Improving Wireless Network Performance Using Sensor Hints

Lenin Ravindranath, Calvin Newport, Hari Balakrishnan and Samuel Madden
MIT Computer Science and Artificial Intelligence Laboratory
{lenin, cnewport, hari, madden}@csail.mit.edu

Abstract

With the proliferation of mobile wireless devices such as smartphones and tablets that are used in a wide range of locations and movement conditions, it has become important for wireless protocols to adapt to different settings over short periods of time. Network protocols that perform well in static settings where channel conditions are relatively stable tend to perform poorly in mobile settings where channel conditions change rapidly, and vice versa. To adapt to the conditions under which communication is occurring, we propose the use of *external sensor hints* to augment network protocols. Commodity smartphones and tablet devices come equipped with a variety of sensors, including GPS, accelerometers, magnetic compasses, and gyroscopes, which can provide hints about the device's mobility state and its operating environment. We present a wireless protocol architecture that integrates sensor hints in adaptation algorithms. We validate the idea and architecture by implementing and evaluating sensor-augmented wireless protocols for bit rate adaptation, access point association, neighbor maintenance in mobile mesh networks, and path selection in vehicular networks.

1 Introduction

With over 172 million devices sold in 2009, smartphones are a rapidly growing market [27]. Some analysts predict that smartphones and pads/tablets will surpass world-wide PC sales by the end of 2011 [20]. These devices may well become the dominant mode of Internet access in the near future [19].

With the proliferation of these "truly mobile" devices, it is increasingly common for wireless network protocols to have to deal with *both* static and mobile usage within a short time period. Consider, for example, a smartphone user at the supermarket who alternates between standing still in front of product displays and moving between aisles, all the while streaming audio through the in-store wireless network. Mobility introduces difficult problems that wireless network protocols must overcome to achieve good performance. During motion, the vagaries of wireless communication become more pronounced: channel quality varies rapidly, losses become more bursty, and assessments of channel behavior are quickly outdated. Because of this, nodes should not maintain long histories, as the rapidly changing channel conditions and network topology would quickly render them invalid. Routing tables may also need to adapt quickly to neighbor changes, and the optimal next-hop may depend on the direction and speed of movement.

However, strategies that compensate for these mobility-related difficulties are unlikely to be optimal in stationary scenarios [4, 25]. When nodes are static, they can average estimates of channel quality, observe their neighbors, and compute routes over long time scales (many seconds), carefully obtaining and updating observations from many packets. In so doing, they can correctly avoid reacting to the inevitable short-term variations that even static wireless networks encounter (e.g., due to short-term fading). Previous work has generally not distinguished between these modes, attempting instead to adapt seamlessly across extremely different network conditions.

The key insight in our work is that nodes can use external (to the network stack) *sensor hints* to improve the performance of wireless network protocols. Our approach is practical and readily implementable because almost every smartphone and tablet today comes equipped with a wide array of sensors like GPS, accelerometers, compasses, and so on. These sensors are used by applications, but are largely ignored by the network stack and protocols. We show how data from these sensors can provide hints to protocols about the *mobility mode* of the device. By "mobility mode," we mean attributes such as whether the device has started moving or is static, its speed of motion, its position, and the heading (direction) of motion—all factors that affect wireless network protocol performance. Protocols can explicitly adapt their behavior and parameters to the current mobility mode.

Sensor hints may be used in different ways in different protocols. When a node generates a hint locally or receives a hint from a neighbor, it may adapt in response to it. The adaptation might be continuous in nature (e.g., updating protocol parameters) or discrete (e.g., switching from a static-optimized to a mobility-optimized protocol). In Section 2, we introduce a novel sensor-augmented wireless architecture that allows devices to extract hints and provide them to protocols. To the best of our knowledge, ours is the first general approach to using sensor hints to augment a variety of network protocols.

In addition to the sensor-augmented network architecture, we make four contributions:

1. Hint-aware bit rate adaptation: In Section 3, we describe and evaluate our implementation of a novel frame-based bit rate adaptation protocol, *RapidSample*, and show through trace-based simulation and testbed experiments that it obtains up to 70% better throughput than existing frame-based and SNR-based rate adaptation protocols, and comparable throughput to SoftRate [25], *when a node is in motion*. We use RapidSample to develop a hint-aware bit rate adaptation protocol that switches strategies based on mobility hints and show through exhaustive trace-based evaluation and testbed experiments that it obtains between 17% and 52% better throughput than SampleRate, 17% and 39% better throughput than RRAA, and 11% and 47% better throughput than SNR-based schemes, in mixed mobility scenarios in various environments.

2. WiFi access point (AP) association: In Section 4, we describe a hint-aware AP association protocol with two modes: maximizing bulk transfer throughput and minimizing handoffs. We show through trace-based evaluation that the hint-aware protocol improves throughput by 30% and reduces the number of handoffs by 40% compared to today's standard scheme.

3. Mobile topology maintenance: In Section 5, we show experimentally that maintaining acceptable error rates for topology maintenance while mobile requires over 20 times more traffic than in the stationary case. We implement a hint-aware protocol that switches to this expensive probing only when in motion.

4. Path selection in vehicular mesh networks: In Section 6, we present a collection of hint-aware path selection metrics for vehicular networks and show, using trace-based simulation, that they increase the stability of short routes by nearly a factor of 5 compared to the hint-free approach.

2 Design

Current wireless protocols adapt their behavior based on *in-network* information such as loss rate, bit errors, or SNR. In contrast, we present a hint-aware protocol architecture that augments this in-network information with hints from external sensors, which can be used at all layers of the network stack to improve performance. In addition to using local sensor hints, a protocol can also adapt based on sensor hints communicated from other nodes.

In this section, we first present a general-purpose hint-aware protocol architecture. We then describe simple and

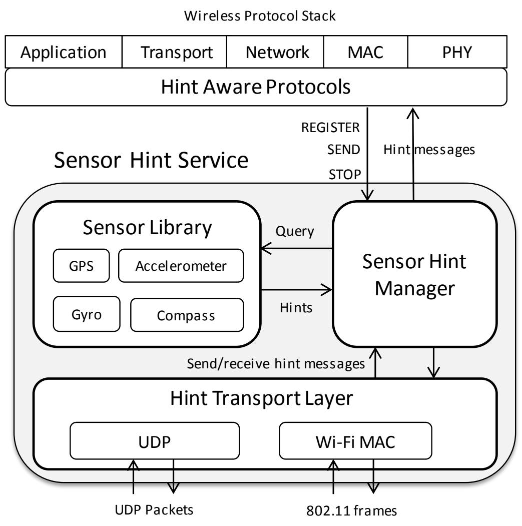


Figure 1: Hint-aware protocol architecture.

Hint Type	Hint Value
Movement	True/False
Walking	True/False
Heading	Degrees Relative to True North
Speed	Miles per Hour
Environment	Indoor/Outdoor

Figure 2: Hint types exposed by the *Sensor Library*.

accurate techniques for extracting mobility hints from sensors such as GPS, accelerometers and compasses.

2.1 Hint-Aware Protocol Architecture

Figure 1 depicts the architecture; the goal is to make it easy to augment wireless network protocols with sensor hints. The architecture provides a *Sensor Hint Service* that abstracts and hides the details of (1) querying various sensors, (2) extracting hints from raw sensor data, and (3) communicating relevant hints over the network. The service exposes well-defined interfaces to achieve these goals. Our current implementation of the Sensor Hint Service runs as a background service on the Android platform and as a Click module for Linux mobile devices. It should be straightforward to incorporate this service into other mobile platforms.

The Sensor Hint Service has three components:

1. Sensor Library. The Sensor Library processes raw sensor data to extract useful hints. We focus on mobility hints and our implementation currently supports the hint types shown in Figure 2. Section 2.2 discusses how these hints are extracted.

2. Hint Transport Layer. Some protocols can benefit from hints from other nodes. For instance, a bit rate adaptation protocol can adapt its bit rate using not only its own movement hints, but also movement hints from nodes the protocol is communicating with. The Hint

Transport Layer provides a protocol-independent way to communicate hints.

When sending a hint to another node, the *Sensor Hint Manager* (described below) constructs a hint message (shown in Figure 3) and delivers it to the Hint Transport Layer, which in turn sends the hint. The hint message consists of the source MAC address and $\langle$hint type, hint value$\rangle$ pairs. When receiving a hint from another node, the Hint Transport Layer delivers the received hint message to the Sensor Hint Manager, which in turn delivers it to the appropriate protocol.

The Hint Transport Layer provides two communication mechanisms to send and receive hints. The first uses UDP. Each node opens a pre-defined UDP port, the HINTS port, to receive hint messages. Hint messages may either be unicast or broadcast to this UDP port.

The UDP scheme works only as long as the nodes are connected through IP. In certain hint-aware wireless protocols (Section 5 and Section 6) nodes do not have IP connectivity, instead communicating via a link-layer protocol such as 802.11's link layer. Thus, for our second scheme, we use a reserved protocol type in the link-layer MAC header to denote a hint message frame (Figure 3). The Hint Transport Layer then listens for unicast or broadcast hints sent in link-layer frames. An alternative scheme might be to overload or piggy-back hints on existing 802.11 frames; we leave the exploration of this possibility to future work.

Because Android phones do not (yet) support sending raw 802.11 frames from user-level, we implemented only the UDP mechanism for phones. For Linux devices, we implemented both schemes. Legacy nodes not running the Sensor Hint Service will simply ignore the hint messages, as long as the HINTS port is not in use by some other application.

3. Sensor Hint Manager. The Sensor Hint Manager arbitrates communication between the protocol, the Sensor Library and the Hint Transport Layer. It exposes a local socket interface (different from the HINTS port) for protocols to interact with the Sensor Hint Service. Protocols register for one or more hints using REGISTER (HintTypes[], ReportRate, CallbackPort, Source). Once registered, the Sensor Hint Manager uses the CallbackPort to stream hints to the protocol. The Source field can be LOCAL, REMOTE, or ALL, corresponding to local hints, remote hints, or both. The protocol can specify a ReportRate, in milliseconds, which indicates how often to report the hint. ReportRate also takes two special values: "0" means "as fast as possible" and -1 means "only when there is a change in the hint state".

Protocols use SEND(HintTypes[], SendRate, ComType, Address) to instruct the service to send hints to other nodes. SendRate takes values similar to ReportRate in the REGISTER command, with the same conventions. ComType specifies the communication types (currently either UDP or MAC frames). Hints may be unicast to a specific node or broadcast in either ComType setting.

REGISTER and SEND both return a unique ID to the protocol. The protocol can use the returned ID to stop sending hints using the STOP (ID) command.

2.2 Extracting Hints

In this section, we describe how to extract the hints shown in Figure 2—*movement*, *walking*, *heading*, *speed*, and *environment*—using standard sensors found on most smartphones and tablets.

Movement hint. Movement is a boolean hint that is true if, and only if, a device is moving, i.e., if either the device's acceleration or its speed is non-zero. We obtain this information from the acceleration sensor indoors, and from the combination of GPS and the acceleration sensors outdoors. Note that it is important to quickly capture the situation when a device has started moving after being at rest, and vice versa, so measuring the acceleration is important.

The accelerometer on most smartphones reports force values for its x, y, and z axes, at a certain sample rate (usually 20–500 Hz). The values are reported either in $\mathrm{m/s^2}$ or in terms of g ($= 9.8\ \mathrm{m/s^2}$). Figure 4 plots a raw accelerometer trace of a smartphone user who walks in the 6–14 second and 22–32 second periods, and is static the rest of the time. The accelerometer shows a significantly higher variance while moving than when stationary. We use this variance to extract a movement hint.

For every new accelerometer sample, we compute the standard deviation of the magnitude of the acceleration over a sliding window (w) of samples. The window slides by one sample for each computation. If the standard deviation in a window exceeds a threshold (a), we detect movement. When the standard deviation is within the threshold for n successive sliding windows, we report that the node is stationary.

We experimented with many values for w, a, and n and determined that $w = 5, a = 0.15\ \mathrm{m/s^2}$, and $n = 10$ gave us few false hints. Figure 5 illustrates our movement hint extraction for the trace in Figure 4. We have implemented the above technique on four different platforms (Android Nexus one, Android Google G1, iPhone 4 and SparkFun accelerometer that connects to a Linux laptop) and found that the parameters offer good performance in all cases.

On the Android platform with a maximum accelerometer sample rate of 50 Hz, we were able to detect movement within 100 ms and detect that the node became stationary within 200 ms. On the Sparkfun platform, with a sample rate of 500 Hz, we were able to detect movement within 10 ms and stationarity within 20 ms.

The movement hint is used by the protocols described

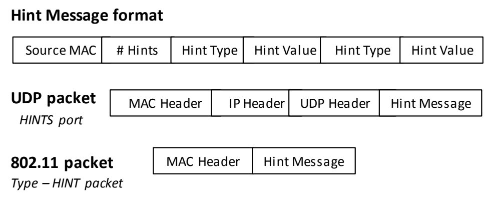


Figure 3: Hint message and packet formats.

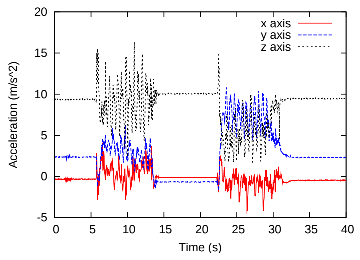


Figure 4: Raw accelerometer trace.

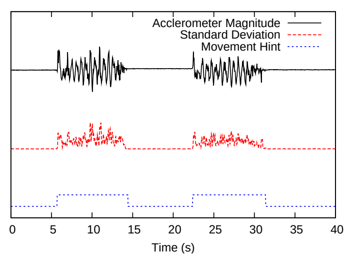


Figure 5: Movement hint extraction from accelerometer data.

in Section 3 and Section 5 to improve bit rate adaptation and topology maintenance, respectively.

Walking hint. Whereas a simple movement hint is useful in some cases, in other situations it is valuable to detect whether a user is walking versus other types of movement, such as when the user is stationary but moving the device. We accomplish this using the walking detector developed in TransitGenie [22] and apply it to AP selection (Section 4).

Heading hint. Heading can be determined from digital compasses (magnetometers) that are available on many devices. GPS also allows us to infer a heading when a device is moving outdoors. These sensors produce a heading in degrees relative to the earth's magnetic north pole. To use a compass to determine the heading of the user holding a device, and not the heading of the device itself, it is necessary to first determine the device's orientation. The standard technique used by inertial navigation systems is to use gyroscope sensors in conjunction with the accelerometer to infer this orientation [21]. In our indoor experiments, we assume we know the orientation of the device, and use only the compass readings. These heading hints are used by the protocols described in Section 4 and Section 6 to improve access point selection and vehicular path selection, respectively.

Speed hint. To determine a speed hint outdoors we can use the speed values reported by GPS. We use this hint in Section 6 for path selection.

Environment (indoor/outdoor) hint. To determine whether a user is indoors or outdoors we use the fact that it is typically impossible to get a GPS fix indoors. In Section 4 we use this hint to improve AP association.

3 Hint-Aware Bit Rate Adaptation

Sensor hints aid in bit rate adaptation because node mobility affects wireless channel conditions, causing large and bursty changes over short intervals of time. When a node moves, bit errors and packet losses exhibit a higher degree of statistical correlation with past behavior as compared to the static case. We demonstrate this effect in Figures 6 (left) and 6 (middle).

Figure 6 (left) plots the *conditional probability* of losing packet number $i+k$ at a given bit rate, *given that* packet number i was lost, for different values of k (the "lag"). In this indoor experiment, we sent back-to-back 1000-byte packets at 54 Mbits/s from a stationary laptop to a stationary smartphone in the static case, and to a smartphone carried by a walking user in the mobile case. A link-layer ACK received from the smartphone indicated a packet success, otherwise the packet was considered lost. The graph shows a significantly higher loss probability for small values of k in the mobile case, demonstrating a larger degree of short-range dependence compared to the static case. In this scenario, for the mobile case, the next packet following a lost packet is significantly more likely to be lost than in the static case, and also compared to larger values of k. For both the static case and the mobile case, the unconditional loss probability was around 23%.

For the same traces, Figure 6 (middle) shows the mutual information between packet success/failure events separated by x ms. Specifically, we compute the mutual information between every pair of two success/failure events separated by a time interval of x ms for a range of different values of x. This measure shows the extent to which the fate of a later packet depends on the earlier one. In the static case, there is no mutual information between packets. But when a node moves, packets exhibit a higher degree of dependence with the past few packets. This dependence drops off at around 10 ms in these experiments. In Figure 6 (right), we plot the mutual information curve for different walking speeds and found the dependence to drop off at around 10–20ms.

These results show that the best strategy for bit rate adaptation is likely to be different when nodes move than when they are static. In more detail, in the static case, where the channel remains relatively stable, it makes sense to maintain a longer history of performance at different bit rates to smooth over periods of short-term fading or contention. Such a long-history approach falters when the device is mobile, because in the mobile case it makes more sense to keep only a short history, react quickly to errors, and perhaps sample other rates aggressively to track the faster changes typical of a mobile channel.

This observation motivates a *hint-aware bit rate adap-*

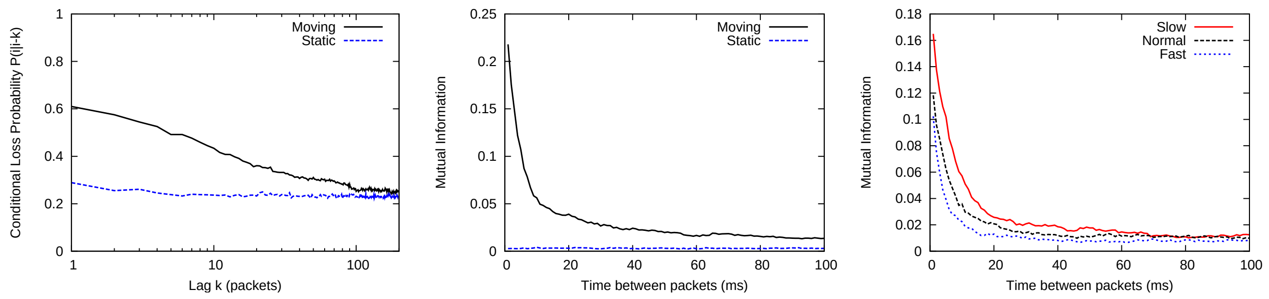


Figure 6: Left: Given a packet loss, the conditional probability of losing the k^{th} packet following the loss, as a function of the lag, k. The unconditional loss probability for both the static and mobile cases was around 23%. Middle: Mutual information between packets separated by x ms specified by the x-axis value. In the static case, there is essentially no mutual information between packets, while in the mobile case, packets separated by less than 10 ms show a high degree of dependence. Right: Mutual information between packets separated by x ms for various walking speeds.

tation scheme, which adapts differently depending on whether or not the nodes are moving. By using external sensor hints rather than making decisions based solely on network information, our goal is to combine schemes tuned separately for the static and mobile cases. The approach requires no training to achieve good performance.

With these remarks in mind, we introduce *RapidSample*, a frame-based rate adaptation protocol designed for a channel undergoing rapid changes due to movement.

```
**RapidSample**$(lastbr, gotack)$ :
  if $(!gotack)$ then
    $failedTime[lastbr] \leftarrow$ **CurrTime**$()$
    if $(sample)$ then
      $br \leftarrow oldbr$
    else
      $br \leftarrow max\{0, lastbr - 1\}$
      $sample \leftarrow 0$
  else
    $sample \leftarrow 0$
    if (**CurrTime**() - $pickedTime[lastbr] > \delta_{success}$) then
      $br \leftarrow max\{i \mid \forall j \leq i :$
                **CurrTime**$() - failedTime[j] > \delta_{fail}\}$
      $sample \leftarrow 1$
      $oldbr \leftarrow br$
    else $br \leftarrow lastbr$
  if $br \neq lastbr$
    $pickedTime[br] \leftarrow$ **CurrTime**$()$
  return $br$
```

Figure 7: The RapidSample bit rate adaptation algorithm. It is called for each packet with *lastbr* describing the bit rate index and *gotack* describing whether an ack was received for the previous packet. Time is reported in elapsed milliseconds.

3.1 The RapidSample Protocol

The RapidSample protocol is shown in Figure 7. It starts with the fastest bit rate. If a packet fails to get a link layer ACK, the protocol switches to the next lowest rate and records the time of the failure. After success at a particular bit rate for more than $\delta_{success}$ milliseconds (5 in our implementation), the sender attempts to sample a higher bit rate. It chooses the fastest bit rate: (a) that has not failed in the last δ_{fail} milliseconds (10 in our implementation), and (b) for which there is no slower bit rate that has failed within this interval. If the faster rate fails, it reverts to the original rate; if it succeeds, it adopts this new faster rate.

There are four ideas motivating RapidSample. First, we observed that when a packet fails while a node is moving, the probability of the next few packets failing at this bit rate is high (Figure 6, left). Therefore, the protocol immediately reduces the bit rate. Second, as we showed in our discussion of Figure 6 (middle), the mutual information between the the fate of packets x milliseconds apart becomes small when x is around 10–15 ms for all the indoor movement speeds we tested. We use a value of 10 ms for δ_{fail} as the minimum time to wait before sampling a previously failed rate, and before sampling any rate higher than the failed rate.

Third, RapidSample attempts higher rates after only a small number of successes at the current rate. We set $\delta_{success}$ to be less than δ_{fail}. In general, it is difficult to tell if the channel conditions are improving or degrading, but under movement, we posit that if conditions are not degrading, they are probably improving because it is unlikely that they are invariant. Thus, even a few successes at one rate provide enough confidence to sample higher rates that have not recently failed. Fourth, if we are wrong about the channel improving, and a higher rate fails, we immediately revert to the original rate.

3.2 Hint-Aware Bit Rate Adaptation Protocol

The Hint-Aware Rate Adaptation Protocol implemented at the sender uses RapidSample when a node is moving and uses SampleRate [3] when a node is static. It relies on movement hints from the receiver to switch between the two. We use SampleRate for the static case as it performed better than other frame-based and SNR-based protocols in various environments (see Section 3.3).

3.3 Evaluation

We use both trace-driven simulation and testbed experiments to evaluate our hint-aware rate adaptation scheme.

3.3.1 Trace-driven Simulation

To replicate the same mobility pattern between different experiments, we used trace-driven simulation—feeding real-world experimental data to a wireless simulator, allowing for both reproducibility and realism. We used the same experimental architecture as [25], which modified the ns-3 network simulator (*v*3.2) to read in experimental traces describing, for each 5 ms time slot, the fate of each packet sent at each bit rate during that time slot. This setup bypasses the physical layer's propagation model, instead referencing the trace file to determine if a packet should be received successfully.

To collect the traces, we configured a Linux laptop as a sender. It ran the Click router using the MadWiFi 802.11 driver, which in turn used an Atheros 802.11 chipset. The laptop sent a constant stream of 1000 byte packets, cycling through the 802.11a OFDM bit rates of 6, 9, 12, 18, 24, 36, 48, and 54, in round-robin order. Each cycle through all 8 bit rates took approximately 5 ms. Indoors, we used 802.11a to minimize interference with local infrastructure networks. We configured a second laptop with the same hardware to act as a receiver, logging every received packet. This laptop was additionally equipped with a SparkFun serial accelerometer for movement hints.

We collected several traces from four different environments for static and mobile scenarios: 1) an *office setting* with no line-of-sight between the sender and receiver, 2) a *long hallway* with line-of-sight between the nodes, 3) an *outdoor setting* with a lightly crowded outdoor pavement area, and 4) a *vehicular setting* where the sender is stationary on the roadside and the receiver is in a moving car near MIT (an urban area).

We evaluated the following frame-based bit rate adaptation protocols: RapidSample, SampleRate [3], RRAA [26], and our *hint-aware* method that switches between RapidSample and SampleRate, depending on the sensor hint. We also evaluated two SNR-based rate adaptation protocols: RBAR [7] and CHARM [8]. For both these schemes, we trained the protocol for the operating environment. We also assumed that the sender has up-to-date knowledge about the receiver SNR. Finally, we compared our protocol to SoftRate [25], a bit rate adaptation scheme that uses SoftPHY hints from a modified physical layer and which can adapt the bit rate on a per-packet basis without requiring training. For this comparison we used the traces from [25].

Figure 8 shows the performance of the hint-aware protocol compared to the other rate adaptation protocols for three of the four environments (we discuss the vehicular setting later in this section). For each environment, we collected 10–20 traces. Each trace is 20 seconds long with 50% static and mobile periods. The receiver was static for 10 seconds and mobile for 10 seconds in each trace. The workload we used was TCP. The graph shows the average TCP throughput of all the schemes as a fraction of the throughput obtained by the hint-aware protocol. The error bars show the 95% confidence interval. In every environment, the hint-aware protocol obtained significant performance gains. It improved over SampleRate by 23% to 52% on average, over RRAA by 17% to 39%, and over RBAR by 11% to 47%. We do not show the numbers for CHARM as the performance of RBAR and CHARM was similar in all cases with RBAR performing slightly better.

We also evaluated the different protocols separately for mobile and static scenarios. For each scenario, we collected ten 20-second traces in each of the test environments. Figure 9 shows the average TCP bulk transfer throughput of all the schemes as a fraction of the throughput obtained by RapidSample, in the mobile case. RapidSample performed significantly better than other schemes in every environment. It obtained up to 75% better throughput on average than SampleRate and up to 25% better than other protocols. These performance gains come from RapidSample's ability to cope up with the rapid fluctuations in the channel conditions when a node is mobile.

On the other hand, RapidSample is the worst-performing protocol in the static case, as shown in Figure 10. It achieved 12% to 28% *lower* average throughput compared to SampleRate and up to 18% *lower* throughput compared to RRAA. The poor performance is because RapidSample aggressively reduces the rate even on a single loss and frequently tries to sample higher rates even when the channel conditions are not changing. Figure 10 also shows that SampleRate usually achieved higher throughput than other protocols when the nodes are static. Hence, we decided to use SampleRate for the static case in our hint-aware rate adaptation protocol.

We also measured the performance of RapidSample in a vehicular setting, where the sender was stationary on the roadside and the receiver was placed in a moving car. We collected 10 traces, each 10 seconds long. Figure 11 shows the results, where the traffic workload is UDP (at a rate of 36 Mbps), as TCP repeatedly times

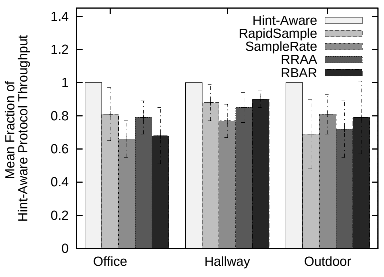


Figure 8: Hint-aware protocol performs better in mixed-mobility setting.

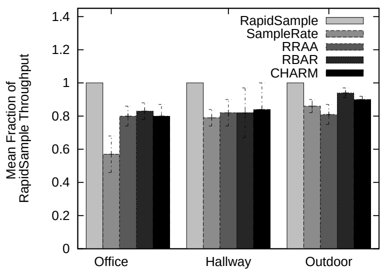


Figure 9: In mobile scenarios, RapidSample performs significantly better than other protocols.

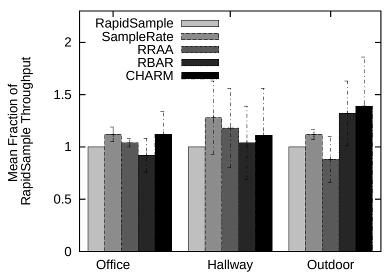


Figure 10: In the static case, RapidSample performs poorly compared to the other schemes.

out when faced with high packet loss rate [6]. Similar to other mobile environments, RapidSample outperformed the other schemes. It achieved about 28% more throughput than SampleRate, 36% more throughput than RRAA and nearly 2× more throughput than the SNR-based protocols.

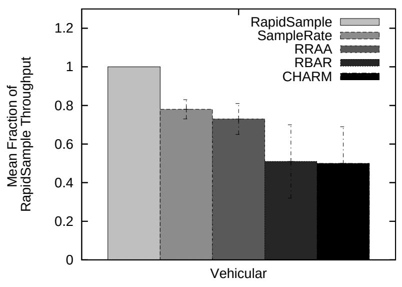


Figure 11: In vehicular environments, RapidSample achieves significantly higher throughput compared to other schemes.

Finally, in Figure 12, we compare RapidSample to SoftRate, SampleRate, and RRAA, using the walking traces and ns-3 protocol implementations from [25]. RapidSample performs nearly as well as SoftRate on these traces, without the aid of SoftPHY hints, further confirming the effectiveness of RapidSample in mobile settings. As a result, our hint-aware protocol performs nearly as well as SoftRate, but is readily deployable on many of today's commodity devices.

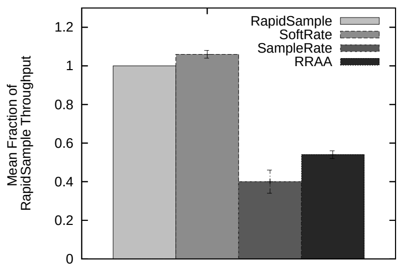


Figure 12: RapidSample performs almost as well as SoftRate on traces collected while walking.

3.3.2 Testbed Experiments

Trace-driven evaluation allows us to compare the performance of various protocols, but there might be a concern that the method used does not correctly account for the channel variations observed in practice. To show that the scheme can work in real-time, we deployed a testbed of Android Nexus One smartphone receivers communicating with a MadWiFi-based Linux laptop sender. We implemented the frame-based rate adaptation protocols (SampleRate, RRAA, RapidSample, and our hint-aware protocol) on the laptop as user-level Click modules; we did not implement the SNR-based protocols as they required SNR feedback from the receiver. The hint-aware protocol used the Sensor Hint Service on the laptop to monitor for movement hints from the smartphone. It switched between SampleRate and RapidSample schemes based on movement hints. The implementation on the smartphone instructed the Sensor Hint Service to send movement hints to the laptop using UDP. The movement hints were sent every second and on hint changes ("static to moving" or "moving to static").

We configured the laptop to send 802.11 data packets to a smartphone's wireless MAC address. Upon receiving the packet, the phone responds with a link-layer ACK. We put the phone in tethering mode, to disable the 802.11 power-saving mode that was on by default. We measure the performance of bit rate adaptation based on the received ACKs.

We evaluated the protocols in two environments: an *office setting* and a *long hallway* setting, the same as in the trace-based evaluation. In each environment, we used 10 distinct mixed-mobility patterns and measured the throughput of each scheme. In each mobility pattern, a user walked in a predefined trajectory at a constant speed and stayed static at predefined locations for predefined amounts of time. Each such *trial* took 45–90 seconds to complete and had an equal amount of static and walking periods. The phone was held by the user in the same way across experiments. Since it was hard to exactly replicate the same mobility pattern, we repeated each trial 3 times and report the average and the standard deviation. A trial consists of running SampleRate, RRAA, RapidSample and the hint-aware protocol back-to-back for the same mobility pattern. Three back-to-back trials correspond to one experimental *run*.

Because the smartphone only had a 802.11b/g card, we did all these experiments in the relatively busy 802.11b/g channels. To minimize interference from the existing access points, we ran the experiments late at night. In every experiment, we sent a stream of 1000-byte packets back-to-back. The bit rate of each outgoing packet was controlled by the rate adaptation scheme. We measured the throughput based on the received link-layer ACKs. The maximum throughput we were able to obtain from the user-level Click implementation was around 27 Mbps.

Figure 13 (left) shows the measured throughput of different protocols in the two environments. For each environment, we plot the average throughput and standard deviation (as error-bars) for 10 different runs. The charts show the results sorted by the throughput of the hint-aware scheme.

In both environments, the hint-aware protocol consistently outperforms the other schemes. In the office setting, it improved over SampleRate by between 10% and 76%, over RRAA by between 8% and 100%, and over RapidSample by between 11% to 41%. On average, it obtained 20% more throughput than SampleRate, 22% more throughput than RRAA, and 19% more throughput than RapidSample. In the hallway setting, the hint-aware protocol obtained 9%–49% more throughput than SampleRate, 8%–80% more throughput than RRAA, and 5%–85% more throughput than RapidSample. On average, it improved over SampleRate, RRAA, and RapidSample by 17%, 37%, and 22% respectively.

Compared to trace-driven results, SampleRate performed better than RRAA in these testbed experiments, especially in situations where the throughput of all the schemes was low. RRAA performed better when the throughput was higher. Otherwise, the testbed results were fairly consistent with the trace-driven simulations.

During each trial, for every packet sent, in addition to logging if an ACK was successfully received, we logged the movement hint as well. We processed these traces and used the movement hint to split them into static and mobile phases and measured the throughput separately for each scenario. Figure 13 (middle) shows the average throughput obtained during the mobile phases of the corresponding experimental runs shown in Figure 13 (left). In mobile scenarios, RapidSample performs significantly better than SampleRate and RRAA in both the environments. On average, it improved over SampleRate by 61% and 40% in the two environments and over RRAA by 16% and 39%. The relative performance of SampleRate was worse in the office setting compared to the hallway setting. This result is consistent with what we found in the trace-driven evaluation. Similarly, Figure 13 (right) plots the mean throughput for the static phases. As found in the trace-based simulation, SampleRate is the best protocol in the static case and RapidSample performed much worse than SampleRate and RRAA.

3.4 Discussion

In our scheme we use only a binary movement hint that indicates whether the device is stationary or mobile. An important conclusion from our results is that even such a simple hint can produce significant performance improvements. An obvious future direction is to generalize our scheme to use additional hints such as speed and heading. Using Figure 6 (right), it is possible to fine-tune parameters in RapidSample for different speeds. While it is easy to get a movement hint, measuring speed accurately indoors using the sensors available on a smartphone is a challenging unresolved problem.

The use of hints for bit rate adaptation may improve PHY-assisted techniques such as SoftRate [25], which perform significantly better than existing protocols in the mobile case using an instantaneous estimate of the bit error rate. Augmented with a movement hint, however, they may be able to adapt their behavior in the static case to average these estimates and avoid reacting to short-term fading.

One benefit of using the accelerometer for a movement hint is that it detects even small movements of the device—e.g., a user moving a smartphone from his head to pocket—which can change the channel conditions. Of course, it is also possible that the channel conditions can change due to changes in the surrounding environment, even if the device is stationary. If such changes are short-lived, then SampleRate, the protocol we use during stationary periods, performs well.

Our trace-driven evaluation shows that the hint-aware protocol performs better than trained SNR-based adaptation in all the tested environments. One question that might arise is whether a protocol could simply use information about *changes* in the observed RSSI values to infer movement and use a protocol like RapidSample in that case, without relying on external sensor hints. We explored this approach and found several problems with it. First, RSSI values showed large variations even when a node was static. Second, the magnitude of these variations depended strongly on the environment and the device. It also varied significantly across time and across different RSSI ranges (low RSSI ranges showed more fluctuations than high RSSI ranges). Third, the reported RSSI was extremely sensitive to movement in the environment and triggered many false hints. Hence, using RSSI was more error-prone than using explicit hints. Furthermore explicit hints avoid the need for training. It

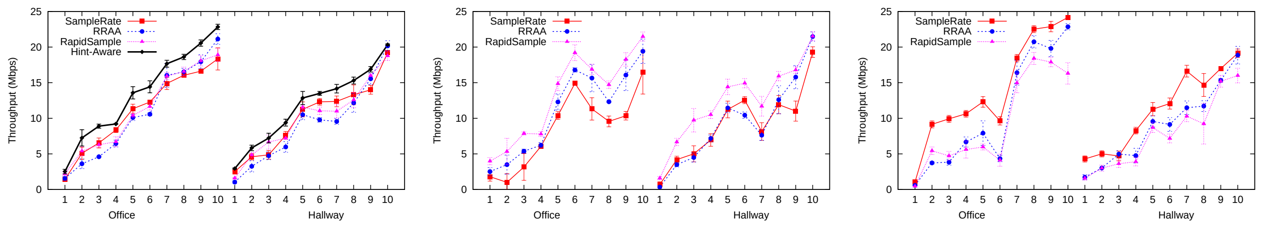


Figure 13: Throughput of the different bit rate adaptation protocols. The left-most chart shows all the protocols with one data point per run; the error-bars are the standard deviations. There are ten runs inside offices and ten in the hallways, with each run lasting 45–90 seconds. The middle chart shows the results considering only times when the device was moving, while the right-most chart shows the results from the same runs considering only times when the device was static. In these experiments, the hint-aware protocol was better than the next-best protocol SampleRate by between 20% (office) and 17% (hallway), with a mean overall improvement of 19%. When mobile, RapidSample outperformed SampleRate by 61% (office) and 40% (hallway), with a mean overall improvement of 50%.

is, of course, conceivable that one could combine RSSI and sensor hints to further improve bit rate adaptation; achieving this goal without environment-specific training remains an open question.

4 Hint Aware AP Association

Most wireless clients associate with the AP with the strongest RSSI (SNR) value. When the RSSI falls below some fixed threshold, the client triggers a handoff, where it scans all the channels and associates with the AP with the strongest RSSI. We refer to this approach as the *standard scheme*.[1] As others have observed [14, 18], the standard scheme is sub-optimal in many settings, particularly when the client is mobile. In this section, we develop a hint-aware association protocol that performs better than the standard scheme.

In our scheme, a node detects whether it is indoors or outdoors using a GPS lock hint. If it is indoors, its association strategy uses the "walking" hint (Section 2.2) to detect motion. The protocol may be configured at run-time to either maximize throughput (Section 4.1), or minimize the number of handoffs (Section 4.2); the former is useful for bulk transfers, while the latter is useful for interactive real-time applications such as telephony for which the hundreds of milliseconds spent during a handoff will disrupt communication, increasing both jitter and packet loss [9] (handoffs took approximately 600 ms on the Android smartphones used in our experiments). When a node is outdoors, it implements a similar strategy, using the position and speed as hints. We do not evaluate the outdoor case in this paper.

We implemented our association protocol as an easily deployable background Android application. Below, we describe the two modes of the protocol and evaluate their performance. Our experimental results with indoor mobility show a median throughput increase of 30% and a median reduction of 40% in the number of handoffs compared to the standard scheme.

[1]Some association schemes do include periodic scans, but they are done only every few minutes, and never while transferring data.

4.1 Using Hints to Maximize Throughput

We present a hint-aware AP association strategy for maximizing throughput. The strategy builds on three observations. First, when a client is moving, the probability that a new AP with a stronger signal enters its range is higher than when the client is static. Hence, when mobile, a client should scan periodically to discover better APs: the throughput gain of associating with a better AP is likely to be higher than the throughput lost to the scan. The periodicity depends on the speed of the client and the expected range of the typical AP in the deployment.

Second, when a client is stationary, it is less likely that new, better, APs will be discovered. In this case, the penalty of a scan is not worth incurring.

Third, when a client stops moving, it may remain static for some period of time. If so, it is worth performing a scan on this transition because the AP with the strongest RSSI is likely (though not guaranteed) to remain optimal until the client moves again. When static, a client should re-scan and re-associate only when it starts moving again, or when the current AP's RSSI becomes weaker than some threshold. In our experiments with the standard scheme, when a client moves from one location to another nearby location, in many cases it remains associated with the original AP (because the signal strength remains above the handoff threshold), reducing throughput. By rescanning immediately following a transition from mobile to static, we avoid this problem.

Our protocol is simple. When the association daemon running on the client detects that the client is walking, it scans periodically, every T_{sc} seconds, for the AP with the highest RSSI. If the client goes from the moving to static state, it performs a scan immediately and associates with the strongest AP. When it is static, it performs no scans, unless the RSSI drops below a threshold or if the client starts moving.

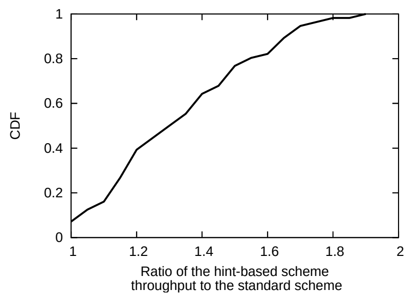


Figure 14: CDF of the ratio of throughput obtained by the hint-aware scheme to the throughput obtained by the standard scheme, calculated for 30 traces.

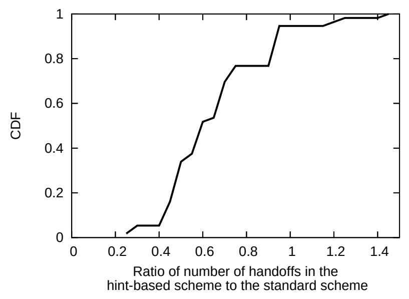


Figure 15: CDF of the ratio of number of handoffs using the heading-aware handoff scheme to the standard scheme calculated over the 30 testing tracks.

The ideal value of T_{sc} is the time it is likely to take for the current AP to no longer be the best one while the user is moving—a factor which depends on how APs are deployed and how fast the user is moving. To get a sense for what it might be in practice, we wrote a data collection application on the Android platform that scans every second, recording the signal strength of every AP it hears. It also records the walking and heading hints with each scan. We convert each RSSI value to a throughput value using a rate map as in [11].

We collected several such traces with different movement patterns and pedestrian speeds indoor in two different buildings on the MIT campus. We found that at pedestrian speeds, a value of $T_{sc} = 8$ seconds maximized throughput. In other words, 8 seconds is approximately the time required to walk between two adjacent APs.

Performance evaluation. To quantify the throughput gains of our hint-aware protocol, we collected 30 walking traces in MIT hallways. These traces are different from the ones we analyzed to determine the value of T_{sc}, but the setting was the same. We had the client transition from moving to stationary states randomly, with roughly equal time spent in each state.

We performed a trace-based evaluation of our hint-aware association protocol compared to the standard scheme, on these traces. Figure 14 shows the CDF of the ratio of throughput obtained by our scheme to the throughput obtained by the standard scheme. The median throughput improvement is about 30%.

4.2 Using Hints to Minimize Handoffs

We now present a hint-aware AP association strategy for minimizing the number of handoffs, which is useful for applications such as VoIP. Our protocol requires lightweight training that can be deployed as a background application on standard phones. The protocol uses the observation that to minimize handoffs, the AP with the strongest RSSI is not necessarily the AP that will yield the longest-lasting connection. If a client is moving towards an AP, for example, it is likely to stay connected longer than if it is moving away, even if the RSSI at the time of the scan is not the highest among the set of observed APs. Our protocol uses heading hints to aid such decisions.

To train our protocol for a specific environment, we use the Android data collection application described earlier. Every second, this application logs the device's heading along with a list of APs and their signal strengths. We use this data to compute a model that maps from a <heading, current AP> pair to a preferred AP, where the preferred AP is the AP to associate with when handing off from the current AP at the given heading to maximize connection time.

Once trained, the protocol works as follows. If it detects the client is stationary it uses the standard scheme. If the protocol detects the client is walking, it extracts a heading hint. It then queries the model using this heading hint and its currently associated AP. The model looks up similar <heading, AP> pairs, and returns the AP that the client should associate with once the current AP's signal strength drops below the handoff threshold. The model attempts to select the AP that will maximize connection time. If the AP returned by the model is not seen during the scan for handoff, the protocol defaults to the standard method of choosing the AP with the highest RSSI.

To evaluate our protocol, we collected 60 tracks using several Android phones, walking through various MIT hallways. For each track, the user chose an arbitrary path in the building complex, and walked between 3–5 minutes. We split the data into training and testing sets—training using the former and testing using the latter. Figure 15 shows that the number of handoffs in our scheme is 40% lower than in the standard scheme. It also shows that for over 90% of the traces, our protocol yielded an improvement of at least 10%.

5 Topology Maintenance

In this section, we study the use of hints to improve the accuracy and efficiency of topology maintenance in wireless mesh (and sensor) networks. Here, each node often maintains a list of neighbor nodes along with the quality of connectivity to each neighbor. The standard method for maintaining neighbors and link quality information in this setting is for a node to send periodic probe packets. Each recipient maintains the packet loss rate of packets from its neighbor, and may exchange this information in the routing protocol's messages. A key parameter is the *probing rate*: how often should these periodic messages be sent? In practice, a node may send these messages at more than one bit rate to produce link quality information at different bit rates.

In determining the frequency of these probes, two opposite considerations must be reconciled. On the one hand, sending frequent probes allows the nodes to maintain an accurate estimate of link qualities and identify changing topologies. Maintaining accurate values for this metric is important to avoid packet losses, which can increase the number of retries and also incorrectly slow down the bit rate. On the other hand, frequent probe packets themselves use large chunks of the bandwidth and increase network contention. This tradeoff becomes even more acute in mobile settings, where link quality changes rapidly. For instance, Figure 16 (left) captures the channel behavior that we observed in a mixed stationary/mobile setting. This plot shows the packet delivery ratio when the user is moving (derived from our movement hint) over time for one specific trace. To calculate the delivery ratio, we bucketed time into intervals of 1 second, during which time the sender transmits approximately 200 packets at each bit rate. The key observation is that motion causes the packet delivery ratio to fluctuate, with many of the jumps in the delivery ratio exceeding 20%.

Our idea is simple: because channel conditions vary much more in the presence of movement, probe frequently when a node receives movement hints from its neighbor or itself moves, and probe less often when the nodes are static.

5.1 Measurement

To evaluate the potential gains, we gathered data in an indoor environment where the sender was static and the receiver was either at a fixed location (*stationary*) or was moved at walking speeds (*mobile*). The sender sends probes at a rate of 200 probes per second. We calculate the actual delivery probability over a sliding window of 10 packets from these rapidly sent probes, subsampling the outcome of these probes to determine the delivery probability at various lower probing rates. We collected 20 stationary and 20 mobile traces, each 180 seconds long. We aggregate the results of the static cases into one set, and the mobile cases into another set. For each set, we calculate the *error in the delivery probability estimate*, which depends on the probing rate, as $|\text{Observed probability} - \text{Actual probability}|$.

Figure 16 (middle) shows the *average* error in delivery probability calculated from all the error samples for the static case as a function of the probing rate; the error bars show the standard deviations. Even a low probing rate of 1 packet every 10 seconds has an error of only 11%, suggesting that the default probing rate of many wireless networks of 1 probe/s may be too high. In contrast, Figure 16 (right) shows that the error in delivery probability is much higher in the mobile case, exceeding 35% even at a probing rate of 1 packet every 2 seconds. To achieve an error of about 10%, the mobile case requires a probing rate of 5 probes per second, which is more than 25× higher than for the static case at the same error rate. For a desired error of 5%, the mobile case needs 10 probes/s, while the corresponding rate for the static case is 0.5 probes per second, a 20× difference.

To understand the reason for this difference, consider a representative 25-second mobile trace in Figure 17 (left). The estimated probability does not track the actual probability except at a high probe rate. This differs from what is observed in the static case.

5.2 Hint-Aware Topology Maintenance Protocol

We implemented a hint-driven topology maintenance protocol using rates of 1 and 10 probes per second for the stationary and mobile cases, respectively. The protocol continues to send at the fast probe rate for one second after the node stops moving in order to estimate the correct metric, before slowing the probe rate down.

Figure 17 (right) compares the performance of our protocol to the standard 1 probe/s protocol. We also plot the movement hint, with a raised value indicating movement. Notice that our adaptive protocol maintains an accurate assessment of the actual delivery probability throughout the experiment, while the non-adaptive 1 probe/s strategy lags by multiple seconds. Note that in some cases, the 1 probe/s approach mis-estimates the delivery probability by more than 30%, whereas, the adaptive estimator is always within 5%.

A simple analysis shows how link mis-estimation degrades throughput. Suppose a node uses ETX [5] to pick the next-hop. Suppose further that there are two choices, one with link delivery probability p_1 and the other with probability p_2; without loss of generality, let $p_1 > p_2$. ETX would choose link 1, with metric $1/p_1$.

Let the error in the average link delivery probability estimate be δ (Figure 16 (right) shows that $\delta > 0.25$ in some cases). The node would pick the wrong link if $p_2 + \delta > p_1 - \delta$. In this case, the extra number of transmissions relative to the optimal value is $\frac{1}{p_2} - \frac{1}{p_1}$. The overhead relative to the optimal is $\frac{p_1}{p_2} - 1$, which can be large for practical parameters; e.g., if $p_1 = 0.8$ and $p_2 = 0.6$, the throughput reduction is 33%.

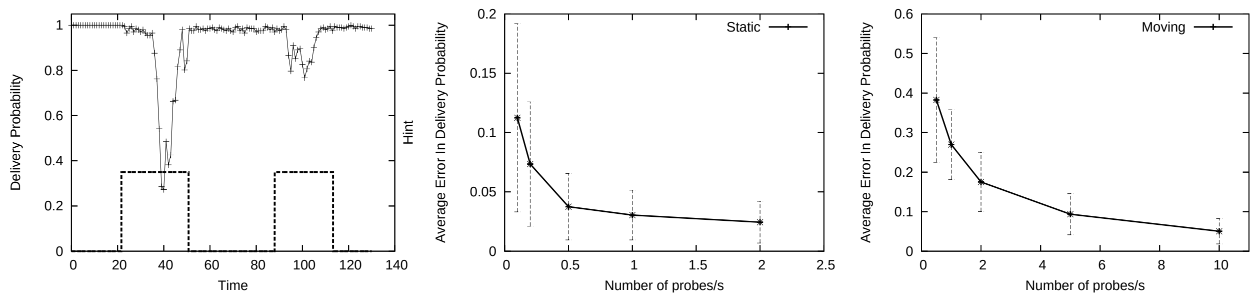


Figure 16: Left: Packet delivery rate for 6 Mbps packets over time; the raised dashed hint line indicates the device is moving. Middle & Right: Average error in delivery probability versus probing rate for static and mobile cases.

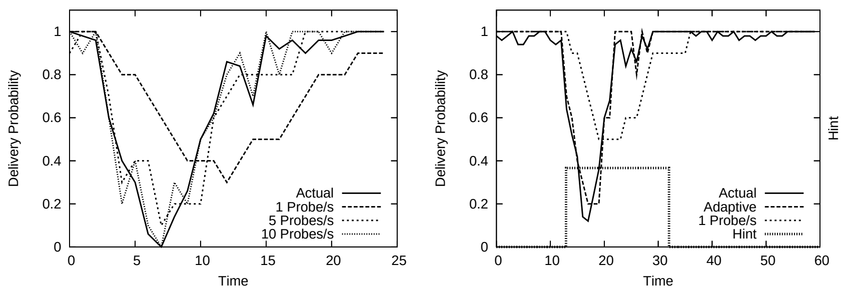


Figure 17: Delivery probability over time for the mobile trace (left) and for a combined static and mobile trace with the dots showing the movement hint (right).

6 Vehicular Network Path Selection

We now investigate whether hints can improve path selection in vehicular mesh networks. Networking strategies in this setting are complicated by dynamic neighbor tables, which can generate a high rate of broken paths. In general, broken paths increase overhead and latency. For this reason, selecting paths with the longest expected connection time may be a good idea.

6.1 Connection Time Estimate Metrics

We present three connection time estimate (CTE) metrics that use *heading* and *speed* hints to estimate whether a path in a vehicular network is likely to be long-lived. Let the ordered sequence $(u_1,...,u_j)$ denote a $j-1$ hop path, $dh(u_i,u_{i+1})$ denote the difference in heading of nodes u_i and u_{i+1} (from 0 to 180 degrees), and $s(u_i)$ denote the speed of u_i (in m/s). Our three CTE metrics, called $cte1$, $cte2$, and $cte3$, are defined for a path $R=(u_1,...,u_j)$:

$$\begin{aligned} cte1(R) &= \prod_{u_i,u_{i+1}\in R} \frac{1}{dh(u_i,u_{i+1})} \\ cte2(R) &= \min_{u_i,u_{i+1}\in R} \left(\frac{1}{dh(u_i,u_{i+1})}\right) \\ cte3(R) &= cte1(R)\cdot\frac{1}{1+\sum_{u_i\in R} s(u_i)} \end{aligned}$$

The metrics use the assumption that a small difference in heading indicates nodes are moving in the same direction on the same road, and are therefore likely to stay connected longer. The $cte3$ metric includes the additional assumption that speed is inversely correlated with connection duration. Because $cte1$ multiplies the inverse of heading differences at each hop, it is biased toward single-hop paths. The $cte2$ metric, by contrast, evaluates a path only by its worst hop, scoring multi-hop paths higher than $cte1$. The $cte3$ metric multiples the inverse of the sum of node speeds with the $cte1$ value. It follows, for example, that doubling the speed of each node on a path approximately halves its $cte3$ score.

To calculate these metrics, each node appends its heading and speed to its mesh neighbor probes. For all three, larger values predict longer-lived paths. These metrics are simple, and require no knowledge of road geometry.

6.2 Evaluation

We evaluated these metrics over a collection of vehicular mobility traces generated from raw position samples gathered from vehicles in the CarTel project over the duration of a year in the Boston metro area, map-matched to an underlying road network [23]. We combine a collection of traces into a *network*, and then simulate, for each second, the position of every vehicle in the network, adjusting the traces so they all appear to begin at the same time. We consider two vehicles to have a *link* at a given time if and only if they are within 100 meters at that time in their traces (we use geographic proximity as a crude surrogate for connectivity).

We measured the relationship between CTE values and path duration over both one and two hop paths.

Specifically, we studied 15 networks consisting of 100 vehicles each. Each simulation lasted for 120 seconds. For each of the over 190,000 routes observed in these simulations, we calculated all three CTE metrics when the path is first formed, and the total duration of the path (in seconds) before it breaks. For each metric, we bucket the CTE scores (into buckets of size $1/20$ for $cte1$, $1/10$ for $cte2$, and $1/200$ for $cte3$), and calculate the median link duration of the paths in each bucket. In Figure 18 we plot these durations for the first three buckets (in descending order of associated CTE score) for each CTE metric. The dashed line indicates the median duration over all paths.

The figure shows that all three CTE metrics provide an effective *filter* for long-lived paths. If a path's $cte1$ value falls into the first bucket, or if its $cte2$ or $cte3$ values fall into the first two buckets, then the path is likely to be long-lived. The median duration of paths in these buckets is 2–5$\times$ longer than the median over all paths.

Identifying long-lived paths might not be a good strategy if the selection mechanism is somehow biased toward routes with low throughput. To evaluate this possibility, we use distance as a rough approximation of achievable throughput (we only have position data from the networks used in this evaluation). We plot in Figure 19 the CDF of time versus distance for the single-hop paths in the first bucket of $cte1$, and the first two buckets of $cte2$, and $cte3$. For comparison we also plot the function for all single-hop paths. This figure confirms that our CTE metrics show no bias favoring links of larger distances (lower throughput).

7 LIMITATIONS

Energy. Sampling sensors consumes energy and reduces the battery lifetime of a mobile device. Figure 20 shows the battery lifetime of an Android G1 device when various hints are sampled at the highest supported rates. Notice that the accelerometer and compass consume much less energy than GPS. To alleviate energy concerns, protocols should extract hints only when transferring data. Moreover, sensors like the accelerometer on a mobile device are usually always on by default (for instance, to continuously detect changes in screen orientation), so extracting hints from them should consume no extra energy. *Triggered sensing* [10] can further reduce the energy consumed by some sensors. Here, a low-power sensor turns on or off a high-power sensor based on certain events; for example, GPS can be turned on only when the accelerometer detects movement. We can also dynamically reduce the sampling rate of sensors to reduce the energy cost [22, 23, 24], and replace expensive GPS with lower-energy position sensors like GSM radios, as in CTrack [24]. In addition, sensor hints can be turned off when the battery is low and protocols can revert to a hint-unaware scheme.

Calibration across devices. The disparity between sensors across different devices and platforms might pose a challenge for hint-aware protocols to work without sensor calibration and tuning. We have implemented the Sensor Library for Android Nexus One, Android G1, and iPhones. The movement hint worked seamlessly across these platforms, but the walking hint detector [22] required a little tuning for each *type* of device.

Privacy. Sharing mobility hints with other nodes might expose private information. For instance, by continuously monitoring movement and heading hints, it might be possible to track a user's behavior more accurately than by just monitoring wireless packets from a device (e.g., I might be able to determine more reliably that you left your office because of the movement hints broadcast by your device); one might alleviate this problem by having all communication go via a (trusted) AP, and encrypting the hints sent to the AP.

8 RELATED WORK

To the best of our knowledge, ours is the first practical work to explore the benefits of systematically integrating sensor hints into a wireless network architecture. Related work that uses information outside the wireless networking stack has mostly focused on wireless power saving. For instance, Wake on Wireless [17] uses an additional low power radio that can be used for signaling to wake up the wireless radio. Cell2Notify [1] uses the cellular radio on a smartphone to wakeup the WiFi interface for VoIP calls thus reducing the energy consumption of WiFi. BlueFi [2] uses GSM towers and nearby Bluetooth devices to predict if WiFi connectivity is available, hence achieving power savings.

In addition to power savings, hints from external sensors for wireless protocols have been used, usually in vehicular network designs. Mobisteer [12] uses directional antennas in vehicles and location hints from GPS to find the best antenna orientation and the AP to associate with. Breadcrumbs [13] predicts the best AP to associate with using a mobility model built using GPS coordinates. CARS [16] is an inter-vehicle bit rate adaptation protocol that uses knowledge of the speed and distance between communicating cars to pick a bit rate. Their method collects a large amount of training data for an environment to determine the best bit rate to use at different speeds and distances; in contrast our hint-aware bit rate adaptation method does not require any such training and performs well across a variety of conditions.

9 CONCLUSION

This paper introduced a network architecture that uses sensor hints to augment and improve wireless protocols.

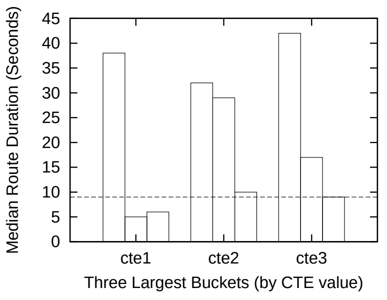


Figure 18: The median route duration for the highest three CTE value buckets. The dashed line is the median over all routes.

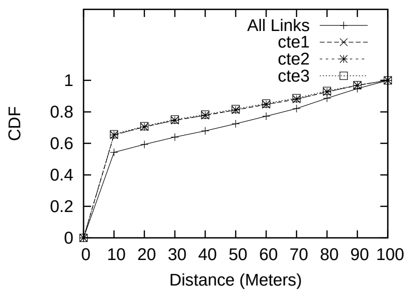
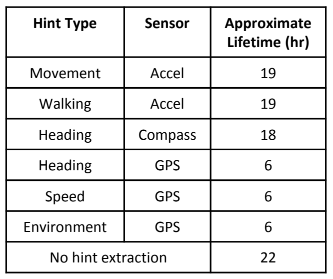


Figure 19: CDF of time spent at each distance for all one-hop links, and the one-hop links from the first $cte1$ bucket, and the first two $cte2$ and $cte3$ buckets.



Hint Type	Sensor	Approximate Lifetime (hr)
Movement	Accel	19
Walking	Accel	19
Heading	Compass	18
Heading	GPS	6
Speed	GPS	6
Environment	GPS	6
No hint extraction		22

Figure 20: Battery lifetime of Android G1 for continuous hint monitoring (with screen at minimum brightness).

The key idea is to use these hints to infer the context in which communication is occurring, and to use that context to adapt the behavior of protocols. We applied this idea to develop hint-aware protocols for bit rate adaptation, access point association, topology maintenance, and path selection in vehicular networks. Sensor hints can also augment other protocols, as described in our earlier position paper [15]. These include: adapting the length of the cyclic prefix to outdoor speeds, scheduling client traffic at an AP taking movement into account, preemptively disassociating clients that have likely moved beyond the range of an AP, and network monitoring.

ACKNOWLEDGMENTS

We thank Aditya Akella for several useful comments. This work was supported by the National Science Foundation under grants CNS-0931550 and CNS-0721702.

REFERENCES

[1] Y. Agarwal, R. Chandra, A. Wolman, P. Bahl, K. Chin, and R. Gupta. Wireless Wakeups Revisited: Energy Management for VoIP Over Wi-Fi Smartphones. In *MobiSys*, 2007.

[2] G. Ananthanarayanan and I. Stoica. Blue-Fi: Enhancing Wi-Fi Performance Using Bluetooth Signals. In *MobiSys*, 2009.

[3] J. Bicket. Bit-rate Selection in Wireless Networks. Master's thesis, Massachusetts Institute of Technology, February 2005.

[4] J. Camp and E. Knightly. Modulation Rate Adaptation in Urban and Vehicular Environments: Cross-layer Implementation and Experimental Evaluation. In *MobiCom*, 2008.

[5] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris. A High-throughput Path Metric for Multi-hop Wireless Routing. In *MobiCom*, 2003.

[6] J. Eriksson, H. Balakrishnan, and S. Madden. Cabernet: Vehicular Content Delivery Using WiFi. In *MobiCom*, 2008.

[7] G. Holland, N. Vaidya, and P. Bahl. A Rate-adaptive MAC Protocol for Multi-Hop Wireless Networks. In *MobiCom*, 2001.

[8] G. Judd, X. Wang, and P. Steenkiste. Efficient Channel-aware Rate Adaptation in Dynamic Environments. In *MobiSys*, 2008.

[9] A. Mishra, M. Shin, and W. Arbaugh. An Empirical Analysis of the IEEE 802.11 MAC Layer Handoff Process. *SIGCOMM CCR*, April 2003.

[10] P. Mohan, V. N. Padmanabhan, and R. Ramjee. Nericell: Rich Monitoring of Road and Traffic Conditions using Mobile Smartphones. In *SenSys*, 2008.

[11] R. Murty, J. Padhye, R. Chandra, A. Wolman, and B. Zill. Designing High Performance Enterprise Wi-Fi Networks. In *NSDI*, 2008.

[12] V. Navda, A. Subramanian, K. Dhanasekaran, A. Timm-Giel, and S. Das. MobiSteer: Using Directional Antenna Beam Steering to Improve Performance of Vehicular Internet Access. In *MobiSys*, 2007.

[13] A. J. Nicholson and B. D. Noble. BreadCrumbs: Forecasting Mobile Connectivity. In *MobiCom*, 2008.

[14] I. Ramani and S. Savage. SyncScan: Practical Fast Handoff for 802.11 Infrastructure Networks. In *Infocom*, 2005.

[15] L. Ravindranath, C. Newport, H. Balakrishnan, and S. Madden. "Extra-Sensory Perception" for Wireless Networks. In *HotNets*, 2010.

[16] P. Shankar, T. Nadeem, J. Rosca, and L. Iftode. CARS: Context Aware Rate Selection for Vehicular Networks. In *ICNP*, 2008.

[17] E. Shih, P. Bahl, and M. Sinclair. Wake on Wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices. In *MobiCom*, 2002.

[18] M. Shin, A. Mishra, and W. Arbaugh. Improving the Latency of 802.11 Hand-offs using Neighbor Graphs. In *MobiSys*, 2004.

[19] Smartphone Owners Lead Rise in Mobile Internet Usage. https://www.strategyanalytics.com/default.aspx?mod=ReportAbstractViewer&a0=5100.

[20] More Smartphones Than Desktop PCs by 2011. http://www.pcworld.com/article/171380/more_smartphones_than_desktop%25%20_pcs_by_2011.html.

[21] S. H. Stovall. *Basic Inertial Navigation*. Naval Air Warfare Center Weapons Division, 1997.

[22] A. Thiagarajan, J. P. Biagioni, T. Gerlich, and J. Eriksson. Cooperative Transit Tracking Using GPS-enabled Smart-phones. In *SenSys*, 2010.

[23] A. Thiagarajan, L. Ravindranath, K. LaCurts, S. Toledo, J. Eriksson, S. Madden, and H. Balakrishnan. VTrack: Accurate, Energy-Aware Traffic Delay Estimation Using Mobile Phones. In *SenSys*, 2009.

[24] A. Thiagaran, L. Ravindranath, S. Madden, H. Balakrishnan, and L. Girod. Accurate Low Energy Map Matching For Mobile Devices. In *NSDI*, 2011.

[25] M. Vutukuru, H. Balakrishnan, and K. Jamieson. Cross-layer Wireless Bit Rate Adaptation. In *Sigcomm*, 2009.

[26] S. Wong, H. Yang, S. Lu, and V. Bharghavan. Robust Rate Adaptation for 802.11 Wireless Networks. In *MobiCom*, 2006.

[27] Smartphone Sales Up 24 Percent. http://techcrunch.com/2010/02/23/smartphone-iphone-sales-2009-gartner/.