International Journal of Emerging Engineering and Technology (IJEET)
ISSN (e): 2958-3764
Volume: 2, Number: 1, Pages: 99- 109, Year: 2023

Innovation Beyond Servers: A Bridge Protocol Framework for Mobile Streaming

Arslan Nasir[1], Mirza Adnan Baig[1], M. Saqib Rehan[1], Abdullah Haider[1], Muhammad Rizwan[2]

[1] The Islamia University Bahawalpur, Pakistan
[2] Khawaja Freed University of Engineering & Information Technology, Pakistan
Corresponding author: email: m.arslan146p@gmail.com
Received: 15/01/2023, Revised: 25/03/2023, Accepted: 20/06/2023

ABSTRACT:In the ever-expanding realm of live video streaming applications, the demand for real-time content delivery has reached unprecedented heights. Yet, challenges persist, tethered to the uncertainties of internet availability and the constraints of server-based resource optimization. As the appetite for efficient and scalable live video streaming continues to surge, the call for inventive solutions that break free from conventional server-centric systems grows stronger. Enter the realm of serverless streaming—a groundbreaking opportunity to surmount these challenges. At the core of this research lies a mission to confront the shackles imposed by internet dependency and the limitations of server-based frameworks in the context of live video streaming. Our primary aim is to usher in a new era of content sharing in a serverless environment, unlocking the potential to extend mobile device connectivity and transcend bandwidth constraints. The proposed solution introduces an ingenious protocol spanning from the Data Link Layer (DLL) to the Application Layer (AL). Through the establishment of two virtual networks and the application of the Bridge Protocol (BP), a bridge emerges, seamlessly connecting them. This framework empowers the sharing of a mobile device's display to other devices via a hotspot, liberating the process from the confines of an internet connection. Preliminary results affirm the efficiency of our framework in transferring the Application Layer (AL) display from one mobile device to others. This achievement is made possible by binding two virtual networks to the same NIC using our bridge protocol. The spotlight is on screen sharing through image streaming between devices, all achieved independent of an internet connection. This research represents a pioneering approach to mobile streaming, placing a distinct focus on peer-to-peer (P2P) technology. It proposes an unprecedented fusion of live video streaming dynamics with the principles of P2P video streaming, culminating in a method that optimizes device interconnection, unshackled from the constraints of traditional internet-based systems.
Keywords: Bridge Protocol, Distributed Mobile Streaming, Serverless Environment.

INTRODUCTION:

In the age of digitization, our lives are seamlessly interwoven with the presence of mobile devices, shaping the very fabric of our daily existence. These devices, in their swift evolution, have not only revolutionized communication but have also redefined the way we engage with content. Among the myriad functionalities that have captured our collective fascination, live streaming stands out as a meteoric phenomenon, solidifying its status as a highly sought-after feature on mobile platforms. As technology propels forward, we find ourselves at a crossroads, witnessing a gradual departure from traditional systems, particularly those rooted in server-driven architectures. This shift is propelled by an unceasing demand for decentralization, the quest for reduced latency, and the pursuit of more efficient resource utilization.

The global digital communication landscape is undergoing a profound transformation[1]. What was once considered a luxury, video streaming has now become an integral part of our digital experience. Countless applications facilitate real-time video broadcasts to millions worldwide. Yet, like any technology in its zenith, live streaming encounters its unique set of challenges[2].

At the core of our digitally connected society lies the essential thread of internet connectivity. But what happens when this thread is disrupted or unavailable? Can live video streaming still be a viable possibility? More crucially, can it maintain efficiency and robustness in the face of such challenges? Our research embarks on a journey into this niche yet vital realm. The pressing issues of internet availability constraints and server optimization demand urgent solutions. In a world dominated by mobile devices, the vision of a server-less environment for live video sharing presents an alluring and tantalizing alternative[3].

Study aim:
At the heart of our research lies a visionary mission: to forge a cutting-edge framework or protocol that liberates users from the shackles of internet dependence. Our aim is to empower individuals to seamlessly share and distribute content in a realm where connectivity barriers dissolve, harnessing the power of a server-less environment facilitated by mobile hotspots. In this transformative landscape, our goal extends beyond mere distribution; we envision a paradigm shift in content dissemination through the art of multicasting, where each node becomes a vibrant source of shared experiences and information. This research endeavors to pioneer a future where connectivity is not a limitation but a gateway to a boundless exchange of knowledge and experiences.

Problem Statement:
At the crux of our present research lies a formidable challenge: the absence of a viable solution for real-time video sharing within a server-less environment, utilizing the dynamic capabilities of mobile hotspots and implementing multicasting of identical content from every node. While existing solutions diligently tackle the optimization of network and server resources, they fall short in addressing the critical challenges posed by sharing content in the absence of an internet connection, as well as the untapped potential for multicasting within these unique environments. Our research boldly ventures into uncharted territory, aiming not only to bridge this gap but to redefine the landscape of live video sharing in server-less ecosystems.

Research Objectives:
At the core of our research endeavors is a visionary goal: to craft a pioneering framework enabling mobile screen sharing without the constraints of internet connectivity within a server-less environment. This ambitious objective unfolds through a meticulously designed approach, where innovation converges with practicality. Our strategy encompasses the following key elements:

1. Creation of a Virtual Network: We embark on the journey by establishing a virtual network for mobile devices, ingeniously leveraging the power of hotspots. This foundational step lays the groundwork for connectivity that transcends traditional boundaries.

2. Bridge Protocol (BP) Implementation: To orchestrate seamless communication within these virtual networks on a single Network Interface Card (NIC), we introduce the Bridge Protocol (BP). This intelligent protocol becomes the linchpin, fostering efficient and effective data exchange.

3.Utilization of TCP and UDP: In the intricate dance of data packet transmission between connected devices, we employ the robust Transmission Control Protocol (TCP) and the nimble User Datagram Protocol (UDP). This dynamic duo ensures a reliable and swift exchange of information, adapting to the unique needs of our server-less environment.

4.Expanding Device Limits and Bandwidth: We push the boundaries of connected devices and bandwidth by ingeniously recreating virtual networks on each mobile device. This ingenious approach aims to enhance the scalability and resilience of our mobile screen sharing ecosystem.

5.Shift towards Broadcasting: Finally, we revolutionize the sharing mechanism itself by embracing broadcasting. This strategic shift ensures seamless content distribution among connected devices, fostering a harmonious and efficient sharing experience.

In essence, our research aspires not only to break the barriers of conventional mobile screen sharing but to redefine the very landscape of how we envision and implement connectivity in a server-less world. Through these innovative strategies, we chart a course toward a future where mobile screen sharing becomes an effortlessly accessible and ubiquitous experience.

Proposed Methodology:
This research venture is centered around the conception and implementation of an innovative protocol, poised for experimentation on Android mobile devices to revolutionize communication. Illustrated in Figure 1.1 is the intricate functionality of the envisioned Bridge Protocol, illustrating the creation of two distinct virtual networks on a single Network Interface Card (NIC) within a mobile device. Each of these networks boasts its own identical configurations, yet direct communication between them remains elusive. The crux of this challenge is the impetus behind the creation of the Bridge Protocol (BP), serving as the linchpin for inter-network communication and facilitating the seamless transfer of video frames from one network's Application Layer (AL) to another.

The experimental phase unfolds on the Android platform, where an ad hoc network is forged using the mobile hotspot. All recipient mobile devices converge within this network, with each recipient joining two virtual networks. One network links to the sender node, while the other is autonomously created within the recipient device. The Bridge Protocol commences its work, bridging these networks within the device and fostering communication between them. All usable ports of one network ingeniously bridge to the other, ensuring a cohesive network architecture.

The Bridge Protocol is meticulously crafted using both TCP and UDP, with TCP facilitating the scanning of the receiving mobile device's network. Acknowledgment is dispatched upon packet reception, prompting a search via ping on each IP port to identify live IPs within range. The live IP, indicative of the sender mobile device, awaits connections on an open port, and Figure 1 exemplifies the seamless bridging of each mobile network to its successive device while concurrently searching

its connected network for live IPs. Upon acknowledgment, the identified IP is cataloged in the client-side list. Subsequently, a handshake request is initiated, and if accepted by the sender, the device becomes an integral part of that network, initiating seamless communication. In the receiving node, dual networks exist—one in which it is connected and the other autonomously created. The Bridge Protocol adeptly unites these networks, fostering a cohesive and integrated communication environment. Moreover, experimental results manifest through an exploration of video stream quality, contingent upon the number of connected mobile devices. Initial experiments involve three mobile devices sharing a single network for communication, with subsequent extensions to record video quality, delay time, and mobile performance as the number of connected devices proliferates. This comprehensive approach ensures a thorough understanding of the protocol's impact on the robustness and quality of mobile communication.

LITERATURE REVIEW:

Serverless edge computing represents an evolving paradigm that extends the advantages of serverless computing to the dynamic edge-cloud continuum. This paper introduces faas-sim, an innovative simulation framework tailored to address the unique challenges of serverless edge computing platforms. In serverless computing, the abstraction of infrastructure from application developers places the responsibility of efficient management on platform operators. However, the transition to edge computing amplifies this challenge due to the absence of reference architectures, design tools, and standardized benchmarks[4].

Faas-sim bridges this gap by offering a threefold solution: (a) a generalized model of serverless systems built on the function-as-a-service abstraction, (b) a trace-driven stochastic discrete-event simulation engine using real-world edge computing testbed data, and (c) a network topology generator to simulate distributed and heterogeneous edge-cloud systems. The contributions of faas-sim extend beyond simulation, encompassing a performance and resource modeling approach based on real profiling experiments. The framework includes a flexible codebase in Python, integrating seamlessly with data science tools and forming a key component of the broader Edgerun project—an ecosystem supporting researchers and practitioners in edge-cloud experiments.

Faas-sim's impact is multifaceted, providing a robust foundation for researchers and platform designers to develop, implement, and evaluate novel serverless edge platforms. The framework allows for the exploration of function adaptations, scheduling algorithms, and load-balancing strategies. Notably, faas-sim has been employed successfully in diverse scenarios, from scheduling performance comparisons to large-scale distributed edge computing simulations. As part of ongoing work, the integration of faas-sim into the control loop of operating serverless systems is explored, offering the potential for real-time refinement of simulation models based on trace data from actual workload executions[5].

faas-sim emerges as a pivotal tool in the realm of serverless edge computing, offering a versatile simulation environment backed by real-world data. Its utility extends to both practical experimentation and theoretical exploration, contributing significantly to the advancement of this emerging and dynamic field[6].

In the swiftly evolving landscape of Information and Communication Technologies (ICT), the integration of modern technologies is reshaping industries, fostering dynamic value chains, and facilitating collaboration among various stakeholders. A compelling example is the emergence of Smart Tourism, where ICT is leveraged to craft immersive and sustainable tourism experiences. However, the decentralized nature of these environments presents a persistent challenge—harmonizing diverse services and data across disparate administrative domains. In response, we introduce APERTO, a decentralized and distributed architecture underpinned by APERTO FaaS, a Serverless platform. This platform streamlines business logic prototyping, reduces entry barriers, and minimizes development costs. APERTO addresses the intricacies of distributed and heterogeneous environments by employing asynchronous and transparent communications between components, thus optimizing solutions for seamless data and service integration.

At its core, APERTO tackles critical issues, providing scalable mechanisms for function composition, end-to-end Quality of Service (QoS) slices, uniform access to diverse data sources, and a decentralized approach for resource access verification. APERTO FaaS, with its innovative features like (zero) fine-grained scaling and reduced management overhead, signifies a significant advancement in creating accessible and cost-efficient solutions in the realm of ICT-driven collaboration and integration. This architecture stands as a transformative step toward a more integrated and streamlined future for industries embracing the potential of Information and Communication Technologies[7].

Introduces Portals, a novel serverless, distributed programming model that seamlessly integrates the exact-once processing guarantees of stateful dataflow streaming frameworks with the compositional, message-driven nature of actor frameworks. The core elements of computation in Portals, termed atomic streams, enable the construction of decentralized applications dynamically, ensuring scalability on demand while adhering to strict atomic processing guarantees. The model is versatile, supporting various distributed programming paradigms and use cases. The paper outlines Portals' capabilities, presenting programming model invariants and the associated system methods that uphold them[8]. Key contributions include transactional processing guarantees, intuitive data-parallel

service composition, and use cases ranging from Sagas to GDPR-support, showcasing the model's generality and effectiveness. A prototype implementation is discussed, providing insights into the design and performance evaluation.

The architecture of Portals comprises the portals programming platform and the portals runtime, featuring components such as the scheduler, registry, atomic streams, workflows, and portals. The implementation leverages distributed transactional logs like Apache Kafka and Pravega. The model is presented as a high-level physical plan, demonstrating its end-to-end processing guarantees and scalable data-parallel execution. Portals represents a significant advancement, combining diverse features into a unified programming model, making it a promising alternative for developing scalable stateful serverless applications with guaranteed atomic processing. Future work includes completing the distributed implementation, refining operational semantics, providing a sound type system, and exploring extensions such as dynamic atom splitting and fusing, along with actor-like references[9].

This work introduces Crucial, a system designed to address the challenges of building highly-parallel stateful serverless applications, particularly those requiring fine-grained support for mutable state and synchronization, such as machine learning (ML) and scientific computing. Crucial seamlessly integrates the simplicity of serverless computing with the scalability of Function-as-a-Service (FaaS) platforms. Leveraging a distributed shared memory layer, Crucial enables the effortless porting of multi-threaded code bases to serverless environments, unlocking the scalability and cost-effectiveness of FaaS platforms[10]. The system is validated through micro-benchmarks and the implementation of various stateful applications, including ML algorithms like k-means and logistic regression. Evaluation results show Crucial outperforming or performing comparably to Apache Spark at a similar cost, with applications achieving up to 30% higher speed than dedicated high-end servers. Crucial's open-source implementation, consisting of around 10K SLOC, is available online, showcasing its versatility and practicality in serverless programming[11].

The architecture of Crucial, discussed in detail, incorporates a dynamic shared object (DSO) layer, written a top the Infinispan in-memory data grid. A Crucial application, written in Java, utilizes Apache Maven for compilation and dependency management. The system employs abstractions for distributed parallelism, including an innovative Serverless Executor Service and Iterative Task[12]. AWS Lambda functions facilitate the execution of cloud threads. The system demonstrates efficient handling of distributed references, synchronization objects, and dynamic parallelism. The evaluation further explores the runtime of Crucial through micro-benchmarks, fine-grained updates to mutable data, and porting of state-of-the-art ML libraries to serverless. The study aims to assess Crucial's ease of programming, benefits in terms of serverless capabilities, efficiency, and cost-effectiveness for both serverless-native and ported applications[13].

Explores the cost implications of choosing between distributed stream processing (DSP) and Function-as-a-Service (FaaS) for cloud event processing applications. Despite architectural differences, both DSP and FaaS can model loosely-coupled job graphs, making them applicable for real-time event processing. The study implements stateless and stateful workflows using the Theodolite benchmarking suite on cloud FaaS and DSP platforms, considering factors such as application type, cloud service provider, and runtime environment[14]. The evaluation provides decision guidelines for cloud engineers based on a comprehensive analysis of the cost-effectiveness of different deployment scenarios.

The benchmarking suite involves the implementation of use cases for DSP and FaaS, emphasizing adherence to best practices and allowing for broader evaluations beyond DSP vs. FaaS. Load generation is achieved through a dedicated load generator within the same cloud datacenter, emulating sensors with varying loads. The benchmark aims to understand the cost implications of operating applications under different data rates rather than focusing on scalability or elasticity[15].

The extensive evaluation section presents a baseline comparison between Google Cloud Functions and Apache Flink, detailing implementations for stateless and stateful use cases. The results demonstrate that FaaS is economically advantageous for stateless applications with low to medium event arrival rates, while the cost of database access in FaaS becomes a limiting factor for stateful applications with higher event processing rates. The study provides insights into the cost breakdown, showcasing the interplay between fixed costs, variable costs per request, and batched variable costs in DSP and FaaS deployments[16].

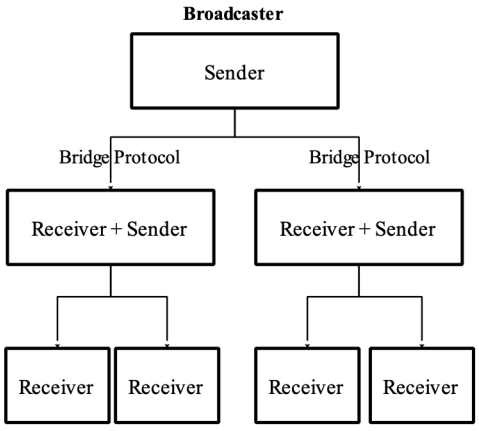

Figure 1.0 *Contextual Diagram.*

This research uses a framework and UDP socket-based protocol to transfer screen capture from one phone to another. An

Android app with sender and receiver modes is used for this experiment. BP is a framework that uses existing mobile technology as a base layer to communicate and broadcast data from sender to receiver. It also bridges two independent networks on the same NIC for further distribution, as shown in figure 3.0.

Figure 2.0 depicts a server-less, internet-free working paradigm. As shown in figure 2.0, one sender opens its hotspot on which other devices connect to WIFI and open their own hotspots with different networks. Bridging combines two data link layer networks by sending requests to the destination address and responding to data based on MAC addresses as shown in figure 4.0. As shown in figure 3.1, suggested BP acts as middleware, broadcasting data unidirectionally via a client socket. This is extended to the nth node; however, the number of devices increases the delay.

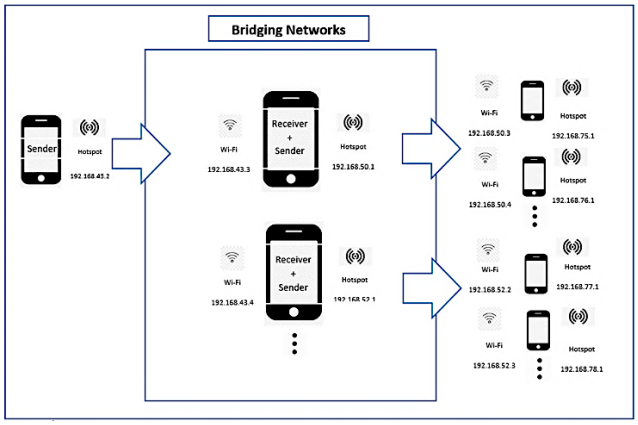

Figure 2.0 *Bridge-Network.*

The complete framework also includes android-specific communication norms and restrictions. Frameworks capture frames based on screen resolution in the application layer and compress them using bitmap compression on the presentation layer. The session layer handles the user connection as a socket connection and manages a user list. Data is then sent to the client using FIFO. The network layer controls connected device data flow and failover connection management, whereas the data link layer handles bridging network and data forwarding regulations. Since we don't have bandwidth congestion, the data link layer buffer isn't used to deliver pure real-time streaming. All additional default network rules and approaches.

Figure 2.0 illustrates BP's whole operation. Sender phone initiates hotspot on dynamic network where all receiver phones are connected. Receiver phones open their hotspots to distribute material further, creating two virtual networks on the same NIC for receiving and broadcasting. Data is sent from Wi-Fi to virtual network (hotspot) using BP.

The application server starts its UDP port on 1002 for streaming, waits for clients to connect, adds client sockets to the list, and creates for connected clients. The receiver automatically generates a UDP client request on the given IP address for port 1002 when they click on any listed device. If the success handshake message is received, the client waits for data, otherwise they try to reconnect with to be a sender, the client launches a mobile hotspot on the same port as a server with a separate network and open handshake port (1001).

Server-side screen images are compressed and transformed into byte arrays for socket transmission. The list of all clients receives the frame size after converting it to a byte array. Any disconnected or failed client socket will be deleted from the list. In case of sender disconnections or failures, the client will allocate the frame size for packet reception and retry to connect. The server sends the byte array to all client sockets. All disconnected clients will be removed by the sender. The server repeats till session end. Frames are turned into images at the client end to display and transmit to other clients and vice versa. Figure 3.0 shows the entire communication control-flow, while figure 4.0 shows model workflow.

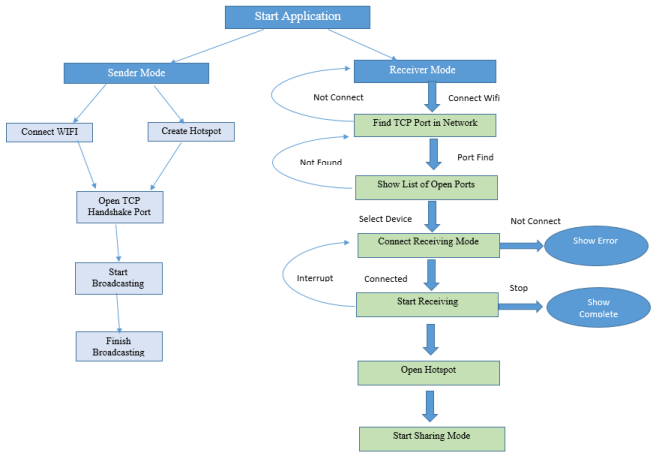

Figure 3.0 *Control flow of communication.*

Figure 5.0 shows the basic functionality of existing bridging technique, where bridge device notes mac addresses of all connected devices, station B requests data, bridge device notes device mac address at physical layer, and bridge device forwards packet to station A. It returns to mac address device after obtaining response. This technique uses request-response. BP creates the client list based on handshake requests and broadcasts data without client requests or mac addresses.

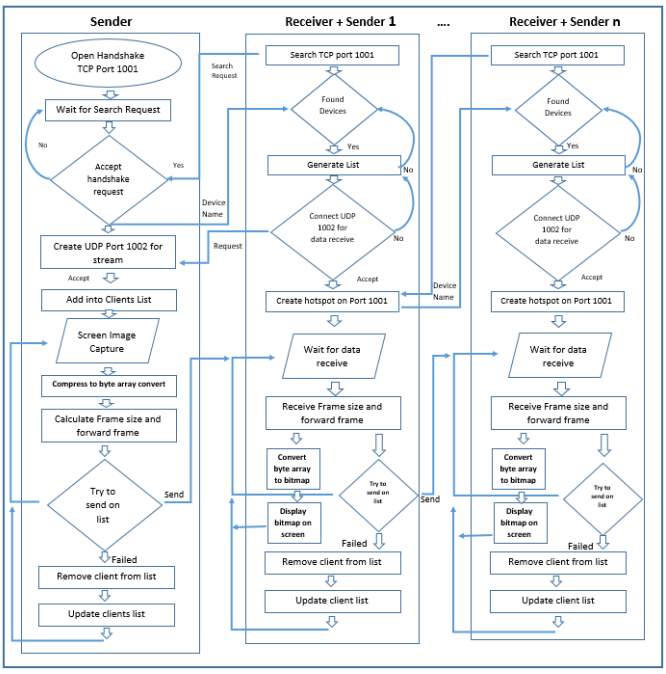

Figure 4.0 *Bridge protocol workflow diagram.*

A bridge connecting two LAN segments

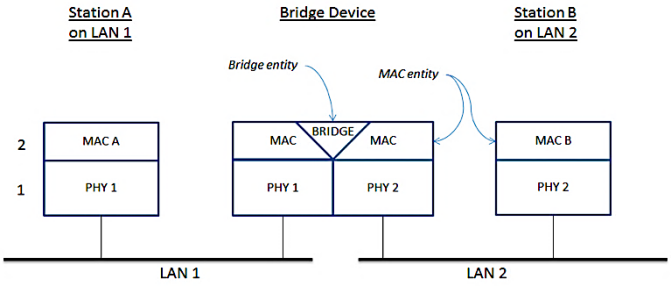

Figure 5.0 *Networks Bridge Methods.*

Table 1.0 *Benchmark experiment parameters.*

Experiment Parameters	Values	Units
No. of max devices connect on single hotspot	10	-
Max Number of frames	25	fps
Sender node buffer	none	-
Receiver node buffer	none	-
Data type	Binary	Bytes
Packet receive Acknowledgment	none	-
Frame size	~30-50	KB
Max number of extended node	nth	-
Shortest path decision	N/A	-
Mobility handling	none	-
Bridging Protocol	PTMP	-
Wireless Fidelity Standard	802.11b/g/n	-
Network Protocol	TCP/UDP	-
Network DHCP	built-in	-
Network Frequencies	2.4-5	GHz
Wireless Security	none	-
Network Bandwidth	54-150	Mbps

The default experimental settings are presented in table 1.0, while additional parameters utilized in the proposed framework may be adjusted in existing approaches or redefined for this experiment.

Dataset

A dataset is based on the number of frames sent between devices and the delay between them. This dataset also shows delay while sending data from a second phone to a third and when the number of devices increases. Android logcat collects data when phone is linked to Android Studio and all logs are monitored on the console. When device sends packet, console shows time, and receiving frame shows time successfully. Time deference is device-to-device delay.

EXPERIMENTAL RESULTS & DISCUSSION:

Results from android phone experiments are presented in tables, graphs, and detailed explanations. Based on results, the description examines the variables to use the advised BP.

Receiver-Multiple-Node Sharing

When sharing a mobile screen across many phones, data packets are delayed by the application. Which depends on packet size and node count.

To calculate the overall delay between sharing and receiving mobiles, Data packets of different frame sizes are captured and delivered to quantify the time in capturing, converting, transmitting, receiving, rendering, and displaying. Table 2.0 demonstrates delays for up to 3 linked devices with 50KB frames. Figure 6.0 depicts the second delay between three 50Kb-frame phones on the same network.

Figure 6.0 and table 2.0 demonstrate that the first, second, and third phones delay 0.13, 0.17, and 0.24 seconds when sending a 50Kb frame. The results showed that sending and receiving mobiles stream less video with a time delay factor. Second receiver delay increased by 0.04 seconds, and third receiver delay increased by 0.07 seconds.

The scheduled task between client lists and the FIFO mechanism for broadcasting caused this delay. The second phone waited until the frames were completely sent to the first phone, therefore the latency increased as the number of devices increased. Sending 5 50Kb frames delays the first receiver 0.16,

the second and third 0.19 and 0.29. These estimated results showed that the delay increased by 0.03 seconds from the first receiver and 0.10 seconds when the third receiver received frames from the second receiver. Sending 10 50Kb frames delays the first, second, and third by 0.18, 0.25, and 0.34 seconds. Thus, second and third have 0.08 and 0.09 delays, respectively. Although for 15 50Kb frames, the first, second, and third delay 0.23, 0.32, and 0.38 seconds. First and second are 0.09 apart, and second and third are 0.06 apart. When sending 20 50Kb frames, the first delay is 0.26 seconds, the second 0.37, and the third 0.43. Between first and second, the delay is 0.11; between second and third, 0.16. For 25 50Kb frames, the first delay is 0.28 seconds, while the second and third are 0.41 and 0.43. With 30 50Kb frames, the first delay is 0.32 seconds, the second 0.49, and the third 0.53. As per Table 4. One delay difference between the 1st and 2nd phones is that the 1st phone had to process some preliminary tasks as a sender node, such as frame capturing, rendering, and transferring, which took longer to send a frame than the 2nd or 3rd phones. The delay is only for data transfer.

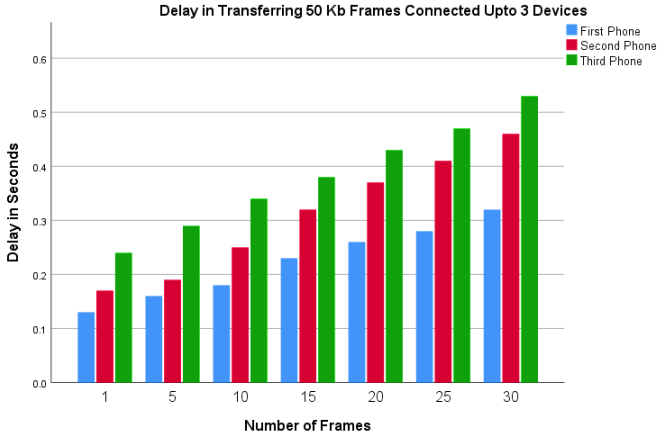

Figure 6.0 *50KB frame sending graph*

Table 2.0 *Dataset of frame 50KB*

One phone to other			
Frame 50 KB	Delay in sec for 1 Phone	Delay in sec for 2 Phone	Delay in sec for 3 Phone
1	0.13	0.17	0.24
5	0.16	0.19	0.29
10	0.18	0.25	0.34
15	0.23	0.32	0.38
20	0.26	0.37	0.43
25	0.28	0.41	0.47
30	0.32	0.49	0.53

Figure 7.0 depicts the second delay between three 30Kb-frame phones on the same network.

Figure 7.0 and table 3.0 demonstrate that the first, second, and third phones delay 0.07, 0.11, and 0.14 seconds while sending a 30Kb frame. The results showed that sending and receiving mobiles stream less video with a time delay factor. Second receiver delay increased by 0.04 seconds, and third receiver delay increased by 0.03 seconds. By delivering 5 30Kb frames, initial receiver delays are 0.09, second and third 0.13 and 0.19. These estimated results showed that the delay increased by 0.04 seconds from the first receiver and 0.06 seconds when the third receiver received frames from the second receiver. Sending 10 30Kb frames delays the first, second, and third by 0.12, 0.17, and 0.22 seconds. First and second are 0.05 apart, and second and third are 0.05. For 15 30Kb frames, the first, second, and third phones had delays of 0.14, 0.21, and 0.27 seconds. A 0.07 delay separates the first and second, while 0.06 separates the second and third. When sending 20 30Kb frames, the first delay is 0.17 seconds, the second 0.24, and the third 0.30. Between second and third, the delay increased 0.06 from 0.07. For 25 30Kb frames, the first delay is 0.20 seconds, while the second and third are 0.26 and 0.33. With 30 30Kb frames, the first delay is 0.24 seconds, the second 0.29, and the third 0.36. Table-3.0 displays delay behavior for 3 linked devices with 30KB frames.

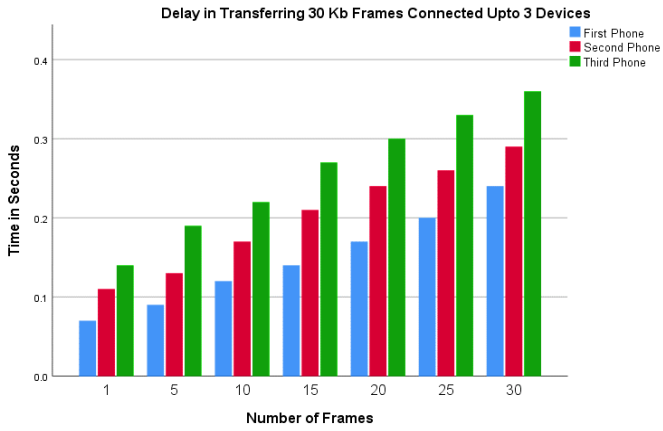

Figure 7.0 *Sending graph for 30KB frame.*

Table 3.0 *Dataset of 30KB frame.*

One phone to other			
Frame 30KB	Delay in sec for 1 Phone	Delay in sec for 2 Phone	Delay in sec for 3 Phone
1	0.07	0.11	0.14
5	0.09	0.13	0.19
10	0.12	0.17	0.22
15	0.14	0.21	0.27
20	0.17	0.24	0.30
25	0.20	0.26	0.33
30	0.24	0.29	0.36

Figure 8.0 depicts the second delay between three 20Kb-frame phones on the same network. Figure 8.0 demonstrate that the first, second, and third phones send 20Kb frames with 0.04, 0.08-, and 0.15-seconds delays. The results showed that sending and receiving mobiles stream less video with a time

delay factor. Second receiver delay increased by 0.04 seconds, and third receiver delay increased by 0.08 seconds. First receiver delay is 0.06 for 5 frames of 20Kb, second and third receiver delays are 0.10 and 0.18.

These estimated results showed that the delay increased by 0.04 seconds from the first receiver and 0.08 seconds when the third receiver received frames from the second receiver. Sending 10 20Kb frames delays the first, second, and third by 0.09, 0.13, and 0.20 seconds. Thus, second and third phones have 0.05 and 0.07 delays between them. For 15 20Kb frames, the first, second, and third have 0.11, 0.16-, and 0.23-seconds delays. The first and second are 0.05 apart, and the second and third are 0.07 apart. When 20 frames of 20Kb are sent, the first delay is 0.14 seconds, the second 0.18, and the third 0.27. The delay between first and second is 0.04 and between second and third is 0.09. For 25 frames of 20Kb, the first delay is 0.16 seconds, the second and third 0.20 and 0.29. With 30 20Kb frames, the first delay is 0.19 seconds, the second 0.25, and the third 0.32. For 3 linked devices with 20KB frames, Table-4.3 displays delay behavior.

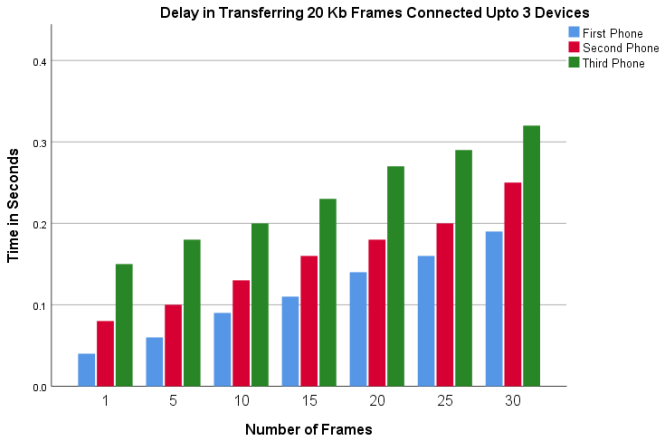

Figure 8.0 *20KB sending frame.*

Figure 9.0 depicts the second delay between three 10Kb-frame phones on the same network. Figure 9.0 and table 4.0 indicate that the first, second, and third phones delay 0.02, 0.05, and 0.10 seconds while sending a 10Kb frame. When sending and receiving videos from mobiles, the time delay factor reduces video streaming. Second receiver delay increased by 0.03 seconds, and third receiver delay increased by 0.05 seconds.

First receiver has 0.04 delay, second 0.08, and third 0.13 for 5 frames of 10Kb. This estimate reveals that the second receiver is 0.04 seconds slower than the first. Third receiver delays are 0.05 seconds. First, second, and third delays in seconds are 0.07, 0.10, and 0.16 for 10Kb frames. Thus, delay between first and second is 0.03 while second and third is 0.06. The first delay in seconds is 0.09, the second and third are 0.13 and 0.19 with 15 frames of 10Kb. The delay between first and second is 0.04 and second and third is 0.06. In 20 frames of 10Kb, the first delay is 0.12 seconds, the second 0.15, and the third 0.22. The delay between first and second is 0.03 and second and third 0.07. With 25 10Kb frames, the first delay is 0.14 seconds, the second 0.18, and the third 0.24. For 30 frames of 10Kb, first, second, and third delays are 0.17, 0.21, and 0.27 seconds. Table 4.4 exhibits delay behavior for 3 10KB-frame devices.

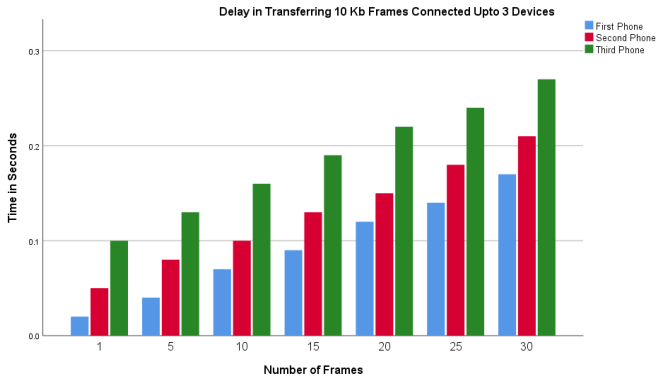

Figure 9.0 *Sending graph of 10KB.*

Table 4.0 *Dataset frame for 10kb.*

One phone to other			
Frame 10 KB	Delay in sec for 1 Phone	Delay in sec for 2 Phone	Delay in sec for 3 Phone
1	0.02	0.05	0.1
5	0.04	0.08	0.13
10	0.07	0.10	0.16
15	0.09	0.13	0.19
20	0.12	0.15	0.22
25	0.14	0.18	0.24
30	0.17	0.21	0.27

In seconds, Figure 10.0 depicts the delay between receiver phones and the following three phones on the same network with 50Kb frame size. From figure 10.0 and table 5.0, sending a 50Kb frame at once delays the receiver phone by 0.21 seconds to the first phone, 0.29 seconds to the second, and 0.35 seconds to the third.

The video streaming between sending and receiving mobiles with time delay factor is less from sender to first receiver, but it increases by 0.08 seconds to second receiver and 0.06 seconds to third receiver. Sending 5 frames of 50Kb delays the first receiver 0.27, the second 0.33, and the third 0.41. The second receiver's estimated delay is 0.10 seconds longer than the first. Delay increases by 0.08 seconds when the third receiver receives the frame. It concludes that sending frames to the third recipient increases delay. After sending 10 50Kb frames, the first delay is 0.29, the second 0.35, and the third 0.47. First and second phones are 0.06 apart, and second and third are 0.12. For 15 50Kb frames, the first delay is 0.32 seconds, the second and third 0.39 and 0.51. The delay ratio between first and second is 0.07 while second and third is 0.12.

With 20 50Kb frames, the first phone delay is 0.36 seconds, the second 0.44, and the third 0.56.

The delay ratio between first and second is 0.08 while second and third is 0.10. First delay in seconds is 0.38, second 0.51, and third 0.62 when sending 25 50Kb frames. First delay in seconds is 0.42, second phone delay is 0.55, and third delay is 0.67 for 30 frames of 50Kb. Table 5.0 illustrates the latency from receiving device to following connected devices up to 3 with 50KB frame size.

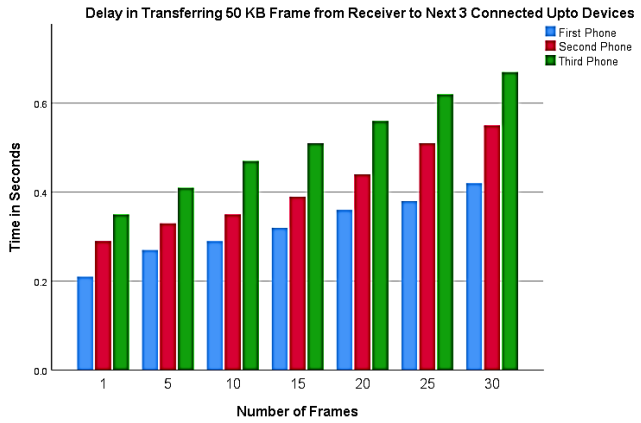

Figure 10.0 *50KB frame sending Graph.*

Table 5.0 *Dataset for frame 50KB.*

One phone to other			
Frame 50KB	Delay in sec for 1 Phone	Delay in sec for 2 Phone	Delay in sec for 3 Phone
1	0.21	0.29	0.35
5	0.27	0.33	0.41
10	0.29	0.35	0.47
15	0.32	0.39	0.51
20	0.36	0.44	0.56
25	0.38	0.51	0.62
30	0.42	0.55	0.67

See Figure 11.0 for the delay in seconds between the receiver's phone and the next three on the same network with 100Kb frame size. Figure 11.0 and table 6.0 demonstrate that sending 100Kb frames at once delays the receiver phone by 0.25 seconds for the first phone, 0.32 for the second, and 0.38 for the third. The time delay factor reduces video streaming between sending and receiving mobiles while sending to the first recipient, increases by 0.07 seconds for the second receiver, and increases by 0.06 seconds for the third receiver. Sending 5 frames of 100Kb delays the first receiver 0.29, the second 0.36, and the third 0.44. This estimate reveals that the second receiver is 0.07 seconds slower than the first. Frame delay increases to 0.08 seconds when the third receiver receives. Thus, sending frames to the third recipient increases delay. First delay in seconds is 0.33, second 0.41, and third 0.48 for 10 100 Kb frames. The delay between first and second is 0.08 and second and third is 0.07.

First, second, and third delays in seconds are 0.37, 0.46, and 0.53 for 15 frames of 100 Kb. By delivering 20 100 Kb frames, the first delay is 0.41 seconds, the second 0.5, and the third 0.58. The first delay in seconds for 25 100 Kb frames is 0.45, the second 0.53, and the phone 0.64. With 30 frames of 100, Kb first and second delays in seconds are 0.51 and 0.55, while third is 0.68. Table-4.10 illustrates the delay from receiving device to following connected devices up to 3 with 100 KB frame size.

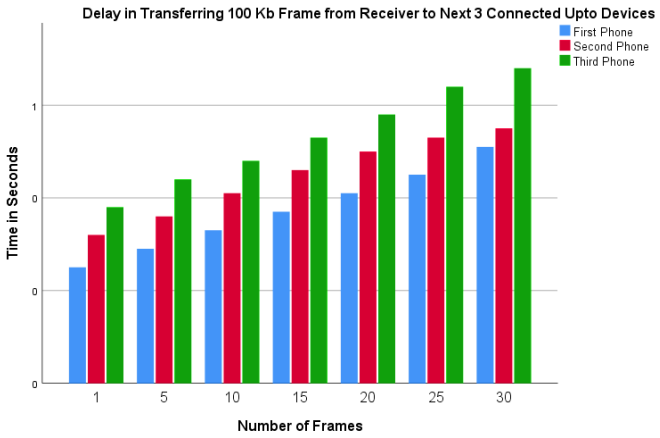

Figure 11.0 *100KB sending frame graph.*

Table 6.0 *100KB frame Dataset*

One phone to other			
Frame 100 KB	Delay in sec for 1 Phone	Delay in sec for 2 Phone	Delay in sec for 3 Phone
1	0.25	0.32	0.38
5	0.29	0.36	0.44
10	0.33	0.41	0.48
15	0.37	0.46	0.53
20	0.41	0.5	0.58
25	0.45	0.53	0.64
30	0.51	0.55	0.68

Figures 12.0 and 13.0 compare delays in three mobile phones that deliver frames from the first phone. The graph also shows the frame size and a bar showing how frame size and number of frames affect delay time.

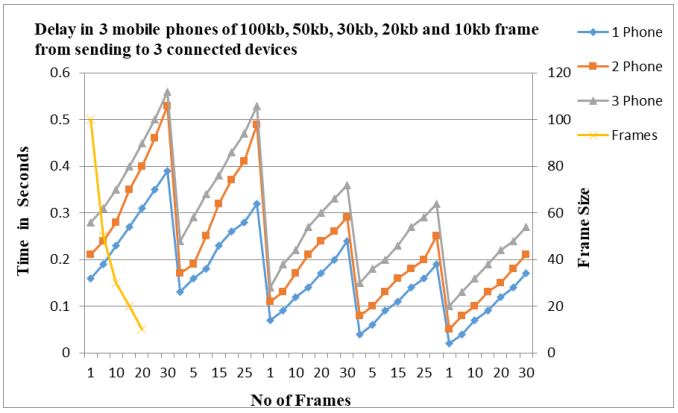

Figure 12.0 *Detailed comparison according to frame size from sender phone.*

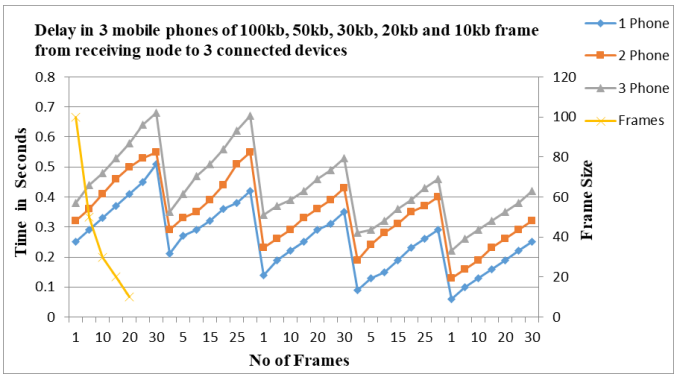

Figure 13.0 *Detailed comparison according to frame size from receiver phone*

The results show that the delay factor between connected and sending phones varies with frame size and phone count. Single-frame streaming reduces video delay but lowers quality. As the number of frames increases, quality improves but delay increases. A 50KB frame streamed between three phones has a minimal latency for the first. The second increases, but the third increases delay but improves quality. So, the conclusion is that delivering more frames between connected phones improves streaming, however delivering one frame reduces delay but increases quality and delay.

CONCLUSION:

This study examined the evolving nature of digital communication, with particular focus on the indisputable impact of live streaming in today's technology-driven culture. The submitted thesis, titled "Sharing Revolution with Distributed Mobile Streaming Bridge Protocol," goes deeply into the nuances of node-to-multi-node communication in a server-less mobile phone context.

Based on our findings, it may be possible to forego using centralized servers and other network infrastructure. At the heart of this system is the Bridge Protocol (BP), which emerges as a critical mediator, acting across two networks on the same NIC and extending its capabilities between phones. This protocol not only improves communication, but also fundamentally alters how mobile interactions are designed and implemented. The given algorithm/framework illustrates a dynamic shift in how several mobile phones can communicate and share screens via the hotspot feature.

Future Directions:

While the current implementation of the Bridge Protocol is groundbreaking, it does pose some difficulties. Delay, mobility, and redundancy must all be optimized for. Research going forward can concentrate on optimizing the algorithm to reduce lag time in communications, increase throughput, and incorporate autonomous mobility features. To further facilitate node-to-multi-node communication, it is important to emphasize more reliable failure detection tools and proactive prevention techniques.

Given the dynamic nature of the technology industry, this study has the potential to pave the way for robust, efficient, and decentralized mobile communication systems in the years to come.

REFERENCES

[1] P. R. Mahalingam, “A Conceptual Framework for Scaling and Security in Serverless Environments Using Blockchain and Quantum Key Distribution,” in *Quantum and Blockchain for Modern Computing Systems: Vision and Advancements*, vol. 133, A. Kumar, S. S. Gill, and A. Abraham, Eds., in Lecture Notes on Data Engineering and Communications Technologies, vol. 133. , Cham: Springer International Publishing, 2022, pp. 157–182. doi: 10.1007/978-3-031-04613-1_5.

[2] A. A. Khan, A. A. Laghari, M. Shafiq, S. A. Awan, and Z. Gu, “Vehicle to everything (V2X) and edge computing: A secure lifecycle for UAV-assisted vehicle network and offloading with blockchain,” *Drones*, vol. 6, no. 12, p. 377, 2022.

[3] S. Hu, X. Chen, W. Ni, E. Hossain, and X. Wang, “Distributed machine learning for wireless communication networks: Techniques, architectures, and applications,” *IEEE Commun. Surv. Tutor.*, vol. 23, no. 3, pp. 1458–1493, 2021.

[4] M. Adhikari, T. Amgoth, and S. N. Srirama, “A Survey on Scheduling Strategies for Workflows in Cloud Environment and Emerging Trends,” *ACM Comput. Surv.*, vol. 52, no. 4, pp. 1–36, Jul. 2020, doi: 10.1145/3325097.

[5] A. A. Khan, A. A. Wagan, A. A. Laghari, A. R. Gilal, I. A. Aziz, and B. A. Talpur, “BIoMT: A state-of-the-art consortium serverless network architecture for healthcare system using blockchain smart contracts,” *IEEE Access*, vol. 10, pp. 78887–78898, 2022.

[6] P. Raith, T. Rausch, A. Furutanpey, and S. Dustdar, “*faas-sim* : A trace-driven simulation framework for serverless edge computing platforms,” *Softw. Pract. Exp.*, vol. 53, no. 12, pp. 2327–2361, Dec. 2023, doi: 10.1002/spe.3277.

[7] A. Sabbioni, “Serverless middlewares to integrate heterogeneous and distributed services in cloud continuum environments,” 2023, Accessed: Nov. 21, 2023. [Online]. Available: http://amsdottorato.unibo.it/id/eprint/10893

[8] S. Trilles, A. González-Pérez, and J. Huerta, “An IoT platform based on microservices and serverless paradigms for smart farming purposes,” *Sensors*, vol. 20, no. 8, p. 2418, 2020.

[9] J. Spenger, P. Carbone, and P. Haller, “Portals: An Extension of Dataflow Streaming for Stateful Serverless,” in Proceedings of the 2022 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, Auckland New Zealand: ACM, Nov. 2022, pp. 153–171. doi: 10.1145/3563835.3567664.

[10] G. Amarasinghe, M. D. de Assuncao, A. Harwood, and S. Karunasekera, “ECSNeT++: A simulator for distributed stream processing on edge and cloud environments,” *Future Gener. Comput. Syst.*, vol. 111, pp. 401–418, 2020.

[11] W. Kassab and K. A. Darabkh, “A–Z survey of Internet of Things: Architectures, protocols, applications, recent advances, future directions and recommendations,” *J. Netw. Comput. Appl.*, vol. 163, p. 102663, 2020.

[12] J. Kim, Y. Jung, H. Yeo, J. Ye, and D. Han, “Neural-Enhanced Live Streaming: Improving Live Video Ingest via Online Learning,” in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, Virtual Event USA: ACM, Jul. 2020, pp. 107–125. doi: 10.1145/3387514.3405856.

[13] D. Barcelona-Pons, P. Sutra, M. Sánchez-Artigas, G. París, and P. García-López, “Stateful Serverless Computing with CRUCIAL,” *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 3, pp. 1–38, Jul. 2022, doi: 10.1145/3490386.

[14] Å. Hugo, B. Morin, and K. Svantorp, “Bridging MQTT and Kafka to support C-ITS: A feasibility study,” in *2020 21st IEEE International Conference on Mobile Data Management (MDM)*, IEEE, 2020, pp. 371–376. Accessed: Nov. 22, 2023. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9162327/

[15] J. A. dela Cruz, N. J. Libatique, and G. Tangonan, “Design of a disaster information system using mobile cloud wireless mesh with delay tolerant network,” in *2019 IEEE Global Humanitarian Technology Conference (GHTC)*, IEEE, 2019, pp. 1–8. Accessed: Nov. 22, 2023. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9033450/

[16] T. Pfandzelter, S. Henning, T. Schirmer, W. Hasselbring, and D. Bermbach, “Streaming vs. functions: a cost perspective on cloud event processing,” in *2022 IEEE International Conference on Cloud Engineering (IC2E)*, IEEE, 2022, pp. 67–78. Accessed: Nov. 21, 2023. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9946366/