Real-Time Centralized Spectrum Monitoring: Feasibility, Architecture, and Latency

Michael Souryal, Mudumbai Ranganathan, John Mink, and Naceur El Ouni
Communications Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, Maryland, USA
Email: {souryal, mranga, john.mink, naceur.elouni}@nist.gov

***Abstract*—This paper describes the implementation and evaluation of a real-time, centralized spectrum monitoring and alert system. Such a system can be used to support emerging spectrum sharing solutions that use a centralized controller to mediate tiered access to spectrum. These controllers rely on real-time awareness of spectrum activity. In addition to describing the architecture and prototype implementation of this real-time monitoring system, we propose a test method to measure the latency of detecting a spectrum occupancy event. This latency is measured as the time from when the event (e.g., a signal transmission) begins to when an alert of that event is delivered to a subscribed client. We used this test method to measure the latency of two different sensor implementations in conjunction with our spectrum occupancy server and found the 95th percentile of latency to be under 80 ms in both cases, plus the network transmission delays of any wide area network involved.**

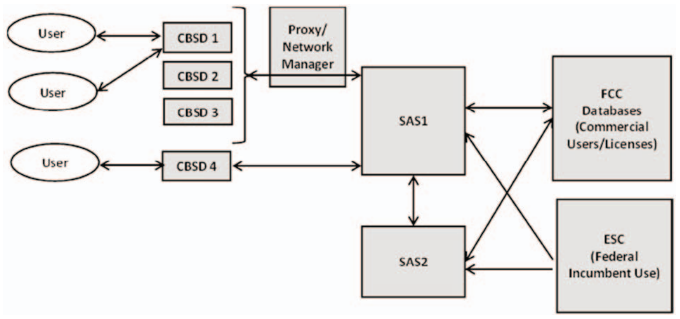
User
User
User
CBSD 1
CBSD 2
CBSD 3
CBSD 4
Proxy/ Network Manager
SAS1
SAS2
FCC Databases (Commercial Users/Licenses)
ESC (Federal Incumbent Use)

Fig. 1. FCC's illustrative end-to-end CBRS architecture [3]

I. Introduction

A critical component of any dynamic spectrum sharing solution is real-time awareness of spectrum usage. In emerging centrally-coordinated solutions, a centralized system is responsible for mediating tiered access to the spectrum such that higher priority users (e.g., an incumbent system) are given preference over lower priority users. Such a system, also referred to as a spectrum access system (SAS) [1] or a licensed shared access controller [2], may rely on sensors to detect changes in spectrum occupancy in real time and take appropriate actions. For example, once an incumbent user is detected on a channel, actions must be taken to clear lower priority users from that channel.

This paper describes research on a real-time spectrum monitoring infrastructure that would support dynamic, centrally-coordinated sharing. The infrastructure consists of spectrum sensors which measure the power spectrum of a given band and a spectrum occupancy server which receives measurements from the sensors in real time over an internet protocol (IP) network. A client can subscribe to alerts from the spectrum occupancy server on activity in individual channels within the band. Whenever activity is detected in those channels, the server notifies the subscriber of that activity. Activity can be defined as the arrival of a signal, the departure of a signal, or both, in the channel(s) of interest.

This work investigates the feasibility and scalability of such a real-time monitoring infrastructure. A key aspect is measuring the latency of detecting a spectrum occupancy event (e.g., the beginning of a signal transmission). The latency in this context is the time from when the signal transmission begins (or ends) and when notification of this event is received by a subscribed client. The primary components of this latency are due to sensor processing, server processing, and network transmission. Implicit in this transaction is detection of the event by the sensor/server, therefore detection and false alarm rates are important measures in addition to latency. The experimental measurements and analysis reported in this paper, however, focus on latency.

Knowledge of the latency of spectrum event detection in centralized spectrum sharing systems has two important implications. First, it indicates the duration of interference an arriving incumbent could experience from lower priority users, because in order for these users to cease causing harmful interference, they must first be notified to do so. Second, knowledge of the detection latency informs the level of temporal granularity that can be achieved by opportunistic spectrum access. For instance, if a typical white space lasts on the order of tens of milliseconds but its detection takes on the order of hundreds of milliseconds, exploitation of these white spaces is unwarranted.

An example of dynamic spectrum sharing that will use centralized spectrum monitoring is the recently adopted Citizens Broadband Radio Service (CBRS) in the U.S. for 3550 MHz to 3700 MHz. In its rules for CBRS [3], the Federal Communications Commission (FCC) described the illustrative architecture shown in Fig. 1. Here, the spectrum sensing function is provided by the environmental sensing capability (ESC). The SAS uses a combination of databases and ESC measurements to assign channels to CBRS devices (CBSDs) while also protecting federal and commercial incumbents. The rules state that once a federal incumbent signal has been detected by an

ESC at certain frequencies, the SAS must clear CBSDs from those frequencies within 60 s. The clearance time must account not only for the time to detect the incumbent signal and notify the CBSD, but also for the SAS to propagate this information to other SASs that may need to clear their respective CBSDs operating at those frequencies.

This work on real-time spectrum monitoring is part of a larger effort pursued jointly by the National Institute of Standards and Technology (NIST) and the National Telecommunications and Information Administration (NTIA) in the U.S. for spectrum monitoring [4]. This broader effort includes establishing a spectrum occupancy repository in support of spectrum management, policy making, and enforcement. It also includes an evaluation of sensors in terms of their fundamental radio frequency (RF) performance as well as their detection capability [5]. This paper addresses the real-time monitoring component of that overall program and its application to spectrum sharing.

In other related work, Schmid *et al.* [6] studied the transmit and receive latencies of a specific software-defined radio (SDR), focusing on the host-radio interface. While we use an SDR for timed signal generation in this work, the sensor under test need not be SDR-based. Nika *et al.* [7] investigated real-time spectrum monitoring with very low cost SDRs that would enable large-scale, crowdsourced spectrum measurements, but their work did not address the infrastructure of the monitoring agency to which the measurements are reported.

The remainder of this paper is organized as follows. Sections II and III describe the design and implementation of the spectrum sensor and occupancy server, respectively. Section IV describes the method for measuring detection latency and presents an analysis of experimental results. Section V concludes with recommendations for future work.

II. Sensor Implementation

A. Detection Scheme

The real-time detection scheme currently employed is based on frequency-domain energy detection. At the sensor, time-domain complex baseband samples are converted into the frequency domain with a discrete Fourier transform. The amplitude-square of the frequency-domain samples are channelized in frequency and aggregated in time to a desired resolution bandwidth and resolution time and are transmitted to the occupancy server which applies a threshold decision to each channel.

The signal processing at the sensor is performed in a pipeline. Down-converted baseband complex samples are continuously received at the input of the pipeline, and power vectors are continuously transmitted to the server at the output of the pipeline. The monitored bandwidth is limited by the sampling rate of the analog-to-digital (A/D) converter as well as by the computational resources of the signal processor. After initial decimation to the desired observation bandwidth, the time-domain samples are not further sub-sampled, decimated, or time-gapped in the computation of the power spectra.

The stages of the pipelined signal processing are shown in Fig. 2. Complex baseband samples arrive at the sampling rate, f_s, in floating-point format and are arranged into fixed-length vectors of length N, where N is the size of the fast Fourier transform (FFT). Successive vectors are non-overlapping and are processed by the FFT block at the rate f_s/N. The FFT block includes windowing of the time-domain samples using a four-term Blackman-Harris window. The magnitude square of the complex outputs of the FFT are then aggregated (summed) into channels of the desired channel bandwidth.

After channelization, a temporal statistic is computed over a specified time interval, T_m. This statistic can be average or peak. If it is average, then the channelized power vectors are averaged to produce a single output vector each T_m seconds. If it is peak, then a vector of the maximum power measurement in each channel over the T_m interval is output. Finally, the resulting average or peak power vectors are serialized, converted to decibels, scaled, and transmitted as signed 8-bit integers in binary format over a secure transmission control protocol (TCP) socket connection to the occupancy server.

The rate of transmission to the server over the socket depends on the choice of number of channels, C, and measurement interval, T_m, and is equal to $8C/T_m$ b/s. Specific values for these and other parameters are given below in an illustrative example.

B. Implementations

We implemented the sensor detection scheme described above in GNU Radio using existing and custom-developed blocks.[1] The custom blocks are the channelization, time statistic, and SSL socket sink blocks in Fig. 2. The GNU Radio implementation of this sensor was tested with two SDR platforms, the Ettus Universal Software Radio Peripheral (USRP) N210 and the HackRF One, each of which performs down-conversion, A/D conversion, and streams complex baseband (I/Q) samples to a general purpose processor running the GNU Radio application.

Before a sensor starts streaming power spectral measurements to the server, it transmits headers containing the sensor's location, specifics about its RF capabilities and antenna, and a description of the ensuing data (monitored frequency range, number of channels, time resolution, and detection scheme).

The frequency range a sensor can monitor at any given time is limited by its maximum sampling rate and possibly any RF or intermediate frequency filters. We have used the USRP N210-based sensor, for example, to monitor up to 40 MHz at a time. Monitoring wider bandwidths, such as the 150 MHz of CBRS, would necessitate either the use of multiple sensors in parallel or, alternatively, sequential monitoring of sub-bands by a single sensor. Sequential monitoring, however, would increase the latency of spectrum event detection.

C. Sample Measurements

Fig. 3 shows a spectrogram of over-the-air measurements made by an indoor sensor that was tuned to the commercial

[1]Certain commercial equipment, instruments, or materials are identified in this paper in order to specify the experimental procedure adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the materials or equipment identified are necessarily the best available for the purpose.

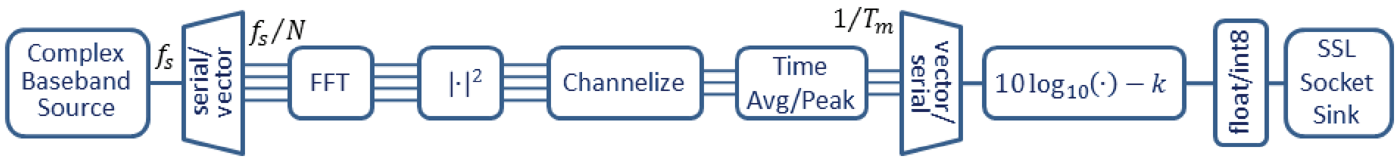

Fig. 2. Sensor processing pipeline

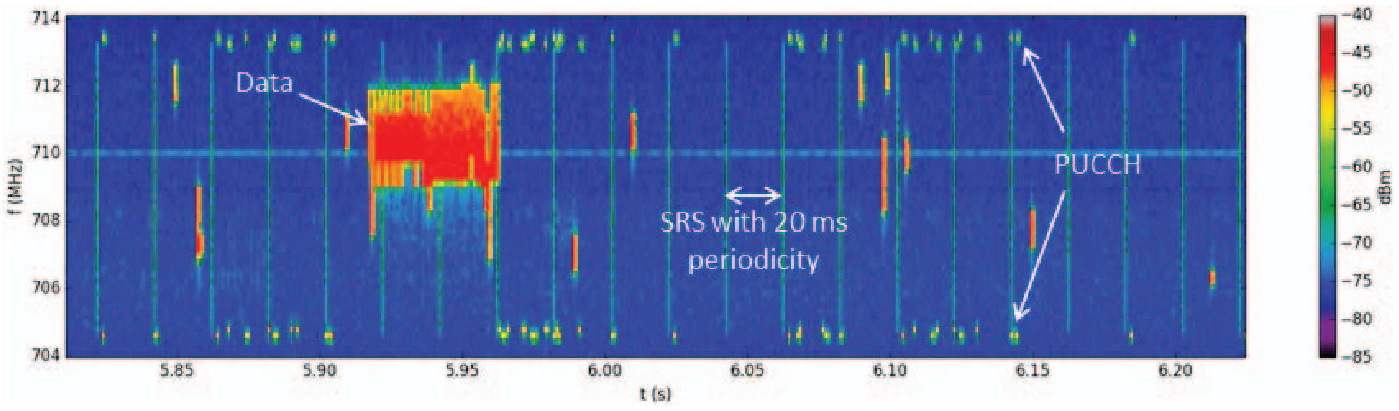

Fig. 3. Measurements of a commercial LTE FDD 10 MHz uplink (band 17) obtained by a real-time sensor

TABLE I. SENSOR SETTINGS FOR MEASUREMENTS IN FIG. 3

Parameter	Description	Value
f_s	Sampling rate	12.5 MHz
N	FFT size	625
T_m	Measurement interval	1 ms
C	Number of channels	56
B_c	Channel bandwidth	180 kHz

long term evolution (LTE) frequency division duplex (FDD) uplink of band 17. This band is nominally 10 MHz wide (9 MHz occupied). The sensor's settings for these measurements are summarized in Table I. The measurement interval, T_m, and channel bandwidth, B_c, were chosen to match LTE's resource block size (1 ms by 180 kHz). The sampling rate and FFT size were chosen to yield an integer number of FFT vectors per measurement interval (20 FFTs per ms) and an integer number of FFT points per channel (9 FFT points per 180 kHz channel).[2] The number of resource blocks in a 10 MHz LTE band is 50; we chose a slightly larger number of channels (56) to include the guard bands. The total measured bandwidth is 56×180 kHz, or 10.08 MHz, centered at 709 MHz. In this configuration, the sensor streams data to the server at a rate of $8 \times 56/0.001$ b/s $= 448$ kb/s.

The emissions on this band come from LTE mobile devices, or user equipment (UE), transmitting to the base station (eNodeB). One can clearly distinguish certain features of the LTE uplink in the spectrogram of Fig. 3. For example, the wideband sounding reference signal (SRS) from a single UE is repeated every 20 ms. (LTE standards permit SRS periodicity to range from 2 ms to 320 ms [8].) Transmissions on the physical uplink control channel (PUCCH) are seen at the upper

[2]The LTE sampling rate of 15.36 MHz was not readily available on the SDR. The selected 12.5 MHz sampling rate is easily derived from the SDR's 100 MHz clock.

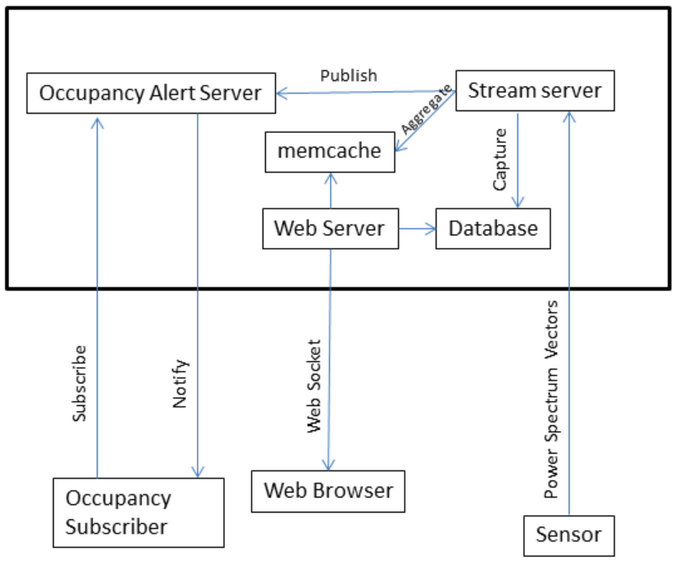

Fig. 4. Architecture of the spectrum occupancy server

and lower edges of the band. Data transmissions occur in between the edges, with one large transmission in this case seen between 709 MHz and 712 MHz.

III. SPECTRUM OCCUPANCY SERVER

Fig. 4 depicts the architecture of the spectrum occupancy server. A streaming server runs as a separate process and handles inbound connections from sensors. When a connection from a sensor is received, the streaming server forks a new Python interpreter in a separate process to handle the inbound connection and sensor data stream. The streaming server keeps track of an occupancy vector. An occupancy vector is a binary vector which is computed by comparing each power value in the received power vector to a user-defined threshold.

Occupancy alerts are delivered using a publish/subscribe interface. Each time the occupancy vector changes, the streaming server publishes a new occupancy vector to an occupancy alert server, which, in turn, notifies subscribed external clients of occupancy changes. A client may subscribe to alerts by connecting to the alert server and providing a sensor ID. It then reads the socket through which it is connected. Each time the occupancy changes, the occupancy alert server writes out a new occupancy vector.

In addition to generating alerts to subscribed users of changes in spectrum occupancy, the spectrum occupancy server also supports the display of a running spectrogram in a client's web browser. To support this graphical user interface (GUI), the streaming server does the following processing with the inbound data for presentation to a web browser:

1) Gathers the data into an aggregated vector with a user-defined aggregation window size (which can be longer than the time interval, T_m, of incoming vectors).
2) Applies an aggregation filter in time on the aggregated data, either mean or maximum (peak-hold).
3) Writes the aggregated vector into a memory cache.

The moving spectrogram in the web browser is entirely driven by server push. This is implemented as follows: The web browser keeps a persistent connection to the web server via a web socket. The web server services this connection by writing the aggregated spectrum out to the web socket at periodic intervals. In order to do so, the web server looks in the memory cache that it shares with the streaming server for updates. Only a single aggregated spectrum is maintained in the cache for a given sensor.

In our current implementation, the sensor continuously transmits power spectral measurements to the server, where the thresholding to detect occupancy changes is performed. An alternative approach would be to apply the thresholding at the sensor, and have the sensor notify the server only when changes in occupancy are detected. The latter approach would certainly reduce the load on the network and on the server. However, the availability of power spectral measurements at the server permits it to fuse measurements from multiple sensors as well as to store the measurements in a database for later retrieval and analysis. For example, the stored data can be used to tune detection thresholds and prediction algorithms.

IV. Spectrum Monitoring Latency

A. Measurement Procedure

We describe a simple, black-box method to measure the latency of a centralized spectrum monitoring system. Specifically, this procedure measures the time between the occurrence of a spectrum occupancy event (e.g., start of a signal transmission) and the delivery of the notification of this event to a subscribing client.

The procedure uses a test signal generator, the sensor and server under test, a client process receiving alerts from the server, and a clock that is common to test signal generation and alert reception, arranged as shown in Fig. 5. In our tests, the test signal generator consists of an SDR (Ettus USRP N210) connected to a general purpose host. A process on this host generates a pulsed continuous-wave (CW) signal through the SDR at regular intervals. The signal generator's RF output port is directly connected via coaxial cable and an attenuator to the RF input port of the sensor. The sensor streams power spectral measurements to the spectrum occupancy server over a local area network. A client process subscribed to receive occupancy alerts from the server runs on the same machine that hosts the test signal generator. The host clock is used to measure the time between transmission of a pulse and the reception of the rising-edge occupancy alert by the client process.

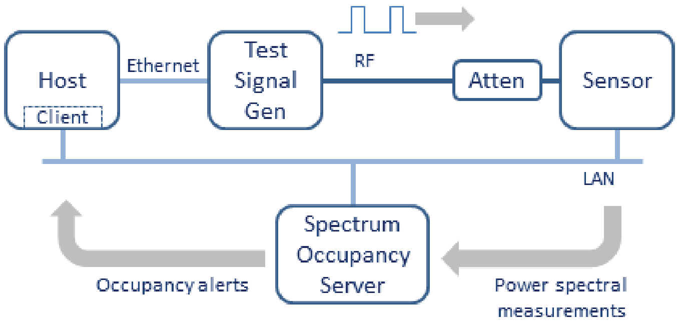

Fig. 5. Latency test arrangement

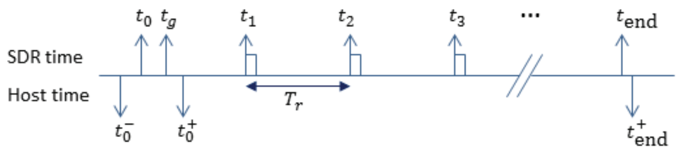

Fig. 6. Measurement timeline

In order to measure latency, we wish to have the transmitter send the pulse at a known, deterministic time. Because of random delays on the interface between the host and transmitting SDR, we cannot rely on the host clock alone to time the transmission of the pulse. Instead, we use the universal hardware driver's (UHD's) scheduled transmission feature based on the SDR's internal clock. While this feature allows us to schedule pulse transmissions deterministically with respect to the SDR clock, there is still some uncertainty in knowledge of the SDR's time since we use the host-SDR interface to set and read the SDR's time register. Furthermore, relative drift between the host and SDR clocks is another source of uncertainty. These sources of uncertainty and the latencies themselves are measured as follows.

The signal generator transmits a train of pulses with a uniform pulse repetition interval, T_r. Prior to sending the first pulse, the time register on the SDR is initialized to zero (denoted as time t_0 in Fig. 6). The host clock is timestamped just before issuing this initialization command to the SDR. This timestamp is denoted as t_0^- in the figure. The host then issues a command to read the SDR's time register. The response is received by the host at time t_0^+ and contains the SDR time t_g. The uncertainty of t_0 in host time is now lower-bounded by t_0^- and upper-bounded by $t_0^+ - t_g$.[3]

The host schedules pulses for transmission by the SDR at multiples of T_r from t_0, that is, at times $t_n = t_0 + nT_r$, $n \geq 1$. The host timestamps each rising-edge occupancy alert it receives. Let these timestamps be denoted by t_n^+. The latency of the n^{th} pulse is calculated as $t_n^+ - \left(t_0^- + nT_r\right)$, the alert reception time minus the pulse's scheduled transmission time.

At the end of the pulse train, the host issues another command to read the SDR's time register. The response is received by the host at time t_{end}^+ and contains the SDR time t_{end}. Clock drift over the duration of the pulse train is calculated as $D = \left(t_{\text{end}}^+ - t_0^-\right) - \left(t_{\text{end}} - t_0\right)$.

Accounting for the uncertainty of the SDR's initialization time, t_0, these latency measurements overestimate the true latency by up to $\Delta_l = t_0^+ - t_g - t_0^-$. Accounting for the uncertainty due to host-SDR clock drift, the latency measurements can either over- or underestimate latency, depending on the direction of the relative drift. If the drift, D, as defined above, is positive—that is, elapsed host time $\left(t_{\text{end}}^+ - t_0^-\right)$ is larger than elapsed SDR time $\left(t_{\text{end}} - t_0\right)$—that implies the SDR clock is slower, actual pulse transmissions are slightly delayed, and the measured latencies are overestimates. If, on the other hand, the drift is negative, then the SDR clock is faster, pulses are sent slightly ahead of schedule, and the measured latencies are underestimates. Thus, the latency, L, can be bounded as follows:

$$\begin{array}{ll} \tilde{L} - \max\{\Delta_l, D\} < L < \tilde{L} & ; \quad D > 0 \\ \tilde{L} - \Delta_l < L < \tilde{L} + |D| & ; \quad D < 0 \end{array} \qquad (1)$$

where $\tilde{L}$ is the measured latency.

B. Latency Measurements

We conducted measurements of latency with two different sensors: (Sensor 1) an Ettus USRP N210 with an Ethernet interface to a host on the local area network (LAN) in Fig. 5, and (Sensor 2) a HackRF One with a universal serial bus (USB) interface to the host. Each sensor monitored the 10 MHz band from 719 MHz to 729 MHz. The test signal generator transmitted a pulsed CW signal at 724.09 MHz with a pulse duration of 10 ms and a pulse repetition period of $T_r = 10$ s. Both sensors were connected and tested simultaneously using a conducted RF connection between the test signal generator and each sensor.

Host timestamps were recorded in terms of CPU clock cycle counts, and latencies were measured in terms of the number of elapsed clock cycles and converted to seconds using the measured clock frequency (approximately 2.5×10^9 cycles per second). Clock cycle counts were obtained using the RDTSCP[4] assembly instruction described in [9]. Furthermore, CPU_SET was used by sched_setaffinity to restrict the RDTSCP to a single core of a single processor.

We collected latency measurements over a period of 29 hours. Initial experiments revealed a relative clock drift between the host and transmitting SDR clocks reaching several milliseconds over this length of time. To mitigate this drift, we reinitialized the SDR on an hourly basis, obtaining new values of t_0^-, t_g, t_0^+, and t_{end} each time.

[3]We regard as negligible the relative clock drift over the short interval, t_g.

[4]Read Time-Stamp Counter and Processor ID

TABLE II. SUMMARY STATISTICS OF LATENCY MEASUREMENTS

Statistic	Sensor 1	Sensor 2
Minimum	7.3 ms	24.3 ms
Maximum	1960.9 ms	1960.6 ms
Mean	17.3 ms	35.5 ms
Median	12.1 ms	29.9 ms
Standard Deviation	33.0 ms	33.4 ms
95th percentile	43.4 ms	70.8 ms

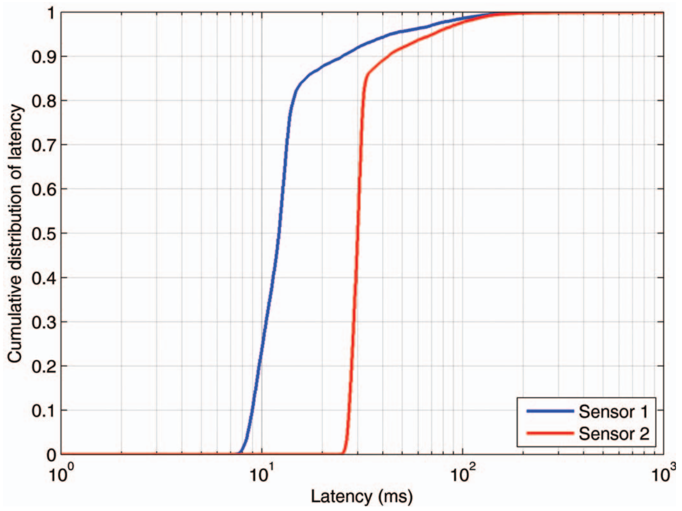

Fig. 7. Measured cumulative distribution of total latency

Table II summarizes the measurement statistics for both sensor configurations. Over 10 thousand measurements were collected for each. The cumulative distributions of these measurements are shown in Fig. 7. We observe that, for Sensor 1, the median latency is 12.1 ms and the 95th percentile latency is 43.4 ms. While the large majority of latencies are under 15 ms (83 %), larger values are likely due to server and or LAN busy periods. Sensor 2, a less expensive device with a USB interface, has somewhat higher latencies. Its alerts were received with a median latency of 29.9 ms and a 95th percentile of 70.8 ms.

Over the hourly initializations of the test signal generator, measurements of Δ_l averaged 190 μs with a standard deviation of 20 μs and a maximum of 229 μs. The relative host-SDR clock drift, D, was observed to be positive at the end of each hour, with an average value of 801 μs, a standard deviation of 78 μs, and a maximum of 962 μs. Given these numbers, and using the bounds in (1), we can conclude that the latency measurements reported above overestimate the true latency by no more than 1 ms.

While these measurements were conducted for systems on a local area network, the extension to a wide area network (WAN) is straightforward. In fact, simply adding the estimated network transmission delays of the WAN to these measurements would provide a good first-order approximation.

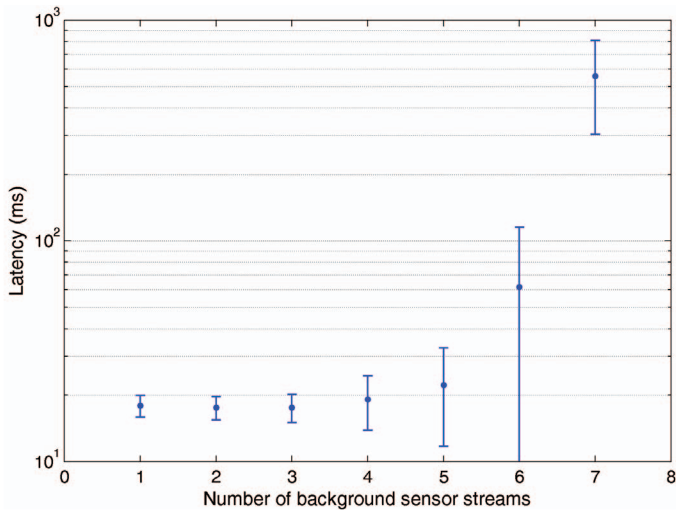

Fig. 8. Server latency as a function of load from concurrent sensor streams

C. Latency vs. Server Load

The results reported above are for a lightly loaded server handling only two concurrent sensor streams. We also tested the performance of the server as a function of increasing streaming load. This test generates a number of background streams from load generators that concurrently stream captured data from a file to the server. Each stream mimics a sensor by streaming a power vector every millisecond from the load generator. We then start a test stream that contains a periodic narrowband pulse at one second intervals, and we measure the time for an alert to be delivered for each pulse. Timings of the pulse transmissions and receptions of the alerts are done on the same machine using the processor's clock.

Fig. 8 plots the average latency as a function of the number of background streams. The averages are over 200 measurements, and the error bars are plus/minus the standard deviation. We start seeing degradation beyond five background streams running in addition to the test stream.

Given that we have implemented our system in Python, the streaming service (and each of its forked children) and the web services containers each run in their own process in order to better utilize server resources. This separates the web server load from sensor load and allows the system to scale with the number of cores on the server. The test server is a four-core 64-bit Linux server with 8 GB of physical memory and 2.5 GHz processors. When streaming with 1-ms power spectrum resolution, each stream utilizes 60 % of one core. Lowering the temporal resolution to 50 ms per power spectrum reduces core utilization to 5 %, at the expense of higher detection latency. While Python was used for quick prototyping, we plan to re-write this component using Java or C++ in order to further reduce resource utilization and improve scalability.

V. Conclusion

In this paper, we described a prototype implementation of real-time centralized spectrum monitoring. In our implementation, the sensors continuously stream power spectral measurements to a central server which analyzes them for changes in spectrum occupancy. The server issues spectrum occupancy alerts to subscribed users whenever it detects changes in the occupancy of individual channels within the band. The server also supports a web browser GUI to display a running spectrogram of the monitored band. We have used this implementation for live monitoring of commercial LTE and other bands, demonstrating the feasibility of real-time spectrum monitoring.

To assess how "real-time" the monitoring is, we proposed a test method for measuring the latency of the system, specifically the time from when a spectrum event begins to when a corresponding alert is received. The proposed method uses a test signal generator made from an SDR and a general purpose computer. As a black-box test, any sensor-server implementation can be tested without modification to the remaining test apparatus. The only requirement is a common procedure and format for sending the occupancy alert.

Spectrum detection latency has important implications for dynamic spectrum sharing systems. It lower bounds the time it takes to clear a channel in which an incumbent user arrives. It also indicates the level of temporal granularity of opportunistic spectrum access that can be achieved in centrally-coordinated spectrum sharing systems. We used the proposed test method to measure detection latencies through our server with two different sensors and found that in both cases the 95th percentile was under 80 ms.

To complement the latency test, future work should address methods to evaluate the detection and false alarm rates of monitoring systems across a variety of RF channel conditions, including a range of signal-to-noise ratio and channel impairments. We also plan to implement and deploy more sophisticated sensing algorithms in the monitoring system that are more resilient to noise and interference and are able to discriminate between heterogeneous signals, such as incumbent and secondary signals. An example of this would be the ability to detect an incumbent radar signal in a background of LTE and other signals for the 3550 MHz to 3700 MHz CBRS. One could then examine the tradeoff between detection performance and detection latency.

Acknowledgment

The authors thank Michael Cotton and his colleagues at the National Telecommunications and Information Administration, Institute for Telecommunications Sciences, Boulder, Colorado, for a fruitful collaboration on spectrum monitoring, out of which this work came about.

References

[1] President's Council of Advisors on Science and Technology, "Realizing the full potential of government-held spectrum to spur economic growth," Jul. 2012. [Online]. Available: http://www.whitehouse.gov/sites/default/files/microsites/ostp/pcast_spectrum_report_final_july_20_2012.pdf

[2] M. Matinmikko, H. Okkonen, M. Palola, S. Yrjola, P. Ahokangas, and M. Mustonen, "Spectrum sharing using licensed shared access: the concept and its workflow for LTE-advanced networks," *IEEE Wireless Communications*, vol. 21, no. 2, pp. 72–79, April 2014.

[3] “Amendment of the commissions rules with regard to commercial operations in the 3550–3650 MHz band,” FCC Report and Order and Second Further Notice of Proposed Rulemaking, GN Docket No. 12-354, Apr. 2015. [Online]. Available: https://apps.fcc.gov/edocs_public/attachmatch/FCC-15-47A1.pdf

[4] M. Cotton, M. Souryal, and M. Ranganathan, “An overview of the NTIA/NIST spectrum monitoring pilot program,” in *Proc. International Workshop on Smart Spectrum*, Mar. 2015.

[5] J. Wepman, B. Bedford, H. Ottke, and M. Cotton, “RF sensors for spectrum monitoring applications: Fundamentals and RF performance test plan,” National Telecommunications and Information Administration, Technical Report TR-15-519, Aug. 2015. [Online]. Available: http://www.its.bldrdoc.gov/publications/2808.aspx

[6] T. Schmid, O. Sekkat, and M. B. Srivastava, “An experimental study of network performance impact of increased latency in software defined radios,” in *Proceedings of the Second ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization*, ser. WinTECH ’07. New York, NY, USA: ACM, 2007, pp. 59–66. [Online]. Available: http://doi.acm.org/10.1145/1287767.1287779

[7] A. Nika, Z. Zhang, X. Zhou, B. Y. Zhao, and H. Zheng, “Towards commoditized real-time spectrum monitoring,” in *Proceedings of the 1st ACM Workshop on Hot Topics in Wireless*, ser. HotWireless ’14. New York, NY, USA: ACM, 2014, pp. 25–30. [Online]. Available: http://doi.acm.org/10.1145/2643614.2643615

[8] “Evolved universal terrestrial radio access (E-UTRA); physical layer procedures,” 3GPP Technical Specification TS 36.213, V12.5.0, Mar. 2015.

[9] G. Paoloni, “How to benchmark code execution times on Intel IA-32 and IA-64 instruction set architectures,” Intel Corporation, white paper, Sep. 2010. [Online]. Available: http://www.intel.com/content/www/us/en/intelligent-systems/embedded-systems-training/ia-32-ia-64-benchmark-code-execution-paper.html