# Intentional Networking: Opportunistic Exploitation of Mobile Network Diversity

Brett D. Higgins*, Azarias Reda*, Timur Alperovich*, Jason Flinn*,

T. J. Giuli†, Brian Noble*, and David Watson†

University of Michigan* Ford Motor Company†

## ABSTRACT

*Mobile devices face a diverse and dynamic set of networking options. Using those options to the fullest requires knowledge of application intent. This paper describes Intentional Networking, a simple but powerful mechanism for handling network diversity. Applications supply a declarative label for network transmissions, and the system matches transmissions to the most appropriate network. The system may also defer and re-order opportunistic transmissions subject to application-supplied mutual exclusion and ordering constraints. We have modified three applications to use Intentional Networking: BlueFS, a distributed file system for pervasive computing, Mozilla's Thunderbird e-mail client, and a vehicular participatory sensing application. We evaluated the performance of these applications using measurements obtained by driving a vehicle through WiFi and cellular 3G network coverage. Compared to an idealized solution that makes optimal use of all aggregated available networks but that lacks knowledge of application intent, Intentional Networking improves the latency of interactive messages from 48% to 13x, while adding no more than 7% throughput overhead.*

## General Terms

Performance

## Categories and Subject Descriptors

D.4.4 [**Operating Systems**]: Communications Management—*network communication*; D.4.8 [**Operating Systems**]: Performance

## Keywords

Wireless network selection, application-aware networking

## 1. INTRODUCTION

Mobile devices face a diverse, dynamic array of networking options. Increasingly, these options have a wide variety of strengths and weaknesses. As a result, there is no single "best choice" in all cases, and such diversity of infrastructure is both a challenge and an opportunity. The challenge lies in managing these changing options to best meet each application's needs, which themselves vary with time. However, by doing so, we can provide significant benefits to applications, exploiting multiple networks concurrently and planning future transmissions intelligently. This is particularly valuable for applications with a mix of on-demand and *opportunistic* network activity—messages that still have value even if deferred for a time.

Unfortunately, current approaches to this problem are insufficient. At one extreme, the operating system or a middleware layer makes all routing and interface decisions on behalf of applications, in a one-size-fits-all solution [6, 18]. However, because the entities that make these decisions are ignorant of the intent of the applications that are using the network, they often miss opportunities for optimization. Worse, in an effort to preserve current wired-network semantics, persistent connections generally end up "stuck" on wide-area (but low-performing) networks. At the other extreme, the system makes applications aware of network changes by exposing the low-level details directly to them [7, 34], and applications must explicitly choose among the available options. This approach is expressive, but neither simple nor elegant; managing multiple wireless networks unnecessarily complicates the task of the application writer.

*Intentional Networking* occupies the middle ground between these two extremes. In our approach, the system manages most of the messy details of discovering and characterizing available network options. Applications provide hints about traffic semantics using a small number of declarative *labels* to express intent. For instance, a label might differentiate between foreground traffic (e.g., a GUI-initiated request for which a user is waiting) and background traffic (e.g., an opportunistic message that need not happen at any particular time). The system then matches network traffic to available interfaces in an informed way.

Application data sent using different networks may arrive out of order. Constraining data delivery to follow in-order TCP-style semantics could dramatically limit the benefit seen by applications, since short, interactive messages would queue behind all previous opportunistic transfers. Thus, Intentional Networking allows applications to express relaxed ordering constraints for data delivery. The scheduling constraints for mobile network usage are similar to synchronization primitives used by threads running on a multi-processor. Based on this observation, we provide two primitives: Isolated Reliable Ordered Bytestreams (IROBs), which provide the mutual exclusion synchronization of mutex locks, and ordering constraints, which provide the must-happen-before synchronization of condition variables.

Finally, there are times when none of the currently available network options are appropriate and network traffic is best deferred.

For this scenario, Intentional Networking supports a *thunk* model of delayed execution in which the application registers a callback function to be invoked when circumstances change so that it becomes appropriate to transmit data with the specified label. Thunks let applications coalesce, rather than defer, redundant network messages; for instance, an e-mail client that periodically checks for new mail can send only one such request when an appropriate network becomes available.

The contribution of our work comes from defining simple and powerful abstractions for exposing the presence of multiple wireless networks to applications. Our work does not define a new over-the-wire protocol, but instead provides a portable, user-level implementation that routes traffic over appropriate networks based on application hints. We show that, for many applications, application-aware network selection outperforms even idealized aggregation strategies that lack knowledge of application intent.

We have modified two existing applications to use Intentional Networking: BlueFS [32], a file system for pervasive computing, and the Mozilla Thunderbird [27] open-source email client. We have also created a new automotive participatory sensing application that uses our API. We evaluated the performance of these applications using measurements obtained by driving a vehicle through WiFi and cellular 3G network coverage. Compared to an idealized solution that makes optimal use of the aggregated available networks but lacks knowledge of application intent, our results show that Intentional Networking improves the latency of interactive messages from 48% to 13x for our three applications, while adding no more than 7% throughput overhead.

## 2. RELATED WORK

There is a large body of work that seeks to route network traffic over multiple interfaces. Prior work largely falls into one of two categories: *application-oblivious*, in which the network over which data is sent is chosen based on system-wide goals such as maximizing throughput and without consideration of application intent, and *application-alone*, in which each application must manage the details of selecting among multiple networks on its own and the system's role is only to expose the details of possible options to the application.

Virtual WiFi [7] is one application-alone solution. It virtualizes a device's wireless interface, fooling applications into believing the device is connected simultaneously to different APs on different channels. This is a step in the right direction, because devices can now exploit all available connectivity in their vicinity. Unfortunately, Virtual WiFi places the burden of access point selection entirely on the application. In contrast, Intentional Networking presents applications and users with a single unchanging network interface that accepts declarative intent.

Application-oblivious systems are more numerous. FatVAP [18] presents an infrastructure similar to that of Virtual WiFi, but operates only within a single layer of an overlay network, and is concerned only with maximizing overall throughput, without concern for other application-level preferences. Other systems attack the bandwidth aggregation problem by designing new multi-path transport protocols to replace TCP, such as R-MTP [24], pTCP [15], mTCP [49], and SCTP [45]. SCTP also supports multi-streaming of independent byte streams; in contrast, Intentional Networking allows applications to specify ordering and atomicity constraints over data sent to a destination computer. Multi-path transport has also been built into the kernel socket-handling functions just above the transport layer [39]. Chebrolu et al. [8] use a modified network layer at the mobile host and at a remote proxy to hide the use of multiple networks, and the resulting reordering of packets, from the transport and application layers. Though all of the above application-oblivious systems are simple for applications to use, they only focus on throughput maximization and cannot take into account other application-specific or request-specific goals such as minimizing latency.

In contrast to application-oblivious and application-alone strategies, Intentional Networking splits the burden of network selection among applications and the system. Applications disclose qualitative hints about their intentions in using the network, and the system reasons about how traffic labeled with those hints should be mapped to specific networks based on their current characteristics.

Rather than target throughput maximization, Wiffler [3] opportunistically routes data over WiFi to minimize cellular usage. Others [8, 38, 48] have argued that throughput maximization is not the only goal of interest to mobile applications and users, and that the ability to specify network usage policies on a per-application basis would be useful. We differ from these prior works in two ways. First, we argue that the application, not the user, should set policies. Application network usage patterns may change quickly, and the proper choice of policy changes likewise; it would place too great a burden on the user to understand their applications' behavior and constantly update the policies. Second, we propose, implement, and evaluate a specific mechanism for applications to set fine-grained policies by describing the intent of each network message.

The push toward ubiquitous computing makes automatic service discovery in new environments more important than ever [41]. Existing work, however, has focused more on enabling application-level services [9, 13, 43] than on choosing and managing a diverse set of network connections from an application's point of view.

Several systems seek to allow clients of one wireless service provider to access foreign wireless hotspots when roaming [5, 11, 25, 40] or between public and private networks [26]. Our work is complementary, since users must find and associate to an access point before negotiating such roaming agreements. This service discovery is similarly critical for grassroots wireless collective initiatives [4, 35, 42].

Contact Networking [6] hides the differences between local and remote communication from users. All communication appears to be local—like a direct Bluetooth connection between two devices—even if infrastructure such as the Internet is actually involved. Like us, the authors recognize that mobile devices typically have several heterogeneous wireless radios at their disposal. Contact Networking is also conscious of the properties of different link layers. Their primary focus, however, is on neighbor discovery, name resolution, and (ultimately) the preservation of application-level sessions in the face of user mobility. Our work does find common ground with the idea that all network connectivity options are not equivalent and the operating system should dynamically assign data flows to the most appropriate link.

Zhao et al. [50] attack problems similar to those addressed by Contact Networking. Their work lies firmly within the framework of Mobile IP [37] as well. The user's Home Agent is required to arbitrate the routing of various data flows. Further, applications must explicitly bind a data flow to a specific interface through their `SO_BINDTODEVICE` socket option. We propose a decentralized solution and envision the operating system automatically assigning flows to the optimal interface, aided at most by simple hints from applications.

Much recent work has argued that the multiple network connectivity options available to today's mobile devices are a blessing, not a curse. Johansson et al. [17], among others, show how Bluetooth radios are often preferable to IEEE 802.11 for short-range, low-power communication. Bahl et al. [2] illustrate scenarios where multiple radios can help devices save energy, enhance their data communication capacity, make wireless AP handoff more seamless, and better tolerate wireless link problems. Draves et al. [10] show how overall throughput can be increased for multi-radio nodes in mesh networks by dynamically choosing the "best" outbound link when forwarding a given packet. Stemm and Katz [44] recognize the hierarchical nature of overlapping wireless networks. Much like cache hierarchies in computer architecture, multiple wireless networks commonly cover one spot, with the utility (e.g., bandwidth) of a network usually inversely proportional to its coverage radius.

Labels are partially inspired by the use of hints to guide power management decisions in STPM [1]. Both projects share the goal of having applications disclose a minimal amount of information to guide resource management decisions. Yet, the domains to which these hints are applied are very different. STPM sets wireless network power management modes, while Intentional Networking changes the scheduling and routing of network messages.

## 3. DESIGN GOALS

We next list the major goals that drove the design and implementation of Intentional Networking.

### 3.1 Separate concerns

Our design is guided by the classic principle of separating policy and mechanism. Applications are best situated to determine the actual intent in using the network, e.g., whether a particular message is driven by interactive use or whether it is background traffic. This intent represents the policy for how data should be transmitted.

On the other hand, the operating system or a middleware library is best positioned to provide a common mechanism to implement the specified policies. A common mechanism makes deploying new applications that use multiple mobile networks considerably easier since each application must only provide hints as to its intent. The details of handling multiple heterogeneous and intermittent mobile networks is encapsulated at lower layers of the system. A common mechanism can also aggregate heterogeneous data transmissions from multiple applications.

Thus, Intentional Networking is designed to have a separation of concerns in which applications disclose policy decisions by labeling the data they transmit and a lower layer of the system implements the mechanism that enacts the policy by mapping data to the networks that best match the labels at the time the data is transmitted.

### 3.2 Be Qualitative

Our design is also guided by the classic principle of keeping the interface as simple as possible, without unduly sacrificing expressiveness. This has resulted in a minimalist, qualitative interface. For instance, we could have required each application to disclose detailed quantitative specifications of the characteristics of the traffic it expects to generate, as well as the quality of service that it requires. However, such a complex interface would place a considerable burden on the application programmer, that of carefully tuning for each possible workload, making it unlikely that the casual developer would use our system.

This principle led to several decisions. Rather than use quantitative specifications, applications express their intentions using only qualitative attributes over the data; i.e., whether a transmission will be small or large, and whether it is interactive or background traffic. We do not mandate what constitutes "small" vs "large". We allow the application to use these labels as it sees fit. While we may eventually add more attributes to our labels as our experience with the system grows, the current interface is sufficiently expressive to handle several complex applications, as discussed in Section 6.

| Properties | Possible values |
|---|---|
| Interactivity | Foreground vs. Background |
| Size | Small vs. Large |

**Table 1: Intentional Networking label properties**

### 3.3 Embrace Concurrency and Failure

Our original goal for Intentional Networking was to provide a single-socket abstraction that assigns labeled traffic to the most appropriate networking option. However, single-socket semantics require data to be delivered in-order for TCP connections. Unfortunately, this severely limits the set of optimizations possible when using multiple networks simultaneously.

After several false starts, it became clear to us that going from one to many networks is akin to the transition from single-threaded programming to multi-threaded programming. Some interleavings of execution orders are very useful and desirable, but others lead to incorrect computations.

Just as concurrent systems include mechanisms to allow the programmer to rule out incorrect orderings, we added synchronization abstractions to express both atomicity and happens-before constraints. These mechanisms are both simple and expressive, and are familiar concepts to programmers with training in monitor-style concurrency control.

In addition to expressing such ordering constraints, we also needed mechanisms to deal with partial failure. There are times when some traffic would be ill-served by any available transmission alternative. Therefore, we provided a callback mechanism—similar to exceptions or continuations—to handle delayed transmissions or disconnections.

## 4. ABSTRACTIONS AND INTERFACE

In this section, we describe the Intentional Networking application interface. We first describe the basic abstractions in the interface. Applications use *labels* to communicate their intent. These are meaningful in the context of *multi-sockets* and are expressed over message units called *IROBs (Isolated Reliable Ordered Bytestreams)*. IROBs provide atomicity (mutual exclusion); applications may also specify *ordering constraints* among IROBs. When operations must be deferred, applications may register *thunks* to resume them. After describing these fundamental abstractions, we show the Intentional Networking API in Section 4.6.

### 4.1 Labels

The label is the principal abstraction available to applications. It is the mechanism by which applications declare the properties of any particular network message. Labels are system-defined qualitative properties of the message. Our present implementation supports only four labels across two dimensions, interactivity and size, as shown in Table 1. A message's label is set to foreground if a user-visible event is waiting for the response. A message is background if its timely delivery is not critical to correct behavior. For example, many hints [46] need not be sent. The small label describes messages that are latency-dominated such as single-packet RPCs, while the large label describes other messages such as those containing multimedia data. We expect to add further dimensions and label values as our experience with applications grows. Yet, the eventual number of possible label values will remain small since interface simplicity is one of our main design goals.

## 4.2 Multi-Sockets

Labels are used in conjunction with label-aware sockets. We call such sockets *multi-sockets*. Intuitively, a multi-socket multiplexes several different labels across a single virtual socket. For the most part, multi-sockets behave exactly as normal ones do. However, multi-socket send calls take a label that is used to assign packets to the best possible interface. Note that the sender is always the entity responsible for assigning labels, and as a consequence, `recv` does not require a label. While we could imagine using one to implement a filtered receive, we have not had to do so for any of our applications so far.

A multi-socket is a single logical connection that dynamically instantiates and uses actual TCP connections over one or more physical interfaces. Multi-sockets provide encapsulation: they hide the presence of multiple network interfaces, routes, and connections from applications. Multi-sockets also encapsulate transient disconnections caused by events such as passing through a wireless dead zone. Applications specify only labels, which are used by the Intentional Networking traffic manager to choose the right network over which to send data. Applications may optionally be notified about network unavailability on a *per-label*, not per-network basis, through the use of thunks, which are deferred execution environments that execute when an event occurs. Thunks are described in more detail in Section 4.5.

Like TCP sockets, multi-sockets support a reliable delivery abstraction. However, multi-sockets relax TCP's ordering constraints by allowing bytes to be reordered subject to application-specified mutual exclusion and ordering constraints, as described in the next two sections.

## 4.3 IROBs

An IROB is the unit of network transmission to which labels are applied. The multi-socket interface guarantees that each IROB is received atomically; i.e., the bytes of the IROB are produced in order without intervening bytes from other network transmissions. However, individual IROBs may be reordered with respect to one another. In other words, an IROB sent after a previously sent IROB may be seen first by the application reading data from the receiving multi-socket. Yet, bytes from the two IROBs will never be intermingled. IROBs thus provide *mutual exclusion* in the same manner that locks provide mutual exclusion for threads in a multithreaded program.

## 4.4 Ordering constraints

Since some applications require *ordering constraints* between IROBs, the multi-socket interface supports the declaration of such constraints. Each multi-socket assigns a unique, monotonically increasing identifier to each IROB. When creating a new IROB, the application may specify the identifiers of any IROB that must be received prior to receiving the one being created. Ordering constraints may only specify IROBs that have a lower unique identifier; this guarantees that such constraints are deadlock free. Applications that desire the sequential byte stream of a TCP socket specify that each IROB must be received after the one with the next lowest identifier; our API provides default send calls with this behavior for simplicity. However, many of our applications have looser constraints; for instance, the BlueFS file system client requires that asynchronous writes be ordered sequentially with respect to one another, but allows them to be arbitrarily ordered with respect to all other RPC types. The ordering constraints in multi-sockets are similar to those provided by condition variables for threads in a multi-threaded program.

## 4.5 Thunks

It is possible that a labeled IROB may not have any "appropriate" network available at the time it is sent. For example, consider an opportunistic bulk transfer initiated when only a low-bandwidth link is available. Such a transfer would preferably be done at a later time, when a high-bandwidth link is encountered. Alternatively, the mobile computer may be in a wireless dead-zone, with no connectivity.

Naturally, we do not want applications to have to poll for such a link. We also do not want applications to have to establish new connections after short periods of transient disconnection. However, in keeping with our design goals, we want to expose such events to applications when appropriate.

In our interface, the operations that create IROBs take an optional *thunk* argument, which is a function/argument pair that will be used to inform the application about IROBs that cannot be immediately sent due to the lack of an appropriate network. When an IROB is deferred, the call that takes the thunk argument returns a special return code. Later, when data with the specified label can next be transferred, the library notifies the application by calling the thunk function with the specified arguments. The ownership of the argument's resources passes with the thunk, and the handler must take responsibility for them. Thunks may be canceled—for example, if a subsequent `send` would invalidate a prior thunked one.

Thunks are useful for applications that send periodic messages, such as checking for new e-mail. Buffering redundant messages during disconnected periods and sending them all later is clearly undesirable. Instead, such applications register a thunk for the send and are notified when an appropriate network is available. The thunk handler sends only one polling request, thereby preserving valuable network bandwidth.

## 4.6 API

Table 2 shows the most important functions in the Intentional Networking API. The `ms_socket` call creates a new multi-socket, and the `ms_connect` call connects it to a remote endpoint, which is specified in the same way as for the `connect` system call. Thus, the only difference between `ms_connect` and the standard `connect` system call is that the first argument is a multi-socket.

Typically, we modify an application by replacing the socket, connect, listen, and accept calls with their `ms_*` counterparts. Applications create a new IROB through `ms_begin_irob`, passing a label that describes the atomic message, as well as any ordering constraints. This function also takes an optional thunk and data to be passed to the thunk function. The application then calls `ms_irob_send` to specify the data sent as part of the IROB; typically, we perform a one-to-one replacement of `send` to `ms_irob_send` calls. The application uses `ms_end_irob` to tell the library that no more data will be sent for the IROB. The `ms_send` call is provided as a convenience; it creates a new IROB that depends on all previous IROBs, specifies the data that comprises the IROB, and ends the IROB. If an application uses just `ms_send` calls, it will provide the behavior of TCP with labels, though no reordering will occur.

The `ms_recv` call returns a label. This is useful for server applications that wish to reply to a client request using the same label provided by the client for the original request. For instance, an IMAP server may wish to reply to client background requests with a background label and reply to foreground requests with a fore-

| Function | Arguments and return values |
|---|---|
| `ms_socket` | `(IN family, IN type, IN protocol, OUT multi-socket);` |
| `ms_begin_irob` | `(IN multi-socket, IN label, IN dependencies, IN thunk, IN thunk_data, OUT irob_id);` |
| `ms_irob_send` | `(IN irob_id, IN buf, IN length, IN flags, OUT bytes_sent);` |
| `ms_end_irob` | `(IN irob_id);` |
| `ms_send` | `(IN multi-socket, IN buffer, IN length, IN flags, IN label, IN thunk, IN thunk_data, OUT bytes_sent);` |
| `ms_recv` | `(IN multi-socket, IN buffer, IN length, IN flags, OUT label, OUT bytes_rcvd);` |

This figure shows the Intentional Networking API for creating and using multi-sockets. Besides the functions shown, multi-sockets also support the traditional socket functions; e.g., `accept`, `select`, and `setsockopt`.

**Table 2: Intentional Networking API**

ground label. Although not shown in Table 2, multi-sockets export similar functions to those provided by traditional sockets such as `listen`, `accept`, `select`, and `setsockopt`.

## 4.7 Discussion

It is useful to consider what an application would need to provide on its own to achieve application-aware functionality equivalent to Intentional Networking. First, an application would need to discover new network options, open sockets for each network option, and monitor the connection quality of each network in order to decide which network to use for each transmission. To prioritize on-demand traffic, the application might create multiple sockets per network, then use a platform-specific method to prioritize traffic from one socket over the other. The application would also need to stripe traffic across connections to improve throughput, then manage the inevitable re-ordering of data that arises from such striping. Finally, the application might poll to achieve the functionality of thunks that allows traffic to be altered or dropped if an appropriate network is not currently available.

In contrast, the Intentional Networking abstraction makes this functionality the responsibility of the lower layer of the mobile system, not the application. The application need only annotate its traffic with the simple API in Table 2 to achieve the same functionality. While strategies that ignore intent can be implemented without application modification, our evaluation shows that such application-oblivious strategies substantially underperform Intentional Networking.

## 5. ARCHITECTURE

When we began our work, we faced a decision about whether to implement Intentional Networking at user-level or in the kernel. Good reasons exist for both choices. A kernel implementation can improve performance by integrating tightly with the network stack. However, we decided to implement our initial prototype at user level to provide portability and simplify deployment. Given the wide array of operating systems used by mobile computers and cell phones, a user-level implementation is much easier to port to new platforms. Further, many popular mobile platforms do not allow kernel modifications at all. Even with a user-level implementation, our prototype performs well, as shown in Section 7. Our implementation consists of a connection scout daemon that runs on the mobile client, plus a library implementing the API.

## 5.1 Connection scout

The connection scout is a stand-alone user-level process, which we have adapted from the implementation of Virgil [30]. It is responsible for discovering and evaluating the performance of the networking options available at any given time. For each of the mobile computer's wireless network interfaces, the connection scout periodically attempts to establish network connections. After a connection is established, the scout measures the throughput and latency of the connection through active probing. The multi-socket library queries network availability and performance data from the scout using a pipe.

We envision that the connection scout could eventually leverage a lower layer that allows a mobile computer to simultaneously connect to multiple access points via a single physical interface [7, 31] by having the lower layer expose each access point as a separate virtual interface.

## 5.2 Library

The Intentional Networking library exports the interface described in Section 4.6. It is responsible for mapping IROBs to interfaces based on their associated labels. For each multi-socket, the library dynamically creates separate TCP sockets for each interface over which it decides to send data. A multi-socket connection persists until no TCP connection can be maintained using any network interface (for example, if the mobile computer moves out of range of a WiFi access point and no other network options are available) or the multi-socket is closed.

We chose to use TCP primarily for simplicity. Since we are not designing a new over-the-wire protocol, TCP's reliability mechanisms limit the amount of effort we must spend implementing ordered delivery of bytes within an IROB or retransmission of bytes lost due to congestion in the network. For the purposes of our prototype and evaluation, we have not found TCP to be a significant source of overhead, but we imagine that a more highly tuned implementation of Intentional Networking would integrate more tightly with the transport layer for optimal performance.

When an initial connection is established over the first TCP socket, a mobile client sends its peer data that includes its available IP addresses and the estimated bandwidth and latency for each one. It piggybacks updates to this information on Intentional Networking headers, as described below. With this information, either peer may establish a new TCP connection when it expects that a new connection would be best suited for data with a specific label.

The library maps labels to TCP connections using active and passive estimates of network bandwidth and latency. The connection scout provides an initial active measurement of connection quality when a new network option is discovered. As the library sends data over the connection, it measures the response time for individual transmissions to generate passive measurements. The connection scout provides periodic active measurements that are used to assess quality during periods where no data are transmitted and passive measurements are unavailable. Active and passive measurements are combined using a flip-flop filter [19] to derive a running estimate of the current connection quality.

The library uses the following strategy to map labels to TCP connections. Foreground data is given the highest priority. IROBs with the {foreground, small} label are sent over the lowest latency TCP connection. IROBs with the {foreground, large} label are sent over the highest bandwidth connection. These may be the same connection (e.g., if there is only one interface that currently offers connectivity). The actual physical interface used for a specific label may change over time as estimates of link characteristics vary. Background data is given lower priority than foreground data.

Background IROBs are striped over all networks that are not currently sending foreground data. Large, background IROBs are broken into smaller chunks, each of which may be sent over a different network. Our decision to stripe background, but not foreground, IROBs is driven by the different goals of the two labels. A foreground label demands low response time; unfortunately, striping can increase the latency for the last packet to arrive unless the networking layer correctly predicts instantaneous latency for each link. In contrast, the background label specifies data that is not latency-sensitive; thus, a striping strategy that maximizes the utilization of each link is ideal.

The library maintains a collection of IROBs that have been created by the application. Each IROB contains data sent by the application but not yet acknowledged by the peer library on the other side of the multi-socket connection. This means that there is some double-buffering with data contained in the kernel TCP socket buffer; this double-buffering is one performance artifact of a user-level implementation.

Each label has a linked list that indexes all IROBs with that label in FIFO order. Each TCP connection has a list of the labels that it currently is eligible to send; for instance, the lowest latency TCP connection may send either background or foreground data. For each connection, the library sorts the labels in order of preference, i.e., with foreground labels preferred over background ones. When the network is able to send data, the library pulls data from the first IROB on the list associated with the label with highest priority. If no such IROB exists, it moves to the label with next highest priority, and so on. The library encapsulates the IROB data with a 32-byte Intentional-Networking-specific header that includes the IROB identifier and its label, followed by the IROB's ordering constraints. Additional information may be piggybacked in the header, such as current estimates of network bandwidth and latency. The library is not constrained to send all of an IROB's bytes over a connection at once; it may decide to break an IROB into smaller chunks, each of which is sent with an individual header. As an example, this allows the library to start sending IROB data before the application has called `ms_end_irob` to indicate the end of the IROB. IROB chunks sent over multiple TCP connections are reassembled by the receiving library so that the bytes of each IROB are delivered atomically and in order.

The receiving library acknowledges each IROB. The acknowledgment is not constrained to travel over the same network over which the chunk was received. This can be useful if, for example, a TCP connection becomes unavailable after data has been received but before the acknowledgment is sent. For efficiency, acknowledgments are piggybacked on outgoing message headers if a message is queued when the acknowledgments are generated. While Intentional Networking generally relies on the underlying TCP acknowledgments and retransmissions to provide reliability, some additional work is required when a TCP connection breaks. In such instances, the sending library polls the receiving library over a different TCP connection to learn the state of any unacknowledged IROBs that were in flight when the connection was broken.

One challenge is that lower-priority messages (e.g., background requests) may be sent over the same the same network as higher-priority messages (e.g., foreground requests). If the library were to send a large amount of background data, it might unnecessarily delay the foreground data. While a kernel implementation could prioritize one over another at the protocol level, a user-level implementation must use other methods. We have chosen to adapt the *anticipatory scheduling* algorithm [16] to solve this dilemma. Since high-priority traffic is likely to exhibit temporal locality, we bound the amount of data buffered in the kernel by a lower-priority IROB to no more than the amount of data that can be sent within 50 ms if a high-priority IROB has recently been sent by the application. This bound is increased up to a maximum of 1 second, as long as no further high-priority IROBs are observed. Anticipatory scheduling therefore optimizes for low latency for foreground IROBs during periods when many such IROBs are sent, and for high throughput for periods with few foreground IROBs.

The library that receives data guarantees that bytes are delivered to the application in a manner that obeys the mutual exclusion and ordering constraints specified by the sender. Once at least one byte from an IROB has been received by an application, no other bytes from another IROB are delivered until all bytes from the first IROB have been delivered. For this reason, the library does not deliver bytes from a low-priority IROB until it has received all of its bytes. Further, the library buffers an IROB until its ordering constraints are satisfied. For instance, if IROB 2 depends on IROB 1, but is received first (because the two IROBs were transmitted over different networks), the library buffers IROB 2 until after IROB 1 has been received by the application. If two IROBs are eligible to be received, the library delivers the higher-priority one first (e.g., a foreground IROB will be received by the application before a background one). Within a label type, FIFO ordering is used to decide which IROB to deliver.

If a TCP connection fails while IROBs are being transmitted, any remaining data for those IROBs will be sent over the next most appropriate connection. The library masks transient disconnections unless all TCP connections fail simultaneously.

When multiple Intentional Networking applications execute concurrently, the activities of all processes are coordinated through shared memory variables and synchronization. We assume that the links closest to the mobile computer are the bottleneck, and that most of these are shared across all paths of interest. Therefore, each library instance updates a shared variable containing the amount of buffered but unsent data on each network that may send foreground data. The total amount of such data across all processes is not allowed to exceed the limit described above for the anticipatory scheduling algorithm, guaranteeing good foreground performance. If an application not modified to use Intentional Networking executes concurrently with one that does use Intentional Networking, the applications use separate connections and do not coordinate with each other. The Intentional Networking application will adjust its estimates of network quality based on passive observations during its execution, and hence will account for the competing traffic in its decisions.

The Intentional Networking library handles connections between two mobile computers with multiple interfaces by potentially establishing a connection per interface-pair. We do not describe this scenario further as our applications to date have all involved communication between a mobile client and a single-homed server.

## 6. APPLICATIONS

We have modified three applications to use Intentional Networking: BlueFS, a distributed file system for mobile clients; Thunderbird, the Mozilla e-mail and news client; and a vehicular sensing application of our own creation.

## 6.1 BlueFS

BlueFS [32] is an open-source, server-based distributed file system with support for both traditional mobile computers such as laptops and consumer devices such as cell phones [36]. A BlueFS client interacts with a remote server through remote procedure call, augmented with bulk-transfer capabilities. BlueFS inherits parts of its design from previous mobile computing file systems such as Coda [20]. BlueFS clients fetch file and directory information on demand from a remote file server. Files are cached locally on the client. Modifications to file system data are propagated asynchronously to the remote server in the background, in the same manner as Coda's weakly-connected mode [28]. Clients also prefetch data from the server into their caches to improve performance and support disconnected operation.

We adapted BlueFS to use Intentional Networking by modifying its RPC stub generator to take three optional arguments: an Intentional Networking label, ordering constraints, and a thunk. The RPC package uses one socket to connect a client and server; we changed this to be a multi-socket. We also modified the RPC package to create a new IROB for each RPC request and response message with the label, ordering constraints, and thunk specified by the BlueFS client.

We labeled RPCs that are used to prefetch data and asynchronously write modifications back to the server as `{background, large}`. Other RPCs which fetch data on-demand from the server were labeled as `foreground`; the vast majority of these are `small` since BlueFS fetches data on a per-file-block basis. While it is true that some demand fetches may be from applications that are not interactive, the Posix API is insufficient to express this to file systems. Therefore, the conservative approach of treating all such requests as `foreground` seemed best.

Since the file server must see modifications in order, we used Intentional Networking ordering constraints to specify that each file modification IROB depends on the previous one of that type (e.g., all such IROBs are delivered sequentially with respect to one another). However, no constraints are expressed with respect to IROBs of other types, so, for example, the library may reorder an on-demand fetch IROB ahead of a modification IROB.

The server RPC library responds to each RPC with the same label used to send the original request. Since the RPC library already uses a unique identifier for each RPC, matching requests and responses was trivial.

In total, we added or modified 400 lines of code in the RPC library to support Intentional Networking, as well as 134 lines of code in BlueFS. For comparison, the original code base has over 44,000 lines of code.

## 6.2 Mozilla Thunderbird

We also used Intentional Networking to improve the interactive performance of Thunderbird [27], Mozilla's mail and news client. For simplicity, we used an IMAP proxy to intercept traffic between Thunderbird and an IMAP server. The proxy running on the mobile computer prefetches e-mail contents and headers from the IMAP server and stores them on the client's local disk. We replaced the proxy's outgoing connection with a multi-socket and labeled the IMAP messages. Prefetch requests and responses are labeled as `background`, while on-demand fetches triggered by the user via the Thunderbird GUI are labeled as `foreground`. Requests are all labeled as `small`, while responses are labeled as `small` or `large`, depending on their actual size. Each response message from the IMAP server is given the same `background` or `foreground` designation as the request that generated the message. Like the previous application, the IMAP protocol includes a unique identifier for each request/response pair, making it trivial to match requests and responses. Out of 2951 lines of proxy code, we added or changed 124 lines to support Intentional Networking.

## 6.3 Vehicular participatory sensing

Finally, we created a new application targeted at participatory sensing for corporate vehicle fleets. This application is based on specifications for a research/teaching platform developed by Ford Motor Company. The application continuously collects data from a vehicle's internal networks and sensors at a data rate of approximately 25 KB/s. Given ample network bandwidth, the raw data are sent to a cloud server, where they are stored. Raw data can be used for suggesting preventative maintenance, route optimization, improving fuel economy, and other participatory sensing uses.

Since automotive hardware must last a minimum of 10 years and cost reduction is key to profits, the vehicle is expected to have limited storage and computational resources. Therefore, raw data is dropped if sufficient network resources are not available to transmit it immediately. In addition to the raw data, a short 4 KB summary of the data is included. By default, metadata summaries are sent every second, though if bandwidth is insufficient, summaries are generated over longer time periods, e.g., the last 10 seconds. Finally, the vehicle also transmits urgent updates when it encounters anomalous conditions, such as information from the traction control system that indicates slippery road conditions or sudden braking. These updates can be used to warn other vehicles of difficult driving situations such as ice, accidents, or unexpected traffic.

The Intentional Networking version of this application labels metadata summaries as `{background, small}` IROBs and raw data messages as `{background, large}` IROBs. Urgent updates are `{foreground, small}` IROBs. We use ordering constraints to ensure that each raw data IROB is received after the metadata message that summarizes it. The application uses the thunk interface to receive a callback if a background IROB cannot be immediately sent. If the callback is not received before the next raw data message is collected, the previous raw data message is dropped, and the metadata summary is updated to average values over the current time period and all previous ones since the last metadata summary was sent.

We also created an additional version of the application that does not use Intentional Networking. This version uses `select` to determine when the socket buffer is full. Like the Intentional Networking version, this version omits sending raw data and aggregates metadata when it is unable to transmit for more than a second. Our vehicular sensing application has 2080 lines of code. We added or changed 186 lines to support Intentional Networking.

## 7. EVALUATION

We evaluated Intentional Networking by measuring how much it improves network performance for our three applications. Our evaluation uses two different types of network connectivity scenarios: synthetic network conditions that are used as microbenchmarks and traces of actual network connectivity collected from a vehicular testbed. In the latter case, the use of traces provides experimental repeatability and allows a careful comparison among strategies.

## 7.1 Experimental Setup

### *7.1.1 Testbed*

We ran all experiments on a testbed in which the client computer is a Dell Precision 350 desktop with a 3.06 GHz Pentium 4 processor and 1 GB DRAM, running a Linux 2.6.22.14 kernel. All servers run on a Dell Optiplex GX270 desktop with a 2.8 GHz Pentium 4

| Scenario | Network Type | Bandwidth (Mbps) | RTT (ms) | Connectivity |
|---|---|---|---|---|
| Crowded hotspot | Low latency | 0.6 | 20 | Continuous |
| | High bandwidth | 2.0 | 400 | Continuous |
| Intermittent | Wide-area | 0.3 | 400 | Continuous |
| | WiFi hotspots | 3.0 | 60 | Intermittent |

**Table 3: Synthetic network scenarios used in the evaluation**

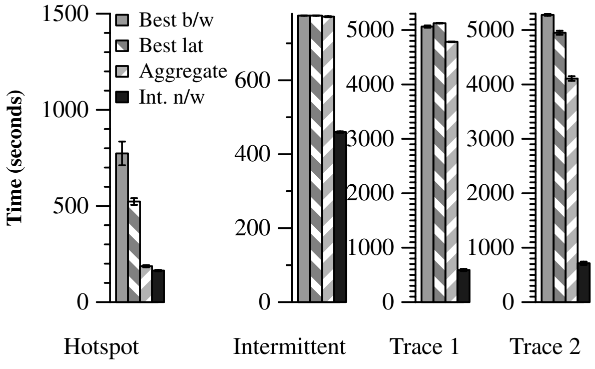

**Figure 1: Interactive latency for BlueFS**

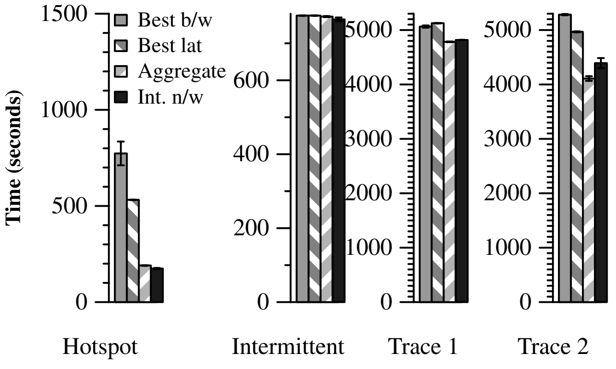

**Figure 2: Background transfer time for BlueFS**

processor and 1 GB DRAM, running a Linux 2.6.18 kernel. These computers are connected via local 100 Mbps Ethernet connections. We emulate wireless network conditions by inserting delays using the netem [22] network emulator and throttling throughput using the Linux Advanced Routing and Traffic Control tools [21].

For Intentional Networking experiments, we modified the client and server component of each application to use our API as described in the previous section and linked each with the Intentional Networking library. We also ran the connection scout on the client computer. For other experiments, the applications are unmodified. All reported values are the mean of 5 trials; graph error bars show 95% confidence intervals.

### *7.1.2 Synthetic Microbenchmarks*

To better understand the behavior of Intentional Networking, we created synthetic network traces that emulate the two network scenarios shown in Table 3. These synthetic traces are intended to help us understand our system's behavior in controlled scenarios rather than precisely emulate actual network behavior. The first scenario replicates the network conditions that would be seen by a user with a high-bandwidth 3G network card sitting at a crowded WiFi hotspot. The 3G network offers higher bandwidth than the crowded AP, but it also inflicts significantly higher latency on network packets. Thus, each network is superior for different types of traffic. Empirically, we observed several instances of such scenarios in the network traces we collected, as described in the next section.

The second scenario emulates a vehicular setting in which a low-bandwidth, high-latency cellular network is continuously available. Opportunistic WiFi connections that offer better bandwidth and latency are intermittently available. We use empirical distributions from the Cabernet project [12] to model the availability of WiFi access points. The distribution of access point encounters has a median of 4 seconds, a mean of 10 seconds, a 99th percentile of 250 seconds, and standard deviation of 0.4 seconds. The distribution of time between APs has a mean of 32 seconds and a median of 126 seconds. Our traces show several instances in which WiFi dominates 3G. However, the traces indicate that this is a simplified view: 3G may also dominate WiFi in many instances; one may offer better uplink bandwidth and worse downlink bandwidth; etc.

### *7.1.3 Trace-driven evaluation*

While the microbenchmarks above help us understand the behavior of our system, we were curious to see how well it would perform in actual vehicular networking conditions. To generate repeatable experiments, we used a two-part process in which we first drove a vehicle with WiFi and Sprint 3G network interfaces. We continuously measured the downlink and uplink bandwidth and latency available through each network interface through active probing to a server at the University of Michigan. We also noted when each type of network was unavailable. The WiFi trace includes only those public APs to which we could associate and establish connections. We collected the traces in Ann Arbor, MI and Ypsilanti, MI at different times of the day. Trace 1 offers better 3G performance overall but encounters fewer public APs. Its median 3G bandwidth is 382 Kbps downlink and 57 Kbps uplink, with maximum bandwidth of 1.3 Mbps downlink and 78 Kbps uplink. Trace 2 has more WiFi access but poorer 3G performance. Its median 3G bandwidth is 368 Kbps downlink and 40Kbps uplink, with maximum bandwidth of 1.2 Mbps downlink and 74 Kbps uplink. Trace 1 has WiFi coverage only 7% of the time, with a median session length of 11 seconds; the longest session was 72 seconds. Trace 2 has WiFi coverage 27% of the time, with a median session length of 7 seconds; the longest session was 131 seconds. In both traces, there are periods where each type of network dominates the other, and where each type of network has better bandwidth but worse latency than the other. Thus, the network conditions are much more variable than in either of our microbenchmarks. These traces will be made available on our project website and on Dartmouth's CRAWDAD archive.

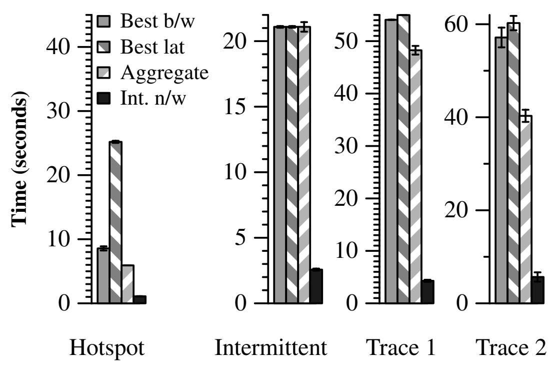

**Figure 3: Average interactive delay for Thunderbird**

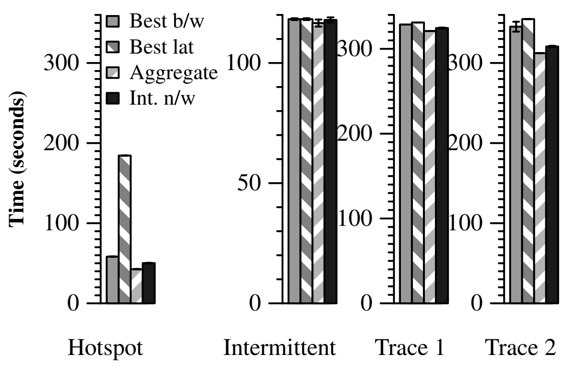

**Figure 4: Background transfer time for Thunderbird**

In the second step, we used the traces to drive the emulator in our testbed. Our traces lasted 138 and 36 minutes, respectively. Because our experiments run for different durations, we use the first portion of each trace for shorter experiments and loop the trace for longer ones.

We chose to use traces rather than measure application performance directly from the vehicle platform to provide repeatable conditions for different network management scenarios. Changing traffic conditions and external load on networks make it very difficult to achieve identical connectivity, even over multiple traversals of the same route. This variability would likely preclude meaningful comparisons across different trials.

### *7.1.4 Comparison strategies*

For each application, we compare Intentional Networking with three strategies. The first two strategies use only a single network at a time but migrate connections to always use the best network according to a specific criteria. The first of these strategies always uses the network with the lowest round-trip time, while the second uses the network with the best bandwidth. We idealize a zero-cost migration by emulating a single virtual network connection that always has the bandwidth and latency of the best current network according to the selection criteria. For example, to create a virtual "best-latency" trace with a single network, we determine whether 3G or WiFi offered the lowest latency for the first second of the original trace, then use the recorded characteristics of that network for the first second of our new trace. We repeat the process for each second. Thus, these strategies show the maximum benefit that could be achieved by a migration strategy if an oracle chooses the best current connection and there is no migration cost.

We also compare Intentional Networking with an idealized version of an aggregation protocol, such as MultiNet or FatVAP, that multiplexes traffic over all available networks. We idealize aggregation by emulating a single virtual network connection that has bandwidth equal to the sum of the bandwidths of all networks and latency equal to the minimum of the latencies of all networks. This virtualized network is ideal in the sense that it offers better connectivity than any protocol that aggregates the individual networks could actually achieve. It therefore offers an upper bound on application-oblivious aggregation performance for each scenario.

## 7.2 Results

### *7.2.1 BlueFS*

To evaluate BlueFS, we run a software development workload that rebuilds the lighttpd (version 1.4.25) Web server source tree. Such "Andrew-style" benchmarks have long been used to test file system performance [14]. Our particular benchmark deletes all object files from the build directory and then runs `configure` and `make` to build lighttpd. The benchmark begins with a cold client file cache, so all files are fetched from the server. We report the total time taken to execute the benchmark (i.e., the interactive performance), as well as the total time to finish propagating updates to the server in the background.

Figure 1 shows the interactive latency for BlueFS (the time to complete the software development benchmark). For the hotspot scenario, the best bandwidth strategy always uses the 3G network. The best latency strategy is an improvement because the workload is dominated by small fetches of 4 KB blocks. The idealized aggregation strategy works very well in this scenario because it is given maximum benefit from the diverse latency and bandwidth of each network. Yet, Intentional Networking still realizes a 14% speedup compared to aggregation by prioritizing foreground over background traffic. Intentional Networking improves interactive latency by 3x compared to the best latency strategy and by 4x compared to the best bandwidth strategy.

To verify that Intentional Networking does not unduly penalize background traffic, we also measured the total time to finish sending all background updates to the server, as shown in Figure 2. Interestingly, Intentional Networking transfers all data 9% faster than the aggregation strategy in the hotspot scenario. At first, this seems anomalous because our idealized aggregation strategy should make maximum possible use of the networks. However, because the benchmark includes computation that depends on foreground transfers, compute episodes start earlier using Intentional Networking. This means that background data is generated sooner in the benchmark. Thus, Intentional Networking is able to use the uplink bandwidth earlier in periods where the aggregation strategy has no data to send. Where data dependencies exist, Intentional Networking can use the network more efficiently than even an idealized aggregation strategy that is unaware of application intent.

In the intermittent scenario, WiFi dominates 3G when it is available. Thus, the best bandwidth and best latency strategies both choose WiFi when available. The aggregation strategy derives a small additional benefit from also using 3G during these periods. Intentional Networking, however, reduces interactive latency by 40%. The benefit compared to aggregation is larger in this scenario because aggregation derives less benefit from its idealized use of two networks to offset Intentional Networking's benefit from understanding application intent. Intentional Networking's total transfer time for all data is 1% better than the other strategies.

The performance of Intentional Networking for latency-sensitive

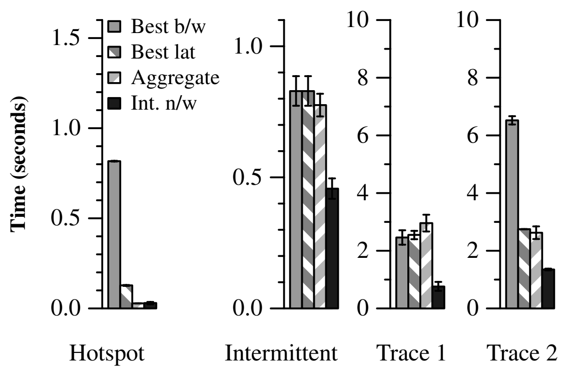

**Figure 5: Urgent update latency for Vehicular Sensing**

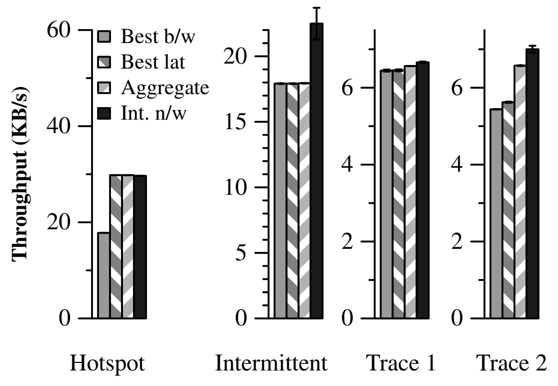

**Figure 6: Background throughput for Vehicular Sensing**

data is even better for the measured vehicular scenarios. Across the two traces, Intentional Networking improves interactive response time by 5-8x compared to aggregation, while increasing total background transfer time by only 1–7%. Compared to the best-bandwidth and best-latency strategies, Intentional Networking improves interactive latency by 7–8x and background transfer time by 5–17%. Despite the increased variability of network quality, Intentional Networking identifies and uses the best network for each type of traffic and thereby maximize benefit to the user.

### 7.2.2 *Thunderbird*

In our Thunderbird benchmark, a user reads e-mail after a period of disconnection. The benchmark first fetches the e-mail headers of 100 messages, then downloads in the background the e-mail messages (with attachments), which range in size from 50 B to 256 KB. While the caching proxy is downloading these messages, the user selects 5 messages to read immediately based on the headers. We report the average interactive delay to fetch the on-demand e-mails, as well as the time to fetch all e-mails in the background.

Results for the Thunderbird e-mail benchmark are shown in Figures 3 and 4. In contrast to the previous benchmark, the migration strategy that maximizes bandwidth is superior to the one that minimizes latency because transfer times are dominated by several large e-mails. Intentional Networking improves interactive latency compared to aggregation by 5x in the hotspot scenario and by 8x in the intermittent scenario. By reordering messages based on application semantics, Intentional Networking is able to deliver superior response time. Total background transfer time is 18% longer in the hotspot scenario, but 1% longer in the intermittent scenario. Results compared to the migration strategies are even better, with Intentional Networking fetching the on-demand e-mails 8–23x faster, while also improving total background transfers by up to 3x.

For the two vehicular measurements, Intentional Networking improves interactive latency by 7–13x compared to the other strategies. The time to transfer all e-mails is within 1–3% of the idealized aggregation strategy and superior to both migration strategies.

### 7.2.3 *Vehicular sensing*

In our vehicular sensing benchmark, the vehicle uploads raw data to a server when network bandwidth is available, as described in Section 6.3. Our benchmark lasts for fifteen minutes. During that time, we have three episodes of urgent data transmissions. Since urgent messages are very often closely correlated in time, we send five messages in a period of seven seconds during each episode. We report the average response time for urgent events and the effective throughput of bulk sensor data, calculated over the entire 15-minute run time of the benchmark.

Figures 5 and 6 show results for the vehicular sensing application. In the hotspot scenario, the aggregate bandwidth is sufficient to prevent background data from interfering with urgent messages. Thus, both Intentional Networking and the aggregation strategy perform very well. The approximately 30 ms average latency for urgent updates is equivalent within experimental error for the two strategies. The aggregation strategy achieves the maximum background data rate of 29 KB/s (a 4 KB summary and 25 KB of raw data per second), and Intentional Networking comes within 0.5% of this rate. Intentional Networking sends foreground data over 4x faster than the best-latency migration strategy.

In the intermittent scenario, Intentional Networking sends urgent events 41% faster than the aggregation strategy and also achieves 25% greater bulk data throughput. The throughput improvement comes from the use of thunks, which allow the Intentional Networking version to avoid polling and better schedule background transmissions.

For the two vehicular traces, Intentional Networking improves urgent event response time by 2–5x compared to the other strategies. At the same time, Intentional Networking improves bulk sensor data throughput by 1–6% compared to the idealized aggregation strategy and by up to 29% compared to the idealized migration strategies.

### 7.2.4 *Concurrent applications*

Finally, we examined the effect of running multiple Intentional Networking applications concurrently by splitting the vehicular sensing application into two separate processes. The first process sends only the urgent messages; the second process sends only the raw sensor data. Figures 7 and 8 show results with two processes, including the two-process version of the application for each of the idealized strategies. The behavior of Intentional Networking with two processes is very similar to that with one process, showing that the cost of using shared memory to coordinate across multiple processes is not significant. The application-oblivious strategies see some benefits from multiple processes in the microbenchmark scenarios because the urgent updates and sensor data transmissions are now concurrent, yet Intentional Networking performance remains comparable to or better than the ideal strategies in all scenarios.

## 8. FUTURE WORK

Applications must currently be modified to use Intentional Networking. As discussed in Section 6, these modifications have not been onerous. Nevertheless, to broaden the applicability of Inten-

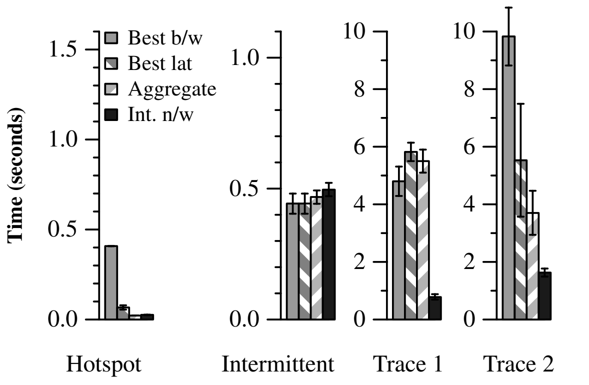

**Figure 7: Urgent update latency, multi-app Vehicular Sensing**

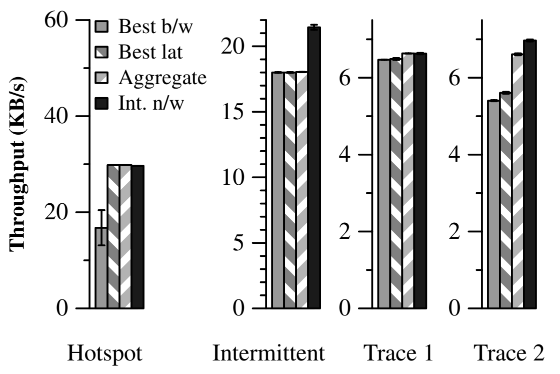

**Figure 8: Throughput, multi-app Vehicular Sensing**

tional Networking, our future plans include providing mechanisms to disclose hints on behalf of unmodified applications.

It may be possible to identify on-demand activity by intercepting user actions and correlating them with network usage. We may even be able to classify opportunistic behavior by observing UI updates that do (or, importantly, do not) happen together with I/O activity. Alternatively, we are planning to combine stack introspection techniques from the security community [47] with causal analysis techniques recently used to create high-performance file systems that provide strong persistence guarantees [33]. This scheme tracks user and UI behavior through the operating system, identifying the set of inputs that can possibly have influenced a set of outputs. Of course, this set is possibly too large because it tracks any relationships that might have been causal. We can prune the set via offline analysis, either by observing many executions of similar code paths and eliminating candidate causal events that only happen some of the time [23] or by using taint checking to profile causality within a process [29].

Our current implementation also requires that both ends of a network connection be modified to use Intentional Networking. When one cannot modify the server, we believe the best solution is to run a proxy in the cloud that converts Intentional Networking traffic from the client to a single TCP connection to the server. The application client can thus use Intentional Networking to manage the wireless connection, which is where the majority of benefit from network diversity is likely to be found.

## 9. CONCLUSION

Mobile nodes face a changing array of diverse networking options, each of which may harbor different strengths and weaknesses. As a result, it is rarely the case that any one networking option is the best choice for all traffic generated by all applications. By using the available options judiciously, an application may see significant improvements in service. Unfortunately, simply exposing the lower-level details of available networks, leaving everything to the application, is unlikely to gain much traction.

Intentional Networking addresses this impasse. It provides a simple, declarative interface for application to express the *intent* behind each network message. The system matches presented network traffic to the best available interface. If no available network is suitable, the traffic is deferred until such a network becomes available. Deferring some types of traffic but not others leads to reordering. Intentional Networking provides mechanisms to express mutual exclusion and ordering constraints over their traffic to match application constraints. Our results using vehicular wireless measurements show that these strategies improve interactive response time from 48% to 13x, while degrading throughput by no more than 7%.

## Acknowledgments

We thank the anonymous reviewers and our shepherd, Victor Bahl, for comments that improved this paper. We used David A. Wheeler's SLOCCount to estimate the lines of code for our implementation. This work is is supported in part by NSF CAREER award CNS-0346686, NSF award CNS-0509089, and the Ford Motor Company. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NSF, the University of Michigan, Ford, or the U.S. government.

## 10. REFERENCES

[1] Anand, M., Nightingale, E. B., and Flinn, J. Self-tuning wireless network power management. In *Proceedings of the 9th Annual Conference on Mobile Computing and Networking* (San Diego, CA, September 2003), pp. 176–189.

[2] Bahl, P., Adya, A., Padhye, J., and Wolman, A. Reconsidering wireless systems with multiple radios. *Computer Communication Review 34*, 5 (2004), 39–46.

[3] Balasubramanian, A., Mahajan, R., and Venkataramani, A. Augmenting mobile 3G using WiFi. In *Proceedings of the 8th International Conference on Mobile Systems, Applications and Services* (June 2010), pp. 123–136.

[4] Bay area wireless users group. `http://www.bawug.org/`.

[5] Brunato, M., and Severina, D. WilmaGate: A new open access gateway for hotspot management. In *Proceedings of the 3rd ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots (WMASH)* (Köln, Germany, September 2005), pp. 56–64.

[6] Carter, C., Kravets, R., and Tourrilhes, J. Contact networking: a localized mobility system. In *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services* (San Francisco, CA, May 2003), pp. 145–158.

[7] Chandra, R., and Bahl, P. MultiNet: Connecting to multiple IEEE 802.11 networks using a single wireless card. In *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies* (Hong Kong, March 2004), pp. 882–893.

[8] Chebrolu, K., Raman, B., and Rao, R. R. A network layer approach to enable TCP over multiple interfaces. *Wireless Networks 11*, 5 (September 2005), 637–650.

[9] Czerwinski, S., Zhao, B., Hodes, T., Joseph, A., and Katz, R. An architecture for a secure service discovery service. In *Proceedings of the 5th International Conference on Mobile Computing and Networking* (Seattle, WA, August 1999), pp. 24–35.

[10] Draves, R., Padhye, J., and Zill, B. Routing in multi-radio, multi-hop wireless mesh networks. In *Proceedings of the 10th International Conference on Mobile Computing and Networking* (Philadelphia, PA, September 2004), pp. 114–128.

[11] Efstathiou, E. C., and Polyzos, G. C. A peer-to-peer approach to wireless LAN roaming. In *Proceedings of the 1st ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots (WMASH)* (San Diego, CA, September 2003), pp. 10–18.

[12] Eriksson, J., Balakrishnan, H., and Madden, S. Cabernet: Vehicular content delivery using WiFi. In *Proceedings of the 14th International Conference on Mobile Computing and Networking* (September 2008), pp. 199–210.

[13] Friday, A., Davies, N., Wallbank, N., Catterall, E., and Pink, S. Supporting service discovery, querying and interaction in ubiquitous computing environments. *Wireless Networks 10*, 6 (November 2004), 631–641.

[14] Howard, J. H., Kazar, M. L., Menees, S. G., Nichols, D. A., Satyanarayanan, M., Sidebotham, R. N., and West, M. J. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems 6*, 1 (February 1988), 51–81.

[15] Hsieh, H. Y., and Sivakumar, R. A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts. In *Proceedings of the 8th International Conference on Mobile Computing and Networking* (Atlanta, GA, September 2002), pp. 83–94.

[16] Iyer, S., and Druschel, P. Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles* (Banff, Canada, October 2001), pp. 117–130.

[17] Johansson, P., Kapoor, R., Kazantzidis, M., and Gerla, M. Personal area networks: Bluetooth or IEEE 802.11? *International Journal of Wireless Information Networks 9*, 2 (April 2002), 89–103.

[18] Kandula, S., Lin, K. C.-J., Badirkhanli, T., and Katabi, D. FatVAP: Aggregating AP backhaul capacity to maximize throughput. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation* (San Francisco, CA, April 2008), pp. 89–103.

[19] Kim, M., and Noble, B. D. Mobile network estimation. In *Proceedings of the 7th International Conference on Mobile Computing and Networking* (July 2001), pp. 298–309.

[20] Kistler, J. J., and Satyanarayanan, M. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems 10*, 1 (February 1992).

[21] Linux Advanced Routing and Traffic Control. *http://lartc.org/*.

[22] The Linux Foundation. *netem*. http://www.linuxfoundation.org/collaborate/workgroups/networking/netem.

[23] Lu, S., Park, S., Hu, C., Ma, X., Jiang, W., Li, Z., Popa, R. A., and Zhou, Y. MUVI: Automatically inferring multi-variable access correlations and detecting related semantic and concurrency bugs. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles* (Stevenson, WA, October 2007).

[24] Magalhaes, L., and Kravets, R. Transport level mechanisms for bandwidth aggregation on mobile hosts. *IEEE International Conference on Network Protocols* (2001).

[25] Matsunaga, Y., Merino, A. S., Suzuki, T., and Katz, R. Secure authentication system for public WLAN roaming. In *Proceedings of the 1st ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots (WMASH)* (San Diego, CA, 2003), pp. 113–121.

[26] Miu, A. K., and Bahl, P. Dynamic host configuration for managing mobility between public and private networks. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS)* (San Francisco, CA, March 2001), pp. 147–158.

[27] Mozilla Thunderbird. *http://www.mozillamessaging.com/en-US/thunderbird/*.

[28] Mummert, L., Ebling, M., and Satyanarayanan, M. Exploiting weak connectivity in mobile file access. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles* (Copper Mountain, CO, Dec. 1995).

[29] Newsome, J., and Song, D. Dynamic taint analysis: Automatic detection, analysis, and signature generation of exploit attacks on commodity software. In *In Proceedings of the 12th Network and Distributed Systems Security Symposium* (February 2005).

[30] Nicholson, A. J., Chawathe, Y., Chen, M. Y., Noble, B. D., and Wetherall, D. Improved access point selection. In *Proceedings of the 4th International Conference on Mobile Systems, Applications and Services* (Uppsala, Sweden, 2006), pp. 233–245.

[31] Nicholson, A. J., Wolchok, S., and Noble, B. D. Juggler: Virtual networks for fun and profit. *IEEE Transactions on Mobile Computing 9*, 1 (January 2010), 31–43.

[32] Nightingale, E. B., and Flinn, J. Energy-efficiency and storage flexibility in the Blue File System. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation* (San Francisco, CA, December 2004), pp. 363–378.

[33] Nightingale, E. B., Veeraraghavan, K., Chen, P. M., and Flinn, J. Rethink the sync. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation* (Seattle, WA, October 2006), pp. 1–14.

[34] Noble, B. D., Satyanarayanan, M., Narayanan, D., Tilton, J. E., Flinn, J., and Walker, K. R. Agile application-aware adaptation for mobility. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles* (Saint-Malo, France, October 1997), pp. 276–287.

[35] NYCWireless. `http://nycwireless.net/`.

[36] Peek, D., and Flinn, J. EnsemBlue: Integrating distributed storage and consumer electronics. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation* (Seattle, WA, November 2006), pp. 219–232.

[37] Perkins, C. IP mobility support for IPv4. RFC 3344, August 2002.

[38] Popa, O. Multipath TCP on mobile phones. Master's thesis, Computer Laboratory, University of Cambridge, 2010.

[39] Sakakibara, H., Saito, M., and Tokuda, H. Design and implementation of a socket-level bandwidth aggregation mechanism for wireless networks. In *WICON '06: Proceedings of the 2nd Annual International Workshop on Wireless Internet* (Boston, MA, 2006).

[40] Salem, N. B., Hubaux, J.-P., and Jakobsson, M. Reputation-Based Wi-Fi Deployment Protocols and Security Analysis. In *Proceedings of the 2nd ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots (WMASH)* (Philadelphia, PA, October 2004), pp. 29–40.

[41] Satyanarayanan, M. Pervasive Computing: Vision and Challenges. *IEEE Personal Communications 8*, 4 (August 2001), 10–17.

[42] SeattleWireless. `http://seattlewireless.net/`.

[43] Snoeren, A., Balakrishnan, H., and Kaashoek, F. Reconsidering Internet mobility. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS)* (Schloss Elmau, Germany, May 2001), pp. 41–46.

[44] Stemm, M., and Katz, R. H. Vertical handoffs in wireless overlay networks. *Mobile Networks and Applications 3*, 4 (December 1998), 335–350.

[45] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., and Paxzon, V. Stream control transmission protocol. Tech. rep., IETF, June 2000.

[46] Terry, D. B. Caching hints in distributed systems. *IEEE Transactions on Software Engineering 13*, 1 (January 1987), 48–54.

[47] Wallach, D. S., Balfanz, D., Dean, D., and Felten, E. W. Extensible security architectures for Java. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles* (Saint-Malo, France, October 1997).

[48] Zaharia, M., and Keshav, S. Fast and optimal scheduling over multiple network interfaces. Tech. Rep. CS-2007-36, University of Waterloo, 2007.

[49] Zhang, M., Lai, J., Krishnamurthy, A., Peterson, L., and Wang, R. A transport layer approach for improving end-to-end performance and robustness using redundant paths. In *Proceedings of the USENIX Annual Technical Conference* (Boston, MA, 2004).

[50] Zhao, X., Castelluccia, C., and Baker, M. Flexible network support for mobility. In *Proceedings of the 4th International Conference on Mobile Computing and Networking* (Dallas, TX, October 1998), pp. 145–156.