DMDL: a Hierarchical Approach to Design, Visualize, and Implement MAC Protocols

Peng Wang
Institute for Networked Systems,
RWTH Aachen University,
D-52072, Aachen, Germany
Email: pwa@inets.rwth-aachen.de

Marina Petrova
COS RADIOSYSTEM LAB,
KTH Royal Institute of Technology,
S-16440, KISTA, Sweden
Email: petrovam@kth.se

Petri Mähönen
Institute for Networked Systems,
RWTH Aachen University,
D-52072, Aachen, Germany
Email: pma@inets.rwth-aachen.de

***Abstract*—This paper studies the modeling of MAC protocols that is comprised by hierarchical finite state machine with concurrency models. In the light of the model, a MAC description language named Decomposite MAC Description Language (DMDL) is proposed and open sourced to provide a unified tool for MAC protocol design and implementation. Following the modular design philosophy, the atomic elements of DMDL are the elementary MAC function components subdivided from a large-scale survey on different types of MAC protocols. A MAC protocol is built as a directed graph modeled by synchronous data flow with the MAC function components at the vertices. A token-like controlling unit is defined to pass signaling and data elements among MAC function components via the directed connections. The high decomposability and modularity enables DMDL to be a pure graphical language, and the concurrency nature of the synchronous data flow inherently ensures concurrency as a fundamental feature of DMDL. To show its capacity and performance, DMDL is developed on GNU Radio-USRP platform with a number of classic MAC protocols. The performances of the protocols are in accord with the theoretical expectations, which proves that DMDL is an efficient MAC designing and implementing tool with high flexibility and reusability.**

I. Introduction

Due to the rapidly increasing demands on wireless communications and new emerged technologies, new protocols will be designed, tested, and standardized much faster than ever before. The modular design principle has been widely accepted due to its high flexibility and reconfigurability [1], [2], [3], [4]. First, modularity decouples the whole MAC protocols into independent components, which simplifies the developing procedure. Second, it enables fast reconfiguration and updating the MAC design during the implementation and optimization which may cost a huge amount of time and resources in conventional integrated design. Third, modular design also enables MAC protocol visualization because its inherent graphical nature. Many MAC designing tools [2], [3], [5], [6] are aimed at providing graphical environment for MAC design, implementation, and even execution. In contrast to the traditional text based environments, the visualization makes the protocol design easier and more straightforward. Even through the module based MAC design has widely used in many years, to the best of our knowledge, a comprehensive and detailed analysis on how to model a MAC protocol and how to optimize MAC decomposition that are still missing.

In this paper, a MAC description language, Decomposite MAC Description Language (DMDL) is proposed. DMDL is a new cross-platform architecture that enables flexible and reconfigurable MAC protocols design and implementation, in particular for fast prototyping MAC protocols on Software Defined Radio (SDR) platforms. MAC protocols in DMDL are modeled as a Hierarchical Finite State Machine (HFSM), in which the states are further refined to connected *actors* of Synchronous Data Flow (SDF) model [7], [8]. The inherent feature of SDF on concurrency can improve the capacity of computation of HFSM which is good at sequential logic control. The modular design principle provides high reusability and flexibility to DMDL so that MAC protocols can be easily built by connecting decomposed MAC modules as a directed graph. DMDL is aimed at becoming an instructive standard which can be used directly as a library for designing only and as a guidance for implementing MAC protocols on hardware based platforms.

The rest of the paper is organized as follow. The existing module based MAC tools and implementations are discussed in Section II. As a summary of existing works, the choices of HFSM model is introduced in Section III. The architecture of the system and decomposition of MAC components are introduced in Section IV. GNU Radio and USRPs are briefly introduced in Section V. Examples of MAC protocols built by DMDL are demonstrated and evaluated in Section **??**. Conclusions are drawn in Section VII.

II. Related Work

Following the modular design principle, two main branches exist of how to divide MAC protocols and define the components. The first approach models a MAC protocol as an FSM and the components are the logic state of the FSM. For instance, Wireless MAC Processor (WMP) [3] defines its the components as states of extended FSM such as `IDLE` or `RX`. Actions take places in state transitions are driven by events. Other forms of FSM also widely used in modeling MACs [9], [10]. The FSM approach has several benefits. First, FSM schedules and executes intricate sequential control logics better than other sequential control model such as general if-else and goto. Second, it not only simplifies the representation of MACs because its well developed visualization such as statechart

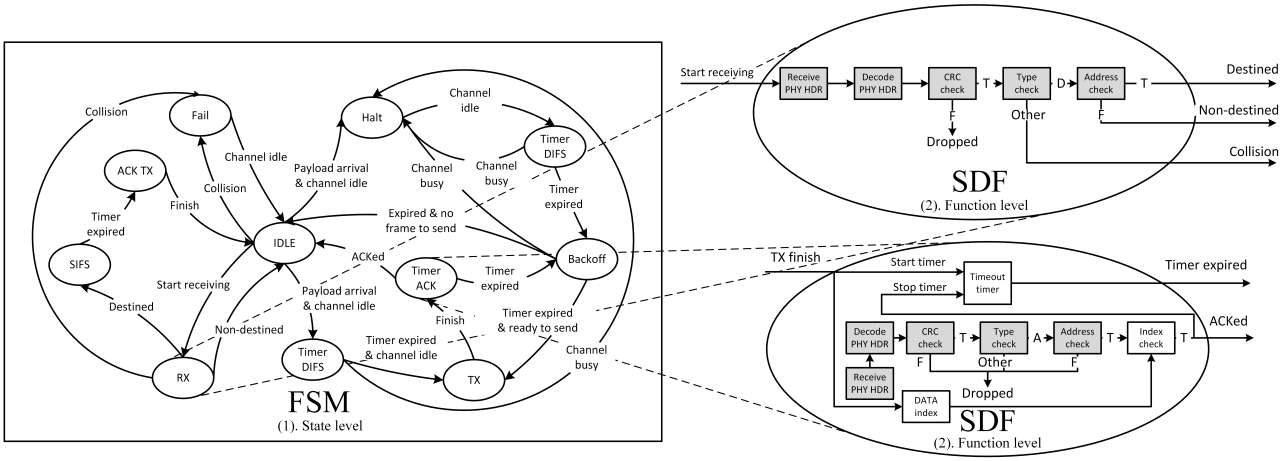


Fig. 1: Hierarchical structure of models of the classic CSMA/CA used in the IEEE 802.11 DCF.

[11], but also enables analysis and synthesis due to its strict mathematical definitions and finite nature respectively.

Another approach of using modular design principle in developing MAC protocols follows the data flow concepts. The data flow approach first create a function set including fundamental MAC functionalities by decomposing numerous MAC protocols and abstracting their similarities. After this tedious and complicated work, building new MAC protocols can be simply performed by reconnecting the well defined MAC components together. By well designing the ports and connections between the elementary function blocks, the flexibility and reconfigurability can be significantly increased compared with conventional integrated MAC development. Some works [2], [1] decomposed the MAC into such components and build MAC protocols by connecting them. The advantage of this approach is, if the decomposition is complete and appropriate, the set of MAC functionalities are supposed to be able to compose most of the MAC protocols with out extra work.

III. Modeling

To enable a comprehensive comparison over both the FSM approach and data flow approach, both the FSM approach and data flow approach are modeled following the HFSM model [8], which supports co-existence of various concurrency models including the data flow hierarchically. Among all supposed concurrency models, the Synchronous Data Flow (SDF) model well addresses the demands of modeling MAC functionalities thus used in this study to model the data flow type of MAC implementation. An example of the classic IEEE 802.11 DCF is modeled as a HFSM shown in Figure 1, where the FSM approach and the SDF approach are named state level and the function level, respectively. Although the implementations in these two levels share quite amount of similarities, there are still several key differences. The state level focuses on what the protocol should behave, and the node level concerns more about how to do. The states in the state level FSM contains 11 distinguish states without reuse any of them. Some of the states, e.g., the `ACK TX` and `TX`, share considerable amount similarity, however, they are not interchangeable. On contrary, the blocks in the function level achieve much higher reusability. For example, all function level blocks which are refined in the state `RX` can be directly used by the state `Timer ACK` without any modification. Moreover, the flexibility of the MAC implementation in function level is also higher than the FSM level. For example, if the user want to the `RX` state to also export the failed frame header, the developer has to revise the source code of the states. However, in a function level realization, users can simply read that value after the block `CRC check` and export the existing value from port `Dropped`. By careful decompose MAC functionalities and define interface, i.e., ports of blocks, the function level decomposition can achieve the highest reusability and balance flexibility and complexity.

IV. Architecture

In the light of the above modeling analysis, DMDL defines MAC protocols as modular based messaging driven reactive system. SDF is chosen for MAC decomposition, while behaviors of the system is still able to be abstracted as a FSM in the higher level which is hidden behind the visualized implementation. The atomic processing element is fine-decomposed MAC component that represents a elementary function in MAC layer, e.g., framing, or MAC related functionalities, e.g., sending and receiving that are MAC/PHY interfaces. Due to the module principle and the statechart gene, the MAC component in DMDL is named block which is the semantic element of DMDL. DMDL has a well-defined set of blocks obtained by analyzing and decomposing a number of different types of MAC protocols. Users can build their own protocol by connecting blocks as a directed graph, where blocks are put on the vertices. In DMDL, blocks are triggered to work by receiving a control token from previous blocks through edges. The control token of DMDL contains the key information of the protocol such as the MAC frames and the destination address. Because blocks are activated by the arrival control token and operate according to the content of the control token,

DMDL calls the control unit *command*. In DMDL graph, the *command* is moved from block to block. The topology of the directed graph represents the executing order of all connected blocks. Due to the length limitation, the complete introduction containing all blocks is not given in this paper. The complete block and port library with all source code can be found in the online repository of DMDL [12].

A. Blocks

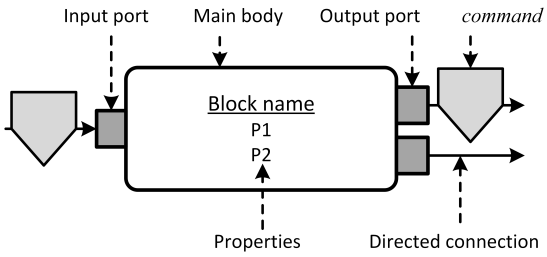


Fig. 2: A DMDL block.

A sample block is shown in Figure 2 which contains all key elements of a block. The detailed introduction to these elements are,

- **Main body** contains the logic of the MAC function the block represents. The main body may also inserts new items into the *command* or modifies existing ones. The block name reveals at least part of the main function.
- **Properties** set the parameters of the MAC function which is defined when the MAC protocol is built, e.g., the length of the MAC header and the timeout duration. The properties can be changed in the run time by the internal logics of the block or the input *command*.
- **Port** is the interface between blocks. According to the SDF model, a block should have fixed number of ports. A block can have any number of input ports and output ports, but a block should have at least one so that it can attach to the directed graph. *command* is always passed from output ports to input ports. An output port can connect to multiple input ports, and vise versa. Both the arrival and departure of *command* in input ports and output ports can be asynchronous, i.e., *command* can arrive at different input ports at different time. *command* can also be passed from the output ports to the input ports of the same block.
- **Directed connection** is the edge of the directed graph. It is the route of *command* thus it can only point the input ports from the output ports.

B. command

In DMDL, *command* is a token type of control signal used to pass the information between blocks which is also shown in Figure 2. A typical *command* may contain a MAC frame which is the main data object of MAC protocols, and several parameters related to the MAC frame, e.g., the address of the MAC frame and the PHY parameters in transmitting the frame. Blocks are supposed to generate or modify *command* according to their functions. For example, the block to check the CRC of the received frame should add an item of the result of the CRC check into the input *command* and export the new *command* out so that can be used by the later part of the MAC protocols.

C. MAC Decomposition

Due to different features and purposes, blocks defined in DMDL can be catagorized into timed blocks, frame-related blocks, and medium-related blocks. Auxiliary blocks is also defined in DMDL to cooperate with other blocks, for example the blocks for logging system intermediate status and perform scientific analysis, which may be useful in DMDL implementations. Currently, the DMDL library contains 76 blocks which can support many classic MAC protocols such as different types of CSMA protocol including the IEEE 802.11 DCF with RTS/CTS and virtual carrier sensing, different types of TDMA protocols, and slide window protocols. Users are also able to fully customize their own MACs, e.g., CogMAC [13] and CogMAC+ [14], [15] have been implemented. All examples are placed in the *examples* folder of the github repository.

It is worth noting that DMDL supports open semantics that allow developers to add third party blocks into the DMDL library. Due to the emerging technologies and new MAC algorithms, new blocks can be defined and implemented to expend the scope and usage of DMDL. For example, new blocks worked for directional MAC protocols are being developed at the time of writing by other researchers in the institute.

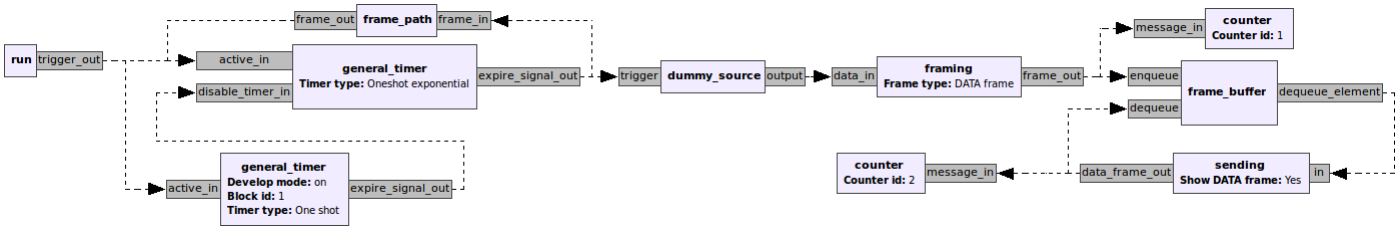
V. GNU Radio-USRP Implementation

In order to verify the feasibility of DMDL and have a comprehensive evaluation on its performance, a GNU Radio-USRP platform based DMDL implementation is developed. GNU Radio is one of the most popular SDR developing environment that can setup and configure signal processing in a graphical way. It has a powerful set of built-in signal processing blocks which provides very high PHY layer reconfigurability. Combining with USRP, users are easily to test and prototype their new ideas. Users are also enabled to add their own blocks into GNU Radio as out-of-tree modules, which provides an very friendly environment for realizing DMDL. All blocks and examples are open sourced on our GitHub repository [12].

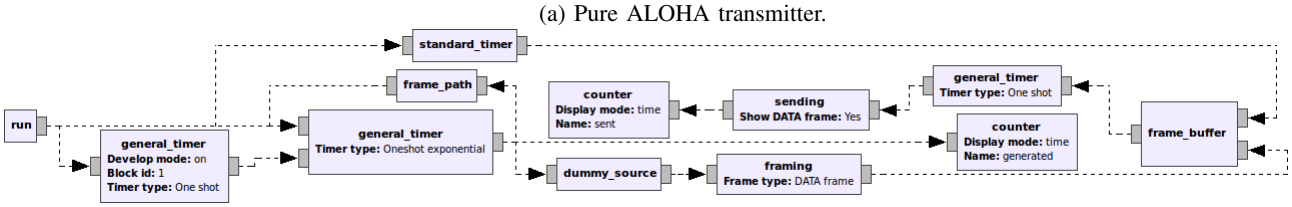
VI. Performance Evaluation on Classic MAC Protocols

In this section, several classic MAC protocols are implemented using DMDL on GNU Radio-USRP platform [16], [6]. Several long term measurements are carried out to compared with the theoretical results.

Several key parameters used in the measurements are listed in Table I. The protocol specified parameters are defined in the corresponding subsections.



(a) Pure ALOHA transmitter.



(b) Slotted ALOHA transmitter.

Fig. 3: The GRC code of transmitters used in ALOHA measurements.

A. Pure and Slotted ALOHA

Pure ALOHA is one of the simplest MAC protocols. It was first developed at the University of Hawaii to provide packetized wireless communication services at Hawaii area where cable connecting all islands is uneconomic. According to the types of synchronization, the ALOHA series protocols can be further categorized into pure ALOHA and slotted ALOHA. In pure ALOHA, the rules is quite simple that nodes send data if they get data to send. In slotted ALOHA, nodes are only allowed to send data in the beginning of a time slot. Due to their simplicity and contention based channel access, pure ALOHA and slotted ALOHA have been well studied and widely used. Therefore, they are also implemented and measured in DMDL to evaluate the performance of DMDL on GNU Radio-USRP platform. The measurements for both protocols include two transmitting nodes and one receiving nodes. The structure of the receiving node is relatively simple, so only the transmitter design are shown in Figure 3. In the measurement, two transmitting nodes are equipped with Poisson traffic source which is one of the most widely used source type to simulate the real traffic. Unlike the well know theoretical analysis in [17] where queueing is not considered, the ALOHA nodes in this measurement are equipped with a transmission buffer as shown in Figure 3a. In contrast to the theoretical model, the new model with buffer is more closed to the real system used in real communication devices.

Parameters	Value
Payload size	837 bytes
Frame size	1000 bytes
Zero padding before and after frame	64 bytes
Frame transmission duration	40 ms
Sampling rate	400 k/s
Carrier frequency	430 MHz

TABLE I: Key parameters used classic MAC protocols evaluation.

The transmitting node can be modeled as a queueing system. According to the analysis in [18], the transmitting node and the channel compose a classic M/D/1 system. Therefore, the arrive rate λ and the service rate μ of the system is the average number of generated frames in time unit and the reciprocal of the frame transmission duration, i.e., $1/T_{\text{fr}}$, respectively. Two conditions have to be satisfied so that one frame can be delivered without collision. First, when the frame is sent out, the queue of another node should be empty. It is due to the fact that if the queue of the another node is not empty, there must be an ongoing transmission, or a frame is going to be sent immediately. Second, during the transmission duration of the frame, no frame should be arrived at the queue of another node. Obviously, it is a necessity for a successful transmission that the traffic load is less than one, otherwise frames are accumulated in the queue. Therefore,only the scenario that $\frac{\lambda}{\mu} < 1$ is considered. As analyzed above, the probability of a successful transmission is,

$$P_{\text{success}} = P_0 P_{0\text{fr}}, \tag{1}$$

where P_0 denotes the probability of empty buffer, and $P_{0\text{fr}}$ denotes the probability of no frame is generated in a frame transmission time. In order to obtain the former probability, the probability of n frames in the buffer is

$$P_n = \frac{\lambda}{\mu} P_n + \left(P_{n-1} - \frac{\lambda}{\mu} P_{n-2} \right) = \left(\frac{\lambda}{\mu} \right)^n P_0. \tag{2}$$

Therefore, the sum of all probabilities from n equals to 0 to ∞ should be one, thus P_0 is

$$P_0 = 1 - \frac{\lambda}{\mu}. \tag{3}$$

Notice that (3) only stands if $\frac{\lambda}{\mu}$ is less then 0. As discussed before, it is the necessary condition of having successful transmission, so (3) always true in the range of parameters used in this measurements. $P_{0\text{fr}}$ in (1) stands for the probability

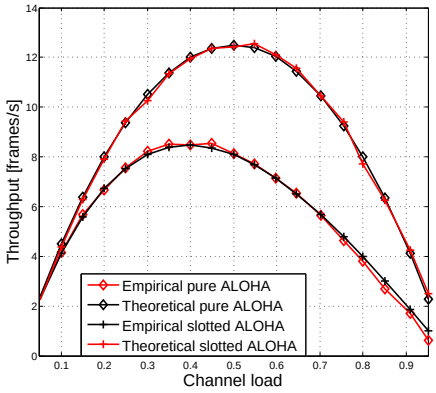


Fig. 4: ALOHA throughputs.

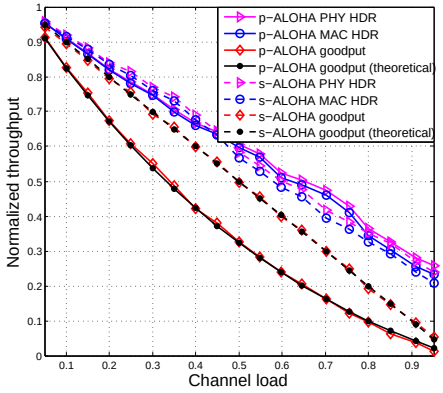


Fig. 5: Normalized ALOHA throughputs.

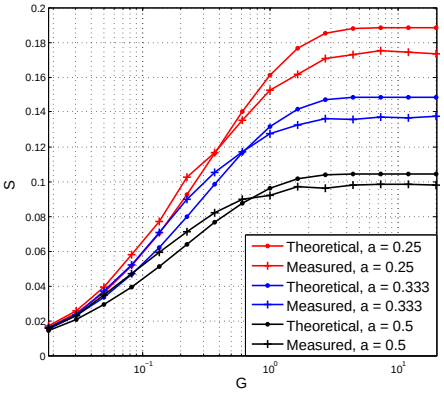


Fig. 6: p-persistent CSMA throughputs.

of no new frame arrives in the buffer of another transmitting node when the frame is being transmitted. According to the Poisson distribution, the probability is,

$$P_{0\text{fr}} = \frac{e^{-\frac{\lambda}{\tilde{\mu}}}(\frac{\lambda}{\tilde{\mu}})^0}{0!}, \tag{4}$$

where $\tilde{\mu}$ is,

$$\tilde{\mu} = \mu \frac{L_{\text{packet}} - L_{\text{padding}}}{L_{\text{packet}}}. \tag{5}$$

The zero padding length L_{padding} is included in the frame transmission but not occupying the channel since the signal strength is zero. To take this into account, μ is replaced by $\tilde{\mu}$ in calculating the $P_{0\text{fr}}$. This way, the improved probability of successful transmission is,

$$P_{\text{success}} = \left(1 - \frac{\lambda}{\mu}\right) e^{-\frac{\lambda L_{\text{packet}}}{\mu(L_{\text{packet}} - L_{\text{padding}})}}, \tag{6}$$

In slotted ALOHA, the deduction of theoretical throughput is slightly different from the pure ALOHA. Because both ALOHA use the same Poisson source, P_0 in slotted ALOHA is unchanged. However, the $P_{0\text{fr}}$ is changed to 1. It is due to the fact that in slotted ALOHA, if there is no frame is waiting in the queue in the beginning of a time slot, then there will be no frame transmitted out in the whole time slot. Thus, the successful transmission probability in a slotted ALOHA network with two transmitting nodes is,

$$P_{\text{success}} = 1 - \frac{\lambda}{\mu}. \tag{7}$$

Because new generated frame in a time slot has to wait for the next time slot to be sent, so the μ can be directly used here without considering the effects of zero padding. In the experiment shown in Figure 3, the transmission time of one frame is 40 ms which is the reciprocal of μ. The mean waiting time of the Poisson source are set from 800 ms to 42.1ms so that the channel loads in these measurements are varied from 0.05 to 0.95. The throughput of the two ALOHA protocols are shown in Figure 4 and Figure 5. In Figure 4, the black curves are the theoretical throughputs and the red lines present the measured results. For pure ALOHA implementation, the correlation coefficient of measured results and the theoretical results is 0.9992 and the Mean Square Error (MSE) of the measured data is 0.022 which is almost negligible in normal situations. For slotted CSMA, the correlation coefficient between empirical results and theoretical expectations is 0.9992, and the MSE is only 0.017. The results show that the ALOHA protocols implemented by DMDL on GNU Radio-USRP platform can achieve very high accordance with the theoretical expectations. Moreover, it verifies that the timing performance and synchronization of DMDL is accurate.

Figure 4 shows the normalized results of the same measurements with more intermediate results, which are recorded by DMDL auxiliary blocks. All measured data are normalized by the number of transmitted frames at both transmitting node. The dashed lines represent the results of slotted ALOHA and the solid lines plot the pure ALOHA performance. Similar with Figure 5, the improved theoretical results are almost overlapped with the measured data which shows the accuracy of the DMDL implementation. The curve *PHY headers* records the total number of correctly received PHY headers, and the curve *MAC headers* records the number of correctly received MAC headers. The gaps between them caused by the collided MAC headers, which relatively small due to their limited size.

B. p-persistent CSMA

In p-persistent CSMA, when a node is ready to transmit data, it senses the channel to detect the channel status. If the channel is idle, it transmits a frame with probability p. Otherwise, it continuously senses the channel until it turns to idle, and then send the frame with the same probability p. In contrast to the slotted ALOHA that time slot is equivalent to the frame transmission duration, a frame in p-persistent CSMA may contain multiple time slots. Nodes can only send a frame in the beginning of a time slot. If the node does not transmit, it waits until the next available time slot.

$$S = \frac{2p\sum_{k=0}^{\infty}\left((1-p)^k - (1-g)^{1+\frac{1}{a}}\left[\frac{p(1-p)^k - g(1-g)^k}{p-g}\right]\right)\left((1-p)^{k+1} - (1-g)^{1+\frac{1}{a}}p\left[\frac{(1-p)^{k+1} - (1-g)^{k+1}}{p-g}\right]\right)}{1 + a + a\sum_{k=1}^{\infty}\left((1-p)^k - (1-g)^{1+\frac{1}{a}}p\left[\frac{(1-p)^k - (1-g)^k}{p-g}\right]\right)}, \tag{8}$$

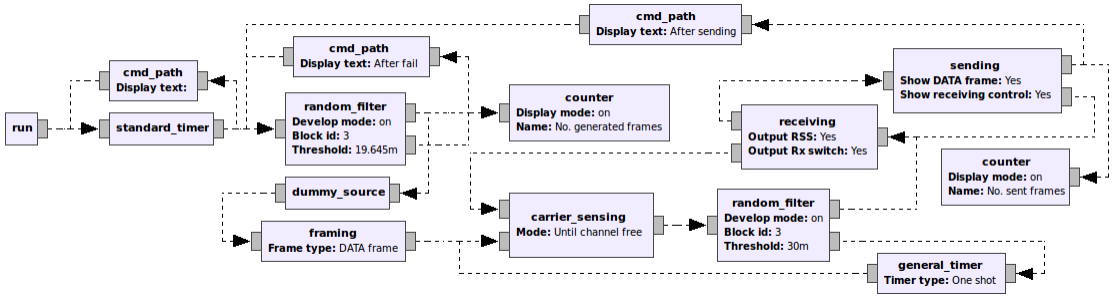


Fig. 7: The transmitter design used in slotted p-persistent CSMA measurement.

The performance of slotted p-persistent CSMA has been well studied. According to the discussion in [19], the throughput of p-persistent CSMA with two transmitting nodes is,

where p is the transmission probability at the beginning of a time slot which after an idle time slot, g is the frame generation probability at the beginning of a time slot in which no frame is being sent, and a is the propagation delay which is also defined as the slot time in terms of frame transmission time. Due to the relatively short distance in laboratory environment, a is set to 4 ms. The design of the CSMA transmitter is shown in Figure 7.

The experimental results are plotted in Figure 6 together with the theoretical values from (8). To ensure $\frac{1}{a}$ is an integer, a is set to 0.25, 0.333, and 0.5, p is set to 0.03 and $G = \min(1, \frac{gM}{a})$, where g is the same frame generation probability per time slot. In Figure 6, the curves of experimental performance has the same trend as the curves of theoretical results with varying channel load G. The results show that the implementation of p-persistent CSMA in DMDL is able to achieve high accordance with theory.

VII. Conclusion

In this paper, the concept and implementation of DMDL are proposed and evaluated. MAC protocols are model by HFSM and then decomposed as elementary blocks. With the help of the DMDL token *command* to convey data and control information among blocks, DMDL can build MAC protocols simply by connecting blocks with directed connections. DMDL has been developed on GNU Radio-USRP platform and has the potential to be applied on more hardware based platforms or software simulators. DMDL is an open sourced project including a fully functional but also expendable block and port library, source code of its GNU Radio-USRP implementation and many implementations of classic MAC protocols as examples. Its fast and straightforward graphical semantics provide DMDL high flexibility, reconfigurability and reusability. The comparisons of empirical results and theoretical expectations show that DMDL implementation on GNU Radio - USRP platform achieves significantly high level of accordance with the theoretical expectations which make it a powerful tool for future protocol design and prototyping.

References

[1] G. Y. X. Zhang, J. Ansari and P. Mähönen, "Trump: Supporting Efficient Realization of Protocols for Cognitive Radio Networks,"

[2] J. Ansari, X. Zhang, A. Achtzehn, M. Petrova and P. Mähönen, "A flexible MAC development framework for cognitive radio systems," in *Proceedings of the IEEE Wireless Communications and Networking Conference*, (Cancun, Mexico), 2011.

[3] I. Tinnirello, G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, and F. Gringoli, "Wireless MAC processors: Programming MAC protocols on commodity Hardware," in *Proceedings of IEEE International Conference on Computer Communications and Networks*, (Orlando, FL, USA), 2012.

[4] G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, F. Gringoli and I. Tinnirello, "MAClets: Active MAC Protocols over Hard-Coded Devices," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pp. 229–240, ACM, 2012.

[5] National Instruments, "LabVIEW, http://www.ni.com/en-us/shop/labview.html, Last Visited: 10.05.2017."

[6] "GNU Radio, http://www.gnu.org/software/gnuradio, Last Visited: 01.04.2017."

[7] E.A. Lee, *A denotational semantics for dataflow with firing*. Electronics Research Laboratory, College of Engineering, University of California, 1997.

[8] A. Girault, B. Lee and E.A. Lee, "Hierarchical Finite State Machines with Multiple Concurrency Models," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 18, no. 6, pp. 742–760, 1999.

[9] F. G. P. Gallo and I. Tinnirello, "On the Flexibility of the IEEE 802.11 Technology: Challenges and Directions,"

[10] L, Yin, T.S. Hong, C.X. Liu, X. Yue and H.E. Z, "A reprogamming protocol based on state machine for wireless sensor network," in *Proceedings of 2010 International Conference on Electrical and Control Engineering (ICECE)*, pp. 232–235, IEEE, 2010.

[11] D. Harel, "Statecharts: A Visual Formalism for Complex Systems," *Science of computer programming*, vol. 8, no. 3, pp. 231–274, 1987.

[12] P, Wang, "Decomposite MAC Description Language, https://github.com/joqoko/gr-inets, Last Visited: 17.09.2017."

[13] J. Ansari, X. Zhang and P. Mähönen, "A Decentralized MAC Protocol for Opportunistic Spectrum Access in Cognitive Wireless Networks," *Elsevier Journal of Computer Communications*, vol. 36, no. 13, pp. 1399 – 1410, 2013.

[14] P. Wang, J. Ansari, M. Petrova, and P. Mähönen, "Demo: CogMAC+ - A Decentralized Multichannel MAC Protocol for Cognitive Wireless Networks," in *Proceedings of the 2014 ACM workshop on Software radio implementation forum*, (Chicago, USA), 2014.

[15] P. Wang, J. Absari, M. Petrova, and P. Mähönen, "CogMAC+: A Decentralized MAC Protocol for Opportunistic Spectrum Access in Cognitive Wireless Networks," *Elsevier Journal Computer Communications*, vol. 79, pp. 22–36, 2016.

[16] Ettus Research, "Universal Software Radio Peripheral (USRP), http://www.ettus.com, Last Visited: 06.07.2015."

[17] A. Tanenbaum, and D. Wetherall, *Computer Networks*. Pearson, 2011.

[18] S.M. Ross, *Introduction to Probability Models*. Academic press, 2014.

[19] H. Takagi and L. Kleinrock, "Throughput Analysis for Persistent CSMA Systems," *IEEE transactions on communications*, vol. 33, no. 7, pp. 627–638, 1985.