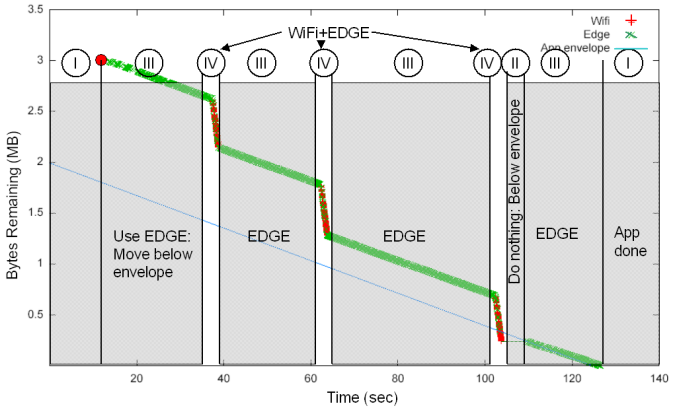


Fig. 7. Single application, aggressive deadline

for data transfer; thus, data delivery must commence for each data stream preceding its deadline by a time of at least (size of remaining data / cellular throughput). If the commencement time for a data stream overlaps with the deadline for another data stream preceding it in the stream ordering, then the commencement time is appropriately adjusted to accommodate both the streams.

This is shown in Fig. 5 as a *threshold envelope* for data delivery: the cellular network will be used only when the amount of data remaining to be transferred exceeds the envelope. The envelope can be computed at any instant of time using dynamic programming, given the deadlines and amount of remaining data for the applications currently in progress. For example, Fig. 5 shows the commencement times for five applications ordered according to deadlines. The cellular network throughput is assumed to be 16KBps ($\sim$ $1MB$ per minute). The figure shows that since the commencement time computed for App-2 overlaps with the deadline for App-1, the corresponding threshold envelope is computed with an earlier commencement time. A similar adjustment is made for App-4 and App-5. In general, given $n$ applications with corresponding deadlines and sizes of remaining data $(d_i, s_i)$, ordered according to deadline such that $d_i \leq d_{i+1}$, and given the mean cellular throughput $f$, the commencement times $(c_i)$ can be iteratively computed as:

if $c_i \geq d_{i-1}$, then $c_i = d_i - fs_i$
else $c_{i-1} = d_i - f(s_i + s_{i-1})$ and the process is repeated.

The threshold envelope is recomputed whenever a new data stream is registered, or a disconnection takes place from a WiFi network that was being used for data transfer. Timers are then initialized for each commencement time, so that data delivery can be triggered at that time instant. Now, whenever a new WiFi network is seen and there is pending data waiting for delivery, the WiFi network is always used for data transfer. On the other hand, the cellular network is used only when the amount of remaining data exceeds the commencement time and there is no WiFi network available, or data is striped across both cellular and WiFi networks in case both are available. At any instant of time, the data stream with an earlier deadline is given preference over other data streams.

Experiments with this algorithm are evaluated in Section VI. We also describe several different policy alternatives in Section VII.

## VI. Evaluation

We used an HP-Compaq 1.8GHz laptop running Windows XP for our experiments. A WiFi connection was provided over 802.11g with a DSL backhaul of 5Mbps, and a cellular connection was provided through an EDGE PCMCIA card. All our experiments were conducted in a stationary environment to minimize the effects of mobility on our results because mobility is not the focus of this paper. We instead implemented an emulation-module in OCMP to emulate WiFi connections and disconnections, assuming an arbitrary mobility schedule. We recognize the fact that our emulation methodology does not take into account factors such as 802.11 association delays, wireless losses due to interference and mobility, automatic rate adjustment, impact of lower layers on TCP and other transport layer implementations, etc. We defer an analysis of such factors to future work, and retain the focus of the evaluations in this paper on the design of the policy based system that we have proposed.

### A. Meeting the design goals

We evaluate how we have met our design goals stated in Section III.

1) *User-directed use of multiple networks*:
Fig. 6 and Fig. 7 show a runtime trace of the number of bytes that remain to be transferred for a single application during one emulation run, while using the simple algorithm of Section V-H. In both the figures, grey regions denote the

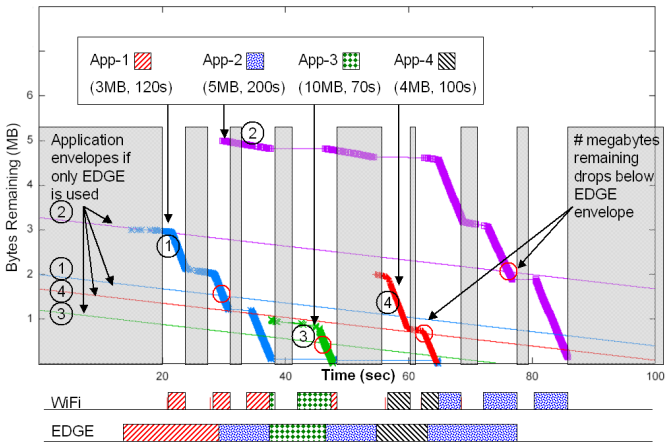


Fig. 8. Multiple applications

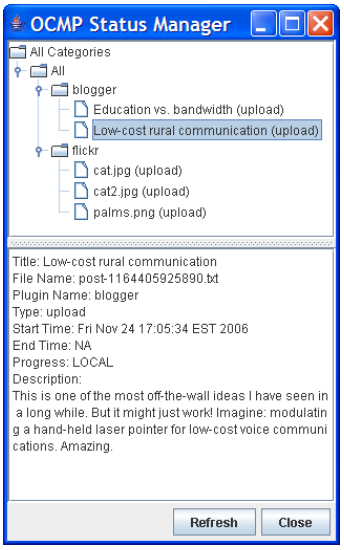


Fig. 9. Screenshot of OCMP status-manager

times when WiFi is absent. We have divided the timeline into multiple zones for clarity of explanation. We will use $r(t)$ to denote the number of bytes that remain to be transferred at time $t$. We call the scenario depicted in Fig. 6 as having an easy deadline because when the application starts, the size of its data to be transferred (1.5 MB) is less than the maximum amount that can be sent on EDGE even if no WiFi network shows up. The scenario in Fig. 7 is correspondingly termed as having an aggressive deadline because the amount of data to be transferred (3 MB) is larger than the maximum amount that can be sent on EDGE alone.

a) *Zone I (Idle):* This denotes an idle state, either when there is no application waiting for data delivery, or all the applications have completed their respective data transfers.
b) *Zone II (Do-nothing)*: This denotes the time during which an application is active, but there is no data transfer is progress because $r(t)$ is below the envelope. Thus, Zone II occurs in Fig. 6 when the application is started at a time that is much before the commencement time calculated assuming that no WiFi network will be available until the application deadline.
c) *Zone III (EDGE)*: This denotes the intervals during which there is no WiFi network available, and $r(t)$ will either cross the envelope if EDGE is not used, or is already above the envelope.
d) *Zone IV (WiFi)*: This denotes the short intervals of time when WiFi networks are available for opportunistic use. In Fig. 7, these opportunities present themselves when $r(t)$ is above the envelope; hence, both WiFi and EDGE are simultaneously used. However, only WiFi networks are used in Fig. 6 because $r(t)$ always stays at or below the envelope.

Fig. 8 shows a runtime trace for 4 applications running under the same policy enforcement algorithm, each application having a single data stream. All applications have been shown to have aggressive deadlines, where they start above the EDGE envelope that is available for them. Note that the applications are labeled according to their start times, but their deadlines are not in the same order: App-3 < App-4 < App-1 < App-2. At any time instant, the application having an earlier deadline is given preference over other applications. Thus, App-1 is allowed to use WiFi in preference to App-2 always, and App-3 preempts App-1 at $t = 38sec$ when App-3 starts and has an earlier deadline than App-1. The earliest-deadline-first ordering is also followed on EDGE. Thus, App-2 which has the last deadline, is able to use EDGE only when all other applications have either completed, or have dropped below their individual EDGE envelopes.

These experiments show that user and application-directed use of multiple interfaces is possible through OCMP.

2) *Ease of application design and implementation*: We believe we have successfully met this goal because both our plugin and directory APIs completely mask the effects of network disconnections and switching. As anecdotal evidence, a student without any knowledge of the underlying details of OCMP was able to develop an Email plugin in just 4 days, "simply by parsing /var/spool/mail and dumping the emails in the OCMP communication directory".
3) *Support for legacy servers*: We have so far built OCMP application plugins for the following services which are otherwise available only through applications that connect directly to the legacy servers hosting these services: send and receive email, upload photos to www.flickr.com, post blogs to www.blogger.com, and receive HTML pages using HTTP GET.

Fig. 9 shows a screenshot of the OCMP status manager that displays the status of data transfer for different applications. In this particular case, two blog entries for www.blogger.com and three images for www.flickr.com were uploaded using OCMP.

4) *Application session persistence across disconnections*: Instances such as the occurrence of Zone II in Fig. 6 and Fig. 7, when no networks are used, show that session persistence is supported in OCMP.
5) *Optimized network switching*: We stated earlier that in the absence of quick notifications of WiFi disconnections sent over the cellular control channel, the proxy will have

TABLE II
DISCONNECTION DETECTION LATENCIES

| | Mean | Std. deviation |
|---|---|---|
| With EDGE | 982ms | 156ms |
| Without EDGE | 49.2sec | 13.7sec |

to maintain a large amount of state until when a TCP timeout occurs. To observe this effect, we ran a set of 10 experimental runs both with and without an EDGE cellular channel, and calculated the mean and standard deviation of the delay incurred by the proxy in inferring a disconnection. We did this by manually disconnecting the laptop from WiFi and observed the delay until when the proxy closed the TCP connection. The results are shown in Table 2.

When EDGE is used as a control channel, the total delays are of the order of 1 sec. The primary component of this delay is the large RTT of an EDGE network ($\sim 700ms$). Note that in these measurements, we did not count the latency incurred by the 802.11 MAC layer to infer a disconnection and report it to OCMP. This is because such link layer indications are not yet a part of the 802.11 standard [5], although they are useful for opportunistic communication. For now, we assume that such link layer triggers exist, and disconnections can be detected by the mobile device within a couple of milliseconds. This is because most wireless cards consider three consecutive MAC retransmission failures as a disconnection [37], which can be done within a few milliseconds.

On the other hand, without an EDGE control channel, TCP timeouts often take over a minute to detect a broken connection. This means, for example, that the proxy maintains flow state over a period of minutes even though opportunistic WiFi network residence times will likely to be of the order of a few tens of seconds [41]. This observation is of significant concern because the OCMP proxy will likely be shared among hundreds of users, and redundant state maintenance can clearly decrease the scalability of the proxy. This also shows that link layer indications can prove helpful for opportunistic communication, and argues for the inclusion of link layer triggers as a part of the 802.11 standard.

6) *Buffered Access Point support*: We have extensively tested the integration of OCMP and DTN with encouraging results obtained on Soekris boxes running as access points. We also plan to repeat our experiments with off-the-shelf AP hardware. On a related note, we have deployed this system in a rural Internet kiosk in India since the OCMP-DTN integration makes our systems usable for mechanical backhaul [25].

### B. Performance comparison with standard FTP

We ran further experiments to compare the performance of OCMP with that of standard FTP to benchmark the efficiency of our implementation. We used a single WiFi network without any disconnections. The results are shown in Fig. 10. It can be seen that our implementation performs reasonably well for filesizes up to 30MB, but the performance drops for larger files. Preliminary observations from profiling tools indicate that the CPU is the bottleneck, most likely due to additional bundle processing in the OCMP scheduler. Although we expect most transfer sizes to be below 30MB, these experiments indicate that policy-based systems do incur greater processing requirements.

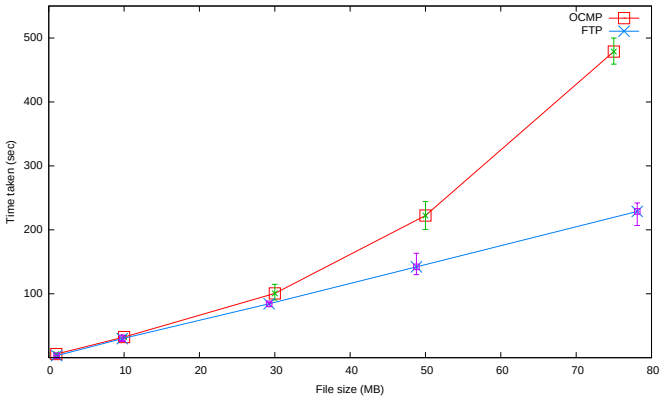


Fig. 10. Performance comparison with FTP

### C. Experience on hand-held mobile devices

We also ran OCMP on an iMate K-JAM having a 195MHz processor and 64MB RAM. Surprisingly, we were able to get a WiFi throughput of only 12KBps as opposed to an Internet Explorer HTTP download rate of approximately 60KBps. We believe that this is because OCMP is coded in J2ME and its performance depends heavily on the resources that are available to the Java virtual machine running on the mobile device. However, our observations reaffirm the previous conclusion that the benefits of policy-based systems can be offset by the greater processing requirements of such systems. Fortunately, the processing capabilities of devices are increasing rapidly, and policy-based systems can provide significant benefits.

## VII. FUTURE WORK

### A. Policy control

We are currently developing more sophisticated and optimal policies for network selection. Some factors we attempt to take into consideration are:

1) Users generally follow a certain mobility schedule; eg. leave home for office at 8:30am, travel along a particular road, return home at 5pm, etc. In the past, researchers have developed mobility models using Markov fields [11] and calendar schedules [12] that capture such regularity in user movements. Knowledge of user mobility patterns based on similar models can help us solve decision processes of policies for network selection. For example, in the context of the simple algorithm of Section V-H, if the device knows with a high probability that it will enter a WiFi network soon then even if the amount of data remaining to be transferred is currently above the EDGE envelope, the device need not use EDGE at all.
2) When a device is in the range of multiple wireless networks, it was shown that aggressive switching across these networks can actually degrade overall throughput because of the switching latencies associated with moving to a different network [28] . A *make-up time* metric was suggested as the minimum amount of time a mobile device must stay

connected to a network before switching to a different one. Knowledge of mobility patterns and expected residence times in different networks can help estimate such 'make-up time' metrics for optimized network selection.

3) During situations of data overload when a large amount of data is to be downloaded or uploaded, users may associate different preferences with different applications so that low priority applications can be temporarily dropped. To model such situations, we plan to associate each application with a utility function that decays with time. We can then use algorithms to maximize the overall system utility by scheduling different applications at different times on multiple wireless interfaces.

### B. Multiple proxies

We mentioned in Section IV-B the need to relocate application state to the proxy 'closest' in terms of RTT to a mobile device. Since TCP throughput is inversely proportional to the RTT [14], this optimization can improve the utilization of end-to-end opportunistic connections between the mobile device and a proxy [23]. We plan to implement inter-proxy state transfer by arranging globally distributed OCMP proxies in an I3 overlay [13]. Mobile devices will be able to register themselves with the closest OCMP proxy (an I3 overlay node), and application state for the mobile device will be automatically moved between proxies whenever a new registration is made. Note that we expect OCMP proxies to have fairly large footprints [24], and therefore selection of the closest proxy need not be done on the order of every minute or even every hour.

## VIII. Related Work

We are not aware of any other work that provides the same set of functionality provided by our system. However, our work is closely related to, and builds on the insights of several threads of past work in this area, as described next.

Policy-based selection of network interfaces was first introduced in [29], and has since been extensively explored in the context of vertical handoffs by researchers [30], [31]. The problem was motivated by the desire to choose the best network that optimizes metrics such as data rates or power consumption in the long term, while preserving seamless connectivity. We have built upon this definition by noticing that seamless connectivity is not required for many non-interactive applications. Therefore, our policy definitions operate at the session layer, unlike the network- and transport-layer policies of past work. This allows us to include a delay component (such as a deadline for data transfer) at a timescale of minutes and hours, which fundamentally alters the system.

Our use of many wireless interfaces in parallel is similar in spirit to pTCP [16]. Unlike pTCP, which assumes an underlying TCP connection, OCMP is transport-agnostic. We are able to make this tradeoff because we are primarily interested in delay-tolerant applications that can deal with a large resequencing buffer. In contrast, pTCP supports interactive applications, and must therefore exploit TCP structure to reduce the size of the resequencing buffer. Unlike pTCP, we have built, deployed and evaluated the performance of the system in a testbed, instead of relying on simulations.

The use of location-independent identifiers for resuming a session was proposed earlier in the context of Rocks-and-Racks [38] and TCP Migrate [27]. However, these earlier solutions are not only TCP-centric but also support only a single interface. Our use of an almost-always-available cellular connection for the transmission of control messages (i.e. data available, and link down) distinguishes us from these proposals. Also, unlike these proposals, we have designed and implemented a session-level reliability protocol.

Our use of a proxy for dealing with session disconnections and the aggregation of multiple transport connections into a single connection is similar to that proposed in PCMP [22]. However, OCMP differs from PCMP in several ways. First, unlike PCMP, OCMP supports the use of multiple NICs in parallel. We also support arbitrary transport protocols including UDP with erasure codes [6], [7], and TCP optimized for cellular networks [8], [20], whereas PCMP is essentially TCP-centric. Unlike PCMP, OCMP nodes can be powered down because application data and control is persistently stored. Our architecture allows session state to be encapsulated and transferred from one proxy to another. This allows us to reassign a mobile to the closest available proxy, greatly improving performance. Finally, servers can push data to OCMP proxies, or plugin daemons can poll legacy servers to pull data, and the data can then be picked up opportunistically by mobile devices. We believe that these differences make OCMP much more suited to a dynamic multi-network environment than PCMP.

Our proposal to use multiple OCMP proxies for better throughput during opportunistic connections is similar to that proposed in DHARMA [3]. DHARMA uses a network of distributed *home-agents* to provide (a) mobile IP like packet forwarding support when client IP addresses change due to mobility, and (b) session persistence for preserving TCP connections across disconnections. We believe that our work is more general because it provides many more features than just session persistence and mobility support.

Intelligent selection of network interfaces with session persistence is also being explored in the context of the Haggle project [17]. However, Haggle is focused on infrastructure-less systems where devices communicate with each other in an ad-hoc manner. Further, they do have a notion of application plugins, proxies, or a control channel.

Finally, our work is complementary to, and extends, recent work in the area of implementing a router for delay tolerant networks (DTN) [9]. Our notions of session persistence, data persistence, bundling, and multi-network support originate in this seminal work. However, we have made several non-trivial extensions. These include the support for fine-grained policy control, the notion of application plugins, the use of a proxy, and the separation of the data and control planes. Some of our detailed design decisions also differ from that made in the DTN reference implementation. For instance, DTN associates a 'convergence layer' with each transport protocol, which means that all NICs that support TCP would use the same convergence layer. In contrast, we associate a connection with each NIC, allowing us to exploit network heterogeneity by optimizing different transport layer implementations for different types of networks [6]–[8], [20]. OCMP also supports a control channel to communicate disconnection and reliable data transfer information between peers, which is not currently possible in DTN. We observe that our work is motivated by a narrower set of problem areas than DTN, which allows us to exploit the inherent problem structure

to make these optimizations.

## IX. Discussion and conclusions

We have designed and implemented an architecture for session-layer policy-based selection of wireless networks for opportunistic communication. We have shown that our system meets all the design goals captured by the use-cases for opportunistic communication, and we have highlighted some interesting avenues for future research. We believe that our system provides a broad foundation for building and studying systems that incorporate smart mobile devices in an environment with multiple wireless networks. Our system is not only usable on smartphones, but it is also highly applicable for rural Internet kiosks [25].

There are many open areas of research related to opportunistic communication. How can a mobile device detect networks in a power efficient manner, without having to keep its radio switched on all the time? What is the interaction between TCP and lower layers during an opportunistic connection interval? What kind of transport layer implementations can efficiently handle these brief connection opportunities? How should wireless devices be designed to make opportunistic communication more power efficient? Lastly, what kind of a security scheme should be used for opportunistic communication such that protocol handshakes are minimized? We plan to address these issues in future work.

## References


[1] T. Armstrong, O. Trescases, C. Amza, E. Lara, "Efficient and Transparent Dynamic Content Updates for Mobile Clients," Proc. ACM/USENIX MOBISYS, Jun 2006.

[2] E. Shih, P. Bahl, M. Sinclair, "Wake on Wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices," Proc. ACM MOBICOM, 2002.

[3] Y. Mao, B, Knutsson, H. Lu, J. Smith, "DHARMA: Distributed Home Agent for Robust Mobile Access," Proc. IEEE INFOCOM, 2005.

[4] H. Balakrishnan, V. Padmanabhan, S. Seshan, R. Katz, "A Comparison of Mechanisms for Improving TCP Performance Over Wireless Links," In IEEE/ACM Transactions on Networking, Vol. 5, No. 6, 1997.

[5] B. Aboba, "Architectural Implications of Link Layer Indications," http://www.ietf.org/internet-drafts/draft-iab-link-indications-01.txt, Jan 2005.

[6] R. Chakravorty, A. Clark, I. Pratt, "GPRSWeb: Optimizing the Web for GPRS Links," Proc. ACM/USENIX MOBISYS, 2003.

[7] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A Digital Fountain Approach to Reliable Distribution of Bulk Data," Proc. ACM SIGCOMM, 1998.

[8] M. Chan and R. Ramjee, "Improving TCP/IP Performance Over Third Generation Wireless Networks," Proc. IEEE INFOCOM, 2004.

[9] M. Demmer, E. Brewer, K. Fall, S. Jain, M. Ho, and R. Patra, "Implementing Delay Tolerant Networking," Intel Research, Berkeley, Technical Report, IRB-TR-04-020, Dec 2004.

[10] S. Jain, K. Fall, and R. Patra, "Routing in a Delay Tolerant Network," Proc. ACM SIGCOMM, 2004.

[11] L. Song, D. Kotz, R. Jain, and X. He, "Evaluating Location Predictors with Extensive WiFi Mobility Data," Proc. IEEE INFOCOM, 2004.

[12] V. Srinivasan, M. Motani, and W. Ooi, "Analysis and Implications of Student Contact Patterns Derived from Campus Schedules," Proc. ACM MOBICOM, 2006

[13] S. Zhuang, K. Lai, I. Stoica, R. Katz, S. Shenker, "Host Mobility using an Internet Indirection Infrastructure," Proc. ACM/USENIX MOBISYS, 2003.

[14] J. Kurose and K. Ross, "Computer Networking," Addison Wesley, 3rd Edition, pg. 271, 2004.

[15] K. Fall, "A Delay-Tolerant Network for Challenged Internets," Proc. ACM SIGCOMM 2003.

[16] H. Hsieh and R. Sivakumar, "A Transport Layer Approach for Achieving Aggregate Bandwidths on Multi-homes Mobile Hosts," Proc. ACM MOBICOM, 2002.

[17] J. Scott, J. Crowcroft, P. Hui, C. Diot, "Haggle: A Networking Architecture Designed Around Mobile Users," Conference on Wireless On-demand Network Systems and Services (WONS), 2006.

[18] M. Kozuch and M. Satyanarayanan, "Internet suspend/resume," In Workshop on Mobile Computing Systems and Applications, 2002.

[19] J. Kurose and K. Ross, "Computer Networking," Addison Wesley, 3rd Edition, pp.271, 2004.

[20] R. Ludwig and R. Katz, "The Eifel Algorithm : Making TCP Robust Against Spurious Retranmission," In ACM Computer Communication Review, January 2000.

[21] R. Moskowitz, P. Nikander. P. Jokela, T. Henderson, "Host Identity Protocol," http://www.potaroo.net/ietf/ids/ draft-ietf-hip-base-00.txt, 2004.

[22] J. Ott and D. Kutscher, "A Disconnection-Tolerant Transport for Drive-thru Internet Environments," Proc. IEEE INFOCOM 2005.

[23] A. Seth, P. Darragh, S. Liang, Y. Lin, S. Keshav, "An Architecture for Tetherless Communication," http://blizzard.cs.uwaterloo.ca/keshav/home/Papers/data/05/tca.pdf, Manuscript, University of Waterloo, July 2005.

[24] A. Seth, S. Bhattacharya, S. Keshav, "Opportunistic Communication Over Heterogeneous Access Networks," http://www.cs.uwaterloo.ca/ a3seth/ocmptech.pdf, Technical report, Sprint Labs, CA, April 2005.

[25] A. Seth, D. Kroeker, M. Zaharia, S. Guo, and S. Keshav, "Low-cost Communication for Rural Internet Kiosks Using Mechanical Backhaul," Proc. ACM MOBICOM, 2006.

[26] A. Seth, "The Use of Mobile Devices as Sensors for Efficient Wireless Monitoring and Resource Utilization," http://www.cs.uwaterloo.ca/ a3seth/cellphones.pdf, Manuscript, University of Waterloo, 2006.

[27] A. Snoeren and H. Balakrishnan, "An End-to-End Approach to Host Mobility," Proc. ACM MOBICOM, 2000.

[28] M. Stemm and R. Katz, "Vertical Handoffs in Wireless Overlay Networks," In Mobile Networks and Applications, Volume 3, Number 4, Pages 335-350, 1998.

[29] H. Wang, R. Katz, and J. Giese, "Policy-Enabled Handoffs across Heterogeneous Wireless Networks," In Mobile Computing Systems and Applications, 1999.

[30] F. Zhu and J. McNair, "Optimizations for Vertical Handoff Decision Algorithms," Proc. WCNC, 2004.

[31] T. Pering, Y. Agarwal, R. Gupta, R. Want, "CoolSpots: Reducing Power Consumption Of Wireless Mobile Devices Using Multiple Radio Interfaces," Proc. ACM/USENIX MOBISYS, 2006.

[32] A. Nicholson, Y. Chawathe, M. Chen, B. Noble, D. Wetherall, "Improved Access Point Selection," Proc. ACM/USENIX MOBISYS, 2006.

[33] M. Sloman, "Policy Driven Management for Distributed Systems," In Journal of Network Systems Management, Vol. 2, No. 4, 1994.

[34] L. Kagal, "Rei: A Policy Language for the Me-Centric Project," http://www.hpl.hp.com/techreports/2002/HPL-2002-270.pdf, Technical report, HP Labs, 2002.

[35] A. Qureshi and J. Guttag, "Horde: Separating Network Striping Policy from Mechanism," Proc. ACM/USENIX MOBISYS, 2005.

[36] V. Vanghi, A. Damnjanovic, B. Vojcic, "The CDMA2000 System for Mobile Communications," Prentice Hall, 1st Edition, pp.224, 2004.

[37] H. Velayos, "Autonomic Wireless Networking," Doctoral thesis, TRITA-S3-LCN-0505, ISSN 1653-0837, ISRN KTH/S3/LCN/–05/05–SE, Stockholm, Sweden, May 2005.

[38] V. Zandy, and B. Miller, "Reliable Network Connections," Proc. ACM MOBICOM 2002.

[39] A. Garai and B. Shadrach, "Taking ICT to Every Indian Village," One World South Asia, New Delhi, India, 2006.

[40] D. Kutcher and J. Ott, "Service Maps for Heterogeneous Network Environments," Proc. Mobile Data Management Conference, 2006.

[41] V. Bychkovsky, B. Hull, A. Miu, H. Balakrishnan, and S. Madden, "A Measurement Study of Vehicular Internet Access Using In Situ Wi-Fi Networks," Proc. ACM MOBICOM, 2006.

[42] "Soekris Net4801," http://www.soekris.com/net4801.htm, 2005.

[43] "Cellular Data Plan Comparison Chart," http://www.jiwire.com/cellular-data-cellular-data-the-plans.htm, 2005.

[44] "Bytemobile," http://www.bytemobile.com, 2006.