A CC2420 Transceiver Simulation Module for ns-3 and its Integration into the FERAL Simulator Framework

Anuschka Igel and Reinhard Gotzhein
Networked Systems Group
University of Kaiserslautern, Germany
{igel,gotzhein}@cs.uni-kl.de

***Abstract*—Simulation is a common approach to assess the functional and non-functional behavior of protocols in wireless sensor networks. In these networks, the CC2420 transceiver is a frequently used communication platform. To make this platform available for simulation purposes, we have developed a CC2420 simulation module for ns-3, a well-known discrete-event network simulator targeted primarily for research and educational use, using an existing CC2420 module for its predecessor ns-2 as starting point. In this paper, we report on this development and, in particular, several functional enhancements of the existing CC2420 module. Furthermore, we present the integration of ns-3 and the CC2420 module into FERAL, a generic simulator framework for the rapid coupling of diverse simulators. Finally, we present results of simulation experiments where we have used the CC2420 module, both stand-alone ns-3 simulations and simulations where ns-3 is a simulator component of the FERAL framework. These experiments show that the CC2420 simulation module is fully operational in the ns-3 context, and that the integration into FERAL provides additional degrees of freedom especially in the early development stages, where abstract models, e. g., Simulink or SDL models, are used to specify system behavior.**

Keywords*-*CC2420 simulation module; ns-3; net device; simulator framework

I. Introduction

Nowadays, Mobile Ad-hoc NETworks (MANETs) consisting of wireless sensor nodes become more and more important. Common application areas are the collection of environmental data in inaccessible areas or health monitoring. These networks are characterized by a lack of fixed infrastructure. Due to node mobility, they are usually restricted concerning power supply. Therefore, the use of energy-efficient hardware is crucial. A common transceiver module used for such nodes is Texas-Instruments' CC2420 transceiver [1], which is compliant with the IEEE 802.15.4 standard [2]. This standard defines wireless transmissions in low-cost networks, with devices that have a low data rate and low power.

Protocols running on nodes in MANETs can be quite complex and therefore should be evaluated before deployment. Since testbeds are expensive and time-consuming to build, simulations are often used to verify functional as well as non-functional behavior of these protocols. Therefore, simulation capabilities for MANETs whose nodes communicate via CC2420 transceivers are desirable. Network simulators suitable for MANETs already exist, including the well-known network simulator 2 (ns-2) [3] and its successor, the network simulator 3 (ns-3) [4]. A CC2420 module for ns-2 has been developed in [5] and integrated into the simulator C-PartsSim (see also [6]); however, ns-2 is no longer actively developed and has several drawbacks compared to ns-3 (see [7]). For example, ns-2 uses a combination of C++ and the Tool Command Language (TCL), while ns-3 is written entirely in C++. Besides, the existing CC2420 module does not realize important features (e. g., changing certain settings of the transceiver). Therefore, we have developed a CC2420 simulation module with several functional enhancements for ns-3, taking the existing module for ns-2 as a starting point. The decision for ns-3 was especially taken because of its modularity and clean design, which allows using many of the existing simulation components – including applications, protocol implementations, and mobility, loss and delay models – together with the CC2420 simulation module.

In addition to this, we have integrated ns-3 and the CC2420 module (and other ns-3 components) into the Framework for the Efficient simulator coupling on Requirements and Architecture Level (FERAL) [8], thus drawing benefit from using it in combination with other simulators such as Simulink. In this paper, we will use a simulation component for the internationally standardized Specification and Description Language (SDL) [9] to simulate the behavior of nodes, while ns-3 and the CC2420 module are used to simulate the behavior of the medium.

The remainder of this paper is structured as follows: In Section II, we survey related work. In Section III, we describe the development and enhancement of the CC2420 simulation module. Section IV reports on the integration of ns-3 and the CC2420 module into the simulator framework FERAL. Section V presents results of simulation experiments using the CC2420 module. In Section VI, we draw conclusions and elaborate on future work.

II. Related Work

Related work can be divided into two categories, namely simulation approaches for the CC2420 transceiver and the development of simulation modules for ns-3. In this paper, we combine these two aspects and report on the development of a CC2420 simulation module for ns-3.

Several simulation approaches for the CC2420 transceiver are described in the literature. The authors of [10] extended the TinyOS SIMulator (TOSSIM) by an improved wireless propagation model and a radio frequency physical stack based on the CC2420 transceiver. TinyOS is a popular operating system for wireless sensor networks. Its source code can be directly utilized for TOSSIM, which also considers operating system overhead. Amongst other features, the proposed CC2420 model uses Clear Channel Assessment (CCA), measures the received signal strength and allows configuration of transmission power and channel. The results obtained with this simulator are independent of a concrete processor, but are bound to the TinyOS operating system. Therefore, it cannot be used to evaluate protocols independently of a concrete operating system.

In [11], AvroraZ is presented, which extends the Avrora simulator by a module for simulating the CC2420 transceiver. Avrora is a cycle-accurate instruction-level simulator for AVR microcontrollers. The simulation module includes address recognition, frame acknowledgement, CCA, and several other features. A similar work is presented in [12], where an instruction-level sensor network simulator using a detailed CC2420 simulation model is introduced. This simulator provides a cycle-accurate processor emulation of the ATmega128, which is independent of the operating system, and also models the internal structure of the CC2420 transceiver, e. g., registers and main memory. However, the paper does not provide much detail about the CC2420 implementation. Both of these approaches provide instruction-level simulators, which are not suitable for large networks, because simulations are very time-consuming. Furthermore, these simulators are tied to special processors and are therefore not usable for the generic evaluation of protocols on higher layers.

Numerous simulation modules for ns-3 have been developed by third parties. Among these are routing protocols, e. g., the Ad-hoc On-Demand Distance Vector (AODV) protocol, which was implemented for ns-3 in [13]. A module for the Destination-Sequenced Distance Vector (DSDV) routing protocol was introduced in [14], while [15] presents an IPv6 stack for ns-3. Besides this, an ns-3 framework usable for spectrum-aware simulations was described in [16]. The authors implemented spectrum-aware channel and physical layer models, which provides the possibility to analyze how the performance of protocols on higher layers is affected by the frequency-related aspects of communication on the physical layer.

III. CC2420 Transceiver Simulation Module

The CC2420 transceiver developed by TexasInstruments is a 2.4 GHz IEEE 802.15.4 compliant transceiver with a data rate of up to 250 kbps and 16 channels [1]. It is low-cost, configurable, energy-efficient and works in the unlicensed ISM band. Therefore, it can be used to build up sensor networks.

A. Functionality

The CC2420 simulation module adopts the state machine described in the data sheet [1]. We abstract from aspects like resetting and switching the transceiver on and off as well as the acknowledgement mechanism and overflow respectively

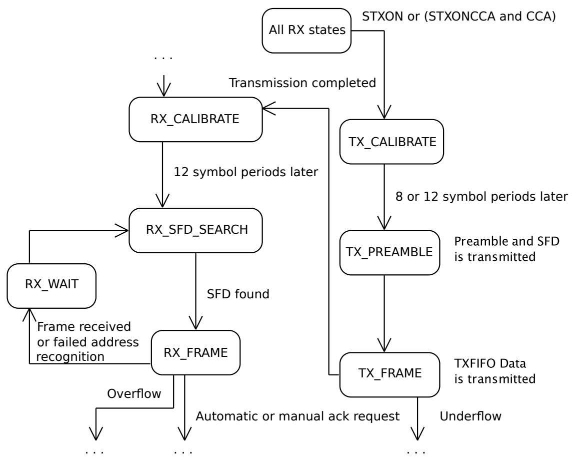
All RX states
STXON or (STXONCCA and CCA)
...
Transmission completed
RX_CALIBRATE
TX_CALIBRATE
12 symbol periods later
8 or 12 symbol periods later
RX_SFD_SEARCH
TX_PREAMBLE
Preamble and SFD is transmitted
RX_WAIT
SFD found
Frame received or failed address recognition
RX_FRAME
TX_FRAME
TXFIFO Data is transmitted
Overflow
Automatic or manual ack request
Underflow
...
...
...

Figure 1. Simulated part of the CC2420 transceiver [1].

underflow detection. Figure 1 shows an excerpt of the CC2420 state machine used for the simulation module.

The transceiver starts in state RX_CALIBRATE. This phase has a duration of 12 symbol periods, after which the state RX_SFD_SEARCH is entered. Since each symbol encodes 4 bits and the data rate of the transceiver is 250 kbps, one symbol period has a duration of 16 μs. The transceiver uses a synchronization header for symbol synchronization of a received frame. This synchronization header consists of a preamble sequence and a so-called Start of Frame Delimiter (SFD) (the frame format is depicted in Figure 4). The default preamble sequence consists of 4 bytes with value 0x00, while SFD is 0xA7, both compliant with the IEEE 802.15.4 standard [2]. The so-called sync word consists of the last preamble byte (which should be zero for compliance with IEEE 802.15.4) and the SFD byte. Preamble length (i. e., the number of leading zero bytes) and sync word can be configured via special registers, but changing the default values makes the transceiver non-compliant with IEEE 802.15.4. When receiving a frame, the transceiver synchronizes to the zero-symbols and searches for the SFD sequence (the preamble length does not matter here; it is only relevant for sending). The reception of an SFD causes the transceiver to switch to the state RX_FRAME. As soon as the frame is received completely, the state RX_WAIT is taken. When the transceiver is ready to receive another frame, it goes again to the state RX_SFD_SEARCH.

Transmission requests are allowed in all RX states. It is possible to transmit with clear channel assessment (signal STXONCCA) or without (signal STXON). [2] defines three different CCA modes, which are all supported by the CC2420 transceiver. Mode 1 means that the channel is clear when the received signal strength (energy on the medium) is below a programmable threshold. Mode 2 signals a clear channel when the transceiver does not receive valid IEEE 802.15.4 data. Mode 3 is a combination of the modes 1 and 2. Furthermore, a hysteresis is defined, which has the purpose of avoiding too frequent CCA changes in modes 1 and 3. By signaling a clear channel only when the energy falls below threshold minus hysteresis, minimal fluctuations do not lead to CCA changes.

A transmission request brings the transceiver to the state TX_CALIBRATE. This phase has a duration of 8 or 12 symbol periods. A parameter named *TX_TURNAROUND* defines which one is used. A duration of 12 symbol periods is compliant with IEEE 802.15.4. Afterwards, preamble and SFD are transmitted (state TX_PREAMBLE), and finally the rest of the frame, taken from a special FIFO memory (state TX_FRAME). After transmitting the frame, the transceiver goes to state RX_CALIBRATE.

B. *Integration into ns-3*

The simulator ns-3 [4] is a discrete-event network simulator usable for, but not limited to, Internet systems. It is the successor of the well-known ns-2 [3], which is still used in academic research, but is no longer actively developed and has several drawbacks concerning its design.

Figure 2 shows the overall structure of an ns-3 simulation. Applications (for generating and processing traffic), protocol stacks (e. g., UDP / IP), and net devices (which provide Medium Access Control (MAC) functionality and define an interface for the network layer to access a physical device) are installed on nodes. The protocol stack can also be omitted, since applications can be defined in such a way that they communicate directly with net devices. The nodes are then connected by channels. Mobility models can be installed on wireless nodes, determining the positions and movements of these nodes. Besides this, ns-3 provides several loss and delay models, which can be attached to (wireless) channels.

The actual simulation is driven by events, which are delivered to a scheduler. Initially, such events are created by applications running on the nodes. Further events are either created by applications as well or result from the simulation flow (e. g., a send event triggers a receive event at nodes in range).

The part of a simulation system covered by the CC2420 simulation module is marked in Figure 2. By developing the module as a part of ns-3, one can benefit from existing ns-3 components. Existing propagation loss and delay models are used for the CC2420 channel, and predefined mobility models

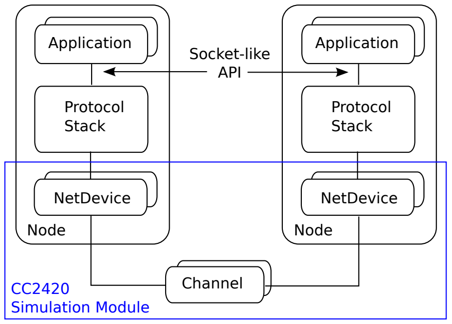

Figure 2. Overall structure of an ns-3 simulation [4].

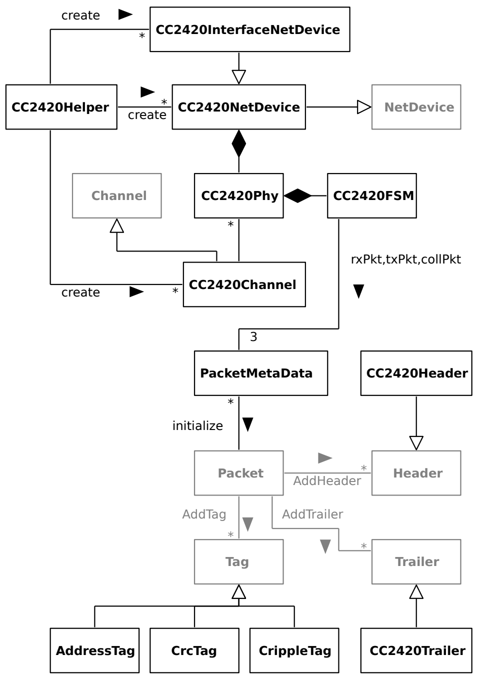

Figure 3. Structure of the CC2420 simulation module.

as well as applications and protocol stacks such as UDP / IP or TCP / IP can be installed on nodes using a CC2420 net device.

A first version of the CC2420 simulation module for ns-3 has been developed in [17] and is based on an existing ns-2 simulation module [5]. Its structure is shown in Figure 3. The classes *NetDevice*, *Channel*, *Packet*, *Tag*, *Header* and *Trailer* and the accordant relations between them are provided by ns-3.

For each node communicating via CC2420, *CC2420Helper* creates a *CC2420NetDevice* or *CC2420InterfaceNetDevice*. Together with the net device, a physical layer is created, which is connected with an existing or newly created *CC2420Channel*.

CC2420NetDevice provides an interface for the simulation module to interact with higher protocol layers. *CC2420InterfaceNetDevice* provides an extended interface, which allows not only sending and receiving of data, but also configuration of the transceiver, etc. (see Section III-C). *CC2420NetDevice* uses a *ReceiveCallback* whose interface is defined in the *NetDevice* class to forward received packets to higher protocol layers. *CC2420InterfaceNetDevice* additionally provides a *MessageCallback*. While a *ReceiveCallback* only allows the reception of regular messages (i. e., ns-3 packets), a *MessageCallback* is used to receive specific messages according to the extended interface.

CC2420Phy holds the configurable parameters of the transceiver (e. g., preamble length, sync word, transmission power, and channel number) and adds an accordant header and trailer (see below) to the ns-3 packets representing the frames of the transceiver. Furthermore, the physical layer controls the *CC2420FSM*, which realizes the state machine described in Section III-A.

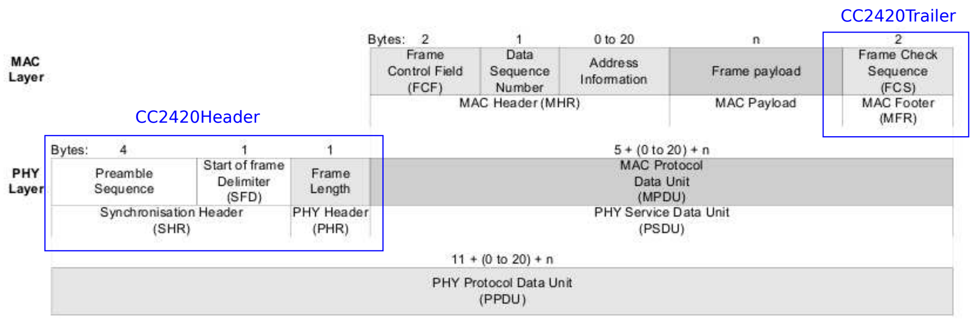

Figure 4. Frame format of the CC2420 transceiver [1].

The state machine used in the simulation module combines the state of the transceiver with the one of the simulated medium. This is the only part which was nearly completely reused from the ns-2 simulation module.

PacketMetaData is a helper class for storing the currently received, currently colliding or currently transmitted frame (represented by an ns-3 packet) together with its received respectively transmitted signal power and its starting time.

CC2420Channel manages the 16 channels of the 2.4 GHz band as subchannels and calculates when a transmitted frame arrives at a node and the signal power. To do this, the mobility models of sender and receiver as well as the transmission signal power are considered. The actual calculations are done by the delay model and the loss model, respectively.

Tags are a possibility to add simulation information to an ns-3 packet without actually extending its length, i. e., its duration on the medium. Source and destination address of a CC2420 frame are stored in an *AddressTag*. The addresses are not encoded in the accordant packet directly, because the CC2420 net device is primarily designed for broadcast transmissions. The actual addressing is done by a higher protocol layer and therefore already encoded in the MAC Protocol Data Unit (MPDU). Therefore, we do not include this information in the packet header, but, for compliance with the ns-3 *NetDevice*, provide it as tag.

CrippleTag is used for marking packets which could not be received correctly, either due to a channel change or because they have an other sync word and are therefore not recognized by the transceiver.

The frame format of the CC2420 transceiver is shown in Figure 4. Preamble Sequence, Start of Frame Delimiter and Frame Length are encapsulated by the class *CC2420Header* and are added to the ns-3 packet. Frame Length denotes the length of the MPDU in bytes; its maximal value is 127, since the highest bit is reserved [1]. Therefore, the MPDU can contain 127 bytes at most. The MAC Header (Frame Control Field, Data Sequence Number and Address Information) is not simulated. Since in the simulation 2 bytes are reserved for the Frame Check Sequence (FCS), the maximal payload size is 125 bytes. The FCS is an additional CRC checksum, which is realized by the class *CC2420Trailer*. It can be configured if this checksum shall be added to the packet. If it is added, the packet is marked with a *CrcTag*, because otherwise, the receiver cannot determine if the checksum is added or not.

The process of successfully sending a message with the *CC2420NetDevice* is as follows: Messages are sent from higher protocol layers in the form of ns-3 packets to the net device, which forwards them to the physical layer. This layer is responsible for registering the send request in the state machine, which causes the physical layer to forward the packet to the channel, when the calibration time has expired. The channel puts the packet to the accordant subchannel and schedules a reception event for all receivers attached to this subchannel by using a timer and an accordant callback. When the timer expires, the scheduler triggers the reception of the packet in the physical layer. A reception request is then sent to the state machine. After reception, the packet is forwarded to the physical layer, which forwards it to the net device. The *ReceiveCallback* is used to deliver it to higher protocol layers.

C. Enhancements

In ns-3, a standard net device only provides the possibility to send and receive packets. But for the CC2420 simulation module, further signals for configuration and information purposes shall be provided. For example, a received packet shall carry its signal strength, CCA changes or the end of a transmission shall be signaled, and changing channel and transmission power shall be supported. In addition, it should be possible to get information about the current configuration of the transceiver. Therefore, we designed the *CC2420InterfaceNetDevice*, which inherits from the standard *CC2420NetDevice* and uses the extended message interface shown in Figure 5.

RawDataMessage provides a generic class to represent payload data. *CC2420Message* provides a unified interface for all messages sent to or received from the *CC2420InterfaceNetDevice*. The messages *CC2420Send*, *CC2420Setup*, *CC2420Config* and *CC2420StatusReq* are sent from upper protocol layers to the simulation module, while *CC2420Recv*, *CC2420Cca*, *CC2420Sending*, *CC2420SendFinished* and *CC2420StatusResp* are sent from the simulation module to upper layers.

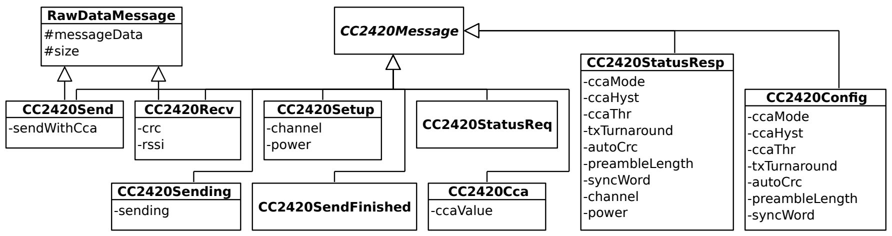

Figure 5. Extended message interface for CC2420 simulation module.

The *CC2420NetDevice* can be used with existing applications and protocol stacks provided by ns-3 without any modifications, since it sends and receives regular ns-3 packets. However, the *CC2420InterfaceNetDevice* requires adaptations in the upper layer(s), i. e., ns-3 applications and ns-3 protocol stacks, because the *CC2420Messages* are to be constructed, and *CC2420Messages* to be received.

The *CC2420Send* message is used to send a packet. It can be configured explicitly if CCA shall be considered or not, i. e., if the medium shall be checked before sending (the standard send method implicitly sends with CCA). *CC2420Recv* contains the received data and additionally provides information on CRC correctness and Received Signal Strength Indicator (RSSI). *CC2420Sending* delivers information about the start of a transmission. The parameter is *true* if the transmission could be started successfully, and *false* if not (e. g., if the medium is busy and sending with CCA is requested). *CC2420SendFinished* has no parameters and is returned when the transceiver has finished a transmission. *CC2420Cca* provides information about the current CCA status and is sent whenever the CCA status changes. *CC2420Setup* and *CC2420Config* are used for configuring the transceiver from higher protocol layers. *CC2420Setup* is used to adjust channel and transmission power, while *CC2420Config* carries values for CCA mode, CCA hysteresis, CCA threshold, TX turnaround, automatic CRC, preamble length and sync word. For the CCA mode, there is no check for valid IEEE 802.15.4 data at the moment. The *CC2420StatusReq* message can be used to get information about the current transceiver configuration. The transceiver module then sends a *CC2420StatusResp* message up, which contains the values of all configuration parameters.

Compared to the CC2420 simulation module for ns-2, we have added the possibility to request the current transceiver configuration. Further, configuration of CCA hysteresis, TX turnaround, automatic CRC, preamble length and sync word are supported, and information on CRC and RSSI is provided. Finally, we have implemented the use of different CCA modes.

IV. Simulator Framework FERAL and Integration of ns-3

In this section, we present the integration of ns-3 and the CC2420 simulation component into FERAL, a simulator framework for the rapid coupling of diverse simulators, such as simulators for Simulink and SDL models.

A. Outline of FERAL

FERAL is a Java-based framework for rapid simulator coupling with the objective to evaluate functional and non-functional requirements of networked systems [8]. A FERAL simulation system consists of a set of simulation components, which are executed by specialized simulators. In particular, existing simulators supporting different kinds of models and targeting different hardware platforms or communication technologies can be used together. Thereby, system components on different levels of abstraction can be simulated, which can, for instance, be applied for early prototyping. One example for this is the use of an SDL simulator together with ns-3 and our CC2420 module, which will be utilized in Section V. Thus, one can simulate existing SDL specifications on a high abstraction level together with a concrete medium model.

The execution of simulation components is controlled by *directors*, which support time-triggered as well as event-triggered semantics. Interaction between simulation components is realized by messages (e. g., event notification).

Three adaptation steps are necessary to build a simulation system with FERAL (see [8]). First, existing simulators, e. g., ns-3, to be used in the simulation system are integrated into FERAL. This is achieved by implementing the *SimulationComponent* control interface of FERAL, which needs to be done only once per simulator. Second, for each integrated simulator and type of simulation component, the FERAL component-specific interface is adapted and implemented. Third, simulation components are instantiated by choosing a simulator integrated into FERAL and by specifying and inserting an accordant behavior or communication model.

B. Integration of ns-3 into FERAL

Since FERAL is written in Java and ns-3 in C++, the integration consists of a Java part and a C++ part, which are connected by the Java Native Interface (JNI). Although ns-3 is an event-based simulator, our component has a time-triggered execution model. This approach was chosen because ns-3 already offers the possibility to execute the simulation for a specified time span. Therefore, no modifications concerning the clock and internal scheduler of ns-3 are necessary.

Figure 6 shows a class diagram of the Java part of the ns-3 simulator component for FERAL. The connection to the C++ part is realized by *NS3Interface* and *NS3Connector*. For each communication medium to be simulated with ns-3, a corre-

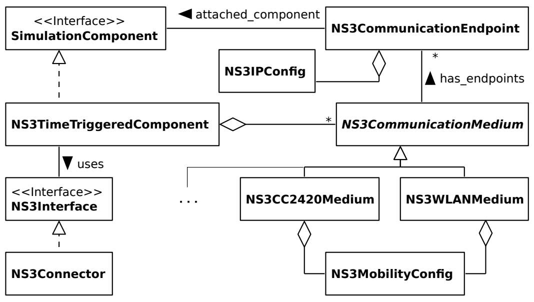

Figure 6. The ns-3 simulator component (Java part).

sponding *NS3CommunicationMedium* is instantiated. For wireless media, an *NS3MobilityConfig* is defined, which determines positions and movements of the nodes attached to this medium. Here, we focus on the CC2420 medium and neglect other media such as Ethernet or WLAN, which are also supported by the simulator component. *NS3CommunicationEndpoints* connect simulation components for functional behavior to the communication medium. On C++ side, these endpoints are represented by special applications.

In a stand-alone ns-3 simulation, applications installed on nodes are used to generate and process traffic. When incorporating ns-3 into FERAL, *virtual applications* are defined, i. e., applications that have no real behavior but rather serve as communication counterparts for the Java endpoints on C++ side. Communication between endpoints and virtual applications is realized via message exchange.

The counterpart of the *NS3CommunicationMedium* is the ns-3 channel, which simulates the actual communication medium. We have introduced a *communication mode* determining the type of communication used for a medium. Currently, the protocols TCP and UDP and communication without using a protocol stack, by sending broadcasts directly via net device are supported as communication modes. When using TCP or UDP, an *NS3IPConfig* must be provided for each endpoint, defining IP address and other parameters of the node. We have implemented one generic message type for all of these modes, which can be used independently of the concrete medium. For each mode, a generic virtual application exists, which transforms the messages to ns-3 packets and sends them according to the communication mode. For TCP and UDP, an accordant protocol stack is installed on the nodes during the initialization. Besides these universal modes, a medium-specific communication mode exists, which is currently only supported for the CC2420 medium.

C. Integration of the CC2420 Simulation Module into FERAL

Besides the integration of ns-3 into FERAL, two further steps are necessary for integrating the CC2420 simulation module. First, a medium class must be provided in order to create a CC2420 medium from a FERAL simulation system. With this medium, it is already possible to use the CC2420 module with the generic message interface described above. To use the CC2420-specific message interface described in Section III-C, the medium-specific communication mode – currently only available for CC2420 – had to be introduced. To use this message interface from FERAL simulation systems, it has to be represented in the framework. Therefore, an (almost) equivalent Java interface has been defined, which mirrors the one from Figure 5. Since communication between the FERAL framework and the ns-3 simulator is done via JNI, accordant code had to be provided to transfer the Java messages to the respective C++ messages.

If the medium-specific communication mode is chosen in combination with the CC2420 medium, a CC2420-specific virtual application is installed on the ns-3 nodes. This application forwards the messages directly to a *CC2420InterfaceNetDevice*, which processes the data and calls the accordant methods. The use of a protocol stack (e. g., IP) is not possible in this case, since communication is done directly via net device.

V. SIMULATIONS USING THE CC2420 MODULE

In this section, we present results of several simulation experiments, which show that the CC2420 simulation module is fully operational. In particular, we present the use of the CC2420 module in stand-alone ns-3 simulations, and in simulations where ns-3 is a simulator component of the FERAL framework.

A. Stand-alone ns-3 Simulations

In our stand-alone ns-3 simulations, simulation systems consist of two nodes acting as sender and receiver, respectively. An *OnOffApplication*, which sends values during configurable time intervals, is installed on the sender node, while a *PacketSink* is installed on the receiver node. Both applications are provided by ns-3. The structure is shown in Figure 7.

By default, the *OnOffApplication* repeatedly pauses for one second and afterwards sends packets for one second. We start this application at two seconds simulation time and simulate five seconds on the whole, which means that packets are sent in the interval between three and four seconds. By varying application data rate and packet size, we obtain three simulation systems. An excerpt of the first simulation system is shown in Listing 1. UDP sockets are used for sending and receiving (see lines 12 and 17), and the IP address of *PacketSink* is used as destination address for *OnOffApplication* (line 12). The first simulation uses an application data rate of 70 kbps and a packet size of 20 bytes (lines 13 and 14).

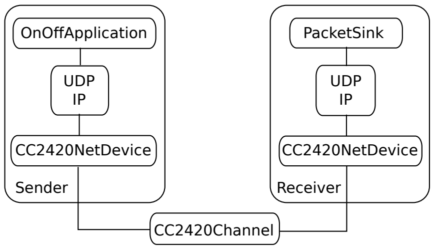

Figure 7. Structure of stand-alone ns-3 simulation systems.

```
NodeContainer nodes; nodes.Create(2);
CC2420Helper cc2420;
NetDeviceContainer devices;
devices = cc2420.Install(nodes, ... ); // CC2420NetDevice
InternetStackHelper stack; stack.Install(nodes);
MobilityHelper mobility;
...
Ipv4AddressHelper addr;
addr.SetBase("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces = addr.Assign(devices);

OnOffHelper onoff ("ns3::UdpSocketFactory",
  InetSocketAddress(interfaces.GetAddress(1), 9));
onoff.SetAttribute("DataRate", StringValue("70kbps"));
onoff.SetAttribute("PacketSize", StringValue("20"));
ApplicationContainer senderApps =
  onoff.Install(nodes.Get(0));

PacketSinkHelper pktSink("ns3::UdpSocketFactory", ... );
ApplicationContainer receiverApps =
  pktSink.Install(nodes.Get(1));
```

Listing 1. Excerpt of simulation system for first simulation.

With these values, all packets reach their destination, which is shown in Listing 2. The total number of bytes sent by *OnOffApplication* equals the number of bytes received by *PacketSink*, which means that all packets have reached their destination.

```
At time 3.00229s on–off application sent 20 bytes to
  10.1.1.2 port 9 total Tx 20 bytes
At time 3.00427s packet sink received 20 bytes from
  10.1.1.1 port 49153 total Rx 20 bytes
...
At time 3.99886s on–off application sent 20 bytes to
  10.1.1.2 port 9 total Tx 8740 bytes
At time 4.00084s packet sink received 20 bytes from
  10.1.1.1 port 49153 total Rx 8740 bytes
```

Listing 2. Output of the first simulation (all packets received).

In the second simulation, an application data rate of 90 kbps instead of 70 kbps is configured. Since UDP and IP headers as well as the CC2420 header and trailer have to be added to the application data rate, and the transceiver calibration time has to be considered, this data rate is too high for the transceiver. Therefore, not all packets can be transmitted, as shown in Listing 3.

The application sends all packets down the protocol stack, but since the CC2420 transceiver can only handle a new transmission request after the current one is finished, some of the packets are discarded by the transceiver. Therefore, we have identified a bottleneck in the system. This behavior can also be identified by additional log outputs of the CC2420 module not shown in the listing.

```
At time 3.00178s on–off application sent 20 bytes to
  10.1.1.2 port 9 total Tx 20 bytes
...
At time 3.99911s on–off application sent 20 bytes to
  10.1.1.2 port 9 total Tx 11240 bytes
At time 3.99932s packet sink received 20 bytes from
  10.1.1.1 port 49153 total Rx 5620 bytes
```

Listing 3. Output of the second simulation.

In the third simulation, the application data rate is set to 70 kbps as in the first one, but the packet size is extended to 150 bytes. This is more than the maximal payload of the transceiver, which is 125 bytes (UDP and IP headers and the CC2420 header and trailer even increase the packet size of the application). Therefore, an IP fragmentation takes place, which means that the IP packet is split into several subpackets to match the Maximum Transmission Unit (MTU) of the *CC2420NetDevice*. However, this does not work with the CC2420 transceiver, because it can handle a new transmission request only after the current one is finished. Because of this, only the first subpacket of each IP packet can be transmitted successfully. Since incomplete IP packets are discarded on network level, *PacketSink* receives no packets at all (see Listing 4).

```
At time 3.01714s on–off application sent 150 bytes to
  10.1.1.2 port 9 total Tx 150 bytes
...
At time 3.99429s on–off application sent 150 bytes to
  10.1.1.2 port 9 total Tx 8700 bytes
```

Listing 4. Output of the third simulation.

Next, we have repeated these simulations with slightly modified *OnOffApplication* and *PacketSink* in order to illustrate the use of the extended CC2420 message interface (see Figure 5). Instead of installing a protocol stack on the nodes, messages are directly forwarded from the application to the net device, which is now a *CC2420InterfaceNetDevice*, and vice versa. This also means that the transmitted packets are smaller, since UDP and IP protocol headers are omitted.

In the modified simulation systems, we have used the extended interface of the CC2420 module to change the channel from the default value 11 to 12 before message exchange is started (in *OnOffApplication* as well as *PacketSink*). A *CC2420StatusReq* message and the corresponding *CC2420StatusResp* message are used to check that the channel change has taken place.

The first of these modified simulations produces nearly the same result as the one with original *OnOffApplication* and *PacketSink*. Since the packets are smaller, they are received slightly earlier. In addition, there are further messages from the *CC2420InterfaceNetDevice*, which are received by the application (see Listing 5). For example, the *CC2420Sending* message with value *true*, which is sent up immediately, indicates a successful transmission start.

```
At time 3.00229s on–off–cc2420 application sent 20 bytes
  total Tx 20 bytes
At time 3.00229s on–off–cc2420 application received
  CC2420Sending message with value true
At time 3.00257s packet sink cc2420 received CC2420Cca
  message with value false
At time 3.00337s on–off–cc2420 application received
  CC2420SendFinished message
At time 3.00337s packet sink cc2420 received 20 bytes
  with CRC=true and RSSI=–67; total Rx 20 bytes
At time 3.00341s packet sink cc2420 received CC2420Cca
  message with value true
...
```

Listing 5. Output of the modified first simulation.

In the second modified simulation, all packets are now successfully transmitted. Since there are no UDP and IP headers, the application data rate of 90 kbps can be handled by the transceiver.

In the third modified simulation, the packet size of 150 bytes now exceeds the MTU size of the net device, since no IP fragmentation takes place. Therefore, no packet is transmitted.

B. Simulations with FERAL and SDL

ITU-T's SDL [9] is a formal specification language designed for distributed and reactive systems. System behavior is defined by extended finite state machines, which are connected through channels and communicate by exchanging signals asynchronously. Channels are also used to connect a system to its environment, e. g., the simulator framework FERAL. The integration of an SDL simulator component into FERAL has been presented in [8]. We use SDL to specify the behavior of nodes in the simulation systems on a high abstraction level, and ns-3 to simulate the medium by means of the CC2420 module.

In the following experiments, simulation systems consist of three nodes, two senders and one receiver, which are specified in SDL and executed by an SDL simulator. These nodes communicate over a wireless medium accessed through a CC2420 simulation module executed by ns-3. The structure is shown in Figure 8. The topology is chosen such that the receiver is positioned between the senders with the same distance to each of them. The extended CC2420 message interface is used for communication with the CC2420 module. The first sender begins transmission at 2,0001 seconds of simulation time and then sends one value every 100 milliseconds. The second sender begins at 3 seconds simulation time and sends one value every 200 milliseconds.

In the first simulation system, the standard configuration of the CC2420 module is applied. In particular, this means that the transceiver transmits at full power. Between 2 and 3 seconds of simulation time, only one sender is active, which means that all signals can be correctly received. Afterwards, every second signal of the first sender collides with a signal of the second sender, which means that only half of the signals of the first sender and none of the signals of the second sender can be received. This behavior is shown in Listing 6.

First, both senders transmit almost simultaneously (lines 2 and 3). Although CCA is used, both transmissions take place, because the calibration time for the first sender has not expired when the second sender begins its transmission. Therefore, the frames collide. The receiver detects that the medium is busy (line 6), but cannot receive a valid frame. Since medium occupancy is only detected by nodes which

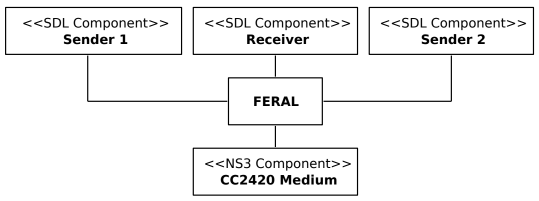

Figure 8. Structure of simulation systems using FERAL.

```
...                                                              1
3,0000: Sender 2: Sent CC2420Send with value 0x20 0x00           2
  0x00
3,0001: Sender 1: Sent CC2420Send with value 0x10 0x00           3
  0x0a
3,0002: Sender 2: Received CC2420Sending with value true         4
3,0003: Sender 1: Received CC2420Sending with value true         5
3,0004: Receiver: Received CC2420Cca with value false            6
3,0007: Sender 2: Received CC2420SendFinished                    7
3,0008: Sender 1: Received CC2420SendFinished                    8
3,0008: Receiver: Received CC2420Cca with value true             9
3,0010: Sender 2: Received CC2420Cca with value true             10
3,0011: Sender 1: Received CC2420Cca with value true             11
...                                                              12
3,1001: Sender 1: Sent CC2420Send with value 0x10 0x00           13
  0x0b
3,1003: Sender 1: Received CC2420Sending with value true         14
3,1005: Receiver: Received CC2420Cca with value false            15
3,1006: Sender 2: Received CC2420Cca with value false            16
3,1008: Sender 1: Received CC2420SendFinished                    17
3,1008: Sender 2: Received CC2420Cca with value true             18
3,1008: Receiver: Received CC2420Recv with value 0x10            19
  0x00 0x0b, CRC true, RSSI -67
3,1008: Receiver: Received CC2420Cca with value true             20
3,1011: Sender 1: Received CC2420Cca with value true             21
...                                                              22
```

Listing 6. Output of the first SDL simulation.

are not in transmission mode, the senders do not detect it here. Since the second sender only transmits every 200 milliseconds, the next frame of the first sender (line 13) can be received successfully (line 19).

In the second simulation system, we use a *CC2420Setup* message to reduce the transmission power of the first sender, while the power of the second sender remains unchanged. Since the distance to the receiver is equal, the signal of the second sender is stronger than the one of the first when arriving at the receiver. This way, a capturing effect can be observed, which means that the reception of the signal from the second sender is not disturbed by the interfering signal of the first sender. This behavior is shown in Listing 7.

```
1  ...
2  3,0000: Sender 2: Sent CC2420Send with value 0x20 0x00
     0x00
3  3,0001: Sender 1: Sent CC2420Send with value 0x10 0x00
     0x0a
4  3,0002: Sender 2: Received CC2420Sending with value true
5  3,0003: Sender 1: Received CC2420Sending with value true
6  3,0004: Receiver: Received CC2420Cca with value false
7  3,0007: Sender 2: Received CC2420SendFinished
8  3,0007: Receiver: Received CC2420Recv with value 0x20
     0x00 0x00, CRC true, RSSI -67
9  3,0008: Sender 1: Received CC2420SendFinished
10 3,0008: Receiver: Received CC2420Cca with value true
11 3,0010: Sender 2: Received CC2420Cca with value true
12 3,0011: Sender 1: Received CC2420Cca with value true
13 ...
14 3,1001: Sender 1: Sent CC2420Send with value 0x10 0x00
     0x0b
15 3,1003: Sender 1: Received CC2420Sending with value true
16 3,1005: Receiver: Received CC2420Cca with value false
17 3,1006: Sender 2: Received CC2420Cca with value false
18 3,1008: Sender 1: Received CC2420SendFinished
19 3,1008: Sender 2: Received CC2420Cca with value true
20 3,1008: Receiver: Received CC2420Recv with value 0x10
     0x00 0x0b, CRC true, RSSI -72
21 3,1008: Receiver: Received CC2420Cca with value true
22 3,1011: Sender 1: Received CC2420Cca with value true
23 ...
```

Listing 7. Output of the second SDL simulation.

As in the first simulation, both senders transmit almost simultaneously (lines 2 respectively 3). Since the frame of the second sender is stronger, it can be received successfully (line 8). This is only possible because the reception of the frame of the second sender begins before the frame of the first sender, since the transceiver cannot switch from a currently received frame to a stronger one. The next frame of the first sender (line 14) can be received successfully (line 20), since the second sender only sends every 200 milliseconds, which means that there is no colliding frame.

The simulation experiments show that our CC2420 module is fully operational, for stand-alone ns-3 simulations as well as simulations with the framework FERAL. Using simulator components already integrated into FERAL provides additional possibilities for specifying the behavior of nodes compared to stand-alone ns-3 simulations.

VI. Conclusion and Future Work

In this paper, we have developed a medium simulation module for TexasInstruments' CC2420 transceiver. As starting point, we have used an existing module for ns-2. By integrating the new module into ns-3, we can use existing ns-3 protocol stacks and applications to model the behavior of nodes using this transceiver and draw benefit from the active development of ns-3. We have then provided several enhancements for the CC2420 simulation module, which are accessible through an extended interface.

To use the simulation module in combination with other simulators, we have developed an integration into the simulator framework FERAL. Therefore, a general simulator component for integrating ns-3 into FERAL has been provided. This component also supports other ns-3 media, such as Ethernet or WLAN. Next, we have integrated the CC2420 module into this simulator component by providing a Java class for the medium and accordant elements to use the extended CC2420 message interface.

The CC2420 simulation module was first used to simulate stand-alone ns-3 systems. Then, the FERAL simulator component was used to simulate SDL systems which communicate via an ns-3 simulated CC2420 medium. These experiments have shown that the CC2420 simulation module is fully operational in the ns-3 context, and that the integration into FERAL provides additional degrees of freedom especially in the early development stages, where abstract models, e. g., Simulink or SDL models, are used to specify system behavior.

In our future work, we plan to further enhance the simulated state machine. At the moment, interference is not accumulated, which would be desirable for a more precise simulation of collisions. Energy consumption is also an interesting aspect to integrate into the state machine. Besides, we will implement further features of the transceiver, e. g., a check for valid IEEE 802.15.4 data in CCA modes 2 and 3. Furthermore, we will use the CC2420 module to evaluate realistic protocols for mobile ad-hoc networks, e. g., MAC and routing protocols. In addition, we are planning to perform real-world measurements with the CC2420 transceiver, in order to assess how accurate its behavior is simulated by our CC2420 module.

References

[1] Texas Instruments, "CC2420 datasheet," 2013, Revision SWRS041c. [Online]. Available: http://www.ti.com/lit/ds/symlink/cc2420.pdf [Accessed: August, 2013]

[2] IEEE, Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs). New York, NY, USA: IEEE Computer Society, Oct. 2003. [Online]. Available: http://standards.ieee.org/getieee802/download/802.15.4-2003.pdf [Accessed: August, 2013]

[3] USC Information Sciences Institute, "The Network Simulator – ns-2." [Online]. Available: http://www.isi.edu/nsnam/ns [Accessed: August, 2013]

[4] "The ns-3 Network Simulator," Project Homepage. [Online]. Available: http://www.nsnam.org [Accessed: August, 2013]

[5] T. Kuhn, "Model Driven Development of MacZ – A QoS Medium Access Control Layer for Ambient Intelligence Systems," Ph.D. dissertation, University of Kaiserslautern, 2009.

[6] T. Kuhn and P. Becker, "A Simulator Interconnection Framework for the Accurate Performance Simulation of SDL Models," in System Analysis and Modeling: Language Profiles, 5th International Workshop, SAM 2006, Kaiserslautern, Germany, May 31 – June 2, 2006, Revised Selected Papers, ser. Lecture Notes in Computer Science, R. Gotzhein and R. Reed, Eds., vol. 4320. Springer, 2006, pp. 133–147.

[7] J. L. Font, P. Iñigo, M. Domínguez, J. L. Sevillano, and C. Amaya, "Architecture, design and source code comparison of ns-2 and ns-3 network simulators," in SpringSim, R. M. McGraw, E. S. Imsand, and M. J. Chinni, Eds. SCS/ACM, 2010, pp. 109:1–109:8.

[8] T. Braun, D. Christmann, R. Gotzhein, A. Igel, T. Forster, and T. Kuhn, "Virtual Prototyping with Feral – Adaptation and Application of a Simulator Framework," in The 24th IASTED International Conference on Modelling and Simulation, 2013.

[9] International Telecommunication Union (ITU), "ITU-T Recommendation Z.100 (12/11) – Specification and Description Language – Overview of SDL 2010," 2012. [Online]. Available: http://www.itu.int/rec/T-REC-Z.100-201112-I [Accessed: August, 2013]

[10] C. Suh, J.-E. Joung, and Y.-B. Ko, "New RF Models of the TinyOS Simulator for IEEE 802.15.4 Standard," in WCNC. IEEE, 2007, pp. 2236–2240.

[11] R. de Paz Alberola and D. Pesch, "AvroraZ: Extending Avrora with an IEEE 802.15.4 Compliant Radio Chip Model," in PM2HW2N, B. Caminero, F. Delicado, and R. W. N. Pazzi, Eds. ACM, 2008, pp. 43–50.

[12] H. Joe, J. Lee, D.-K. Woo, P. Mah, and H. Kim, "Demo Abstract: A High-Fidelity Sensor Network Simulator Using Accurate CC2420 Model," in IPSN. ACM, 2009, pp. 429–430.

[13] A. B. Paul, S. Konwar, U. Gogoi, A. Chakraborty, N. Yeshmin, and S. Nandi, "Implementation and Performance Evaluation of AODV in Wireless Mesh Networks using NS-3," in 2nd International Conference on Education Technology and Computer (ICETC) 2010, vol. 5, 2010, pp. V5–298 – V5–303.

[14] H. Narra, Y. Cheng, E. K. Çetinkaya, J. P. Rohrer, and J. P. G. Sterbenz, "Destination-Sequenced Distance Vector (DSDV) Routing Protocol Implementation in ns-3," in SimuTools, J. Liu, F. Quaglia, S. Eidenbenz, and S. Gilmore, Eds. ICST/ACM, 2011, pp. 439–446.

[15] S. Vincent, J. Montavont, and N. Montavont, "Implementation of an IPv6 Stack for NS-3," in 2nd International Workshop on NS-2 (WNS2 2008), October 2008.

[16] N. Baldo and M. Miozzo, "Spectrum-aware Channel and PHY layer modeling for ns3," in VALUETOOLS, G. Stea, J. Mairesse, and J. Mendes, Eds. ACM, 2009, pp. 2:1–2:8.

[17] R. Groh, "Contributions to the Integration of CC2420 and SDL into ns-3," Bachelor Thesis (in German), University of Kaiserslautern, Computer Science Department, 2012.