Rethinking Multi-Channel Protocols in Wireless Sensor Networks

Chieh-Jan Mike Liang
Department of Computer Science
Johns Hopkins University
cliang4@cs.jhu.edu

Andreas Terzis
Department of Computer Science
Johns Hopkins University
terzis@cs.jhu.edu

Abstract

The availability of multiple frequency channels on modern radios has provided a way to improve networking performance. Nevertheless, current multi-channel protocols lack the architectural consistency and flexibility to support a diverse set of applications. In this paper we argue that it is necessary to integrate channel switching to the emerging wireless sensor network architecture and propose a way to decompose the problem into two reusable components: the channel allocation component that is integrated with network layer protocols and a shared channel synchronization component at the MAC layer. Furthermore, we outline how existing multi-channel protocols can be re-factored to comply with the proposed architecture and present *ViR*, an initial implementation of the channel synchronization component. Finally, using realistic applications synthesized from existing protocols, we show how ViR reduces conflicts among protocols and reduces packet losses.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Routing protocols*; D.4.7 [**Operating System**]: Organization and Design—*Real-time systems and embedded systems*

General Terms

Design, Experimentation, Standardization

Keywords

Sensor Networks, Protocol Architecture, Network Abstraction, Radio Virtualization

1 Introduction

Modern radios used in wireless sensor networks, such as those that implement the IEEE 802.15.4 PHY standard [5], can operate in multiple frequency bands. Previous work has shown that this capability can be used to increase the throughput of WSNs [16], reduce intra-network interference [9, 18], and reduce the impact of external interference to WSNs [11]. Despite these benefits, the mechanisms proposed thus far are implemented at different layers of the protocol stack, are incompatible, make assumptions about the applications' characteristics, and do not support multiple applications at the same time.

In this paper we argue that it is time to rethink how multi-channel protocols are engineered and reach a consensus about how they fit in the emerging WSN architecture [2, 6, 10]. Such an agreement will promote interoperability and code reuse and is likely to simplify the development of applications that will benefit from radio channel diversity. The additional challenge is to do this integration with minimal disruption to the existing protocol stack and application archetypes.

We posit that multi-channel protocols can be factored into two different components: a *Channel Allocation (CA)* component responsible for allocating one or more frequency channels to upper layers and a *Channel Synchronization (CS)* component that determines the time interval that a node should switch its radio to a certain channel. Network-level protocols can include their own CA components, allowing them to express specific ways of using frequency channels. For example, real-time network protocols can reduce data delivery latency due to channel switching by switching all nodes over an end-to-end network path to the same channel. On the other hand, a single CS component is implemented at the MAC layer that exposes the same MAC-layer interface and augments it with a minimal interface for using multiple frequency channels.

We describe a straw man implementation of the proposed components and outline how existing multi-channel protocols can be re-factored along these lines. Finally, we show how the proposed architecture can be used to virtualize the radio among multiple network layer protocols that run concurrently on the same mote.

2 Background

2.1 Benefits of Using Multiple Channels

We group multi-channel protocols into three categories based on their primary purpose for using multiple frequency channels.

Improve network throughput. Wu et al. proposed the TMCP multi-channel tree collection protocol and observed

HotEmNets 2010, June 28-29, 2010, Killarney, Ireland

that the network throughput doubled as the number of available channels increased from two to eight [16]. Zhang et al. proposed the TMMAC MAC protocol and showed that it achieved seven times the throughput of the standard 802.11 DCF protocol in a simulated network with 40 concurrent flows when six channels were available [17].

Minimize intra-network interference. Since the radio is a shared medium, concurrent transmissions in the same broadcast domain result in collisions and possibly packet losses. Wu et al. presented empirical results suggesting that 802.15.4 radios have a maximum of eight orthogonal channels[1] [16]. Zhou et al. [18] and Liang et al. [9] used channel diversity to reduce the number of conflicting transmissions. The latter work also observed that channel diversity promotes spatial reuse and reduces the latency of network-wide dissemination.

Avoid external interference. Considering that multiple radio standards operate in the same unlicensed frequency bands (e.g., 802.11, 802.15.4, and 802.15.1 all operate in the 2.4 GHz band), channel diversity is one method to mitigate external interference among collocated networks. The WirelessHART standard offers an example of this approach [11]. It employs a TDMA protocol, in which nodes switch their radio frequency at the start of each time slot. The frequency selection is based on the current time slot index and a secret channel offset shared by the two communicating nodes.

2.2 Limitations of Current Multi-Channel Protocols

While the discussion in Section 2.1 suggests that channel diversity can improve performance, we argue that current approaches lack the flexibility and the generality to support diverse application requirements.

Lack of Architectural Consistency. Based on the experience from developing and deploying WSN applications, an architecture for wireless sensor networks is starting to emerge [2, 6, 10]. Nevertheless, there is no consensus as to where channel switching belongs in the architecture and how it interfaces with the rest of the protocol stack. Existing multi-channel protocols are mostly implemented at the MAC level ([7, 17, 18]) or integrated to network layer protocols ([8, 9, 13, 16]). In turn, this lack of agreement impedes code reuse and protocol interoperability.

Restrictive Application Assumptions. Existing multi-channel protocols generally assume simplistic application behaviors —data collection with constant traffic rate being the most common one. For example, Le et al. proposes a multi-channel MAC protocol that is optimized for collection and aggregation traffic patterns [7]. However, supporting an expanding application landscape requires a multi-channel framework that can support multiple traffic patterns and QoS levels.

Monolithic Applications. After a decade of active development, open-source implementations of many WSN protocols are publicly available. This availability has enabled the development of applications that compose multiple network protocols to implement their logic. For example, one could combine a data collection protocol with a dissemination protocol to implement an environmental monitoring application with dynamic retasking [12, 14]. However, as Choi et al. pointed out, applications running multiple protocols can experience inter-protocol conflicts [1]. For example, two protocols running on the same node can independently decide to switch to two different channels, leading to packet losses for one of the protocols. Instead, multi-channel protocols should support multiple *concurrent* upper-level protocols and automatically resolve such conflicts (e.g., by switching between the two channels).

[1]The IEEE 802.15.4 standard specifies 16 non-overlapping channels in the 2.4 GHz band.

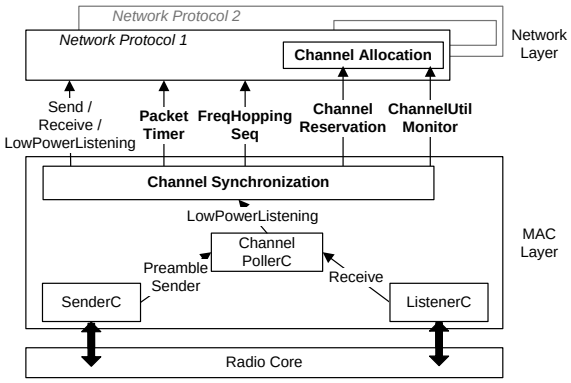

Figure 1. Proposed decomposition and interfaces of multi-channel protocols.

3 A Multi-Channel Protocol Framework

Section 2 argues that we need a consistent way to integrate frequency diversity to the WSN architecture. Doing so requires addressing three challenges. First, we need to identify and group tightly coupled functions that multi-channel protocols require into reusable components. Second, we should structure and position these components in ways that leverage the functionalities of existing architecture components. Finally, define a minimum set of interfaces that can be used to implement different network protocols and end-to-end applications.

3.1 Architecture Overview

We propose to decompose multi-channel protocols into a **channel allocation (CA)** component and **a channel synchronization (CS)** component. The goal of the CA component is to assign one or more channels to the network's nodes in a way that optimizes application-specific metrics such as network throughput. The CS component controls when radios should switch to each of the CA-selected channels so that nodes can communicate with each other. The CS component should base these scheduling decisions on data delivery metrics such as latency and energy usage.

Contrary to existing work that implements multi-channel protocols at either the network or the MAC layer, we propose a cross-layer approach that positions the CA component at the network layer and the CS component at the MAC layer.

We allow each network protocol to implement its own CA component for two reasons. First, since different network

layer protocol have different sets of requirements, a common CA component would not be suitable for all use cases. Second, because the CA component is part of the protocol it has access to the protocol's node state and information exchanged among protocol peers.

On the other hand, there is a single CS component per MAC protocol that interacts with all CA components at the network layer. This design has two advantages. As the single point of entry for all radio control and I/O requests, the CS component has the visibility on radio requests to resolve possible inter-protocol conflicts. In addition, the CS component can leverage the MAC layer functionalities to deliver messages to next-hop neighbors. Such functionalities include neighborhood state maintenance and link-level acknowledgments.

The decomposition of multi-channel protocols to CA and CS components is similar to the separation of routing and forwarding planes in the Internet; the CA component and the routing plane make network-wide decisions, while the CS component, similar to the forwarding plane, implement the mechanisms necessary to forward packets to the next hop.

Figure 1 illustrates the position of the proposed CA and CS components in the WSN architecture. The section that follows describes the interfaces these components expose and use.

3.2 Component Interfaces

The MAC layer provides three basic services to upper layers: transmit a packet, receive a packet, and turn the radio on and off. The CS component minimally extends this narrow interface to allow upper layers to leverage the benefits of multiple radio channels. At the same time, the existence of the CS component is transparent to legacy protocols that are agnostic to the availability of multiple radio frequencies.

We divide the new CS interfaces to those that the CA component uses (§3.2.1) and those that network layer protocols can use to improve their performance (§3.2.2). We note that while we use nesC to describe TinyOS-compliant interfaces, they are not specific to TinyOS.

3.2.1 Interfaces for the CA Component

Channel Reservation. The `ChannelReservation` interface offers two services. First, CA components can issue the `numChannels()` command to retrieve the number of available radio channels. Second, the `reserveChannel()` command allows a CA component to reserve a channel on the local node. Since the CS component intercepts all radio control requests, if a CA component attempts to switch to a previously reserved channel, the request will fail and return the appropriate error code.

```
interface ChannelReservation {
  command uint8_t numChannels();
  command error_t reserveChannel(uint8_t channel);
}
```

Channel Utilization. The `ChannelUtilMonitor` interface reports the utilization of a channel. Although the interface leaves the definition of utilization open, common choices include the number of network-layer protocols that intend to use a particular channel and the number of packets that have been sent and received on a channel. The interface mandates the `getUtilization()` command to return an integer between 0 and 255, corresponding to the lowest and the highest utilization respectively.

```
interface ChannelUtilMonitor {
  command uint8_t getUtilization(uint8_t channel);
}
```

3.2.2 Interfaces for Network Layer Protocols

Frequency Hopping Sequence. The CS component switches the radio channel according to an internal schedule to handle radio transmit/receive requests across different channels. The `FreqHoppingSeq` interface allows protocols at the network layer to query for the next time that the CS component switches the radio to a particular channel. This information can be useful for setting protocol timeout values, especially since multiplexing a radio essentially reduces the effective bandwidth for each protocol.

```
interface FreqHoppingSeq {
  command uint32_t nextTime(uint8_t channel);
}
```

Timers. Existing multi-channel mechanisms generally follow an eventual delivery service model due to the delay that channel synchronization introduces. Moreover, they do not provide upper layers the ability to express the urgency of their packet transmission requests. For example, a time synchronization protocol could indicate that its timing beacons need urgent transmission.

To meet these requirements, the CS component introduces the `PacketTimers` interface with commands that set per-packet timing attributes. First, we borrow the urgent bit concept from the SP architecture [10]. The urgent bit notifies the CS component that a radio request should have a higher priority and be processed before others. Second, we allow upper layers to set service duration timers for their packet transmissions. When one of these timers fires, the MAC layer should discard the corresponding transmission request and signal the proper return error code to the network layer.

```
interface PacketTimers {
  command void setUrgentBit(message_t* pkt);
  command void clrUrgentBit(message_t* pkt);
  command void setServiceDuration(message_t* pkt,
                                  uint32_t timeout);
  command uint32_t getServiceDuration(message_t* pkt);
}
```

3.3 Protocol Composition Example

Next, we demonstrate how multi-channel protocols can be developed using the interfaces proposed in the previous sections. We do so through Typhoon, a network-wide dissemination protocol that uses a dedicated control channel to initiate and negotiate the channels used for subsequent data transfers [9].

During bootstrapping, the CA component uses the `ChannelReservation` interface to reserve the dedicated control channel and to query the total number of available radio channels. Typhoon nodes then advertise a summary of

objects that they have. These advertisements allow nodes to detect any stale objects within their local neighborhood and initiate requests to retrieve new version of the objects.

To retrieve an object from neighbor x, node y initiates the handshake by sending a request message over the common control channel. Node x then queries its CA component to get a new channel for the data transfer. The CA component on node x can select a random channel, or a relatively free channel by querying the `ChannelUtilMonitor` interface for each channel's utilization. Node x finishes the handshake by responding to y with the new channel number. Finally, both nodes call the MAC layer API (e.g., the `CC2420Config` interface in TinyOS) to switch to the new channel for the data transfer. The CS module intercepts these requests as it exposes this interface (see Fig.1). After the data transfer completes, x and y return to the control channel.

4 Prototype Implementation

We now describe *ViR*, a straw man implementation of the channel synchronization component. While other implementations are possible, we use ViR to show how the proposed architecture can support multiple concurrent network layer protocols with different radio frequency usage patterns.

4.1 Basic Protocol

Being the channel synchronization component, ViR sits above all MAC layer components and intercepts I/O and radio control requests (e.g., requests to switch channels) from the network layer. The ViR uses this information to multiplex the physical radio across the different network layer protocols.

ViR divides time into equal-length slots, with nodes occupying one of the *active* channels during each time slot. A channel is active if at least one network layer protocol has previously requested to switch to that channel. We note that while all nodes use the same slot duration, they do not have to synchronize their schedules.

When a node has no pending outgoing packets, it cycles through all active channels in ascending order. Incoming packets are immediately delivered to the intended protocol. Nodes broadcast a *CH_SCHEDULE* packet at the beginning of each slot to help neighbors learn their channel schedule. *CH_SCHEDULE* packets carry timing offsets that indicate the next time a node is scheduled to be on each of the active channels.

One change is necessary to this basic scheme to handle outgoing packets. Specifically, a node can deviate from the predefined channel schedule in order to rendezvous and transmit packets to a neighbor. The probability of this deviation depends on local node's confidence that the intended neighbor will be on the desired channel during the next time slot. For instance, the age of the cached *CH_SCHEDULE* packets from a neighbor can influence this confidence. However, if a node x frequently deviates from its predefined channel schedule, it becomes difficult for other nodes to rendezvous with node x. For this reason, nodes stay on their scheduled channel during the next slot with probability $p = 0.25$. Nodes switch to one of the N channels with pending outgoing packets with probability $(1-p)/N$.

	CTP delivery rate (%)	CTP avg. latency (ms)
CTP only	99.7	56.8
Without ViR	52.3	642.8
With ViR	95.4	979.2

Table 1. When running both CTP and Typhoon on a 3×3 mesh, ViR improves the CTP delivery rate to 95%. The increase in CTP latency is due to ViR serving simultaneous Typhoon and CTP traffic.

After deciding the channel for the next time slot, ViR switches the physical radio channel and then it repeatedly sends any pending packets intended for the current channel one by one. Similar to the sender-initiated packet delivery approach, ViR stops transmitting a packet once the intended receiver acknowledges its reception, or when the packet exceeds its lifetime (the `PacketTimers` interface in Section 3.2.2 can be used to assign packet transmission deadlines).

Optimizations are possible on top of this basic scheme. If a node has pending packets for several neighbors, it can attempt delivery using a round-robin schedule to avoid head of line blocking by a node that is late switching to the current channel. Furthermore, if a node knows when a neighbor will enter the channel (e.g., from *CH_SCHEDULE* messages) it can postpone the transmission of packet(s) intended to that neighbor.

4.2 Evaluation

The primary goal of the channel synchronization component is to deliver packets from multiple network layer protocols with different radio frequency usage patterns. In this section, we evaluate how well the ViR implementation of the CS component meets this requirement. Specifically, we explore a scenario in which CTP [3], the standard data collection protocol in TinyOS, operates concurrently with Typhoon, a multi-channel network dissemination protocol [9].

We implemented ViR in TinyOS 2.1 and used the TOSSIM simulator to measure the protocols' performance. The evaluation results were collected with the default TinyOS MAC without low-power listening (LPL). We modified TOSSIM to simulate a physical layer that supports multiple channels. We simulated a 3×3 node grid in which the top left node acts as a base station. Each node is able to communicate with its immediate grid neighbors. All CTP nodes generate one packet every second. In addition to being a CTP sink, the base station initiates a network-wide dissemination of a 50KB object using Typhoon.

Table 1 summarizes the experimental results. When both protocols are active, ViR can approximately double CTP's delivery rate. We used the experiment's traces to understand the cause of this performance deterioration. We found that Typhoon's channel-switching behavior segmented the network paths that CTP used to deliver packets to the root. Note that if it is the CTP sender that switches to a different channel, all the alternate parents that CTP previously maintains to improve data delivery will not help. ViR mitigates this problem by exchanging frequency hopping sequences among neighbors and scheduling transmissions when both the sender and the intended receiver are on the desired chan-

nel. Since multiplexing the radio effectively reduces the available bandwidth for each protocol, the end-to-end latency of CTP packets should increase when both protocols are active. The results from Table 1 validate this.

5 Discussion

So far we have described how the CS module can multiplex requests from one or more network layer protocols that want to use multiple frequencies. At the same time, the CS component can be used to virtualize the radio among all the network protocols that run on the same mote. Specifically, the CS component offers the following properties:

Traffic Isolation. Concurrent traffic flows can experience collisions leading to energy waste and performance degradation. Choi et al. [1] cited a case in which the bursty traffic from Deluge [4] changed MintRoute's perception on link qualities [15]. Using the mechanisms proposed in this paper, one can de-conflict traffic flows by placing them over orthogonal channels.

Process Isolation. The CS component can track protocol-specific radio states and ensure per-protocol consistency. These states include the frequency channel, node address, and radio power state. Furthermore, the CS component ensures that state conflicts among protocols will not cause failures.

Resource Management. The CS component manages the physical radio using its knowledge of protocol-specific radio states. For example, it can shut down the physical radio after all consumers have indicated no further intention in using the radio.

5.1 Support for Low-Power MACs

Duty-cycling a radio is desirable since it reduce idle listening and conserves energy. Integrating duty-cycling and channel switching effectively multiplexes the radio in two different dimensions: time and frequency. We outline a method to achieve this integration.

We leverage the MAC Layer Architecture (MLA), a component-based architecture for duty-cycling MAC layers [6]. Since there are similarities between the operations of the CS component and duty-cycling MACs, the CS component can reuse functionalities in the MLA. The MLA defines three high-level components: `ChannelPollerC`, `SenderC`, and `ListenerC`. `ChannelPollerC` maintains the radio power state by both following the duty-cycling schedule and sampling the radio channel for activity. `SenderC` and `ListenerC` are concerned with one-hop packet delivery by using link-level mechanisms such as preambles and retransmissions. As Figure 1 illustrates, we position the CS component above `ChannelPollerC` so that minimum changes are required to the MLA. In addition, the CS component can delegate one-hop packet delivery and radio power state control to the MLA's components.

However, two changes are necessary to the protocol described in Section 4.1, for the CS component to work above the MLA. Namely, the ViR slot size should be equal to one duty cycle period and the ViR slots should be aligned with the start of each duty cycle period. These requirements imply that each ViR slot occupies one channel and one duty cycle period. Since the CS component delegates the radio's on/off state to the duty-cycling MAC below, we add callback events to notify the CS component of the radio's state changes.

Acknowledgments

We thank the anonymous reviewers for their help in improving the paper. Mike Liang and Andreas Terzis are partially supported by NSF awards CNS-0627611 and CNS-0546648.

6 References

[1] J. I. Choi, M. A. Kazandjieva, M. Jain, and P. Levis. The Case for a Network Protocol Isolation Layer. In *SenSys*, 2009.

[2] C. T. Ee, R. Fonseca, S. Kim, D. Moon, A. Tavakoli, D. Culler, S. Shenker, and I. Stoica. A Modular Network Layer for Sensornets. In *OSDI*, 2006.

[3] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection Tree Protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 1–14, Nov 2009.

[4] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *SenSys*, 2004.

[5] IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks. Specific requirements – Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs). Available at `http://www.ieee802.org/15/pub/TG4.html`, May 2003.

[6] K. Klues, G. Hackmann, O. Chipara, and C. Lu. A Components-Based Architecture for Power-Efficient Media Access Control in Wireless Sensor Networks. In *SenSys*, 2007.

[7] H. Le, D. Henriksson, and T. Abdelzaher. A Practical Multi-Channel Medium Access Control Protocol for Wireless Sensor Networks. In *Proceedings of the Seventh International Conference on Information Processing in Sensor Networks (IPSN)*, St. Louis, Missouri, USA, Apr. 2008.

[8] C.-J. M. Liang, J. Liu, L. Luo, A. Terzis, and F. Zhao. Racnet: a high-fidelity data center sensing network. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (Sensys)*, pages 15–28, New York, NY, USA, 2009. ACM.

[9] C.-J. M. Liang, R. Musăloiu-E., and A. Terzis. Typhoon: A Reliable Data Dissemination Protocol for Wireless Sensor Networks. In *Proceedings of the Fifth European Conference on Wireless Sensor Networks (EWSN)*, pages 268–285, Jan. 2008.

[10] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, and I. Stoica. A Unifying Link Abstraction for Wireless Sensor Networks. In *SenSys*, 2005.

[11] J. Song, S. Han, A. K. Mok, D. Chen, M. Lucas, and M. Nixon. WirelessHART: Applying Wireless Technology in Real-Time Industrial Process Control. In *RTAS*, 2008.

[12] G. Tolle and D. Culler. Design of an Application-Cooperative Management System for Wireless Sensor Networks. In *EWSN*, 2005.

[13] X. Wang, X. Wang, X. Fu, G. Xing, and N. Jha. Flow-Based Real-Time Communication in Multi-Channel Wireless Sensor Networks. In *EWSN*, 2009.

[14] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and yield in a volcano monitoring sensor network. In *OSDI '06*, 2006.

[15] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *SenSys*, 2003.

[16] Y. Wu, J. A. Stankovic, T. He, J. Lu, and S. Lin. Realistic and Efficient Multi-Channel Communications in Wireless Sensor Networks. In *INFOCOM*, 2008.

[17] J. Zhang, G. Zhou, C. Huang, S. Son, and J. A. Stankovic. TMMAC: An Energy Efficient Multi-Channel MAC Protocol for Ad Hoc Networks. In *ICC*, 2007.

[18] G. Zhou, C. Huang, T. Yan, T. He, and J. A. Stankovic. MMSN: Multi-Frequency Media Access Control for Wireless Sensor Networks. In *INFOCOM*, 2006.