



A dynamic reconfigurable routing framework for wireless sensor networks [☆]

Mukundan Venkataraman ^{a,*}, Mainak Chatterjee ^a, Kevin Kwiat ^{b,1}

^a Department of Electrical Engineering and Computer Science, University of Central Florida, Orlando, FL 32816, United States

^b Air Force Research Laboratory, Rome, New York, United States

ARTICLE INFO

Article history:

Received 8 March 2010

Received in revised form 13 January 2011

Accepted 17 January 2011

Available online 22 February 2011

Keywords:

Wireless sensor networks

Multi-hop routing

Dynamic routing

ABSTRACT

Sensornet deployments of the future are expected to deliver a multitude of services, ranging from reliable sensing, real time streams, mission critical support, network reprogramming and so on. Naturally, no one routing protocol can sufficiently cater to the network layer functionalities expected. Severe resource constraints further limit the possibility of multiple routing protocols to be implemented. Further, vertically integrated designs of present protocols hinder synergy and code-reuse among implementations. In this paper, we present an architecture that allows applications to send different *types* of flows, often with conflicting communication requirements. A flow's requirements are made visible to our framework by using just 3 bits in the packet header. The core architecture is a collection of highly composable modules that allows rapid protocol development and deployment. We show that our framework can provide: (i) flow based network functionality that ensures each flow gets an application specific network layer which is dynamically knit as per the flow's needs, (ii) modular organization that promotes code-reuse, run time sharing, synergy and rapid protocol development and (iii) pull processing that allows flows to dictate their traffic rate in the network, and implement flexible scheduling policies. This creates a framework for developing, testing, integrating, and validating protocols that are highly portable from one deployment to another. Using our framework, we show that virtually any communication pattern can be described to the framework. We validate this by gathering requirements for one real world application scenario: predictive maintenance (PdM). The requirements are used to generate a fairly complete and realistic traffic workload to drive our evaluation. Using simulations and 40 node MicaZ testbed experiments, we show that our framework can meet the deployments demands at granularities not seen before in sensornets. We measure the costs of using this framework in terms of code size, memory footprints and forwarding costs on MicaZ nodes.

© 2011 Elsevier B.V. All rights reserved.

1. Introduction

Protocol development in sensornets have to meet the challenges of correctness and efficiency, while functioning on a resource constrained hardware platform. Hence, typical sensornet deployments thus far have focused on performance more than functional flexibility by building deployments that perform limited tasks. In fact, most sensornet deployments today have a highly focused and rather singular objective. Technology trends indicate that

[☆] This research was supported, in part, by the Air Force Office of Scientific Research (AFOSR) under the federal Grant No. FA9550-07-1-0023. Approved for Public Release; Distribution Unlimited: 88ABW-2011-0368 dated 28 Jan 2011.

* Corresponding author. Tel.: +1 321 274 7591.

E-mail addresses: mukundan@eecs.ucf.edu (M. Venkataraman), mainak@eecs.ucf.edu (M. Chatterjee), kevin.kwiat@rl.af.mil (K. Kwiat).

¹ An earlier version of this paper appeared in IEEE WCNC 2009 [19].

sensornets of the future will be expected to do multiple, and often conflicting, tasks simultaneously. For example, a comprehensive deployment solution could simultaneously send periodic sense-and-disseminate flows, real time streams, mission critical alerts, network reprogramming data, patched updates, interactive queries, commands and so on. Naturally, no *one* routing optimization can sufficiently cater to such diverse communication requirements. Accommodating arbitrary communication patterns while conserving limited resources calls for dynamism in route selection.

Providing a dynamic network layer functionality is, however, non-trivial. Extreme scarcity of resources makes full implementations of multiple protocols infeasible. Further, existing protocols are highly optimized for a given deployment and its associated traffic type. This optimization has come at a high cost: limited portability, limited code-reuse, and non-extensible designs. To promote synergy and maintain adaptability, there is a need to bring flexibility closer to protocol design.

In this paper, we present an architecture that allows an application to generate multiple *types* of flows, often with conflicting communication requirements. Our architecture provides an *application specific* network layer for each flow based upon its needs. To make this possible, a flow's requirements are made visible to the framework using three descriptive bits in the packet header. Likewise, protocol modules inside the framework carry a similar 3 bits description that advertises their suitability for a particular flow. The core of our framework is a modular software architecture for implementing network layer functionalities composed of “modules”. Modules are highly composable, lightweight components that implement small network tasks. Functionalities in the network layer are built by arranging a sequence modules in the form of a graph. New protocols can be written by extending, adding or re-arranging the modules already present. We regularize the various interface assumptions and data structure requirements by the use of a shared neighbor table, and decouple core protocol features from resource management functionalities. In effect, we integrate the vast body of work in network protocols available in sensornets by distilling core protocol features for various traffic types. This would foster tremendous synergy across various deployment efforts since the core component protocols can be easily migrated from one effort to another. While we present specific implementation choices for both the preamble bits and the component protocols, we note that this is *only one* instance of our proposed framework. We envision various preamble bits and accompanying protocols based on the need at hand. In other words, this paper's contribution is a frameworks that can handle conflicting requirements and not the component protocols themselves.

Specific properties of this framework make it a powerful tool for building, testing and deploying complete sensornet applications. *Pull processing* allows elements closer to the transmitting interface regulate packet flow on a need basis. *Flow based network functionality* allows inspection of a flow's needs, and helps construct an appropriate set of functionalities for that flow by knitting relevant

modules. It provides a deeper insight to providing reliability for certain flows, speedy delivery for certain other, aggregation for delay tolerant flows and so on.

We evaluate the overhead of using this framework in terms of code size, memory footprint and processing overhead by implementing various combinations of modules on MicaZ motes [27]. Our experience shows that we achieve upto 30% reduction in code size and more than 40% reduction in memory footprints compared to full implementations. Our framework can forward upto 4437 bytes/sec before a livelock [13] occurs.

We also bring out the practical benefits of using this framework by conducting a case study of a fairly large scale industrial application of predictive maintenance (PdM). We show how our framework can adapt to the various communication requirements of an operational PdM, and offer custom network layer function for every instance. We gather and meet PdM's operational demands on a 40 node wireless MicaZ testbed for 2 days, and a 9-week worth simulation time on a 1000 node simulation to analyze behavior at scale.

2. Related research

This paper builds upon two previous approaches by us with a dynamic network framework for sensornets [19–21], by providing a comprehensive network layer architecture and framework for flow based network functionality. We also provide implementation insights, performance evaluation and a case study to substantiate previous work.

There has been an excellent series of work that address the problem of promoting synergy and providing sufficient abstraction for rapid protocol development. Work by Culler et al. [3] argued that the abstraction that IP provides in the Internet can be provided at the link layer for sensornets, and this was substantiated by the SP architecture [16]. The SP architecture deals with the link layer, and as such, is complimentary to our work. In fact, our dynamic routing framework could very well be implemented *on top* of the SP architecture. The case for a modular network layer, where core network layer protocols were decomposed into an abstract set of modules, was then proposed by Cheng et al. [1]. They address issues of code-reuse and runtime sharing, and postulate a co-existence of routing protocols. However, they do not address how multiple flows can co-exist, and how a unified framework could provide flow based network functionality or pull processing. The issue of code-reuse and ease of protocol development has been investigated in [7,9]. Though we share the same motive as some of these approaches (promoting a programmable framework for rapid deployment, unifying various research efforts in routing data), our approach and methodology are different. Unlike other approaches, our scheme revolves around a preamble that describes communication intent of a flow, with the intent specifying requirements such as reliability, speed of delivery, and nature of payload.

Modularizing network layer functionality has also been witnessed with Internet architectures such as the x-kernel [10,15] and the Click [14] modular IP router. The x-kernel

is an operating system and a platform for implementing network protocols with a stress on composability and performance. However, it dealt with relatively resource rich Internet terminals, and allows for full implementations of multiple protocols to form dynamic stacks [15]. The Click IP router, on the other hand, organizes router functions as composable “elements” that allows push and pull based functionality, much like our architecture. However, much of the Internet’s proposals are inappropriate for sensor networks: (i) Internet is about client-server architectures and point-to-point communications, while sensornets are wireless (broadcast) with completely different routing requirements, (ii) resources are not in scarcity in the Internet (energy, bandwidth, memory, processing, etc.), while a mote is highly resource impoverished and (iii) full implementations of multiple protocols are not out of reach in the Internet; such a design is nearly impossible on sensor-net hardware because of the very limited nature of resources. A case may also be made about the type-of-service (TOS) field in the IP-header, which is purported to provide service differentiation in the Internet. The TOS field classifies a flow according to its relative priority. Routers interpret a flow accordingly as being lesser or more important based on the TOS field. Again, our bits in the preamble provide *no* notion of any relative priority amongst flows; rather, they convey the communication *intent* of a packet in terms of reliability, timeliness, and nature of payload. The bits are used to match a packet to the best possible set of routing components that can cater to it.

The issue of supporting concurrent applications, with each independent application sharing the resources of a mote, has been addressed in Mate [12] and Melete [25]. Mate is primarily concerned with code dissemination for network reprogramming. Melete significantly extends the Mate architecture, and addresses the issues of reliable storage and runtime sharing of multiple concurrent applications. Though it effectively substantiates the possibility of running concurrent applications on a mote, it does not address the conflicting communication requirements of various applications. Our work complements their by utilizing that fact that multiple concurrent applications can reside on a mote, and we address every application’s communication requirements.

3. Protocol description

3.1. Routing framework overview

Application packets present a 3 bits preamble to the framework that describes its communication intent. The framework dynamically optimizes routing decisions based on these preamble bits. The routing layer is now a composable set of routing components that perform similar functions, but are optimized for different classes of traffic. The selection of a routing component is hence a mapping between what the application demands and what the component has to offer. To unify interface assumptions, the routing components share a universal neighbor lookup table that houses relevant information about various neighbors which saves storage space and makes way for

consistent interface assumptions. Component protocols make ranged queries into the neighbor table to derive the best candidate hop for a given application payload. In designing this framework, we address challenges of composing routing protocols into core components, and regularizing the various interface and data structure requirements. But before that, we begin with the fundamental problem of making visible an applications demands to the stack.

3.2. Specifying communication intent

Applications need a way to express their communication requirements in a format that is both completely expressive and minimal. Various approaches have been taken to increase application visibility to the communication framework, and much of the effort has been to prioritize data. For example, the SP architecture [16] argues for a 1 bit descriptor that describes the urgency of a packet. ISP’s in the present day Internet also use priority tags to differentially route packets from preferred customers and offer them a higher quality of service. However, assigning a priority for any class of traffic is a highly subjective task, and these assignment rules would not be consistent from one deployment to another. We believe that application naming should revolve around fundamental communication requirements rather than blatant priorities. This would allow one to construct meaningful and consistent inferences of a packets requirement across deployments.

In our quest to best characterize an application to the communication framework, we experimented with various possible parameters: number of recipients (anycast, multicast, broadcast), loss tolerance, delay tolerance, priority, sensitivity to congestion or link losses, soliciting retransmissions, aggregation, load balancing, control information v/s data packets and so on. However, we found that communication patterns can be best described using three fundamental axes: nature of payload, reliability, and time criticality.

Nature of payload: Traffic in the network can be broadly classified as data or control traffic. Data traffic is all of the push based traffic generated by a sensor node (mote). Control traffic is traffic used for control plane management (like beacons, ACK’s, etc.), and to a certain extent, user generated traffic.

Reliability: This bit denotes the tolerance to loss: a 0 means that the flow is agnostic to loss, and 1 means that the flow demands reliability.

Time criticality: Some sensed value might be of no use if it does not make it to the destination within certain time bounds. A flow that demands speedy delivery sets the time criticality bit to 1, while a 0 means that the flow is agnostic to delay.

Note that these 3 bits are *only one* instance of creating a preamble. We choose these because they are fundamental to data-exchange. For example, energy is an important criteria that we do not consider. However, the preamble can be easily extended to accommodate various other metrics of interest.

3.3. Preamble bits

Fig. 1 shows the three axes of communications reduced to a simple 3 bits scheme that each packet carries. These bits are set or unset by the application programmer using simple API calls. The 3 bits, taken in combination, provide a characteristic description of an applications requirements. Fig. 2 provides a complete combination of the preamble bits and their inference. For example, a beacon packet would carry a signature of [1,0,0], denoting a control packet that is loss tolerant and insensitive to delay. A data packet demanding reliability over delay would publish [0,0,1]. Similarly, a real time packet which only demands speed of delivery would publish [0,1,0]. An anomalous case is made when a packet demands both reliability and speed of delivery (bits [0,1,1] and [1,1,1]). Ensuring reliability inherently adds delay in transit, and we interpret such packets as mission critical packets, which see the need to both make it to the destination and in as short a time as possible. In general, the bits when combined with other information available in the packet headers make way for a powerful expression of precise communication demands. Note that the bits do not convey any notion of relative priority amongst packets, just a set of actual communication requirements.

We note that the 3 bits shown in the design are by no means binding: protocol developers are free to extend, add or modify the design to better suit their requirements. However, we do believe that the eight flow types that emerge from using 3 bits are mostly sufficient to describe a very accommodating set of flows (a real world case study is presented in Section 5).

3.4. Shared neighbor table: unified view for routing protocols

Since the routing layer is now a collection of independent routing components, each component assumes the presence of various state information to be available to perform routing decisions. The neighbor table houses values such as the node-ID, energy available, congestion level, depth, link quality estimate and a 'last heard' bit. The columns can easily be extended by future protocols. Since routing components share this table, it decouples core protocol features from interface assumptions and regularizes data structure requirements. This leaves the routing layer with a composable set of routing components that can be seamlessly ported across various research efforts.

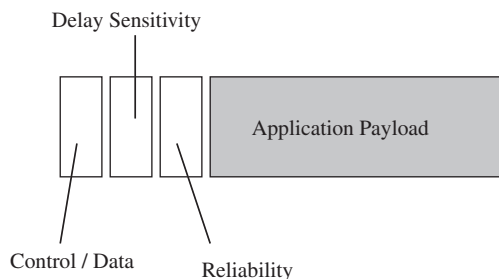


Fig. 1. Three-bit preamble appended to the application payload.

3.4.1. Universal beacon packet

We regularize the various beacons assumed by component protocols by the use of a universal beacon packet. The creation and maintenance of the neighbor table is performed by the exchange of this universal beacon at periodic intervals.

The beacon packet contains information such as the ID of the node, the advertised depth, a 1 bit congestion indicator, and a 1 bit energy available indicator. A node sets the congestion bit if 75% of its buffer capacity is full. Likewise, the node sets the energy bit if less than 25% of its battery life is available.

3.4.2. Neighbor table creation and maintenance

Even though there may be many potentially good neighbors available in the vicinity, there is a limit to the number of neighbors a node can maintain due to limited resident memory. It is crucial to be able to identify the best possible neighbors and retain them in the neighbor list [23]. We use a statistical approach to identify the goodness of a neighbor as we continue to retain it.

Neighbor table creation: At the start of the network, a node aggressively enters into its neighbor table every entry that advertises a depth lesser than its own. As time progresses, and owing to the presence of “gray” areas [23,26], a node continues to hear more often from certain neighbors and less frequently from others. Since the beacon exchange is uniform among all participating nodes, a node maintains a ratio of the number of beacons received to the number that *should* have been received since the entry was made. This ratio (p) gives a good indication of the quality of link to a neighbor by estimating the number of transmissions required as $1/p^2$ [4,23]. Link estimations apart, this ratio is helpful in establishing the true depth of a node. Consider a node which receives a beacon from a neighbor advertising a depth of n with a reception ratio of 0.3, and another neighbor with a depth $n + 1$ and a reception ratio of 0.9. The given node correctly infers its true depth to be $n + 2$, since it has a stronger and more stable link to the latter neighbor. In general, a node infers its true depth to be one greater than the neighbor with the largest reception ratio and least depth.

Maintenance and eviction: The maintenance of the neighbor table is governed by a timer driven “scan and update” (SAU) module. The module is invoked each time the timer fires, whose periodicity is equal to the beacon interval, and offset of 10 seconds. As a node continues to receive beacons from neighbors present in the table, the various fields advertised in the beacon are used to update entries in the table. Associated with every entry is a “last heard” bit which is set to true upon the reception of a beacon. SAU unsets the bit each time it is invoked. SAU also updates the reception ratio of a neighbor by using the last heard bit. An entry is evicted from a neighbor table if one of two things happen: (i) a node discovers that a neighbor’s depth is greater than its own or (ii) the reception ratio to that neighbor drops below 0.3 for a minimum statistical interval of 100 beacon cycles. Nodes evicted from the table are entered into a pool of blacklisted neighbors, who are not considered as potential neighbors for the next 500 beacon intervals. This prevents stale neighbors re-appearing in

Data(0)/ Control(1)	Real-Time(1)/ Non-Real-Time(0)	Reliable(1)/ Un-Reliable(0)	Inference
0	0	0	Unreliable, non Real Time, Data Packet
0	0	1	Reliable, non Real Time, Data Packet
0	1	0	Time Critical, Unreliable, Data Packet
0	1	1	Mission Critical Data packet
1	0	0	Unreliable, non Real Time, Control Packet
1	0	1	Reliable non Real Time, Control Packet
1	1	0	Real Time, Unreliable, Control Packet
1	1	1	Mission critical control packet

Fig. 2. Combinations of the preamble bits and their inferences for eight classes of traffic types.

the table and gives an opportunity to other potential neighbors in the vicinity.

The notion of a “good” neighbor quickly blurs when we have multiple routing components, each with its own yardstick of goodness based on the parameter that is to be optimized (reliability, delay, throughput, etc.). In general, entries are driven by their depth in the network more than by any other factor. We have found that this results in a good blend of neighbors with various values of depths, link qualities, congestion levels and energy values, and all of whose depths are lesser than that of the given node. Routing modules make ranged queries into the neighbor table to derive the next hop for a particular packet. However, it may so happen that a routing module fails to find any potential next hop candidate from the table. This could be either because the neighbor table is starved of good neighbors for that routing component, or due to inavailability of neighbors in the vicinity. When this happens, the SAP module is triggered with information about the routing component, and it marks for an insertion of an entry that matches the routing components needs from future beacons. This could possibly lead to an eviction of an entry from the table. We use the LRU algorithm to choose an entry for eviction.

3.5. Decomposing routing protocols to core components

The routing layer consists of a collection of routing components, with each component optimized for a certain class of traffic. Like the application payload, the component protocols also publish 3 bits that advertise their suitability for a particular application. It is crucial to

decompose component protocols at the right granularity to allow rapid protocol development. As Cheng et al. [1] note, choosing the granularity at which a protocol is to be decomposed is highly challenging: fine grained decomposition will result in un-necessary run time overhead, while too large a granularity will fail to leverage code sharing among components and could result in significant re-implementations. They show that a composable set of protocols that share common code result in smaller memory footprints, and are in general lucrative considering resource scarcity on a mote. While the precise needs of future protocols are highly debatable, we extend their work by decomposing a routing protocol to the following components: the dispatcher, the naming and addressing unit, the forwarding unit, and the scheduling unit. The internal layout of a typical component protocol is shown in Fig. 3.

Dispatcher: The first step in accepting a packet is to check the preamble bits to establish the right unit to which the packet is to be forwarded. The dispatcher is agnostic to intricacies of a protocols implementation or its naming and addressing format, and is shared by all the component protocols.

The *naming and addressing unit* (NAU) is the component that understands the addressing format for a protocol. Different addressing units are used by various protocols: while some protocols assume flat ID's, some other expect gradients or location information. Owing to the broadcast nature of the wireless medium, a node may receive a packet that it is not intended to. In effect, the NAU determines if a packet has made it to the destination, or if it needs to be forwarded further.

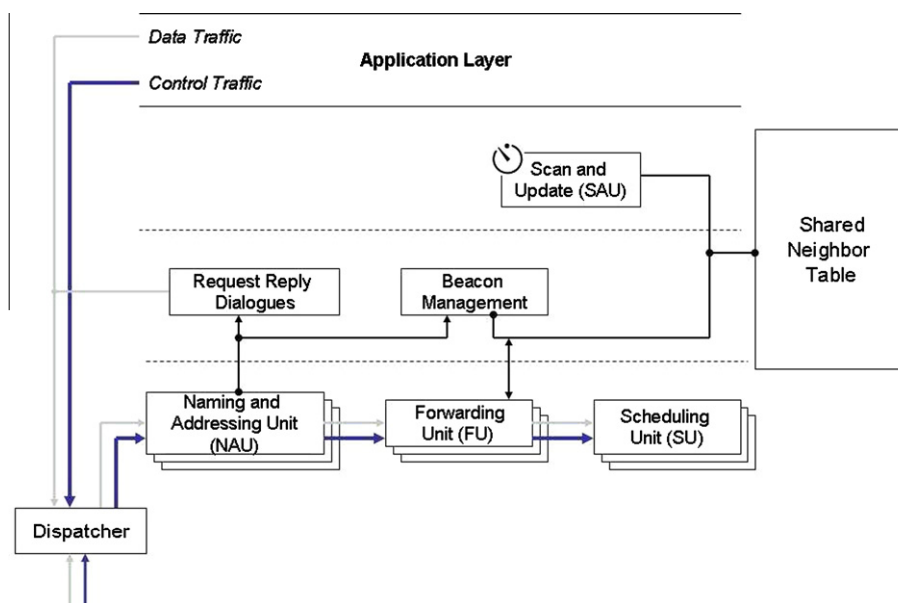


Fig. 3. Internal components of the routing protocols.

The *forwarding unit* (FU) is primarily concerned with establishing the next hop for a packet. Packets make it to this unit if the given node is not already the destination. The FU arrives at the next hop by making a ranged query into the shared neighbor table. For example, a routing protocol that selects a route based on link qualities, congestion and depth at its neighbors would make a ranged query of the form $f(link_quality, congestion, depth)$ into the neighbor table to arrive at a list of potential neighbors. The FU then applies its set of optimizations to arrive at the best neighbor(s) to which the packet is to be forwarded. In general, the NAU in conjunction with FU deal with protocols that use different routing and addressing units.

The *scheduling unit* (SU) is a buffer data structure that can order packets awaiting transmission. It can internally schedule the order in which packets are to be forwarded to the link layer for injection into the wireless medium. Certain protocols might need to buffer packets for potential retransmission, while other might need to hold packets to perform aggregation stalling for similar information to arrive. Certain other protocols might want to reschedule the order of injecting packets into the link layer to transmit mission critical alerts ahead of regular traffic. We note that the SU performs some actions (like retransmissions) above the link layer. In the traditional, monolithic protocol stack design, the link layer has a set of actions that *universally* apply to all packets passed from the higher layers. For example, consider a packet from an application that demands reliability, while a packet from another that does not. Both these packets will be subject to the *same* link layer policies irrespective of what they demand. Our intent with the dynamic routing is to offer reliability for packets that require it, and not to the others. In this regard, retransmissions are handled by the routing layer in the way it passes packets that want reliability to the link layer: this design makes no assumption about the functionalities of

the link layer (which may or may not support retransmission).

Apart from these units, there are certain other units in the routing layer that accept packets for further processing. For example, beacon packets make it to insertion/eviction module, which considers the beacon as a potential neighbor. Likewise, control traffic originating from the base station is a typical query into the node, to which the node responds by performing certain local computations. The shared neighbor table is controlled by the scan and update unit, which periodically scans through the neighbor table to evict stale neighbors and enter them into a blacklisted pool.

4. The dynamic routing framework

We show how the 3 bits drive customization and protocol selection within the dynamic routing framework. Apart from communication requirements of reliability or delay tolerance, the bits fundamentally divide the traffic as being of control or data type. Control or data traffic do not necessarily demand differential routing in terms of shorter or longer paths. In fact, most of control traffic (like beacons or ACKs) are for one hop use only. Beacons in particular are simply broadcast, and require no route selection or optimization. Communication requirements for control and data packets are best characterized by a need to be *scheduled* differentially inside the framework, and then, routed optimally. We accommodate differential treatment by the use of separate virtual queues for control and data traffic, while we dynamically switch routing components based on demands for reliability and delay.

A detailed interaction diagram of packet with the framework is shown in Fig. 4. Application presents a blend of control (C0–C3) and data packets (D0–D3). The suffix indicates the status of the reliability and real time bits.

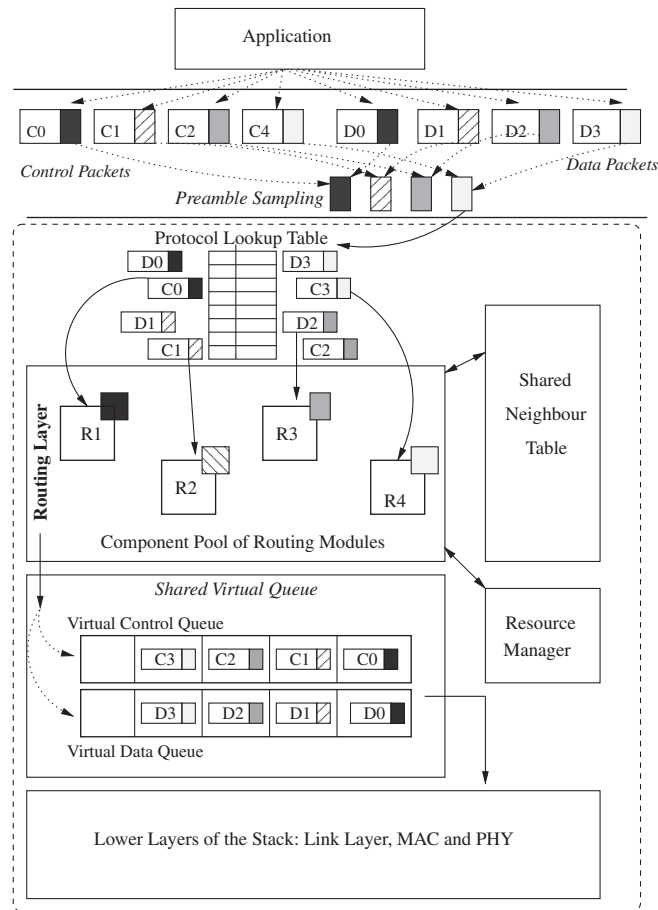


Fig. 4. Overview of the dynamic routing framework: preambles announce data requirements that are mapped to routing components.

For example, packet D3 would have preamble bits set to [0, 1, 1], with the first bit indicating a data packet and last 2 bits account for the suffix 3. Selection of a routing component is driven by the suffix number. In other words, both D_x and C_x are offered the same routing component. Data or control traffic, however, are scheduled differently after route selection is established. Two virtual queues, one each for data and control traffic, take the incoming packets and schedule them for transmission to the lower layers of the stack.

4.1. Flow based network functionality

We provide a brief description of the routing protocols that cater to the eight traffic types. Table 1 lists the various flow types, and the routing optimization (or modules) used to construct network layer functionality for each type of flow. Since data and control traffic are routed similarly, we have four different flavors of network layer functionalities.

4.1.1. Reliable, non-Real Time Routing

Numerous approaches have been taken for this class of routing in the past [5,18,23,24], to name a few. Routing emphasizes reliable delivery over time to deliver. A maxi-

imum of three link level retransmissions are used to ensure successful reception [16]. Since a retransmission is costly in terms of energy and bandwidth, the protocol selects the strongest link to a neighbor closer to the destination. High estimations of link quality, or a form of ETX [5], is the most lucrative option here. The packet takes numerous short hops of high quality links to make it to the destination with minimum losses. Hence, speed is compromised by the use of numerous short and reliable hops. Delay tolerance for this class of traffic make it highly conducive to aggregation, since the scheduling unit can stall for similar data to arrive. Nodes with growing congestion and low energy are avoided as much as possible as the next hop.

4.1.2. Unreliable, Real Time Routing

Certain classes of traffic have a time associated with them, after which the data is virtually of little or no use. Speed of delivery is emphasized, and losses can be tolerated to a certain degree. Various protocols have been proposed for this class of traffic [2,8,17]. Our approach greedily forwards traffic to the base station primarily based on depth, avoiding congestion and low energy nodes, and ignoring link quality estimates or indications of poor packet reception.

Table 1

Table showing preamble bits, their inference and flow based routing optimizations.

Preamble bits			Flow inference	Routing optimization					Traffic type
Data(0)/control(1)	Real time(1)/non-RT(0)	Reliable(1)/unreliable(0)		Aggregation	LQI thres.	Depth weightage	Multiple copies	RetX	
0	0	0	Unreliable, non-real time data E.g., low priority sensed data	✓	–	X	X	X	Type 0
0	0	1	Reliable, non-real time data E.g., reliable data transport	✓	>0.75	X	X	✓	Type 1
0	1	0	Time critical, unreliable, data E.g., Real time data streams	X	>0.35	✓	X	X	Type 2
0	1	1	Mission critical data flow	X	>0.5	✓	✓	X	Type 3
1	0	0	Unreliable, non-realtime, control E.g., beacons, ACK's, etc.	X	–	X	X	X	Type 4
1	0	1	Reliable, non-real time, control E.g., Queries, commands	✓	>0.75	X	X	✓	Type 5
0	1	0	Time critical, unreliable, control E.g., Real time remote control	X	>0.35	✓	X	X	Type 6
0	1	1	Mission critical control	X	>0.5	✓	✓	X	Type 7

4.1.3. Reliable, Real Time Routing

This is a contradictory requirement, where packets need to maximize their chances of reception and yet want the shortest path to a destination. Waiting time for ACKs and retransmissions only add to the delay of the packet in transit. To meet both delivery deadlines and thwart link losses, the routing protocol first chooses nodes with the minimum depth. Among those, nodes with the highest link quality are chosen, and the protocol injects *multiple* copies of the same packet to minimize link losses. The exact number of duplicates is based on the estimated link quality of a particular node. For example, if a neighbor with a link quality of p is chosen, the protocol transmits $\lceil 1/p \rceil$ packets with the same data. This form of routing is costly in terms of resources, but this type of traffic is reserved for mission critical alerts which is arguably rare in the network.

4.1.4. Unreliable, non-Real Time Routing

This form of routing is applied to packets that neither demand reliable transmission, nor demand a speedy delivery. A good fraction of sense-and-disseminate traffic would fall into this category, where data values exhibit significant redundancy. Every packet adds little value to already existing information in the network, perhaps even repeating known observations. This form of traffic is predominant in delay tolerant networks and is an excellent candidate for aggregation, where the scheduling unit can stall for long periods of time to effectively aggregate observed results to reduce the traffic to a handful of packets. Our form of routing makes a compromise on both link quality and depth to route packets to the destination. Alternatively, this form of data centric routing could also utilize a minimum Steiner tree to maximize chances of aggregation.

4.2. Virtual queues and pull processing

Packets that egress the routing modules would need to be passed onto the link layer queue for subsequent transmission. Scheduling packets into the link layer queue is controlled by two virtual queues, one each for data and control traffic. The virtual queues do not maintain any

independent buffer of their own: they only maintain the order in which packets will be forwarded. While packets await their transmission, they are physically housed in the scheduling unit of their respective component protocol. The scheduling unit internally marks a packet to be ready for transmission after it performs its set of optimizations (e.g., aggregation, reordering, etc).

Modules after the virtual queues implement pull processing, where the transmitting interface dictates packet intake from the higher layers. There is also a flexibility to implement various scheduling policies. For example, we have implemented *fair queuing* from the virtual queues themselves to the link layer. This ensures that an even mix of control and data traffic are output from the framework, even though one or the other type may have a different packet arrival rate. Other scheduling policies like flexible power scheduling, epoch-based proportional fairness, round robin, weighted round robin, etc. can also be easily implemented.

5. Application background

With our dynamic routing framework in place, we proceed to analyze the various intricacies of a fairly complete large scale deployment by choosing predictive maintenance (PdM) as an example target application. We begin by gathering the various communication demands of this deployment. We employ our 3 bit scheme in exposing these requirements to the communication framework. Using these requirements, we chart a comprehensive traffic workload to drive our evaluation of the dynamic routing framework.

5.1. Predictive maintenance

Productivity is a key weapon for manufacturing companies to stay competitive in a growing global market. Increased productivity comes through increased availability. The need to be continuously available has driven many companies to focus on effective maintenance strategies. PdM is a set of techniques which help determine the

condition of in-service equipment in order to predict when maintenance should be performed. PdM uses a variety of vibration analysis, oil analysis, infrared thermography and ultrasonic detection with an objective of reducing catastrophic equipment failures, and the associated repair and replacement costs much before the failure occurs. PdM enables machinery stakeholders to monitor, access, predict and in general understand the working of physical assets.

One alternate way of implementing PdM is by use of on-line motoring systems of wired sensors that can continuously gather statistics. Wired sensors are constrained by a 1:1 point-to-point connectivity links, and such systems are expensive to procure and install. This trend in general has limited their penetration into the market. In fact, most small to medium sized companies would not procure them owing to their higher cost and lower return for investment, resulting in their mere 10% market penetration [11]. Sensornets break ground by promising to be cheaper, providing higher returns on investment, flexibility owing to a broadcast wireless communication pattern, self-healing abilities when it comes to node failures and adapting to changing network conditions. Hence, there is great potential for sensornets to tap into the market provided they can completely satisfy all of a deployers requirements by allowing them a greater flexibility in communication patterns.

The case for PdM as a potential killer application for sensornets has been investigated by Krishnamurthy et al. [11]. They create an excellent groundwork for PdM using sensornets by demonstrating a viable return on investment. However, their focus is primarily on the effect of hardware architectural impact on performance. Their study is limited to PdM vibration analysis, where the deployment generates just one type of traffic: vibration signatures that need to be reliably transported to a base station. We significantly extend PdM's requirements in terms of complete *communication* requirements from a deployers point of view by enumerating various communication patterns possible. Accommodating the various communication patterns in such deployments is an important problem to solve. This is because the cost of deploying and running such an infrastructure would be dominated by *software and service management*, while diminishing hardware costs would make it cheap to network hundreds of motes. In effect, our study builds upon and complements their work, and we show that sensornets can be a powerful platform to perform PdM.

5.2. Requirements

Most deployers want a system of sensors to autonomously report sensed values and raise alerts, while they also want the presence of a human element in the process. Human intervention can be best characterized by an ability to interact with the deployment (query-response), apply patched code updates, reprogram the network with newer protocols, manage or change various threshold values and so on. In general, there is a constant need to evolve, interact and update the deployment with changing requirements, which allows for continuous customization of the sensornet behavior.

Based on our survey of PdM requirements, we have found that deployers wanting to adopt sensornets as a means to build their maintenance regime demand the following aspects:

Req. 1. Periodic reports: Deployers expect the network to report sensed values on a regular interval. Such statistics make way for monitoring, analysis, trend forecasting and prediction of equipment behavior which can help them chart maintenance schedules. Apart from maintenance predictions, it also helps assess performance of new machinery within a warranty period.

Req. 2. Streaming Real Time Values: Apart from periodic updates of sensed values, there are numerous instances when there is a need to report values in *real time*. Examples could include protected zones with cameras which need to capture a video of a personnel when he walks into the facility. This would require sensors to report co-ordinates in real time to camera installations, which would then pan and zoom to capture streaming video.

Req. 3. Query-response: Users or administrators need to query and get a response from the network at will. As with any interactive system, there is a need to maintain short turn around times within user irritation levels. In fact, there are various levels of accuracy and delay associated with each response. Some responses need to be time critical, while others demand accuracy with more acceptable levels of turn around times. In general, while keeping turn around times low, there is a need to model such subtle variations even within these interactive dialogs. Deployers further stress on the need to keep high levels of interactivity at all stages of the deployment, especially when mission critical events occur.

Req. 4. Continuous Customization: Requirements are not static, and would continuously evolve with time. This means that administrators see a need to constantly update or change network behavior. Examples could include changing threshold values for alerts, commanding specific motes to raise or lower sensing fidelity and so on. This in general emphasizes robust communication between an administrator and the network, and could even involve minor and incremental changes to network software.

Req. 5. Network Reprogramming: Deployers want the liberty to completely reprogram an entire sensornet deployment when a more stable or efficient communication suite is available. Such modes of communication emphasizes on the need to reliably bulk transfer large pieces of code to the entire deployment.

Req. 6. Mission Critical Alerts: Apart from the above modes of communication, there are certain mission critical events which need to be reported reliably in as short a time as possible. Such events may trigger a cascade of events within the network. In fact, there is a need to accommodate a large number of interactive queries and commands to various motes when such events occur. In short, when serious anomaly occurs in the deployment, there is a need to gracefully alert, shutdown and recover as much as possible before the situation exacerbates.

These set of requirements are not isolated to the case of PdM alone. Virtually any practical sensor network deployment, if put under a microscope, would usually result in an enumeration of similar requirements to take place *concurrently* throughout the lifetime of a network with varying levels of emphasis on one requirement or another.

The requirements reveal significant variations in terms of communication needs. Variations exist in terms of reliability, time criticality, mission critical alerts and levels of interactivity with the deployment. We proceed to show how these set of requirements can be made visible to our framework using just three intent bits.

5.3. Translating requirements using 3 bits

We show how just 3 bits can be used to make visible all of PdM's requirements to the communication framework. We proceed by identifying the nature of traffic (data/control) from the requirements, and subsequently analyze requirements of reliability and delay in transit.

The first step is to distill control from data traffic. We broadly define data traffic to be traffic that a mote creates using a sensed value, for consumption at its destination. Control traffic is all of those packets that are used for control plane management (e.g., beacons), and traffic from end users. User generated traffic includes queries, responses, commands, and code updates. This is done to ensure that user traffic, along with control plane traffic, is scheduled differently from sensed data. While traffic is normally low during normal operations, critical events in the facility are usually coupled with a surge of traffic in the network. It is precisely at such times that congestion begins to surface [22]. And again, at such times, it is highly likely that an end user or an administrator will issue queries, commands or updates into the network. In effect, while the data plane reports values of interest, user interactivity and network management at such times are not compromised. For the rest of the discussion on mapping various requirements using 3 bits, refer to Fig. 2.

Data traffic can be easily grouped as per their requirements of reliability and speed of delivery. PdM requires periodic reports from every sensor to build a statistical record of the facility (Req. 1). This type of traffic publishes $[0,0,1]$: data traffic demanding high reliability and no urgency of delivery. Real time streams (Req. 2) would publish $[0,1,0]$, stressing on urgency and lesser demands of reliability from the network, since the payload could be rendered useless if reception is delayed. Mission critical traffic (Req. 6) would publish $[0,1,1]$, emphasizing on both urgency and reception.

Beacon packets publish $[1,0,0]$. Since they are usually broadcast into the medium, there is no notion of reliability or time to deliver. Interaction with the network (Req. 3), however, can be modeled at very fine granularities with these 3 bits. Using the axes of reliability and urgency, four levels of interaction are made possible. For example, queries of type $[1,1,0]$ would result in very fast turn around times, but with compromised accuracy. Interactions using $[1,0,1]$ preamble would be high on accuracy, but with longer turn around times. Critical real time alerts or updates shall present $[1,1,1]$, which would ensure short turn

around times *and* high accuracy. Packets for patched updates, commands or network reprogramming (Reqs. 4 and 5) would publish $[1,0,1]$, which stress on reliable delivery.

We conclude our mapping by summarizing the criteria to validate PdM's requirements: (i) Types 1 and 5 should have high delivery ratios compared to other forms of traffic, with an acceptable compromise on transit delay; (ii) Types 2 and 6 should have short transit times, with a compromise on reliability; (iii) Types 3 and 7 traffic should have high delivery ratios *and* short transit time and (iv) interactivity with the deployment should be maintained at all time, even in times of growing congestion and mission critical events. This in general emphasizes differential treatment to data and control traffic.

5.4. A realistic traffic workload

We use PdM's requirements to create a realistic traffic generation scenario of a typical operation to drive our evaluation. We are interested in fairly large scale operations, and seek to identify the various activities that shall typically take place over a large window of time. We reword the requirements in terms of traffic generation, and create a comprehensive workload to drive our evaluation.

All traffic from the motes converge towards the base station. The base station issues queries to random motes, and replies are likewise routed back to the base station. The base station applies patched updates to specific motes, while network reprogramming is applied to every mote. Every mote generates a sensed value at a periodicity of one hour (Type 1), which needs to be reliably transported to the base station. This would help the deployment build a statistical monitoring record of the facility to drive maintenance. Nodes also generate asynchronous real time streams of value, five packets at a time (Type 2). Real time streams are generated at a mote with a probability of 0.1 at any instant of time, which needs to be transported to the base station as fast as possible. End users generate queries at will modeled as a Poisson process. Each query has a random flavor to it (Types 4–7), some queries demand high interactivity (small turn around times), while other demand high accuracy (commands, queries or patched updates). The administrator performs a network wide patched update roughly once a week, when the network is fine tuned to slightly update or customize behavior (Type 5). Nodes observe mission critical data with a probability of 0.05, and generate five packets at a time (Type 3). Each time this happens, it triggers a surge of real time streams of values at motes close to the phenomena. The situation is responded to by the user, who generates multiple queries to the zone, and with a probability, issues numerous commands and patched updates.

We use this pattern to generate synthetic workloads for a 9-week evaluation period of a large scale plant employing PdM. The generation is better captured in Fig. 5. Unless otherwise stated, we consistently use this traffic workload for every experiment.

6. Simulation results

We evaluate our dynamic routing framework by both simulations and experiment on testbeds of MicaZ motes.

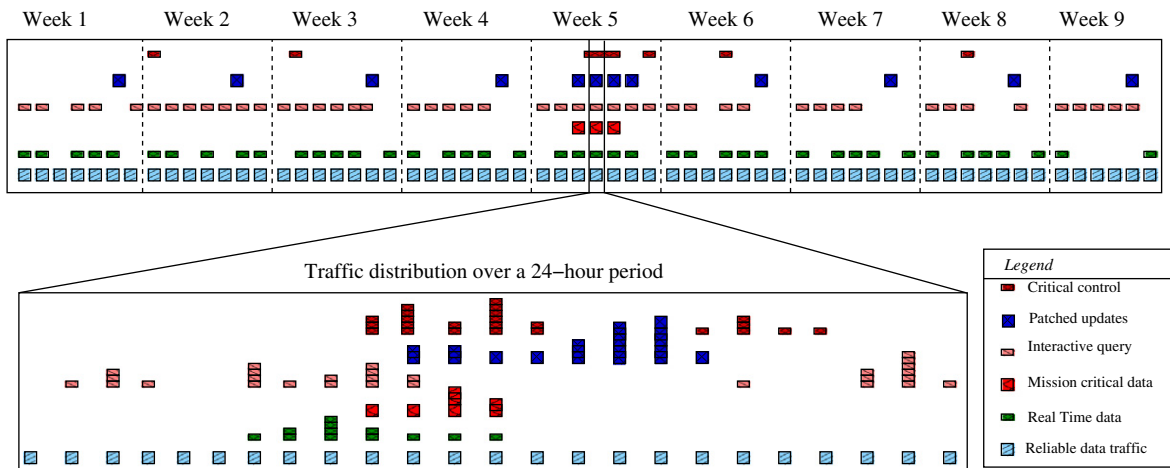


Fig. 5. Traffic generation for a 9-week PdM derived from requirements. The workload generates periodic samples, real time streams and mission critical alerts. The user/administrator interacts with the deployment at will with queries, weekly patched updates, and one major network reprogramming.

We resort to a simulation based study to analyze behavior at scale, where we simulate behavior for thousands of motes.

We designed a packet level simulator in Java, and we modeled link behavior between any pair of nodes is closely modeled around results by Woo [23] and Zhao [26]. Using that model, we also implement the typical broadcast behavior of all packet transmissions and receptions that take place in the network. As our next round of results, we also implement our framework and evaluate it on a 40-node MicaZ testbed. Though a simulator tends to hide certain physical characteristics of actual motes, it allows us to demonstrate the effectiveness of our framework for operations at various scales and densities. Nevertheless, we make every effort to model network behavior to as close to reality as possible (Table 2).

6.1. Evaluation criteria

Evaluation is driven by a comprehensive 9-week long workload detailed in Section 5.4. Nine weeks of simulation time correspond to approximately 12 h of wall clock time to perform the simulation. This workload was chosen to better mimic the traffic generation rate to as close to reality as possible, thereby helping us gain an insight into long term service differentiation for operations at scale. The workload generates a good blend of the various traffic types, while carrying out the requirements of PdM. Our first set of results analyze behavior of a dynamic routing

framework at scale. We vary the number of nodes from 4 to 1024, and show how PdM's subjective requirements for various traffic types are met at scale. In effect, we show that our framework can adapt to the demands of: (i) reliability for traffic Types 1 and 5; (ii) the real time nature of Types 2 and 6; (iii) the mission critical nature of Types 3 and 7, which demand both reliability *and* short turn around times; and (iv) interactivity with the networks using Types 5–7². Table 3 outlines the mapping of PdM's requirements to various traffic types. In effect, we show that such diverse communication patterns can co-exist, and also meet their subjective demands. Since the workload used mimics PdM operations, our results in turn validate the deployments goal for operations at scale.

6.2. Delivery ratio

We begin by analyzing the average end-to-end success rate for all the traffic generated by the application (Fig. 6a).

We see good delivery ratios for reliable data traffic (Type 1), reliable control traffic (Type 5) and mission critical control information (Type 7). High delivery ratio for Type 1 traffic validates PdM's expectation of samples from every mote at periodic intervals. Type 5 traffic denotes interactivity with demands on accuracy, which are likewise met. Reliable traffic in general is driven by a retransmission upon failure, which significantly raises chances of successful packet reception. Similarly, the odds that mission critical traffic (Type 7) makes it to the destination are also high. Mission critical control traffic would be routed through the control queue, and it maintains a high delivery ratio of around 0.9. Delivery ratio is poor for real time data traffic (Type 2), which is routed greedily based on a shortest path algorithm.

While similar routing is applied to both Types 1 and 5 traffic, the delivery ratio of reliable data (Type 1) dips a lit-

Table 2
Simulation parameters.

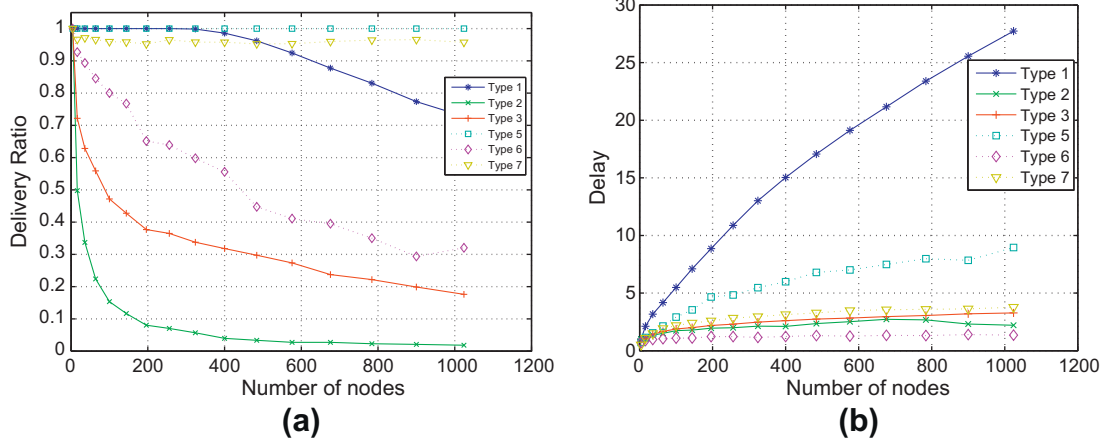
Parameter	Default value
Packet size	29 Bytes
Queue size	50 Packets
Grid size (max)	100 × 100
Internode spacing	10 ft.
Packet reception	Broadcast
Beacon interval	20 s

² We do not generate Type 0 traffic, since none of PdM's requirements map to this. Also, we do not profile Type 4 since it is made up of beacon and ACK packets in our workload.

Table 3

PdM's requirements mapped to various traffic types.

Preamble bits	Inference	PdM Req	Traffic type
[0,0,0]	Unreliable, non real time packet; exhibits significant redundancy	X	Type 0
[0,0,1]	Reliable data traffic, demands high throughput; agnostic to delay	Req. 1	Type 1
[0,1,0]	Real time data stream, demands short delivery time; agnostic to loss	Req. 2	Type 2
[0,1,1]	Mission critical data, demands reliability and speedy delivery	Req. 6	Type 3
[1,0,0]	Unreliable, non real time control packet	Beacons, ACKs	Type 4
[1,0,1]	Reliable control traffic; agnostic to loss	Reqs. 3–5	Type 5
[1,1,0]	Real time control packet; demands speedy delivery	Req. 3	Type 6
[1,1,1]	Mission critical control; demands reliability and speedy delivery	Req. 6	Type 7

**Fig. 6.** (a) Delivery ratio for different traffic types and (b) end-to-end delay for different traffic types.

tle with increasing number of nodes as compared to Type 5. This is mostly because data traffic is mapped onto a separate virtual queue than control traffic. While Type 1 would face rising congestion with increasing number of nodes, Type 5 traffic hardly witnesses any of this. Likewise, delivery ratio for real time queries (Type 6) are significantly higher than real time data streams (Type 2). Mapping data and control traffic in separate virtual queues has enabled us to provide high levels of interactivity with the network.

6.3. End to end delay

We profile the average end-to-end delay experienced for all traffic that makes it to the destination. Delay in the network is primarily a function of the number of hops a packet takes, and the queuing delay at every hop. As a packet increases its hop count to destination, end-to-end delay intuitively increases.

The average end-to-end delay for all traffic types for increasing number of nodes is shown in Fig. 6b. All traffic demanding speedy delivery experience short transit time. Real time data experiences least delay (Types 2 and 6), reliable traffic experience maximum delay (Types 1 and 5), while mission critical traffic experiences delay that is only marginally more than real time data.

Small delay profiles for Types 2 and 6 directly validate PdM's requirements of real time streaming communication. Likewise, short delivery times of mission critical traf-

fic (Types 3 and 7), coupled with their high delivery ratios, validates PdM's requirements for mission critical communication. High delay values for Types 1 and 5 only emphasizes on their ability to select numerous short hops of high quality.

The interplay of control and data traffic, which receive differential scheduling due to separate virtual queues, is also captured in the plot. Overall, data traffic experiences more delay than control traffic. For example, though Types 1 and 5 are offered the same routing, the overall delay for Type 1 is almost twice that of Type 5. This highlights the ability of the framework in meeting PdM's overall requirements of maintaining network interactivity. With PdM's overall objectives met, we now analyze the causes of losses in the network.

6.4. Link losses

Effectiveness of the various routing components in their ability to choose the right path for every traffic type is best characterized by profiling the link loss distribution. Link loss is a typically attributed to the unreliable wireless medium, where packets are corrupt or lost while in transit. Routing components that stress on reliability need to understand the nature of links, and use these to derive a suitable next hop while keeping the requirements of the payload consistent.

We profile link losses for various traffic types in Fig. 7a. As the number of nodes in the network increases, so does

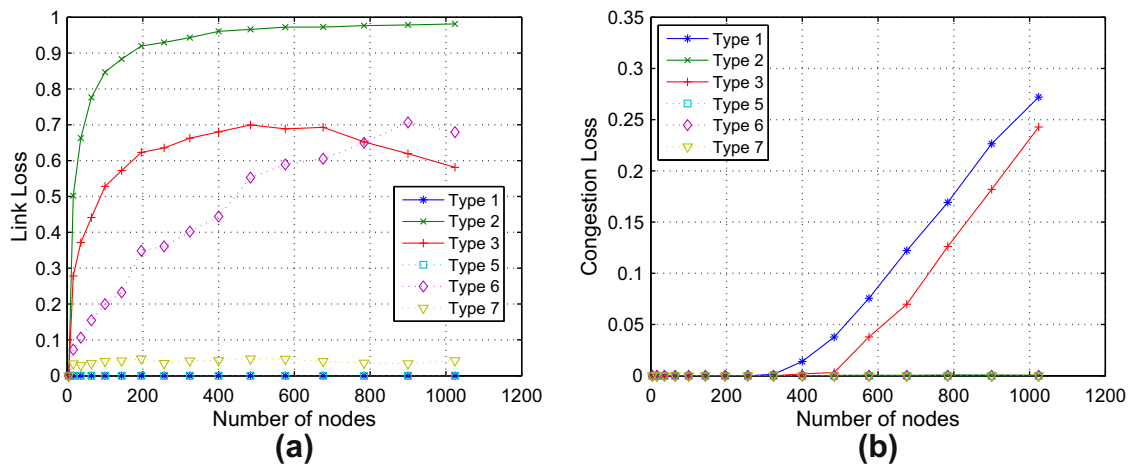


Fig. 7. (a) Fraction of packets loss due to link losses and (b) fraction of packets lost as congestion drops.

the effective number of hops that a packet takes to reach its destination. This in effect increases the probability of a link loss. Real time data streams (Type 2) experience maximum link losses, largely because of the nature of route selection which greedily forwards traffic to nodes closest to the base station. Reliable traffic (Types 1 and 5), however, make ranged queries into the neighbor table with high thresholds of link estimates. Likewise, they experience nearly zero link related losses in the network. Because of inter-node spacing in this experiment (10 feet), neighbors closest to a node do not fall over into the gray area. Mission critical alerts (Type 7), likewise experience low values of link losses since they thwart link error by multiple copies per packet transmission.

6.5. Congestion losses

Congestion occurs when nodes inject more packets than the network can handle. While our workload generates traffic that can normally be serviced by the network, congestion does occur for a variety of reason. First, all data traffic is destined to one node (base station). Hence, all of the network's traffic converges towards nodes closer to the base station to be routed via them. Even though we try to avoid congested nodes in route selection, a point comes when all neighboring options for a node are congested. Congestion particularly increases with rising number of nodes in the network, which simply translates to rising traffic levels for nodes near the base station to service. Based on PdM's requirements, we also notice that congestion is likely to occur when serious anomaly is detected. When a mission critical failure is noticed, a surge of events takes place in the network. Nodes report mission critical alerts, and some other nodes in the vicinity would begin to send streams of real time values. The end user or administrator would add on to this by issues commands, queries and triggering actions. In our workload, both these causes are sufficiently represented. We now analyze the role congestion plays in the network, and profile the various congestion related losses for the traffic types.

The fraction of packets lost due to congestion are shown in Fig. 7b. For network scales of a few hundreds of nodes, congestion is not really a pressing problem because of the low duty cycle of nodes. However, congestion starts to surface for networks with more than 300 nodes, primarily because of increased load on nodes closer to base station. We notice that Type 1 traffic witnesses maximum congestion related losses. As packets begin to approach the base station, traffic from other types (real time streams or mission critical alerts) would try to avoid congested nodes nearby and choose low quality links with faster transit times. At this same stage, reliable traffic would take two or three additional hops to ensure high quality links.

It is interesting to see that mission critical data (Type 3) also experiences congestion losses. This has a few implications for congestion control in general. When mission critical anomaly is detected, activity of nodes suddenly peaks. Various nodes start to simultaneously inject traffic into the network. Congested links, coupled with multiple copies per packet from Type 3, only makes matters worse for mission critical data. This suggests that dropping *any* packet in a FIFO manner, as most current congestion control schemes do, only undermines performance. In general, utilizing information about nature of payload and dropping packets of relatively lesser importance should be an added metric to future congestion control algorithms. Lastly, we also observe that control traffic (Types 5–7) do not experience congestion drops. This means that even in times of congestion, interactivity is kept high because control traffic is offered differential scheduling. This further validates PdM's requirements of maintaining high interactivity with the network even in times of congestion and mission critical events.

6.6. Interactivity with deployment

While the effects of scheduling control and data traffic differentially are brought out, we seek to understand the interplay of various types of interactive control traffic within the virtual 'control' queue. Three levels of

interactivity are made possible by the use of preamble bits: reliability driven queries (Type 5), real time queries (Type 6), and mission critical interaction (Type 7). We analyze the average round trip times (RTT) for various kinds of queries into the network. Our workload generates queries to random nodes in the network at various distances. For a 9-week long interaction, we summarize the interactivity times for networks at scale.

The interaction RTTs are plotted in Fig. 8. Dynamic routing plays a major role in ensuring that interactivity times are kept low for real time queries (Type 6), acceptable for mission critical queries (Type 7) and relatively higher for reliability driven queries (Type 5). Coupled with high delivery ratios of Types 5 and 7, and short turn around for Type 6, we successfully meet the subtle variations in interactivity demanded by PdM.

6.7. Average path distribution

We finally characterize the path distribution statistics for various traffic types in the network (Fig. 8). This simulation was run for a collection of 1024 nodes arranged using a 32×32 grid, with a 10 ft. inter-node spacing. For every packet received at the base station, we measure the number of hops that it took build a frequency distribution for various hop counts. The curve is representative of route selection since each traffic type generates sufficient number of packets at various distances from the base station.

Requirements of PdM apart, nature of route selection is best captured in this plot. Reliable traffic (Types 1 and 5) take numerous short hops of high quality links, and register large hop counts. Real time traffic (Types 2 and 6), which is routed greedily based on shortest paths, takes the least number of hops. Mission critical data are offered hops that range in between reliable and real time traffic.

7. Testbed implementation

This section presents results from an implementation perspective of the dynamic routing framework. Our moti-

vation behind implementation was to: (i) establish a proof of concept (sanity check) that dynamic routing is indeed feasible on real hardware and (ii) to measure the overhead incurred by replacing monolithic designs with dynamic protocol behavior. We compare the overhead incurred against the simplest routing protocol possible (serial-forwarder) and MintRoute, the default TinyOS routing module.

7.1. Code size and memory footprint

We implemented our framework on MicaZ [27] motes. We report on code size and memory footprint of our implementation. As our next round of results, we also measure the processing overhead that our framework incurs. Code size refers to the program memory occupied by the routing logic. The default TinyOS routing logic (MintRoute) has a code size of 2548 bytes, and a more processing intensive code such as BVR [6] results in 6542 bytes of code. A routing layer composed of four independent routing protocols, as in our case, would result between 12,000 and 15,000 bytes of code. However, due to a modular and composable architecture, and shared resources such as the neighbor table, we were able to implement our architecture in just 10,097 bytes of code.

Memory footprint is the amount of RAM allocated to the running code. Space constraints call for minimizing memory consumption. On an average, most independent routing protocols take about 1500 bytes of RAM. For example, MintRoute consumes an average of 1497 bytes, while BVR consumes 1428 bytes. On extrapolation, four concurrent routing protocols would consume an average of 7500 bytes of RAM. Again, due to resource sharing, our implementation consumes 2800 bytes of RAM.

7.2. Processing overhead

A dynamic network layer would intuitively consume more processing. To measure the processing overhead, we set up an experiment consisting of three MicaZ motes:

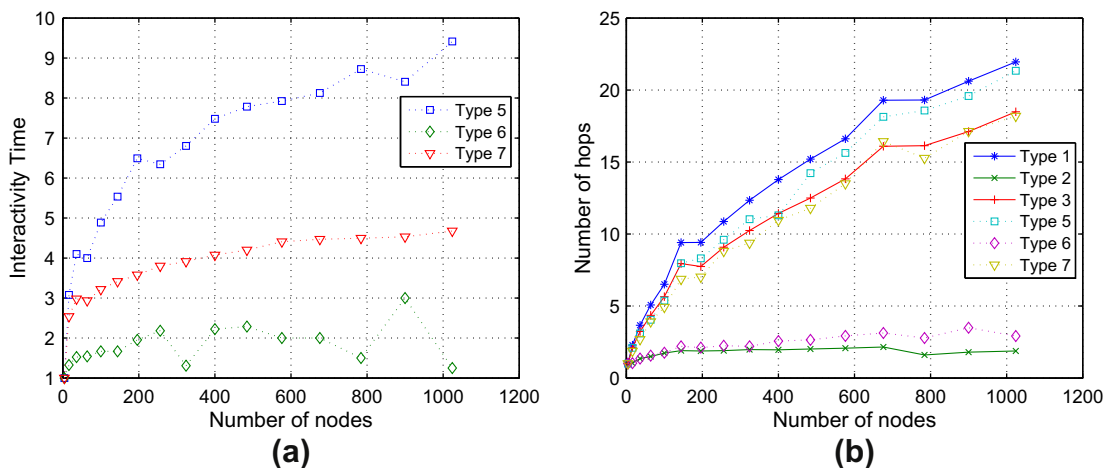


Fig. 8. (a) Average round trip times for interactive queries with the deployment and (b) path distribution statistics for various traffic types for a deployment of 1000 nodes.

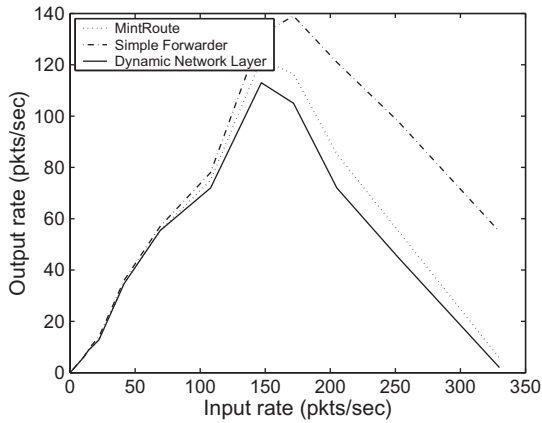


Fig. 9. Measuring processing overhead by recording the output rate of packets through a forwarding node for different input rates on a 3-node MicaZ testbed.

a source, a forwarder (where processing is tested), and a base station connected to a terminal arranged in a straight line. The source sends packets at a uniform interval using a timer, and sends up to 10,000 packets of 29 bytes each destined for the base station. The forwarder stores, processes, and forwards the packets that it receives from the source to base station. The source and base station are outside radio range of each other. In effect, packets traverse up and down the processing modules of the forwarder's stack. The destination (base station) passes packets to a terminal which counts the number of packets received, and sets up a comprehensive log of events.

We measure the output rate (at the terminal) for a range of input rates from the source. To better measure the processing overhead, we also measure the processing of MintRoute (TinyOS's standard routing protocol), and a simple forwarder. The simple forwarder takes packets and sends them all out to node-0 (base station), denoting the least amount of processing possible.

The plot for this is shown in Fig. 9. For small input rates, the output rate closely trails the $y = x$ line. Our dynamic

networking framework forwards up to 153 packets/sec, amounting to 4437 bytes/sec. Inputs above this rate exhibit a livelock [13]. This means that more processing is spent in servicing interrupts of packet reception than forwarding packets. In comparison, MintRoute forwards up to 170 packets/sec while simple forwarder can process 201 packets/sec. This amounts to 11.7% processing overhead compared to a MintRoute, and a 25.31% overhead compared to a hypothetical simple forwarder which sends all packets to the base station.

We now turn to evaluating our framework by implementing them on a 40 node MicaZ [27] wireless testbed arranged in a rectangular 8×5 grid. The experiments were conducted in an indoor lab environment.

Having understood the effectiveness of a dynamic routing framework for networks at scale from simulations, we begin with analyzing the effect of varying node densities and analyze link and channel losses. Finally, we validate PdM's requirements by analyzing behavior over windows of time.

7.3. Validating PdM requirements on the testbed

We finally turn our attention to characterizing our ability to meet PdM's requirement. PdM's requirements emphasize on two important axes: (i) meeting baseline values of number of samples expected from the network to help build statistical records for machine parts and (ii) ability to maintain interactivity with the network throughout its lifetime. We broadly deal with data traffic (periodic samples, real time streams and mission critical data), and control traffic as interactive queries. We continue using the 8×5 grid topology, but with an inter-node spacing of 5 ft. along both axes.

7.3.1. Data delivery

The first aspect we investigate is the ability to provide periodic samples reliably over the network. In fact, the core logic of PdM revolves around the ability to collect and maintain samples from moving parts of the machine at

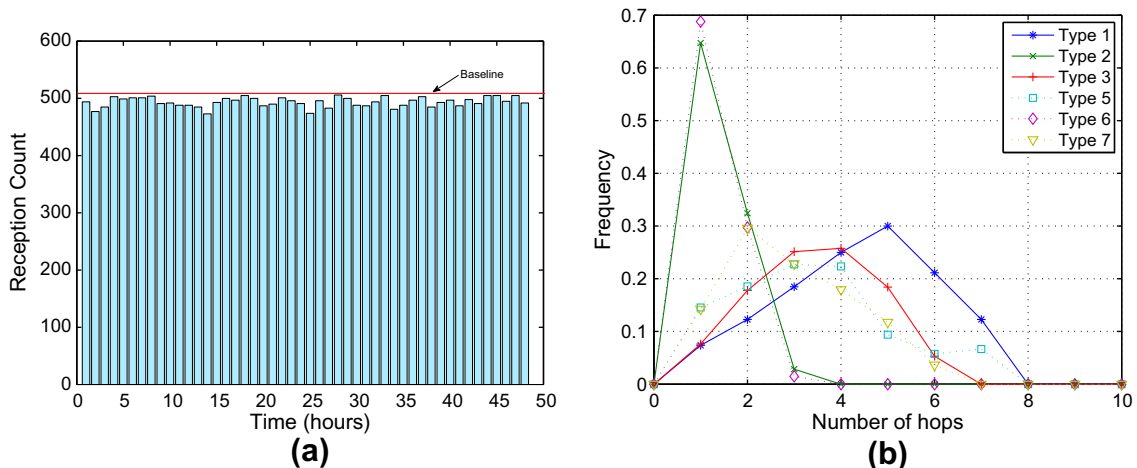


Fig. 10. (a) Volume of traffic Type 1 received every hour for a 2 day observation from 40 MicaZ nodes and (b) average hop count of various traffic types, normalized to show their relative frequencies.

regular intervals. Along with other traffic types in the workload, the nodes generated 13 samples of Type 1 traffic each per hour. With one node programmed as the base station, this leaves us with $39 \times 12 (=507)$ samples to be expected per hour from the network. We observe the number of samples received in a window of 48 hours. The data collected is shown in Fig. 10a. We deliver high reception ratios expected by this traffic type.

7.3.2. Characterizing differential routing

The effectiveness of our framework to offer different paths is best captured in Fig. 10b. For each packet received, we monitored the number of hops taken to base station for a 24 h operation period. Real time packets (Types 2 and 6) take very short hops to the base station. In fact, none of these packets take more than 4 hops. Reliable traffic (Types 1 and 5) takes numerous short hops, and frequents hop counts between 2 and 5. Mission critical traffic frequents hop counts in between reliable and real time traffic.

8. Conclusions

Typical deployments would consist of multiple concurrent applications, all of whose success leads to the fulfillment of a deployment's objective. With every application placing its own subjective communication demand on the framework, there is an urgent need to both expose these requirements to the communication framework, and dynamically customize behavior for every type of application. We have presented a simple scheme of using just three intent bits to completely describe communication patterns the stack, and we use this to drive a dynamic routing framework that customizes its routing behavior for every packet type in the system. We have proved its effectiveness in meeting the demands of a fairly complete deployment of industrial monitoring using PdM, where we analyzed behavior at scale for thousands of nodes, and implemented a prototype of a 40 node wireless testbed.

Diversity in application requirements for sensor networks led to an explosion of network protocols. Protocol developers focus performance for a particular traffic type, and likewise validate protocols for that type of traffic. Our framework allows for rapid protocol development, integration and validation in the face of realistic workloads. With a need to emphasize performance, developers further make assumptions about interfaces and functionalities that further limits synergy across research efforts. In our quest to build a configurable framework, we have regularized interface assumptions to distill core protocol features as individual components. This would ensure that the core components can evolve independently, and research efforts on any component can be seamlessly ported across deployments.

The role of in-network processing is currently limited in sensor networks. With the application requirements made visible to the stack, there is great potential to design application specific processing at every node. Our dynamic routing is just one example of using the requirements to switch routing behavior at the network layer. In general, there is excellent potential for designing medium access

protocols, scheduling protocols, congestion control algorithms and energy efficiency modules at various layers of the stack using the preamble bits. We conclude this paper with the hope that this work opens new directions in dynamic and need based protocol development.

References

- [1] T.E. Cheng, R. Fonseca, S. Kim, D. Moon, A. Tavakoli, D. Culler, S. Shenker, I. Stoica, A modular network layer for sensorsets, in: Proceedings of 7th Symposium on Operating Systems Design and Implementation (OSDI), November 2006.
- [2] O. Chipara, Z. He, G. Xing, Q. Chen, X. Wang, C. Lu, J. Stankovic, T. Abdelzaher, Real-Time power-aware routing in sensor networks, in: Proceedings of IEEE International Workshop on Quality of Service (IWQoS), June 2006.
- [3] D. Culler, P. Dutta, C.T. Ee, R. Fonseca, J. Hui, P. Levis, J. Polastre, S. Shenker, I. Stoica, G. Tolle, J. Zhao, Towards a sensor network architecture: lowering the waistline, in: HotOS X, June 2005.
- [4] D.S.J. De Couto, D. Aguayo, B. Chambers, R. Morris, Performance of multihop wireless networks: shortest path is not enough, in: First Workshop on Hot topics in Networks (HotNets-I), October 2002.
- [5] D.S.J. De Couto, D. Aguayo, J. Bicket, R. Morris, A high-throughput path metric for multi-hop wireless routing, in: ACM Mobicom, September 2003.
- [6] R. Fonseca, S. Ratnasamy, J. Zhao, T.E. Cheng, D. Culler, S. Shenker, I. Stoica, Beacon-vector routing: scalable point-to-point routing in wireless sensor networks, in: Proceedings of Usenix NSDI, July 2005.
- [7] Y. He, C.S. Raghavendra, S. Berson, Robert Braden, An autonomic routing framework for sensor networks, Cluster Computing 9 (2) (2006).
- [8] T. He, J.A. Stankovic, C. Lu, T. Abdelzaher, SPEED: a stateless protocol for real-time communication in sensor networks, in: Proceedings of ICDCS'03, May 2003.
- [9] J. Heidemann, F. Silva, Y. Yu, D. Estrin, P. Haldar, Diffusion Filters as a Flexible Architecture for Event Notification in Wireless Sensor Networks, Technical Report ISI-TR-556, USC/Information Sciences Institute, April, 2002.
- [10] N.C. Hutchison, L.L. Peterson, The X-Kernel: an architecture for implementing network protocols, IEEE Transaction on Software Engineering 17 (1) (1991).
- [11] L. Krishnamurthy, R. Adler, P. Buonadonna, J. Chhabra, M. Flanigan, N. Kushalnagar, L. Nachman, M. Yarvis, Design and deployment of industrial sensor networks: experiences from a semiconductor plant and the North Sea, in: ACM Sensys, November 2005.
- [12] P. Levis, D. Culler, Mate: a tiny virtual machine for sensor networks, in: Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), October 2002.
- [13] J.C. Mogul, K.K. Ramakrishnan, Eliminating receive livelock in an interrupt-driven kernel, ACM Transaction on Computer Systems 15 (3) (1997) 217–252.
- [14] R. Morris, E. Kohler, J. Jannotti, M.F. Kaashoek, The click modular router, ACM SIGOPS Operating Systems Review 33 (5) (1999) 217–231.
- [15] S.W. O'Malley, L.L. Peterson, A dynamic network architecture, ACM Transactions on Computer Systems (TOCS) 10 (2) (1992).
- [16] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, I. Stoica, A unifying link abstraction for wireless sensor networks, in: ACM Sensys, November 2005.
- [17] D. Sharma, V. Zadorozhny, P. Chrysanthis, Timely data delivery in sensor networks using whirlpool, in: Proc. 2nd International Workshop on Data management for Sensor Networks, August 2005.
- [18] F. Stann, J. Heidemann, RMST: reliable data transport in sensor networks, in: First IEEE Intl. Workshop on Sensor Network Protocols and Applications (SNPA), May 2003.
- [19] M. Venkataraman, M. Chatterjee, K. Kwiat, Traffic based dynamic routing for wireless sensor networks, in: Proceedings of IEEE WCNC, Budapest, Hungary, April 2009.
- [20] M. Venkataraman, K. Muralidharan, P. Gupta, Designing new architectures and protocols for wireless sensor networks: a perspective, in: IEEE Seccon, September 2005.
- [21] M. Venkataraman, P. Gupta, Stack aware architectures for mobile ad hoc networks, IETF Internet Draft, May 2004, (draft-venkataraman-stack-00.txt).
- [22] C.Y. Wan, S.B. Eisenman, A.T. Campbell, CODA: congestion detection and avoidance in sensor networks, in: ACM Sensys, 2003.

- [23] A. Woo, T. Tong, D. Culler, Taming the underlying challenges of reliable multihop routing in sensor networks, in: ACM Sensys, 2003.
- [24] C.Y. Wan, A.T. Campbell, L. Krishnamurthy, Pump-slowly, fetch-quickly (PSFQ): a reliable transport protocol for sensor networks, IEEE Journal on Selected Areas in Communication (JSAC) 23 (4) (2005) 862–872.
- [25] Y. Yu, L. Rittle, J. LeBrun, V. Bhandari, MELETE: supporting concurrent applications in wireless sensor networks, in: ACM Sensys, November 2006.
- [26] J. Zhao, R. Govindan, Understanding packet delivery performance in dense wireless sensor networks, in: ACM Sensys, November 2003.
- [27] MicaZ Motes Specification. <www.xbow.com/products/product_pdf_files/wireless_pdf/6020-0060-01_a_micaz.pdf>.



Mukundan Venkataraman is a Ph.D. candidate at the Department of Electrical Engineering and Computer Science (EECS) at the University of Central Florida (UCF). Prior to UCF, he was a member of Research Staff at the Pervasive Computing team at SET-Labs, Infosys Technologies Ltd., Bangalore, preceded by an internship at the SERC, Indian Institute of Science, Bangalore.

His findings have won him the Best Student Paper Award in Asia-Pacific (R10) by IEEE in 2003 at Penang, Malaysia; Best Student Paper awards consecutively in 2002 and 2003 by IEEE Bangalore (B.R.V. Varadan Paper Award) and an invited paper at ACM/IEEE CNDS 2004 in San Diego, CA. He is also the recipient of Presidential Doctoral Fellowship from UCF for pursuing graduate studies; as well as Outstanding Contributions Award and Best Outgoing Student Award from his undergraduate Institute. He received a B.E. in Computer Science from VTU in June 2003.



Mainak Chatterjee is an Associate Professor in the School of Electrical Engineering and Computer Science at the University of Central Florida. He received his Ph.D. from the department of Computer Science and Engineering at the University of Texas at Arlington in May 2002. He received his M.E. degree in Electrical Communication Engineering from the Indian Institute of Science in 1998. Prior to that, he did his B.Sc. in Physics from Presidency College, University of Calcutta. His research interest is in the broad area of computer communications and wireless networking and is associated with the Networking and Mobile Computing Lab (NetMoC) at UCF. He is a recipient of the AFOSR Young Investigator Program (YIP) award.



Kevin A. Kwiat has been a civilian employee with the U.S. Air Force Research Laboratory in Rome, New York for over 27 years. Currently he is working in the Cyber Science Branch. He received the B.S. in Computer Science and the B.A. in Mathematics from Utica College of Syracuse University, and the M.S. in Computer Engineering and the Ph.D. in Computer Engineering from Syracuse University. He received the following Air Force Research Laboratory Information Directorate Awards: Basic Research Award, Best Paper Award, and

Technology Team Award. He holds 4 patents. In addition to his duties with the Air Force, he is an adjunct professor of Computer Science at the State University of New York at Utica/Rome, an adjunct instructor of Computer Engineering at Syracuse University, and a Research Associate Professor with the University at Buffalo. He completed assignments as an adjunct professor at Utica College of Syracuse University, a lecturer at Hamilton College, a visiting scientist at Cornell University, and as a visiting researcher while on an Air Force Office of Scientific Research “Window on Europe” at the University of Edinburgh. His main research interest is dependable computer design.