Review of Research Activities: CSE 693 Final Report

Kaleb Byrum
Dept. of Electrical and Computer Engineering
University of Louisville
Louisville, KY, USA
kabyru01@louisville.edu

***Abstract*—This report details the activities done throughout the Summer 2020 semester for CSE 693: Implementation of Ad-hoc Network Authentication. The report will detail the literature review conducted to learn about the different ad-hoc network protocols available, as well as the different secure implementations of these reviewed protocols. Finally, this report will detail the implementation measures taken to develop a custom secure ad-hoc network model that is designed specially for aerial drone swarms.**

Keywords—CSE, 693, final, report

I. Introduction

Contrary to the established norm of using access points to create an interconnected infrastructure of devices, modern computers and embedded devices are also able to connect to each other using *ad-hoc networking*. Ad-hoc, Latin for "to this," is a computer networking philosophy that aims to connect devices *to each other as a mesh*, rather than through a static (or, in Latin, "a priori") access point, which then directs traffic to other connected devices. Thanks to over thirty years of research efforts, ad-hoc networks are well-established in the Computer Science world, with different routing protocols being available for use. However, a field that is not as established is the concept of *security measures* for ad-hoc systems.

The security of infrastructure is a paramount priority when implementing network systems. When constructing static network systems, common security measures implemented to counter attackers include encryption, proxy certificates, as well as One Time Passwords (OTPs). As will be discussed in this report, recent publications show that researchers and industry engineers are working towards implementing these static network security features into ad-hoc networks. However, this task presents a unique set of challenges since *ad-hoc networks are not static in structure*.

After summarizing findings resulting from a literature review of ad-hoc networks and their various security implementations, a custom secure pro-active ad-hoc network implementation will be introduced. Features of the implementation will be discussed, as well as the future work that can be done in later semesters.

II. A Literature Review of Different Ad-hoc Network Routing Protocols

A. Introduction

The first task undertaken for this course was an extensive literature review of the various ad-hoc networking protocols available for use today. As will be discussed later, the explored routing protocols differ from one another in various pivotal aspects of how they handle routing data and tracking devices in the network. These routing protocols either make a network *proactive* or *reactive*. A proactive network is an ad-hoc network that keeps constant track of what nodes are in the network, even if there is no data to send to those nodes. *Proactive networks* benefit from ensuring transmission of data reaches destinations in an efficient manner but suffer from the performance penalty caused by constant network traffic. *Reactive networks* benefit from low performance penalty at the cost that nodes within the network cannot fully know the layout of the network and rely on updates brought by packets that contain data to send. Well-known implementations of both proactive and reactive networking protocols exist, as well as numerous secure implementations of these protocols. During this literature review, proactive and reactive networks were explored and will be summarized.

B. Destination-Sequenced Distance Vector Routing (DSDV)

The first networking protocol explored was **DSDV,** or Destination-Sequenced Distance Vector routing. DSDV is a proactive network routing protocol that constantly updates node tables on each connected device [1]. DSDV is a table-driven routing scheme based on the *Ballman-Ford Algorithm*, which aims to solve the *routing-loop problem*, which is the anomaly where data is stuck being sent in a loop of connected devices. Each entry in the routing table contains a sequence number, which in turn determines if data can be forwarded to a certain node. DSDV always considers all routes, which is possible through the proactive updating messages sent to nodes that ensure each node understands the topology of the network at any given time. Since DSDV considers all routes, it is also effective in determining the best route possible for use at a given time. However, since DSDV's priority to keep the network topology known always, the network suffers from excessive amounts of overhead, and as a result is not well suited for large-scale applications [1]. Due to this, DSDV is not used much today, but has been forked to new protocols that are commonly found today, including AODV. An example DSDV node table is found in Figure 1:

Destination	Next Hop	Number of Hops	Sequence Number	Install Time
A	A	0	A 46	002000
B	B	1	B 36	002200
C	B	2	C 28	002500

Figure 1: DSDV Routing Table, containing destinations and best routes. Sequence numbers are used to prevent routing loops.

C. Ad-hoc On-Demand Distance Vector Routing (AODV)

The next networking protocol explored was **AODV**, or Ad-hoc On-Demand Distance Vector routing. This protocol is effectively the sequel to DSDV, as one of the main authors of the former is the main author of the latter. In contrast to DSDV, AODV is a reactive protocol that only updates node lists when sending data. Due to this revision, AODV enjoys significantly less overhead when compared to DSDV [2]. AODV makes use of bi-directional links, meaning that connections are handled on a local-level rather than tracked on a global-level by each node. Instead of always considering all routes, AODV makes use of bi-directional links to perform *route discovery*, which consists of the source node sending a preliminary packet to each of its neighbors to learn the local network topology in the effort of finding the best route possible to the destination. This action is similar in nature to *flooding*, where a message is sent to each node in the network stemming from the source. To reduce network overhead, discovered routes are cached by nodes and used again should data need to be sent to that destination again. These cached routes are deleted should later transmissions discover that the route is no longer valid. Like DSDV, sequence numbers are used to prevent routing loops, but are now also used to detect routes that are no longer valid [2]. A demonstration for how Route Requests conducted by AODV nodes is shown below:

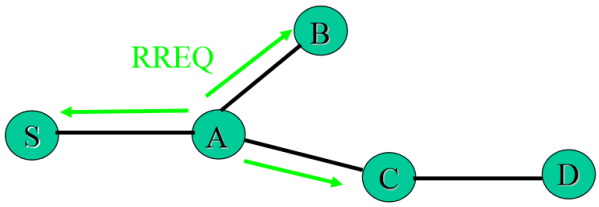

Figure 2: Demonstration for how Route Requests are handled. Node S is looking for a route to D, and since neighbor A does not have a direct route to D, A *floods* the network looking for Node D.

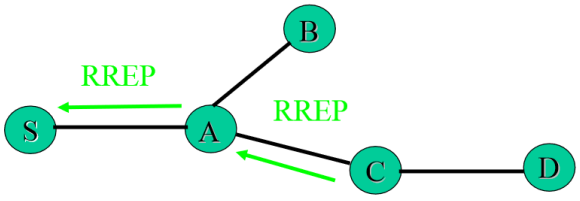

Figure 3: Once a route to D is found, packets are sent back to the source using cached addresses to the senders. This does not require flooding of the network.

D. Dynamic Source Routing (DSR)

The next networking protocol explored was **DSR**, or the Dynamic Source Routing protocol. DSR is a reactive protocol that benefits from the fact that its overhead *can be scaled to zero*. Like AODV, DSR operations can be placed into two categories: *route discovery* and *route maintenance* [3]. Route discovery operations are processes where the source node determines a path to the destination node. This process is like AODV's route discovery process, as both protocols do not fully know the topology of the network at a given time. Route maintenance operations are processes where the source node is able to detect that a previously determined path to the destination node is no longer available, and a new path from the stored cache must be used instead, or a new route must be constructed using route discovery operations. In contrast to AODV, DSR allows for special links between nodes, including both uni-directional and bi-directional, which *allows nodes of different interfaces to communicate with one another* [3]. Like AODV, route discovery operations take place in a manner shown below:

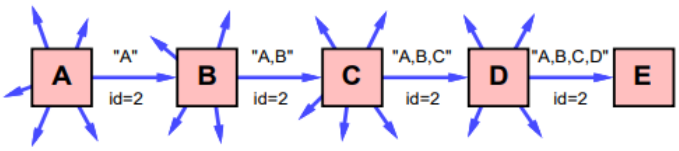

Figure 4: Route discovery example. Node A is the source, and node E is the destination.

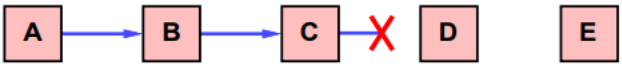

Figure 5: Example Route maintenance situation. Since this cached path is no longer valid, a new one will need to be found in cache or discovered using the process in Figure 4.

Another unique feature to DSR is that nodes that are otherwise not the destination of a target can learn the path to the target by *overhearing* packets of information. In this way, paths to destinations never requested are constructed so that route discovery is not needed for every request for a new node [3]. This predictive measure has its limitations though, and risks creating routing deviations or loops if not properly handled.

In regards to the custom implementation that will be introduced later, the most prominent feature of DSR is its ability to handle *reply storms*, which occur when nodes within the network independently react to reply requests at the same time, causing the network to be excessively flooded with information that otherwise isn't needed. Using delay timers and elective measures, reply storms are prevented by dictating that the *first node to reply to a request should be the only one replying to the request* [3]. An example of a routing storm is shown below:

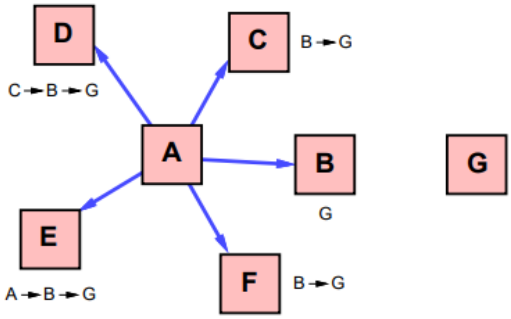

Figure 6: Example of a reply storm.

E. Better Approach to Mobile Ad-hoc Networking (B.A.T.M.A.N.)

The final protocol explored in this literature review was **B.A.T.M.A.N,** or Better Approach to Mobile Ad-hoc Networking. Like DSDV, BATMAN is proactive, and aims to have each node in the network updated with the most recent form of the network topology at all times [4]. Designed as a replacement for OLSR, BATMAN functions similarly in the fact that despite being a proactive protocol it handles packets node-by-node and does not compute whole paths at a time. Where BATMAN differs from other protocols explored in this review is the fact *that an implementation of the protocol exists as an open source distribution* [5]. As a result, BATMAN enjoys high use and is actively maintained. This is in contrast to protocols such as DSDV, which per this literature review does not have an open source implementation available. Extensive documentation of BATMAN has enabled its implementation on various platforms including x86, x86-64 and ARM.

BATMAN aims to be an all-inclusive solution for *mobile ad-hoc networks*, or MANETs, which other protocols other than DSR may not be well suited for due to the extensive overhead and hardware penalties accrued from use. The protocol begins with a node transmitting to other neighbors in proximity about its existence, which is then forwarded to the neighbor's neighbors and so forth. The network in this way is flooding with these messages which allows for each node to learn the topology of the network quickly. BATMAN uses a route ranking system to learn and decide which route to a destination is the most efficient route. However, unlike AODV and DSR, *BATMAN keeps track of all routes at all times*, which is based on the assumption that these routes are rankable and otherwise still usable at some point [4]. By tracking the number of nodes between a source and destination, BATMAN can determine the best route, using the logic that less nodes means a quicker route.

III. A Literature Review of Different Secure Ad-hoc Network Implementations

A. Introduction

Before beginning efforts to implement a custom secure ad-hoc network solution, a literature review of already existing secure implementations is warranted to learn more about the decisions engineers and researchers have made when designing these systems. Secure implementations are often built on top of existing routing protocols as a basis, which means that the actual network design is left unchanged.

B. Secure Efficient Ad-hoc Distance in Mobile Wireless Ad-hoc Networks

The first secure ad-hoc network implementation explored in this literature review was **SEAD**, or Secure Efficient Ad-hoc Distance in Mobile Wireless Ad-hoc Networks. SEAD is robust against multiple uncoordinated attackers that create incorrect routing states. SEAD is built on top of DSDV, which makes this implementation proactive.

SEAD makes use of hash chains to authenticate paths, which only provides a lower bound protection against hackers. For example, a hacker can increase the sequence number but *SEAD prevents the decreasing of the sequence number*. Thanks to DSDV's use of sequence numbers, SEAD also provides protection against routing loop attacks [5].

Unfortunately, *no source code implementation of SEAD openly exists.* Efforts to contact the authors of the protocol have not been fruitful, and unfortunately any discussions about the protocol have been dated. This in turn means that any custom implementations based on SEAD are unfortunately infeasible with currently available materials [5].

C. Ariadne, Secure On-Demand Routing Protocol for Ad-hoc Networks

The next secure ad-hoc implementation explored was **Ariadne**, which is a secure on-demand routing protocol. Ariadne withstands node compromise by relying solely on highly efficient symmetric cryptography. Ariadne is commonly paired with *Tesla*, which is an authentication scheme that makes use of loose-time synchronization to verify communication with nodes [6].

Ariadne can authenticate using three different schemes: either using shared secrets between each pair of nodes, using shared secrets between communicating nodes combined with broadcast authentication, or using digital signatures. Ariadne also makes use of *per-hop hashing*, which is a process where communication between each node that is on the way from the source to the destination, is authenticated to ensure secure transmission of information each step of the way [6].

Ariadne is built on top of DSR, which makes the implementation on-demand in route discovery and route maintenance. Unfortunately, like SEAD, Ariadne has no source code implementation openly available, rather only a publication that discusses the decisions made creating the protocol.

D. Secure Routing Protocol, Lightweight Security for DSR

The next secure ad-hoc implementation reviewed was **SRP**, which is a lightweight security implementation for DSR. SRP is an extension that is attached to Route Request and Route Reply packets. It delegates Route Maintenance operations to use the *Simple Mail Transfer Protocol* (SMTP). SRP is also able to detect packet modification on the Route Request and Replies. This is possible by requiring a security association with the nodes. As SRP is based on DSR, the protocol is on-demand [7].

SRP uses a group key with a signature to authenticate nodes to enter the network [7]. This key feature will come

into prominence later when the custom ad-hoc network solution is introduced.

SRP answers the issue of network security in a lightweight manner, as the authors of the protocol make note that secure implementations suffer from adding additional overhead to the original protocol, to the point where using the original protocol is no longer justified [7].

E. Authenticated Routing for Ad-hoc Networks

The next secure ad-hoc implementation explored was **ARAN,** or Authenticated Routing for Ad-hoc Networks. ARAN is based on the AODV protocol, which makes this implementation on-demand.

In ARAN, each node has a certificate signed by a trusted authority. This process associates the node's IP address with a public key. These certificates are held by each node in the network and are required for each hop of the process [8].

Using these techniques, ARAN is effective in securing against modification of sequence numbers, the modification of hop counts, the modification of source routes, spoofing of routes, and the fabrication of source routes (commonly called *cache poisoning*). The following table describes the security features of ARAN compared to DSR and AODV:

Attack	AODV	DSR	ARAN
Remote redirection			
modif. of seq. numbers	Yes	No	No
modif. of hop counts	Yes	No	No
modif. of source routes	No	Yes	No
tunneling	Yes	Yes	Yes, but only to lengthen path
Spoofing	Yes	Yes	No
Fabrication			
fabr. of error messages	Yes	Yes	Yes, but non-repudiable
fabr. of source routes (cache poisoning)	No	Yes	No

Figure 7: Comparison of security features within AODV, DSR, and ARAN. ARAN is based on AODV [8].

Route discovery and route maintenance are handled as described in these figures:

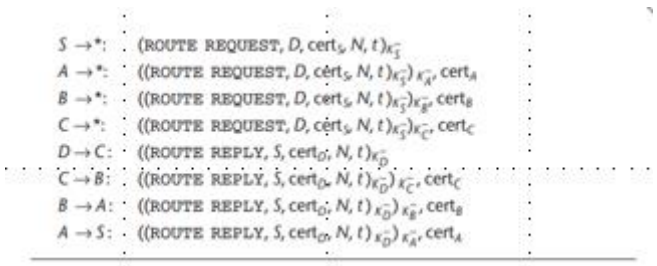

Figure 8: Route discovery in ARAN. Note in this figure, node S is discovering a route to node D. Each node rebroadcasts the first route request packet it receives from each route discovery. When the route request reaches the target, the distination returns a route reply to the node from which it heard the route request. Each node hearing a route reply forwards the reply to the node from which it heard the request [9].

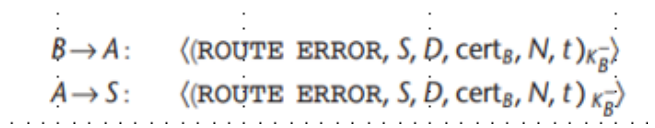

Figure 9: Route maintenance in ARAN. When node B determines that its next hop, to destination D is unreachable, it broadcasts a signed route error message indicating the broken route. Each node using B as a next hop for D rebroadcasts this error but does not re-sign it [9].

Unfortunately, no open-source implementation of ARAN seems to exist.

F. Secure AODV

The next secure ad-hoc implementation investigated was **SAODV**, or Secure AODV. SAODV uses signatures to authenticate most fields of a route request and route reply and uses hash chains to authenticate the hop count. Like ARAN, per-hop authentication is used. SAODV allows for intermediate note replies through Route Reply Double Signature Extensions, or RREP-DSE. These packets are essentially a route request and a route reply rolled into one. SAODV uses signatures to protect ROUTE ERROR messages as well [10]. This is different from ARAN, where attacks are handled by simply blacklisting the source.

According to studies conducted by the authors of this protocol, SAODV succeeds in securing the AODV protocol from *black hole attacks*, which is the spoofing of a neighbor list to trick a node to send all data to a particular malicious node [11].

Unfortunately, no open-source implementation of SAODV seems to exist.

G. Secure Link-State Protocol

The next secure ad-hoc implementation explored was **SLSP,** or Secure Link-State Protocol. In SLSP, digital signatures and one-way hash chains are used to ensure the security of link-state updates. SLSP makes use of the Intrazone Routing Protocol as a basis, which is proactive in nature, but only limited to a node's immediate neighboring area. SLSP link-state updates are signed and propagated through a limited number of hops [12].

As in SEAD and SAODV, a hash chain is used to authenticate the hop count, and the hash chain values are authenticated using the hash chain's anchor. These values are included in the signed portion of the SLSP link-update state. SLSP also includes lightweight flooding prevention mechanisms [12].

Unfortunately, no open-source implementation of SAODV is available.

H. Confidant

Another secure ad-hoc implementation investigated was **Confidant**, which is a reputation-based system implemented on top of DSR. Confidant consists of multiple pieces working together. These pieces include: the *monitor*, which acts as the master code within the node that determines the final trust ratings for each nearby node, the *reputation system* which contains sets of algorithms that determine a final trust rating to provide to the monitor about a certain node, the *trust manager*, which evaluates the current states of the nodes and flags alarms of changes within the network,

and finally the *path manager*, which handles the paths for each node, which change depending on the trust ratings of each node [13]. The following figure describes the relationship between the four parts:

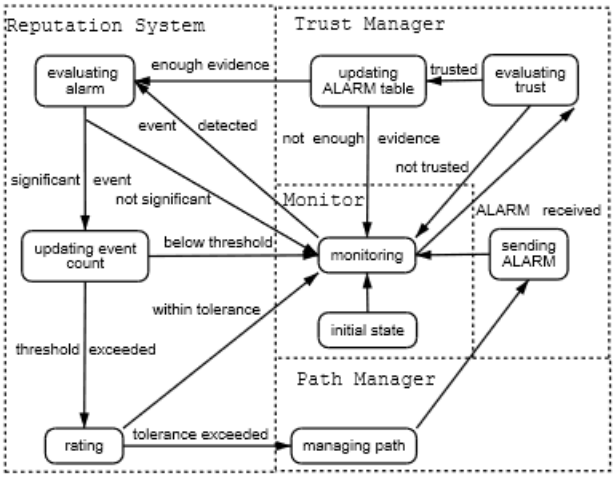

Figure 10: The trust architecture and finite state machine within each node [13].

Local ratings are held by nodes, which determine if a node can accept a connect from a new but rated node. These lists, however, can be overridden by a highly trusted node. This implementation of secure networking is unique compared to others discussed as it uses *trust* as the main metric of determining paths of transmission [13]. This is also the first implementation that attempts to override the original method of which the underlying protocol determines paths, as the original DSR protocol did not support mapping paths using trust.

Despite these innovations, no open-source implementations of this protocol exist.

I. Awerbuch's Reputation Ad-hoc Routing Protocol

As it turns out, Confidant's innovations are not alone, as using *trust* as a metric of determining routes is a growing field. Another protocol that uses *trust* in this manner is **Awerbuch's protocol**, which buildings on top of DSR as well. Awerbuch proposes a protocol that is like DSR in structure but uses a *weight list* instead of a node list. This effectively replaces how DSR handles routing.

The weight list is used in conjunction with a Fault Detection algorithm, which feeds into a Link Weight Management system. Essentially, if a fault is detected on a specified path, the *weight* of the path is affected so that the network understands that *this path is not to be as highly trusted as the others* [14]. As a result, a confidence-based system governs how routes are treated in the network. This is similar in implementation to CONFIDANT.

However, where Awerbuch shines is in the search for faulty links. Awerbuch makes use of *binary searches* to find faulty links, which is particularly effective in finding such issues within the network. However, binary searches are not effective in finding multiple issues along a single path, and as such the protocol is not well suited for such attacks [14]. An example of how path weights are determines is described below:

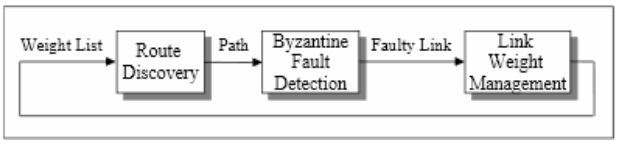

Figure 11: Awerbuch protocol's method for calculating link weights [14].

Unfortunately, no implementation openly exists.

J. Secure Traceroute

Another reputation-based system investigated is the **Secure Traceroute** protocol. Secure Traceroute diversifies itself in that probe packets are indistinguishable from ordinary packets, and it can use secret sharing to differentiate between individual losses and losses beyond a certain threshold. Another difference is that ST performs *linear searches* to find attackers as opposed to *binary searches* that are equipped by Awerbuch. As a result, ST is more robust against multiple attackers on the route but is much slower to find faulty links. Secure Traceroute is also capable of *reinforcement learning*, which can be used to choose the best routes from source to destination based on experience. This can be used to avoid malicious nodes as well [15].

Typical Secure Traceroute operation, which implements reinforcement learning, is described below:

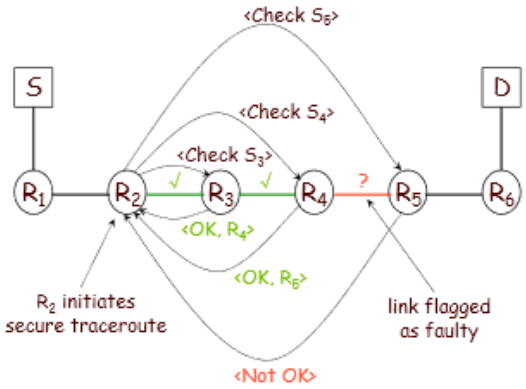

Figure 12: An illustration of secure traceroute in operation. The node R2 initiates the process, and determines that between R4 and R5, the route is faulty. This faulty route is taught to the other nodes and avoided [15].

Unfortunately, no open-source implementation of this protocol seems to exist.

H. Secure Ad-hoc Network using BATMAN

The final secure ad-hoc implementation investigated in this literature review was **Secure Ad-hoc Network Implementation using BATMAN.** As this implementation is on-top of the BATMAN protocol, this implementation is proactive. This implementation makes use of proxy certificates to prevent common ad-hoc network attacks, including *wormhole attacks* [16].

New network nodes are required to show their proxy certificate to be authenticated by a neighbor node. Once the node is authenticated, the network welcomes the node into the network and engages it as part of the system. This

implementation takes advantages of the pro-active nature of BATMAN by authenticating nodes when BATMAN proactively updates [16].

An example of the different entities within this network structure is shown below:

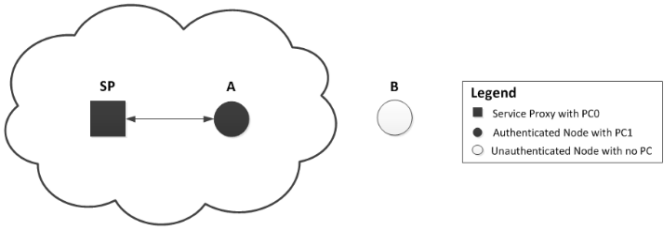

Figure 13: Different entities within the network. Authenticated nodes must come in contact with the SP to become authenticated by the network.

Unlike every previously discussed secure implementation, *this implementation is open-source and available for use*. This fruitful discovery proved to be a prime starting point for implementing a custom solution for aerial drones. The custom implementation that will be discussed builds on-top of this implementation, and therefore ultimately builds on top of BATMAN. This was done due to the fact that the established framework necessary for this project is proven to work in this implementation, and only requires the *addition of new features* and the *modification of existing code to port to the modern Linux kernel.*

IV. Implementing a Custom Secure Ad-hoc Network – The Democracy Secure Ad-hoc Network

A. Introduction

Equipped with knowledge gained by the literature review, the goal of this research effort is the custom implementation of a secure ad-hoc protocol that meets the needs of a swarm of autonomous aerial drones. This application closely aligns with the requirements and struggles experienced by Mobile Ad-hoc Networks, with the additional consideration that this network intends to be fully autonomous in decision making. Because of this additional consideration, a focus of this custom implementation is that *it must be fully capable of reconfiguring and rebooting on its own.* As will be discussed later, this implementation meets that requirement.

The Democracy Secure Ad-hoc Network is a secure implementation of the pro-active ad-hoc routing protocol BATMAN Daemon. It builds off the work of Graarud's *Implementation of a Secure Ad-hoc Network*, which provided the groundwork for interacting with the BATMAN Daemon code, as well as the secure transmission of data between nodes. This implementation is better described as a *fork* of Graarud's work as it aims to keep the same functional philosophy with the addition of bug-fixes that adapt it to modern Linux systems, as well as new features and mechanisms that are tailor-made to this implementation's main focus: *aerial drone networking*.

B. Definition of Network Nodes Classified in Network

Before continuing further, the hierarchy of the network nodes will be delved. There are two types of network nodes that are used in this network. The first is the **Service Proxy**, which is the Master Node of the network. This node serves Proxy Certificates for other nodes in the network . In this scheme, *only one SP is allowed in the network at a time*. If a SP goes missing in the network, the remaining nodes within the network will consult with each other to decide which node should become the new SP. Only the SP can authenticate nodes that are new to the network. If the network intends to always look for new nodes to add, then an SP is always necessary in the network. *This implementation assumes that an SP is always wanted.*

The second type of node is an **Authenticated node**, which are "normal" nodes within the network. These nodes cannot add new unauthenticated nodes within the network, but they can re-add already authenticated network nodes that have gone missing at some point.

C. Building the Network: Adding new nodes in the network using an SP

When the SP is initialized, an origin proxy certificate, *PC0*, is generated. This certificate is self-signed and requires no other proxies to create. This certificate will be used to sign other Proxy Certificates within the network. Because of this, if new nodes are assumed to join the network during operation, an SP node is always necessary to ensure their secure addition into the system.

After initializing at least one SP and one Authenticated node, the network can now be constructed. The Authenticated node will initialize as a node without a Proxy Certificate and will search for an SP node so it may join the network. Once the SP Node and newly created Authenticated node become direct network neighbors, the two engage in an authentication process that results in the creation of a *signed proxy certificate* for the Authenticated node to use. This certificate is referred to as a *PC1*, as it originates from a *PC0* certificate that is held by the SP. *PC1 certificates cannot sign other PC1 certificates. All PC1 certificates within the network must be signed by the same SP.*

D. Message Sending Between Nodes

BATMAN is a proactive network, meaning that *whether data needs to be sent or not, messages are still sent between network nodes to ensure they are still connected*. Taking advantage of this, the DSAC uses packet messages between nodes to ensure that only authenticated nodes are allowed within the network, which is achieved by proactively providing keystream messages that validate the authentication relationship between nodes of all roles.

Building off this, the DSAC adds message functions that ensure an SP is always present within the network. Should, through these messages, it be discovered that an SP node is *missing* from the network, then these same established messaging functions are used to decide *where* a new SP node should be, as well as to *reboot* the network so it uses the proxy certificates provided by the newly-created SP node.

E. When Is an SP Not Needed?

In some situations, the full implementation of the DSAC may not be necessary. This includes situations where *you already know the number of nodes that will be in your*

network and you can authenticate them initially. An example of this scheme is if you authenticate a swarm of drones before they take off, and no new drones are expected to join the network mid-air. This means that only drones within the swarm can rejoin the network if lost, ultimately creating a *walled garden* ad-hoc network. However, in situations where the consistent performance of a node is in question (i.e. the node has the possibility of powering off) or new nodes are expected to join mid-operation, an SP node should be kept in the network.

F. *What Happens When an SP is Potentially Missing from the Network?*

During normal Authenticated node operation, after engaging with other nodes to ensure authentication status is intact, the node will check its direct neighbor list to see if an SP node resides in the network as a neighbor. Should the node detect that an SP node is not available as a direct neighbor, the node will flag this as a potential issue to investigate.

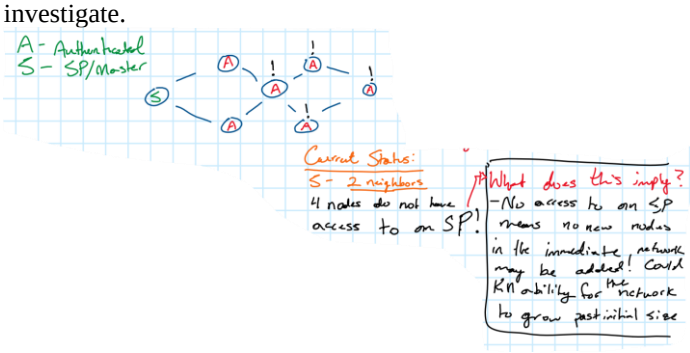

Figure 14: Example of nodes not finding SP in neighbor list.

As seen in Figure 14, it is possible for multiple Authenticated nodes to learn of this issue at the same time. To prevent multiple SP search operations, an *election* process will take place where the nodes within the network will decide between themselves which node will potentially become the SP Node should one not be found. In this implementation, the current criteria for what node becomes the SP is that the first node that detects the issue will become the SP. However, this can be modified to work with different types of criterion, including a merit-based system like CONFIDANT. The "election" process is implemented using custom messaging functions that trigger special functions within each node's code should they already be in a state that they have flagged in issue. The *election* process also ensures that *only one SP node will exist in the network at a time*.

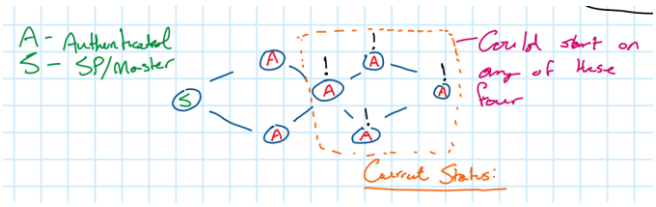

Figure 15: Example of the election process, which takes place between the four flagged nodes.

Before making assumptions that an SP node must be missing, there are possible explanations to this problem that do not mean an SP is missing. For example, an SP node could reside in the network, but just is not a neighbor to the node. Or the node that flagged the issue has no neighbors at all. Should it be the latter, this issue is immediately detected and resolved. The following mechanism will determine if the issue is the former:

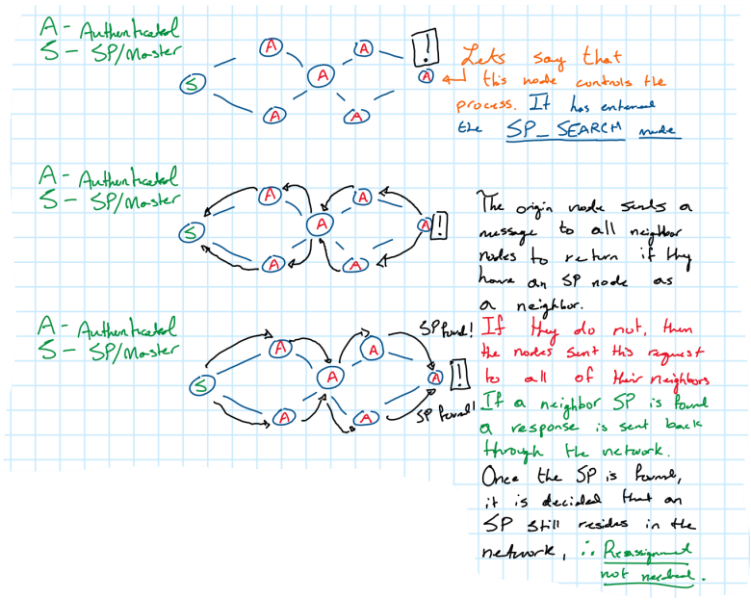

Figure 16: SP Search Process, in the case that an SP Node is found within the network.

To ensure that the missing SP node is not just due to the fact that its within the network but not a neighbor, the alerted node will begin a search for an SP node in the network by making use of a newly created feature: **Neighbor Nudge**. Neighbor Nudge is a custom-developed lightweight way to send small packets of information about the network, such as authentication information.

Using Neighbor Nudge, the search for an SP node within the network is conducted by flooding the network with messages containing requests for each node to search their neighbors for an SP node. One of two outcomes are possible from this process: either an SP node is found in the neighbor list of an authenticated node in the network, and the location of this node is sent back to the originator of the request; or after a specified timeout period, it is assumed that an SP node does not exist in the network, and a reboot process is initiation for a remaining node in the network to take the former SP Node's place.

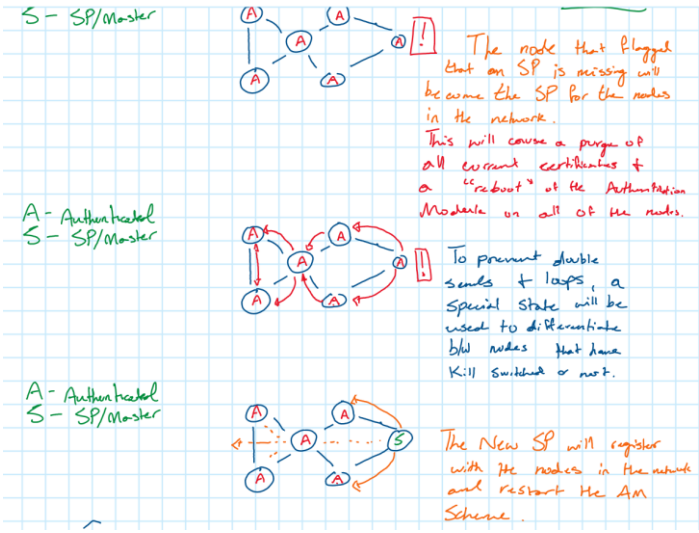

Figure 17: SP Search & Reboot Process, in the case that an SP Node is not found, and the network must be reconfigured to a new SP Node.

Should the original SP node be found, the network infrastructure is kept as-is, and if it is not, then it is rebooted and reconfigured to work with the new SP node. When the node is rebooted and reconfigured, the missing SP node will be independently rebooted as an Authenticated node, so that it may join the network as a member should it reappear later. It will have to register a PC1 with the new SP node, however. The following figures describe this process:

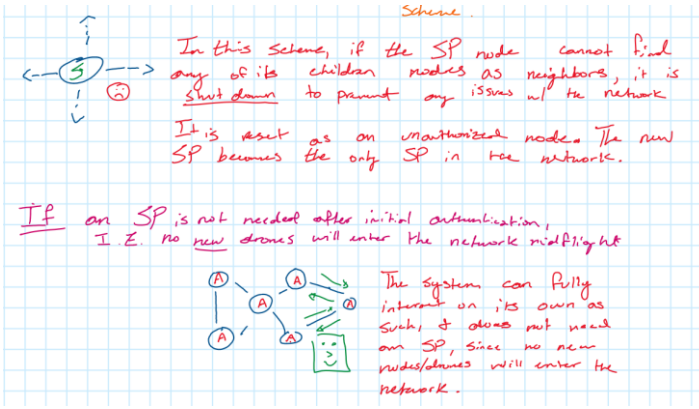

Figure 18: Lost SP Node reboot process. Also describes that if an SP Node is not needed, the network can be configured to become a "walled garden" as discussed earlier.

After a node is rebooted, a cooldown period of a specified time is engaged to prevent repeated reboots from one command. After this cooldown period, each node in the network is set to engage in normal activity.

G. Suggested Future Work for this Implementation

As this implementation is a *fork* of an already existing secure implementation, the full source code of this project is readily available for use and modification. However, due to the constrained timeframe of the Summer semester, as well as the unique circumstances brought by COVID-19, some steps do remain before the confident implementation of this system in aerial drone networks.

First and foremost, the source code needs to be ported to ARM. As of now, the source code of this project is proven to compile and run on x86-64 systems, which was the only architecture available due to this project being developed remotely, and without access to a laboratory setting. Theoretically, this source code should port to ARM smoothly since it is written in C and uses commonly implemented Linux libraries, but testing and debugging may be necessary to ensure the porting goes smoothly.

Second, after being ported to ARM, this implementation needs to be tested in a laboratory setting. While theoretically the source code implementation of this protocol should work as expected, the inaccessibility of hardware prevents a level of certainty. Testing and debugging may be necessary to ensure that the program works on drones as expected.

Third, in future iterations of this program, I suggest finding better criteria for what dictates the best SP Candidate should the original SP go missing. In the initial implementation, the only criteria considered is *when* the winning candidate flagged the issue, when this may not be the best criteria to use. Perhaps basing the decision of the SP Candidate on criteria such as location within the network or through a merit-based system is a better route to do. Unfortunately, due to time-constraints, this effort was not pursued.

V. Conclusion

The security of ad-hoc networks is a paramount concern should these networks be transmitting sensitive or otherwise important data. In the case of autonomous aerial drone swarms, ad-hoc networks are used to transmit instructions between drones and collectively make decisions that all nodes in the network follow. The hacking of this information can be detrimental to the long-term performance of the network. However, using proxy certificates and proactive networking, the securitization of these types of ad-hoc networks are possible, ensuring that the behavior of these networks is solely influenced by the drones themselves, rather than an outside party. This implementation of a secure ad-hoc network also adds features of autonomy, as the nodes within the network are capable of collectively deciding which node within the network should become the master node should the original Service Proxy go missing from the network, a feature that was previously unavailable. Also, this implementation introduces lightweight features for messaging nodes within the network as well as methods for flooding the network, which goes in line with how BATMAN proactively keeps track of network activity. This implementation, using these established functions, is customizable to various needs, but aims at meeting the demand of securing autonomous drone networks.

VI. Acknowledgements

The author would like to acknowledge and thank Dr. Adrian Lauf of the University of Louisville for his unwavering support and guidance during this project. I would also like to thank Mr. Espen Graarud for his groundbreaking work in securing networks, as well as his publications detailing his research efforts.

VII. References

[1] C. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers," [Online]. Available: https://www.cse.iitb.ac.in/~mythili/teaching/cs653_spring2014/references/dsdv.pdf.

[2] B. Awerbuch and A. Mishra, "Ad hoc On Demand Distance Vector Routing Protocol," [Online]. Available: https://www.cs.jhu.edu/~cs647/aodv.pdf. [Accessed June 2020].

[3] D. Johnson, D. Maltz and J. Broch, "DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks," [Online]. Available: https://sites.cs.ucsb.edu/~ravenben/classes/290F-w05/papers/dsr-chapter00.pdf. [Accessed June 2020].

[4] Freifunk, "BATMAN Protocol Concept," [Online]. Available: https://www.open-mesh.org/projects/open-mesh/wiki/BATMANConcept. [Accessed June 2020].

[5] open-mesh-mirror, "open-mesh-mirror/batmand," GitHub, [Online]. Available: https://github.com/open-mesh-mirror/batmand. [Accessed June 2020].

[6] Y.-C. Hu, D. Johnson and A. Perrig, "SEAD: secure efficient distance vector routing for mobile wireless ad hoc networks," 21 June 2002. [Online]. Available: https://ieeexplore.ieee.org/document/1017480. [Accessed June 2020].

[7] Y.-C. Hu, D. Johnson and A. Perrig, "Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks," 2005. [Online]. Available: https://netsec.ethz.ch/publications/papers/ariadne-journal.pdf. [Accessed June 2020].

[8] P. Papadimitratos and Z. Haas, "Secure Routing for Mobile Ad Hoc Networks," [Online]. Available: https://people.ece.cornell.edu/haas/Publications/MC2R-papadimitratos-haas.pdf. [Accessed June 2020].

[9] Sanzgiri, Kimaya and e. al., "Authenticated Routing for Ad hoc Networks," [Online]. Available: https://www.dsm.fordham.edu/~mathai/papers/ Authenticated_routing_for_ad_hoc_networks.pdf. [Accessed June 2020].

[10] H. Yih-Chun and A. Perrig, "A Survey of Secure Wireless Ad Hoc Routing," 2004. [Online]. Available: https://www.cs.jhu.edu/~cs647/class-papers/Security/ AdHocSurvey.pdf. [Accessed June 2020].

[11] L. Songbai and e. al., "SAODV: A MANET Routing Protocol that can Withstand Black Hole Attack," 11 December 2009. [Online]. Available: https://ieeexplore.ieee.org/document/5376147. [Accessed June 2020].

[12] M. Shurman and e. al., "Black Hole Attack in Ad hoc Networks," 3 April 2003. [Online]. Available: https://www.researchgate.net/publication/ 220996481_Black_hole_attack_in_mobile_Ad_Hoc_networks. [Accessed July 2020].

[13] P. Papadimitratos and Z. J. Haas, "Secure Link State Routing for Mobile Ad hoc Networks," 28 January 2003. [Online]. Available: https://people.ece.cornell.edu/haas /wnl/Publications/saahn03.pdf. [Accessed June 2020].

[14] S. Buchegger and J.-Y. Le Boudec, "Performance Analysis of the CONFIDANT Protocol (Cooperation Of Nodes: Fairness In Dynamic Ad-hoc NeTworks)," 9 June 2002. [Online]. Available: https://www.cs.jhu.edu/~cs647/class-papers/Security/BucheggerL02.pdf. [Accessed June 2020].

[15] B. Awerbuch and e. al., "An On-Demand Secure Routing Protocol Resilient to Byzantine Failures," 28 September 2002. [Online]. Available: https://cnitarot.github.io/papers/wise2002_sec_routing.pdf. [Accessed June 2020].

[16] V. Padmanabhan and D. Simon, "Secure Traceroute to Detect Faulty or Malicious Routing," 2002. [Online]. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/07/hotnets2002.pdf. [Accessed June 2020].

[17] E. Graarud, "Implementing a Secure Ad hoc Network," June 2011. [Online]. Available: https://core.ac.uk/reader/52107318. [Accessed June 2020].