A Study on Spring and Kafka Integration

1st Saksham Garg
Department of Information Science and Engineering
R.V. College of Engineering Bengaluru

2nd Kiran Hegde
Department of Information Science and Engineering
R.V. College of Engineering

3rd R.O. Sugam Kumar
Department of Information Science and Engineering
R.V. College of Engineering Bengaluru

4th Rashmi R
Department of Information Science and Engineering
R.V. College of Engineering Bengaluru

5th Rekha B S
Department of Information Science and Engineering
R.V. College of Engineering Bengaluru

6th Dr. Mamatha G S
Department of Information Science and Engineering
R.V. College of Engineering Bengaluru

***Abstract*—This paper is being written as to publish findings on the different ways to integrate a spring boot application with kafka and find the best way of performing this using different types of architectures**

Index Terms:- Spring boot, kafka, messaging, microservice, monolith, architecture

I. INTRODUCTION

In this paper, we attempt to look at the various ways of integrating kafka into an application making use of spring boot. Spring boot provides us with the facility of creating a robust application. This allows us to integrate kafka into a apring boot application in a variety of ways. Using this paper we will draw conclusion as to which architectures supports kafka integration the best and how to structure our application according to the usecase.

II. SPRING BOOT APPLICATIONS

A. What is a Spring Boot Application

Microservices can be made using spring boot which is an open source java based project. It is created by Pivotal Team and is utilized to assemble independent and creation prepared spring applications.Spring boot provides a ready to go application that can just be run without much setup. We are able to start having minimaldependecies without the requirement for a whole Spring design setup.

@EnableAutoConfiguration when used in classpath can look and auto initalize and configure from the source, for eg in mysql if you have not made the fields , this annotation will generate an in memeory dataset following this schema

A spring boot application will contain the @SpringBootAp- plication annotaion in class which defines the scope of the

application and makes the application applicable to spring features.

Consequently Spring Boot examines every one of the parts remembered for the undertaking by utilizing @ComponentScan explanation.

B. Spring Boot and the Micro Service architecture

Micro Service is an engineering that permits the designers to create and convey benefits autonomously. Each assistance run- ning has its own cycle and this accomplishes the lightweight model to help business applications.Microservices are an ad-vanced way to deal with programming whereby application code is conveyed in little, reasonable pieces, autonomous of others.Their limited scope and relative confinement can prompt numerous extra advantages, like simpler support, im-proved usefulness, more prominent adaptation to non-critical failure, better business arrangement, and more.

A micro service architecture is used in distributed systems , which comes with its own challenges. Spring boot helps overcome these challenges by fulfilling acting as multiple components which would be needed, like an API Gateway or a load balancer.

C. Spring Boot and the Monolith architecture

In a monolithic framework, the administrations that involve the framework are coordinated coherently inside a similar code base and unit of arrangement. This permits the interdepen-dency between administrations to be overseen inside the equiv-alent runtime, and can imply that regular models and assets can be shared by the different segments of the framework. The interconnectivity between the subsystems inside a solid frame- work implies that business rationale and information spaces can accomplish a high measure of reusability in reflections and utility capacities, regularly through close coupling, yet with the possible advantage of having the option to find out what a solitary change may mean for the whole framework. That extravagance comes at the cost of versatility of the individual parts of the framework, and implies that the asset impression for the framework is helpless before its most un- adaptable viewpoints.

III. APACHE KAFKA

A. What is Apache Kafka

Apache Kafka is a community distributed message stream-ing stage fit for dealing with trillions of occasions a day.The project expects to give a brought together, high-throughput, low-inactivity stage for taking care of continuous information takes care of. Kafka can interface with outside frameworks (for information import/send out) by means of Kafka Connect and gives Kafka Streams, a Java stream preparing library. Kafka utilizes a twofold TCP-based convention that is streamlined for proficiency and depends on a ”message set” deliberation that normally gathers messages to diminish the overhead of the organization roundtrip.

B. Apache kafka in Micro service architecture

One of the customary methodologies for imparting between microservices is through their REST APIs. Notwithstanding, as your framework advances and the quantity of microservices develops, correspondence turns out to be more unpredictable, and the engineering may begin taking after our old companion the spaghetti against design, with administrations relying upon one another or firmly coupled, hindering improvement groups. This model can display low idleness however possibly works if administrations are made profoundly accessible.

To conquer this plan weakness, new structures mean to de-couple senders from recipients, with nonconcurrent informing. In a Kafka-driven engineering, low inertness is saved, with extra benefits like message adjusting among accessible buyers and brought together administration.

Apache Kafka is the most famous apparatus for microser-vices since it settles large numbers of the issues of mi-croservices coordination while empowering ascribes that mi-croservices mean to accomplish, like versatility, effectiveness, and speed. It likewise works with between administration correspondence while safeguarding super low latency and fault tolerance.

C. Apache kafka in Monolithic architecture

Event Driven Architecture allude to rather old idea of Soft-ware Engineering that acquired a ton of significance as of late because of the requirement for development in big business framework reconciliation, particularly when discussing always developing intricacy in microservice arrangements.

A significant nature of such frameworks is free coupling — idea that engages programming segments of an appropriated framework to prevail at correspondence whether they are available at now is the ideal time.

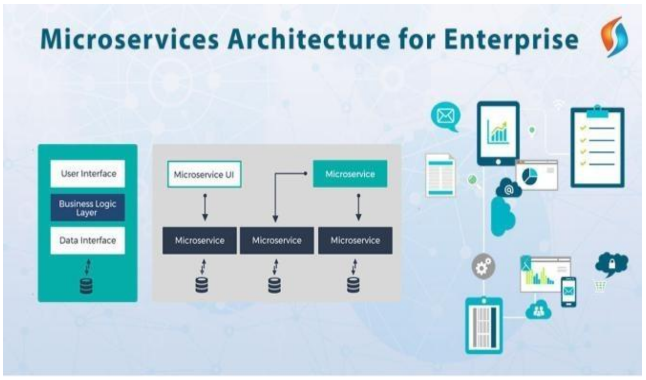

Fig. 1. Microservice Architecture

Free coupling defines clear limits around segment duties, empowering quicker turn of events, testing and incorporation inside more modest cross-utilitarian groups.

Occasion driven design comprises of three fundamental parts: maker, buyer, and merchant. The maker trusts that the occasion will happen then it makes an impression on the dealer administration which associates it to the customer.

The shopper can peruse the message from the dealer and makes a move as a callback – a capacity that reacts to that occasion. Here the maker doesn’t anticipate a message from the customer, in contrast to the RestAPI, there should be a reaction that may prompt a break in the wake of sitting tight for a reaction.

Events can be put away in something many refer to as occa- sion logs, which can store the last condition of an event.The principle justification utilizing Kafka for an occasion driven framework is the decoupling of microservices and production of a Kafka pipeline to interface makers and customers. Without checking for new information, all things being equal, you can basically pay attention to a specific occasion and make a move.

Kafka offers a half breed model that incorporates preparing and informing that is consistently versatile.

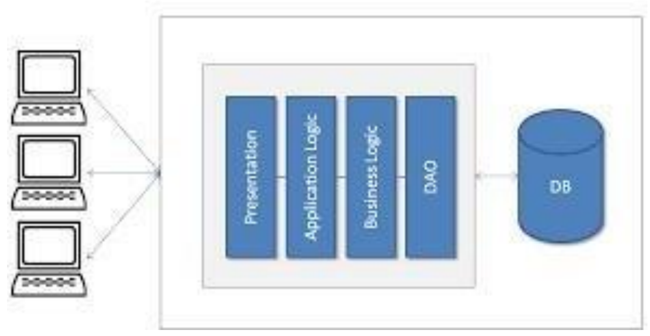

Fig. 2. Monolith Architecture

IV. APACHE KAFKA INTEGRATION WITH SPRING BOOT APPLICATIONS

A. Why use Apache Kafka

Applications produced increasingly more information than any time in recent memory and a tremendous piece test - even prior to being examined - we obliging the heap in any case. Apache’s Kafka addresses this difficulty. This was initially planned by the social media site Linked in and this way publicly released in 2011. The undertaking plans to give a brought together, high-throughput, low-inactivity stage for dealing with continuous information takes care of. The plan is vigorously impacted by exchange logs. It is an informing framework, like conventional informing frameworks such as ActiveMQ, MQSeries, RabbitMQ, however this is the best option for collection of logs, diligent informing, quick (many megabytes each second!) peruses and composes, and can oblige various customers. This gives it the ability to scale to a cloud native level and handle requests at scale.

Kafka, these days has been adopeted widely and is powering a lot of production grade applications in many companies throughout the world. LinkedIn, the company where kafka was initially developed uses it to power their News and Daily section with flowing activity data that will eventually make its way to a Massively Parallel Hadoop Cluster.The stream processing pipeline at the heavily used microblogging site Twitter makes intensive use of Kafka. Foursquare, The location data provider also makes use of

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 06, June-2021

Kafka to keep its online and offline capabilities synchronized. At Square, it has been repurposed as a message bus for movement of system events throughout all its data centres across the world. It is observed to have been propogating a variety of data like custom metrics, application logs etc. It is also versatile with its integration on the consumer side with Esper, Graphite and Splunk. It has also stood the test of time with the ability to scale to 400 to 650 billion messages per day on Netflix . The list of companies currently having Kafka in their active production environments is endless, it is known to be used in at least 4/5 of the Fortune 500 companies. The tool has stood the test of time and has been actively kept up to the latest trends and standards that makes Kafka an obvious choice.

B. Reactive Programming against Imperative programming

Imperative writing computer programs is the sort of programming we as a whole begin with. Something occurs, all in all an occasion happens, and your code is told of that occasion. For instance, a client clicked a catch and where you handle the occasion in your code, you choose what that activity should intend to your framework. You may save records to a DB, call another help, send an email, or a blend of these. The significant piece here, is that the occasion is straightforwardly coupled to explicit activities occurring.

Reactive programming empowers you to react to occasions that happen, frequently as streams. Various concerns can buy in to a similar occasion and let the occasion have it's impact in it's area, paying little heed to what occurs in different spaces. All in all, it considers approximately coupled code that can without much of a stretch be reached out with greater usefulness. It's conceivable that different huge down-stream frameworks coded in various stacks are influenced by an occasion, or even an entire pack of serverless capacities executing some place in the cloud.

C. Apache Kafka against message queues

To comprehend what Kafka will bring to your design, we should begin by discussing message lines. We'll begin here, on the grounds that we will discuss it's restrictions and afterward perceive how Kafka tackles them.

A message line permits a lot of endorsers of pull a message, or a bunch of messages, from the finish of the line. Lines for the most part consider some degree of exchange when pulling a message off, to guarantee that the ideal activity was executed, before the message gets eliminated.

Not all queueing frameworks have a similar usefulness, yet once a message has been handled, it gets taken out from the line. Looking at this logically, it's basically the same as imperative programming, something occurred, and the starting framework concluded that a specific activity ought to happen in a downstream framework.

Despite the fact that you can scale out with numerous shoppers on the line, they will all contain a similar usefulness, and this is done just to deal with burden and interaction messages in equal, at the end of the day, it doesn't permit you to start off various free activities dependent on a similar occasion. Every one of the processors of the line messages will execute a similar kind of rationale in a similar space. This implies that the messages in the line are really orders, which is fit towards imperative programming, and not an occasion, which is fit towards reactive programming.

With Kafka then again, you distribute messages/occasions to subjects, and they get endured. They don't get taken out when shoppers get them. This permits you to replay messages, how- ever more critically, it permits a large number of purchasers to handle rationale dependent on similar messages/occasions.

You can in any case scale out to get equal handling in a similar space, yet more significantly, you can add various sorts of customers that execute distinctive rationale dependent on a similar occasion. All in all, with Kafka, you can embrace a reactive bar/sub engineering.

This is conceivable with Kafka because of the way that messages are held and the idea of purchaser gatherings. Purchaser bunches in Kafka recognize themselves to Kafka when they request messages on a subject. Kafka will record which messages (balance) were conveyed to which shopper bunch, with the goal that it doesn't serve it up once more.

D. Working of Apache Kafka

A broker of Kafka group may have at least 1 worker win which each of these brokers will have atleast one broker cycle running. kakfa is intended to be highly easy to use and accessible. There are no special nodes, all nodes work in a peer to peer level and can be used interchangeably . Information

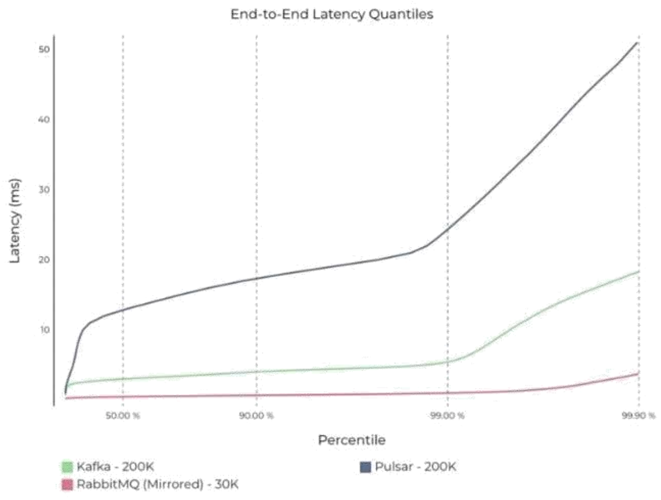

Fig. 3. Latency Comparison between Kafka and MQ

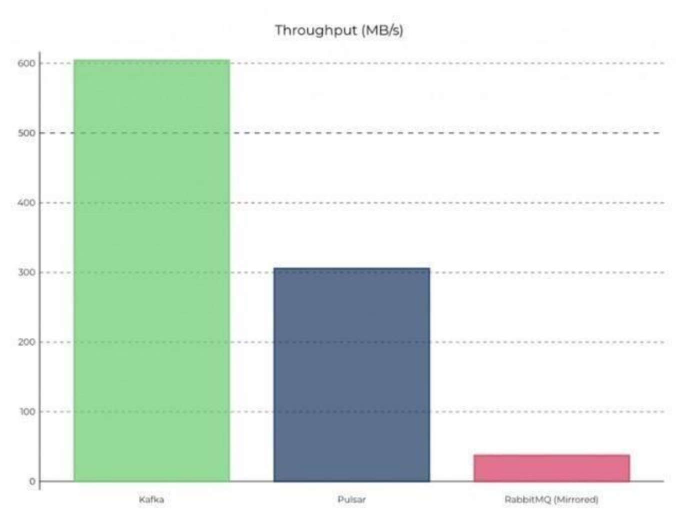

Fig. 4. Throughput(in Mb) of MQ

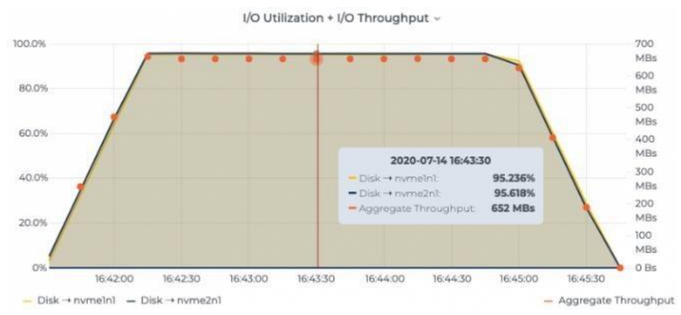

Fig. 5. Utilization of MQ

and data can be easily replayed, reproduced and fault tolerant in case a service goes down.

In Kafka, a subject is a classification, like a JMS objective or both an AMQP trade and line. Points are parceled, and the decision of which of a theme's segment a message ought to be shipped off is made by the message producer. Each message in the segment is allocated a novel sequenced ID, its coun-terbalance. More parcels permit more prominent parallelism for utilization, however this will likewise bring about more documents across the brokers.

Producers send messages to Apache Kafka broker points and determine the segment to use for each message they produce. Message creation might be simultaneous or offbeat. Producers additionally indicate what kind of replication en-sures they need.

Consumers tune in for messages on points and interaction the feed of distributed messages. As you'd expect in the event that you've utilized other informing frameworks, this is normally (and conveniently!) nonconcurrent.

Like Spring XD and various other distributed framework, Apache Kafka utilizes Apache Zookeeper to facilitate group data. Apache Zookeeper gives a common various leveled namespace (called znodes) that hubs can share to comprehend bunch geography and accessibility (one more explanation that Spring Cloud has approaching help for it..). Animal specialist is available in your connections with Apache Kafka. Apache Kafka has, for instance, two distinctive APIs for going about as a consumer. The more elevated level API is less complex to begin with and it handles every one of the subtleties of taking care of dividing, etc. It will require a reference to a Zookeeper example to keep the coordination state.

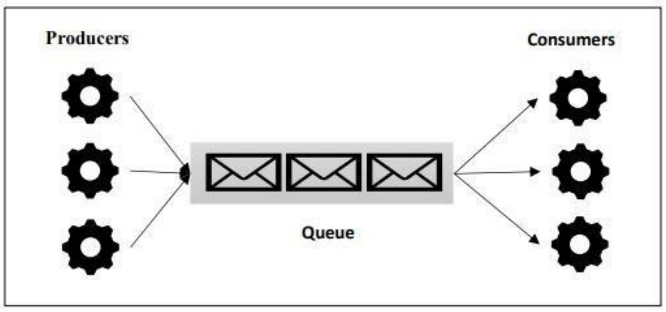

Fig. 6. Architecture of a generic message queue

CONCLUSION

From all the points that we have talked about above, we come to a conclusion that a spring boot application is best suited to use a microservice architecture with reactive programming, using this type of a base not only helps us build scalable applications which are easier to make and debug but are also the best usecase if we want to integrate kafka, message queues can be used if we want to use a monolith application via MVC.Some benefeits of Apache Kafka are:

highly scalable Tolerant to faults

uses a fair publisher-subscriber messaging architecture

High throughput

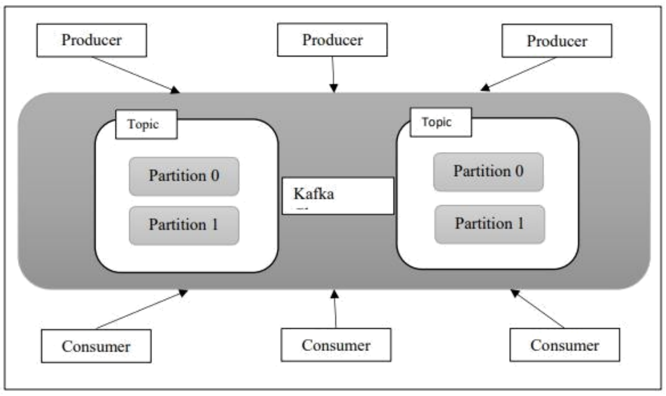

Fig. 7. Architecture of a Kafka Architecture

Durable

Reliable

Very Performant

making it a great fit with the microservice architecture which complements the same features.

ACKNOWLEDGMENT

This paper was made possible thanks to our guides and mentors from R.V. College of Engineering- Associate Dean Mamatha G.S. , Assistant Professor Rashmi R, Assistant professor Rekha B.S.

REFERENCES

[1] H. Wu, Z. Shang and K. Wolter, ”TRAK: A Testing Tool for Studying the Reliability of Data Delivery in Apache Kafka,” 2019 IEEE International Symposium on Software Reliability Engineer-ing Workshops (ISSREW), 2019, pp. 394-397, doi: 10.1109/ISS- REW.2019.00101.

[2] C. N. Nguyen, J. Kim and S. Hwang, ”KOHA: Building a Kafka-Based Distributed Queue System on the Fly in a Hadoop Cluster,” 2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W), 2016, pp. 48-53, doi: 10.1109/FAS-W.2016.23.

[3] K. Guntupally, R. Devarakonda and K. Kehoe, ”Spring Boot based REST API to Improve Data Quality Report Generation for Big Scientific Data: ARM Data Center Example,” 2018 IEEE International Conference on Big Data (Big Data), 2018, pp. 5328-5329, doi: 10.1109/BigData.2018.8621924,

[4] L. xuchen and L. chaoyu, ”Design and Implementation of a Spring Boot-Based Data Collection System,” 2020 12th International Conference on Intelligent Human-Machine Sys-

[5] tems and Cybernetics (IHMSC), 2020, pp. 236-239, doi: 10.1109/IHMSC49165.2020.00059.

[6] R. Shree, T. Choudhury, S. C. Gupta and P. Kumar, ”KAFKA: The modern platform for data management and analysis in big data domain,” 2017 2nd International Conference on Telecommunica-tion and Networks (TEL-NET), 2017, pp. 1-5, doi: 10.1109/TEL-NET.2017.8343593.

[7] Y. Drohobytskiy, V. Brevus and Y. Skorenkyy, ”Spark Struc-tured Streaming: Customizing Kafka Stream Processing,” 2020 IEEE Third International Conference on Data Stream Min-ing ‘I&’ Processing (DSMP), 2020, pp. 296-299, doi: 10.1109 DSMP47368.2020.9204304.

[8] V. John and X. Liu, ”A survey of distributed message broker queues”, arXiv preprint arXiv. 1704.00411, 2017

[9] . R. Mayer, L. Brunie, D. Coquil and H. Kosch, ”On reliability in publish/subscribe systems: a survey”, International Journal of Parallel Emergent and Distributed Systems, vol. 27, no. 5, pp. 369-386, 2012.

[10] J. Kreps, N. Narkhede, J. Rao et al., ”Kailca: A distributed messaging system for log processing”, Proceedings of the NetDB, pp. 1-7, 2011.

[11] P Carbone, S. Ewen, G. Fora,´ S. Haridi, S. Richter and K. Tzoumas, ”State management in apache flink®: consistent stateful distributed stream processing”, Proceedings of the VLDB Endow-ment, vol. 10, no. 12, pp. 1718-1729, 2017.

[12] K. Shvachko, H. Kuang, S. Radia and R. Chansler, ”The Hadoop Distributed File System”, Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST‘10), 2010.

[13] ”Spring Boot Reference Guide”, Spring Boot Reference Guide, [online] Available: docs.spring.io/spring-boot/docs/current/reference/htmlsingl e.

[14] K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, et al., ”Apache Hadoop YARN: Yet Another Resource Negotiator”, Proceedings of the 4th Annual Symposium on Cloud Computing (SoCC‘13), Oct. 2013.

[15] H. Wu, Z. Shang and K. Wolter, ”Performance Prediction for the Apache Kafka Messaging System,” 2019 IEEE 21st In-ternational Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2019, pp. 154-161, doi: 10.1109/HPCC/SmartCity/DSS.2019.00036.

[16] S. Shorgin, A. Pechinkin, K. Samouylov, Y. Gaidamaka, E. Sopin and E. Mokrov, ”Queuing systems with multiple queues and batch arrivals for cloud computing system performance analysis,” 2014 International Science and Technology Conference (Mod-ern Networking Technologies) (MoNeTeC), 2014, pp. 1-4, doi: 10.1109/MoNeTeC.2014.6995600.

[17] S. Wang, X. Li and R. Ruiz, ”Performance Analysis for Het-erogeneous Cloud Servers Using Queueing Theory,” in IEEE Transactions on Computers, vol. 69, no. 4, pp. 563-576, 1 April 2020, doi: 10.1109/TC.2019.2956505.

[18] S. U. Sharma and Y. B. Gandole, ”Virtualization approach to reduce network latency for thin client performance optimization in cloud computing environment,” 2014 International Conference on Computer Communication and Informatics, 2014, pp. 1-6, doi: 10.1109/ICCCI.2014.6921753.

[19] T. G. Rodrigues, K. Suto, H. Nishiyama and N. Kato, ”Hybrid Method for Minimizing Service Delay in Edge Cloud Computing Through VM Migration and Transmission Power Control,” in IEEE Transactions on Computers, vol. 66, no. 5, pp. 810-819, 1 May 2017, doi: 10.1109/TC.2016.2620469.

[20] C. Jeong, T. Ha, J. Kim and H. Lim, ”Quality-of-service aware resource allocation for virtual machines,” 2017 International Con- ference on Information Networking (ICOIN), 2017, pp. 191-193, doi: 10.1109/ICOIN.2017.7899502

[21] Jiann-Liang Chen, Y. T. Larosa and Pei-Jia Yang, ”Optimal QoS load balancing mechanism for virtual machines scheduling in eucalyptus cloud computing platform,” 2012 2nd Baltic Congress on Future Internet Communications, 2012, pp. 214- 221, doi: 10.1109/BCFIC.2012.6217949.