CONFab: Component based Optimization of WSN Protocol Stacks using Deployment Feedback

Junaid Ansari, Elena Meshkova, Wasif Masood,
Arham Muslim, Janne Riihijärvi and Petri Mähönen
Institute for Networked Systems, RWTH Aachen University, Kackertstr. 9, D-52070, Aachen, Germany
Email: jan@inets.rwth-aachen.de

ABSTRACT

Wireless sensor networks are characterized by a large number of non-standardized protocols and varying application requirements. This creates need for a systematic approach to rapidly design and optimize deployment specific protocol stacks. We employ component based optimization as a candidate solution, and use it as a basis for an extensible software framework called CONFab. We treat a particular protocol stack as a collection of interdependent configurable components. CONFab captures a deployment scenario description, relates it to the desired performance metrics, and correspondingly suggests suitable protocol stacks and parameter settings. It utilizes ontology centric knowledge base to select components from a pool of alternatives, reason on their compatibility and, thus create appropriate protocol stacks. The framework is equipped with a number of additional plugins that allow, for instance, to incorporate feedback from deployed systems and user inputs to anticipate network performance. We use a set of well-known MAC and routing protocols to validate the framework on the Indriya testbed in different user specified application and deployment conditions. The results indicate that CONFab with its component based approach helps in obtaining suitable protocol stacks and thereby achieving high performance characteristics.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless communication

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

Ontology, knowledge base, components-based design, protocol stack composition, optimization, deployment feedback

1. INTRODUCTION

Wireless Sensor Networks (WSNs) and embedded networked systems are deployed for a wide range of applications [1]. The constrained nature of WSNs requires optimization of the resources in order to achieve longer lifetimes and support the desired quality of service (QoS). Often optimization has to be performed on per application basis since WSNs have a wide range of deployment conditions and distinct application requirements. A large number of protocols have therefore been developed to address different performance demands and address particular scenarios. However, as far as we are aware of, there exist no systematic tool to enable reusability of existing solutions for different deployments and application demands. Scenario-specific *protocol stack composition* and benefiting from the insights on the performance characteristics obtained from previously deployed solutions thus emerges as one of the *core design tasks* for sensor networks. Furthermore, the economic cost and the required human effort should be reduced for protocol stack design by *partial automation* of the related activities.

In this paper we propose a framework and software implementation, called CONFab (Component based Optimization for Networks with deployment Feedback), that assists and partially automates the task of selecting an appropriate protocol stack for a given WSN application. We adapt an optimization driven approach [2, 3] to this problem and take the component based (CB) view on realizing a protocol stack [4, 5]. In line with the TinyOS [6] component oriented design philosophy, we view a stack as a collection interconnected components of various granularities, ranging from simple modules, such as timers and packet sending functionalities, to complete protocols or even whole stacks. These components carry general and scenario-specific metrics and can be interconnected in diverse ways, each with their own characteristics, to provide the desired services. The reader should note that protocol functionalities obey complex interdependencies with one another that constrain interconnection and reconfiguration possibilities. Even though there are still considerable number of connectivity options that can be reasoned upon, for instance, on the functional dependencies. This way the design space for the WSN protocol stacks can be represented as a multitude of alternative components, their configurations, and interconnections (or "wirings") of the components. The effectiveness of a particular protocol stack highly depends on the deployment scenario. An appropriate component configuration can be chosen to realize a protocol stack based on the scenario descriptions, possible available insights from similar deployment experiences and expert inputs. Using a suitable level of abstraction, we can formally represent this process as an optimization task, consisting of a derivation of a pool of available component stacks and the selection of the best alternative according to the scenario definition. This optimization task can be formally solved, producing an abstract description of a protocol stack that can be mapped to real implementation and deployment.

The core of CONFab is a Knowledge Base (KB) that employs an ontology and a description logic [7] for suggesting an appropriate

MobiWac'12, October 21–22, 2012, Paphos, Cyprus.

combination of components. The KB stores information on protocol stack organization, available components, and scenarios in a structured manner, and further reasons on it. The natural advantage of the KB is its ability to capture and operate on descriptions at various levels of granularity, provide mapping and define relationships among different categories and sets of components. For example, this allows non-experts to define high-level scenario goals that are later on mapped by CONFab to protocol performance metrics. Similarly, these mechanisms enable stack performance prediction for scenarios on which prior experience is available. CONFab learns from such past experiences and utilizes expert inputs for decision making and ranking the composed protocol stacks. The framework operates on already existing protocol stacks, and provides possible wirings of components and their parameter configurations to compose new ones. CONFab is easily extendible and can be enriched with further functionalities realized either as plug-ins to the framework or through additions to the structure of the KB as extension of its ontology. For example, we introduce a plug-in to find commonalities among different stacks to derive a new stack.

We validate CONFab through well-known WSN MAC and routing protocols on the Indriya testbed [8] in different application and deployment settings. We show that CONFab with its component based approach can assist and partially automate a protocol stack design that leads to improved performance and higher network lifetimes. For instance, we observe up to 37 % increase in energy efficiency at comparable packet delivery ratio for medium sized networks (30 – 70 nodes) compared to the corresponding monolithic protocol stacks (cf. Figures 9(b) and 9(c)).

The rest of the paper is structured as follows. Section 2 reviews the related work. Section 3 describes the design methodology and realization of CONFab, which is evaluated using popular MAC and routing protocols on the Indriya TelosB testbed in Section 4. Finally, Section 5 concludes the paper.

2. RELATED WORK

To the best of our knowledge, there is no other framework for the development of WSN protocol stacks completely analogous to CONFab. However, there is of course considerable amount of related work on the concepts employed by our framework.

A large number of independent MAC [9] and routing [10] protocols have been proposed during the past years. While the performance evaluation of individual protocols have been carried out by designers, one of the dilemmas that practitioners and researchers face is that there is a limited understanding of the behavior of a particular WSN stack given the application requirements and the deployment conditions. Furthermore, since many applications have varying QoS demands over time, a protocol stack is expected to adapt its behavior to the changing application requirements and the environmental conditions. One of the major problems with existing WSN stacks is their composition based on protocols with monolithic style of implementation and arbitrary incoherent interfaces which restricts possibilities for run-time adaptability and code reuse [11, 12]. It has been long argued that careful construction of protocols and their interfaces can significantly improve the overall network efficiency [13, 14]. Choi *et al.* argue that protocols depict complicated and unpredictable interactions and an isolation layer can harmonize the interfaces [15]. Research on the cross-layer design also targets the same objective [16].

In order to realize an adaptable, reusable and scalable stack, component oriented approach to network design has been proposed and actively studied in the past years [4, 5, 17–19]. Decomposing a protocol stack into discrete, reusable components with well-defined interfaces that implement different functionalities/services and appropriately binding them, can lead to considerable performance gains [20]. One of the underlying reasons is that the interconnections of these components can be dynamically altered to obtain different protocol functionalities. This approach therefore allows mitigating the impact of network dynamics. Furthermore, this modular design also facilitates cross-layer optimization.

While component oriented approach to protocol design is more than two decades old [4], protocol realizations in practice have emerged only in the recent years. For component oriented MAC implementations in WSNs, we refer the reader to [11, 12]. Ivan *et al.* present a middleware based approach to realize routing protocols for traditional wired networks while decomposing them into fundamental building blocks and outlining the role of each component as a unit or auxiliary component [21]. Köpke *et al.* suggest separation of the packet flow and meta-data. They propose a publish/subscribe based method for exchanging meta-data among different components [22]. One of the examples for component based view on realizing routing in MANETs is the proposal by Baras *et al.* [23], where performance is measured at the component level rather than at the protocol level.

Although these proposals aim at modular designs, none of them describes how to decompose WSN protocols into reusable components and how to realize the desired protocol stacks from them according to the application demands.

In this paper, we follow the optimization-driven approach to network development and run-time management [2, 3]. The related research has mainly been in the theoretical domain, with a focus on re-engineering existing protocols, and applying techniques from optimization and control theory to study protocol design [20].

In order to autonomously solve the task of protocol stack composition, we need to describe a network and its performance characteristics and also to capture the application QoS requirements and reason on those. An appropriate taxonomy can be realized using ontology and related reasoning frameworks. Web Ontology Language (OWL) [24] and its framework [25] is a natural choice in this context, and has been successfully applied to, for example, composing web-services [26]. We can take the similar approach to decide valid wirings among the components composing a protocol stack. There are a few proposals for using ontology-based reasoning for WSNs. For example, [27] provides a high-level description of a network and its performance parameters. Unfortunately, this proposal lacks implementation and the evaluation results. Moreover, this scheme does not explicitly foresee the possibility of learning from the past deployment experiences.

Other alternatives to modeling of protocol stacks as part of a networked system include well-known meta-modeling frameworks such as UML and SDL. These systems can provide an alternative to the description of network services and the corresponding modules, but do not support descriptions of wirings of components and iteration between different combinations based on the user-specified goals. In the context of WSNs, a number of solutions have been proposed that aim at easing the development and run-time configuration of protocol stacks. Alkazemi *et al.* propose a framework [28] to automate and standardize network code construction according to the application goals for WSN. Amirhosein *et al.* develop a component based framework to simplify WSN software implementation and configurations, by simplifying middleware services, enabling tunability of operating system software by wrapper components, and providing support of component reusability [29]. The Semantic Streams [30] is one of the programming models that abstracts the functionality of a sensor network and allows to formulate complex queries without understanding the inner low-level protocol behavior. Other models, such as, [31] enable run-time and

compile-time reconfiguration of WSN nodes, but unfortunately do not support learning from the experiences of actual deployments. Our framework enables the latter option by introducing plug-ins that integrate our prior works [12, 32].

3. CONFAB DESIGN

We start by stating the component-centric optimization problem for network protocol stack composition. Then as a practical solution, we decompose selected MAC and routing protocols into components, describe the architectural details and ontology description of the KB and the decision logic. We walk through an example to show how the framework reasons on components to select an appropriate protocol stack.

3.1 Optimization-Centric Problem Statement

We consider *networks* as complex dynamic systems consisting of interdependent software and hardware elements, each possibly effected by the external environmental factors, serving to satisfy a combination of goals directly or indirectly stated by network stakeholders. *Network performance* is a measure of satisfaction of these goals, examples being application performance or efficiency of utilized network and system resources. A *scenario* comprises all aspects of the external environment, stakeholder goals, operational conditions and their dynamics, applications behavior, network and system characteristics. Thus, the aim of network development can be formulated as building of a system that shows the best network performance for a given scenario.

Development of a networked system, especially a WSN, involves assessment of alternative design and configuration. This optimization problem is constrained by the presence of limited resources including technological and economical aspects. For example, there is a limit on the software and hardware components. More formally, we seek a solution

$$S_{sc.} = \arg\max_{s \in \mathcal{S}} N(s \mid O), \tag{1}$$

where $S_{sc.}$ is the optimal solution for a networked system for a given scenario, selected from the collection of allowable contraint-satisfying systems $\mathcal{S}$, and measured in terms of network performance N in an environment O. Our current implementation operates only with the software elements and decides on a common protocol stack for all nodes of a WSN. Therefore, in the context of our work a solution s is a network protocol stack, consisting of *components* c and their wiring w. The collection of components c might be fixed, or form a part of the optimization process considered. Finally, each component might have a collection of configurable *parameter* settings k. Systems can therefore be written:

$$s \in \Big\{ \big(\{c_i\}, \{k_j \mid c_i\}, w(\{c_i\})\big) \mid w \text{ is valid wiring} \Big\}, \tag{2}$$

where the validity of the wiring w is accessed in terms of the component interfaces. We assume that there exists a pool of components, alternatives for their combinations and parameter configurations k, as well as possibilities for development of new ones.

We will also limit the structure of our measure of network performance, focusing on applications' performance $P(s)$, and degree of satisfaction of additional constraints $\mathcal{C}(s)$, e.g., w.r.t. consumed hardware resources in our implementation, as further discussed in Section 3.3. Therefore, the optimization task from (1) becomes

$$S_{sc.} \approx \arg\max_{s \in \mathcal{S}} N\Big(\big(P(s), \mathcal{C}(s)\big) \mid O\Big), \tag{3}$$

where the approximation is due to the limitation of the considered goals compared to the general original formulation.

The problem (3) cannot be solved directly, since the functional form of the performance measures are not explicitly known for any realistic system. Therefore, we have to construct estimators for these either based on *models* of the "real world", or prior knowledge from, for example, earlier deployments. It is clear that we cannot capture the "world" we are reasoning about in full detail, and we have to operate on a set of its approximations, which can both increase the speed of the solution acquisition, as well as considerably reduce its accuracy.

Approximation of the original network performance measurements might be beneficial at different stages of designing the solution. For example, optimization of N and its parts could be approximated by a constraint-satisfaction problem, which might be applied at earlier design stages to filter out clearly inapplicable solutions. This allows splitting the potentially NP-hard problem of protocol stack composition into several parts and simplifying the computation by gradual filtering of the stack compositions. At the same time if parameters or their boundaries for filtering the potential solutions are inaccurately set, this will lead to exclusion of potentially valuable alternatives thus degradation of the overall quality of the achieved results. We address the modeling aspect of the optimization process by employing multi-stage reasoning in CONFab. This is also enabled by multiple plug-ins to CONFab that can operate on objects of different complexity and accuracy of representation. For example, we support both QoS class-based definition of the application demands, as well as utility-based reasoning [20, 32]. Our framework also naturally supports scalable reasoning, where relevant methods are applied in batch manner.

3.2 Components and Protocols for Network Stack Design

Solution of the optimization problem given in Section 3.1 leads to a particular protocol stack that provides optimal (in the sense of maximizing N) performance in a given scenario. Through (2), this stack can be decomposed into a set of components, their interfaces and configuration settings. This decomposition task is, however, challenging in practice and has been tackled only under heavy mathematical modeling assumptions [20].

We suggest merging top-down (decomposition task) and bottom-up approaches to derive components for a protocol stack design. Using stack decomposition, we seek to exploit existing or create new re-useable components, conditioned on the available development resources. We then apply forward engineering in CONFab for employing these generic components to solve scenario-specific optimization tasks. If the desired results cannot be achieved through a valid combination of already existing components, new components need to be developed and added to the pool of existing ones.

We take the classical software engineering approach and decompose a protocol stack into components derived on *behavioral* and *structural* basis. The goal of the decomposition process is to create highly re-usable components with inter-connectable interfaces. The granularity of these components is selected so as to facilitate the protocol stack composition suitable to diverse applications and deployment scenarios. Moreover, it is desirable to have a minimal developer expert knowledge into inner logic of the stacks.

The protocol stack from the behavioral perspective can be viewed as a flow of *services* hosted by logical units (*functionalities*). Different levels of services are involved in a top-down flow starting from an input as the application request for data delivery and down to radio communication. We consider the following core high level services: data communication, forwarding, addressing, (sub)network value assessment, media access, radio transreceiving, and adaptability. These services are hosted by functionalities such as stack

unit, routing, MAC, decision-making, radio unit. Components and interfaces are structural reflections of these elements, i.e., functionalities are implemented by components and services are reflected in interfaces. Our experience indicates that designing services and realizing components at a very coarse granularity restricts their reusability, maintainability and performance due to increased inner complexity. One simple example is a monolithic realization of a protocol, which offers multiple services. This certainly limits the possibility of sharing these services between different protocols. For example, both MAC and routing protocols might benefit from link quality estimates. For strictly layered protocol stack implementations this functionality can be redundant, which can be eliminated through a common component interface. Current protocol implementations often do not use unified interfaces, which make information exchange between layers more difficult or even impossible [33]. Obviously absence of unified interfaces also limits the opportunities for autonomous stack design. On the contrary, very fine grained stack decomposition leads to complex component interdependencies that result in high wiring and reasoning overheads.

3.3 Protocols Chosen for Design Validation

We have selected well-known and widely used MAC and routing protocols for our design validation and performance evaluation. The three chosen MAC protocols, BoXMAC-2 [34], X-MAC [35] and TrawMAC [36]), are based on the preamble sampling principle while the routing protocols, CTP [37] and S4 [38], use Expected Transmission indeX (ETX) for link estimation. Though the behavior of the selected MAC and routing protocols vary significantly depending upon the network and traffic conditions, both the sets of MAC and routing protocols share common functionalities. We use two realizations of these protocols as inputs for the CONFab reasoning: One is based on their monolithic implementation from an openly available code, while the other uses the component based realization as we will describe further. The components include certain configuration parameters, such as duty cycle (or sleep interval), beaconing rate, routing decision metric, etc. Platform specific features or constraints can also be supported in the components such as the hardware ACK or address recognition provided by the CC2420 radio chip. These parameters are set at the design time by CONFab or through a reconfiguration component at the run-time.

The diversity of WSN applications makes it challenging to define all the parameters and their settings which can fully capture an application scenario. Besides the network life-time, application data rate, fault resilience, and network dynamics are the most prominent parameters. Other metrics supported by CONFab includes RAM/ROM usage, which is typically employed as a constraint on the available node memory. As a part of a scenario description, the expected deployment size of a network, application data rate, and a characterization of its behavior (e.g., bursty or constant application traffic rate) can also be specified. However, the results presented in Section 4 focus only on the Packet Reception Ratio (PRR) and power consumption since we consider these attributes to be highly important for WSNs.

3.4 Protocol Decomposition

Protocol decomposition in CONFab is carried out as described in our previous works [12]. This is not the only decomposition approach that can be accommodated by CONFab. With appropriate ontology adjustments (in terms of valid interrelations between different components and services, and other/additional meta-data), many of the related works reviewed above can also be incorporated with the same core ontology and CONFab structure.

We found it useful to reason not only on the behavioral and structural abstractions of a protocol stack, but also consider *two* levels of component granularity. The primitive level components typically include the low-level hardware dependent functionalities providing platform independent interfaces such as timers, random number generators, carrier sensing, send-/receive frames, etc., in the case of MAC protocols. Primitive components enhance protocol development due to their reusability and shared interfaces [12]. However, due to a low granularity, it is not viable to directly use them for semi-autonomous protocol stack design based on user and expert inputs and the deployment feedback. Typically more than ten such components are required in realizing a MAC protocol [12]. These components display complex interdependencies. Therefore, it is highly challenging to autonomously assess their individual contribution in order to observe the high-level performance metrics, which could be directly accessed by a user.

On the other hand, these primitive level components can be combined to form other component(s) that offers more complex (higher level) service(s). This certainly reduces design complexity and easily enables semi-autonomous reasoning in CONFab. As an example, medium access service provided by MAC protocols can benefit from a *preamble sampling* and/or *neighborhood sleep schedule* functionalities which are common to many of WSN MAC protocols [9]. Furthermore, for instance, the *preamble sampling* MAC component consists of primitive components: *timer* and *send-frame*. Similar to the MAC case, CONFab reasons on functionalities that distinguish between different methods of route construction such as tree- or cluster-based routing. The respective high level decomposition include 1) a *beaconing* component, which broadcasts protocol specific knowledge set; 2) *forwarding engine* calculates efficient routes based on the indicated routing metric; 3) *routing daemon* stores the routing information for different types of routing protocols. The routing protocols use a common interface. Different types of component based solutions can be realized using these three basic building blocks. A number of routing decisions can be supported, for instance, using the *link estimator* component, which provides ETX or *node energy detector* component.

Our approach (cf. Section 4.1) shows that treating protocol stack as a set of services that are provided by respective components easily allows realizing different protocol behaviors without loosing the performance characteristics compared to monolithic implementations. In fact, component based realization enable us to easily change the functionality of one of the protocols used in our case studies (S4 [38]), which shows improvement in energy efficiency by approximately 50 % as can be inferred from Figure 6(d). At the same time granularity of the established services, developed components and the complexity of their interconnection logic remain simple enough to be easily accommodated for automatic reasoning such as enabled by CONFab.

3.5 Implementation Architecture

This section describes the overall architecture and implementation details of different parts of CONFab. It further discusses how composite scenarios are defined and new stacks are realized.

3.5.1 Basic Design

As discussed above, the task of protocol stack composition requires several inputs, including a pool of components with specifications of their interfaces and configurable parameters, description of the logic of their valid wirings, estimates on the performance of whole stacks or individual components conditioned on particular operational conditions, and the overall scenario description. These inputs can be provided by experts (e.g., component definitions), users (e.g., scenario descriptions), or they can be obtained through

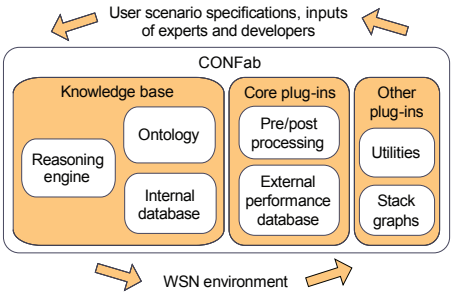


Figure 1: The overall architecture of CONFab.

post-processing of data feedback from the deployment sites. This performance data is stored in a separate database. The reasoning logic on component wiring and filtering according to *qualitative* metrics is encoded as part of the framework through the KB and its ontology. The developed plug-ins also allows *quantitative* utility-based assessment of results. CONFab is extendible: It can be enriched by further functionalities and reasoning capabilities realized either as plug-ins that can be implemented in any framework and connected via a Java-interface, or through additions to the structure of its KB via extensions of the ontology. The overall CONFab architecture is shown in Figure 1. It can be viewed as a collection of feedback loops with interactions from users, experts and WSN deployment sites, corresponding to the entities discussed above.

The CONFab framework has been implemented using a number of development environments and languages (cf. Figure 3). The ontology is realized using the OWL language [24] and the KB is built within the Protégé framework [25]. The plug-ins are connected to the KB through the Jena Java-based interface [25] bound with TCP/IP sockets. The plug-ins themselves can be realized in any language supporting basic communication facilities. For example, in our implementation we use Java, Matlab and R for GUI realization, performance results processing, utility-based reasoning and component graph comparison. FaCT++ [39] is used by our system as a reasoning engine. As its integral part the framework also provides a centralized storage that is useful for further development of the components of a stack.

3.5.2 *Knowledge Base and Reasoning*

The core element of the framework is the KB which comprises an ontology, a database and a reasoning engine. The KB can also be used as a unifying framework for switching among different models (see Section 3.1); realized through different levels of system abstraction incorporated within the ontology. For example, we in-

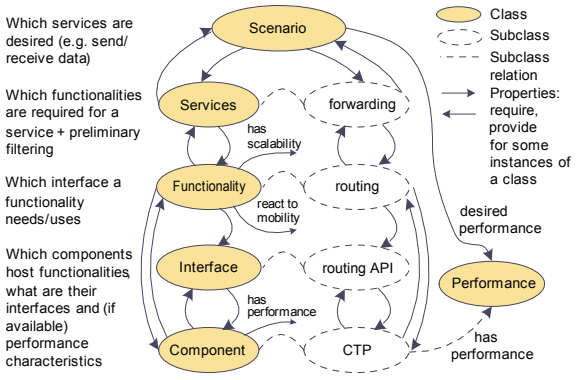


Figure 2: Sample ontology classes with the chosen properties and their interrelations.

voke upper-level behavioral reasoning that abstracts a component as a functionality that provides and requires certain services, i.e., we employ a form of the service-oriented design. From the other side we reason on components and their interfaces that map to concrete implementations (see Sections 3.1 and 3.2). Figures 2 and 4 show example classes of the ontology and relations between them, defined through properties, that are used for protocol stack construction. Figure 4 shows the major steps of the reasoning process for the stack construction starting from the upper-level behavioral reasoning on required services, followed by the deduction of the functionalities that provide those services. These are then mapped to components and interfaces, respectively, which leads to a valid protocol stack and definition of the relevant TinyOS/nesC components. These components can be automatically combined through appropriate interfaces and deployed using the system and XML-based meta-language as in [12].

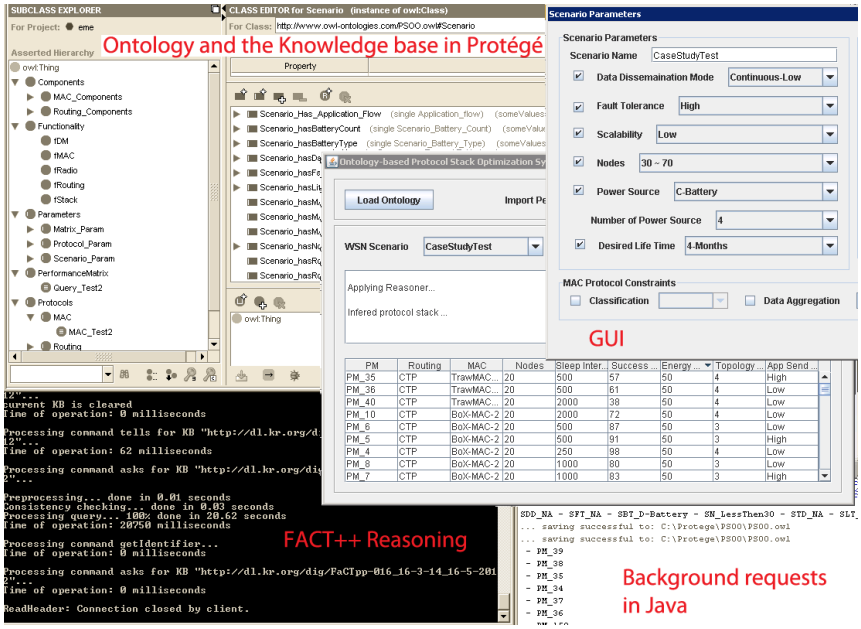
CONFab allows set- and utility based scenario definitions. In the first method we operate on sets of values without making distinctions between those falling into one range. For example, in our implementation networks of size 40 to 70 nodes are denoted as "medium-sized". Similarly, application data rates below 0.01 packets per second are denoted as "low-rate". The performance values are clustered in the same manner, e.g., power consumption, latency, etc. This allows to split the reasoning process into several stages thus providing the possibility to offload computation as a background batch processing and limit the time a user has to wait online for CONFab reasoning to finish.

CONFab executes multiple levels of reasoning because it is difficult to model the influence of a component on a protocol stack separately without considering its interdependencies with other elements. Therefore, we filter on the first stage of reasoning on the components using only the expert inputs to specify whether a component is suitable for, e.g., large-scale deployments or high mobility conditions. Note that such characteristics are properties of both functionality and component classes of different levels of granularity. For example, cluster-based routing protocols (e.g., S4), irrespective of their implementation details, are typically scalable. For better accuracy in the reasoning process, a more generic class carries the most conservative metric of its members. The functionality and service classes are used to reason only on the logical interconnections, and not on the implementation specific properties – making this level of reasoning independent of implementation specifics. Set-based performance reasoning is applied on the level of components to limit the number of stacks for which we query the performance database to enable utility-based reasoning. This is the last, most detailed, level of reasoning and is applied before displaying the recommended stacks to the user.

Using the KB and the above reasoning process, CONFab recommends the most suitable protocol stack to a user. It learns from its experience and utilizes expert inputs in cases where component characteristics cannot be derived automatically, or where no performance feedback is yet available. CONFab operates not only on the predefined protocol stacks, but also provides recommendations on possible wirings and configurations of monolithic protocols or the components they consist of in order to create new stacks.

3.5.3 *Composite Scenarios and New Stacks*

CONFab allows two types of scenario definitions. In the first, it does not portray dynamics in the system behavior and one may merely express a static transition in the state or parameter such as a change in the network application rates from low and high. This type of definition influences only the reasoning on prediction of results for a stack from other deployments. CONFab is also capa-



(a) Sample CONFab screenshot that include the GUI, the Protégé development environment, the FACT++ server, and some of the Java background processing.

```
<Scenario rdf:ID="HabitatMonitoring">
  <Scenario_hasBatteryCount rdf:resource="#SBC_4"/>
  <Scenario_hasFaultTolerance rdf:resource="#SFT_Low"/>
  <Scenario_hasLifeTime rdf:resource="#SLT_4-Months"/>
  <Scenario_hasNodes rdf:resource="#SN_31-70"/>
  <Scenario_hasTopologyDynamics rdf:resource="#STD_None"/>
  ...
</Scenario>
<owl:ObjectProperty rdf:ID="Scenario_hasNodes">
  <rdfs:domain rdf:resource="#Scenario"/>
  <rdfs:range rdf:resource="#Scenario_Nodes"/>
  <rdfs:subPropertyOf rdf:resource="#Scenario_Object_Param"/>
  ...
</owl:ObjectProperty>
<Scenario_Nodes rdf:ID="SN_31-70"/>
<Scenario_Nodes rdf:ID="SN_GreaterThen70"/>
<Scenario_Nodes rdf:ID="SN_LessThen30"/>
<Scenario_Nodes rdf:ID="SN_NA"/>
<owl:Class rdf:ID="StackComposition">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasStack"/>
      <owl:someValuesFrom rdf:resource="#Stack"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasServices"/>
      <owl:someValuesFrom rdf:resource="#SendReceive"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>
```

(b) Sample OWL classes, properties and instances. Only a few illustrative parts are shown.

Figure 3: Sample CONFab screenshots and a short example of the OWL code.

ble, albeit in a limited fashion, of addressing dynamics in network behavior, including variations in goal statements or changes in the wireless environment. This is done through sub-scenario definitions and with a graph-comparison plug-in implemented as an extension to the basic framework. CONFab allows users to explicitly state for an example that the current scenario comprises 30 % of a particular behavior of the scenario A and 70 % of the scenario B as illustrated in Figure 4.

The composite scenario definition can be exploited for derivation of composite stacks, which aims at facilitating adaptive behavior. For example, we evaluate in Section 4 a scenario where an application switches between two transmission rates. The user defines this as a combination of two sub-scenarios that differ in application rates only. CONFab starts the reasoning process and first deduces the lists of recommended stacks for each of the sub-scenarios. The next step is to determine which stacks from these can be combined through the introduction of a separate decision-making (DM) component. The DM acts according to a decision tree and determines on which branch of the component graph should be executed based on the given conditions. When combining stacks, the two initial component graphs are compared and where their directed paths diverge in the merged version an empty vertex is added, that corresponds to a DM component. Its output is a new protocol stack with a decision-making component selected either by finding an interface compatible DM already present in the system or indicating to the user the absence of the appropriate component and supplying a template to be filled in.

The constructed stacks that do not fulfill the various constraints on the available system resources, e.g., program memory, are filtered out during the process. Each stack is also checked for compatibility to run-time distributed behavior, which is realized by introducing a dedicated component parameter property. For example, we typically cannot use different duty cycle settings on different nodes as this may prevent them from communicating with each other. The derived composite stacks, if showing good performance, are added to the pool of stacks for future consideration. For example, in our work the Composite MAC is a result of such a reasoning illustrated in Figure 4, as TrawMAC shows better energy efficiency while BoXMAC-2 shows higher reliability for bigger traffic volumes. We found that switching between those two behaviors can be made based on the retransmission counts and traffic patters/type. The CTP/Composite MAC stack turned out to be the most efficient for our experiments as will be discussed in detail in Section 4.2.

4. PERFORMANCE EVALUATION

We have carried out an experimental campaign for 700+ hours on the Indriya [8] TelosB testbed at the National University of Singapore to establish the KB for CONFab, validate and evaluate different performance characteristics. We have studied the effects of different parameters of our above mentioned MAC and routing protocols (cf. Section 3.3) for diverse application traffic patterns in different network sizes. Furthermore, we have also conducted experiments for studying the behavior of different component combinations in varying application and network conditions. Each of the experimental tests was performed for 20 min and was repeated three times to obtain better statistics. In order to systematically extract the performance statistics from the network, software adapters for monitoring energy consumption (similar to [40]) and packet transmission/ reception statistics at the MAC and routing protocols were deployed at each node. The performance data logged for experiments on the testbed was fed to the CONFab KB.

4.1 Component based Stack Implementation

In this section, we first compare the performance results of component based protocol stack implementation to its monolithic counterpart and then study new possibilities for stack design.

4.1.1 Design Validation

We have conducted comprehensive performance comparison studies of monolithic implementation combinations (TrawMAC/S4, Traw-

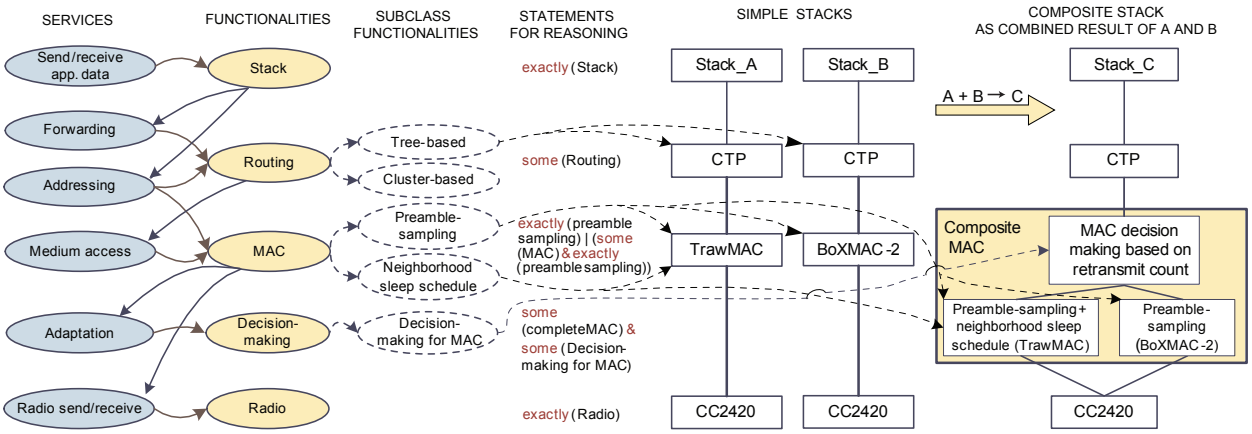


Figure 4: Sample relations between classes and reasoning statements that lead to different stack compositions. The stacks shown are among the ones used for the evaluation process. Stack C evolves as a combination of stacks A and B.

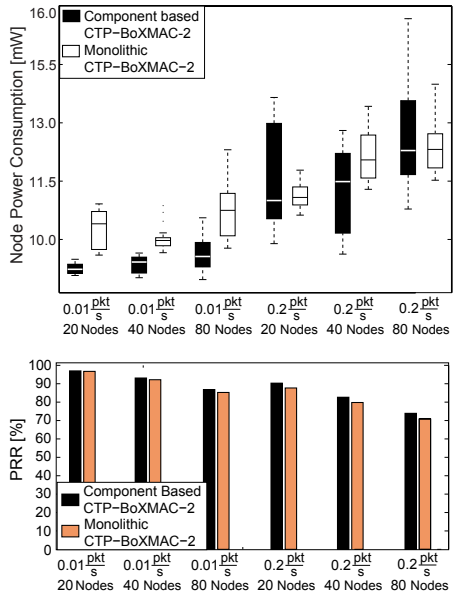


Figure 5: Per node power consumption (top) and average PRR (bottom) of component based CTP/BoXMAC-2 stack and its monolithic counterpart.

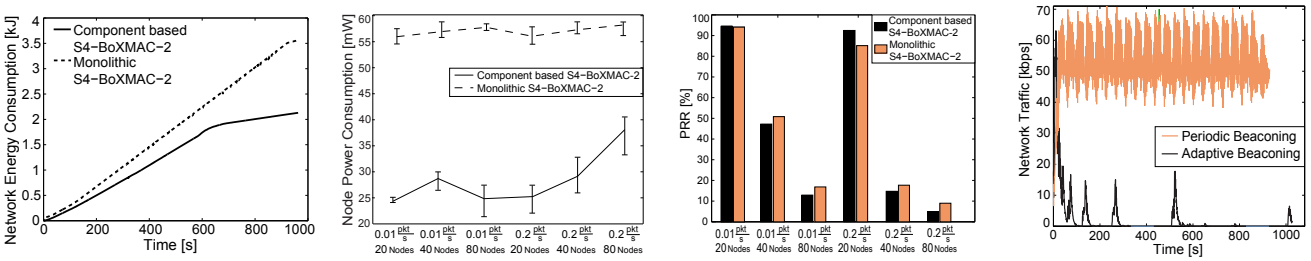
MAC/CTP, BoXMAC-2/S4 and BoXMAC-2/CTP) of the selected MAC and routing protocols to their corresponding CB implementations. The experiments were performed on 20, 40 and 80 nodes at a low application transmission rate of one packet every 100 s and a high application transmission rate of one packet every 5 s. The duty cycle was kept at 2.2 % with a channel sensing duration being 11 ms in all of these experiments. We have observed that the performance characteristics of the component based implementations of the protocols closely resemble to their monolithic counterparts in terms of the energy consumption and the application PRR in all the cases. Figure 5 shows the average power consumption and application PRR comparison for CB-CTP/CB-BoXMAC-2 to their monolithic counterparts at different application traffic rates and network sizes. The average per node power consumption comparison shows an increasing trend with higher application traffic rate, whereas the average application PRR shows a reverse trend.

4.1.2 New Component Combinations

Common data structures for a MAC-routing stack allow saving memory usage while combining control signaling (synchronization, beacon messages, neighborhood and route discovery, etc.) leads to a significant reduction in the network traffic. A lower network traffic results in bandwidth and energy conservation. Figure 6(a) shows the energy consumption for a protocol stack consisting of CB-S4 and CB-BoXMAC-2 with comparison to its monolithic counterpart in a network consisting of 80 nodes at an application rate of 0.2 pkt/s. In the monolithic implementation of S4, each node periodically broadcasts reverse link qualities of its neighbors to establish bidirectional link qualities. CB-S4 implementation uses the Link Estimator component to piggyback reverse link qualities for each broadcast transmission. This considerably reduces the amount of link specific messages in a network as can be observed from Figure 6(a). Power savings achieved for this case range from 35 % to 55 % depending upon the network size and application rate as can be observed from Figure 6(b). Figure 6(c) shows that while allowing significant amount of energy conservation, the average application packet delivery ratio of the CB implementation for different network sizes remains almost the same as for monolithic implementation. It can, however, be seen that as the network size grows, the average packet delivery ratio decreases. This is because of the absence of packet queues in S4, which leads to packet drops as was also observed earlier [38].

Unlike the layered approach of protocol stack implementation, component based approach allows combining different functionalities among protocols to achieve improved performance characteristics. However, all the combinations are neither possible nor they guarantee network efficiency. An experimental basis, simulation study or analytical modeling is required to establish the application specific efficient combination of components. In the following, we look specifically into routing protocols. Typically, beacon messages are broadcast at fixed periodic intervals. The beaconing interval is generally selected based on the dynamics in a network and the required allowance time for updating information across a network. A short interval usually results in higher traffic overhead but enables to propagate information (time synchronization, new routes, new nodes, removal of outdated links, unreachable nodes, etc.) in a shorter time. Adaptive beaconing efficiently handles this tradeoff thereby simultaneously achieving faster update of information and keeping the network traffic overhead at a lower level. We use the Trickle algorithm [41] for adaptive beaconing component, which is also part of our CB-CTP implementation. Component oriented design makes it possible to plug the adaptive beaconing component to S4 instead of its original periodic beaconing scheme.

Figure 6(d) shows the comparison of network traffic overhead



(a) Energy consumption at an application rate of 0.2 pkt/s in a 80 nodes network.

(b) Average per node power consumption at different application rates and network sizes.

(c) Average PRR at different application rates and network sizes.

(d) Network traffic generated with periodic and adaptive beaconing.

Figure 6: Performance comparison of CB and monolithic implementations of S4 and BoXMAC-2. (a)-(c) display the results for the CB stack with piggy-backed link quality. (d) shows the effect of adaptive beaconing with the CB stack.

between periodic beaconing and adaptive beaconing schemes for S4/BoXMAC-2 stack in a network consisting of 80 nodes. BoXMAC-2 operates at a duty cycle of 2.2 % with a periodic channel sensing duration of 11 ms. The periodic beaconing interval is set to the default value of 17.5 s. For the case of adaptive beaconing, we can observe a sharp peak in the beginning while all the nodes compete for the channel at the highest beacon transmission rate of 128 ms. This rate gradually slows down after every successful transmission and ultimately settles down to a value of 8.3 minutes. The settling duration, however, depends on the number of nodes in a network. In the presented results, the network topology is established within an interval of 200 s and, therefore, the network traffic decrease drastically afterwards. This gives rise to high amount of energy savings in S4. Applying adaptive beaconing to S4, as a representative example, validates the idea of composing efficient protocol behavior by simply combining different components.

4.2 Composite MAC/Routing Stack and Run-time Adaptivity

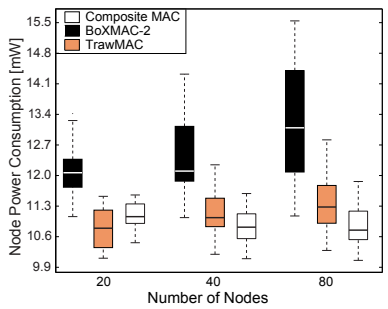
We have studied the performance characteristics of different combinations of protocols under varying application transmission rates (0.01 pkt/s and 0.2 pkt/s), network sizes (20, 40 and 80 nodes) and, duty cycle values (9.9 %, 4.4 %, 2.2 %, 1.1 %, 0.6 %) in order to identify the performance trends. These experiments enable building hypothesis for static optimization decisions. For instance, we observe that TrawMAC, though more energy efficient, has a lower application packet success ratio compared to BoXMAC-2 regardless of the network size. This is because of TrawMAC's lower channel utilization and propensity towards deafness. The remaining preamble time, advertised in each micro-frame, forces the non-recipient nodes to sleep till the end of a particular transmission. This creates adverse effect over hidden terminals since they will start their transmissions according to the sleep schedule of their potential recipients, which, on the other hand, would be sleeping to avoid any overhearing. TrawMAC drastically reduces idle listening and overhearing for broadcast transmissions and shows considerable energy efficiency compared to BoXMAC-2, but it also creates deafness among unicast transmissions which ultimately results in reduced application PRR and increased retry counts. These observations are also evident from the graphs presented in Figure 7 where the power consumption is categorized into network setup time (with heavy broadcast transmissions) and stable operating regime (with more unicast transmissions).

Based on these experimental observations, we deduce the following. In order to achieve high packet delivery ratio with least energy consumption, broadcast transmissions should be attempted with TrawMAC and if the first unicast packet transmission fails with TrawMAC, it should later be attempted through BoXMAC-2. Component based implementation of protocols allows composing a composite MAC stack which may benefit from such a run-time decision as part of CONFab (cf. Section 3.5.3, Figure 4). In order to allow TrawMAC and BoXMAC-2 behaviors to coexist simultaneously, we define common data structures for storing neighborhood sleep schedule information and a field to identify the type of the MAC scheme as part of the DM component. Figure 7 shows the power consumption comparison and the application PRR of BoXMAC-2, TrawMAC and a Composite MAC. The duty cycle and the channel sensing duration for all the MAC protocols are set to be 2.2 % and 11 ms, respectively. CTP is used as the underlying routing protocol. Figure 7 shows that the Composite MAC gives the most energy efficient operation while achieving the highest PRR.

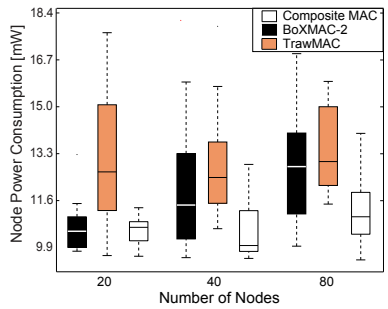
We have also experimented with variable application traffic data rates captured as a composite CONFab scenario (cf. Section 3.5.3). We have carried out an experiment where the application traffic rate is changed during the experiment. After the topology is established, the network starts with a low application traffic rate of 0.01 pkt/s for a duration of 300 s. Later on, the application traffic rate is changed to 0.2 pkt/s for 200 s, which falls back to 0.01 pkt/s for a duration of 300 s. Figure 8(a) shows the energy consumption per node for a successful delivered packet during the low traffic conditions. We can observe that Composite-MAC/CTP adapts its behavior and is thereby able to achieve 5 %, 10 % and 3 % performance gains over monolithic MAC stacks for 20, 40 and 80 nodes network, respectively. For low application rates, the PRR stays similar for all the network sizes so we see satisfying performance of TrawMAC. Figure 8(b) shows the energy consumption per node for a successful delivered packet for low and high traffic rate conditions. The Composite MAC stack when estimating high traffic conditions behaves like BoXMAC-2, whereas it adapts TrawMAC like behavior during the low traffic conditions. For high traffic rates and larger network sizes, the PRR of TrawMAC drops by 50 % due to deafness problem and high amount of retransmissions in the network. Composite MAC adapts BoXMAC-2 like behavior when the packet retransmissions increase. Compared against the monolithic alternatives the Composite MAC stack achieves performance gains of 25 %, 16 % and 6 % for networks consisting of 20, 40 and 80 nodes, respectively.

4.3 Protocol Stack Composition using Performance Prediction

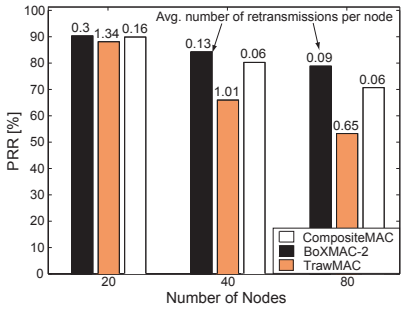
We evaluate the protocol stack composition suggested by CONFab for a scenario that is not already present in the KB. We model it after the habitat monitoring testcase [42] with the network size of 50 nodes and application data rate of 0.02 pkt/s. In order to



(a) Per node power consumption comparison for an application rate of 0.2 pkt/s during network setup time.

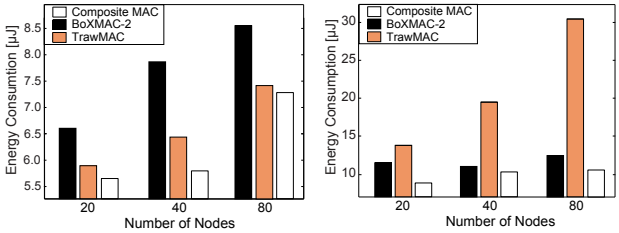


(b) Per node power consumption comparison for an application rate of 0.2 pkt/s during stable operating regime.



(c) Average PRR comparison for an application rate of 0.2 pkt/s during the stable operating regime.

Figure 7: Comparison of BoXMAC-2, TrawMAC and Composite MAC. CTP is used as the routing protocol.



(a) Low-traffic rate. (b) Low+high traffic rate.

Figure 8: Average energy consumption per node per successfully delivered packet for a network with variable traffic rate using CTP as the routing protocol.

recommend protocol stack for a particular scenario CONFab, first, discretizes the scenario characteristics into corresponding ontology classes to enable quantitative reasoning. If inputs are already formatted as class values (e.g., medium network size with low application rate) this step is skipped. Then, the framework queries the KB for performance results in similar deployments, i.e., scenarios with characteristics falling into the same or nearby classes. In our case, there are records for 40 and 80 node deployments with low and high application rates of 0.01 and 0.2 pkt/s. Based on these, CONFab formulates a list of viable stack configurations. Next, the framework combines quantitative performance results for these scenarios in a weighted manner in order to obtain the prediction estimates. We have introduced the weighted approach, since solely using the quantitative ontology-based reasoning makes the framework sensitive to definition of the discretization class boundaries. However, if these are wisely defined on the scenario specific basis, this part of the reasoning process can be omitted.

Figure 9(a) shows the predicted power consumption and the application level PRR at the suggested duty cycle values for different protocol stacks during the stable operating regime (after the network topology has been established). These values are shown under *Stack*, *DC* and *Expected PRR* and *P* columns of the table in Figure 9(a). It is worth noting that besides the monolithic combination of protocols, CONFab also suggests Composite MAC together with CTP. The actual performance metrics at the suggested duty cycle values are also listed. The results obtained from actual deployments of the recommended stacks are shown in the last column in the table. Figures 9(b) and Figure 9(c) show the performance (average network power consumption and application PRR) of these and other protocol stack configurations for the habitat monitoring scenario. It can be observed from the figures that CONFab is able to predict the protocol stack performances with high accuracy. In Figure 9(a), the best results for the CB and monolithic protocol stacks are highlighted. We may observe that the most recommended protocol stack which includes Composite MAC at a duty cycle of 4.2 % overall outperforms the best monolithic protocols based stack (CTP/BoXMAC-2) by showing 37 % improvement in energy efficiency with a slightly lower PRR.

5. CONCLUSIONS

We have presented a framework, called CONFab, which aims at automating the development cycle of a WSN protocol stack. CONFab treats the component based composition of a protocol stack as an optimization problem, which is solved based on the user-defined goals, deployment experiences and expert inputs. The components, CONFab operates on, are configurable software modules of various granularities, which include both monolithically realized protocols and their decomposition. We also discuss the abstractions and possible decomposition of WSN MAC and routing protocols that are appropriate for autonomous reasoning and leads to well-performing scenario-specific stacks.

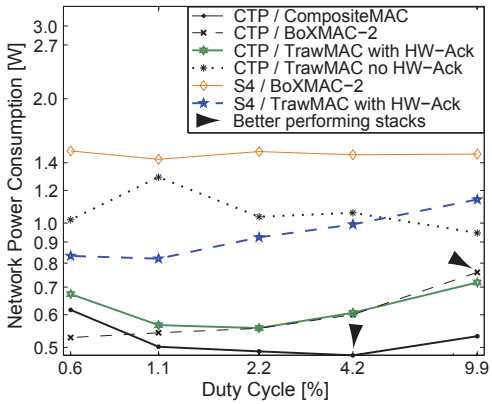
Extensive evaluation studies for 700+ hours have been conducted on the Indriya TelosB testbed to observe the performance characteristics and the component behavior of MAC and routing protocols. Our results show that CONFab helps in composing efficient component based protocol stacks according to the application scenario. For instance, we have observed over 37 % increase in network lifetimes without losing PRR using a composite MAC-routing stack compared to classical layered implementation approach. The performance achieved can be further enhanced by extending the KB and additional plug-ins allowed by the modular architecture of CONFab. As an example, we have presented and further validated that the performance feedback from similar deployment scenarios allows CONFab to compose optimized protocol stack and parameter settings. The ontology and plug-ins of CONFab in future can be enriched to capture more details on WSNs and improve the reasoning quality. Obviously, the framework will also benefit from additional feedback on existing and new sets of components. We are currently exploring the possibilities for public release of the software and the gathered data traces.

Acknowledgment

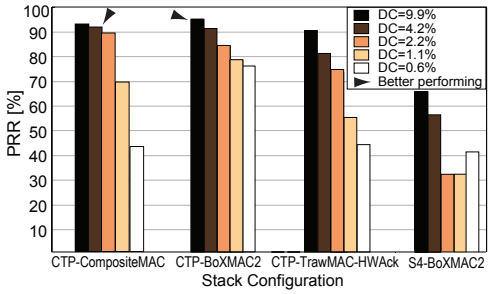
We would like to thank partial financial support from DFG through UMIC Research Centre and European Union through 2PARMA project. We also thank Xi Zhang for useful discussions.

ID	Stack		DC	Expected		Real Perf.	
	ROU	MAC	[%]	PRR [%]	P [W]	PRR [%]	P [W]
► c10	CTP	Composite MAC	4.2	>90	<0.6	92	0.48
c11	CTP	Composite MAC	9.9	>90	<0.6	93	0.53
c09	CTP	Composite MAC	2.2	>80	<0.6	90	0.50
► p12	CTP	BoXMAC2	9.9	>90	<0.8	95	0.76
p20	CTP	TrawMAC	9.9	>90	<0.8	91	0.72
p14	CTP	BoXMAC2	4.2	>90	<0.7	92	0.60
p18	CTP	BoXMAC2	2.2	>80	<0.6	84	0.56
p16	CTP	TrawMAC	4.2	>80	<0.6	81	0.61

(a) The CONFab predicted estimates and the actual performance metrics for sample stacks.



(b) Network power consumption of protocol stacks at different duty cycles for the target scenario.



(c) Application PRR of protocol stacks at different duty cycles for the target scenario.

Figure 9: Actual performance and predictions by CONFab on the pool of monolithic and composite CB stacks.

6. REFERENCES

[1] T. Arampatzis, J. Lygeros, and S. Manesis, "A survey of applications of wireless sensors and wireless sensor networks," in *Proc. of Mediterranean Conference on Control and Automation*, 2005.

[2] K. Balasubramanian *et al.*, "Developing applications using model-driven design environments," *Computer*, vol. 39, no. 2, pp. 33–40, 2006.

[3] A. Pinto *et al.*, "System level design paradigms: Platform-based design and communication synthesis," *ACM Tran. on Design Automation of Electronic Systems*, vol. 11, no. 3, pp. 537–563, 2006.

[4] S. O'Malley and L. Peterson, "A dynamic network architecture," *ACM Tran. on Computer Systems*, vol. 10, no. 2, pp. 110–143, 1992.

[5] D. G. Messerschmitt, "Rethinking Components: From Hardware and Software to Systems," *Proc. of the IEEE*, vol. 95, no. 7, pp. 1473–1496, 2007.

[6] P. Levis and D. Gay, *TinyOS Programming*. Cambridge University Press, 2009.

[7] D. Nardi *et al.*, "An introduction to description logics," *The description logic handbook: theory, implementation, and applications*, pp. 1–40, 2003.

[8] "Indriya telosb testbed at national university of singapore." `http://indriya.comp.nus.edu.sg/motelab/html/index.php`.

[9] B. Yahya and J. Ben-Othman, "Towards a classification of energy aware MAC protocols for wireless sensor networks," *Wireless Comm. and Mob. Computing*, vol. 9, no. 12, pp. 1572–1607, 2009.

[10] K. Akkaya and M. Younis, "A survey on routing protocols for wireless sensor networks," *Ad Hoc Networks*, vol. 3, no. 3, pp. 325–349, 2005.

[11] K. Klues, G. Hackmann, O. Chipara, and C. Lu, "A Component Based Architecture for Power-Efficient Media Access Control in Wireless Sensor Networks," in *Proc. of SenSys*, 2007.

[12] J. Ansari, X. Zhang, O. Salikeen and P. Mähönen, "Enabling Flexible MAC Protocol Design for Wireless Sensor Networks," in *Proc. of the IEEE Int. Conf. on Wireless On-demand Network Systems and Services*, 2011.

[13] J. He, J. Rexford, and M. Chiang, "Don't optimize existing protocols, design optimizable protocols," *SIGCOMM CCR*, vol. 37, pp. 53–58, 2007.

[14] K. Rerkrai, J. Riihijärvi, M. Wellens and P. Mähönen, "Unified Link-Layer API Enabling Portable Protocols and Applications for Wireless Sensor Networks," in *Proc. of ICC*, 2007.

[15] J. Il Choi, M. A. Kazandjieva, M. Jain, and P. Levis, "The case for a network protocol isolation layer," in *Proc. of SenSys*, 2009.

[16] X. Lin, N. B. Shroff, and R. Srikant, "A tutorial on cross-layer optimization in wireless networks," *IEEE JSAC*, vol. 24, pp. 1452–1463, 2006.

[17] R. Braden, T. Faber, and M. Handley, "From protocol stack to protocol heap: role-based architecture," *SIGCOMM CCR*, vol. 33, no. 1, pp. 17–22, 2003.

[18] C. T. Ee *et al.*, "A modular network layer for sensornets," in *Proc. of USENIX OSDI*, 2006.

[19] M. Ahmed et al., "A component-based architecture for cognitive radio resource management," in *Proc of CrownCom*, 2009.

[20] M. Chiang, S. Low, A. Calderbank, and J. Doyle, "Layering as optimization decomposition: A mathematical theory of network architectures," *Proc. of the IEEE*, vol. 95, no. 1, pp. 255–312, 2007.

[21] C. Ivan, V. Dadarlat, and K. Pusztai, "Managing complexity of designing routing protocols using a middleware approach," in *Proc. of CAiSE*, 2002.

[22] A. Köpke, V. Handziski, J. Hauer, and H. Karl, "Structuring the information flow in component-based protocol implementations for wireless sensor nodes," in *Proc. of WIP EWSN*, 2004.

[23] J. S. Baras and H. Huang, "Component based routing: A new methodology for designing routing protocols for MANET," in *Proc. of ASC*, 2006.

[24] D. McGuinness, F. Van Harmelen, *et al.*, "OWL web ontology language overview," *W3C recommendation*, vol. 10, pp. 2003–04, 2004.

[25] H. Knublauch, R. Fergerson, N. Noy, and M. Musen, "The Protégé OWL plugin: An open development environment for semantic web applications," *The Semantic Web–ISWC*, pp. 229–243, 2004.

[26] D. Martin *et al.*, "OWL-S: Semantic markup for web services," *W3C Member Submission*, vol. 22, pp. 2004–07, 2004.

[27] R. Jurdak, C. Lopes, and P. Baldi, "A framework for modeling sensor networks," in *Proc. of the Building Software for Pervasive Computing*, 2004.

[28] B. Y. Alkazemi and E. A. Felemban, "Towards a framework for engineering software development of sensor nodes in wireless sensor networks," in *Proc. of SESENA*, 2010.

[29] A. Taherkordi, F. Loiret, R. Rouvoy, and F. Eliassen, "A Generic Component-Based Approach for Programming, Composing and Tuning Sensor Software," *The Computer Journal*, vol. 54, no. 8, 2011.

[30] K. Whitehouse, F. Zhao, and J. Liu, "Semantic streams: a framework for composable semantic interpretation of sensor data," in *Proc. of EWSN*, 2006.

[31] R. Sugihara and R. Gupta, "Programming models for sensor networks: A survey," *ACM Transactions on Sensor Networks*, vol. 4, no. 2, p. 8, 2008.

[32] E. Meshkova, K. Rerkrai, J. Riihijärvi, and P. Mähönen, "Optimizing component-oriented systems: A case study in wireless sensor networks," in *Demonstration paper at SIGCOMM*, 2008.

[33] J. Polastre *et al.*, "A unifying link abstraction for wireless sensor networks," in *Proc. of SenSys*, 2005.

[34] M. Avvenuti *et al.*, "Increasing the efficiency of preamble sampling protocols for wireless sensor networks," in *Proc. of WiCOM*, 2006.

[35] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks," in *Proc. of SenSys*, 2006.

[36] X. Zhang, J. Ansari and P. Mähönen, "Traffic Aware Medium Access Control Protocol for Wireless Sensor Networks," in *Proc. of MobiWac*, 2009.

[37] O. Gnawali *et al.*, "Collection Tree Protocol," in *Proc. of SenSys*, 2009.

[38] Y. Mao *et al.*, "S4: small state and small stretch routing protocol for large wireless sensor networks," in *Proc. of NSDI*, 2007.

[39] D. Tsarkov and I. Horrocks, "FaCT++ description logic reasoner: System description," in *Proc of. Automated Reasoning Conference*, 2006.

[40] A. Dunkels, F. Osterlind, N. Tsiftes, and Z. He, "Software-based on-line energy estimation for sensor nodes," in *Proc. of SenSys*, 2007.

[41] P. Levis *et al.*, "Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks," in *Proc. of the NSDI*, 2004.

[42] R. Szewczyk *et al.*, "An analysis of a large scale habitat monitoring application," in *In Proc. of SenSys*, 2004.