UAV based SDN system for wireless sensor networks

ABSTRACT

Abstract goes here.

1. INTRODUCTION

Why we are SDN ?

Software defined network is able to support flexible network programmability by using programmable data plane and centralized network controller.

OpenFlow focus on wired networks.

Challenges and opportunities of SDN for WSN:

Challenges: Limited resources of WSN nodes:

- energy
- processing
- memory
- communication

Opportunities:

- Improve resource reuse
- Implement node retasking
- Node and network management
- Enable experiments with new protocols

Why and How we can implement AI ?

How we combine Ai with other applications?

AI

AI systems have been improving, and new advances in machine intelligence are creating seamless interactions between people and digital sensor systems.

In sensor systems, applications can be found for a variety of tasks, including selection of sensor inputs, interpreting signals, condition monitoring, fault diagnosis, machine and process control, machine design, process planning, production scheduling, and system configuring. Some examples of specific tasks undertaken by expert systems are: * Assembly * Automatic programming * Controlling intelligent complex vehicles * Planning inspection * Predicting risk of disease * Selecting tools and machining strategies * Sequence planning * Controlling plant growth.

.

The tools and methods described have minimal computation complexity and can be implemented on small assembly lines, single robots, or systems with low-capability microcontrollers. These novel approaches proposed use ambient intelligence and the mixing of different AI tools in an effort to use the best of each technology. The concepts are generically applicable across many processes.

minimum energy, data loss, reliability, robustness, etc., in place during the design and operation of wireless sensor networks

a specific set of protocols for medium access, localization and positioning, time synchronization, topology control, security and routing are identified based on the current configuration of the network, the requirements of the application and the topology of their deployment.

2. RELATED WORK

2.1 Software Defined Wireless Sensor Networks

Existing SDN for WSN:

- Flow-Sensor
- Sensor OpenFlow
- SDWN
- TinySDN
- SDN-WISE

All of these are evaluated by simulations

Flow-Sensor [MahmudandRahmani2011], Sensor OpenFlow [Luoetal.2012] SDWN [Costanzo et al. 2012] TinySDN [de Oliveira et al. 2014] SDN-WISE [Galluccio et al. 2015]

2.2 Applications for Wireless Sensor Networks

3. ARCHITECTURE

The architecture of the UAV based SDN system for wireless sensor networks.

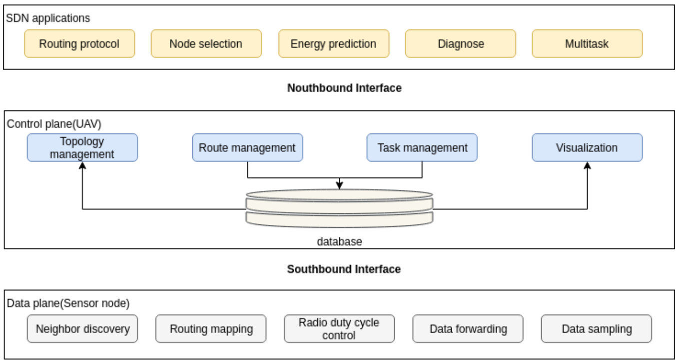

Figure 1: Architecture of the system.

Listing 1: An example of deploy routing algorithm

```
topology = get_topology();
//calculate flowtable for each node
//based on topology
for(node in nodeset){
   node.flowtable =
      calculate_flowtable(topology);
}
//set flowtable for each node
for(node in nodeset){
   UAV fly to node;
   for(flow in node.flowtable)
        set_route(flow);
}
```

Listing 2: An example of AI selection and Mutitasks

```
AI_Multitasks(taskset){
   create_scheduler();
   scheduler.create_buffer();

   for(task in taskset)
      scheduler.task_buffer_add(
         task,
         defaultset);

   scheduler.task_update();
   ...
   ...
   for(task in scheduler.task_buffer){
      data = get_collected_data();
      nodeset =
         SRSSS_selection(dataset);
      scheduler.task_buffer_update(
         task,
         nodeset);
   }
   scheduler.task_update();
}
```

4. APPLICATIONS

4.1 Overview

Traditional applications can not achieve complicated and efficient goals due to the limited processing power and memory space of sensors.

In XX, applications for wireless sensor networks are inspired by greater potential with the UAV based SDN controller. The central controller helps sensors execute complex calculations such as AI model training, as well as store global information. Besides, UAVs have flexible features and can deploy tasks to sensors by one-hop communication directly. Thus it enables the sensor network to achieve much more intelligent applications.

In XX, applications can be found for a variety of purposes, including routing, AI node selection, AI energy prediction, multi-tasks and network diagnosis. We design all these applications and provide easy-to-use interfaces to users as in Table 1.

Table 1: System API

Structure && Function	**Description**
	Sensor Control Interface
struct node	Sensor node structure
struct nodeset	A set of sensor nodes
struct neighbor_list	Neighbor infomation
struct energy_item	Energy statistic information
struct flow_table	Flow table
struct duty_cycle_table	Duty cycle control table
struct sensor_enable_table	All the nodes's states. Node state: {on,off}
switch_node(node,state)	Turn on or turn off the node
get_node_info(node)	Get node's information, including node's position, duty cycle, power, etc.
set_node_attr(node,attrTag,value)	Set node attribute, including duty cycle, radio strength, etc.
get_neighborlist(node)	Get the neighbor list of a node
	UAV Application Interface
	Routing
get_topology()	Get the topology of the network
get_flow_table(node)	Get the flow table of a node
set_flow(flow,node)	Set the flow of a node
	AI Node selection
nodeset simple_selection(nodeset)	Select sensor set by location information
nodeset SRSSS_selection(dataset)	Select sensor set by AI algorithm based on sensing data
	AI Energy Prediction
model_selsct(modeltype)	Select an AI model
model.train(dataset,ratio)	Train an AI model with learning ratio on the data set
model.test(dataset)	Test the AI model on the data set
model.predict(node)	Do the energy prediction for a node
	Multi-tasks
create_scheduler()	Create a task scheduler
scheduler.create_buffer()	Create a task buffer
scheduler.task_buffer_add(task,nodeset)	Add a new task to task buffer
scheduler.task_buffer_remove(task)	Remove a new task to task buffer
scheduler.task_buffer_update(task,nodeset)	Update a task to task buffer with a new nodeset
scheduler.task_update()	Schedule the added or removed tasks in the buffer
	Diagnosis
detect()	Detect problematic region with probes
get_topical_topology(nodeset)	Construct topical topology
diagnose_network(topology,nodeset)	Diagnose the failure nodes or lossy links

4.2 Routing

one round
cluster header

Table 2: Flow Table

Header Fields	Counters	Actions

Table 3: Header Fields

Ingress port	Ether Source	Ether Dst	IP src	IP dst

Actions:

- Forward
- Drop
- Report
- Forward
- Drop
- Report
- Drop
- Report

4.3 Network Diagnosis

4.3.1 Motivation

Sensor nodes are provisioned with low-capacity batteries and will run out of energy in the end. Besides, uncertain environmental factors will lead to the failure of communications. Upon these two factors, there occurs network faults such as failure nodes and lossy links from time to time.

Network fault diagnosis is then designed to help network administrators monitor the network operational status and maintain a sensor network system. The key idea of the existing works on fault diagnosis is to collect running information from nodes and deduce root causes of network exceptions. The running information is collected by a mechanism of probing.

We implement the network diagnosis process as an application in our XX system. Since in XX we have our mobile UAV as a controller, a more flexible way is to infer the suspicious nodes of the network fault in traditional way first and then set the UAV to check out the fault sources.

4.3.2 Design

We first introduce a state-of-the-art algorithm named DID [**?**], which is a directional diagnosis approach. We utilize the node tracing module and the tracing collection module of DID to infer the suspicious nodes in our network diagnosis application, and then set our UAV controller to confirm the network elements being faulty.

The node tracing module is conducted in each sensor node. Every time a packet arrives at a node, it counts the source of the packet . The tracing collection module is set in the UAV controller. When network exception is detected, it gathers the tracing information in the tracing module of the relevant nodes and infers a suspicious node set. Next we set the UAV to fly through the node set and check each node first to diagnose the failed nodes. Then UAV collects the neighbor lists of all the nodes in the suspicious node set. With the collected neighbor lists, the UAV controller reconstructs the topical topology and compares it with the default topology to find the failed links.

Compared to DID, the diagnosis application in XX releases the complicated inference computations and can achieve accurate diagnosis since the UAV can fly to the sensors to confirm the network faults. XX can realize the four types of fault sources the same as DID:

- Node failure. This network failure is caused by the node itself.
- Link failure. This network failure is caused by the communication links between nodes, mainly relating to traffic flow in networks.
- Temporary failure. This network failure is caused

by complex interior or exterior interferences and quick self-recovery

- Multiple failures. This network failure is caused by multiple failures above.

4.4 AI Node Selection

4.4.1 Motivation

It is inevitable that there will be a part of redundant sensors when deploying a practical wireless sensor network. These redundant nodes have overlaps of observation regions, and what makes the matter worse is that redundant nodes may cause great communication interference. Therefore it is significant to select proper sensors to avoid data redundancy and save the sensor network energy consumption.

In XX, we provide the node selection application to users. The SDN controller executes the selecting algorithm and send the control instructions to activate the selected nodes.

4.4.2 Design

Our XX system provide two main node selecting methods: greedy selection algorithm and SRSSS algorithm. This application will be extended to more elegant algorithms in our future work.

Greedy selection algorithm. We first provide a simple method to select the redundant nodes by a greedy selection algorithm, as described in Alg. 1. The key idea is to select nodes as less as possible to coverage the whole area based on the location and sensing range.

We implement the greedy selection algorithm in XX. The evaluations in section 6 show it greatly saves the sensors' energy and thus prolongs the network lifetime.

Algorithm 1 Greedy Selection Algorithm

```
Input: Sensor set N, Selected set M, Target area Ω, Covering area Φ;
Initialize : M = ∅, Φ = ∅
while M ≠ N do
    if Φ = Ω then
        break; \\ Selected set has been found
    end if
    if ∀n_i ∈ (N − M) : range(n_i) ⊂ Φ then
        break;\\ Cannot cover the target area;
    end if
    Find n_i : argmax(Φ ∩ range(n)), n_i ∈ (N − M);
    Φ = Φ ∪ n_i
end while
Output: M;
```

Spatially regularized streaming sensor selection (SRSSS). To realize more intelligent and effective sensor selection, we introduce a state-of-the-art AI algorithm named spatially regularized streaming sensor selection (SRSSS) proposed in [1].

Different from the greedy selection algorithm, SRSSS is a multi-variate interpolation framework and focuses on selecting a subset of sensors in a streaming scenario to minimize collected information redundancy.

Traditional wireless sensor network is not suitable to implement an AI selection approach due to the limited computational capability of sensors. Some work use the database to collect data and make decisions by multi-hop communications. However, in this way the network will use up a great deal of energy. In our XX, the UAV fly through the nodes to pick up the collected data and executes the computations. Then it sends the control instructions to the nodes by one-hop communication and greatly saves the network energy.

The aim of SRSSS is to optimize its objective function which is an equation given certain constraints of collected information, location and energy consumption. The objective function is formulated as:

$$
\begin{aligned}
&(W_{k+1}, z_{k+1}) \\
&= arg \min_{W,z} \sum_{i=1}^{k} \mu^{k-i} \| X_k^i D_z W(I - D_z) - X_k^i (I - D_z) \|_2^2 \\
&+ \alpha \sum_{i,j=1}^{n} \|y_i - y_j\|_2 |W_{i,j}| - \beta \sum_{i,j=1}^{n} \|y_i - y_j\|_2 z_i z_j \\
&+ \lambda \|W\|_F^2 \\
&s.t. z = [z_1, ..., z_n] \in \{0,1\}^n, c^T z \le P
\end{aligned}
\tag{1}
$$

The first term in (1) is to minimize the prediction error of the collected data and the following two terms incorporate spatial information. The last term constrains the complexity of the learned matrix W. The energy constraint is controlled by the inequality in (1). Because of the limitation of length, we leave out all the details. The meaning of the parameters and the mechanism of SRSSS can be seen in [1].

With the AI sensor selection process, XX becomes a smarter and adaptive sensor systems.

4.5 Multi-tasks

4.5.1 Motivation

Wireless sensor networks (WSN) generally comprise of a group of spatially dispersed sensors. In a wireless sensor network, sensor nodes are equipped with various types of sensors monitoring and recording environmental conditions like temperature, sound, sunlight, humidity, etc.

A given sensing task involves multiple sensors to achieve a certain quality-of-sensing. Generally, an efficient task scheduling for the nodes is that nodes are able to perform multiple tasks simultaneously. For example, sensors deployed in a grove are assigned tasks to collect sunlight, temperature and humidity data and these tasks require different number of nodes with respective sensing range, rate and duration. However, traditional sensor networks are not suitable to conduct this multi-tasks due to the limitations of computation complexity for task arrangement of each node.

In our XX system, we implement the multi-tasks application with the help of the central controller. The SDN controller maintains programmable task scheduling and management modules while sensor nodes are loaded with interfaces to receive task control instructions.

4.5.2 Design

A deployed wireless sensor networks are usually assigned with different data collection requirement. In XX, we design and implement multi-task application and provide easy-to-use interfaces to users.

When a user assign a task to XX, the UAV controller will first check out the energy and storage constraints of the required sensor set, as described in Alg. **??**. If the task requirement exceeds the capacity of the sensor set, it will be sent to a task queue; Otherwise it will be put into the task buffer to conduct.

Algorithm 2 Sensor Constraint Detection

```
 1: Input: Sensor set N, Selected set M, Target area Ω, Covering area Φ;
 2: Initialize : M = ∅, Φ = ∅
 3: while M ≠ N do
 4:    if Φ = Ω then
 5:       break; \\ Selected set has been found
 6:    end if
 7:    if ∀n_i ∈ (N − M) : range(n_i) ⊂ Φ then
 8:       break;\\ Cannot cover the target area;
 9:    end if
10:    Find n_i : argmax(Φ ∩ range(n)), n_i ∈ (N − M);
11:    Φ = Φ ∪ n_i
12: end while
13: Output: M;
```

A sensor node may have different sensing ranges for different tasks.

There are several practical requirements.

Different tasks have different requirements, including time, sensing range, sensing ratio, etc.

For example tasks like sunlight collection only need to be carried out during the daytime.

Our system provide a task scheduling to

Sensors are usually assigned multi-tasks.

Sensors are assigned tasks to monitor a specific area.

Different tasks have different requirements, i.e.

- **Node set.** Users can assign tasks to
- **Sensing rate.**
- **Sensing range.** The maximum distance that a node can detect.
- **Sensing duration.** The sensing time from start to end. There is no need to collect sunlight data at night.

Task scheduler do the arrangement.

Task buffer.

Task queue.

Scheduling table.

...

5. IMPLEMENTATION

Implementation goes here.

6. EVALUATION

Evaluation goes here.

7. CONCLUSION

Conclusion goes here.

8. REFERENCES

[1] Li, C., Wei, F., Dong, W., Wang, X., Yan, J., Zhu, X., Liu, Q., and Zhang, X. Spatially regularized streaming sensor selection. In *Thirtieth AAAI Conference on Artificial Intelligence* (2016).

Table 4: Task Buffer

Task ID	Node set	Sensing rate	Sensing range	Sensing duration
Task ID	Node set	Sensing rate	Sensing range	Sensing duration
...	...	...	...	...
Task ID	Node set	Sensing rate	Sensing range	Sensing duration
...	...	...	...	...