Papyrus: A Software Platform for Distributed Dynamic Spectrum Sharing Using SDRs

Lei Yang, Zengbin Zhang, Wei Hou†, Ben Y. Zhao, Haitao Zheng
Department of Computer Science, University of California, Santa Barbara
† Department of Electronic Engineering, Tsinghua University, Beijing, China
{leiyang,zengbin,ravenben,htzheng}@cs.ucsb.edu, weihou2008@gmail.com

ABSTRACT

Proliferation and innovation of wireless technologies require significant amounts of radio spectrum. Recent policy reforms by the FCC are paving the way by freeing up spectrum for a new generation of frequency-agile wireless devices based on software defined radios (SDRs). But despite recent advances in SDR hardware, research on SDR MAC protocols or applications requires an experimental platform for managing physical access. We introduce Papyrus, a software platform for wireless researchers to develop and experiment dynamic spectrum systems using currently available SDR hardware. Papyrus provides two fundamental building blocks at the physical layer: flexible non-contiguous frequency access and simple and robust frequency detection. Papyrus allows researchers to deploy and experiment new MAC protocols and applications on USRP GNU Radio, and can also be ported to other SDR platforms. We demonstrate the use of Papyrus using Jello, a distributed MAC overlay for high-bandwidth media streaming applications and Ganache, a SDR layer for adaptable guardband configuration. Full implementations of Papyrus and Jello are publicly available.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Design, Experimentation

Keywords

Dynamic Spectrum Access, Cognitive Radio, Testbed

1. INTRODUCTION

Wireless technologies are expanding into all aspects of our daily lives. For example, today's digital home uses a wide range of wireless devices to replace messy and cumbersome audio, video, telephone and data cables. Future wireless technologies are expected to deliver high-quality streaming media between rooms (see Figure 1). While mom or dad streams a cooking video from the office PC to the kitchen counter, kids in the den can watch an HD movie streamed from the Digital Video Recorder (DVR) in the living room.

These networks of the future require significant amounts of radio spectrum. Changing policies at the FCC are paving the way by freeing up spectrum for a new generation of spectrum-worry-free wireless devices based on software defined radios (SDRs). SDR devices can be programmed to intelligently sense locally available spectrum and coordinate with other communication endpoints to occupy specific frequency bands for reliable, high throughput communication. By operating on different frequency ranges, multiple wireless flows can coexist without mutual interference in high density environments, similar to prior OFDMA approaches. Towards this goal, recent research projects have produced a number of viable SDR platforms for research experimentation and deployment, such as AirBlue [5], KNOWS [2], SORA [10], USRP GNU Radio [4], and WARP [13].

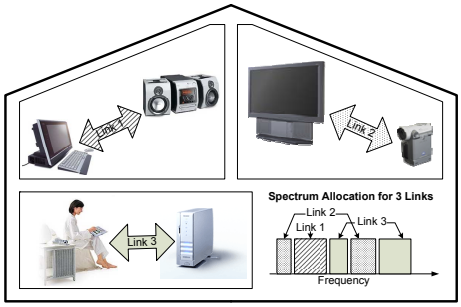

Figure 1: Home networking devices share radio frequency to provide high-quality media streaming.

Despite the availability of these experimental platforms, several significant challenges must be addressed before we can begin research on developing key media access control (MAC) protocols necessary for network applications. More specifically, two issues remain at the physical layer. First, SDR devices must be able to quickly sense specific spectrum ranges that are unused and available. Second, the bandwidth requirements of future network applications will be highly heterogeneous and highly dynamic over time. The result is that available spectrum is likely to be in fragments of all sizes. To support bandwidth hungry applications, SDR devices must be able to combine multiple spectrum fragments into single channel for network transmissions. On top of these two issues, SDR devices must also be able to operate in a decentralized manner without central control.

In this article, we describe our work on *Papyrus*, an experimental software platform for building and evaluating MAC protocols and applications on dynamic spectrum access systems. Papyrus provides a fully programmable decentralized OFDMA system suitable for deploying MAC protocols and applications. By providing a higher level interface to the

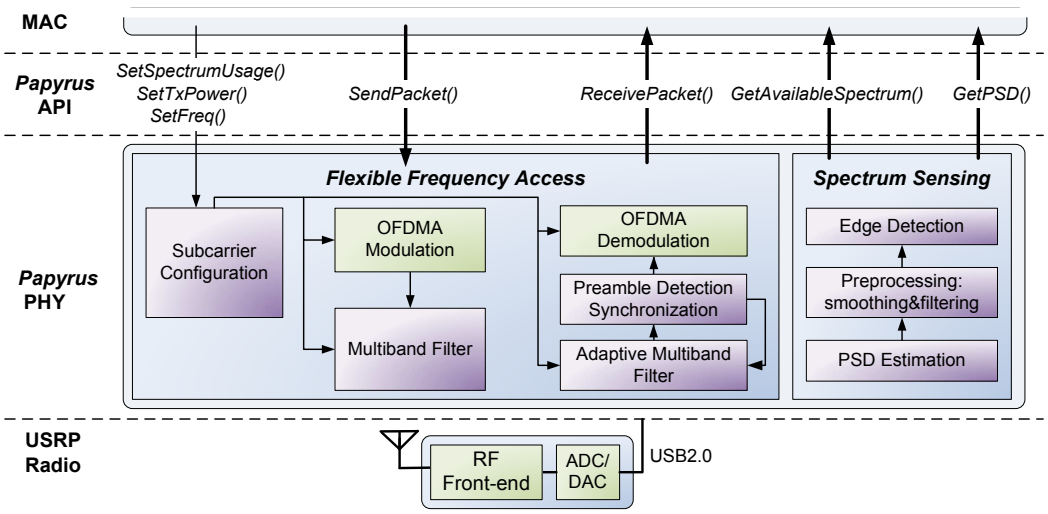

Figure 2: Papyrus software architecture and its interfaces with the MAC layer. Papyrus's contributions are marked by the purple colored blocks.

MAC-layer while abstracting away key challenges of spectrum sensing and access, Papyrus simplifies implementation and experimentation at the MAC layer.

To illustrate Papyrus usage, we describe the implementation of two example systems. The first is Jello, a distributed MAC overlay for high-bandwidth media streaming applications [14]. Jello devices support high-quality delay-sensitive media sessions by accessing and sharing wireless medium in the frequency domain. These devices utilize Papyrus's physical layer components to sense, identify, and access unused spectrum, allowing multiple flows to work in parallel on isolated frequencies. Jello devices also self-defragment spectrum on-the-fly, and scavenge multiple frequency fragments for use by individual, high-speed transmissions. Our second example is Ganache [15], a system that intelligently sets and adapts "guardbands" in the frequency domain to insulate frequency-adjacent transmissions from mutual cross-band interference. Ganache leverages Papyrus to measure frequency signal strength, configure link spectrum usage, and make adjustments to local guardband settings.

Our full implementation of Papyrus is available for download at `http://link.cs.ucsb.edu/papyrus`. While our instance of Papyrus runs on the USRP GNU Radio platform, Papyrus can be ported into all other current SDR platforms, including SORA and AirBlue.

2. THE PAPYRUS PLATFORM

Recent advances in hardware platforms [5, 10, 12, 13] are building the groundwork for experimental research on SDR protocols and applications. To take this significant step forward, we propose *Papyrus*, a flexible, software platform that manages the complexities of dynamic spectrum access at the physical layer, exporting a clean and manageable abstraction to the MAC layer.

2.1 Overview

Figure 2 illustrates Papyrus's software architecture and its interfaces to the MAC layer. Papyrus provides to the MAC layer two fundamental physical layer functions of dynamic spectrum access:

- **Flexible spectrum access** which transmits and receives packets from any combination of frequency segments as specified by the MAC layer;
- **Spectrum sensing** which senses and identifies locally usable spectrum segments at frequency ranges defined by the MAC layer.

Both components operate in a *fully decentralized* manner without relying on any central control or dedicated control radio. Both components are also modular and thus easy to reconfigure or modify.

The Papyrus API. Using these two components, we define the interface between Papyrus and the MAC layer by the Papyrus API functions, listed in Table 1 as well as Figure 2. The API implements the physical layer functionalities of dynamic spectrum access, allowing the MAC layer to obtain and investigate the energy/interference profile on each frequency subcarrier, discover locally usable spectrum, send and receive both control and data packets on desired frequency ranges, and finally configure transmit power profile and central carrier frequency on each transmission. For example, to send a packet on subcarriers specified by `carrierUsage`, the MAC layer calls two functions sequentially:

```
SetSpectrumUsage(carrierUsage)
SendPacket(payload).
```

In the following, we briefly describe the design of these two components as well as their USRP GNU radio implementation. For a detailed explanation, we refer the readers to [14]. In Section 3, we will show that using these two components, researchers can build MAC protocols that dynamically configure spectrum usage in both time and frequency.

2.2 Flexible Frequency Access

We start from describing how Papyrus devices access and share spectrum in the frequency domain. The spectrum band is divided into a large set of frequency subcarriers. Each Papyrus sender can transmit on any subset of the subcarriers, either contiguously or non-contiguously aligned in

API function	Description
int * GetAvailableSpectrum( )	It performs edge-detection based sensing, and returns a binary array that records each subcarrier's availability (idle or busy).
void SetSpectrumUsage(int * carrierUsage)	A function used by the MAC layer to configure the spectrum (subcarrier) usage of Papyrus PHY. The input parameter **carrierUsage** is a binary array that defines the set of subcarriers to use.
void SendPacket(int * payload)	Sends a packet using Papyrus PHY. The input parameter **payload** holds the packet content.
void ReceivePacket(int pktStatus, int * payload)	The packet reception callback function for Papyrus PHY. Upon detecting a packet (via preamble), it returns **pktStatus**, a boolean value that indicates whether the packet is correctly received, and **payload** the content of the packet if it is received correctly.
float * GetPSD( )	Gets the current power spectrum density map. It returns an array of floats that stores the observed PSD at each subcarrier.
void SetTxPower(float * powerProfile)	Sets the transmission power of Papyrus PHY. The input parameter **powerProfile** is a float array that defines the transmission power at each subcarrier.
void SetFreq(float freq)	Sets the central carrier frequency **freq** of Papyrus PHY.

Table 1: The API functions that Papyrus provides to the MAC layer.

frequency. Each receiver can listen to the entire set of subcarriers at once, and decode its desired signal. As shown by the top halves of Figure 3, multiple transmissions can occur simultaneously on isolated frequency subcarriers without mutual interference, while adapting their frequency usage to time-varying traffic demands.

Papyrus offers this level of flexible frequency access by implementing decentralized orthogonal frequency division multiple access (OFDMA). OFDMA is a multi-user version of the orthogonal frequency-division multiplexing, a digital modulation scheme. By dividing the frequency band into a large set of subcarriers and letting transmissions operate on isolated subset of subcarriers, OFDMA allows several interfering links to transmit simultaneously. Because of its flexibility and efficiency, OFDMA has been widely used in centralized wireless networks such as WiMAX.

Implementing OFDMA on distributed networks, however, faces several significant challenges. Existing designs in centralized networks rely on global synchronization to maintain subcarrier orthogonality, so that transmissions on different subcarriers do not interfere with each other. In distributed networks, where global synchronization is infeasible, this subcarrier orthogonality no longer exists, and transmissions create harmful cross-band interference to each other.

The bottom half of Figure 3a shows a frequency-domain snapshot of two transmissions (link 1 and 2) in an OFDMA system, in terms of the power spectrum observed by link 2's receiver. In this example, link 1 occupies subcarriers 23-30 and link 2 occupies subcarriers 33–40. Both transmissions leak power to unoccupied subcarriers due to out-of-band emissions. When these transmissions are tightly synchronized, i.e., their symbols start and end at the same time and their central carriers are identical, their receivers' demodulation operation will remove unwanted signals and maintain the subcarrier orthogonality. But when operating in a decentralized network without tight synchronization, link 1 and 2's OFDM symbols are mis-aligned with each other in time when they arrive at link 2's receiver (see Figure 3b). In this case the receiver can no longer cancel the out-of-band emission and will suffer significant damage in its preamble detection and packet decoding [7].

Addressing Cross-Band Interference. Papyrus overcomes this problem using advanced signal processing techniques, essentially by reengineering receivers to "filter" out unwanted signals and restoring the desired transmission orthogonality. Specifically, Papyrus introduces two new modules on top of the conventional OFDMA design.

- *Adaptive Multi-band Receiver Filter* – Papyrus applies a filter at both the transmitter and receiver to remove signals from unwanted subcarriers. The filter is multi-band so that devices can access multiple non-contiguously aligned frequency subcarriers. Each Papyrus receiver also dynamically tunes the filter's carrier frequency and width to compensate the difference between transmitter and receiver in their central carrier frequency, an artifact known as the frequency offset. At initialization, the receiver starts from a loose filter in order to capture its desired signal. After each successful packet decoding, it estimates the frequency offset using the packet preamble and adjusts the filter's carrier frequency accordingly. It also shrinks the filter bandwidth to remove as much interference as possible. If the filter becomes too narrow and fails to detect any packet preamble, the receiver again expands the filter to capture more signals.
- *Sensing-assisted Guardband Protection* – The adaptive filter can effectively reduce cross-band interference, but cannot fully remove it. To suppress the impact of residual interference, Papyrus inserts frequency guardbands at link boundaries to protect them from interfering each other [7]. Instead of blindly inserting guardbands, Papyrus measures interference power levels using its sensing component (Section 2.3), and marks severely affected frequency subcarriers as busy. In addition, the MAC layer can use sensing reports to adjust guardband configuration. This technique, combined with the adaptive filtering, enables Papyrus devices to effectively control cross-band interference.

2.3 Sensing and Detecting Vacant Frequencies

When sharing spectrum with peers, wireless devices must sense the spectrum to quickly and reliably identify locally usable frequency. Based on the frequency range defined by the MAC layer, Papyrus configures its OFDMA-based transceiver to listen to the spectrum span in this range and produce a power spectrum density (PSD) map that measures the received energy level across the frequency band.

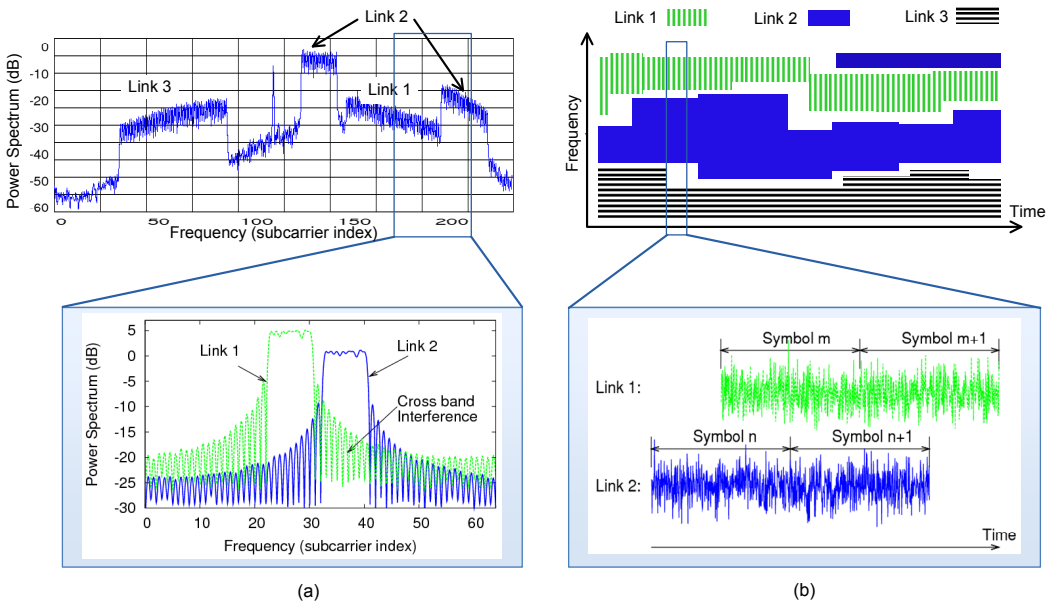

Figure 3: An example of Papyrus's flexible spectrum access using OFDMA. Three transmissions access and share radio spectrum in the frequency domain. Link 2 operates on two non-contiguous spectrum blocks to form a single transmission. The closer look at the frequency and time signals show the impact of unsynchronized OFDMA.

Papyrus then uses the PSD map to identify usable frequency by detecting "busy" subcarriers.

To detect busy subcarriers, conventional approaches apply a threshold-based energy detector over the PSD map. A subcarrier is busy when its PSD value exceeds the threshold and otherwise idle. This approach, however, is highly dependent on the choice of the detection threshold and finding a uniformly optimal value is unrealistic [8].

Sensing via Edge Detection. Papyrus instead exploits a unique property of radio transmissions in the frequency domain. Regardless of energy levels, all active transmissions display pairs of rising and falling edges in the PSD map (see Figure 4). By identifying these edges, Papyrus can reliably identify active transmissions and usable subcarriers.

To do so, Papyrus first smooths the PSD map by averaging it over several signal samples, filtering out most noise before trying to locate edges. Papyrus also applies energy detection with two extreme thresholds τ_{max} and τ_{min} on the smoothed PSD map to filter out subcarriers with very high and very low power – any subcarriers with power level higher than τ_{max} are considered busy, and any subcarriers with power level lower than τ_{min} are considered idle. Next, Papyrus applies the edge detection mechanism used in image processing [1] to detect rising and falling edges (see Figure 4). A frequency block with a rising edge to its left and a falling edge to its right is busy, and the rest are free. While this new method still requires a threshold to detect edges, our experimental results confirm that its sensitivity to both noise and threshold choice is much smaller than that of the energy detector [14].

Detecting External Users. The above spectrum sensing design focuses on detecting available spectrum among peering Papyrus devices. It can also detect external users and even primary users whose spectrum usage can be stably represented by the PSD map. For other forms of transmissions, such as wireless microphones with intermittent transmissions, Papyrus can apply other sensing mechanisms, e.g., feature detection based sensing, to obtain reliable results.

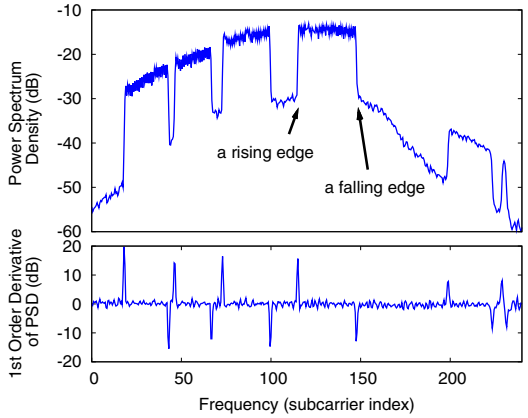

Figure 4: A sample PSD map and its first-order derivative, used to detect edges. Papyrus identifies occupied frequency blocks using edge detection. While the absolute signal strength varies significantly across the frequency, the rising/falling edges are easier to detect. Image from [14].

2.4 Prototyping Papyrus on GNU Radios

We have implemented a prototype of Papyrus on USRP GNU Radios, a widely available software reconfigurable radio that supports full reconfiguration across various protocol layers. Each Papyrus device consists of a USRP GNU radio and a laptop (or desktop) running Ubuntu Linux. A USB 2.0 interface connects the USRP radio with the laptop.

Following the GNU Radio software framework, we im-

plemented Papyrus's flexible frequency access component in C++ and the spectrum sensing component in Python. The architecture of our implementation is shown in Figure 2. The original USRP GNU Radio package only includes the basic OFDM functions but does not contain any decentralized OFDMA implementation. We add/modify the purple colored blocks to implement the proposed distributed OFDMA, allowing devices to flexibly access any combinations of subcarriers without central control. These include *preamble configuration and detection* which sends and detects packet preamble on a subset of subcarriers using a modified PN correlation approach [11], *adaptive multi-band filter* at both sender and receiver to filter out unwanted interference, and *edge detection based spectrum sensing* to identify usable spectrum subcarriers. From these components we then build the Papyrus API functions. We believe that these are the basic set of functions to implement dynamic spectrum access protocols and applications.

Our USRP implementation supports operations in both 50MHz - 2.9GHz and 4.9GHz - 5.85GHz bands by using the RF daughter-boards WBX or XCVR2450 from [12]. Running on a standard dual-core laptop, a Papyrus device can communicate over any 1MHz frequency band. The frequency band can be divided into 64, 128, 256 or 512 subcarriers, where each subcarrier can use either BPSK or QPSK modulation. There are two primary limitations to the bandwidth of the system, one is the USB 2.0 interface that connects the USRP board to the laptop; the other is the limited computation power of the laptop. We expect that these artificial limitations will play considerably smaller roles as our platform migrates to improved SDR hardware in the near future.

Finally, since Papyrus code is modular and freely available, researchers can easily implement additional physical layer functions or refine existing functions. For example, the current Papyrus implementation uses the same modulation scheme for all subcarriers, since they experience similar channel conditions in our narrow-band GNU Radio implementation. To exploit frequency diversity present in wideband systems, we can apply frequency-aware rate adaptation [6, 9] to configure the modulation and coding scheme on a per-subcarrier basis. This also requires per-subcarrier signal strength estimation [6].

3. USING PAPYRUS

To demonstrate the impact of Papyrus as an experimental platform, we have used it to implement two MAC protocols on dynamic spectrum access systems. *Jello* [14], is a distributed MAC overlay to support multiple concurrent media streaming applications, *e.g.* in digital homes. *Ganache* [15], is an intelligent guardband configuration system that dynamically sets and adapts frequency guardband to insulate links from cross-band interference. In this section, we briefly describe the architecture of both systems, and describe how they are implemented on top of Papyrus.

3.1 Jello: A MAC Overlay on Papyrus

Jello is a decentralized MAC overlay that enables multiple point-to-point media sessions (shown in Figure 1) to operate in parallel by sharing spectrum in the frequency domain, thus avoiding costly temporal contention and ensuring continuous spectrum access. These sessions also make use of the spectrum efficiently by adapting their frequency usage to time-varying traffic demands.

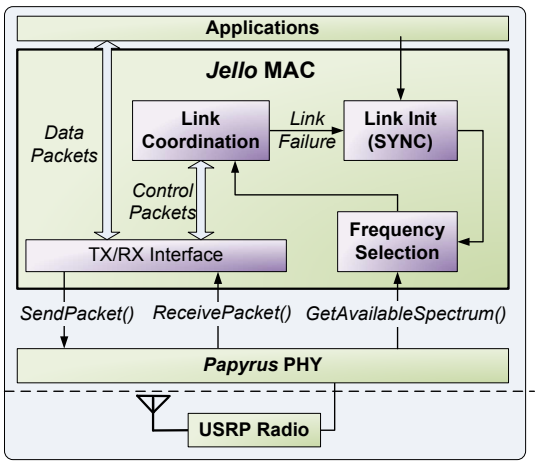

Figure 5: Jello and its interface with Papyrus.

As a MAC protocol, Jello essentially implements a distributed coordination framework that allows devices to configure and adapt their spectrum usage on the fly. It consists of three key components: 1) initialization, where end-devices coordinate their initial frequency usage to set up links; 2) selecting and adapting frequency usage, 3) maintaining link connection by coordinating frequency usage within each Jello sender and receiver pair. All three components operate without relying on any central control. We refer the readers to [14] for a detailed description of Jello.

Utilizing Papyrus's API functions, we have implemented Jello as a user-level Python program on GNU radios. It interfaces with Papyrus's C++ code using Swig (`http://www.swig.org`). Figure 5 illustrates the structure of Jello, and its interaction with the Papyrus API.

We describe Jello's MAC operations using pseudocode below. When a pair of Jello devices initiate a link, they first enter the SYNC state and communicate with each other on the SYNC frequency band. After establishing the session, they communicate with each other on the negotiated frequency subcarriers. Jello devices use the *best-fit* algorithm to determine their frequency usage. During the session, the devices may change their spectrum usage by repeating the spectrum sensing, frequency selection and link coordination steps. While configuring and adapting their spectrum usage, each pair of Jello devices exchange coordination messages on the current frequency to negotiate and synchronize their spectrum usage. In rare cases where coordination fails, both devices enter the SYNC state to restore communication.

```
%% Initialization
availSubcarriers = GetAvailableSpectrum()
SetSpectrumUsage(SYNC_FREQ)
SendPacket(availSubcarriers + coordinationMsg)

%% Selecting and adapting frequency usage
%% current frequency usage == carrierUsage
availSubcarriers = GetAvailableSpectrum()
newCarrierUsage = BestFit(availSubcarriers,demand)

%% Coordination between sender and receiver
SetSpectrumUsage(carrierUsage)
SendPacket(newCarrierUsage + coordinationMsg)
```

```
%% Sending packet on newly selected frequency
SetSpectrumUsage(newCarrierUsage)
SendPacket(dataPacket)
```

To match device frequency usage to traffic demand, Jello devices make frequent use of Papyrus's functions for reporting available spectrum and setting spectrum usage.

Initial Deployment. To validate our prototypes of Jello and Papyrus, we built a small ad-hoc indoor network of eight USRP1 GNU radios, creating four ad-hoc but simultaneous media sessions with time-varying traffic. This provides a proof-of-concept verification of our MAC and physical layer design and implementation. Our measurements confirm that Jello (and Papyrus) can provide reliable spectrum access for media applications and significantly improve spectrum usage efficiency. We refer the readers to [14] for more detailed experimental results.

Our initial deployment, however, has been limited by the hardware bandwidth and the unpredictable OS scheduling and processing delay between USRP and GNU radio software [3]. Our Jello devices currently only operate on a 500 KHz band at 2.38 GHz. We partition the spectrum into 256 subcarriers, each of size 1.953 KHz. We have to use relatively large timing parameters in the proposed link coordination and frequency adaptation, which limit the efficiency of both components. However, we expect the impact of these factors will decrease significantly as we port Jello and Papyrus to new SDR hardware such as SORA [10].

3.2 Ganache: Guardband Configuration

Ganache [15] is an intelligent guardband configuration system for dynamic spectrum networks. Because transmissions on one frequency band will spill energy into adjacent bands, guardbands must be placed at link frequency boundaries to insulate transmissions. Ganache applies both centralized planning and dynamic per-link tuning to protect links against cross-band interference. A Ganache server first estimates required guardband sizes from measurements of power levels over frequency-adjacent links. Then, it performs network-level frequency planning to allocate frequency usage to links, and configures an effective set of guardbands. During transmissions, individual Ganache links monitor physical distortion of their signals to detect residual cross-band interference, and adjust guardband locally to compensate.

Ganache is also implemented as a user-level Python program on the Papyrus API. The Ganache server schedules each transmitter i to send training packets, where all the receivers measure the channel response via the getPSD() API function. After receiving the frequency allocation, individual links configure their frequency usage for data communication. Each individual link also monitors physical signal distortion to detect residual cross-band interference, and locally adjusts its guardband usage (by reducing frequency usage) to suppress interference.

```
%% Signal measurements
FOR EACH transmitter i
    SendPacket(allSubcarriers)
    FOR EACH receiver j
       chanResponse(i,j) = GetPSD()

%% Centralized frequency planning
FOR EACH transmitter i
    SetSpectrumUsage(carrierUsage)
    SendPacket(data)

%% Local guardband adaptation at each link
IF Cross-band Interference Detected at Subcarrier k
    newCarrierUsage = {carrierUsage} - k
    SetSpectrumUsage(newCarrierUsage)
    SendPacket(data)
```

Note that for both Jello and Ganache, the large majority of code is dedicated to internal system functions. By providing higher level abstractions to query for and set spectrum usage, Papyrus allows MAC developers to greatly simplify their interactions with wireless spectrum.

4. CONCLUSION

In this article we describe Papyrus, a software platform that enables researchers to design, deploy and experiment dynamic spectrum access systems using currently available SDRs. Papyrus implements two fundamental building blocks of dynamic spectrum access: flexible non-contiguous frequency access and robust usable frequency detection. Papyrus devices operate in a fully decentralized manner without requiring a centralized controller or external control radios. We also describe Jello and Ganache, two specific MAC protocols for dynamic spectrum access systems, and their implementation via Papyrus.

We have made both Papyrus and Jello's GNU Radio implementation public to the research community, available at `http://link.cs.ucsb.edu/papyrus`. Because both Papyrus and Jello follow a modular design, researchers can modify individual modules and add additional modules to fit their research topics.

5. REFERENCES

[1] Canny, J. A computational approach to edge detection. In *IEEE Trans. on Pattern Analy. and Mach. Intell.* (1986).

[2] Chandra, R., et al. A case for adapting channel width in wireless networks. In *Proc. of SIGCOMM* (2008).

[3] Ge, F., et al. Software defined radio execution latency. In *Proc. of SDR Technical Conference* (2008).

[4] `http://gnuradio.org/trac/wiki`.

[5] Gummadi, R., et al. AirBlue: A system for cross-layer wireless protocol development and experimentation. In *MIT Report* (2008).

[6] Halperin, D., Hu, W., Sheth, A., and Wetherall, D. Predictable 802.11 packet delivery from wireless channel measurements. In *Proc. of SIGCOMM* (2010).

[7] Hou, W., et al. Understanding the impact of cross-band interference. In *Proc. of Coronet* (2009).

[8] Rahul, H., et al. Learning to share: narrowband-friendly wideband networks. In *Proc. of SIGCOMM* (2008).

[9] Rahul, H., et al. Frequency-aware rate adaptation and MAC protocols. In *Proc. of MobiCom* (2009).

[10] Tan, K., et al. SORA: High performance software radio using general purpose multi-core processors. In *NSDI* (2009).

[11] Timothy M., S., and Donald C., C. Robust frequency and timing synchronization for OFDM. In *IEEE Transactions on Communications* (1997).

[12] `http://www.ettus.com/`.

[13] `http://warp.rice.edu/`.

[14] Yang, L., et al. Supporting demanding wireless applications with frequency-agile radios. In *NSDI* (2010).

[15] Yang, L., Zhao, B. Y., and Zheng, H. The spaces between us: Setting and maintaining boundaries in wireless spectrum access. In *Proc. of MobiCom* (2010).