COmBAT: Cross-layer Based Testbed with Analysis Tool Implemented Using Software Defined Radios

Jithin Jagannath*†, Hanne Saarinen*, Timothy Woods*, Joshua O'Brien*, Sean Furman*,
Andrew Drozd*, Tommaso Melodia†
*ANDRO Advanced Applied Technology, ANDRO Computational Solutions, LLC, Rome NY,
{jjagannath, hsaarinen, twoods, jobrien, sfurman, adrozd}@androcs.com
†Department of Electrical and Computer Engineering, Northeastern University, Boston MA, melodia@ece.neu.edu

***Abstract*—In this paper, we discuss the implementation of a CrOss-layer Based testbed with Analysis Tool (COmBAT). COmBAT is developed to enable the design and development process of next-generation cross-layer based wireless communication technologies for tactical ad-hoc networks. The COmBAT architecture comprises of two major components; (i) Adaptive cross-layer (AXL) framework implemented on each node in the testbed and (ii) NEtwork Analyzing Tool (NEAT) that provides a graphical interface for users to track and analyze network metrics as well as provide a single seat control over the network parameters on-the-fly. In this paper, we discuss the design and implementation of these components in detail and demonstrate its feasibility by implementing three cross-layer based algorithms on COmBAT using a nine node ad-hoc network. The results validate the modularity and adaptability of COmBAT and demonstrate how COmBAT can be used to develop as well as refine current and future cross-layer algorithms providing a feasibility study that lends itself to the transition of theoretical network research from testbed to relevant military hardware.**

I. Introduction and Background

Wireless communication is a critical component of todays' modern military. Over the past decades much work has been performed to improve spectrum utilization by enabling dynamic spectrum allocation (DSA) [1], [2]. Past research has shown that it is advantageous to examine interactions between routing, spectrum allocation, session management and channel conditions for the purpose of developing a cross-layer algorithm that is capable of jointly maximizing different parameters (throughput, delay, among others.) of the network [3], [4]. The cross-layer based approaches have been bolstered by the advent of software defined radios (SDRs). In recent past, there have been several efforts to increase the flexibility of radios to provide more dynamic reconfiguration capabilities.

GNU radio is an open-source signal processing software that provides great flexibility specifically at the physical layer of SDRs. GNU radio comprises of various signal processing and digital communication blocks and is an excellent tool to control SDRs. However, the majority of the contribution is limited only to the Physical layer. There has also been an effort to relocate some of the processing functions to Field-programmable gate array (FPGA) [5] to improve the delay performance. This makes it difficult to integrate new algorithms for testing and evaluation purposes. Some other works [6], [7] aim to provide reconfigurable MAC protocols by decomposing the overall design into core fundamental blocks. In [6], the implementation of these fundamental blocks are split between PC and FPGA depending on the time critical nature of the blocks. In [7], the authors implement an abstract execution machine on a resource-constrained commodity WLAN card. Recently, software defined network (SDN) using an Open-Flow [8] based approach has been proposed for evaluating routing protocols. The overall concept of Open-Flow is to keep the data path on the Open-Flow switch itself while moving the high-level routing decision to a separate controller (server). The switch performs packet forwarding based on the flow table defined by the controller and use Open-Flow protocol to communicate with each other. The majority of the work on OpenFlow has been concentrated at the network layer of the protocol stack. Real-time reconfigurable radio framework with self-optimization capabilities (RcUBe) [9] is a flexible design that promotes a cross-layer approach to develop a reconfigurable protocol stack. The RcUBe framework was developed to provide the required modularity to implement adaptive algorithms that can reconfigure SDR dynamically on-the-fly. In this work, we adopt some of the design concepts of RcUBe to develop the cross-layer framework required for COmBAT. The design is discussed in detail in section II.

Even with these advancements, a major challenge faced by developers who are transitioning algorithms and protocols to military grade hardware is the lack of a cross-layer based multi-hop testbed architecture that enables easy implementation of cross-layer technologies. These testbeds are essential to corroborate the results obtained in simulations and evaluate how to refine these algorithms to ensure a successful transition process. Some of the requirements of such a testbed include, a flexible cross-layer based protocol stack [9], modularity to integrate new algorithms with ease, framework to accommodate both centralized and distributed solutions, tools to monitor the performance of the network in real-time, and having the ability to run unsupervised scripted experiments over extended periods of time. Having such a testbed expedites the design and development process of next-generation wireless communication technologies destined for a military SDR system. Therefore, in this paper we discuss the design and implementation of COmBAT developed using SDRs. COmBAT is envisioned to become a state-of-the-art platform for designing, developing and validating cross-layer based algorithms and protocols before transitioning to a military platform. To validate the modularity and flexibility of COmBAT, we have implemented and evaluated the joint

[1]This material is based upon work supported by the US Air Force Research Laboratory under Award No. FA8750-14-C-0098 and Award No. FA8750-16-C-0086

978-1-5090-3781-0/16/$31.00 ©2016 IEEE

routing and spectrum allocation algorithm (ROSA) [3], routing with fixed allocation (RFA) and routing with dynamic allocation (RDA). We further discuss how the real-time monitoring ability is used to analyze the performance of the network in terms of throughput, spectrum management and instantaneous routing decision.

The rest of the paper is organized as follows. In Section II, we discuss the design and implementation of COmBAT. Next in Section III, we describe the evaluation process to establish the feasibility of COmBAT and prove its adaptive capabilities. Finally, the conclusion is presented in Section IV.

II. COMBAT DESIGN

In this section, we discuss the design and implementation of different constituents of COmBAT.

A. Adaptive Cross-Layer (AXL)

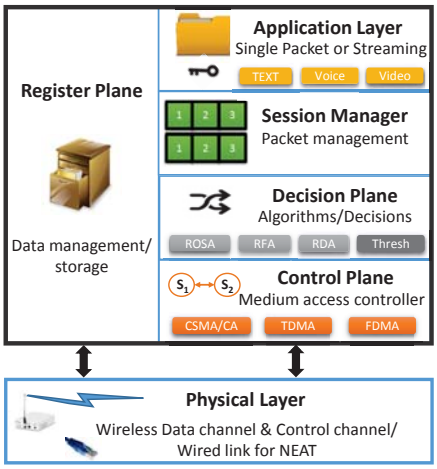

Fig. 1: Adaptive cross-layer (AXL) framework.

The overall AXL framework used in COmBAT is depicted in Fig. 1, which is an adaptation of the RcUBe framework. AXL retains three essential planes proposed by RcUBe namely; the decision plane, register plane and control plane. Each node in the network uses the AXL framework in place of a traditional protocol stack and are therefore referred to as AXL nodes throughout this paper. As noted, AXL consists mainly of the application layer, session manager, decision plane, control plane, register plane and the physical layer. Below, we discuss the implementation of these layers/planes in detail.

In implementation, the AXL framework consists of Python multiprocessing processes which are initialized at node startup using an AXL daemon. The daemon imports the main modules and properties that are used in the different layers/planes of the framework as required. The properties include predefined values for the network such as data timeout duration, node IP and MAC addresses, payload sizes and limits on the number of sessions the framework supports. However, most of the properties are dynamic in nature and they can be reconfigured on-the-fly based on network optimization strategies or user input. The processes that are started by the daemon run continuously until shutdown. These processes include the register plane, session manager, control plane and the physical layer. The decision plane is not a process but a collection of functions that can be called by the framework when needed. Each layer/plane can share information with each other by a combination of three methods; direct function calls, shared memory or by overhearing global events (global with respect to node, not the entire network) that can be triggered by any process in the framework. These functionalities allow for a flexible cross-layer communication between all network protocols.

Application (APP) Layer. The current AXL software package provides two data generation APPs to evaluate the performance of the network. The APPs support either text or audio and can be operated in packet streaming mode or packet-by-packet mode. For streaming mode, the source data is repeated until a user specified amount of data has been generated. The streaming mode is generally used in experiments requiring a constant bit rate (CBR) source for a fixed duration of time. Packet-by-packet mode is used to evaluate the effect of bursty transmissions or alternatively as an end-to-end user friendly APP for simple network demonstrations. The audio applications are implemented using Python and a standard Linux library called Advanced Linux Sound Architecture(ALSA). The APPs connect to the AXL daemon via a TCP/IP socket. For each APP that connects, a unique connection object is created that manages data transfer between the APP and the AXL framework. When this connection is created, the APP declares itself as either a text or audio APP and whether or not it will be sending or receiving data. The sending APP passes each packet to the connection object via the socket interface. Each packet contains the user generated parameters and quality of service (QoS) parameters. The packet is parsed and then sent to the session manager for the next processing stage. Receiving applications listen on the socket interface until data is pushed up the stack. When data is received by the application, the message will be played back to the user as text or audio accordingly.

Session Manager. In the AXL framework, the session manager provides the capability of simultaneous multi-session management. When a packet arrives at the session manager, the session manager creates a session object based on the packet parameters, which include process ID that is created by the OS, source and destination IP, data type, any QoS parameters, and the packet number generated at packet creation. Packets that correspond to existing session objects are appended to their appropriate session queue. Packets receive their network headers based on the parameters in the session object mentioned earlier. Packets that have arrived at their destination node are stripped off of their network headers and forwarded to the receiving APP via AXL daemon.

In implementation, the session manager is designed as a multiprocessing first in first out (FIFO) with a user specified update period. The update period dictates the timeout for updating packet queues. During each update period, the session manager stores the current packet queue length of each session in the register plane and triggers an event flag which indicates that the transmitter has backlogged session ready for routing decisions.

Decision Plane. As the name suggests, this is the component where all the logical decision making and algorithm

executions take place. These algorithms pertain to routing algorithms, spectrum allocation, automatic modulation classification and other resource allocation decisions. The complexity of the algorithms can vary from threshold decision to iterative algorithms like the Expectation and Maximization (EM) algorithms or solvers for convex optimization problems. Decision algorithms can (i) modify a parameter in a protocol, (ii) trigger switching among different modes within a protocol, and (iii) enable switching among different protocols altogether [9].

In implementation, there are currently three routing algorithms, stored as software modules, available for use (discussed in detail in section III). Each routing module can be passed as an instance for the decision plane to use during runtime. The chosen and active routing module is requested by the session manager to execute the algorithm when a session is backlogged. The results of the executed algorithm is stored in the register plane for other layers/planes to access. Conversely, to execute the algorithm, the decision plane obtains the information from the register plane. After executing the algorithm within the routing module, the decision plane triggers an event flag that prompts the control plane to schedule a transmission.

It should be noted that the interactions between the decision plane and the other layers/planes in AXL take place via the register plane, through global events or through direct function calls. Therefore, when users want to implement a new decision algorithm, they can simply include their new decision module as part of the framework and initialize it in the AXL daemon as a new routing option. It is recommended that the new module follows a similar software structure as the already existing algorithms, so that the need to change any function or event name from inside the module is minimized. If the new module is triggered by something other than events that are currently included in the AXL framework, the user can add new events via an initialization file. The user can achieve the required interaction between the AXL layers/planes by setting and clearing the events inside the framework.

Control Plane. The control plane houses the control logic used to access the wireless medium. The control plane contains the finite state machine (FSM) used to implement different MAC protocols. The chosen MAC protocol defines the exact set of states, events, conditions and actions required to operate FSM. The control plane can be initialized to use multiple different MAC protocols depending on the situational awareness gathered from other layers/planes of the stack as shown in [9]. Each MAC protocol should have its FSM implemented in the control plane as a separate FSM initialization function. Future developers can take advantage of the baseline FSM model that is already defined in the control plane by modifying its states and actions as needed by the protocol. An example of a state transition diagram for a carrier sensing multiple access with collision avoidance (CSMA/CA) based MAC protocol [3] is given in Fig. 2. The state transition diagram describes the interaction between all possible states, events and actions for the receive and transmit paths. As shown in Fig. 2, when an event *Data_available* is set, *Send_RTS* (request-to-send) action is taken as the FSM goes from an idle state to wait *CTS* (clear-to-send) state. The next event that the FSM is looking for is either CTS received or CTS timeout and the FSM transitions depending on which event was observed. The rest of the state transition diagrams can be interpreted in similar manner.

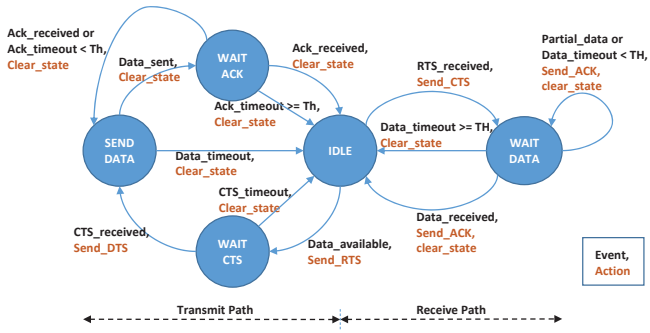

Fig. 2: Finite state machine in control plane.

The FSM is generally in an idle state until the corresponding global AXL events are flagged to invoke a state transitioning process. These global AXL events are used in cross-layer communication between different layers/planes and should not be confused with the events used by the FSM itself. The events in the FSM are strictly defined by the chosen MAC protocol and dictate the state transitioning process that allows the control plane to manage medium access. The global events such as *SESSION_ROUTING* that transition the FSM from an idle state are usually set in the decision plane after routing decision has been made. Some other examples of such global events used throughout AXL include *SESSION_PROCESSING* event which is used by the session manager to indicate that the node is busy processing a session and *START_SENSE* event that prompts the PHY layer to perform spectrum sensing. Therefore, we can state that the overall AXL framework follows an event driven design.

Register Plane. The register plane is essentially a node database used to share information across layers/planes. Although the register plane does not perform any computation and does not have any decision making ability, it is an integral part in the overall cross-layer design. The register plane can be considered a central information hub that can be accessed by different layers/planes of the AXL framework. Data sharing among multiple processes is achieved through Python manager dictionaries. The global information that needs to be shared among all layers is stored in a manager dictionary which allows for only one process to read or write information in the register plane at a time. The main dictionaries that reside in the register plane are a global register dictionary (GRD), global values dictionary (GVD) and session backlog dictionaries (SBD).

AXL nodes learn about their environment by overhearing control packets on the common control channel (CCC) discussed further in section III. Each node stores local information in a node dictionary in the GRD. The node dictionary is appended to every control packet sent on the CCC. The information in the node dictionary is continuously updated as new information becomes available. Node dictionary information includes IP and MAC addresses, the node location, local noise plus interference, session packet queue lengths, current routing algorithm among others. Nodes maintain a copy of its own node dictionary, as well as a copy of its neighbor's dictionary in the GRD. The GRD also contains information like the designated frequencies, possible next hops and neighbors. The GVD stores the current routing decision parameters as well

as the current state of the FSM. SBD has a list of all local sessions and their most up to data packet queue lengths. The routing algorithm is able to access this information stored in the register plane as it optimizes the routing parameters. Other layers/planes can similarly read or write information in the register plane as needed. Additionally, in contrast to RcUBe, the register plane of each AXL node is also connected to an external analysis tool that collects updated network information through an Ethernet connection. By tapping into each node's register plane, the analysis tool is able to display important information about the current status of the network.

Physical (PHY) Layer. The PHY layer is easily separable from the rest of the framework as the goal is to allow the integration of different radio frontends and signal processing software. The PHY layer consists of a hierarchical implementation where the lowest level includes signal processing software specific libraries such as GNU Radio and a universal hardware driver (UHD) interface used with the universal software radio peripheral (USRP) family of products from Ettus. The PHY hierarchical module, consists of functions that are directly accessible by the control layer and the register plane. This implementation allows for a simple interface between AXL and a PHY layer making this design SDR hardware agnostic.

B. Network Analysis Tool (NEAT)

Implementation. NEAT is a network global entity that is connected to each AXL node via Ethernet as shown in Fig. 4. The global entity implies that NEAT can receive and send messages to all AXL nodes present within the testbed. NEAT is connected to each node through TCP/IP sockets using two ports to relay data between the AXL nodes and itself. One port is used to listen to the control packets exchanged by the nodes. Each node that transmits a control packet on the wireless control channel also transmits the same for NEAT over the Ethernet as a mirror copy. The second port is used for direct exchange of data between NEAT and a specific node's register plane. The use of these two different ports is to listen for control packets and register plane data in independent threads.

In implementation, Qt Designer is used to develop the graphical interface and the PyQt library is used to link generated Qt Designer code with python code. NEAT utilizes three classes; one for the graphical aspect of NEAT, one for exchanging data with the register plane of the nodes, and the other for listening to CCC. The slots and signals of PyQt allow data to be exchanged between the graphical class and the other two classes. Therefore, slots and signals allow data acquired from the register plane and control packets to initiate changes to the displayed graphics.

Additionally, it should be noted that since COmBAT is envisioned to be used with different types of SDR it is important to separate the communication required for NEAT from the actual wireless links used by the network to perform metric evaluation. This is the reason why NEAT uses Ethernet rather than wireless links for monitoring. This will also ensure reliability and consistency in the performance of NEAT irrespective of the SDR being used. In cases where we want NEAT to emulate a command and control (C2) node using only wireless link, we will have to redesign NEAT. One possible solution will be to extend the idea of internet of things (IoTs) where every node is connected to the cloud and C2 Node is able to monitor and control the nodes using the cloud interface.

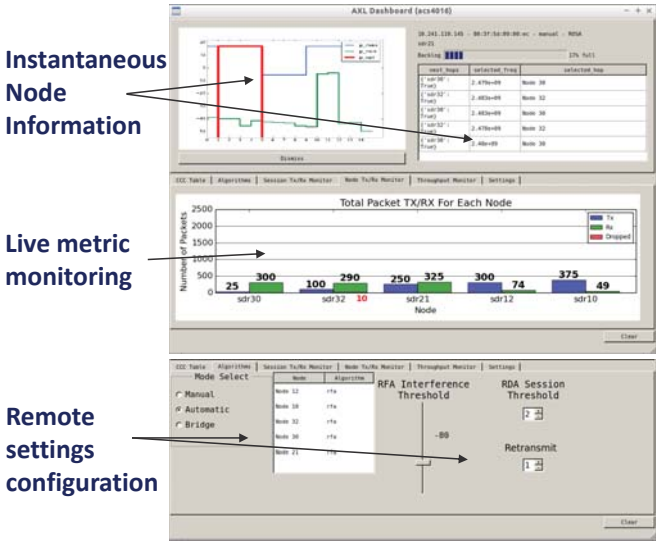

Fig. 3: Screenshots of NEAT.

Features. The major functionalities of NEAT can be divided into three different categories; (i) accessing instantaneous node information, (ii) live Metric monitoring and (iii) remote configuration control. NEAT collects the IP addresses and MAC addresses of every AXL node currently active in the network. As shown in Fig. 3, users can monitor each node's instantaneous queue length, channel sensing information, routing algorithm, optimization decisions, among others. NEAT observes all the control packets exchanged between different AXL nodes. All these information allows the user to check the status of the network and identify nodes that are faulty or not responding. This eliminates the need to be at each node's location to verify the operation of individual nodes.

Performance metrics are important for any network evaluation process. NEAT currently provides live updates of the throughput achieved by the overall network and individual sessions. Additionally, number of packets transmitted, received and dropped by selected nodes of the network can also be displayed in real-time. This functionality helps the user to visualize how packets are being handled at each node, making it easy to identify any traffic bottlenecks present in the network. These metrics are also used to compare the performance of different algorithms with each other. As need arises, users can add other metrics to be monitored in the network in similar manner. As mentioned earlier, the other functionality of NEAT is the ability to trigger configuration changes in the nodes on the network. For instance, NEAT can manually switch the routing algorithms being used by the nodes in the network. Users can also set the network to an automatic mode, via NEAT, where each node dynamically chooses a routing algorithm depending on the situation awareness gathered from its neighbors. Other parameters that can be altered on-the-fly include; number of retransmission attempts by the control plane, different thresholds used by decision plane, among others that a developer can specify.

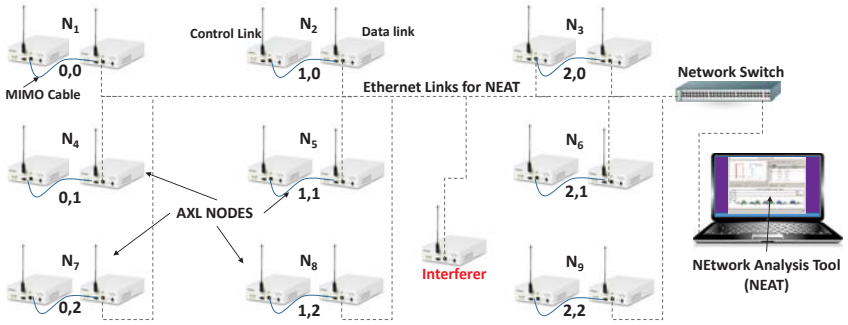

Fig. 4: Nine Node COmBAT Topology.

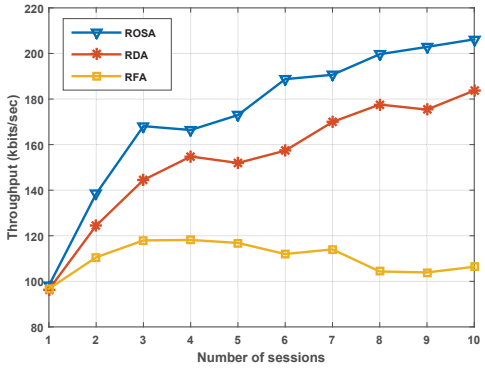

Fig. 5: Througput vs Number of sessions.

III. Testbed Implementation and Evaluation

Figure 4 depicts the current configuration of COmBAT consisting of a nine node network that is arranged in a grid topology. Each node consists of two USRPs (control and data) connected to each other via a MIMO cable. Each node is connected to a Linux-PC (not shown in the figure) and the PC is then connected to the network switch. NEAT is running on an independent PC that is also connected to the switch and performs live network monitoring. Though the overall design of AXL is abstracted from the target SDR platform, here we use USRP N210s along with GNU Radio modules to achieve the required PHY layer adaptability. The AXL framework for each node is implemented using Python programming language on the PC connected to each node. The advantage of using Python is ease of programming and faster development turnaround time. The drawback is large delays incurred by the framework. In future, we plan to move the implementation of the AXL framework to the kernel space using C++ programming language. Since the goal of this work was to establish the feasibility of the proposed cross-layer testbed, we will show how different algorithms are implemented and compared to each other's performance using the current configuration.

The wireless channel is divided into two non-overlapping channels called CCC and data channel. In the current implementation, the CCC is fixed and used by all nodes to coordinate data channel access and exchange relevant information. This feature is critical for a distributed network since it allows the sharing of information between neighbors. This information is then used by the cross-layer optimization algorithm at each node for distributed decision making. To prove the feasibility of using COmBAT, we implement three routing algorithms namely ROSA, RFA and RDA [3], [10].

ROSA. ROSA is a distributed joint routing and spectrum allocation algorithm that uses collaborative virtual sensing (CVS) to gather information from its neighbors to optimize the overall network throughput. This information is comprised of the results from spectrum sensing performed at the neighboring nodes and the instantaneous queue length of each of their active sessions which are used to calculate a predefined utility function by the nodes. During each time slot t, each node chooses the transmission parameters that maximizes the utility function. The detailed analysis of ROSA is discussed by authors in [3]. Due to lack of space, we eliminate these details but try to give an overall understanding of the utility function used by ROSA. Assuming a node i with neighbors $j \in N_i$, where N_i is the set of all neighbors of node i. The Shannon capacity of the link between node i to node j using frequency f is denoted as $C_{ij}(f)$. It is essential to understand that $C_{ij}(f)$ is dependent on transmit power $P_{ij}(f)$ used by the node i. The feasible choice of $P_{ij}(f)$ is constrained by two parameters referred to as $P_{min}(f)$ and $P_{max}(f)$. The $P_{min}(f)$ denotes the minimum transmit power that is required to ensure that the intended receiver achieves the required bit error rate (BER). Similarly, $P_{max}(f)$ represents the maximum transmit power that can be used on frequency f, such that the interference caused by this transmission on neighboring receiver nodes does not violate the BER constraint of those on going receptions. In other words, the interference caused, due to any new opportunistic transmission, should not disrupt any active communication. Accordingly, a spectrum opportunity is described as all frequencies f, such that $S(f) > 0$, where $S(f)$ is given by,

$$S(f) = P_{max}(f) - P_{min}(f). \tag{1}$$

Therefore, using CVS the node i determines the feasible transmit powers and calculates the achievable Shannon capacity $C_{ij}(f)$. Let the queue length of session s at node i be given by Q_i^s. Similarly, using CVS, node i gathers the queue length Q_j^s of the corresponding session s at the neighboring node j. The differential queue length along with the $C_{ij}(f)$ is used to calculate the utility function as follows,

$$U_{ij}^s = C_{ij}(f)(Q_i^s - Q_j^s). \tag{2}$$

Therefore, the overall objective of every AXL node using ROSA is to maximize its utility function by choosing the optimal session, next hop, transmit power and frequency spectrum.

RFA and RDA. To perform a comparative evaluation of ROSA, two other algorithms (RFA and RDA) are described in [3]. In RFA, the network uses a fixed predefined part of the RF spectrum but routes the packets through different nodes depending on the differential queue lengths (i.e. RFA chooses best next hop depending on differential queue length). Therefore, RFA does not consider changes in the achievable capacity which makes it susceptible to external interferes trying to disrupt the network. On the contrary, RDA performs DSA and allocates resources such that it maximizes the achievable

capacity but fails to exploit the spatial diversity using multipath routing. Instead, RDA uses a fixed shortest path route for any chosen destination. This indicates that the throughput of the network will suffer if any particular node experiences high congestion.

A. Experiment 1

In this first set of experiments, we evaluate how throughput of the network varies with increasing number of sessions. The CCC channel is set to be static and centered at 2.45 GHz. The data channel is divided into 16 sub-bands each with bandwidth of 500 KHz each. Therefore, the total spectrum available for data channel to use dynamically is 8 MHz, which extends from 2.440 GHz to 2.448 GHz. The bandwidth that can be used for the data channel are 500 KHz, 1 MHz, 1.5 MHz and 2 MHz. Both the channels use Gaussian minimum shift keying (GMSK) modulation. Each unique session in these experiments are defined with respect to its source-destination pairs.

Figure 5 shows how ROSA outperforms RDA and RFA as the number of sessions increases from one to ten. This is because ROSA jointly optimizes the routing and spectrum allocation thereby maximizing the utilization of available resources. RFA has the poorest performance because it has a fixed spectrum allocation and therefore cannot dynamically adapt to use different parts of the spectrum. Though RDA has the ability to use different portions of the spectrum, it is unable to exploit the advantages of multipath routes. This leads to bottlenecks in the network while alternate less busy routes are available to the intended destination. Therefore, in this section we have successfully implemented AXL, ROSA, RDA and RFA and utilized the concept of CVS which enable cross-layer optimization. Thus, we have experimentally corroborated the advantages put forth by authors in [3].

B. Experiment 2

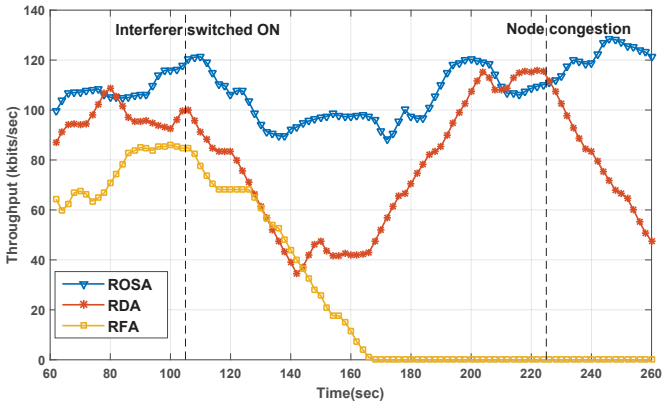

Fig. 6: Throughput performance in constrained scenarios.

In this section, we will evaluate how different algorithms adapt under the influence of interferer and traffic congestion. In this experiment, a session is initiated between node N_4 and N_6. The throughput at any instant t in Fig. 6 represents the throughput measured between $(t-60\,\text{s})$ to t s. The external inference is started at $t = 105$ s. As we can see in Fig. 6, all the algorithms are initially affected by the interferer but ROSA and RDA are able to recover because of their inherent dynamic spectrum access capabilities. The throughput of RFA deteriorates and eventually becomes zero. Next at $t = 225$ s, two sessions are initiated between N_1 to N_5 and N_7 to N_5 to increase congestion at node N_5 (these sessions do not contribute to the throughput). Figure. 6 shows that the performance of RDA deteriorates due to the congestion at N_5 as RDA is unable to exploit alternate routes to the destination (N_6). On the contrary, ROSA with its joint routing and spectrum allocation capability survives both these disruptions and continues to route packets to the destination.

IV. Conclusion

In this paper, we have introduced COmBAT that enables design, development and evaluation of next-generation cross-layer technologies. We have implemented three cross-layer based resource allocation algorithms to demonstrate the adaptability and validity of COmBAT. COmBAT is envisioned to be the state-of-the-art platform that facilitates transitioning of novel algorithms and protocols for multi-hop tactical ad-hoc networks. Therefore, we look forward to collaborate with the community to help them experimentally validate new cross-layer solutions and expedite the transition process.

References

[1] K. Liu and Q. Zhao, "A Restless Bandit Formulation of Opportunistic Access: Indexablity and Index Policy," in *Proc. of IEEE Annual Comm. Soc. Conf. on Sensor, Mesh and Ad Hoc Communications and Networks Workshops (SECON)*, June 2008.

[2] Y. Yuan, P. Bahl, R. Chandra, T. Moscibroda, and Y. Wu, "Allocating Dynamic Time-spectrum Blocks in Cognitive Radio Networks," in *Proc. of the ACM Intl. Symp. on Mobile Ad Hoc Networking and Computing (MobiHoc)*, September 2007.

[3] L. Ding, T. Melodia, S. Batalama, J. Matyjas, and M. Medley, "Cross-layer Routing and Dynamic Spectrum Allocation in Cognitive Radio Ad Hoc Networks," *IEEE Transactions on Vehicular Technology*, vol. 59, pp. 1969–1979, May 2010.

[4] J. Jagannath, T. Melodia, and A. Drozd, "DRS: Distributed Deadline-Based Joint Routing and Spectrum Allocation for Tactical Ad-hoc Networks," in *Proc. of IEEE Global Communications Conference (GLOBECOM)*, Washington, DC USA, December 2016.

[5] M. C. Ng, K. E. Fleming, M. Vutukuru, S. Gross, Arvind, and H. Balakrishnan, "Airblue: A System for Cross-layer Wireless Protocol Development," in *Proc. of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, October 2010.

[6] G. Nychis, T. Hottelier, Z. Yang, S. Seshan, and P. Steenkiste, "Enabling MAC Protocol Implementations on Software-defined Radios," in *Proc. of the USENIX Symposium on Networked Systems Design and Implementation*, April 2009.

[7] I. Tinnirello, G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, and F. Gringoli, "Wireless MAC processors: Programming MAC protocols on commodity Hardware," in *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, March 2012.

[8] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *ACM Transaction SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, March 2008.

[9] E. Demirors, G. Sklivanitis, T. Melodia, and S. N. Batalama, "RcUBe: Real-Time Reconfigurable Radio Framework with Self-Optimization Capabilitites," in *Proc. of IEEE Intl. Conf. on Sensing, Communication, and Networking (SECON)*, Seattle, WA, June 2015.

[10] A. L. Drozd, T. Arcuri, J. Jagannath, D. A. Pados, T. Melodia, E. Demirors, and G. Sklivanitis, "Network Throughput Improvement in Cognitive Networks by Joint Optimization of Spectrum Allocation and Cross-layer Routing," *Proc. of NATO Symposium on "Cognitive Radio and Future Network " (IST-123)*, May 2014.