VAST: A Decentralized Open-Source Publish/Subscribe Architecture

Victory Opeolu*
25818198@sun.ac.za
Electrical & Electronic Engineering Stellenbosch University
21571767@sun.ac.za, Western Cape, South Africa

Shun-Yun Hu
Imonology Inc.
Taiwan
syhu@imonology.com

Charl Marais
Electrical & Electronic Engineering Stellenbosch University
Stellenbosch, Western Cape, South Africa
21571767@sun.ac.za

Prof. Herman Engelbrecht
Electrical & Electronic Engineering Stellenbosch University
Stellenbosch, Western Cape, South Africa
hebrecht@sun.ac.za

ABSTRACT

Publish/Subscribe (pub/sub) systems have been widely adopted in highly scalable environments. We see this especially with IoT/IIoT applications, an environment where low bandwidth and high latency is ideal. The projected growth of Iot/IIoT network nodes are in the billions in the next few years and as such, there is a need for network communication standards that can adapt to the ever-growing nature of this industry. While current pub/sub standards have produced positive results so far, they all adopt a "topic" based pub/sub approach. They do not leverage off modern devices having spatial information. Current open-source standards also focus heavily on centralized brokering of information. This makes the broker in this system a potential bottleneck as it means if that broker goes down, the entire network goes down. We have developed a new, unique and innovative open-source pub/sub standard called VAST that leverages spatial information of modern network devices to perform message communication. It uses a unique concept called Spatial Publish/Subscribe (SPS). It is built on a peer-to-peer network to enable high scalability. In addition to this, it provides a Voronoi Overlay to efficiently distribute the messages, ensuring that network brokers are not overloaded with requests and ensures the network self-organizes itself if one or more brokers break down. It also has a forwarding algorithm to eliminate redundancies in the network. We will demonstrate this concept with a simulator we developed. We will show how the simulator works and how to use it. We believe that with this simulator, we will help encourage researchers adopt this technology for their spatial applications. An example of such is Massively Multi-user Virtual Environments (MMVEs), where there is a need for a high number of spatial network nodes in virtual environments.

MMSys '23, June 7–10, 2023, Vancouver, BC, Canada

ACM ISBN 979-8-4007-0148-1/23/06.
https://doi.org/10.1145/3587819.3592554

CCS CONCEPTS

• **Computer systems organization** → **Client-server architectures**; **Peer-to-peer architectures**; *Self-organizing autonomic computing*; • **Networks** → *Data path algorithms*; *Network performance modeling*; • **Computing methodologies** → *Self-organization.*

KEYWORDS

open-source, publish/subscribe, pub/sub, Spatial publish/subscribe, VAST, Voronoi Partitioning, MQTT, IoT, IIoT, MMVE

ACM Reference Format:
Victory Opeolu, Shun-Yun Hu, Charl Marais, and Prof. Herman Engelbrecht. 2023. VAST: A Decentralized Open-Source Publish/Subscribe Architecture. In *Proceedings of the 14th ACM Multimedia Systems Conference (MMSys '23), June 7–10, 2023, Vancouver, BC, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3587819.3592554

1 INTRODUCTION

Publish/Subscribe (pub/sub) architecture is a common theme across scalability-driven environments [29]. It is a design pattern that provides a framework for exchanging messages between publishers and subscribers.

With Machine-to-Machine (M2M) communication, pub/sub algorithms have been proven extensively to be scalable, flexible and highly efficient [34]. The IoT space is growing at exponential rates, with projections of 100 billion connected IoT devices by 2025 [37]. We see a similar trajectory with Massively Multi-user Virtual Environments (MMVEs) [25]. This growth combined with the constant evolving needs of industry means pub/sub protocols are faced with newer challenges and communication requirements.

The aim of this paper is to showcase a new pub/sub standard we call VAST. VAST is a multilayered pub/sub algorithm, built with a distributed architecture, a self-organizing Voronoi Overlay Network (VON) Partition to effectively distribute messages sent, and a forwarding algorithm to eliminate redundancy. VAST is a 20+ year ongoing research project [20], and we are designing this algorithm to be robust enough to support arbitrary data (i.e. any application that has some form of spatial data involved). We discuss how it works with a unique concept called Spatial Publish/Subscribe. We will also demonstrate how it works with the simulator we developed, with relevant links to access the resources required.

Before then, we provide a bit of background as to pub/sub standards used in IoT/IIoT applications in the next section. We will also touch on Massively Multi-user Virtual Environments (MMVEs), an application we believe this algorithm can prove to be useful.

2 BACKGROUND & THEORY

In this section, we discuss in more detail how pub/sub works, some standards that are being used, and a short introduction to a pub/sub library called VAST. We will also discuss Massively Multi-user Virtual Environments (MMVEs) briefly as it is a field that the VAST library can be extended further to support.

2.1 Publish/Subscribe Architecture

This architecture is an interaction pattern that characterizes the exchange of messages between publishing and subscribing clients [26]. It involves a messaging pattern where senders of messages (publishers) are not programmed to send messages directly to specific receivers (subscribers). Instead, the messages are categorized into classes by a "broker", ensuring the subscribers do not have the knowledge of their publishers and vice-versa. Subscribers generally show interest in receiving specific messages and publishers simply publish messages without specifying the destination client for a message. These classes are generally referred to as topics or channels [8, 36].

This pattern is used primarily because it provides better network scalability and a more dynamic network topology. It is also used to enable event-driven architectures, or decouple applications to improve overall performance, reliability and scalability [26].

Fig. 1 illustrates the working of a channel-based pub/sub.

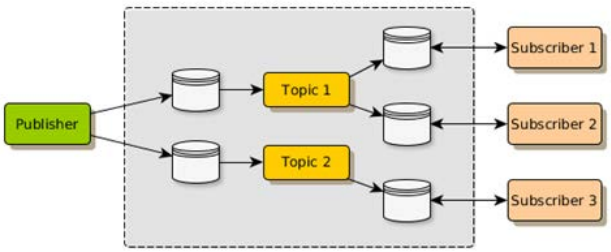

Figure 1: Standard pub/sub architecture [40]

With IoT/IIoT's, there are a few standards used employ the pub/sub architecture. They include (and not limited to):

- Message Queuing Telemetry Transport (MQTT) [33]
- Zero Overhead Network Protocol (Zenoh) [2]
- Constrained Application Protocol (CoAP) [1]
- Advanced Message Queuing Protocol (AMQP) [42]
- Extensible Messaging and Presence Protocol (XMPP) [9]

A survey by [34] documents and does a quality review towards these various pub/sub standards. We will however discuss MQTT and Zenoh standards briefly. We discuss MQTT because it is widely used as a pub/sub standard in IoT applications and Zenoh because the core tech is slightly related to what we are demonstrating. We will also briefly talk about MMVEs and why they are relevant to this research.

2.1.1 *Message Queuing Telemetry Transport (MQTT).*

MQTT is the most widely used communication protocol for IoT networks that utilizes a standardised topic/channel based pub/sub protocol [5]. It was designed for use in constrained environments where small code footprints are required and network bandwidth is limited. It is a lightweight pub/sub messaging protocol ideal for low-bandwidth and high-latency environments [33].

Each published message must be associated with a topic, and the MQTT broker broadcasts the message to all clients subscribed to that topic. This is illustrated in Fig. 2. In this case, temperature sensors become the "publishers", different PCs becomes the "subscribers", and you have a server in the form of a "Broker" facilitating the exchange of messages.

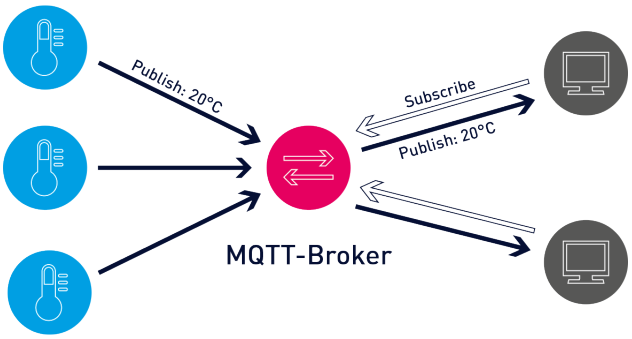

Figure 2: MQTT pub/sub architecture [35]

2.1.2 *Zero Overhead Network Protocol (Zenoh).*

Zenoh is a pub/sub protocol unifying data in motion, data at rest and computations [2]. It combines traditional pub/sub algorithms with geo-distributed storage, queries and computations, while retaining a level of time and space efficiency that is well beyond any of the mainstream stacks [2, 9].

Zenoh was designed to be efficient and robust with respect to asymmetric systems and varying interaction models, such as device on duty cycles, especially with edge[1] applications [2]. The Zenoh protocol enables efficient scalability (scaling up and down). This scalable flexibility means it can be used for applications such as microservices, to seamlessly reach Internet-scale. While Zenoh provides a distributed open-source pub/sub architecture, it does not have an network overlay partition.

2.1.3 *Massively Multi-user Virtual Environments (MMVEs).*

Massively Multi-user Virtual Environments (MMVEs) are online virtual worlds where a vast number of users can interact with each other and the environment in real-time [41]. A key feature of MMVEs is the use of event-driven mechanisms to manage communication between users. Users can send messages to each other and the virtual environment, triggering events that other users can respond to. This allows for a more dynamic and interactive experience, where users can collaborate and coordinate their actions.

However, as the number of users in MMVEs grow, especially on a global scale, MMVEs can become expensive to maintain [20].

[1]Edge computing is a new computing paradigm that performs computing at the edge of the network i.e. It takes computing closer to the source of the data[4]

We see this especially in server-client architectures, which is what a lot of MMVEs are built on. The sheer volume of messages exchanged between users can strain the system's resources and result in slower response times. This has led to a need for new standards and technologies to help MMVEs adapt to the growing market. This event-driven design of MMVEs makes it a good platform to implement a pub/sub algorithm to facilitate message exchanges.

There has been some early testing of VAST in virtual environments [17]. The most recent being by Miguel Smith [38], who was able to extend the VAST library to support Minecraft. Each time, VAST demonstrated itself to be a highly scalable algorithm. In the past however, these concepts were usually demonstrated with isolated layers of the library. For example, Spatial pub/sub and Voronoi Overlay Network were the layers used at the time it was extended to support Minecraft. VAST has however been updated since then. The algorithm's key functionalities are discussed in further detail in section 3.

2.2 Publish/Subscribe Architecture Limitations

There are various publications that speak to the limitations of pub/sub standards like MQTT. They include payload limits, message deliverability, security concerns, performance issues that come with added security layers, etc [13, 39]. The issues we see our research solving are discussed below.

- The first is that most pub/sub systems make use of channel/topic based messaging i.e. publishers send a message and subscribers "subscribe" to specific topics that the broker uses to disseminate information. They do not cater for applications with spatial requirements. There are some that have geo-support like geoMQTT [14, 15] and even Zenoh. However, it is limited to geo-location and does not necessarily support generic spatial locations. It is also limited to MQTT applications, meaning it is not robust to extend to fields that have spatial pub/sub requirements e.g. Massively Multiuser Virtual Environments (MMVEs). We have observed that more and more network devices are being built with some spatial capability [24, 32]. As such, it raises the need for more robust pub/sub protocols that can cater to and leverage off this capability.
- Another issue we observed is that there is not a lot of support for distributed/decentralized systems. The few that do are usually missing a partitioning overlay network to manage server loads or they don't have an efficient algorithm for message forwarding. The topic-based approach used in pub/sub means there is an increased need for computing power on the broker, thus a need for distributed architectures. There are publications that made an attempt to solving this with varying results [7, 12, 28] but there has not been a drive to create an open-source industry standard. We highlight this as a concern because decentralized and distributed architectures have been proven to enable high scalability [3, 21, 30]. A distributed/decentralized pub/sub architecture will enable hyper-scalability. It would combine the best of both pub/sub and decentralized scaling capability. By making it open-source, it facilitates accelerated growth because it will provide a platform to collaborate with multiple researchers and developers.

We have introduced a unique and innovative concept called spatial publish/subscribe (or spatial pub/sub). Simply put, it is an approach that leverages spatial data to perform communication. So rather than a broker matching publishers and subscribers based on their topics, it brokers that communication dynamic using the spatial location. This concept is encompassed in our network library called VAST. We discuss VAST in more detail in the next section.

3 SPATIAL PUBLISH/SUBSCRIBE ARCHITECTURE (SPS)

The spatial pub/sub architecture [17] is an extension of the channel-based pub/sub design architecture. The key difference is how the architecture matches and filters messages and subscriptions. Standard pub/sub systems performs a match based on the "channel" (topic) while spatial pub/sub performs its match based on the intersection between subscription and publishing areas [19].

SPS is a scheme whereby subscriptions and publications are associated with a point, an area or volume in a spatial or virtual environment. As illustrated in Fig. 3, for two-dimensional environments, there are four key SPS operations:

- An area subscription specifies the intent to receive all point publications that fall within the specified subscription area and all area publications that overlap with the subscription area.
- A point subscription specifies the intent to receive all area publications that cover the specified point.
- An area publication sends a message to an area and is received by any point subscriptions within the area and area subscriptions that overlap with it.
- A point publication sends a message to a point and is received by all area subscriptions that cover the point.

Note that each area is defined by the coordinates x, y, r, where x and y denote the Cartesian coordinates of the center point of the area, and r denotes the radius of the area.

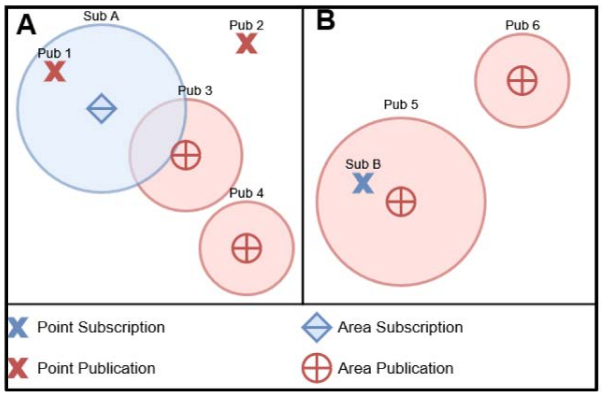

Figure 3: Spatial Publish/Subscribe: Area and Point Publications and Subscriptions

In a spatial pub/sub scenario, a node may send a message to its specified publication area, which will then be delivered to all nodes whose subscription area overlaps with the publication area. These pub/sub areas are then allowed to be updated continuously with node movements. Publication and subscription requests are sent from a participating node to an interest matcher, whose responsibility is to record the requests and match a published message with subscribers that are interested in the message (i.e. interest management of the publishers and the subscribers) [17].

Using Fig. 2 as a reference, SPS in this case means that a client will be subscribed to temperature messages originating from a spatial location. Temperature messages will also be published to a specific spatial point or area. At this point, interest management is still centralized. This means that the performance of the system is limited to the available resources of the broker. i.e. the overall performance of the interest matching is limited by how powerful the broker is.

To perform scalable SPS, the system load must be distributed between multiple brokers, with each broker managing clients, subscriptions, and publications that fall within its spatial region. VAST is a lightweight network library that is able to perform this role in SPS. VAST is discussed in further detail in the next section.

3.1 VAST Architecture

A key component of this research is the VAST library. VAST has a peer-to-peer network layer and a SPS architecture that utilises Voronoi partitioning to distribute loads between multiple SPS brokers (we refer to them as matchers). Each matcher has a position in a virtual environment and a Voronoi region and is responsible for all clients within its associated region. It also has a forwarding algorithm we call VoroCast, which eliminates redundancies in the network. This concept is illustrated in Fig. 4.

We believe VAST can become a new standard in pub/sub communication. This library is designed to be robust, highly scalable and can potentially support multiple applications where there is some form of spatial data requirement.

3.1.1 P2P Network. The network component of the library was built using a decentralized peer-to-peer (P2P) approach. It was designed such that nodes can JOIN the network, LEAVE the network, as well as MOVE their spatial locations within the network [16]. A P2P architecture means there is no single point of failure, and thus local message dissemination will continue without interruption if a node unexpectedly leaves for some technical reason. A key advantage of using P2P networks is system reliability & security [10]. If one component is compromised, the rest of the network is not compromised and will remain operational. Even if a cyberattack did breach a single node in the system, there is no guarantee that it would contain the valuable data the attackers are hoping to find and exploit.

3.1.2 VON (Voronoi Overlay Network) Partitioning. The Voronoi Overlay Network partitioning is designed to manage the spatial partitioning of the brokers. This is to ensure nodes are not overloaded or underutilized as clients and/or matchers JOIN, MOVE, or LEAVE the network. This approach takes the spatial relationship into consideration and not the physical proximity. This layer makes

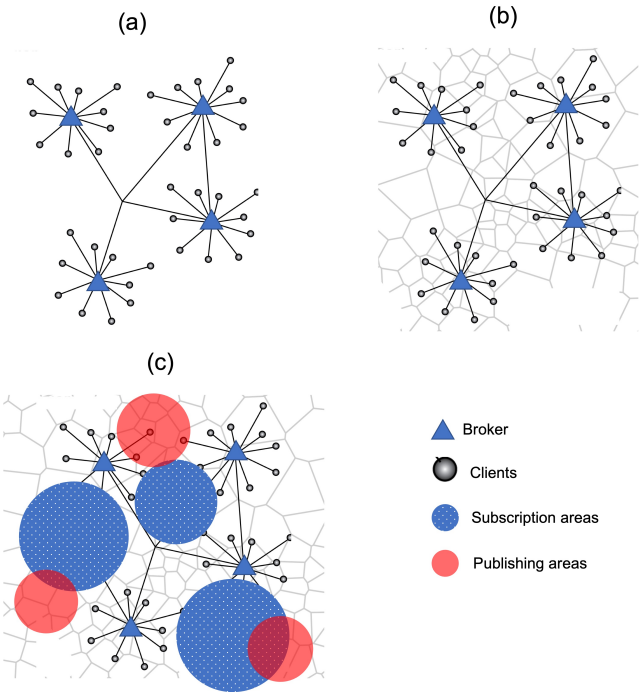

Figure 4: VAST Breakdown. Note that the layers are all in one library. We separate them into 3 for simplicity. (a) This illustrates the P2P network, where clients are managed by different broker in closest proximity. (b) illustrates the Voronoi partitioning on top of the P2P network and (c) illustrates brokers/matchers exchanging messages over subscription and publishing areas. Note that this happens through a forwarding algorithm and it is discussed further, later in this section.

use of the Voronoi partitioning concept, one that has been well researched [11]. This partitioning allows for the identification of enclosing and boundary neighbors for a given site. The enclosing neighbors are defined as neighbors whose regions share a common edge with a given node's own region while the boundary neighbors are defined as the nodes whose regions overlap with the node's Area of Interest (AOI) boundary. This concept is illustrated in Fig. 5. Note that there are scenarios where an enclosing neighbor may also be a boundary neighbor.

3.1.3 VON Spatial Forwarding. We describe VoroCast, a spatial forwarding algorithm that utilises the concept of selecting children based on tree-root distances as proposed in [27]. VoroCast is an algorithm that constructs a spanning tree across all matcher neighbors using data from the VON layer, allowing messages to be sent without redundancy. It basically performs greedy forwarding using a spanning tree from information gathered from the VON layer (neighbor identities and positions). The spanning tree is designed to identify the most efficient pathway to the destination.

It utilizes parent and child node rules, such that after a matcher has been made a "parent", the matcher notifies its immediate neighbors of the status so that no other node can become a parent of the

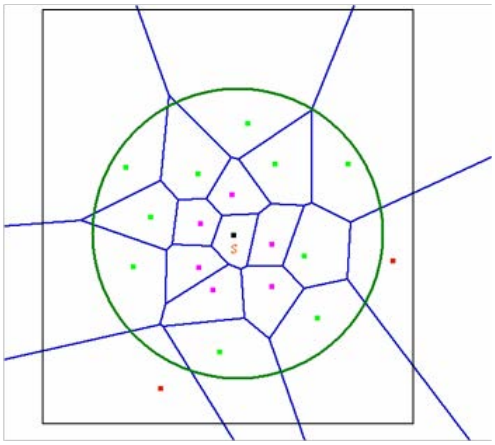

Figure 5: Voronoi Overlap Partitioning where S indicates an area, pink squares indicate the enclosing neighbors and the green squares indicate the boundary neighbors [18]

same neighbor thus eliminating repetitive forwarding. We use this approach because it has been tested and proven to be a much better forwarding model compared to standard peer-to-peer models. This model is designed to account for moving matchers as well, making it a useful tool for dynamic cases. VoroCast is well documented in [27], with results showing the scalability capabilities. To understand this further, interested readers are encouraged to consult this paper by Charl [31]. He addresses concepts like the neighbors, how they are chosen, its limitations and so on.

4 VAST SIMULATOR

Before we go into how the simulator works, we first breakdown the key terminology to using the simulator.

4.1 Relevant Terminology

- **Gateway Matcher**: This is the entry point to the network for all new clients. The gateway matcher is the entry point to the VON for all new matchers. Currently, there may only be a single gateway on both layers, but clients are able to re-enter the network from any matcher. While support for multiple gateways is achievable, it has not been implemented yet.
- **Matcher**: The matchers are responsible for all subscription handling and publication forwarding. Matchers differ from traditional brokers in that they participate in a Voronoi Overlay Network (VON) and take ownership of any clients positioned within their own Voronoi cell.
- **Simulator**: Simulator.js is an implementation of a discrete event simulator specifically for VAST. The simulator interprets instructions from a text file, instantiating matchers and clients on the local machine. Subscription, publication, and other events may be triggered and the results captured in a special log file. These logs may then be interpreted and visualised in the "visualiser.html" tool.
- **Visualizer**: The visualizer loads the events logs generated by matchers, typically for the results of a simulator test.
- **Instruction File**: This is a text file that contains the instructions for the simulator. In this file, users are able to create nodes in the form of a gateway, multiple matchers and multiple clients. It is in this file that different case scenarios are developed to run on the simulator.

4.2 Using the Simulator

To use the simulator, these are the steps that need to be taken:

(1) Clone the repository from GitHub at [23]. The codebase is readily available. Be sure to work from the "dev" branch and ensure all the dependencies are installed.
(2) Setup the instruction file to execute your instructions. Note that there must be at least one Gateway Matcher to start the system. In this paper, we setup the configuration based on example script in the core library. The script file contains one Gateway Matcher, two "standard" Matchers and two clients publishing and subscribing to specific regions and topics at specific time intervals.
To simulate the instruction file, run this command in terminal.

```
node simulator.js example_script.txt
```

After running the simulator, the different results are stored as log files in a "logs_and_events" folder. The log files are then loaded in the visualizer to see how the nodes subscription and publication areas overlap and exchange messages.
(3) Next, run the "visualizer.html" file.

A tutorial explaining running the simulator and visualizer can be found at [22].

4.3 Visualizer Results

In this section, we will show what the results look like. We illustrate the current state of the simulator and the visualizer.

As discussed earlier, the simulator logs every event that occurs after execution and the visualizer displays the results in a virtual 2D Plane. These results are illustrated in the figures below.


```
{
  "time": 1680686823037,
  "event": 0,
  "id": 1,
  "alias": "GW",
  "aoi": {
    "center": {
      "x": 500,
      "y": 500
    },
    "radius": 100
  },
  "pos": {
    "x": 500,
    "y": 500
  }
},
```

```
{
  "time": 1680686823149,
  "event": 0,
  "id": 2,
  "alias": "M2",
  "aoi": {
    "center": {
      "x": 167,
      "y": 500
    },
    "radius": 100
  },
  "pos": {
    "x": 167,
    "y": 500
  }
}
```


Figure 6: Matcher Logs

```
[
  {
    "time": 1680686823194,
    "msg": "Client[C1]: socket connected"
  },
  {
    "time": 1680686823198,
    "msg": "Client[C1]: assigned to matcher[1]"
  }
]
```


Figure 7: Client Logs


```
[
  {
    "time": 1680686823151,
    "msg": "Matcher[3]: VON peer joined successfully"
  },
  {
    "time": 1680686823151,
    "msg": "Matcher[3]: listening on 20020"
  },
  {
    "time": 1680686823197,
    "msg": "Matcher[1]: a new client has joined GW. Send
      MOVE request"
  },
  {
    "time": 1680686823197,
    "msg": "Matcher[1]: move client[C1] to pos[300; 100]"
  }
]
```


Figure 8: Simulator Logs

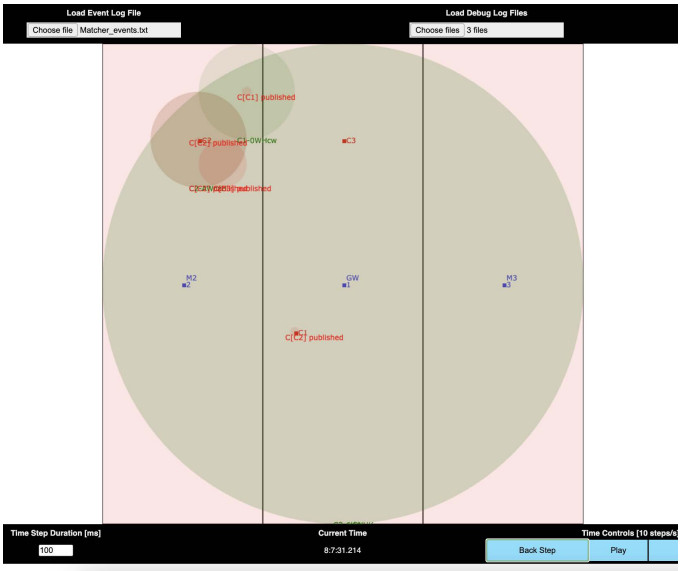

Figure 9: Visualizer showing simulated nodes with their subscription and publication areas as they overlap

Fig. 6 shows the logs captured by the matcher. The data captured includes the unique ID, the time the type of matcher (i.e. a Gateway Matcher or a Standard Matcher), the Area of Interest (AOI) [2] and its position. It also captures every pub/sub request it receives from the network nodes it manages.

The client log contains the events captured by the client(s) during simulation. The format follows the illustration in Fig. 7 It includes pub/sub requests sent and/or received, when the requests were sent and when they join or leave the network.

Fig. 8 shows the Simulator logs. This log documents the sequence of events for every instruction executed by the matcher(s) and client(s).

Fig. 9 shows the front-end display when you run the visualizer with the generated log files.

Note that these logs are captured independently and as such, there is a need for a tool that measures the correctness and performance of the VAST library. This is on a roadmap to be implemented in the immediate future. The details of this work are captured in the next section.

5 CONCLUSION AND FUTURE WORK

To summarize, we aim to demonstrate a decentralized open-source library called VAST. This library uses a concept called spatial publish/subscribe (SPS), one that leverages the spatial data of network devices to perform communication. VAST is built on a P2P architecture and has a forwarding algorithm to ensure messages are not duplicated when delivered.

This library is new and we believe it to be an innovative approach to solving for scalability in modern network systems (e.g. IoT/IIoTs and MMVEs). As such, we have developed a simulator to demonstrate this concept. We hope that this helps to drive early adoption from researchers or industries who may be looking to try new technology or scale their existing technology. We demonstrated the tool, how to use it, and the type of results it generates. The library is designed to fit any network applications that require large-scale message exchanging and have some form of spatial application.

Here are some of the things that are currently ongoing and planned for the immediate future.

(1) Develop a testing framework (e.g. an emulator) that measures the correctness and performance of the SPS library. The testing framework must also include tools to measure for measuring latency, throughput as well of the scalability of the SPS communication library:
   - Throughput. We will showcase how much of the messages actually get delivered when sent during exchanges between clients and matchers. This will be measured in %. (% of packets that get delivered and/or lost).
   - Latency. The goal here is to understand how quickly or slowly communication occurs between the clients and the matchers. We will measure this in milliseconds (ms).
   - Scalability. We will evaluate the performance over a long period of time. (e.g. 3 days). This way, we can observe if the performance quality (latency and message deliverability) appreciates, depreciates, or stays the same over a period of time. This will also be measured using latency as a reference.

(2) Enable the current library to run as independent nodes on network devices (physical and virtual). At the moment, the nodes are simulated with the localhost on a local computer.

[2] An Area of Interest (AOI) is a defined region within a virtual environment or MMVE where a users attention is focused or where a significant event is occurring [6].

The aim is to enable seamless deployment of VAST on network devices or virtual machines like Docker, making it easy to deploy for researchers.

REFERENCES

[1] Fahd A Alhaidari and Ebtesam J Alqahtani. 2020. Securing Communication between Fog Computing and IoT Using Constrained Application Protocol (CoAP): A Survey. *J. Commun.* 15, 1 (2020), 14–30.

[2] Gabriele Baldoni, Julien Loudet, Luca Cominardi, Angelo Corsaro, and Yong He. 2021. Facilitating Distributed Data-Flow Programming with Eclipse Zenoh: The ERDOS Case. In *Proceedings of the 1st Workshop on Serverless Mobile Networking for 6G Communications* (Virtual, WI, USA) *(MobileServerless'21)*. Association for Computing Machinery, New York, NY, USA, 13–18. https://doi.org/10.1145/3469263.3469858

[3] Balázs Bodó, Jaya Klara Brekke, and Jaap-Henk Hoepman. 2021. Decentralisation: A multidisciplinary perspective. *Internet Policy Review* 10, 2 (2021), 1–21.

[4] Keyan Cao, Yefan Liu, Gongjie Meng, and Qimeng Sun. 2020. An overview on edge computing research. *IEEE access* 8 (2020), 85714–85728.

[5] Fu Chen, Yujia Huo, Jianming Zhu, and Dan Fan. 2020. A review on the study on MQTT security challenge. In *2020 IEEE International Conference on Smart Cloud (SmartCloud)*. IEEE, 128–133.

[6] Ravi Dubey, Yu Li, and Jie Zhang. 2019. Area-of-Interest based optimization for real-time rendering in large-scale MMVEs. *J. Parallel and Distrib. Comput.* 127 (2019), 1–16.

[7] Eli Fidler, Hans-Arno Jacobsen, Guoli Li, and Serge Mankovski. 2005. The PADRES Distributed Publish/Subscribe System.. In *FIW*. Citeseer, 12–30.

[8] Nikos Fotiou, Dirk Trossen, and George C Polyzos. 2012. Illustrating a publish-subscribe internet architecture. *Telecommunication Systems* 51 (2012), 233–245.

[9] Eclipse Foundation. [n. d.]. Zenoh - The Zero Overhead, Pub/Sub, Store, Query, and Compute Protocol. https://zenoh.io/

[10] Fox Geoffrey. 2001. PEER-TO-PEER NETWORKS. *COMPUTING IN SCIENCE ENGINEERING* 3 (2001), 75 – 77. https://doi.org/10.1109/5992.919270

[11] Leonidas Guibas and Jorge Stolfi. 1985. Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi. *ACM Transactions on Graphics (TOG)* 4 (1985), 74–123. Issue 2. https://doi.org/10.1145/282918.282923

[12] Abhishek Gupta, Ozgur D Sahin, Divyakant Agrawal, and Amr El Abbadi. 2004. Meghdoot: content-based publish/subscribe over p2p networks. In *Middleware 2004: ACM/IFIP/USENIX International Middleware Conference, Toronto, Canada, October 18-22, 2004. Proceedings 5*. Springer, 254–273.

[13] Daniel Happ and Adam Wolisz. 2017. Limitations of the Pub/Sub pattern for cloud based IoT and their implications. *2016 Cloudification of the Internet of Things, CIoT 2016* (2017). https://doi.org/10.1109/CIOT.2016.7872916

[14] Stefan Herle, Ralf Becker, and Jörg Blankenbach. 2016. Bridging GeoMQTT and REST. In *Proceedings of the Geospatial Sensor Webs Conference*. 29–31.

[15] Stefan Herlé, Ralf Bill, and Jörg Manfred Blankenbach. 2019. *A GeoEvent-driven architecture based on GeoMQTT for the Geospatial IoT*. Ph. D. Dissertation. Geodätisches Institut, Rheinisch-Westfälische Technische Hochschule.

[16] Shun-Yun Hu. 2005. Scalable Peer-to-Peer Networked Virtual Environment.

[17] Shun-yun Hu. 2009. Spatial Publish Subscribe. *IEEE Virtual Reality (IEEE VR)* (2009), 6. http://pap.vs.uni-due.de/MMVE09/papers/p8.pdf

[18] Shun-Yun Hu, Chen, and Tsu-Han Chen. 2006. VON: A Scalable Peer-to-Peer Network for Virtual Environments. *IEEE Network Magazine* (2006).

[19] Shun-Yun Hu and Kuan-Ta Chen. 2011. VSO: Self-Organizing Spatial Publish Subscribe. In *2011 IEEE Fifth International Conference on Self-Adaptive and Self-Organizing Systems*. 21–30. https://doi.org/10.1109/SASO.2011.13

[20] Shun Yun Hu and Guan Ming Liao. 2004. Scalable peer-to-peer networked virtual environment. *Proceedings of the ACM SIGCOMM Workshop on Network and System Support for Games, NetGames'04* (2004), 129–133. https://doi.org/10.1145/1016540.1016552

[21] Mats Ake Hugoson. 2008. Centralized versus Decentralized Information Systems: A Historical Flashback. *IFIP Advances in Information and Communication Technology* 303 (2008), 106–115. https://doi.org/10.1007/978-3-642-03757-3_11

[22] Imonology Inc. [n. d.]. How to use VAST.js Simulator and Visualizer - YouTube. https://www.youtube.com/watch?v=2_X-TwGXcRs

[23] Imonology Inc. [n. d.]. VAST.js. https://github.com/imonology/VAST.js/tree/dev_CFM

[24] Yoshio Inoue. 2020. Satellite- and drone-based remote sensing of crops and soils for smart farming – a review. *Soil Science and Plant Nutrition* 66, 6 (2020), 798–810. https://doi.org/10.1080/00380768.2020.1738899

[25] Laura Itzel, Florian Heger, Gregor Schiele, and Christian Becker. 2011. The quest for meaningful mobility in massively multi-user virtual environments. In *2011 10th Annual Workshop on Network and Systems Support for Games*. IEEE, 1–2.

[26] Hans-Arno Jacobsen. 2009. *Publish/Subscribe*. Springer US, Boston, MA, 2208–2211. https://doi.org/10.1007/978-0-387-39940-9_1181

[27] Jehn Ruey Jiang, Yu Li Huang, and Shun Yun Hu. 2009. Scalable AOI-cast for peer-to-peer networked virtual environments. *Journal of Internet Technology* 10 (2009), 119–125. Issue 2.

[28] Reza Sherafat Kazemzadeh and Hans-Arno Jacobsen. 2009. Reliable and highly available distributed publish/subscribe service. In *2009 28th IEEE International Symposium on Reliable Distributed Systems*. IEEE, 41–50.

[29] Dmitrij Lagutin, Kari Visala, and Sasu Tarkoma. 2010. Publish/subscribe for internet: PSIRP perspective. *Towards the Future Internet: Emerging Trends from European Research* 4 (2010), 75–84. https://doi.org/10.3233/978-1-60750-539-6-75

[30] Robin Jan Maly, Jan Mischke, Pascal Kurtansky, and Burkhard Stiller. 2003. Comparison of centralized (client-server) and decentralized (peer-to-peer) networking. *Semester thesis, ETH Zurich, Zurich, Switzerland* (2003), 1–12.

[31] Charl Marais. 2021. Extending VAST to Support Distributed Spatial Publish and Subscribe. www.eng.sun.ac.za

[32] Mario Marchese, Aya Moheddine, and Fabio Patrone. 2019. IoT and UAV Integration in 5G Hybrid Terrestrial-Satellite Networks. *Sensors* 19, 17 (2019). https://doi.org/10.3390/s19173704

[33] MQTT.org. [n. d.]. MQTT - The Standard for IoT Messaging. https://mqtt.org/

[34] Michael Nast, Hannes Raddatz, Benjamin Rother, Frank Golatowski, and Dirk Timmermann. 2022. A Survey and Comparison of Publish/Subscribe Protocols for the Industrial Internet of Things (IIoT). (2022). https://doi.org/10.1145/3567445.3571107

[35] Paessler. [n. d.]. What is MQTT? Definition and Details. https://www.paessler.com/it-explained/mqtt

[36] Matthew Riordan. 2020. Publish-Subscribe: Introduction to Scalable Messaging – The New Stack. https://thenewstack.io/publish-subscribe-introduction-to-scalable-messaging/

[37] Karen Rose, Scott Eldridge, and Lyman Chapin. [n. d.]. The Internet of Things: An Overview Understanding the Issues and Challenges of a More Connected World. ([n. d.]).

[38] Miguel Smith. 2020. VAST : a scalable spatial publish and subscribe system integrated with Minecraft by. Issue March. http://hdl.handle.net/10019.1/108007

[39] Dipa Soni and Ashwin Makwana. 2017. A survey on mqtt: a protocol of internet of things(IoT). *International Conference on Telecommunication, Power Analysis and Computing Techniques (Ictpact - 2017)* (2017), 0–5. Issue April.

[40] Darius Suchojad. 2019. Publish/subscribe, Zato services and asynchronous API integrations. https://zato.io/blog/posts/pubsub-service.html

[41] Richard Sueselbeck, Gregor Schiele, Sebastian Seitz, and Christian Becker. 2009. Adaptive update propagation for low-latency massively multi-user virtual environments. In *2009 Proceedings of 18th International Conference on Computer Communications and Networks*. IEEE, 1–6.

[42] Steve Vinoski. 2006. Advanced message queuing protocol. *IEEE Internet Computing* 10, 6 (2006), 87–89.