ARL-TR-8593 ● Nov 2018

US Army Research Laboratory

A Vision toward an Internet of Battlefield Things (IoBT): Autonomous Classifying Sensor Network

by John Zhu, Egan McClave, Quan Pham, Sujay Polineni, Sam Reinhart, Ryan Sheatsley, and Andrew Toth

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator

ARL-TR-8593 ● Nov 2018

US Army Research Laboratory

A Vision toward an Internet of Battlefield Things (IoBT): Autonomous Classifying Sensor Network

by John Zhu, Egan McClave, Quan Pham, and Sam Reinhart
US Army Educational Outreach Program–College Qualified Leaders, Adelphi, MD

Sujay Polineni
US Army Educational Outreach Program–Science and Engineering Apprenticeship Program, Adelphi, MD

Ryan Sheatsley and Andrew Toth
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
November 2018	Technical Report	20 June to 18 Aug 2017

4. TITLE AND SUBTITLE
A Vision toward an Internet of Battlefield Things (IoBT): Autonomous Classifying Sensor Network

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
John Zhu, Egan McClave, Quan Pham, Sujay Polineni, Sam Reinhart, Ryan Sheatsley, and Andrew Toth

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
ATTN: RDRL-CIN-T
2800 Powder Mill Road
Adelphi, MD 20783-1138

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-8593

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The focus of Internet of Battlefield Things (IoBT) is to provide situational awareness of the battlefield utilizing a network of interconnected sensors, actuators, and analytical devices. The US Army Research Laboratory's Tactical Network Assurance Branch developed a narrative of using IoBT to deploy multiple sensor nodes in an unknown or potentially hostile environment in which the system performs basic classification, identification of allies and adversaries, and inter-node communication via ad-hoc wireless network. Various types of sensors resulted in more robust classification due to multiple sensor data sources of complementary modalities. Sleeping algorithms extended sensor-node viability while maintaining network activity. Development of countermeasures to denial-of-service and distributed denial-of-service attacks were also explored in this work.

15. SUBJECT TERMS
Internet of Battlefield Things, IoBT, sensor networks, tactical networks, sensor security, sensor energy efficiency

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Andrew Toth
Unclassified	Unclassified	Unclassified	UU	37	19b. TELEPHONE NUMBER (Include area code) (301) 394-2746

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures v

List of Tables v

1. Introduction 1

2. Implementation 1

2.1 Cross-Referencing Sensors 1

2.2 Sleeping Algorithms 2

2.2.1 Geographic Adaptive Fidelity (GAF) Algorithm 3

2.2.2 Connected *k*-Neighborhood (CKN) and Energy Consumed Connected *k*-Neighborhood (EC-CKN) 5

2.2.3 Classifying Network Sleeping Algorithm 7

2.3 Data Analysis 7

2.3.1 Node-RED Testing 7

2.3.2 Clustering Algorithm 8

2.4 Security 9

3. Experimental Setup 10

3.1 Arduino UNO 10

3.2 Raspberry Pi 11

3.3 XBee Series 2 11

3.4 Sensors 12

3.5 Network Simulation 17

4. Results 19

4.1 Physical Prototype 19

4.2 Python Simulations 19

4.2.1 GAF Results 19

4.2.2 CKN Results 22

4.2.3 EC-CKN Results 25

5. Conclusion **26**

6. References **28**

List of Symbols, Abbreviations, and Acronyms **29**

Distribution List **30**

List of Figures

Fig. 1 Example of a virtual grid in GAF 3
Fig. 2 State transitions in GAF 4
Fig. 3 Network lifetime broken up into epochs 5
Fig. 4 Pseudo code for the CKN algorithm 6
Fig. 5 Pseudo code for the EC-CKN algorithm 6
Fig. 6 Node-RED flow for testing different clustering methods 8
Fig. 7 Example of protection against packet overflow (DoS attack) 10
Fig. 8 Arduino UNO board 11
Fig. 9 Zigbee mesh network layout 12
Fig. 10 Placement of nodes in the network 18
Fig. 11 Example of the GAF network. Type of node is shown by shape. Active nodes are green and sleeping nodes are blue. 20
Fig. 12 Lifetime of the GAF network 21
Fig. 13 Percentage of events detected in the GAF network over time 22
Fig. 14 Example of the CKN network. Green means active, blue means sleep, a purple ring means a node has sensed an event, and a purple circle means an event. 23
Fig. 15 Lifetime of the CKN network 24
Fig. 16 Percentage of events detected in the CKN network over time 24
Fig. 17 Example of the EC-CKN network. Green means active, blue means sleep, a purple ring means the node has sensed an event, and a purple circle means an event. 25
Fig. 18 Lifetime of the EC-CKN network 26
Fig. 19 Percentage of events detected in the EC-CKN network over time 26

List of Tables

Table 1 Sensors considered in the network 13
Table 2 Sensors used in the network: detectors 15
Table 3 Sensors used in the network: classifiers 16
Table 4 Sensors used in the network: communication 17

1. Introduction

The focus of Internet of Battlefield Things (IoBT) is to provide situational awareness of the battlefield utilizing a network of interconnected sensors, actuators, and analytical devices. Sensors could detect enemy movement and then relay that information in real time to analysts, enabling them to make tactical decisions on positioning, areas to avoid, or who is crossing a certain area. This capability would potentially save resources and Soldiers' lives, making IoBT an important topic for the US Army Research Laboratory's (ARL) Network Science Research Laboratory to investigate. The concept of Internet of Things (IoT) is to create a network of communication with any kind of device, from a car to a fridge. Translate this concept to a battlefield environment, and one can imagine the possibilities that IoBT can bring. There has been significant interest in IoBT devices from the Army to learn, develop, and take these ideas from the laboratory to the field.

For this effort, a group of summer students in ARL's Tactical Network Assurance Branch developed a narrative of using IoBT to deploy multiple sensors in an unknown or potentially hostile environment. We call this system the Autonomous Classifying Sensor Network. Sensors within the system perform basic classification, identifying whether an ally or adversary is present based on generated events, and communicate with each other using an ad-hoc wireless network. By using multiple sensors of different types, the classification results are more robust because they are from multiple sources of sensor data of varying modalities. In addition, to ensure a long network lifetime, the sensors employ a sleeping algorithm where nodes enter a low-power mode while maintaining network activity. Lastly, the effort focused on developing a countermeasure to denial-of-service (DoS) and distributed denial-of-service (DDoS) attacks, which are universal threats that can shut down networks.

2. Implementation

2.1 Cross-Referencing Sensors

Each sensor node was connected to an XBee Series 2 module,[1] which acted as the base layer for communication. We selected the XBee because of its low cost and similarity to tactical wireless radios versus other commonly available WiFi or Bluetooth devices. Our implementation had two types of nodes, a detector node and a classifier node. Detectors were used to detect whether an event has occurred in the general area of the network. The classifiers collected data that were later used to classify an event as an ally or adversary event. Our vision was to have the

detector nodes sense whether anyone has entered the field and relay that information to nearby classifier nodes to wake them up and start surveying the environment. The classifier nodes would continue to sense until either an event is triggered by an individual or the allotted time for sensing has ended. The information from the classifiers would then be sent to the base station either directly or through multihopping.

Detector nodes were equipped with a passive IR (PIR), ultrasonic, or vibration sensor, and classifier nodes were equipped with a radio-frequency identification (RFID), magnetometer, microphone, or camera sensor. To conserve energy, a sleep algorithm was implemented in our network so that the detector nodes would not be awake sitting idle at every timestep. Sensors were attached to either an Arduino UNO[2] or a Raspberry Pi[3] single-board computer. Arduino UNO devices were used with most of the sensors except for the camera and microphone, which required the additional computational resources offered by the Raspberry Pi. Both devices are described in detail in Sections 3.1 and 3.2.

Data collected by the classification sensors are sent to the base station, where they are aggregated into a single data entry. This aggregate data set represents the area sensed over the past few seconds. The data are then analyzed by a clustering algorithm to classify the profile of the individual who triggered the sensors to predict whether an ally or adversary triggered the event. This process is described in greater detail in Section 2.2.3.

2.2 Sleeping Algorithms

An important aspect of a wireless sensor network is network lifetime, which can be defined as either the time it takes for the last node to stop functioning (from failure or exhausting the power resources) or when the coverage or connectivity of the network reaches a certain connectivity threshold. If the network can sustain viable traffic longer, it decreases the need to replace or maintain the network on the battlefield, saving time and reducing the risk to Soldiers. Because increasing the battery capacity of each node is a costly solution, we propose implementing a sleeping algorithm to prolong the lifetime of the network. Nodes that are not actively scanning and are not vital to network connectivity can be put to sleep to save energy. These nodes can then be activated at a later time to take over the roles of nodes with less available energy. For our implementation, three different strategies were explored while measuring their lifetime and coverage.

2.2.1 Geographic Adaptive Fidelity (GAF) Algorithm

One of the sleeping algorithms we explored is the GAF algorithm developed in a joint effort by the Information Science Institute and University of California.[4] The algorithm determines which nodes should sleep by using a virtual grid based on geographic location. This virtual grid splits up nodes into separate groups, where all nodes in each group are considered equivalent with respect to its route. In other words, each node can assume the responsibility of another node as long as they are in the same group. In addition, any node in one group will be able to communicate with any node in an adjacent group. To ensure that any node from one grid can communicate with any node from an adjacent grid, the length of each block has to follow the relationship $r^2 + (2r)^2 \leq R^2$, where r is the length of each grid and R is the communication range. This equation simplifies to $r \leq R/\sqrt{5}$. Nodes in the same grid will then communicate with each other to decide which node in the group should stay awake by broadcasting their energy. If a node receives a broadcast message from another node that has more energy, it will go to sleep. Otherwise, if it receives a broadcast message from a node with equal or less energy, it will ignore that message. If it does not receive a message from a node with higher energy within a certain period of time, it will go into active mode.

Figure 1 demonstrates how the virtual grid works. In this example, nodes 2, 3, and 4 in grid B are considered equivalent. Thus, any of those nodes can relay packets between node 1 and node 5. Because there are two extra nodes that are redundant in grid B, the one with highest energy can remain activated, while the other two nodes sleep. When the node that has been awake begins to have less power than the other nodes in grid B, it can then sleep and allow one of the other two to take its place.

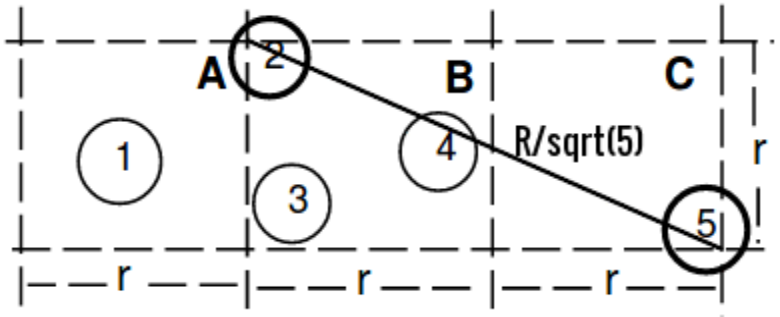


Fig. 1 Example of a virtual grid in GAF[4]

In addition, each node can be in one of three states: sleep, active, and discovery. The states are shown in Fig. 2. In discovery mode, each node turns on its radio and listens to energy broadcasts from its neighbor nodes, but does not turn on its sensors. It will stay in this state for a random period of time between zero and a

constant period T_d. In our experiment, this constant was set to 0.5 s. If the node has received a broadcast from a node with higher energy, then the node will go into sleep mode. Otherwise, if it does not receive a broadcast from a higher energy node, then it will go into active mode.

In sleep mode, the node will sleep for T_s seconds, which is a random amount between *enat*/2 to *enat*, where *enat* stands for estimated node active time. In this experiment, *enat* is set to half of the estimated remaining node lifetime. Once the random time period has passed, it will go into discovery mode for a random period of time between zero and T_d.

In active mode, the node will poll its sensors for information on the surroundings, as well as broadcast its energy every 0–T_d seconds. It will stay in this state for T_a seconds, which in this simulation is set to *enat* seconds. To prevent constant switching when *enat* becomes small, each node is set to stay awake for at least 10 s. However, if the node receives a discovery message from another node in its group that has a higher amount of energy, then it will promptly go into sleep mode.

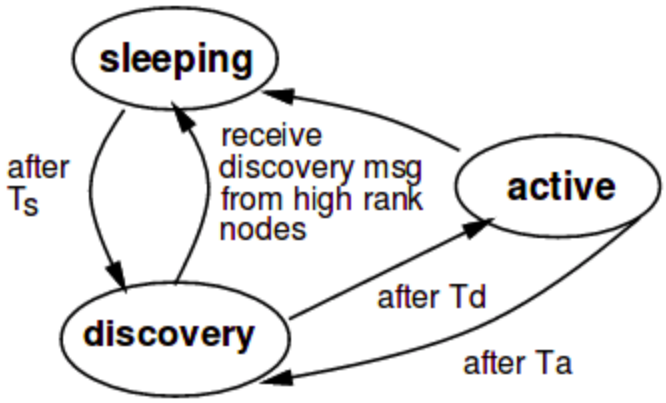


Fig. 2 State transitions in GAF[4]

Because our network contains a heterogeneous mixture of classifier and detector nodes, the GAF algorithm is implemented such that the detector nodes are considered the same type, while classifier nodes only react to broadcast messages of the same type. For example, detector nodes with PIR or ultrasonic sensors will be considered the same, while classifier nodes such as RFID or magnetometer are considered different types.

2.2.2 Connected *k*-Neighborhood (CKN) and Energy Consumed Connected *k*-Neighborhood (EC-CKN)

Compared to the GAF algorithm, which determines redundant nodes based on geographical locations, CKN[5] and EC-CKN[6] use the number of neighbors to determine the state of the node. These algorithms focus on maximizing the number of sleeping nodes while keeping a *k*-connected network. Each node determines its number of neighbors, and then decides whether these neighbors can assume its responsibilities. Not only does this algorithm focus on connectivity, but it also allows the user to change the robustness of the network by altering *k*. For example, if the network requires extra robustness to ensure a packet is not dropped when a node dies, then we can set the network to be a two-connected network. Thus, there are at least two routes available for each node.

Both algorithms operate in the same manner, as shown in Fig. 3. The network lifetime is broken up into epochs of period *T*, where each epoch is composed of the time for transmission and the time for running the sleep-scheduler algorithm. In each epoch, each node will poll its sensors to detect events and then transmit them. After transmission time has ended, each node will then execute a sleep scheduler algorithm to decide whether the node should be active or asleep in the next epoch.

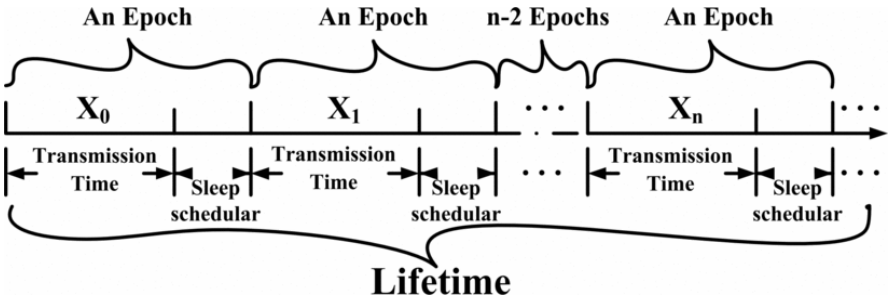


Fig. 3 Network lifetime broken up into epochs[6]

The CKN sleep scheduler algorithm, for which the pseudo code is shown in Fig. 4, works by first assigning a random rank to each node (Step 1). Each node would then broadcast its own rank, then listen and memorize the rank of its one-hop neighbors. After broadcasting, each node would then receive the ranks of its neighbors and store it in a buffer R_u (Step 2). Each node would then broadcast R_u, and receive the ranks of its neighbor's neighbors, giving the node information about its two-hop neighbors (Step 3). After transmitting all the messages, the node checks whether the number of single-hop neighbors it has or the number of its neighbor's neighbors is less than *k*. If the number is less than *k*, then to keep a *k*-connected network the node should remain awake (Step 4). Otherwise, if it potentially has neighbors that can assume its responsibilities, then it will go to sleep

if two conditions pass. First, each node calculates which of its neighbors have a higher rank than itself and stores them into a buffer called C_u (Step 5). It will then check whether it passes the two conditions stated in Step 6, which ensures that all nodes in C_u are connected by nodes with higher ranks than itself (Step 6).

(Run the following at each node s_u *)*

1. Pick a random rank $rank_u$;
2. Broadcast $rank_u$ and receive the ranks of its currently awake neighbors N_u. Let R_u be the set of these ranks.
3. Broadcast R_u and receive R_v from each $s_v \in N_u$.
4. If $|N_u| < k$ or $|N_v| < k$ for any $s_v \in N_v$, remain awake. Return.
5. Compute $C_u = \{s_v | s_v \in N_u \text{ and } rank_v > rank_u\}$;
6. Go to sleep if both the following conditions hold. Remain awake otherwise.
   - Any two nodes in C_u are connected either directly themselves or indirectly through nodes which is in the s_u's 2-hop neighborhood that have $rank_v$ larger than $rank_u$;
   - Any node in N_u has at least k neighbors from C_u.
7. Return.

Fig. 4 Pseudo code for the CKN algorithm[6]

As opposed to CKN, EC-CKN does not assign a random rank to the node each time the sleep scheduler algorithm is run, but instead measures its remaining energy and broadcast that instead as its rank. This ensures that nodes that have more energy will be more likely to be active, while those with less energy are more likely to sleep. Theoretically, this implementation will distribute energy consumption more evenly throughout the network, increasing the network lifetime. The pseudo code referenced for the EC-CKN algorithm is shown in Fig. 5.

(Run the following at each node s_u *)*

1. Get the information of current remaining energy $Erank_u$;
2. Broadcast $Erank_u$ and receive the energy ranks of its currently awake neighbors N_u. Let R_u be the set of these ranks.
3. Broadcast R_u and receive R_v from each $s_v \in N_u$.
4. If $|N_u| < k$ or $|N_v| < k$ for any $s_v \in N_v$, remain awake. Return.
5. Compute $E_u = \{s_v | s_v \in N_u \text{ and } Erank_v > Erank_u\}$;
6. Go to sleep if both the following conditions hold. Remain awake otherwise.
   - Any two nodes in E_u are connected either directly themselves or indirectly through nodes which is in the s_u's 2-hop neighborhood that have $Erank_v$ larger than $Erank_u$;
   - Any node in N_u has at least k neighbors from E_u.
7. Return.

Fig. 5 Pseudo code for the EC-CKN algorithm[6]

2.2.3 Classifying Network Sleeping Algorithm

While both of these algorithms would work well for a homogeneous ad-hoc network with a high sampling rate, our network requires a sleeping algorithm for a heterogeneous network with classifier and detector nodes that have a low sampling rate of 0.5 Hz. Thus, for this implementation, we implemented a simple sleep algorithm on the prototype. There are several assumptions we made about the network for this algorithm to be effective. One is that our base station will always be on. Another is that the detector nodes will be used to detect events and send an event message to classifier nodes. The information from detector nodes does not need to make it to the base station. Lastly, the classifier nodes will be mostly sleeping, but will be able to detect event messages from other classifier nodes. If they do receive one, then they will start polling the environment for events and send information to the base station.

Because we have a low sampling rate of 0.5 Hz, we have all detector nodes wake up every 2 s to poll the environment for events using different sensors. If a sensor senses an event, it will then broadcast that it has detected an event, which will be received by the classifier nodes. Otherwise, if it doesn't sense anything, it does nothing. Once a sensor is done executing and transmitting, it will then sleep for 2 s. Thus, the active duty cycle of each detector node is very low and does not waste energy reporting when nothing is sensed.

Each classifier node will check whether it has received a message from a detector node every 2 s. If a message is not received, the classifier node will sleep for 2 s. If the classifier node does receive a message, it will then stay on for 4 s and continuously poll its environment. If the classifier node detects a sense event, it will send a message to the base station and then go into sleep mode for 2 s. Otherwise, if it senses nothing for 4 s, it will not transmit any message and go back into sleep mode for 2 s.

2.3 Data Analysis

With the increase of available data, it is necessary to have a methodology to interpret data and reason about the information. We initially prototyped our data collection and classification through Node-RED[7] via synthetic inputs. We chose the programing language R for our final implementation to classify sensor data as an adversary or ally event.

2.3.1 Node-RED Testing

Node-RED[7] is an open-source, flow-based programming language that is used for rapid prototyping. We primarily used it for programming the IoBT devices, but

there are many more use-cases than those demonstrated in this project. Prior to completion of sensor development, we used Node-RED to supply data for the clustering algorithm, which helped us to understand the problem and select the appropriate clustering algorithm as well as an accurate template for the kind of data the sensors might produce. The flow-based programming was very useful in checking the accuracy of the clustering, and made it simple to add and delete sensors during testing. The graphical nature of flow-based programming made it easier for team members to follow the code under development. This was also a great tool for testing edge cases, which in turn made a more robust product. The Node-Red code is displayed in Fig. 6.

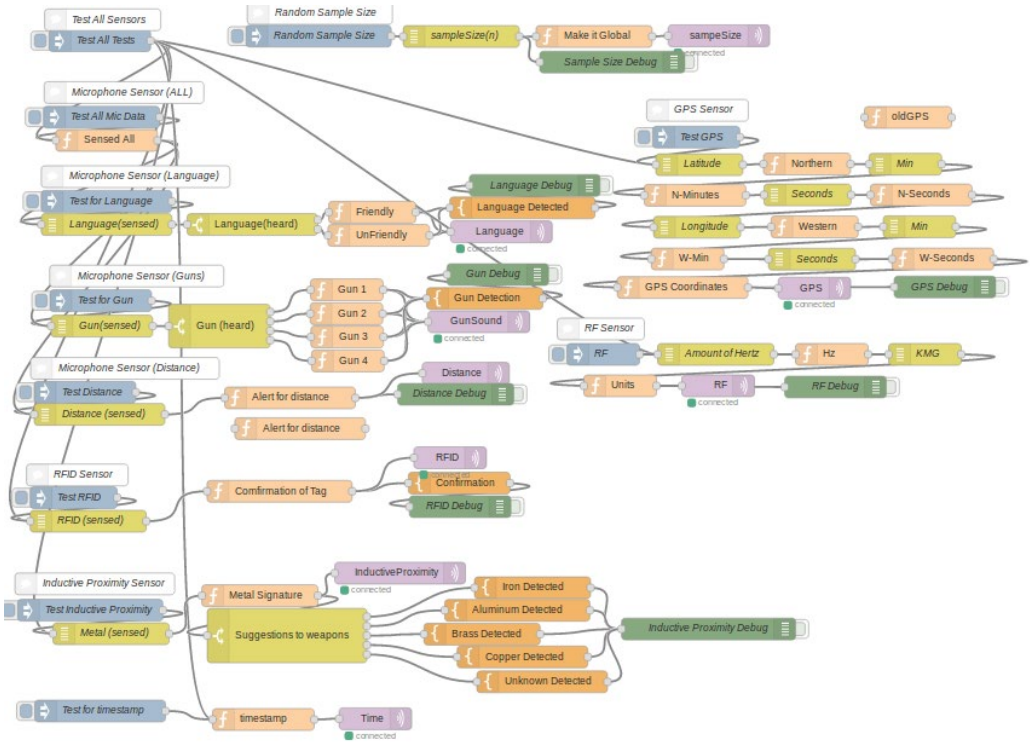


Fig. 6 Node-RED flow for testing different clustering methods

2.3.2 Clustering Algorithm

To achieve our vision of being able to classify an observation to a particular profile using these basic sensors, we implemented a data fusion schema that would integrate all of our data sources into a single classification. We discovered several algorithms (k-medoids, k-means, and Agglomerative Hierarchical Clustering)[8] that would take the data collected from the sensor field and produce a clustering based on the similarity of the points.

We first crafted labeled training data and supplied them to the algorithms so they could understand the structure of the data. Developing robust training data is important because the data are sampled from an uncontrolled environment, meaning not all sensors might operate or send data correctly to the controller. Thus, such situations must exist in the training data so the algorithms can cluster the observations if sensor readings are missing. Supplying the training data also tests the effectiveness of each method as we selected our final algorithm based on the largest percentage of correct cluster assignments. The most reliable of the three methods tested was k-means.

Once all the information from the sensors is received at the base station, it is organized into a dataframe object. The dataframe object consists of the sensor node ID, its node type (detector or classifier), and the sensor reading. All of these values are encoded as hex values to minimize the number of bytes sent over the network to reduce congestion and packet loss. This object is passed as an argument into the clustering algorithm model along with the clustering assignments of the training data. The model then returns the clustering assignment for this new observation. This assignment is registered as "ally" or "adversary".

2.4 Security

Any wireless network is going to have a few security risks, so it is important to address them. The first issue is the fact that the sensors are broadcasting all the information they have continuously. An adversary listening into the network could easily see all the information being broadcast, or even worse, fabricate their own data to confuse the clustering algorithm. The other issue is that an adversary listening to the network could attempt to flood the network with packets to prevent the sensors from transmitting their data (DoS or DDoS).

To prevent adversaries from flooding the network with useless data in a DoS/DDoS attack, we checked for malformed data and switched the XBee radio's personal area network (PAN) ID. If the check detects that the network is getting flooded with data, then the system switches to another network. This unfortunately does not solve the problem entirely because of duplication attacks, which take a valid piece of data and flood the network with copies of it. However, that type of attack could be handled by including single-use numbers in the data so that the base station could detect whether the same data have been sent multiple times.

In the simulation results shown in Fig. 7, a node is captured by the adversary (shown as the brown line) and reprogrammed to continuously send packets to the base station in an attempt to congest the network and, effectively, jam it. However, during the fourth iteration, a packet threshold is triggered and all nodes (including

the base station) switch PAN IDs, so that packets coming from the rogue node are promptly dropped, thus restoring network functionality. Finally, a recovery technique could be implemented to restore the original program (or sketch, in the context of Arduino devices) and reintegrate the patched rogue node back into the network.

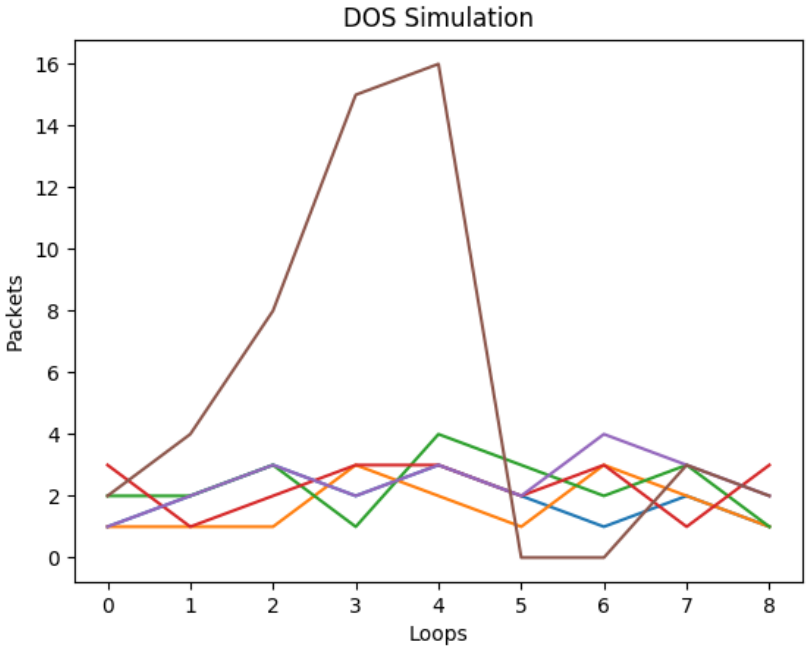


Fig. 7 Example of protection against packet overflow (DoS attack)

3. Experimental Setup

3.1 Arduino UNO

Arduino UNO is a microcontroller board using ATmega328P, which is light, inexpensive, and compatible for programming many different sensors. It is equipped with 14 digital pins (input/output), 6 analog input pins, a 16-Mhz quartz crystal, and a USB connection for programming. The Arduino UNO devices used for this project were connected to several sensors as well as an XBee Series 2 radio module. To attach that many sensors, an Arduino Shield was used to accommodate the lack of space. We programmed these microcontrollers in the C/C++ language using the Arduino Integrated Development Environment.

The UNO can be uploaded with programs that can perform numerous functions, such as reading voltage levels from analog pins or performing serial communication with the XBee modules. In addition, its small form factor makes it practical to be used as a representative unattended ground sensor. What made the UNO the clear

choice for our implementation was the vast amount of public documentation and various example codes that allowed for rapid prototyping. Figure 8 shows the location of the major components of the microcontroller.

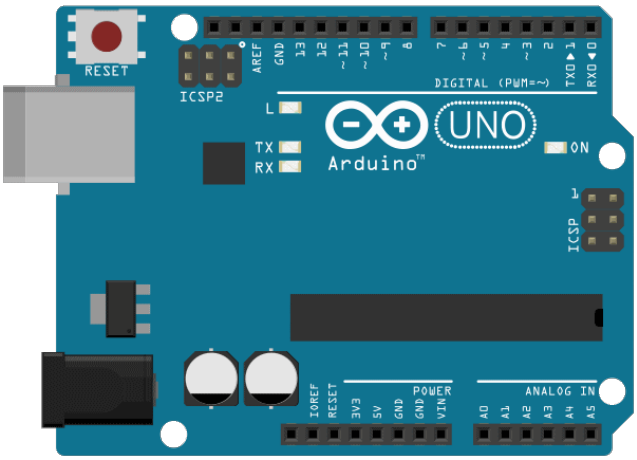


Fig. 8 Arduino UNO board

3.2 Raspberry Pi

A Raspberry Pi is an inexpensive and low-power single-board computer capable of running the Linux operating system and is available as various models. The model 3+ used for our work is equipped with four USB ports, 1-GB RAM, a port for an Ethernet connection, a high-definition multimedia interface port, many general-purpose input/output pins, and WiFi and Bluetooth wireless networking. With this level of computing, size, and cost efficiency, it is practical to use these minicomputers to do high-level computations in a small, energy-efficient package. In this experiment, the Raspberry Pi was used as the base station to which all of the sensor data would be sent. The primary reason for this was it had more computing power than the Arduino devices and was capable of running the clustering algorithm. Some of the Raspberry Pi devices were equipped with a camera and performed basic image template matching of a military camouflage pattern. A Raspberry Pi with a connected camera was used as a classifier node, and since the clustering was done on the Raspberry Pi, the image was efficiently pushed to the algorithm.

3.3 XBee Series 2

To enable communication between nodes, we chose to use the XBee Series 2 as our radio module. These modules were chosen because they incorporate the Zigbee mesh network protocol, which allows for reliable point-to-point or multipoint communication similar to that of a tactical network. In addition, the range of the

XBee is around 90 m, which helps reduce the number of nodes needed to cover a given area of interest. This surpassed the Bluetooth Low Energy module that we had originally planned to use, which had an effective range of 10 to 15 m.

The mesh network topography was designed with three different components: the coordinator, the router, and the end devices, as shown in Fig. 9. The end devices are the most basic, as their only task is to request pending messages from the parent node, which is either a router or coordinator device. End devices have the ability to change parents in response to lost connections, in which case they will notify the whole network. The router module controls and routes the traffic between the nodes, as well as stores and sends information to the child nodes (routers or end devices). It is comparable to a gatekeeper because its primary responsibility is adding new nodes to the network.

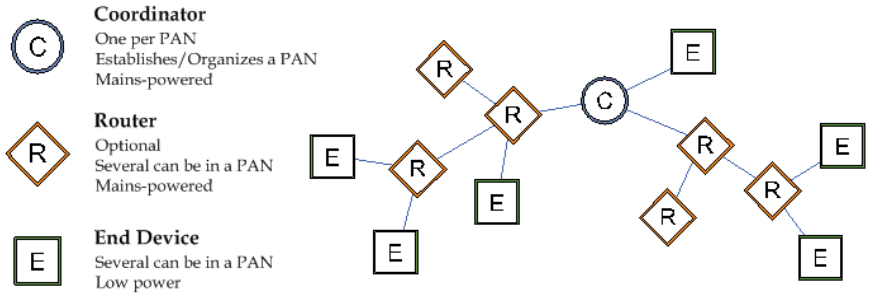


Fig. 9 Zigbee mesh network layout

The coordinator is the centerpiece of the topography because it is in charge of forming the network. The coordinator acts as a special router, which has all the capabilities of the router but also takes care of selecting the appropriate wireless communication channel. It also manages extension of the network and security for the network, acting as the trust center that authenticates new nodes and hands out network keys for the new nodes.

3.4 Sensors

In this section, we describe the sensors that were considered in our network and would work with our desired classifications. We describe how we envision their use and their limitations (Table 1). We were fortunate enough to have a large list of sensors to choose from, but not all of the sensors listed were used. These sensors are still included on the list for future work to show our thought process. Images of the sensors are shown in Tables 2–4.

Table 1 Sensors considered in the network

Name of sensor	Use-cases
Sound (microphone)	- Pick up potential languages that are considered hostile or friendly. - Identify different type of weapon sounds; could check the weapons database to match potential guns that match a certain group of people. - Calculate noises in the distance or how far certain threats could be.
RFID	- Rudimentary way to confirm that allies are passing by because RFID tags would be assigned to all Soldiers. - (**Limitation**) Range is within a couple of inches with the RFID model associated with the Arduino, but larger RFIDs, which require more power, can range from 1 to 15 m.
Camera (imaging)	- Low quality, used for discerning color patterns of uniforms. - High quality, would be able to take snapshots of objects moving in the field of view. (**Limitation**) Would consume significant battery capacity to start up, store, and send images. - Considering a central camera at the base, given data from sensors in the field would be able to angle itself to face snap high-quality images of incoming threats. Power would not be as much of an issue.
GPS	- Used to track location of sensors registering events. - Could also be able to activate sleeping nodes if a specified path has been identified. - (**Limitation**) GPS susceptible to jamming.
RF	- Track radio frequencies over the area of interest.
Infrared/thermal	- Sensor to detect heat signatures and notify the other sensors to start collecting information. - (**Limitation**) Range is limited.
Vibration	- Used to detect vibration and stress experienced by the sensor, and generates an electric charge based on the stress. The spikes in the electrical readings can indicate footsteps or vehicles roaming through the area.
IFF (friend or foe)	- Identification of friend or foe, used by the military currently in conjunction with radar to discern friendly and civilian vehicles and military units.

Table 1 Sensors considered in the network (continued)

Name of sensor	Use-cases
Ultrasonic	- Sends and receives ultrasonic waves to detect changes in distance, which signifies movement. This will be used to trigger the other sensors, and start collecting data around the area. - Triggers classification sensors to start collecting data to help determine ally or adversary.
PIR	- Detects motion by measuring IR signals emitted by objects in its field of view. If sufficient signals are detected, it will trigger an alert and start the classification process.
Magnetometer	- A classifier sensor that detects the presence of ferromagnetic materials. - Ideally, a Soldier would have a piece of metal with a unique magnetic frequency in their boot. When they walk by, the sensor would then detect that specific frequency and classify them as an ally. - (**Limitation)** The range of the magnetometer is limited, not more than 1 ft.

Table 2 Sensors used in the network: detectors

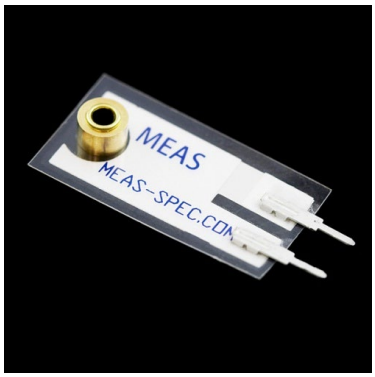
Detectors	
Ultrasonic sensor	
PIR sensor	
Vibration sensor	

Table 3 Sensors used in the network: classifiers

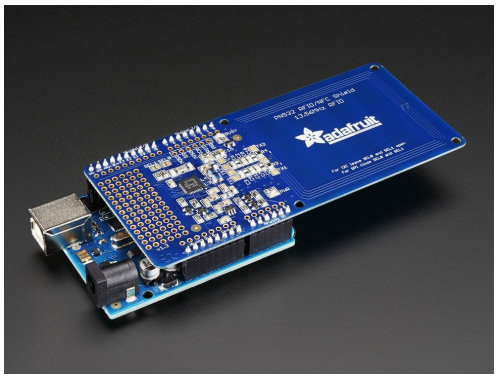
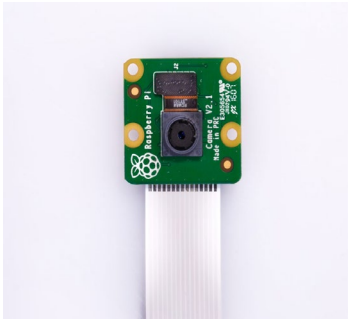
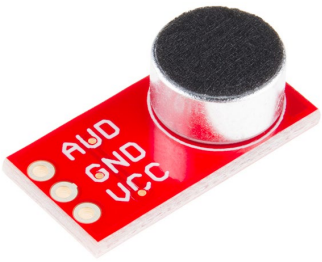
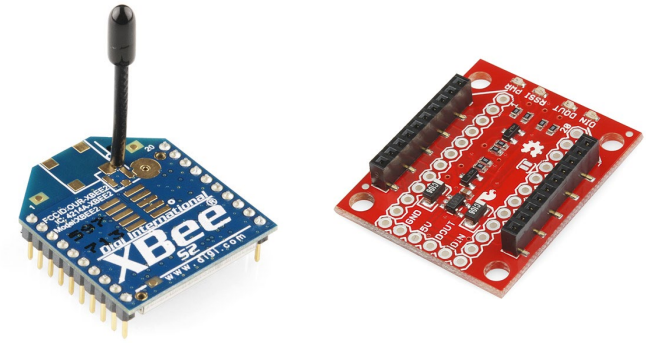
Classifiers	
Magnetometer	
RFID	
Camera	
Microphone	

Table 4 Sensors used in the network: communication

Communication	
XBee Series 2	

3.5 Network Simulation

Python scripts were used to simulate the network using different sleeping algorithms in order to observe the lifetime and connectivity of the network. Python was the language of choice because Python contains Socket and Multiprocess libraries. The Multiprocess library can be used to generate multiple processes, each of which can be used to simulate a single node. The Socket library can then be used to open multiple sockets between each process to simulate radio communication.

In each experimental setup, a sleeping algorithm was implemented with the same network layout and energy model so that we could compare the difference in performance. Each simulation was tested on a network comprising 100 nodes and a base station, where half of the nodes are classifier nodes and the other half are detector nodes. All of the nodes are placed within a 50 × 50 m area, where the base station has a fixed coordinate at position (25, 25) while the other nodes have a semirandom placement. Each node is randomly placed within a specified area within the 50 × 50 m area as shown in Fig. 10. The detector nodes are randomly placed within 0–12.5 and 37.5–50 m on the y-axis, while the classifier nodes are randomly placed within 12.5–37.5 m on the y-axis, as shown in Fig. 10.

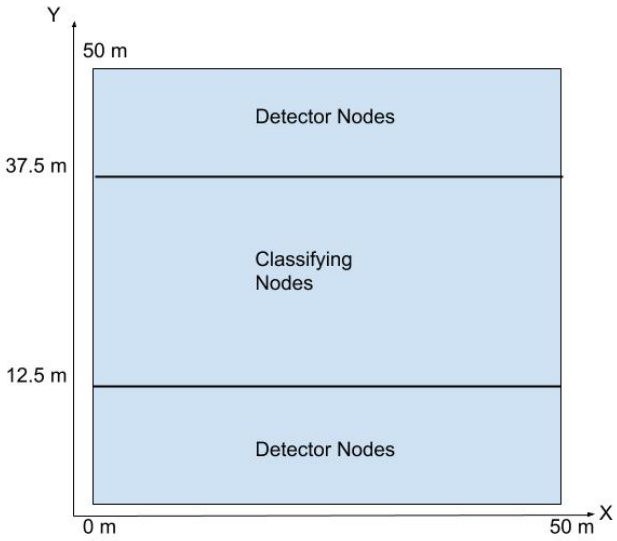


Fig. 10 Placement of nodes in the network

In addition to node placement, each node has different sensing ranges depending on the type of sensors attached to the Arduino board. For example, one node may have a sensing range of 14 m because it uses an ultrasound sensor, while another node has a magnetometer and has a range of about 1 m. By including this characteristic into the model, we can see how many events are actually captured by the network and get a rough idea about its connectivity. To maximize the connectivity range of the sensors, we decided on an effective range of 10 m for optimization of sensors and effective range of communication.

The simulation includes a node-based energy model used to calculate available battery power of each node, beginning with 0.05 J with a slight deviation of a random amount from 0 to 0.001 J. The power consumption of being in awake mode is 1 mW, while the energy it takes to transmit and receive a message is based on the first-order radio model.[9] In this model, we developed equations to calculate how much energy is consumed by a node based on the communication range and number of bits the load contains. Based on the communication range, the model either follows multipath fading or shadow fading. Because our simulated network will take place in a desert with few obstructions and the communication range is less than the defined threshold of 87 m, we used the shadow path fading model. In this model, the energy it takes to transmit a message of l bits is

$$E_T(l,d) = E_{elec} * l + e_{amp} * l * d^2, \quad (1)$$

where d is the communication range. To receive a message, the energy consumed is $E_R(l,d) = E_{elec} * l$. As reference, the constant values are E_{elec} = 50 nJ/bit and E_{amp} = 100 pJ.

Lastly, a simple flooding algorithm was implemented as a routing protocol. When a node receives an event message from another node, it first checks whether it has already received a copy of that message. If it has, then it will drop the packet. If it has not, then it checks how many hops the message has gone through. If the message exceeds a certain number of hops, then the node drops it. Otherwise, the node will broadcast the message with an updated amount of hops. In these simulations, a message can have a maximum of four hops.

4. Results

4.1 Physical Prototype

To prove that our project is viable, our group implemented a physical prototype to demonstrate its ability to classify whether an ally or adversary is crossing the field. Each node comprised an Arduino Uno or Raspberry Pi as the microprocessor, XBee Series 2 radio module, and a sensor. The network was implemented using seven sensors. For classification nodes, RFID, magnetometer, language, and camera sensors were used. As for detector nodes, PIR, ultrasound, and vibration sensors were used. Out of the seven nodes, the base station node was implemented with a Raspberry Pi equipped with a camera and sound sensor. This setup was chosen because the Raspberry Pi microprocessor had sufficient computational power to perform speech recognition and image convolution, as opposed to the weaker Arduino Uno. The rest of the sensors were then equipped with either a detector sensor or a classifier sensor.

As for the results of this physical prototype network, we can conclude that our sensors are able to sense results and distribute them to the base station. Each of the detector nodes was able to sense an event occurring within a certain range that we expected, while the classifier nodes were able to give correct information on the event, such as if the target was equipped with a RFID card. Finally, our chosen clustering algorithm for our demo K-centroids demonstrated a high accuracy when attempting to cluster our sensor data. Through multiple combinations of sensor readings, K-centroids was able to reliably classify whether or not an ally or adversary traversed through the sensor field.

4.2 Python Simulations

4.2.1 GAF Results

As shown in Fig. 11, the GAF simulation extends the network lifetime by finding nodes that are of the same type within the same group based on a virtual grid that

has a length of $r \leq R/\sqrt{5}$. For example, at the top-right corner, we can see that there are two detector nodes, node 42 and 50, in the same grid. Because they are both detector nodes, one of them is put to sleep while the other is kept awake.

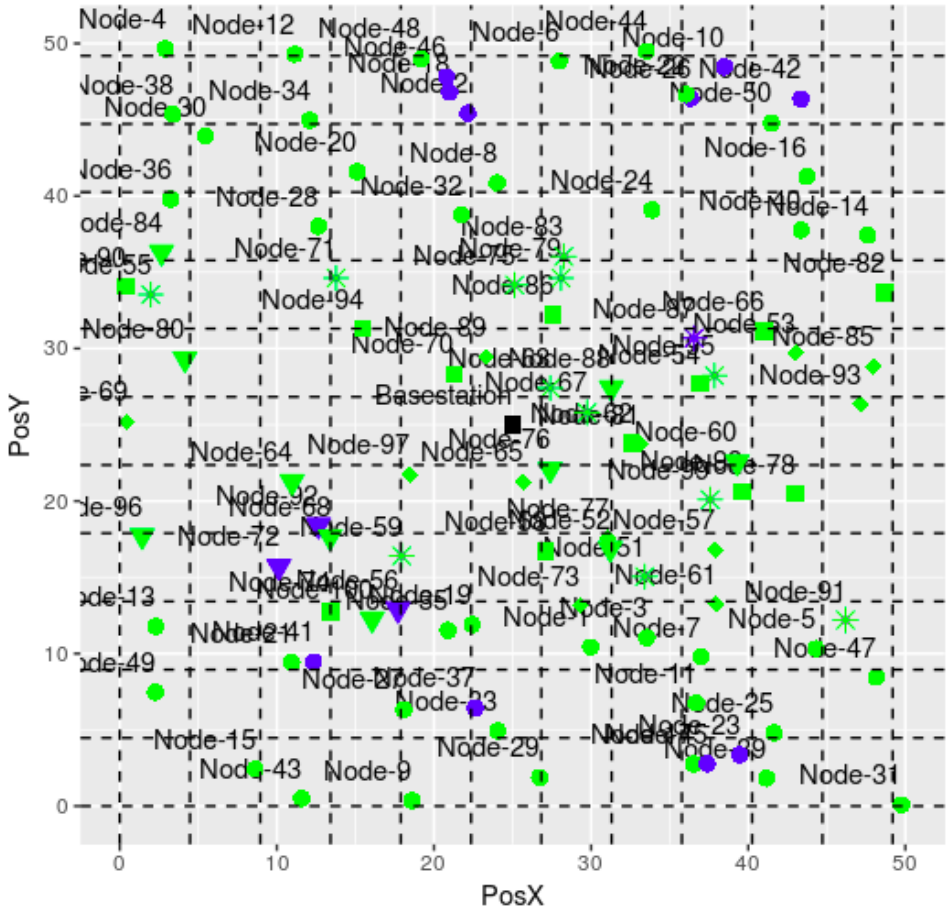


Fig. 11 Example of the GAF network. Type of node is shown by shape. Active nodes are green and sleeping nodes are blue.

Because the algorithm keeps at least one node active within a grid, we can expect full connectivity while minimizing the energy consumption of the network. As shown in Fig. 12, we can see that many nodes start running out of power after 24 s. The steep drop in number of nodes participating is because many classifier nodes are not paired with the same types of nodes, which forces them to stay awake the entire time. Another reason why this could be possible is the algorithm does not consider whether another grid could assume the responsibilities of a grid with less overall energy. The very short lifetime is a result of the low initial energy value of 0.05 J assigned to each simulated node, which was intentionally selected to limit the time required to execute the simulation. When testing with only active nodes, the lifetime of the network was about 10 s.

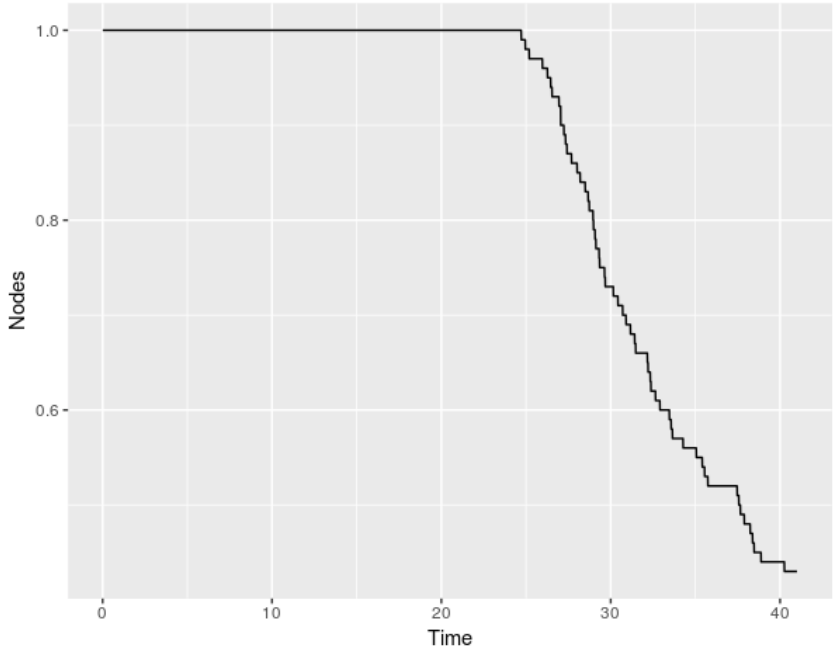


Fig. 12 Lifetime of the GAF network

As opposed to the lifetime of the network, the number of events the network is able to capture averages to around 30%, as shown in Fig. 13. While this is a low number, this could be considered a good result for two reasons. One is that the network uses a simple flooding algorithm that only allows messages to hop four times. As a result, the event was sensed many times, but was a few hops away from making it to the base station. Another reason why this is a good result is due to the sensing range of some of the nodes. Some nodes have a sensing range of 1 m, which is very small. Thus, there are many cases where an event happens close to a node, but is not within range of it. For the capabilities of this network, 30% is an average result.

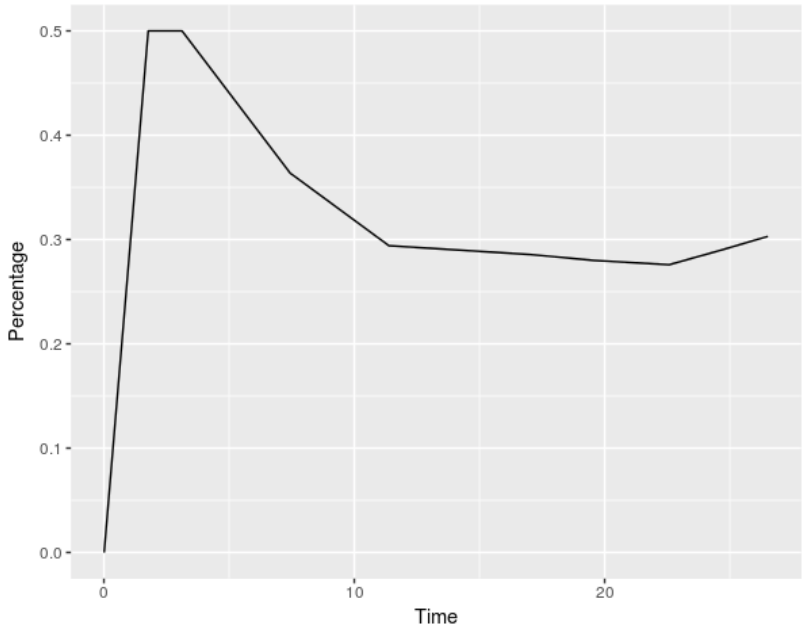


Fig. 13 Percentage of events detected in the GAF network over time

4.2.2 CKN Results

In this simulation, pictured in Fig. 14, we test the CKN algorithm with an input of $k = 1$. Thus, this should create at least a k-connected network. The reason it will create at least a k-connected is because the two conditions have a relatively low chance of being fulfilled. Thus, in some cases, many nodes may stay awake to ensure that connectivity of the network is high.

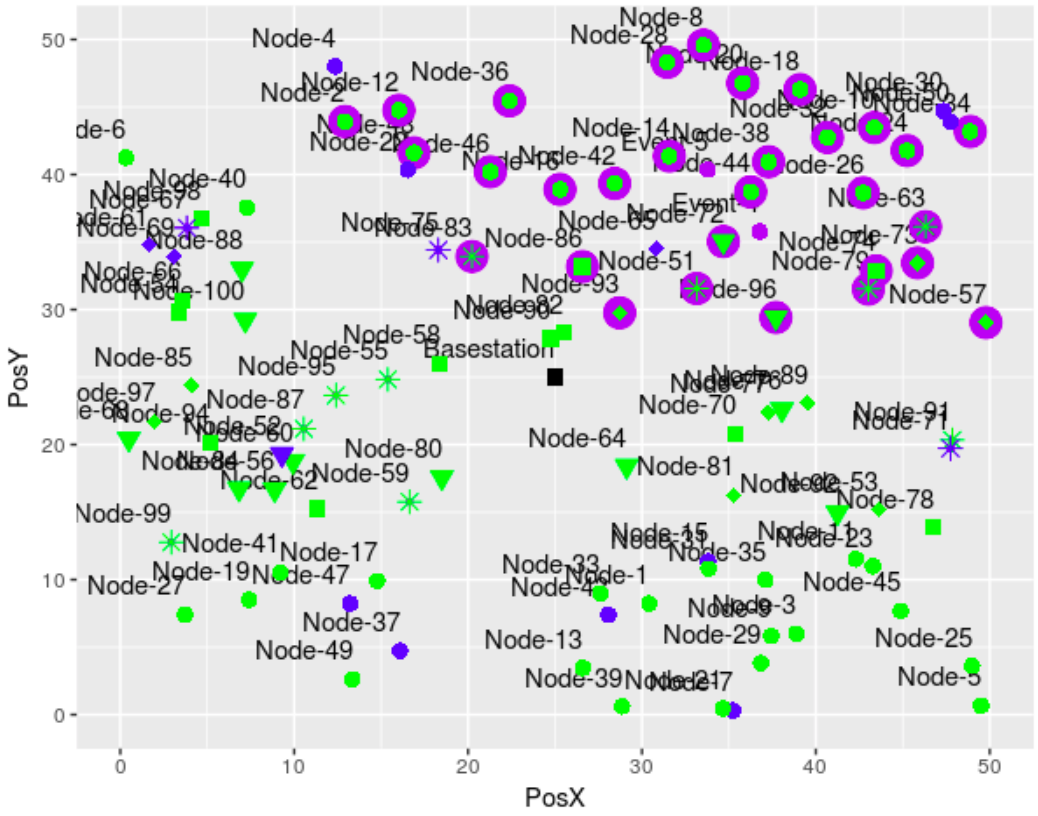


Fig. 14 Example of the CKN network. Green means active, blue means sleep, a purple ring means a node has sensed an event, and a purple circle means an event.

As a result, because of how CKN works and more nodes staying awake, the lifetime of the network is less than GAF. Figure 15 shows the lifetime of the network in epochs, which in these simulations are 2 s long. Compared to GAF, which had about 40% of the nodes alive at 40 s, the CKN network only had about 25% of the nodes alive. However, we can also see that the slope of CKN network lifetime is much less steep than GAF. In other words, it causes nodes to power off at a slower rate.

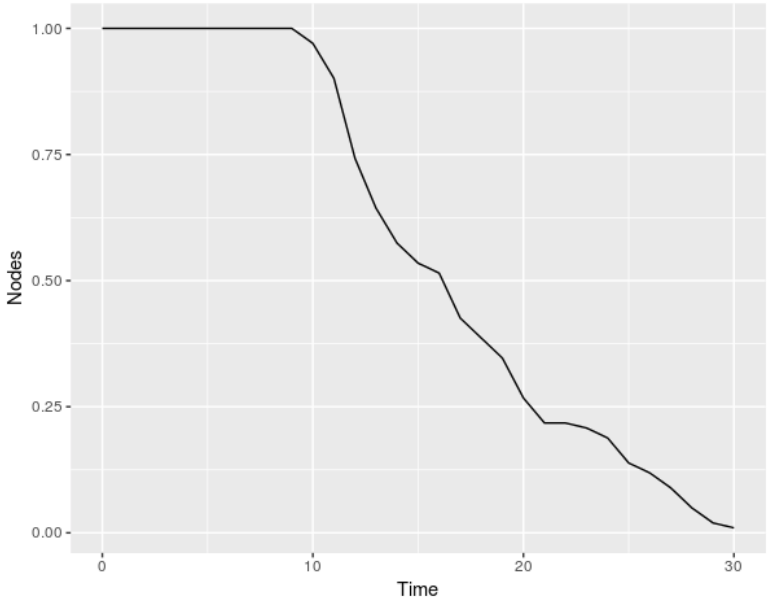


Fig. 15 Lifetime of the CKN network

In addition, we can see in Fig. 16 that the overall percentage of events detected is much higher. About 50% of all events were detected by the base station, which is much higher than the GAF algorithm performance. However, because all of the nodes near the base station powered off at the seventh epoch or so, in reality after 14 s the network fails to capture any events. One way to fix this is to implement a clustering algorithm, which may solve the issue of not having enough nodes go to sleep.

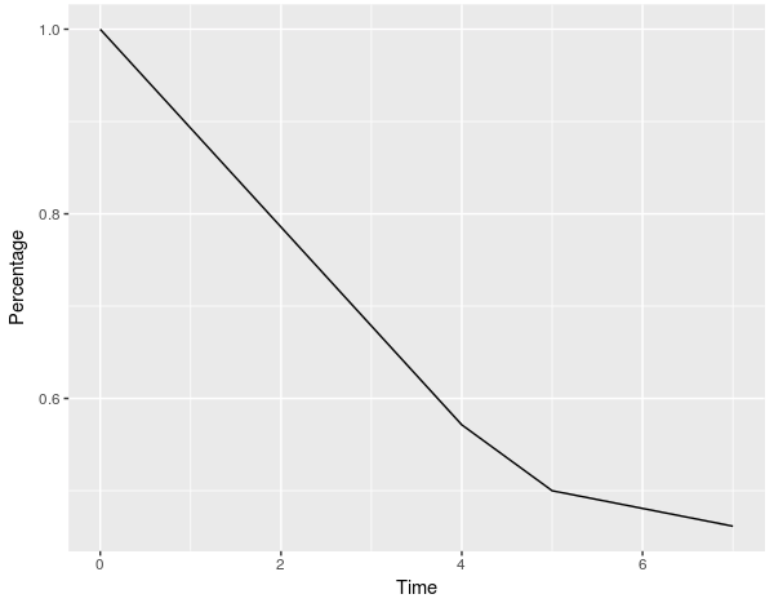


Fig. 16 Percentage of events detected in the CKN network over time

4.2.3 EC-CKN Results

Compared to using CKN, using EC-CKN does not seem to cause a significant difference in results. The network layout, lifetime, and percentage of events detected seem to generate similar results. The network layout is displayed in Fig. 17 and the lifetime of the EC-CKN network is shown in Fig. 18. Figure 19 displays the percentage of events detected in the network.

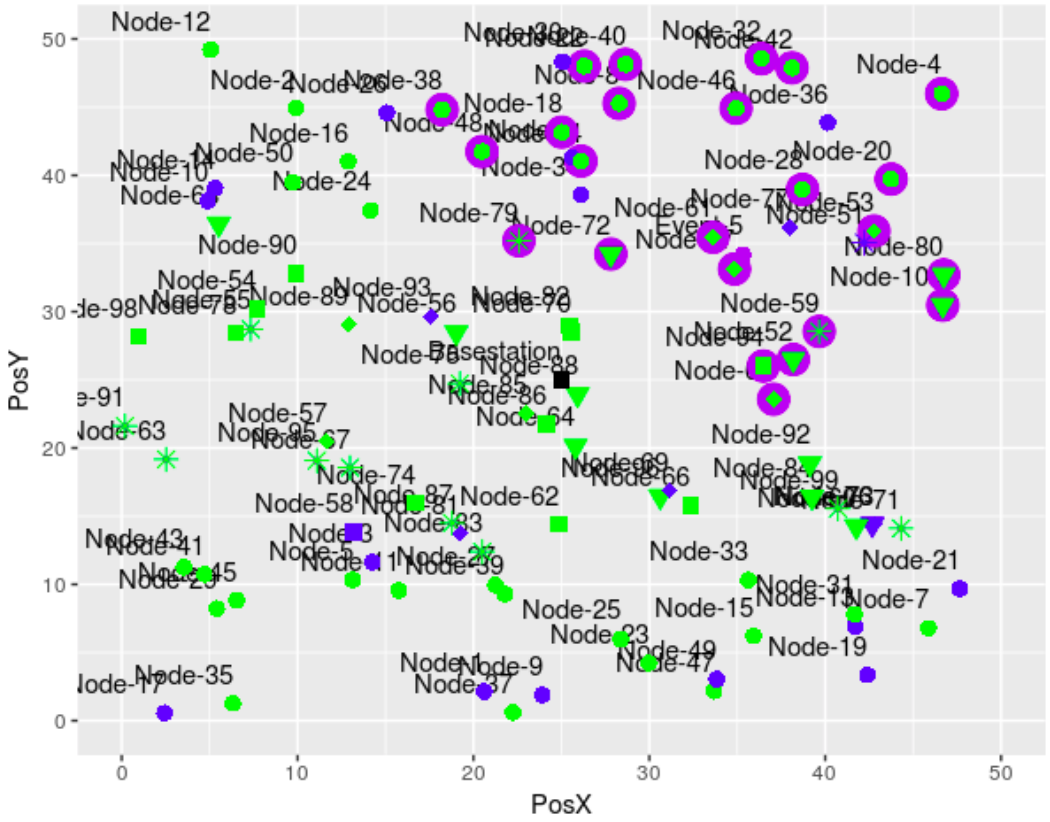


Fig. 17 Example of the EC-CKN network. Green means active, blue means sleep, a purple ring means the node has sensed an event, and a purple circle means an event.

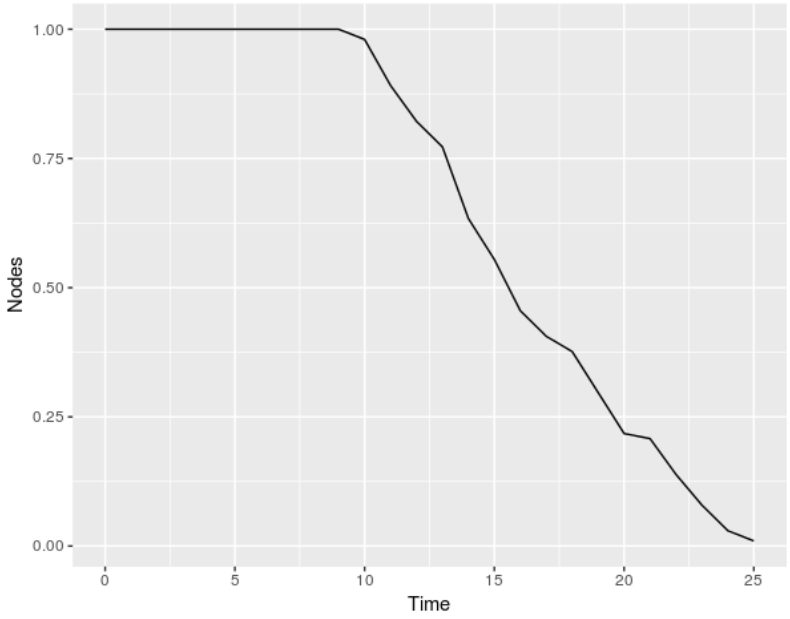


Fig. 18 Lifetime of the EC-CKN network

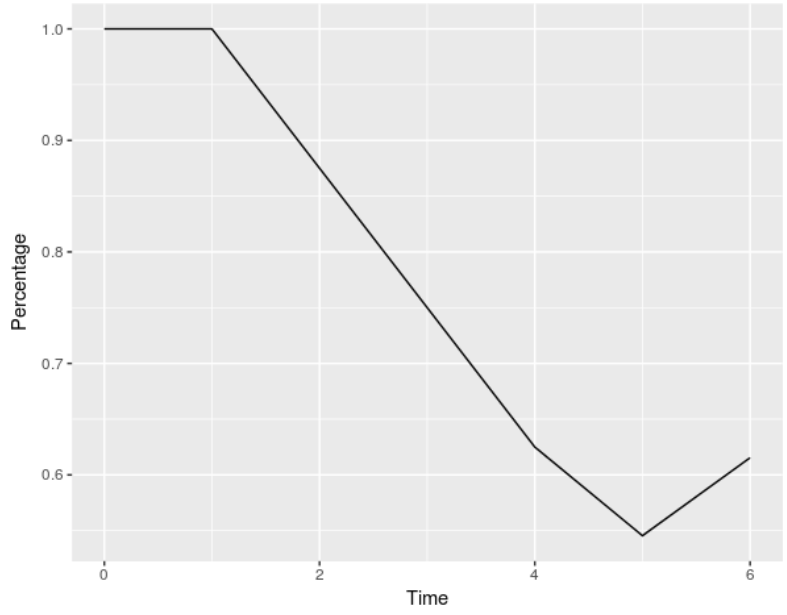


Fig. 19 Percentage of events detected in the EC-CKN network over time

5. Conclusion

Too many times we associate IoT with connecting a specific device, such as a refrigerator, to the Internet and receiving a Twitter feed, but that is just a primitive application of such an integrative technology. This concept of IoBT is still

relatively new and therefore this project is only scratching the surface of the many potential applications. Although the results were positive and the clustering algorithm was accurate enough to suggest an ally or adversary had crossed the plain, there are many more aspects to consider. Future work will include additional security and advancement in the clustering algorithm to classify more than just ally and adversary. From a security standpoint, DoS/DDoS protection is a start but ideally implementation of encryption and simplified intrusion detection methods, such as cyclic redundancy checks, would make these devices much more resilient in the field. Encryption would prevent adversaries from being able to sniff messages across the network and also protect against falsified messages that the adversaries would send to fool the clustering algorithm. Adding software or malware protection to these devices would be ideal, but due to limited storage it is difficult to install. This topic is discussed in more detail in the paper, "Lightweight Hardware Monitoring of IoT Devices".[10] Eventually, these sensors will be able to organize movement across the field into other categories like civilians, vehicles, or even indigenous animals. This is an open problem that will improve over time as we continue to develop better technology. Also, more sensors will be included in the network, giving us a better perspective of the field that they are in.

6. References

1. Digi XBee/XBee-PRO ZigBee modules (S2) - formerly ZB. Minnetonka (MN): Digi; n.d. [accessed 19 September 2018]. https://www.digi.com/support/productdetail?pid=3430.

2. Arduino Uno rev3. Arduino; 2018 [accessed 19 September 2018]. https://store.arduino.cc/usa/arduino-uno-rev3.

3. Raspberry Pi. Cambridge (UK): Raspberry Pi Foundation; n.d. [accessed 19 September 2018]. https://www.raspberrypi.org.

4. Xu Y, Heidemann J. Geography-informed energy conservation for ad hoc routing. Information System Institute, Proceeding MobiCom '01 Proceedings of the 7th Annual International Conference on Mobile Computing and Networking; 2001; Rome, Italy. pp. 70–84.

5. Wang Lei, Yuan Zhuxiu, Shu L, Shi L, Qin Z. An energy-efficient CKN algorithm for duty-cycled wireless sensor networks. International Journal of Distributed Sensor Networks 2012;06439. Hindawi Publishing Corporation.

6. Yuan Z, Wang L, Shu Lei, Hara T, Qin Zhenquan. A balanced energy consumption sleep scheduling algorithm in wireless sensor networks. 2011 7th International Wireless Communications and Mobile Computing Conference; 2011 July 4–8; Istanbul, Turkey.

7. Node-Red: flow-based programming for the Internet of Things. JS Foundation; n.d. [accessed 19 September 2018]. https://nodered.org.

8. Types of clustering methods: overview and quick start R code. STHDA; n.d. [accessed 19 September 2018]. http://www.sthda.com/english/articles/25-cluster-analysis-in-r-practical-guide/111-types-of-clustering-methods-overview-and-quick-start-r-code/.

9. Heinzelman WR, Chandrakasan A, Balakrishman H. Energy-efficient communication protocol for wireless microsensor networks. Massachusetts Institute of Technology, Proceedings of the Hawaii International Conference on System Sciences; 2000 January 4–7; Maui, HI.

10. Toth A, Rapczynski D, Wampler JA. Lightweight hardware monitoring of IoT devices. Proc Cyber Sensing 2018. 2018; 10630. SPIE Defense + Security; 2018; Orlando, FL.

List of Symbols, Abbreviations, and Acronyms

ARL	US Army Research Laboratory
CKN	connected *k*-neighborhood
CRC	Cyclical Redundancy Check
DoS	denial of service
DDoS	distributed DoS
EC-CKN	energy consumed connected *k*-neighborhood
GAF	Geographic Adaptive Fidelity
GPS	global positioning system
ID	identification
IFF	identification friend or foe
IR	infrared
IoT	Internet of Things
IoBT	Internet of Battlefield Things
PAN	personal area network
PIR	passive IR
RAM	Random Access Memory
RF	radio frequency
RFID	RF identification
USB	universal serial bus

1 (PDF) DEFENSE TECHNICAL INFORMATION CTR DTIC OCA

2 (PDF) DIR ARL
IMAL HRA
RECORDS MGMT
RDRL DCL
TECH LIB

1 (PDF) GOVT PRINTG OFC
A MALHOTRA

1 (PDF) ARL
RDRL CIN T
A TOTH