Passive Network-Awareness for Dynamic Resource-Constrained Networks

Agoston Petz,[1] Taesoo Jun,[2] Nirmalya Roy,[3]
Chien-Liang Fok,[1] and Christine Julien[1]

[1] The University of Texas at Austin, Austin, Texas, USA
{agoston, liangfok, c.julien}@mail.utexas.edu
[2] Samsung, Korea
hopelist@gmail.com
[3] Institute for Infocomm Research, Singapore
nroy@i2r.a-star.edu.sg

Abstract. As computing becomes increasingly mobile, the demand for information about the environment, or *context*, becomes of significant importance. Applications must adapt to the changing environment, however acquiring the necessary context information can be very expensive because collecting it usually requires communication among devices. We explore collecting reasonably accurate context information *passively* by defining passively sensed context through network overhearing, defining context metrics without added communication cost. We use this framework to build a small suite of commonly used context metrics and evaluate the quality with which they can reflect ground truth using simulation. We also provide an implementation of this passive sensing framework for Linux, which we evaluate on a real mobile ad-hoc network using mobile autonomous robots with commodity 802.11 b/g wireless cards.

1 Introduction

The increasing ubiquity of small, mobile computing devices has introduced applications that find themselves in constantly changing environments, requiring adaptation. Research in context-aware computing has created applications that adapt to location (e.g., in tour guides [1]), time (e.g., in reminder applications [6]) and even weather conditions (e.g., in automated field-note taking [17]). Several toolkits provide abstractions for accessing such context information [7, 11]. While adaptation to physical characteristics is the most obvious use of context-awareness, the ability to respond to the condition of the *network* is just as crucial. Network-awareness is especially important for protocol adaptation as it allows communication protocols to change their behavior in response to the immediate network conditions or the available network resources. Network context can also be used directly by applications, for example to change the fidelity of the data transmitted when the available bandwidth changes.

Traditional means of measuring context are *active* in that they generate extra control messages or require nodes to exchange meta-information. Metrics that report message latency require nodes to exchange ping messages, measuring the amount of latency these messages experience. Traditional measures for determining the degree of mobility in a mobile network require nodes to periodically exchange location and

velocity information. The extra network traffic these mechanisms generate places an increased burden on the already taxed network, making it difficult to justify the use of context-awareness in the common case. If the overhead of sensing context information can be reduced, the benefit of the availability of the information is increased.

We define a framework for defining passively sensed context metrics based on network eavesdropping (Section 3). Our approach focuses on sensing context with zero additional communication overhead. Our context metrics do not provide the exact measure of context that their active counterparts may provide, but we demonstrate the measures' fidelities match traditional measures of context. We use this framework to create instantiations of three common network context measures (Section 4). For each of these metrics, we evaluate the specificity of the passively sensed context metric with respect to the ground truth (Section 5). Our work shows that passive sensing of network context can inexpensively provide information about the state of the world and that, especially when these metrics are correlated with each other, enable adaptive applications in environments where traditional *active* context sensing is cumbersome.

2 Related Work

The demand for adaptive mobile applications indicates the need for efficient context-awareness. Much work has focused on supporting software engineering needs through frameworks and middleware that provide programming abstractions for acquiring and responding to context. For example, Hydrogen [10] defines a completely decentralized architecture for managing and serving context information. Hydrogen's abstractions are unconcerned with *how* context is sensed; clearly, performing context acquisition efficiently is important to the success of such a framework. Many other projects have also looked at reducing the cost of context sensing. Several of these take an application-oriented perspective, identifying what high-level information the application desires and only acquiring information necessary to support an application's desired fidelity [26]. SeeMon [13] reduces the cost of context by only reporting *changes* in context; other time- and event-based approaches also limit overhead this way [8].

Many existing projects provide network context-awareness through dedicated software that sends and receives control messages [4], for example by separating characteristics sensed about the wireless portion of a mobile network from those sensed about the wired portions. The approach does not apply to infrastructureless networks and incurs communication overhead to sense context. There is also a need for network-awareness in mobile agent systems [3]. When supporting mobile agents, however, network-awareness concerns are different due to the fact that an agent's notion of "connectivity" does not necessarily match the network's provision of physical connections. Our work focuses on applications that require awareness of local network conditions.

Active network monitoring has been explicitly separated from passive network monitoring. Komodo [22] defines *passive context sensing* as any mechanism that does not add network overhead. Komodo requires knowledge of the entire network (even, and especially, network links not currently in use), so the project implements an active sensing approach. Given that we focus on mobile networks based on wireless communication, we promote an approach that takes advantage of the inherent broadcast nature of communication, passively gathering information about links that may not be present at the application level. Passive measurement of network properties has been explored in a scheme that uses perceived signal strength to adapt a routing protocol [2]. This approach requires that nodes are able to easily discern the signal strength of incoming

packets and requires nodes to send periodic "hello" messages to monitor their neighbor set, which adds network overhead. A different approach monitors packet traffic to provide routing protocols information about packets dropped at the TCP layer [27]. This information allows protocols to more quickly respond to route failures. We undertake a similar approach in this work but focus on gathering a local measure of network properties instead of boosting performance on a particular end-to-end flow.

These related projects lay the foundation for our work in developing a comprehensive framework for passively sensing network context information. These previous projects have demonstrated 1) a need for context information to enable adaptive communication protocols and applications; 2) a requirement for the acquisition of context to be extensible and easy to incorporate into applications; and 3) a desire to accomplish both of the above with low network communication overhead.

3 Defining Passive Context through Eavesdropping

In this section, we introduce a framework for adding passive context sensing into mobile computing architectures. A schematic of the architecture is shown in Fig. 1.

The physical and MAC implementations handle packet reception and transmission. Our framework inserts itself in two places: first between the MAC layer and the routing layer, and second above the routing layer before the application. The former point serves as an *interceptor* that allows eavesdropping on existing communication. The information overheard through this interceptor will be used to infer various context metrics as described below. The portion of the framework inserted between the routing and application layers exposes the passively-sensed context information to the application, enabling it to adapt to the current context.

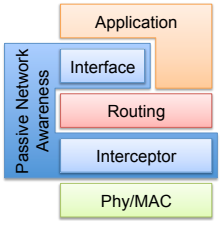


Fig. 1. Architecture for Passive Context Sensing

Existing Network Communication in MANETs. Passive sensing can benefit from information exchanged by these protocols, so it is useful here to provide a brief explanation of their functionality. Unicast routing in MANETs requires every node to serve as a router and can be either table-driven or on-demand. In Destination Sequenced Distance Vector Routing (DSDV) [19], a table-driven protocol, each node maintains a table containing the best known distance to each destination and the next hop to use. These tables are updated by periodically exchanging information among neighbors, generating a fairly constant overhead that is independent of the amount of useful communication. On-demand algorithms like Ad hoc On-demand Distance Vector Routing (AODV) [20] and Dynamic Source Routing (DSR) [12] determine routes only when a packet needs to be sent. These protocols broadcast a route request that propagates to the destination. A reply is returned along the same path. AODV stores routing information in tables on each node, and the tables are updated via periodic exchanges among neighbors. In DSR, the packet carries the routing information, and no beaconing is required. Each approach has advantages and disadvantages [23]; details are omitted as they are not the focus of this paper.

Protocols also exist that add hierarchy [18] or use location to assist routing [14]. In general, the routing protocols share several characteristics: they all generate control messages to discover and maintain routes and they all transport data packets across established routes. A common control message is generated when a node detects that a

path has been broken due to a failed link; the detecting node commonly sends a *route error packet* to its active predecessors for that destination,. In the passive metrics we devise, we will use the discovery, data, and route error messages generated by MANET routing protocols to infer various types of network context information.

Passive Metrics: Some Examples. The following three metrics each measure a dynamic condition of the physical or network environment. In all three cases, the sensed information can be useful to communication protocols that adapt their transmission rates or patterns, and to applications that adapt high-level behaviors.

Network load. The simplest metric in our passive metric suite provides a direct measure of the local traffic on the network. Adapting to this information, applications can prioritize network operations, throttling communication of low importance when the network traffic is high. Communication protocols can also change routing or discovery heuristics in response to changing amounts of network traffic to avoid collisions.

Network density. In dynamic networks, a node's one-hop neighbors can constantly change, and applications can adapt their behavior in response. When the number of neighbors is high, common behaviors can increase collisions and therefore communication delay, while when the number of one-hop neighbors is low, conservative communication can lead to dropped packets and loss of perceived connectivity. To most easily measure the local network density, nodes exchange periodic hello messages with one-hop neighbors. While some protocols already incur this expense, adding proactive behavior to completely reactive protocols can be expensive. We devise a metric for passively sensing network density regardless of the behavior of the underlying protocol(s).

Network Dynamics. Our final example passive metric measures the mobility of a node relative to its neighbors. Traditional measures of relative mobility require nodes to periodically exchange velocity information. We approximate this notion of relative mobility by eavesdropping on communication packets to discern information about links that break. We show how this simple and efficient metric can correlate well with the physical mobility degree in dynamic mobile ad hoc networks.

The Specificity of Passive Metrics. A major hurdle in passively sensing context information is ensuring that the quality of the measurement sensed passively (or the *context specificity*) closely approximates the value that could have been sensed actively for increased cost. This may differ from the actual value for the context metric since even active metrics may not exactly reflect the state of the environment. For each of the passive metrics we define, we generate its context specificity by comparing its performance to a reasonable corresponding active metric (if one exists). This not only allows us to determine whether the particular passive metric is or will be successful, but it also helps us tune our approaches to achieve better specificity.

Adaptation Based on Passive Metrics. One of the most important components of our framework is its ability to make passively sensed context information available to applications and network protocols. As shown in Fig. 1, we provide an interface that delivers passively sensed context directly from the sensing framework. We provide a simple event-driven approach and allow applications to request that a fidelity level be associated with context reports to indicate how confident the sensing framework is in the passively sensed context measure in question.

4 Building Common Context Metrics

To acquire context information at no network cost and little computation and storage cost, we created a passive network suite in C++. Our implementation takes network

packets received at a node, "intercepts" them and examines their details, all without altering the packets or their processing by the nodes. Our implementation also provides an event-based interface through which applications can receive information about passively sensed context. We describe the concrete architecture and implementation of our passive metric suite and look in detail at the specifics of our three sample metrics.

4.1 Implementing Passive Metrics

Fig. 2 depicts our implemented passive context sensing framework. Solid arrows represent the movement of packets. Specifically, packets no longer pass directly from the radio to the MAC layer or from the MAC layer to the network layer; instead they first pass through the passive context sensing framework. Dashed arrows indicate potential uses of the passively sensed context in the form of event registrations and callbacks.

In our passive sensing suite, the interceptor (*passive sensing* in the figure) eavesdrops on every received packet. For each of the passively sensed metrics, the framework generates an estimate of the metric's value based on the information from the data packets at a specified time interval, ν. This time interval can be different for each passively sensed metric depending on its sensitivity in a particular environment. To define a passive metric, a new handler for the metric must be provided that can parse a received packet. The handler defines its own data structures to manage the necessary storage between estimation events. When any packet is intercepted, a copy is passed to the handler for each instantiated metric, and the handler updates its data structures.

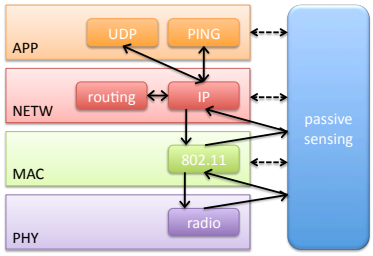


Fig. 2. Concrete Architecture for Passive Context Sensing

Each new metric must also define an estimator that operates on the context information stored in the metric's data structure and generates a new estimate. When the passive framework is instantiated, each metric is provided a time interval for estimation (ν). The framework then calls the metric's estimator every ν time steps to generate a new metric estimate. Larger intervals result in lowered sensing overhead (in terms of computation) but may result in lower quality results (as discussed in Section 5).

4.2 The Passive Metrics

For each metric, our interceptor takes as input the sensed context value at time t and the estimated value at time $t - \nu$ and creates an estimate of the next value of the time series. For each metric, this results in a moving average, in which previous values are discounted based on a weight factor γ provided for each metric. When γ is 0, a new estimate for time t is based solely on information sensed in the interval $[t - \nu, t]$.

Network Load. Network load can be sensed directly by measuring the amount of traffic the node generates and forwards. The network load metric's handler eavesdrops on every received packet, logging the packet's size in a buffer. To generate an estimate, the metric's estimator function simply totals the number of bytes seen in the interval ν and adjusts the moving average accordingly. Specifically, the network load metric nl_i

of a node i is defined as the total of the sizes of the packets that the node has seen within a given time window $[t-\nu, t]$:

$$nl_i(t) = \gamma nl_i(t-\nu) + (1-\gamma) nl_i^m(t-\nu)$$

where $nl_i^m(t-\nu)$ denotes the total size of packets seen by the node in the time interval $[t-\nu, t]$ (i.e., the measured value).

Network Density. Our second metric measures a node's network density, or its number of neighbors. This metric's handler examines each packet and logs the MAC address of the sender. When the estimator is invoked at time t, it tallies the number of unique MAC addresses logged during $[t-\nu, t]$. The network density of a node i is estimated by calculating the number of distinct neighbors of the node:

$$nd_i(t) = \gamma nd_i(t-\nu) + (1-\gamma) nd_i^m(t-\nu)$$

where $nd_i^m(t-\nu)$ calculates the number of distinct neighbors observed in the previous time window. Node i was isolated during $[t-\nu, t]$ when $nd_i^m(t-\nu) = 0$.

Network Dynamics. Our third metric captures the relative dynamics surrounding a particular node. This metric is, to some degree, a measure of how reliable the surrounding network is. In our previous work, we have shown that we can approximate this notion by eavesdropping on communication packets to discern link quality [24]. A node can do this by observing the quality of the received packets directly or by looking at the semantics of packets that indicate link failures.

In the former case, a node observes packets transmitted by neighboring nodes to determine the link quality lq_i^j, which is a normalized representation $\in [0,1]$ of the quality of the link from node j to node i:

$$lq_i^j(t) = \gamma lq_i^j(t-\nu) + (1-\gamma) lq_i^{j,avg}(t-\nu)$$

where $lq_i^{j,avg}(t-\nu)$ calculates the average of the link quality values of the packets received from node j in the current window.

In our implementation in the next section, instead of directly measuring link quality, we rely on the presence of *route error* packets in the communication protocol to indicate faulty links. The metric's handler eavesdrops on every packet, counting those indicating route errors. When the context estimator is invoked, it returns the number of route error packets seen per second in the time interval $[t-\nu, t]$:

$$lq_i^j(t) = \gamma lq_i^j(t-\nu) + (1-\gamma) nre_i^{j,m}(t-\nu)$$

where $nre_i^{j,m}(t-\nu)$ is the number of route error packets from j in $[t-\nu, t]$.

5 Evaluating Passive Context Sensing

We evaluated our passive context sensing by integrating it with the OMNeT++ network simulator [25] with the INET framework. We describe the nature of this evaluation and present some results. In the next section, we translate this prototype in simulation to a concrete implementation on real devices.

Evaluation Settings. We simulated two different environments, both consisting of 50 nodes distributed in 1000m × 1000m. In the first situation, every node attempted to ping a sink (node 0) every 10 seconds (with a ping packet size of 56 bytes). This creates relatively symmetric traffic over the field. In the second situation, one randomly

selected node opened a UDP stream to one other randomly selected node, sending five 512 byte packets every second. In this situation, the traffic is asymmetric; nodes in the path of traffic tend to have more packets to overhear and therefore better information about the passively sensed metrics. We used both AODV and DSR to provide routing support; we did not find statistically significant differences between the two approaches; for consistency's sake, we report results using AODV. We ran five different types of node mobility. In the first case, all nodes were stationary, then they moved at speeds evenly distributed around 2m/s, 4m/s, 8m/s and 16m/s. We used the random waypoint mobility model for node movement with a pause time of 0 seconds.

In these initial evaluations, we aimed to see, in the simplest cases, whether passively sensing context was a viable alternative to active sensing. For this reason, we did not perform any smoothing of the results over time (i.e., all of the weighting factors (γ) were set to 0). As a result, only measurements made in time interval $[t-\nu, t]$ were used to estimate the passive metric at time t. We experimented with five different values of ν: 10, 25, 50, 75, and 100 seconds. For all graphs, we calculated 95% confidence intervals; in most cases, they are too small to see.

Sensing Network Load. In our first metric, we intercepted every received packet and added its size to calculate the average load over the time window $[t - \nu]$. The specificity of this metric is exact; we are *measuring* the metric instead of *estimating* it.

Fig. 3 shows the load measurements our passive framework generated for the PING application for the five different values of ν, plotted as a function of increasing node speed. This figure simply serves to demonstrate the load information we were able to observe. The load increases as speed increases due to the overhead involved in repairing broken routes. It does level out at higher speeds; this reason is discussed below when we examine the network dynamics more carefully.

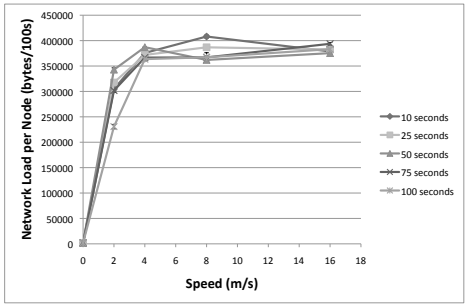


Fig. 3. Network Load Passive Metric

Sensing Network Density. Our node density metric estimates the number of neighbors of a node at time t, based on the observed packet senders over $[t - \nu, t]$. This metric's specificity relates to its ability to correctly identify the number of unique neighbors a node has at time t. Therefore, we calculated the *neighbor error rate* as:

$$ner_i(t) = \frac{|nd_i(t) - nn_i(t)|}{nn_i(t)}$$

where $nn_i(t)$ is the actual number of neighbors i had at time t, retrieved from an oracle.

Fig. 4 plots the neighbor error rate for the PING scenario for the five different values of ν. This metric was the most sensitive to the size of ν. Especially at higher speeds, a wider sensing interval led to very poor estimates of network density. Even with our smallest tested value of ν, the estimation error at node speeds of 16m/s was almost 17%. However, as discussed below in correlating our passive metrics, this was not always the case; correlating this metric with the network load can lead to a better understanding of the reliability of the estimate. The results for estimating the neighbor density in the UDP application scenario were, on average, significantly worse. Many nodes in the network saw very little network traffic, so they had very little information

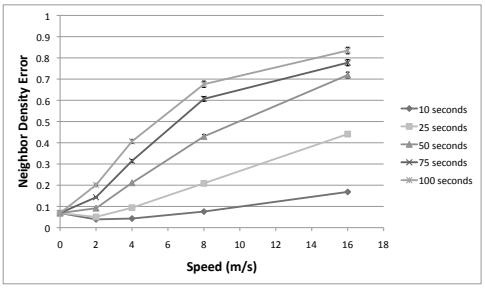


Fig. 4. Network Density Metric

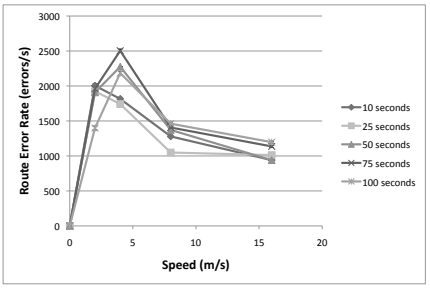


Fig. 5. Network Dynamics Metric

to base their neighbor density estimates on. Again, correlating these estimates with the amount of traffic a node observes can provide better reliability, as discussed below.

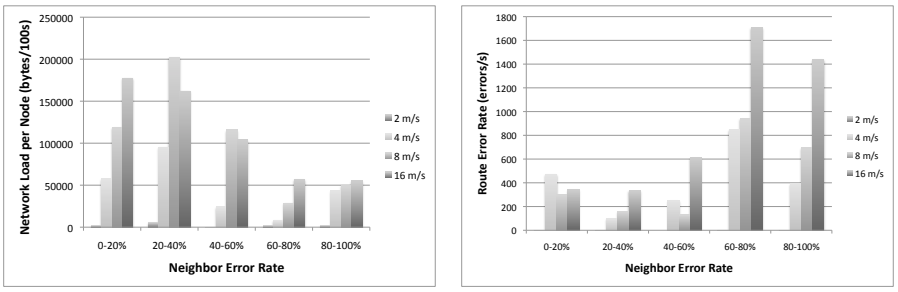
Sensing Network Dynamics. Our final metric relied on overhearing route error packets from either DSR or AODV to estimate the rate of link breaks, which we used to estimate the relative mobility of a node and its neighbors. The ground truth we compare with is the *actual* average speed of the nodes as set in the simulation settings.

Fig. 5 plots the rate of error packets against the average speed of the nodes for the five different values of ν. The expected relationship holds for the lower sets of node speeds (up to 4 m/s). However, at the higher node speeds, the relationship degrades. We conjecture that this may be a result of border effects in our simulation environment in conjunction with the long lag between sending PING packets; these two together may cause the fast moving nodes to disconnect but reconnect without a link break being detected by our passive metric. We look at accounting for these errors in the next section, when we correlate passively sensed metrics with each other.

While this appears to be a disappointing result, Fig. 5 compares this metric with an oracle. A corresponding active metric in which neighboring nodes periodically exchange speed information would add to the network traffic. As shown in Fig. 3, the network load in these highly dynamic situations is already high, and this would lead to a similar degradation in the estimated context value as well. However, from the perspective of applying a passive metric to these dynamic environments for sensing network dynamics, we argue that the measure we provide in our passive metric (i.e., the rate at which nodes *experience* errors in delivering their data packets) is itself a useful measure of the network quality. Therefore, while this passive metric for network dynamics does not show complete specificity with its corresponding oracle, the metric does provide useful information about the quality with which the network can support communication.

Correlating Passive Metrics. Fig. 6(a) shows how one metric can be used to assess the quality of another. This figure shows results for the UDP experiment with $\nu = 50$ for both load and network density. The chart plots the network load observed by nodes based on the node's *neighbor error rate*. The nodes that more correctly estimated their neighbor density (to the left of the figure) were more likely to have seen more network traffic than nodes that were more incorrect in their neighbor estimates.

Similarly, Fig. 6(b) shows the correlation between the neighbor error rate and the route error rate for the same experiment. The results are fairly intuitive; the nodes that experienced fewer route errors were more likely to be correct about their estimate of the number of neighbors. These results motivate applications to use multiple passively sensed metrics in conjunction since information from one metric can provide an indication as to how reliable estimates from another metric are.



(a) (b)

Fig. 6. Correlating Passive Metrics: (a) Neighbor Error Rate and Load; (b) Neighbor Error Rate and Route Error Rate

6 Implementation of Passive Context Sensing

We implemented the passive sensing metrics using the Click Modular Router [15], a flexible, modular, and stable framework for developing routers and network protocols, and we evaluated our implementation on autonomous robots from the Pharos testbed [9]. The following sections describe our implementation, the Pharos testbed, and our experimental setup and results.

Implementation in Click. The Click framework is written in C/C++, runs on Linux and BSD, and includes components to manipulate the network stack from hardware drivers to the transport layer. A Click implementation of a network stack consists of a number of separate *elements*—modular components that each operate on network packets—connected in way that provides the desired functionality. We implemented three such elements, `PCS_Load`, `PCS_Density`, and `PCS_Dynamics`, which implement the three passive sensing metrics described in Section 4.2. Each element also has an external handler to allow other elements or processes to retrieve the computed context value. We have made our implementation available for download[4]. Fig. 7 shows the configuration we used in our experiments. The three passive sensing elements are connected such that all inbound packets are copied and processed by all three elements; the copy of the packets is then discarded. Although it is possible to configure Click to run as a kernel module so it can process the original packets instead of copying them to userspace, this was an unnecessary optimization for our experiments.

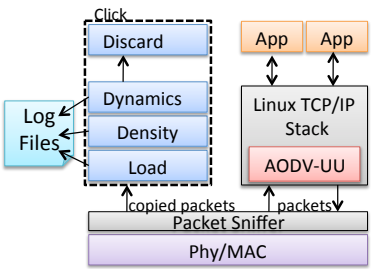


Fig. 7. Click Passive Sensing Implementation

The Pharos Testbed. To fully evaluate our passive sensing implementation, we used the Pharos testbed [9], a highly capable mobile ad hoc network testbed at the University of Texas at Austin that consists of autonomous mobile robots called Proteus nodes [21]. We used eight of the Proteus nodes shown in Fig. 8. Each robot runs Linux

[4] Our implementation is at `http://mpc.ece.utexas.edu/passivesensing`

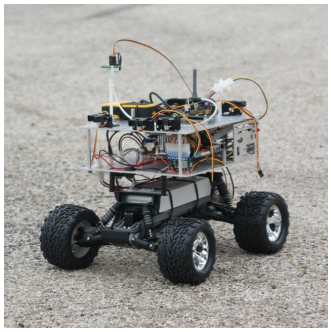


Fig. 8. The Proteus Node

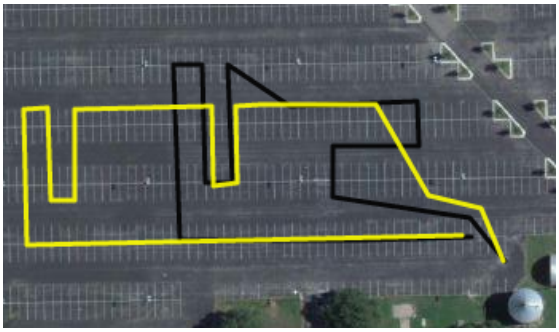


Fig. 9. Waypoints for Experiments

v.2.6 and is equipped with an x86 motherboard, and Atheros 802.11b/g wireless radio. The robots navigate autonomously using their onboard GPS and a digital compass.

Experimental setup. In addition to the passive context sensing suite, each node was running the AODV routing protocol [16] implementation from Uppsala University and sent UDP beacons to every other node at 1s or 10s intervals (depending on the run). This beaconing was independent of the passive sensing suite, and simply provided network load. We used two mobility patterns, a short pattern (shown in black in Fig. 9) which took about 5 minutes to complete, and a long pattern (shown in yellow) which took about 10 minutes [5]. Each pattern had a series of longer jumps punctuated by 2 series of tight winding curves. The robots were started 30 seconds apart and drove at 2m/s (though this varied based on course corrections and imperfect odometer calcuations), and the winding curves were designed to trap several robots in the same area to ensure the formation of a dynamic ad hoc network. To ensure occasional link-layer disconnections in our 150m x 200m space, we turned the transmit power on the radios down to 1dBM (using the MadWiFi stack[6]).

Results. We ran several experiments, and a comprehensive analysis of all of the experiments is not possible in this paper[7]. Fig. 10 shows values of the passively sensed metrics for one robot navigating the longer mobility model with 1s beacon intervals, the weight factor (γ) set to 0, and the time interval ($[t - \nu, t]$) set to 10 seconds to better show the instantaneous context values. Fig. 11 shows a different run with seven robots, the beacon interval set to 10s (instead of 1s), and with each robot instantiating a 1MB file transfer to one randomly chosen destination every 10 seconds. Seventy-eight total file transfers were attempted, of which 43 succeeded and 35 eventually timed out or were interrupted. Although the raw data is not extremely meaningful in isolation, it does show the degree of variation of context observed by a single node even in a small experiment. There are obvious correlations between node density and load and node density and network dynamics that were evident in our real world tests—some of this can be seen in the figures as well.

Comparing real-world results to simulation. To compare the real-world experiments to the simulated results, we took the recorded GPS trace of the Proteus nodes' exact location/time and created trace files that were compatible with OM-

[5] Waypoints generated using `http://www.gpsvisualizer.com`

[6] `http://madwifi.org/`

[7] The raw data as well as videos of the individual experiments can be found at `http://pharos.ece.utexas.edu/wiki/index.php/Missions`.

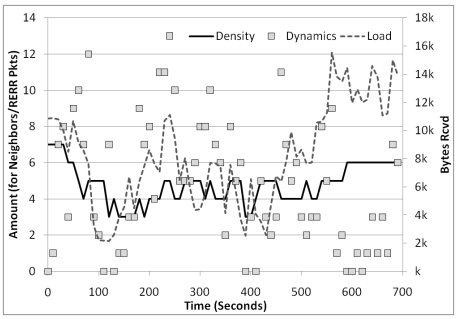


Fig. 10. 8 nodes, 1s beacons, no file-tx

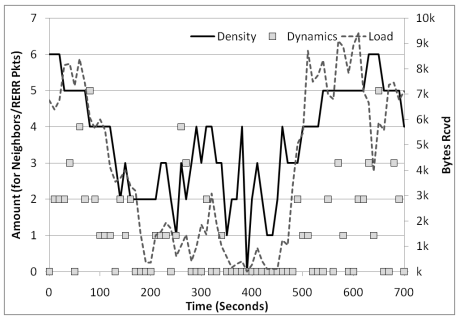


Fig. 11. 7 nodes, 10s beacons, file-tx

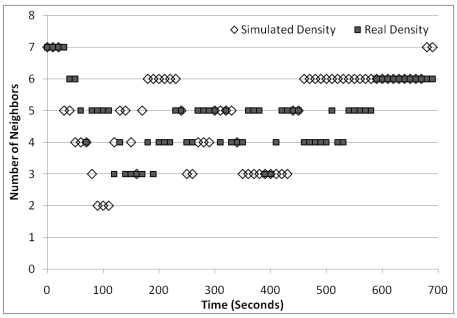


Fig. 12. Sim. vs. real world density

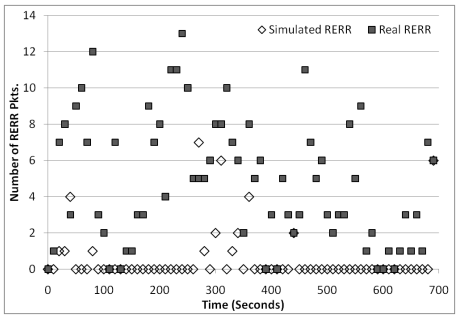


Fig. 13. Sim. vs. real world route errors

NeT++. In this way, we could simulate the exact mobility pattern executed by the robots, including variation from the intended waypoints due to GPS and compass error, steering misalignment, and speed corrections. Figs. 12 and 13 show the simulated results for the same node as Figure 10. We used the same simulation setup as in the previous section, but we set the simulated transmit power to 0.001mW in order to simulate the same number of neighbors on average for each node—this value of 0.001mW was empirically determined by comparing simulations with the observed number of neighbors from the real-world experiments. We were able to correlate the node density between simulation and the real-world well on average, but the number of route error packets seen by the nodes differ significantly. We assume this is due to inaccuracies in the wireless model used in the OMNeT++ simulator.

We have demonstrated that passive context sensing can be implemented in the real world, and we have designed and built a Click-based passive context suite and made our implementation available to the research community. We have also compared the real world tests with simulations using GPS data from the Proteus nodes to recreate the same mobility traces executed during the experiments.

7 Discussion and Future Work

In this section, we briefly discuss some lessons learned and future directions for our passive context sensing suite.

Passive Sensing Sensitivity. We found that the different metrics could be sensitive to different parameters to different degrees. For example, the network density metric was highly sensitive to the value of ν. For our existing metrics and any newly introduced metrics, determining the best values for these parameters will be required and may be the most difficult challenge of passive sensing. However, our work with correlating passively sensed metrics with each other offers promise. Specifically, we showed that correlating the network density with the load can indicate to an application when to "trust" the network density metric; i.e., when the node experiences a higher load, its neighbor density estimate is more likely to be correct. Similarly, we expect that using passively sensed metrics to adjust other metrics' sensitivities may be useful. For example, when a passively sensed network dynamics metric indicates a high degree of dynamics, the node's passively sensed network density metric should likely use a smaller value of ν to achieve better results.

Extending the Passive Context Suite. Our passive context suite is straightforward to extend. Defining a metric requires determining the signature of the packets to be overheard, the information required from those packets, and how the metric should be defined over those overheard values.

As an example, consider adding a link stability metric based on MAC layer information. This new metric utilizes properties of MAC packets to estimate the reliability of a communication link. Upper layer protocols and applications can use this information to adapt to, for example, decrease traffic on low reliability links to reduce the overall probability of collisions. This new passively sensed metric assumes the IEEE 802.11 MAC protocol and its Distributed Coordination Function (DCF), in which, before sending a data packet to another node, the sender performs Request to Send (RTS)/Clear to Send (CTS) handshaking, thereby reserving the shared medium. After receiving a valid CTS packet, the sender and receiver exchange data and acknowledgment packets. In a mobile environment, a node can encounter failures in either of these exchanges for several reasons. To recover from these failures, a node simply retransmits the data following the same procedures until it reaches a retry limit. The retry statistics for a node's outgoing links are reported in the MAC Management Information Base (MIB). A new interceptor in our passive sensing framework can access the MIB at the end of every ν interval to acquire information about the stability of the node's links. The MIB contains two pertinent values in this case: the *dot11FailedCount*, which tallies the number of discarded packets, and *dot11TransmittedFrameCount*, which tallies the number of successfully delivered packets. We can easily define a passive metric of link stability that defines the probability of a successful packet delivery over one hop from node i (ps_i):

$$ps_i(t) = \gamma ps_i(t-\nu) + (1-\gamma)\frac{N_s^{MIB}}{N_s^{MIB} + N_f^{MIB}}$$

where N_f^{MIB} and N_s^{MIB} are *dot11FailedCount* and *dot11TransmittedFrameCount*, respectively. This new passive metric is easily inserted into our existing OMNeT++ framework simply by examining the MIB storage located in the 802.11 MAC implementation. The implementation of the passive context suite in Click is similarly easy to extend.

Adapting to Passively Sensed Context. We have made our passively sensed context metrics available through an event based interface. Upper-layer protocols and applications can register to receive notifications of changes in passively sensed context metrics and adapt in response. We have already begun integrating passively sensed context into a pervasive computing routing protocol, Cross-Layer Discover and Routing

(CDR) [5], in which we use this passively sensed context to adjust the proactiveness of a communication protocol in response to sensed network dynamics. In highly dynamic situations, the protocol avoids proactiveness due to the overhead incurred in communicating information that rapidly becomes outdated. However, in lower dynamic environments, some degree of proactiveness makes sense to bootstrap on-demand communication.

8 Conclusions

Mobile and pervasive computing applications must integrate with and respond to the environment and the network. Previous work has demonstrated 1) a need for context information to enable this expressive adaptation; 2) the ability to acquire context information with little cost; 3) the ability to easily integrate new context metrics as they emerge; and 4) software frameworks that ease the integration of context information into applications and protocols. In this paper, we have described a framework that achieves all of these goals by enabling the *passive* sensing of network context. Our approach allows context metrics to *eavesdrop* on communication in the network to estimate network context with no additional overhead. We have shown that our framework can be easily extended to incorporate new metrics and that the metrics we have already included show good specificity for their target active metrics in both simulation and a real network deployment. We have provided implementation of the passive sensing metrics for both the OMNeT++ simulator, and for Linux nodes using the Click Modular Router. Additionally, we have shown that applications can even adapt the context sensing framework by correlating the results of multiple passively sensed context metrics. This information enables adaptive applications and protocols in environments where active approaches are infeasible or undesirable due to the extra network traffic they generate.

References

1. G. Abowd, C. Atkeson, J. Hong, S. Long, R. Cooper, and M. Pinkerton. Cyberguide: A mobile context-aware tour guide. *ACM Wireless Netw.*, 3:412–433, 1997.
2. P. Basu, N. Khan, and T. Little. A mobility based metric for clustering in mobile ad hoc networks. In *Proc. of ICDCS Workshops*, pages 413–418, 2001.
3. W. Caripe, G. Cybenko, K. Moizumi, and R. Gray. Network awareness and mobile agent systems. *IEEE Comm. Magazine*, 36(7):44–49, July 1998.
4. L. Cheng and I. Marsic. Piecewise network awareness service for wireless/mobile pervasive computing. *Mobile Netw. and App.*, 7(4):269–278, August 2004.
5. A. Dalton and C. Julien. Towards adaptive resource-driven routing. In *Proc. of Percom*, 2009.
6. A. Dey and G. Abowd. Cybreminder: A context-aware systems for supporting reminders. In *Proc. of HUC*, pages 172–186, 2000.
7. A. Dey, D. Salber, and G. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human Computer Interaction*, 16(2–4):97–166, 2001.

8. A. Ferscha, S. Vogl, and W. Beer. Ubiquitous context sensing in wireless environments. In *Distributed and Parallel Systems: Cluster and Grid Computing*, volume 706 of *The Springer Int'l Series in Eng. and Comp. Science*, pages 98–106, 2002.
9. C.-L. Fok, A. Petz, D. Stovall, N. Paine, C. Julien, , and S. Vishwanath. Pharos: A testbed for mobile cyber-physical systems. Technical Report TR-ARiSE-2011-001, AriSE, University of Texas at Austin, 2011.
10. T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann, and W. Regschitzegger. Context-awareness on mobile devices: the hydrogen approach. In *Proc. of HICSS*, 2003.
11. J. Hong and J. Landay. An infrastructure approach to context-aware computing. *Human Computer Interaction*, 16(2–4), 2001.
12. D. Johnson, D. Maltz, and J. Broch. DSR: The dynamic source routing protocol for multihop wireless ad hoc networks. In *Ad Hoc Netw.*, pages 139–172, 2001.
13. S. Kang, J. Lee, H. Jang, H. Lee, Y. Lee, S. Park, T. Park, and J. Song. SeeMon: Scalable and energy-efficient context monitoring framework for sensor-rich mobile environments. In *Proc. of Mobisys*, pages 267–280, 2008.
14. B. Karp and H. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proc. of Mobicom*, pages 243–254, 2000.
15. R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek. The click modular router. *SIGOPS Oper. Syst. Rev.*, 33(5):217–231, 1999.
16. E. Nordstrom and H. Lundgren. Aodv-uu implementation from uppsala university.
17. J. Pascoe. Adding generic contextual capabilities to wearable computers. In *Proc. of ISWC*, pages 92–99, 1998.
18. G. Pei, M. Gerla, X. Hong, and C.-C. Chiang. A wireless hierarchical routing protocol with group mobility. In *Proc. of WCNC*, pages 1538–1542, 1999.
19. C. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computingers. In *Proc. of SIGCOMM*, pages 234–244, 1994.
20. C. Perkins and E. Royer. Ad-hoc on-demand distance vector routing. In *Proc. of WMCSA*, pages 90–110, 1999.
21. http//proteus.ece.utexas.edu.
22. M. Ranganathan, A. Acharya, S. Sharma, and J. Saltz. Network-aware mobile programs. In D. Milojicic, F. Douglis, and R. Wheeler, editors, *Mobility: Processes, Computers, and Agents*, pages 567–581, 1999.
23. E. Royer and C. Toh. A review of current routing protocols for ad hoc mobile wireless networks. *IEEE Personal Comm. Magazine*, 6(2):46–55, April 1999.
24. R. Srinivasan and C. Julien. Passive network awareness for adaptive mobile applications. In *Proc. of MUCS*, pages 22–31, May 2006.
25. A. Vargas. The OMNeT++ discrete event simulation system. In *Proc. of ESM*, pages 319–324, 2001.
26. Y. Wang, J. Lin, M. Annavaram, Q. A. Jacobson, J. Hong, B. Krishnamachari, and N. Sadeh. A framework of energy efficient mobile sensing for automatic user state recognition. In *Proc. of Mobisys*, pages 179–192, 2009.
27. X. Yu. Improving TCP performance over mobile ad hoc networks. In *Proc. of Mobicom*, pages 231–344, 2004.