A Network Simulation Platform for Flying Light Specks

Yi Chen
UC Riverside
ychen1329@ucr.edu

Zhaowei Tan
UC Riverside
ztan@ucr.edu

ABSTRACT

Emerging applications with Flying Light Specks (FLS) require efficient network traffic exchange among nodes. To evaluate the suitability of various wireless technologies and configurations for specific applications, we designed and developed NS-FLS, a platform that allows researchers to test FLS mobility, traffic patterns, network configurations, and gather statistical performance metrics. We build NS-FLS on ns-3 and build graphical interfaces for user-friendly operation. We validated NS-FLS with traces from a real FLS application, demonstrating its capacity to assess network performance.

Holodecks Reference Format:
Yi Chen and Zhaowei Tan. A Network Simulation Platform for Flying Light Specks. Holodecks, 2(2): 12-17, 2024.
doi:10.61981/ZFSH2402
Holodecks Reference Format:
Yi Chen and Zhaowei Tan. A Network Simulation Platform for Flying Light Specks. Holodecks, 2(2): 12-17, 2024.
doi:10.61981/ZFSH2402

Holodecks Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/GeNeLa/NS-FLS.

1 INTRODUCTION

Flying Light Specks (FLS) are miniature drones equipped with lights capable of displaying diverse colors and textures [3]. These drones can detect user touch and provide kinesthetic feedback by exerting force [2, 3]. Their unique capabilities enable applications ranging from immersive, FLS-enhanced tabletop gaming, where virtual characters come to life, to healthcare, where FLS technology assists in real-time MRI analysis. Creating 3D visualizations with FLS typically requires numerous devices, sometimes in the millions.

Many FLS applications depend heavily on real-time communication, both for receiving instructions from a central orchestrator and for sharing sensory data among devices [4]. For instance, an FLS may detect obstacles and broadcast this data to neighboring units.

Proceedings of the Holodecks Foundation, Vol. 2, No. 2.
doi:10.61981/ZFSH2402

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@holodecks.quest. Copyright is held by the owner/author(s). Publication rights licensed to the Holodecks Foundation.
Proceedings of the Holodecks Foundation, Vol. 2, No. 2 ISSN 2150-8097.
doi:10.61981/ZFSH2402

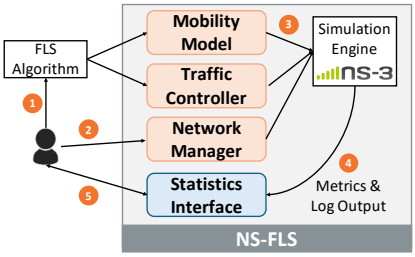


Figure 1: System Architecture of NS-FLS.

Supporting such communication demands a robust wireless network to ensure efficient real-time data exchange. Previous study has reviewed short-range communication technologies like Wi-Fi and Bluetooth that might support FLS applications [6]. The choice of technology depends on factors like drone mobility, traffic patterns, power requirements, and latency for the FLS application.

In this work, we build NS-FLS, a Network Simulator tailored for FLS systems and built on the widely used ns-3 platform [5]. NS-FLS allows researchers to input movement patterns and timed user traffic, simulating FLS communication. These traces could be generated from real FLS applications. Users can configure wireless technologies for the network simulation. NS-FLS currently supports popular network technology of Wi-Fi, Bluetooth, and Zigbee. NS-FLS will execute the simulation and return metrics such as packet loss, delay, and throughput offered as feedback.

With NS-FLS, a researcher could assess any FLS application's network performance under varied conditions. To showcase the usage of NS-FLS, we process a real FLS trace generated by Closest Available Neighbor First (CANF), an application that allows FLSs to form groups, as described in [1]. NS-FLS demonstrates its capability to evaluate the applicability and performance of several Wi-Fi variants for this application. Moving forward, we aim to enhance NS-FLS by incorporating new wireless technologies, power-related metrics, and enabling user-defined protocols.

The following sections are organized as follows. §2 presents an overview of NS-FLS. §3 describes how we implement each NS-FLS component. §4 provides preliminary results of using NS-FLS to evaluate real FLS application traces. We will discuss the future directions for NS-FLS development in §5.

2 SYSTEM ARCHITECTURE

We present overall architecture for NS-FLS in Figure 1. It consists of five major components. Three components control simulation behavior. Mobility Model can model the movement of FLSs and record the location information dynamically. Traffic Controller manages message exchange between FLS devices, and the Network Manager

configures wireless technology and parameters (e.g., experiment-specific parameters and noise). These components relay configurations and runtime information to the Simulation Engine, currently using ns-3 [5] to conduct simulations. The results will be forwarded to Statistics Interface. It will process and calculate relevant statistics and present them to users.

To use NS-FLS, a user follows the workflow as in Figure 1. In Step 1, the user runs an FLS algorithm and uses script to produce both mobility and traffic traces, following the format requirements of NS-FLS. We note that NS-FLS does not handle trace processing itself, as different FLS algorithms might use different languages and formats. For this paper, we wrote a script to convert traces from a specific usage scenario (will be discussed in §4) to NS-FLS-compatible format. In Step 2, the user specifies the network setting that she wants to evaluate. NS-FLS offers an intuitive interface for easily selecting the demanded configurations. In Step 3, NS-FLS loads all three settings into ns-3 and executes the simulation. Finally, NS-FLS collects feedback from ns-3 and provides statistics back to the user over a graphical interface.

3 DETAILS OF THE SIMULATION PLATFORM

In this section, we introduce each component in NS-FLS in detail.

3.1 Mobility Controller

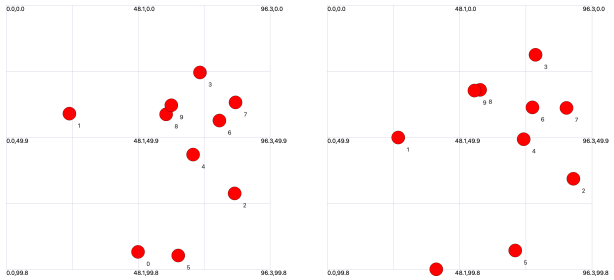
The Mobility Controller processes FLS mobility information, enabling simulation execution to update simulated node locations over time. To be compatible with ns-3, we leverage its customizable mobility model support by creating a trace-based mobility model class. It extends ns-3's mobility model with functions tailored for FLS. Below, we outline key components of this mobility class.

Class Members The class includes two primary maps: `m_trace`, which stores position data at various timestamps, and m_interpolatedTrace, an interpolated version of `m_trace` for finer-grained movement essential to accurate simulation.

loadTrace() This method loads location data into the ns-3 simulation from trace files, with each line formatted by timestamp and x, y, and z coordinates. For example, a line in "1.txt" might read "2.0 49.984 93.038 8.379," which places FLS 1 at the specified coordinates at t =2 second. We read each line from the trace file and store the information in `m_trace`, and then interpolate to `m_interpolatedTrace` if necessary.

interpolate() When an input trace file has large intervals between timestamps, we use linear interpolation to create finer-grained movements. NS-FLS allows adjusting the interpolation granularity.

updatePosition() This method uses ns-3's simulator system and updates the FLS's position continuously with `m_interpolatedTrace`. We use the current simulator's time and an iterator in maps to abstract the position and notify the ns-3 execution engine. Users can visualize these updates via ns-3's graphical interface (in 2D only, with FLS positions projected on a 2D space).



(a) t=0s (b) t=1s

Figure 2: FLS's 2D coordinates in different time.

3.2 Traffic Controller

This component records the traffic pattern from FLS algorithm, including the timestamp, sender, receiver, and packet size. We design Traffic Controller so it is compatible with the mechanism of network traffic simulation of the ns-3 platform.

NS-FLS maintains a vector called `m_packetTraces` to store each FLS's traffic information. Before the start of simulation, we first load the trace file and push packet information of each FLS into `m_packetTraces`. Each line is in the format of (*timestamp*, *packet_size*, *FLS_ID*). Since ns-3 requires a unique IP address for each node, NS-FLS maps each FLS ID into a unique IP address format (e.g., FLS 15 to IP 10.0.0.15).

After we load all traces from the input file and the simulation is started, NS-FLS creates and maintains a local socket with ns-3. This channel is used to notify the execution engine of the packets that will be transmitted during the life cycle. Traffic Controller goes through `m_packetTraces` and gets the size and destination of the next packet the simulation needs to send. A message is created accordingly to notify the Simulation Engine by triggering sendPacket() function provided by ns-3.

3.3 Network Manager

This module allows users to select the network setting to test. On a higher level, NS-FLS integrates multiple wireless technologies for simulation. In our current version, NS-FLS supports wireless standards such as Wi-Fi, BLE, and ZigBee, as shown in Table 1. We select to support them as they are studied to be potential short-range wireless technology suitable for FLS [6]. Other standards are under development and will be supported in future.

On a lower level, NS-FLS allows users to change the configurations for a wireless technology. Table 2 lists all configurations that a user can select for the simulation if Wi-Fi is selected as the technology. Such flexibility allows a user to derive the best configuration to support an FLS application. We also make a user-friendly, web-based interface as illustrated in Figure 3.

3.4 Simulation Engine

This subsection describes how we set up ns-3 for Simulation Engine. In essence, we need to load the user-input setting information into ns-3 and configure it correctly for simulation. We first create a NodeContainer instance in ns-3 which contains a list of nodes as FLSs. All our following operations work with NodeContainer as an

Table 1: Supported wireless technology for NS-FLS.

Technology	Support Status
Wi-Fi	✓
Bluetooth	✓
Zigbee	✓
Z-Wave	†
NFC	†
RFID	†
VLC	†
Infrared	†

Note: ✓: supported, †: under development.

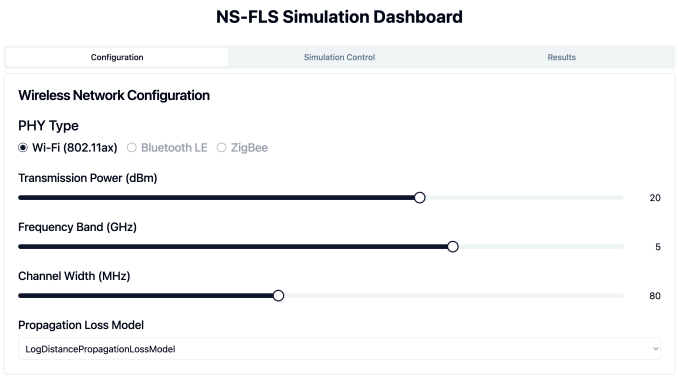


Figure 3: Dashboard for changing network configurations and getting results.

abstraction. To support FLS movement, we use MobilityHelper to install our Mobility Controller on each node, which allows them to move according to the traces. After that, we load the network settings that users want from the Network Manager. We use several network configuration helpers to set up the network environment, according to which wireless technology the user chooses, and then use NetDeviceContainer to install these settings on each node. On the application level, we set each node's traffic separately with Traffic Controller. Finally, the simulation is executed.

We now introduce some important helper functions.

NodeContainer This class is the fundamental management class in ns-3, acting as a crucial container for handling network nodes in simulation scenarios. It provides a systematic approach to organizing and managing collections of nodes, which represent Flying Light Specks in our simulation environment.

MobilityHelper It is used to install our Mobility Controller on each node in the NodeContainer. Through this setup, all nodes can move according to our scheduled movement patterns during the simulation. MobilityHelper helps us manage and control the movement for all nodes at the same time.

Wireless Network Config Helpers NS-FLS acquires the settings that users want through Network Manager, and then uses different config helpers to set these settings in the simulation engine. For instance, if Wi-Fi is selected as the technology, WifiHelper manages the main Wi-Fi settings and handles network devices, while other Helper classes set up information such as channel conditions (YansWifiChannelHelper), and physical settings (YansWifiPhyHelper).

```
Node 34:                    Flow 538 (10.1.1.29 -> 10.1.1.35)
  Transmitted Packets: 2359   Tx Packets: 41
  Received Packets: 1107      Rx Packets: 41
  Lost Packets: 1127          Lost Packets: 0
  Packet Loss Rate: 47.7745%  Packet Loss Ratio: 0%
  Mean Delay: 5.57748 ms      Throughput: 0.00333257 Mbps
  Mean Jitter: 5.1638 ms      Mean Delay: 57.8482 ms
  Throughput: 0.0674848 Mbps  Mean Jitter: 62.5902ms
Node 23:                    Flow 569 (10.1.1.15 -> 10.1.1.21)
  Transmitted Packets: 2219   Tx Packets: 39
  Received Packets: 777       Rx Packets: 39
  Lost Packets: 1240          Lost Packets: 0
  Packet Loss Rate: 55.881%   Packet Loss Ratio: 0%
  Mean Delay: 4.70379 ms      Throughput: 0.00290461 Mbps
  Mean Jitter: nan ms         Mean Delay: 2.73451 ms
  Throughput: 0.0537369 Mbps  Mean Jitter: 1.883ms
```

Figure 4: Result Statistics.

NetDeviceContainer In ns-3, the net device represents both the software driver and the simulated hardware. A net device is installed in a Node so it can communicate with other Nodes through Channels in the simulation. Similar to creating a NodeContainer that manages nodes, we create a NetDeviceContainer for a set of net devices and manage these different net devices. After setting up the network environment, we use NetDeviceContainer to install this setting on each node in NodeContainer.

3.5 Statistics Interface

NS-FLS retrieves results from ns-3 and presents the results in the dashboard as shown in Figure 3. NS-FLS currently supports printing seven metrics statistics: Transmitted Packets, Received Packets, Lost Packets, Packet Loss Rate, Mean Delay, Mean Jitter, and Throughput. We print these metrics for each FLS node and for each flow between two nodes. Figure 4 shows an example of printed output for a round of experiments.

4 PRELIMINARY EVALUATION RESULTS

To showcase how to use NS-FLS for networking for FLS research, we replay a real FLS trace generated by Closest Available Neighbor First (CANF) as described in [1]. This scenario includes 454 static nodes, and each node continues to send broadcast messages. We convert the traces kindly offered by the paper's authors into compatible formats, and execute the simulation engine for 30 seconds. We compare the performance of this application with two Wi-Fi variants: Wi-Fi 802.11b and Wi-Fi 802.11ax. 802.11ax is a newer version with higher throughput available.

Figure 5 shows the comparison between for node453. While the performance on the sending side for this node stays the same (33 packets and 18,108 bytes sent, 0.00536255 Mbps Send Rate), the receiving performance shows clear differences. The number of received packets increased by 1146.3% from 918 to 11,441. The correctly delivered bytes increase by 1119.7%, from 430.9 KB to 5255.6 KB. The receiving throughput improves by 1084.6% from 0.13 Mbps to 1.54 Mbps. For all FLSs, we see similar improvements. The number of total packets that are successfully received increases

Table 2: Configurable Parameters for Wi-Fi Network in NS-FLS.

Configuration Parameter	Description	Available Options
Basic Settings		
Wi-Fi Standard	Selection of IEEE 802.11 standard	802.11a/b/g/n/ac/ax
Channel Settings	Channel bandwidth and frequency band configuration	20/40/80/160 MHz; 2.4/5/6 GHz
Advanced Features		
Transmission Power	Transmission power levels for the wireless interface	0-30 dBm
Power Save Mode	Set power save mode for FLS devices	On/Off
MIMO Configuration	Single-user or multi-user MIMO capabilities	SU-MIMO/MU-MIMO
Number of Antennas	Number of physical antennas on the device	1x1, 2x2, 4x4, 8x8
Spatial Streams	Maximum supported Tx/Rx spatial streams	1-8 streams
Beamforming	Beamforming capability configuration	Enabled/Disabled
QoS Support	Enable QoS supported for wireless network	Enabled/Disabled
Error Model	Model for simulating transmission errors	YANS/NIST/Default
Frame Capture Model	Model for handling concurrent signal reception and interference	SimpleFrameCaptureModel
Rate Control Algorithm	Algorithm for adapting transmission rates	Aarf/Amrr/Arf/Cara/Onoe/Ideal/Minstrel
MAC Channel Access Mechanisms	Method used to access the wireless medium	DCF/EDCA
Mobility Model	Model for node movement patterns	Random Walk/Waypoint/Constant etc.
Simulation Parameters		
Guard Interval	Time gap between transmitted symbols	0.4/0.8/1.6/3.2 µs
Retransmission Limit	Maximum number of retransmission attempts	4-7 attempts
RTS/CTS Threshold	Size threshold for RTS/CTS packet	0-2347 bytes
Fragmentation Threshold	Maximum size of a packet that can be transmitted without fragmentation	256-2346 bytes
Contention Window Size	Min/Max contention window size setting	CWmin: 15, CWmax: 1023
Slot Time	Duration of a slot in MAC protocol	9/20 µs
Beacon Interval	Time between beacon frame transmissions	20-1000 ms/50ms by default
CCA Threshold	Clear channel assessment sensitivity threshold	-82 to -62 dBm
Energy Detection Threshold	Minimum energy level for signal detection	-100 to -60 dBm
Channel Switch Delay	Time required for channel switching	1-100 ms
Antenna Gain	Rx/Tx antenna gain values	0-30 dBi
MAC Queue Parameters	Queue delay limit and size configurations	Size: 100-1000 packets Delay:100-1000ms
Block Ack Window	Size of block acknowledgment window	16-64 frames
Environmental Settings		
Propagation Loss Model	Model for signal propagation loss	Friis/Log-distance/Nakagami
Interference Sources	Configuration of interference effects	None/Adjacent/Co-channel
Noise Parameters	Background noise and thermal noise settings	-96 to -90 dBm
Building/Terrain Effects	Environmental obstacles and terrain effects	Indoor/Outdoor/Hybrid

by 1145.6% from 416,307 to 5,185,491, and the total received bytes increase by 1119.0% from 195.4MB to 2.4GB.

These significant improvements are aligned with the technical advances from 802.11b to 802.11ax. 802.11ax brings several key improvements: it uses a wider bandwidth with more efficient OFDMA technology for data transmission, and supports advanced PHY technology such as MU-MIMO and interference management. These features help reduce packet collisions and data loss in dense networks, such as our FLS scenario with 454 nodes.

```
Node 453 statistics:
  Sent:
    Packets: 33
    Bytes: 18108
    Send Rate: 0.00536255 Mbps
  Received:
    Packets: 11441
    Bytes: 5255622
    Receive Rate: 1.53599 Mbps

Overall Statistics:
  Total Sent: 55092 packets (25234453 bytes)
  Total Received: 5185491 packets (2382258597 bytes)
  Average Reception per Node: 11421.8 packets
```

(a) Wi-Fi Standard = 802.11ax

```
Node 453 statistics:
  Sent:
    Packets: 33
    Bytes: 18108
    Send Rate: 0.00536255 Mbps
  Received:
    Packets: 918
    Bytes: 430852
    Receive Rate: 0.126358 Mbps

Overall Statistics:
  Total Sent: 55092 packets (25234453 bytes)
  Total Received: 416307 packets (195430089 bytes)
  Average Reception per Node: 916.976 packets
```

(b) Wi-Fi Standard = 802.11b

Figure 5: Test results using different Wi-Fi standards.

5 FUTURE STEPS

We aim to improve NS-FLS in the following aspects.

Supporting More Network Technologies As discussed in §3.3, current NS-FLS supports three most popular short-range wireless technologies, Wi-Fi, Bluetooth, and Zigbee. We aim to expand support to a broader range of technologies, as listed in Table 1, to enable a comprehensive evaluation across diverse wireless options.

Programmable Interfaces While NS-FLS currently offers graphical interfaces, researchers may benefit from programmatic control, which allows them to enumerate configurations and compare results automatically. We plan to add interfaces to run simulations programmatically, retrieve results, and generate custom outputs.

Estimating Power Consumption At present, NS-FLS tracks performance metrics like delivery rate and throughput. However, power consumption is crucial in FLS systems. We plan to integrate energy consumption estimation during simulation to support energy-aware analysis.

We will measure energy consumption in two main parts. The first part covers FLS's self-movement energy use during activities like hovering in the sky, flying, and releasing light. The second part looks at the energy used for communication, such as when receiving or sending data packets. The ns-3 energy framework has multiple modules available, and we will use two of them for our work. For the first part, we will extend the DeviceEnergyModel and adapt it for FLS. For the second part, we will use the Radio Energy Model to simulate how much energy is used during communication.

Support Customized Algorithms Standard technologies may not always meet specific application needs. Researchers may wish to test novel network protocols optimized for certain FLS usage scenario. In addition to standardized technologies, NS-FLS will enable NS-FLS to support user-defined wireless technologies on both PHY and MAC layers.

6 CONCLUSION

This paper introduces NS-FLS, a platform specifically designed for network simulation in FLS applications. NS-FLS enables researchers to evaluate the network requirements of an FLS application before actual implementation. It encourages further exploration of FLS networking, while in the meantime deriving new insights for development of FLS algorithms that make efficient use of network.

REFERENCES

[1] Hamed Alimohammadzadeh, Heather Culbertson, and Shahram Ghandeharizadeh. 2023. An Evaluation of Decentralized Group Formation Techniques for Flying Light Specks. In *Proceedings of the 5th ACM International Conference on Multimedia in Asia*. 1–7.

[2] Yang Chen, Hamed Alimohammadzadeh, Shahram Ghandeharizadeh, and Heather Culbertson. 2023. Towards Enabling Complex Touch-based Human-Drone Interaction.. In *IROS Workshop on Human Multi-Robot Interaction* (Detroit, USA).

[3] Shahram Ghandeharizadeh. 2021. Holodeck: Immersive 3D Displays Using Swarms of Flying Light Specks. In *ACM Multimedia Asia* (Gold Coast, Australia).

[4] Shahram Ghandeharizadeh. 2022. Display of 3D Illuminations using Flying Light Specks *(ACM Multimedia)*.

[5] George F Riley and Thomas R Henderson. 2010. The ns-3 network simulator. In *Modeling and tools for network simulation*. Springer, 15–34.

[6] Zhaowei Tan. 2023. Enabling Physical Link for Flying Light Specks. In *The First International Conference on Holodecks* (Los Angeles, California) *(Holodecks '23)*. Mitra LLC, Los Angeles, CA, USA, 23–26. https://doi.org/10.61981/ZFSH2306