ISTANBUL TECHNICAL UNIVERSITY
FACULTY OF COMPUTER AND INFORMATICS

BATMAN-Adv With CentAir

Graduation Project Final Report

Ahmet Secaettin Alıcı
150190097

Department: Computer Engineering
Division: Computer Engineering

Advisor: Gökhan Seçinti

June 2025

Statement of Authenticity

I/we hereby declare that in this study

1. all the content influenced from external references are cited clearly and in detail,
2. and all the remaining sections, especially the theoretical studies and implemented software/hardware that constitute the fundamental essence of this study is originated by my/our individual authenticity.

İstanbul, Haziran 2025

Ahmet Secaettin Alıcı

Acknowledgments

Gökhan Seçinti Hocam
Sultan Çoğay Hocam

BATMAN-Adv With CentAir

(SUMMARY)

Let us consider an ad hoc network utilizing WiFi, as described in [1]. Nodes that are considered central according to various centrality metrics, which often physically correspond to the geometric center of the network, tend to handle more traffic than others. These central nodes become more critical for routing.

During periods of high traffic load, these central nodes turn into bottlenecks. Collisions that don't occur during periods of low traffic begin to emerge at these central nodes, leading to performance degradation that cannot be predicted from observations under light traffic conditions.

Moreover, due to their predictable and important positions in the network traffic flow, the overall network becomes more vulnerable. It may become more difficult for the network to recover in both unexpected traffic surges and in the presence of a jammers.

Among several centrality metrics mentioned in the literature, I selected the one I found most suitable for BATMAN-adv. In this approach, each node enters a mode in which its hop penalty is proportional to the number of its neighbors. Since each node already knows the total number of routers in the network and is also aware of its own neighbors, it can calculate its hop penalty using the formula below:

$$\text{Hop penalty} = (\text{broadcasted number}) \times \frac{(\text{number of neighbors})}{(\text{number of routers})} \tag{1}$$

The more neighbors a router has relative to the total number of routers, the higher its centrality metric, and accordingly, the higher the hop penalty it incurs. This encourages traffic to be routed away from central nodes toward more outer nodes. As a result, traffic is distributed more evenly, and collision risks that would otherwise affect central nodes during heavy traffic are proactively mitigated.

To enable this behavior in the ad hoc network, I implemented a new hop_mode command in batctl. Using this new command, two hop modes can be defined for the network. In the uniform mode, all nodes assign the same broadcasted value as their hop penalty. The second mode, called CentAir, enables the behavior described above: each node calculates its hop penalty using the formula above based on a broadcast from the node running the command.

To implement, verify, and test this approach, I used a virtual test environment. In this environment, QEMU virtual machines running BATMAN-adv are connected to each other via Linux bridges over TAP interfaces. Links between nodes in a test topology are defined using nftables, which also controls link quality and, if needed, bandwidth constraints.

To manage this virtual environment, I developed a bash script that can execute commands across all nodes simultaneously or on individual nodes and send the output of the command run inside the VM back to the host terminal. I also wrote a Python

script that reads a JSON configuration file describing the topology and sets up the entire environment accordingly. These tools proved very useful for setting up, observing, and controlling the environment. All of those were very useful to me create a testing pipeline.

However, this virtual setup does not allow real collisions to occur. Since one of CentAir's main advantages is to prevent performance loss due to collisions at central nodes under high traffic, I implemented a mechanism to emulate collisions. According to a predefined threshold, if a packet arrives too soon after a previously received packet, it is considered a collision and is dropped. This threshold, collision mode, and collision statistics can also be configured and monitored using batctl.

CentAir'li Batman-Adv

(ÖZET)

[1] Wifi'ın kullanıldığı bir adhoc network'ünü düşünelim. Çeşitli merkezi metriklerine göre merkezi kabul ettiğimiz aynı zamanda bu network'lerin 3 boyutlu uzaydaki merkezine de denk gelen node'ların üzerinden diğer node'lara göre daha fazla trafik akarlar ve bu node'lar routing için daha kritik hale gelmeye başlarlar.

Trafik yoğunluğunun arttığı zamanlarda bu bahsettiğimiz merkezi node'lar bottleneck haline gelirler. Trafik sakinken yaşanmayan collision'lar bu merkezi node'larda meydana gelmeye başlayarak performansı az önce sakin trafikten tahmin edilemeyecek şekilde düşürmeye başlar.

Aynı zamanda network trafiğindeki önemli ve tahmin edilebilir konumları yüzünden tüm network daha savunmasız hale gelir. Hem beklenmeyen trafik yoğunluklarında hem de bir jammer karşısında network'ün toparlanması daha zor olabilir.

Makalelerde bahsedilen birkaç merkezi metriklerden BATMAN-adv için en uygulanabilir olduğunu düşündüğüm metriği seçtim. Buna göre her node komşu sayısına doğru orantılı olarak hop penalty'leri olduğu bir moda girecek. Zaten her node network'te kaç tane router olduğu biliyor, zaten kendi komşularını da biliyor, bu bilgilerin ışığında hop penalty'sini aşağıdaki formüle göre hesaplıyor:

$$\text{Hop penalty} = (\text{broadcasted number}) \times \frac{(\text{number of neighbors})}{(\text{number of routers})} \tag{2}$$

Her router toplam router sayısına oranla ne kadar çok sayıda komşusu varsa o kadar merkezi metriği fazla oluyor ve buna göre penalty hesaplanıyor. Böylelikle trafik merkezi olan node'lardan daha az merkezi dış node'lara doğru yönelmiş oluyor. Böylece hem trafik daha homojen dağılmış oluyor hem de trafiğin yoğunlaştığı periyotlardaki merkezi node'larda gerçekleşecek olan collision riskine karşı önceden önlem alınmış oluyor.

Tüm bu ad hoc network'ün hop modunu belirlemek için batctl'e hop_mode isimli yeni bir commmand ekledim. Bu yeni command ile network için iki tür hop modu belirleyebiliyoruz. Uniform, tüm node'lar bu command'in çalıştığı node'dan broadcast edilen sayıyı hop penalty olarak atıyorlar. Diğer mod ise asıl bahsettiklerimi mümkün kılan CentAir modu, bu command'in çalıştığı node'den broadcast edilen sayı yukarıdaki formüle göre diğer node'ların hop penalty'si olarak atanıyor.

Bu bahsettiklerimi uygulamak, çalıştığından emin olmak ve test etmek için nasıl bir ortam kullandım? Bu sanal ortamda: Batman-adv'yi çalıştıran Qemu virtual machine'leri tap interface üzerinden birbirlerine linux bridge'i ile bağlılar. Test edilmek istenilen topolojideki node'ların arasında linkler tap interface'leri üzerinden nftables'ta belirleniyor. Linklerin kalitesi ve gerekirse bandwidth'i bile gene nftables'tan kontrol ediliyor.

Aynı anda tüm node'larda command'ları çalıştırabilmek veya tek bir node'a command'ı

çalıştırmak ve virtual machine'in içinde execute edilen bu command'in output'unu host terminal'ine gönderebilen bash script'i; topolojiyi yaratmak için configuration file olarak json file'ını okuyup tüm ortamı yapılandıran python script'i de bu bahsedilen sanal ortamı yapılandırmak, gözlemlemek ve kontrol edebilmek için çok işime yarıyor. Tüm bunlar güzel bir testing pipeline'yı oluşturmam için çok işe yaradı.

Ancak bu bahsedilen ortamda collision mümkün değil ve CentAir'in en büyük faydalarından biri yoğun trafikte merkezi node'lardaki collision'lardan dolayı oluşan performansı kaybını önlemek. Buna göre, ben de belli bir threshold'a göre batman driver'ına gelmiş en son pakete çok yakın gelmiş bir paket collision sayılır ve paket drop ediliyor. Gene bu threshold, collision modu ve istatistikleri batctl ile kontrol edilebiliyor ve gözlemlenebiliyor.

Contents

1 Introduction and Project Summary

Batman-Adv is a kernel Linux driver for ad hoc networks, where there is no central router to administer the communication between nodes, thus every node have to play the router's role.

Batman operates on OSI layer 2. It processes and forways raw ethernet packets. Hence, to outside every batman nodes seems like a switch's ports. A batman network is esentially a distributed switch.[2].

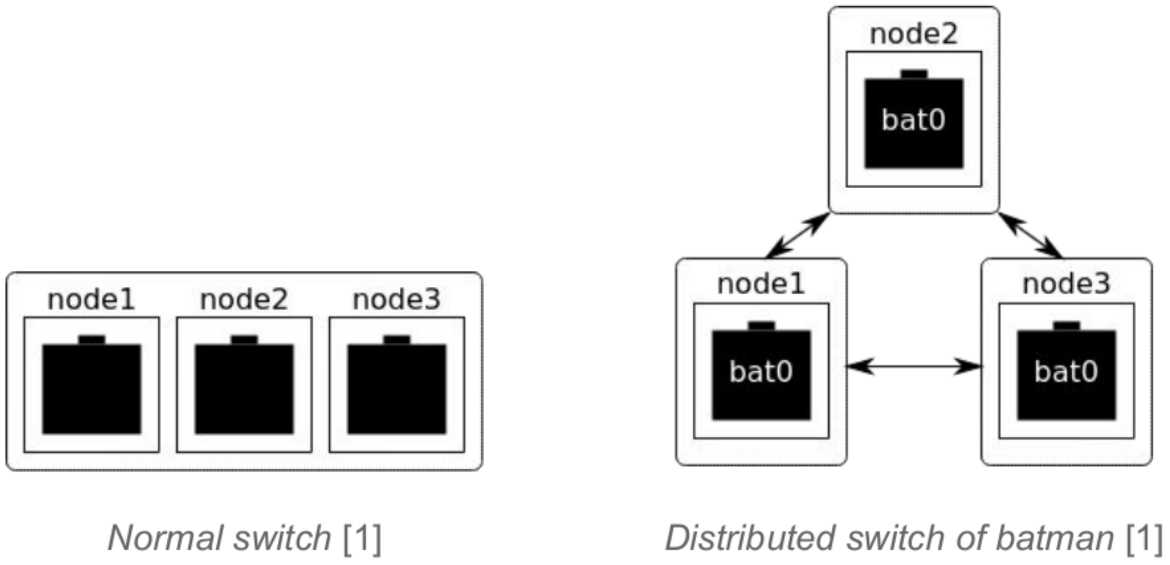


Figure 1.1: Batman network as distributed switch[2]

Because, generally nodes in this type of networks are inclined to be mobile, a lot of the routing decisions must be dynamic, paths between nodes must be reevaluated every once while to keep track of mobile routers. This is not an easy task. To solve this mobility issue, a batman node, at certain intervals, broadcasts discovery packets to announce its presence to its neighbors and other batman nodes in the network. These discovery packets are named OGM packets, OGM stands for originator message. A batman node can broadcast other batman nodes' OGM packets too. Via these packets, batman nodes can calculate transmission qualities between them and decide the routing decisions[3].

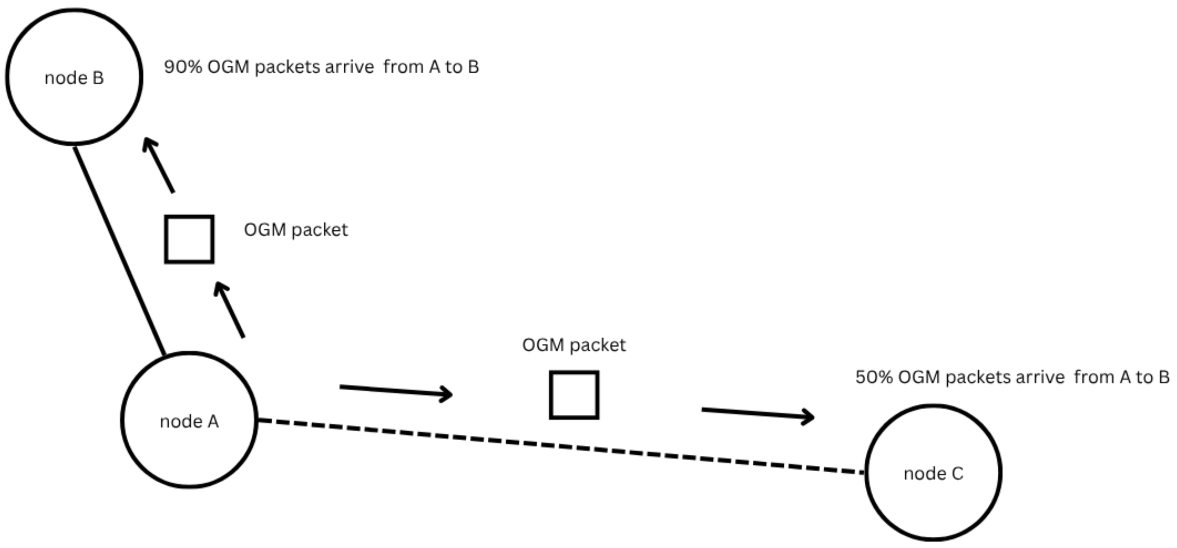

node B
90% OGM packets arrive from A to B
OGM packet
node A
OGM packet
50% OGM packets arrive from A to B
node C

Figure 1.2: broadcasting OGM packets

But this broadcasting of those OGM packets happens in millisecond intervals and collisions can be in mere only nanoseconds, thus, packets losts due to collisions might not be observed by batman's discovery system.

Because, in the virtual environment there is no collision logic, I implemented this feature in the batman driver itself. If current packet comes under certain threshold, after the last packet, current packet is dropped and collision counter increments by 1. In statistics there is a counter that counts every coming packets as well as collisions.

In wifi it is not possible to detect collisions, thus it is not statistics about collisions are not possible. One can analyze and guess where collisions might occur and try to sway the traffic away from those dangerous nodes to more safer nodes. That's where CentAir comes in to play. Batman might not notice collision related packets loss, and especially central nodes are prone to collisions because of the dense traffic around it. According to some centrality metric, CentAir detects these dangerous central nodes which might become bottleneck during burst traffic, and directs the traffic to the more outer nodes.

2 Comparative Literatre Survey

[4] is about a emulator for batman nodes. It uses virtual distributed ethernet (vde) to connect VirtualBox VM instances with each other. At the beginning my project, before modify batman source code, first I needed an environment to test the changes I made, or just only vanilla batman itself. I stumbled upon this paper. I tried to build same thing but instead of virtualbox, I use Qemu. After a while, I found a better, more flexible approach that I have more control of the testing environment.

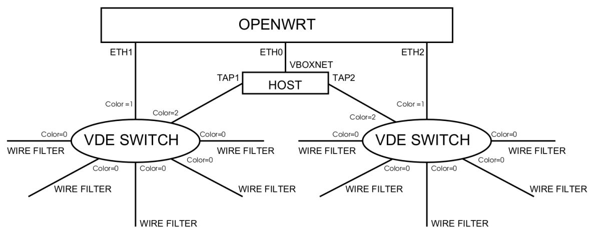


Figure 2.1: Diagram of emulator in [4]

In [5], the testing environment consists of Qemu instances, connected each other via their TAP interfaces and a Linux bridge, but in this case, there are no multi-hop and imperfect paths between the nodes, every node is connected to each other and can hear everything.

[6] takes the previous environment and adds the multi-hop and imperfect links between nodes with nftables. This is the virtual environment I used to test my BATMAN-Adv driver. Here, links between node, their imperfection and every other feature about the bridge itself are defined in nftables by using its filtering ability on packets from tap interfaces of Qemu instances that connected with bridge. Here [6], they used openwrt, I used Ubuntu Server as operating system.

[1] talks about the CentAir routing algorithm itself, which is an algorithm which according to some graph centrality measure, adjust transmit power of the central nodes to both reduce interference rate and congestion around the those central nodes which are vulnerable to become bottlenecks, then it decides which channel to use. For path finding, it uses D* Lite algorithm. I implemented the very basic version of CentAir to Batman-Adv.

[7] talks about couple of graph centrality measures. This metrics are degree, eigenvector and pagerank centrality measures. One node's degree centrality is simply numbers of neighbors divided by number of nodes in the network. I use this metric to calculate centrality of a batman node.

In [8], there are three new modes of CentAir: resilient routing, quick routing, hybrid routing. Resilient routing is like original CentAir, quick routing enables central nodes to use medium more often before other nodes. I have not implemented these modes but nevertheless they are giving interesting perspective about the problem of this project tries to solve.

3 Developed Approach and System Model

In this section, I will talk about the virtual environment, what its components are and how they are connected with each other.

How CentAir is implemented, and incorporated to Batman driver, how user space and kernel space communicate with each other, how virtual collision logic is implemented in batman nodes are also the subject I am going to talk about and explain too.

3.1 Data Model

Batctl is the user space program that can configure, tweak the batman driver and observe its condition and statistics. Batctl and batman-adv communicate with each other via netlink sockets[9].

3.1.1 Netlink Related Data Structures

Netlink provides a asynchronous communication between kernel and user space [10]. We will see what it is and how it is connect to this project.

Netlink Attributes: They are the basic data units in netlink messages. They're type-length-value (TLV) structures that carry information between userspace and kernel space. Each attribute has:

- A type identifier (what kind of data this is)
- A length field (how much data)
- The actual data payload

BATADV_ATTR_HOPMODE_COMMAND: This is not cmd of netlink, this is directly related with the hop mode itself. This hop mode command can be stop, uniform and centair.

Stop command stops the broadcasting in the broadcasting node; uniform puts the network in uniform mode and CentAir puts the network in CentAir mode.

BATADV_ATTR_HOPMODE_PENALTY: This is the broadcasted number in the OGM packets' TVLV container. According to this number every node calculates its hop penalty.

BATADV_ATTR_COLLMODE_THRESHOLD: If current packet comes to batman driver under this threshold after the last packet, this means there happened a collision. Bigger this value, easier there to be collision in this node.

BATADV_ATTR_COLLMODE_MODE: This attribute tells in this node, the collision mode is active or not. If it is not active, collisions are not possible.

BATADV_ATTR_COLLMODE_LAST_DIFFERENCE: This tells us how many nanoseconds there are between the last two packets.

BATADV_CMD_HOPMODE: This is for cmd field of struct genl_small_ops, when batman module sees this in a netlink message it runs the function in this structure according to the defined permissions in the system. In this case, that function is batadv_netlink_hopmode.

3.1.2 TVLV Related Data Structures

TVLV stands for T(ype) V(ersion) L(ength) V(alue). OGM packets carry raw routing algorithm alongside tvlv containers which can contain various things about batman network. Hop mode uses tvlv infrastructure to broadcast and process network's data about hop mode[11].

Format of TVLV Container: Below can be seen a TVLV container's format[11]

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
TVLV Type	Version	Length	Value

Format of OGM Packet: Below can be seen an OGM packet's format[3]

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
Packet Type	Version	TTL	Alignment
Interval			
Sequence Number			
Originator Address			
Originator Address		Flags	Gateway Flags
TQ	TT Num Changes	TT VN	TT CRC

Figure 3.1: BATMAN Adv. OGM Packet Format

struct batadv_tvlv_hopmode_data: Broadcasted tvlv containers in OGM packets contains this vital structure to make possible hop mod of batman-adv. This struct containst 8 bits unsigned integer hop mode command and 16 bits unsigned integer penalty number. This penalty number is calculated according to hop mode command.

3.2 Structural Model

3.2.1 The Virtual Environment

First let's talk about the virtual environment that connects the nodes that run the batman-adv driver.

Nodes are Qemu instances, they are connected to each other via TAP interfaces and a Linux bridge.

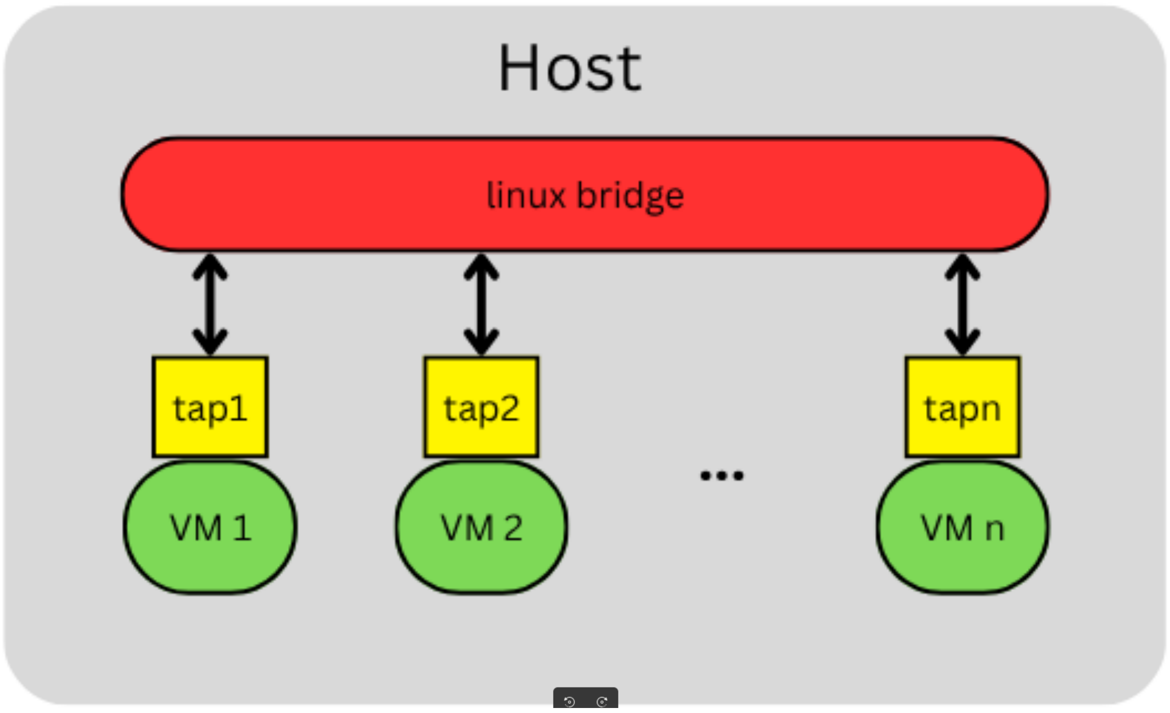


Figure 3.2: Structure of the virtual environment

To simulate multi-hop behavior and imperfect link conditions, I use nftables' filtering. Bridge's packet delivery of between TAP interfaces is called forwarding, and the filtering is applied this kind of packets[12].

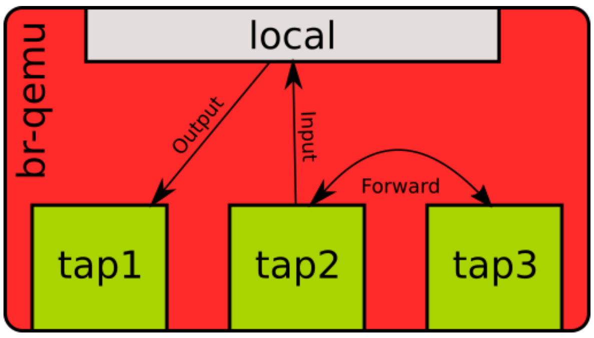


Figure 3.3: my nftable entries filters the forward packets which are the packets between TAP interfaces in the bridge

Below you can see an example of the power of the filtering. We can dictate successful packets ratio and we can even limit the bandwidth. A lot of possible with nftables but I just used limiting of the successful packet ratio feature, not the limiting the bandwidths.

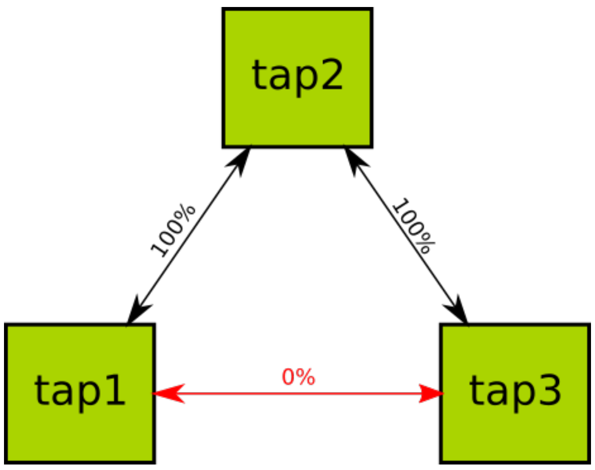
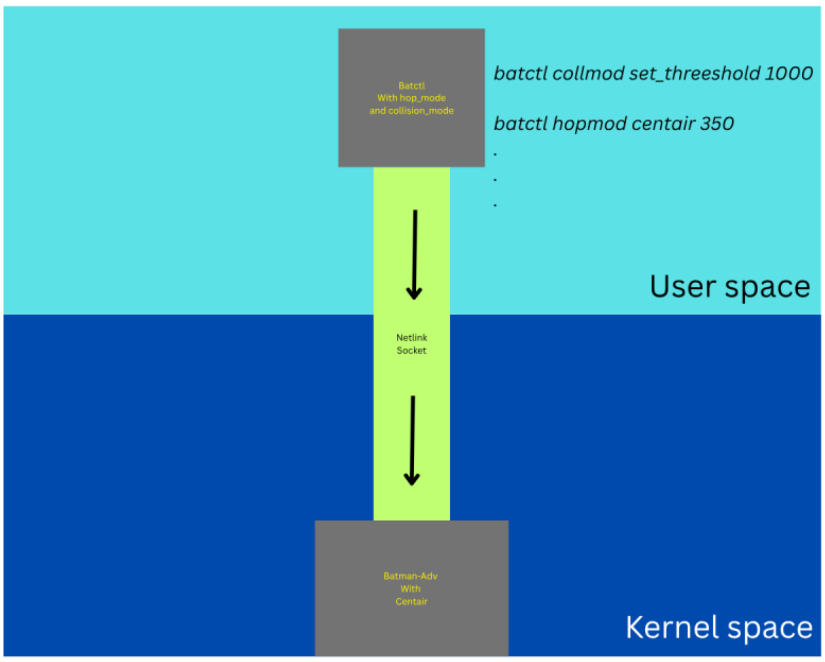


Figure 3.4: Example of the filtering between TAP interfaces

3.2.2 Interaction Between Batctl and Batman-Adv

Via batctl one can observe the statistics of the node, its neighbors and originators in the network, which neighbors it will use to send packet to any originator, etc.

In terms of this project, we can open and close the collision mode, we can set collision threshold, stop and start a hop mod, set the broadcasted hop mode, and configure hop mode's parameters. Below is a diagram that shows a simple example.



```
batctl collmod set_threeshold 1000

batctl hopmod centair 350
.
.
.
```

Figure 3.5: simple example of an interaction between batctl and batman-adv

How to Use hop mode and collision mode:

- batctl hopmod (hop mode (”uniform”, ”centair”)) (penalty number)
- batctl hopmod (hop mode¿ stop)
- batctl collmod set_mode (0,1)
- batctl collmod set_threshold (threshold number)
- batctl collmod reset: This would reset statistics of number of collisions and total number of arrived packets in the node, and set the collision mode to 0, set the threshold and the arrival time difference between last two packet INVALID.
- batctl collmod: This will print collision mode, threshold in nanoseconds and difference between last two packet’s arrival.
- batctl s: We can see number of collision and total arrived packet number since last reset

3.2.3 Configuring and Managing The Virtual Environment

This emulated batman nodes have two tap interfaces, one for the link between themselves for the tested network topology, other is between node and the host itself, so that from host computer I can control and observe them. All of the necessary configuration needed before running virtual environment is handled by the create_topology.py script. Controlling the nodes is handled by the remote_exec script.

create_topology.py: Firstly, this script takes a configuration JSON file as argument. This file contains information about which node is connected to which node, what quality every link has, bridge name that connects the nodes in the emulated network, is everything ready for emulation or not.

According to configuration file, python script prepare the two kind of TAP interfaces, one is for communication between nodes, other is for between nodes and the host. According to their type TAP interfaces gather under two Linux bridges. Nftable entries that defines links’ qualities is written and applied by this script. Every node is assigned MAC address according to number in their names. Script only assigns IP address to the bridge for the host to nodes communication, assigned IP is 192.168.102.1 as gateway. IP assignation of the node is handled by the nodes themselves according to their MAC addresses.

- ./create_topology.py bridge_name_conf_file –create : This prepare everything in the system to the bringing up of the virtual environment according to configuration file.
- ./create_topology.py bridge_name_conf_file –kill: This deletes and cleans up everything that was made for this configuration file by –create argument.

remote_exec: With this bash script, I can send and execute the same commands at the all of the batman nodes simultaneously. If I want I can target a node and by specifying its IP address I can execute the command only there. After executing the command, their stdout, stderr and their exit code parsed to my host's terminal node by node, categorized by their IP addresses. remote_exec knows the IP addresses from the hosts file which is one of the outputs of the create_topology.py. Here is how to use:

- ./remote_exec "command": This will execute the command at the all nodes
- ./remote_exec -h "IP address of a node" "command": In this case, only the node with the specified IP address will execute the code.

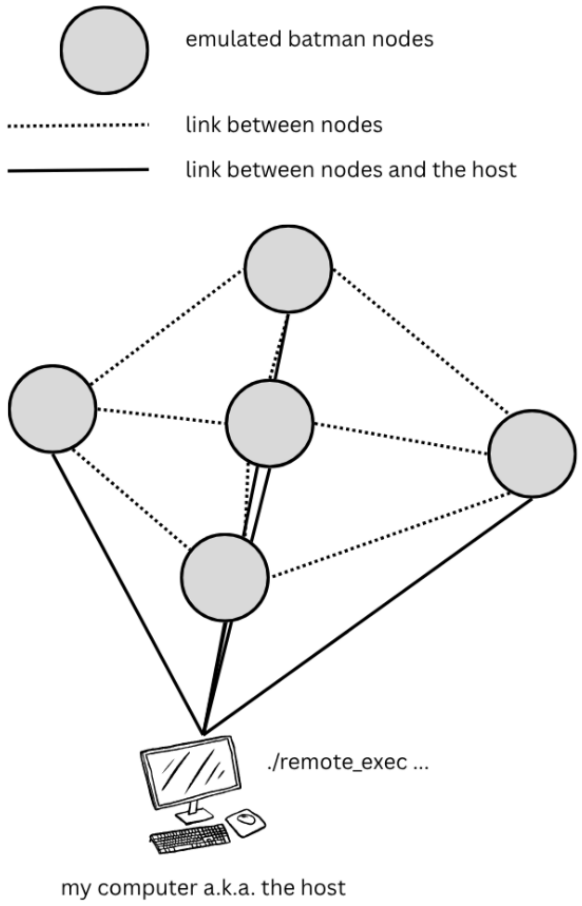


Figure 3.6: Link between the host and emulated batman nodes in the virtual environment

3.3 Dynamic Model

The drones diagram were taken from [7], their topologies change, because wifi cards' range changes according to its centrality metric in that paper's version of CentAir. In this project version's CentAir does not change topology but diagrams are very informative when it comes to the traffic density as heat diagram so I used them in here too.

3.3.1 Hop Modes

As hop mode, uniform mod is the default one. In this hop mode, according to how high hop penalty set, routing seeks shortest path between source and destination. Thus, central nodes get the highest traffic density around them. We can see it below diagram of a ad-hoc network of drones.

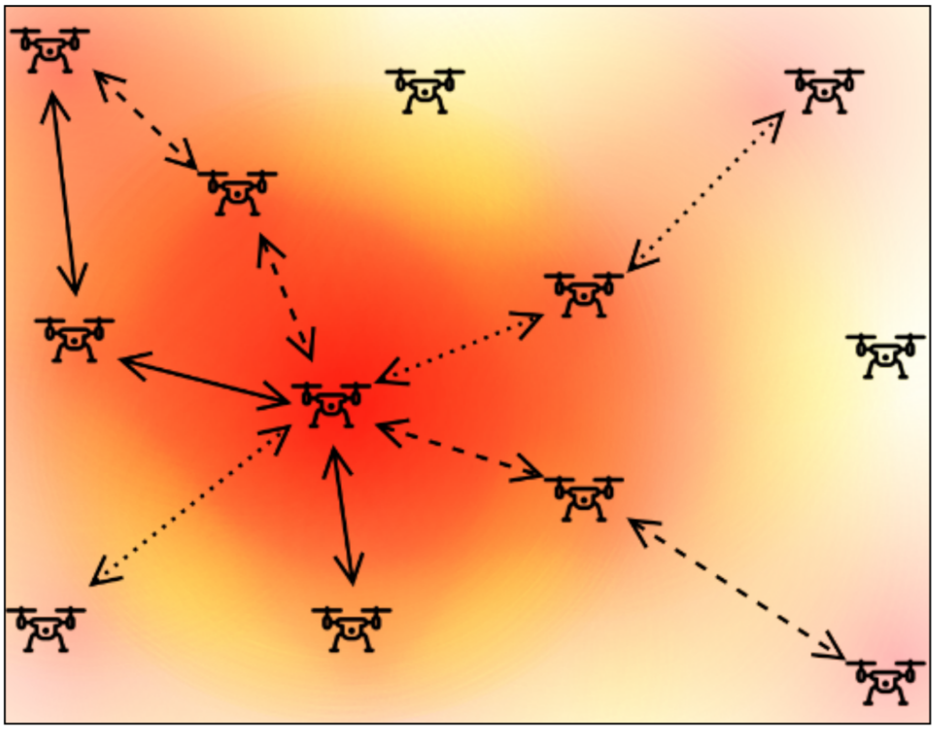


Figure 3.7: Uniform hop mode, every node has the same hop penalty[7]

If network is in CentAir hop mode, hop penalty in batman nodes are calculated according to equation below:

$$\text{Hop penalty} = (\text{broadcasted number}) \times \frac{(\text{number of neighbors})}{(\text{number of routers})} \tag{3}$$

Hence, traffic is directed away from central nodes which are prone to become bottleneck without detecting by batman-adv due to collisions, to more safer outer nodes.

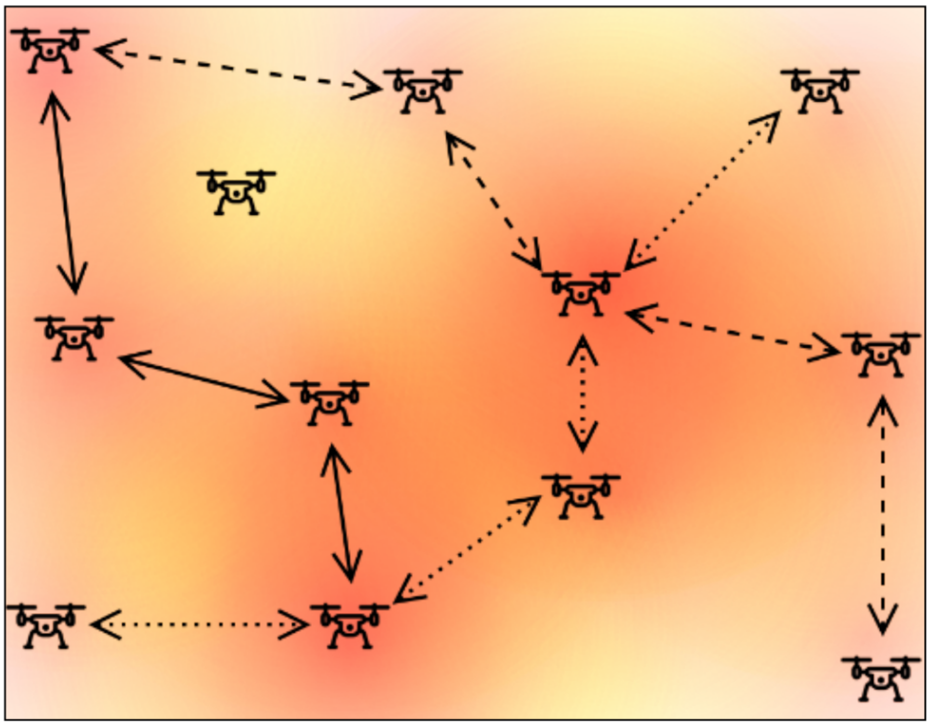


Figure 3.8: CentAir hop mode, nodes' hop penalty is proportional to number of their neighbors[7]

3.3.2 Collisions

In this virtual environment, there is no collisions in the TAP interfaces. To make packet losses from collision make possible, I implemented collision logic into the batman-adv.

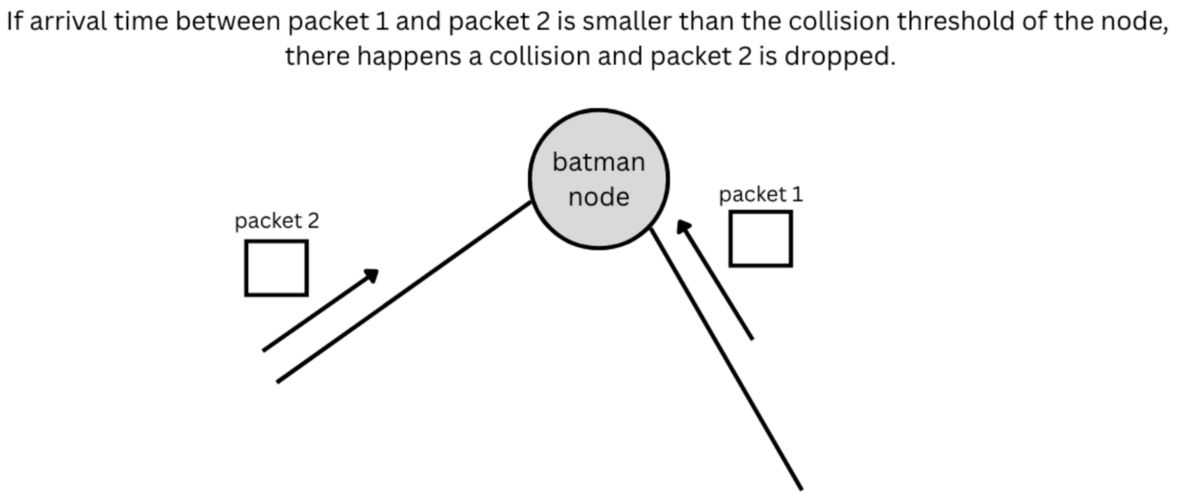

If arrival time between packet 1 and packet 2 is smaller than the collision threshold of the node, there happens a collision and packet 2 is dropped.
batman node
packet 2
packet 1

Figure 3.9: How collisions happens in this virtual environment

4 Experimentation Environment and Experiment Design

Our tested source and destination is node 0 and node 3. The tested client sends UDP packets in 25 MB/s bandwidth to its server. At the same time, other clients send their servers UDP packets in 25 MB/s too to create UDP storm. In this conditions, we observe iperf3's UDP tests' result metrics in node 3's server log.

In this tested topology, central node is slightly damaged, its collisions threshold is 10000 nanoseconds, and other nodes' thresholds are 1000 nanoseconds. Batman-Adv's OGM packets are not enough to discover central node's condition that causes it to become bottleneck during bursty traffic conditions. 10 tests for Batman-Adv without CentAir, 10 test for Batman-Adv with CentAir are made and we will see and compare those results in the next section.

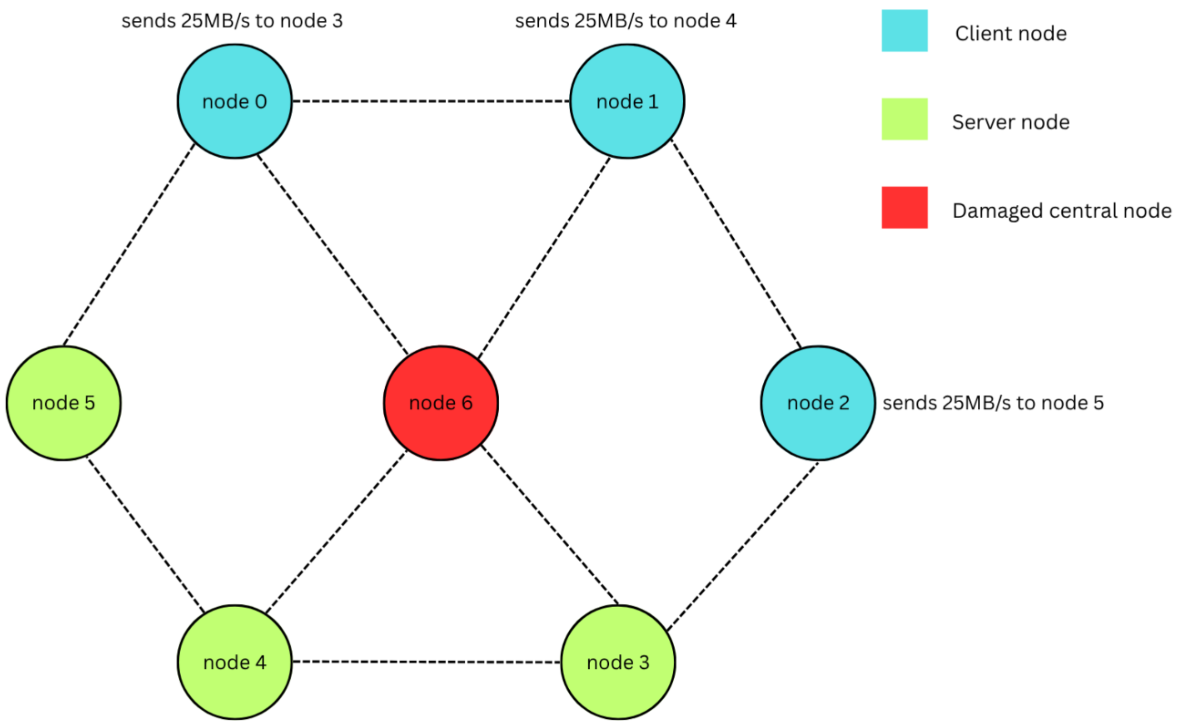


Figure 4.1: Topology of the tested network

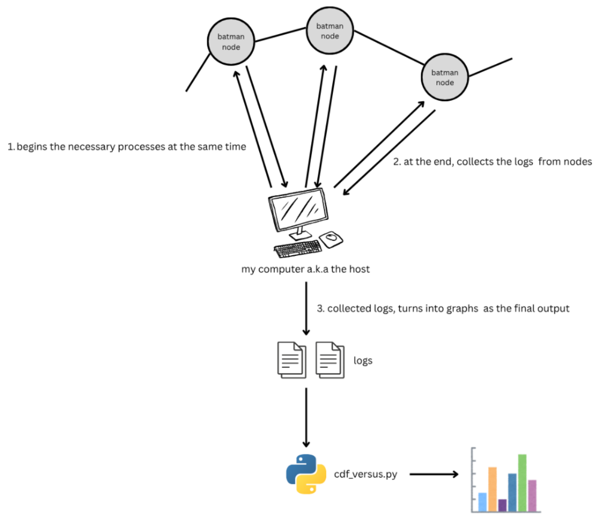

batman node
batman node
batman node
1. begins the necessary processes at the same time
2. at the end, collects the logs from nodes
my computer a.k.a the host
3. collected logs, turns into graphs as the final output
logs
cdf_versus.py

Figure 4.2: Testing pipeline

4.1 tester.sh

This script takes as arguments, UDP stormer nodes' bandwidth, client's bandwidth, is network in CentAir mode or normal vanilla Batman-Adv, and duration of the test.

It resets the state of the nodes, runs the UDP stormers' client and server processes. If client processes do not start and output nothing, it tries to start them again and again. At last, script ask the user, is everything okay?

After taking yes as an answer, it would begin the tested client process and collision collecter in every node and ask user again is everything okay? After taking yes as an answer, the test begin.

After the test finishes, all of the collision logs, and tested client's server's log are gathered into a folder named by time and by the test conditions. During graphing phase, this naming convention will be used to find collision csv logs and tested server's logs.

4.2 collect_collision.py

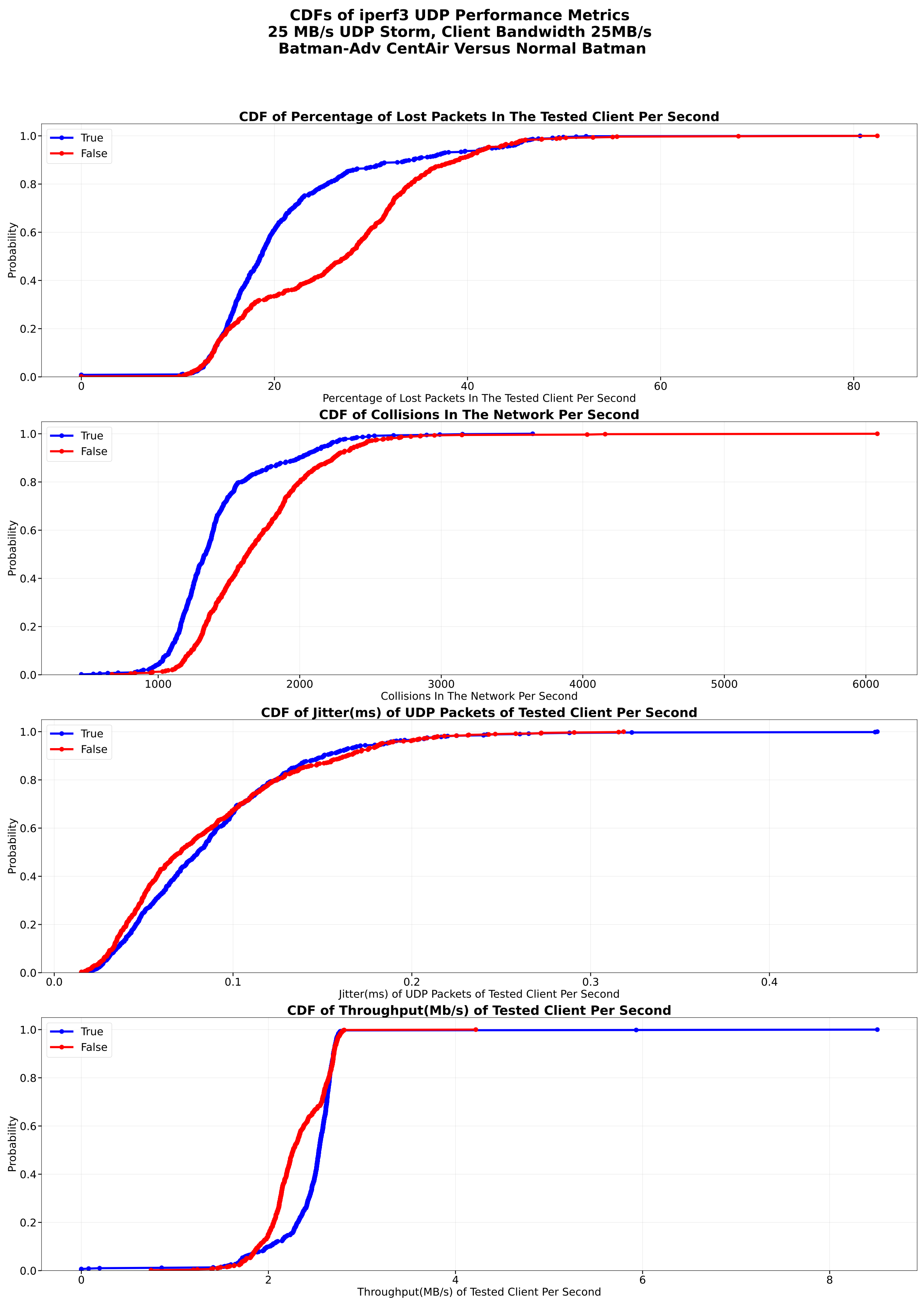
During the 60 seconds test, every node runs collect_collisions.py, which collects number of collisions per second during testing in that node and logs that number into a csv file.

4.3 cdf_versus.py

According to testing conditions, this script searches appropriate logs of those tests and graphs them as cumulative distribution function. Observer can see vanilla batman versus Batman-Adv with CentAir's results on the graphs.

5 Comparative Evaluation and Discussion

One half of the tests was running vanilla Batman-Adv, other half was running Batman-Adv with CentAir. In the tests; jitter, throughput, lost packet count, and collision numbers were compared. Actual differences between two networks were in the number of collisions in the network and percentage of lost packets in the client. According to results, Batman with CentAir reduces number of collisions in the network by 16.62% and lost Batman-Adv packets by 4.85%



Statistics:
Expected Total Number of Collisions In The Network: 101746.80
Expected Total Number of Packets From The Tested Client: 134118.60
Expected Jitter(ms) Of The Tested Client: 0.09 ms
Expected Throughput(MB/s) of The Tested Client: 2.30 MB/s
Expected Percentage of Lost Packets In The Client: 26.35%
Number of Tests:10

Statistics:
Expected Total Number of Collisions In The Network: 84831.50
Expected Total Number of Packets From The Tested Client: 134117.90
Expected Jitter(ms) Of The Tested Client: 0.09 ms
Expected Throughput(MB/s) of The Tested Client: 2.45 MB/s
Expected Percentage of Lost Packets In The Client: 21.50%
Number of Tests:10

Figure 5.1: Cumulative distribution functions of the results of the tests

6 Conclusion and Future Work

This project aims to enhance the Batman-Adv's routing algorithm where sometimes because of millisecond sensitivity of its discovery packets, it may fail to detect packet loss caused by collisions, adapt the network according to these conditions and direct away the traffic away from the dangerous nodes in time. Especially, batman is vulnerable to collisions during burst traffic and with damaged central nodes which can be susceptible to become bottlenecks under the heavy load. An admin can observe the network and decide to act upon by putting the network CentAir mode, as aggressive as he/she desire.

However; essentially, an admin sets hop penalty coefficient per neighbor in every node with this CentAir mode. He may set that coefficient 1000 or 0, one will makes all hop penalties maximum number 255, other makes hop penalties the minimum number 0. Admin may be mistaken while observing the network, a central node might not always be a bottleneck to avoid in every condition.

Thus, some weight should be taken from admin. Other parameters should be taken to account about nodes and the network, number of neighbors in a node is not enough by itself.

Let's look at our tested network topology: if central node is not damaged, CentAir makes it even worse, if it tries to direct traffic away from central node to outer nodes aggressively. In this scenario, central node is not a bottleneck and it is an reasonable node even in the bursty traffic conditions in that topology. CentAir should detect central node is reasonable or not before vanilla BATMAN-Adv, right now it cannot do that.

As well as number of neighbors in a node, we can calculate hop penalty in node according to forwarded packets compared to total packets directed to the node itself which tells us something quantitative how much a router the node is. We can check node's interval of OGM packets and check how this node's paths between this node and neighbors is perceived by its neighbors according to OGM packets; path qualities might tell us something wrong and collisions happening in this node (nothing new here) and fewness of interval tells us how much to trust these path qualities. Taking into account other parameters, we might exaggerate these values while counting them into hop penalty. We might be able to take them into account before vanilla Batman-Adv. In a network with a bursty traffic pattern, we might prepare the network for this kind of conditions with CentAir mode before the bursty traffic comes and get a better performance from network over all.

References

[1] T. T. Sarı and G. Seçinti, "CentAir: Centrality based cross layer routing for software-defined aerial networks," in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*. Las Vegas, NV, USA: IEEE, 2022, pp. 741–742.

[2] open-mesh.org, "B.a.t.m.a.n. advanced," https://www.open-mesh.org/doc/batman-adv/Wiki.html, accessed: Jun. 24, 2025.

[3] ——, "Originator message (ogm)," https://www.open-mesh.org/doc/batman-adv/OGM.html, accessed: Jun. 24, 2025.

[4] J. D. Britos, S. Arias, N. Echániz, G. Iribarren, L. Aimaretto, and G. Hirschfeld, "BATMAN adv. mesh network emulator," in *Proceedings of the Network Research Conference*. Córdoba, Argentina: Universidad Nacional de Córdoba, 2014.

[5] Open-Mesh Project, "OpenWrt in QEMU," https://www.open-mesh.org/projects/devtools/wiki/OpenWrt_in_QEMU, 2024, accessed: 2024.

[6] ——, "Advanced bridge virtual network," https://www.open-mesh.org/projects/devtools/wiki/Advanced_Bridge_virtual_network, 2024, accessed: 2024.

[7] T. T. Sarı and G. Seçinti, "Intelligent cross-layer routing framework based on D* lite for resilient aerial networks," in *2022 18th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, 2022, pp. 328–333.

[8] ——, "Using centrality based topology control for FANET survivability against jamming," *Computer Networks*, vol. 242, p. 110250, 2024, available online 15 February 2024.

[9] open-mesh.org, "Using batctl," https://www.open-mesh.org/projects/batman-adv/wiki/Using-batctl, accessed: Jun. 24, 2025.

[10] Thomas Graf, "Netlink library (libnl)," https://www.infradead.org/~tgr/libnl/doc/core.html, accessed: Jun. 24, 2025.

[11] open-mesh.org, "Tvlv," https://www.open-mesh.org/doc/batman-adv/TVLV.html, accessed: Jun. 24, 2025.

[12] nftables.org, "What is nftables?" https://wiki.nftables.org/wiki-nftables/index.php/What_is_nftables%3F, accessed: Jun. 24, 2025.