Universität Stuttgart

IPVS

Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master's Thesis Nr. 3739

Evaluation and Analysis of Realizing Broker-based Content Routing Protocols in SDN

Lobna Hegazy

Course of Study:	Computer Science
Examiner:	Prof. Dr. Kurt Rothermel
Supervisors:	Dr. M. Adnan Tariq
	Dr. Amr El-Mougy
	Sukanya Bhowik
Commenced:	2015-08-01
Completed:	2016-08-29
CR-Classification:	C.2.1,C.2.4

Dedicated to my mother.
Everything I did, I do, or I will is just to make you proud.
My life is just my way to you in heaven in shaa' ALLAH.

Acknowledgments

Before giving any acknowledgements, I have to thank ALLAH for giving me such an opportunity, for providing me with the strength to complete this work, for guiding me when I feel lost, for the limitless blessings, for anything and everything. I want to express my gratitude for my family for always being by my side as a concrete supportive system. Full respect and appreciation to my supervisors Dr. M. Adnan Tariq, Dr. Amr ElMougy, and Sukanya Bhowmik, M.S. I would never be able to do it without your continuous guidance and kindly supporting me both technically and personally. Special thanks goes to my friends for always being there for me when it is needed and when it is not. I would like to take this as an opportunity to particularly acknowledge Yomna AbdElRahman, M.S. great role in making my experience such a beneficial one, not only technically, but personally as well. The affirmative influence that the German University in Cairo reflects on shaping me as a person before as a student cannot be denied, I will always be that loyal ambassador. Finally, I want to ask ALLAH for giving me the needed ability, willingness and support to successfully get my masters degree based on the work presented here, hoping that it serves the wellness of the mankind.

Abstract

Publish/subscribe provides a valuable communication model to the future Internet due to the decoupling of end-users from each other. One of the stubborn challenges that face recent content-based publish/subscribe systems is the trade-off between the usage of the network bandwidth and the end-to-end delay of published events. This trade-off is imposed by the fact that most implementations depend on software brokers to filter incoming messages towards received requests from subscribers. Although this approach for filtering may present the most bandwidth efficient solutions, the use of brokers adds to the network end-to-end delay. The installed brokers are implemented at the application layer and hence the original path between publishers and subscribers is extended which adds to the delay in which messages are forwarded from publishers to subscribers. Along with the delay imposed by the extended path, another processing delay is added to the system based on the time needed for filtering incoming messages at the brokers.

As the time factor is crucial to the real-world applications that depend on the content-based publish/subscribe paradigm, recent implementations try to tackle this problem by exploiting the deployed hardware in the underlying infrastructure for filtering operations. In-network filtering is enabled with the help of Software Defined Networking (SDN) technology as it allows the installment of content filters directly to the network switches/routers. Even though this approach significantly reduces the end-to-end delay, it suffers when the bandwidth efficiency is evaluated. Caused by the inherited hardware limitations, installing content filters on hardware network elements limits their expressiveness. This increases the number of published messages from publishers to subscribers on different network links which requires more bandwidth. As an intermediate solution between the two filtering approaches, the work of this thesis is the realization of a hybrid content-based publish/subscribe middleware that allows filtering operations in both network and application layers.

Contents

Chapter 1

Introduction

The Internet of the future is expected to host tens of billions of devices generating huge amounts of data according to Cisco recent statistics [8]. Thus, even though current advancements of technology in the field of networking now enables a high degree of connectivity across a large number of computers, applications, and users, it will not be enough to support future demands of users. This is because the traditional end-to-end networking paradigm, which is primarily based on directly connected hosts, will be severely inadequate to support the enormous scale of the future Internet. In a traditional scenario, Content Providers (CPs) store (cache) their data at a number of geographically distributed servers called Content Distribution Networks (CDNs). Based on this content distribution architecture, whenever a user requests a content object, the location of the resource hosting server must be identified first by querying a search engine. In order to serve requested content to consumers from the nearest server, CDNs work by creating an overlay network over the Internet. After getting feedback from the search engine about the resource address, the content is delivered to the user. By considering billions of connected devices, such location-dependent distribution fails when network scalability is to be taken into account. Moreover, there are other concerns, e.g., about the increased network traffic on servers across different locations, variable delivery cost on network different paths, and high storage requirements due to the existence of the same content replicas across different servers [3].

Information-Centric Networking (ICN) paradigm [25] has been proposed as an alternative model to address a set of issues related to the present Internet and content distribution architectures. Instead of storing data on distributed hosting servers, ICN exploits installed in-network elements, i.e., servers, routers, nodes, and data centers for caching data. This allows content providers to be more loosely coupled of users, meanwhile, content delivery is independent of the resource physical location as now it is considered to be *centric*. The main abstraction of ICN is the replacement of host addresses with content names. A key advantage of this paradigm is that it has led to the development of content distribution models where consumers are able to access named content, without actually requiring a specific host location. One of these paradigms is the known content-based Publish/Subscribe (Pub/Sub) communication model.

In content-based Pub/Sub systems, an information-centric communication paradigm is used, where subscribers declare their interests by means of selection predicates while publishers simply publish their content. The service consists of delivering to any and all subscribers each message that matches the selection predicates declared by those subscribers. The flow of the messages from publishers to subscribers is routed by the content of the message, rather than explicit addresses assigned by publishers and attached to the messages. As the concept seems to be a promising replacement solution for the location-based delivery model, applying the content-based Pub/Sub paradigm to access named content can be really challenging. In the most common scenario, publishers and subscribers have no reciprocal knowledge about each other. In addition, traditional switches and routers have no processing capabilities to examine the content of transmitted packets of data for matching published messages against subscriptions requests. As a solution, the idea of implementing content brokers was proposed. Here, publishers connect to the broker, which is a content-based router, to publish events they want to make the world aware of, and subscribers connect to the broker to establish subscriptions, in which they specify the set of messages content they are interested in receiving. By using content filters, which are content-based forwarding rules installed on brokers, the main function of the broker is to match published messages with subscriptions and deliver to the subscribers the messages of interest. The broker on behalf of the subscribers works on evaluating the subscriber's selection criteria against the incoming messages. Brokers here can be seen as the combination of efficient and reliable multicast delivery with advanced filtering capabilities. This described scenario represents the major advantage of content-based routing using brokers, which is improving system overall bandwidth efficiency by only forwarding content to subset of subscribers who are actually interested in the published content.

Although many Pub/Sub systems implementations supported content-based routing at overlay networks of software brokers to offer a bandwidth-efficient solution, these implementations failed to take advantage of performance benefits of communication protocols implemented on the network layer when it comes to system responsiveness. This is because the installed brokers between publishers and subscribers are implemented at the application layer, introducing by that an additional significant delay caused by extending the path length between publishers and subscribers, in addition to the processing delay for matching published messages. To make use of the benefits of the implemented communication protocols on the network layer, content-based Pub/Sub systems relied on the technology of Sotware Defined Networking (SDN) [15]. SDN is an evolving and emerging technology in the world of networking that is usually defined as the decoupling of control and the forwarding/data planes. The difficulty of manipulating network control functions, which often hard-coded in the firmware of switches and routers, represents one of the main limitations of today's networks. As an attempt to solve this crucial problem, SDN proposes the transfer of the control functions to a piece of software running on an in-network server called the *controller*. The controller is mainly responsible for all the network control functions that include routing, security, load balancing, and traffic isolation. By making the controller responsible for all the control functions, SDN successfully achieves the decoupling of data and control planes. This decoupling introduces significant manageability and flexibility in the network and enables the deep programmability of SDN that allows it to be dynamically reconfigured.

Based on the fact that the controller has a global view of the network underlying structure, one of its main tasks is to create the paths between publishers and subscribers when integrated with content-based Pub/Sub systems. Using one of the many protocols standards that implement

SDN like OpenFlow [15], the controller has the capability of installing forwarding rules that act as content filters directly on the Ternary Content Addressable Memory (TCAM) of the SDN-enabled switches informing them how to deal with the different incoming packets. Thus, when events arrive from different publishers, it gets matched against these installed rules on switches TCAM, and then forwarded to interested subscribers. PLEROMA [21] discusses a middle-ware that exploits SDN technology by following in-network filtering approach to improve the performance of content-based Pub/Sub systems. Though this scheme offers in line-rate forwarding and high throughput rates, it suffers from challenging inherited limitations caused by in-network switches restricted capabilities in terms of processing and memory. Due to the limitation in processing capabilities, the only way to match incoming events against installed forwarding rules (flows) is to represent a new field in the packet header as a representation of the content of the message. This new field is to be matched against switches installed flows. The concept is simple but the question here is how to effectively represent the message contents in a packet header field.

The proposed model by PLEROMA builds on the principle of spatial indexing to provide such a representation where a series of bits, called *dz*-expression, is embedded in the packet header as a content presenter. As already another fields in packets headers occupy a large number of bits, e.g., source and destination IP addresses, there is a restriction on the number of available bits that can be used for representing the message content. Another challenge that comes as a consequence for the limited TCAM memory resources is the number of flow table entries that can be installed on one switch. In this scenario, it should be noted that the number of bits used for the match field inversely affects the number of flow entries that can be installed. And here we face a trade-off between filters expressiveness that affects overall system bandwidth efficiency and system scalability capabilities. To combine the bandwidth-efficient performance of application-layer filtering, and the line-rate forwarding of filtering in the network layer, this thesis builds on top of PLEROMA middleware to propose and implement a hybrid content-based Pub/Sub solution between the investigated two filtering approaches.

1.1 Thesis Organization

The contents of the thesis can be outlined as follows:

Chapter 2 gives a high-level specification of used concepts and technologies for the work of this thesis. Followed by another section surveying the state of the art in this research area where some famously implemented solutions are presented and the design issues are discussed.

The specifications of PLEROMA, a SDN-based Pub/Sub middleware, is discussed in chapter 3. It shows how PLEROMA takes advantage of the decoupling of control and data planes in SDN to offer a system with line-rate performance. This chapter gives a formal description of the semantics of the content-based Pub/Sub system used for the work of this thesis. After discussing some design issues the proposed middleware, the problem statement is formulated.

Chapter 4 presents the concepts behind the design of the proposed solution that serves as a solution to the previously stated problem. It starts by modeling the system architecture, followed by the logic design of implemented algorithms.

Chapter 5 is dedicated to analyzing and evaluating the performance of the proposed implemented solution in comparison with other solutions. To serve the purpose of this chapter, a number of experiments were run on the system using different data set. The specifications and the results of each experiment are included.

Chapter 2

Background

This chapter considers the basic concepts behind exploited technologies for the ground work of this thesis. It starts with the specifications of SDN technology, followed by the known Pub/Sub communication paradigm. For building more robust background, some real implementations are investigated for the purpose of stating the technology art.

2.1 Concepts and Technologies

Before going through the technical concepts of this thesis work, it is necessary to understand the concepts that relate to the integration of SDN with the Pub/Sub systems. This section gives a closer look to SDN technology and the general specifications of Pub/Sub systems separately. Along with introducing SDN, OpenFlow is introduced as one of the most widely used protocols that enable SDN.

2.1.1 Software Defined Networking

Network Control and Management

The technology evolution that mankind witness today, along with the Internet advancement, makes network management and monitoring in continuous evolution. Functionalities of today's network are managed with three main planes. Firstly, the *data plane* (forwarding plane) that is responsible for delivering data packets between endpoints. Secondly comes the *control plane* that is responsible for providing network elements with needed routing information. Lastly, the *management plane* that is responsible for monitoring the network and the configuration of data plane mechanisms and control plane protocols. A novel approach to network management and control can be found in [13].

SDN Overview

Traditional networks design has become ill-suited to the dynamic computing processes required for today's demanding applications. In the traditional static design, routing decisions that are carried by the control plane, and forwarding tasks that are carried by the data plane, occur on the same hardware device which makes it hardware-centric. This design poorly limits any network reconfiguration abilities as these tasks are programmed on dedicated hardware devices such as Application Specific Integrated Circuits (ASICs) in a firmware manner [15]. SDN is emerging as one of the most promising networking technologies of recent years to tackle this problem. As the name implies, it is an approach to networking in which control is decoupled from hardware and given to a software application called the *controller*, which is the heart of any SDN-based system. The main advantage of SDN comes from this separation, as decoupling the control from the hardware significantly improves the inflexibility of managing and controlling traditional networks. This is very useful for many real-time applications as the network can be dynamically reconfigured and the administration is much more flexible [12].

SDN Controller

As mentioned earlier, the main component of any SDN-based system is the controller. The controller substitutes the control functions, traditionally performed by network hardware devices. This is achieved by taking the control off the network hardware and running it as a piece of software instead, the device that runs this software is the controller. Controller functions include, but not limited to, system configuration, management and flow control. Now, the control plane and data plane are said to be decoupled, and any communication between different applications and physical network devices is done through the controller. This solves the inflexibility problem of reconfiguring traditional networks as it is not needed anymore to replace hardware elements for system reconfiguration, it just requires updating the controller software.

As one of the controller functions is to create paths between network end-points, it has a global view of the underlying network architecture. Those paths are represented as the installed forwarding rules (flows) on routers/switches forwarding tables (flow tables). Using one of SDN standard protocols, the controller communicates with network elements in the data plane to provide it with needed information to establish its flow tables. For the aim of this thesis, OpenFlow is the used protocol for the communication between the controller and SDN-compliant switches, discussed in the next section.

In order to implement SDN-based applications, you have the opportunity to choose from a wide spectrum of available controllers. The work presented in [17] provided a comparison between most common SDN controllers implementations. The one adopted for the work presented in this thesis is the floodlight controller. Floodlight project is an open source java based controller that is easy to use thanks to the easily-found documentation. More details can be found at the project official resource [1].

OpenFlow Protocol

Many protocol standards exist on the use of SDN for different applications. One of the most popular standards-based SDN is OpenFlow. OpenFlow emerged as a protocol to standardize the communication between the separated control plane and data plane [17]. It enables the implementation of SDN concept in both hardware and software. A distinguishable feature of OpenFlow protocol is that it can utilize the existing hardware to design new protocols without the need for installing new specialized devices as a wide spectrum of commercially available routers and switches support OpenFlow [15]. OpenFlow implements SDN concepts by allowing the centralized controller to remotely instruct the underlying data plane devices through providing it with forwarding tables. So is such a scenario, the control plane firset generates the forwarding rules and forward is to the data plane while the data plane uses these rules to send incoming packets [4].

SDN Architecture

The basic architecture for SDN-based networks is shown in Figure 2.1. It consists of three layers: forwarding plane, control plane, and application plane. The forwarding plane preliminary consists of interconnected SDN-compliant switches. Those switches can be *pure* SDN switches, or *hybrid* ones. Pure SDN switches supports only SDN required functions, while hybrid switches are traditional switches support SDN functions. One or more SDN controllers can de found in the control plane. The application plane has end-points running different applications.

The communication between the control plane and the data plane happens through the *southbound* Application Programmable Interface (API). The southbound API represents the used SDN standard protocol for control functions between controller(s) and switches. For the communication between the control plane and the application plane another separate API is used, the *northbound* API, which is controller vendor-specific. The work of this thesis uses FloodLight controller which exploits the Java API for communicating through the northbound API.

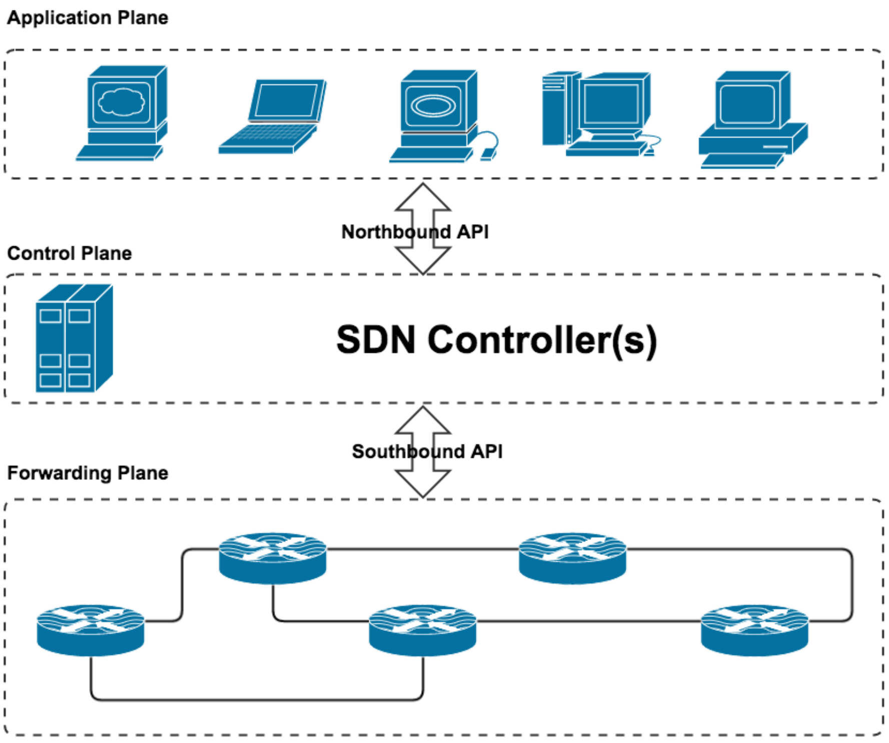

Figure 2.1: SDN Architecture

2.1.2 Publish/Subscribe Systems

Overview

Pub/Sub systems [23] are a special type of the well-known event notification services, they are a key technology for data sharing applications nowadays. Commonly, it is defined as asynchronous communication paradigm between application components, i.e., data senders and receivers. Asynchronous communication means that senders and receivers are decoupled with no direct contact, the interaction takes place through the intermediate Pub/Sub system. What differentiates Pub/Sub systems from the intuitive event notification services is that data receivers are not getting notified of all happening events, only a subset based on a selection mechanism provided by the receivers themselves. Compared with a model that does not use selection mechanism and notifies receivers about all events, the Pub/Sub paradigm offers significant savings in terms of bandwidth, especially if the selection mechanism provided by receivers is highly expressive.

System Model Elements

A typical Pub/Sub scenario involves two clients: data producers and data consumers. Data producers, referred as *publishers*, publish data in the form of *events* to the Pub/Sub system, while data consumers, referred to as *subscribers*, receive these events based on a special type of request called a subscription request. Subscribers express their interests about receiving events by sending subscription requests that define a specific type of events these subscribers are interested in. Upon receiving an event from publishers, the Pub/Sub system performs matching operations towards subscription requests. If one event is found to match any subscription, the corresponding subscriber gets notified about this event. A high-level model of this scenario is shown in figure 3.1.

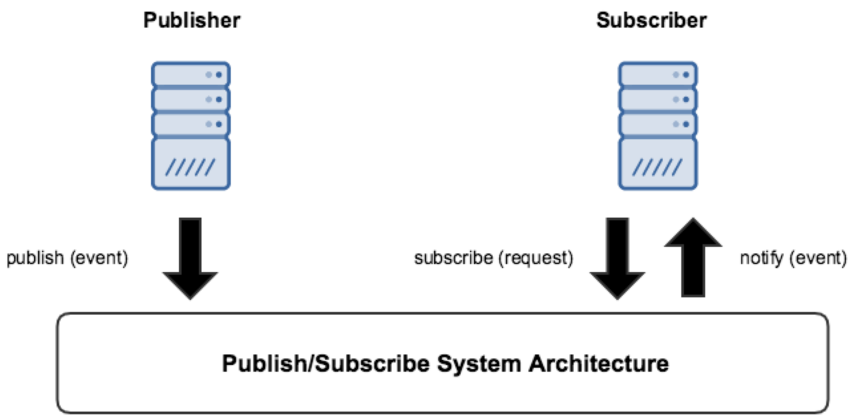

Figure 2.2: Pub/Sub Systems High-Level Interaction Model

One of the metrics that evaluates the power of a Pub/Sub system is the level of filters expressiveness it offers to subscribers. The word expressiveness here is used to measure how the used format for defining subscribers requests is capable of delivering an accurate description about subscribers interests. The following subsections discuss three Pub/Sub models that offer a different level of expressiveness for filters. For better understanding, this section uses a running example.

Topic-based Model

The earliest model of Pub/Sub systems is the topic-based one. In the topic-based model, the event space is subdivided into topics, relying only on strings as key. In other words, events are grouped based on a specific topic they represent, so when an event is received by the Pub/Sub system, it is added to a group based on its topic. Subscribers express their interests by sending a formatted string that specifies a topic or a group of topics. In such a representation, each topic corresponds to a logical event channel, ideally, connecting each possible publisher to all interested subscribers. So when a subscriber sends a request for a topic, it is get linked to the channel corresponding to this topic. That is, there exists a static communication channel between an event topic and all its interested subscribers. Hence, when an event is published, it is not needed to calculate all paths from the publisher to the interested subscribers.

The static nature of topics enables efficient implementations by exploiting known multicast primitives for topic-channels creation, however, filtering events based on the topic may lead to the subscribers need to further filter received events. This is because some subscribers may be only interested in a sub-category of events of a specific topic, however, they are enforced to receive all events related to this topic. For more precise understanding, consider a topic-based application offers *stock market* as one of the topics it disseminates events about. This topic has two sub-definitions: *stock quotes* and *stock requests*. Quotes are characterized with company *name* and *price*. One of the subscribers, assume S, is interested in receiving *stock quotes* notification if the *name* of the company is "JEBRA" when the *price* ≤ 120. As filters can be defined only using the topic name as a key, this subscriber will subscribe to the group *stock market*, and hence enforced to receive all events that belong to this group.

Based on the previous example, the topic-based model is found to be poor if evaluated with respect to filters expressiveness. This results in wasted bandwidth usage as many events are being routed to subscribers, yet, they may not be interested in receiving those events. This forms the main limitation of topic-based Pub/Sub systems.

Type-based Model

As an enhancement model for the topic -based one, type-based Pub/Sub systems [11] may be thought of as topic-based with additional sub-topics offered for subscribers. So instead of subscribing to a topic, subscribers are allowed to outline a sub-topic which is the event *type*. In such models, the event space is divided based on the event type. Events are usually referred as *objects*, and any published object must be an instance of application-defined types.

Following the same *stock market* example, now the event space can be more subdivided into *stock quotes* and *stock requests*, and subscriber S, has the ability to show its interest in receiving events about *stock quotes* as defined type, instead of receiving all events related to *stock market*. Although this model offers more expressive filters when compared with the topic-based model, still, subscriber S cannot express the interest in a certain *price* value, and may still receive events out of interest.

Content-based Model

An increased level of expressiveness is obtained with the content-based Pub/Sub model [5], in which subscribers use custom filters about the content of the events they wish to receive instead of a topic or type. Removing the limitations of statically defined distinct topics/types gives the flexibility to subscribers to express their interests accurately. This is achieved by representing publishers events with a set of attributes known to subscribers. Subscribers express their interests by specifying values for these attributes. In such a model, filters are seen as a conjunction of events attributes values. Based on the fact that the Pub/Sub system has no prior knowledge about publishers future events, paths from publishers to subscribers are being calculated during runtime upon receiving an event. A network link is established for forwarding an event to a subscriber if this event matches one of the subscribers requests.

Getting back to the *stock market* example, for *stock quotes* type, the event space is further subdivided based on the company *name* and *price*. And finally subscriber S can subscribe to company "JEBRA" when the $price \leq 120$. Although this model provides the highest level of filters expressiveness, and hence, yields best bandwidth usage results, it comes with a trade-off between the end-to-end delay. Expressiveness has an impact on the overall system performance such that the more filters are expressive, more required resources for matching events, and hence more increase in the end-to-end delay. This challenge is investigated as part of this thesis work.

2.2 Related Work

This section specifically deals with real implementations for Pub/Sub systems. As there is a large number of implemented Pub/Sub systems that can be found in literature, it was not possible to fully cover all of them. Five popular systems are presented here that relates to the general proposed framework.

2.2.1 SIENA

One of the earliest implementations of distributed content-based Pub/Sub system is Scalable Internet Event Notification Architecture (SIENA). SIENA [7] is known to be a generic content-based Pub/Sub system that was primarily implemented to maximize expressiveness and enhance scalability as wide area event notification service.

SIENA discusses three different distributed server architectures for the event notification service. It assumes that these architectures are implemented on top of a lower level network infrastructure:

1. *Hierarchical Client/Server Topology*: As the name implies, each pair of servers interacts with each other in an asymmetric client/server fashion. In such a topology, any server can have an unlimited number of incoming connections from other *client* servers, but the outgoing connection is only limited to one, which its *master*. This topology is known to be a centralized one because it has only one root server, which is the one that has no master. This topology suffers from critical failure problems because each server in this topology acts as a single point of failure; failure in one server will cause the failure of all connected servers is the lower network level. Moreover, due to its hierarchical organization, another

drawback is the increased load on servers high in the hierarchy which limits scalability capabilities of this topology.

2. *Acyclic Peer-to-Peer Topology*: It interconnects event brokers in a way that they shape an acyclic undirected graph with no redundant routes. As it is built on the peer-to-peer concept, all servers have the ability to communicate with each other symmetrically. This topology has the same single-point-of-failure drawback of the hierarchical topology.

3. *Generic Peer-to-Peer Topology*: In this topology, the creation of cycles is allowed giving more than one path to the destination. These redundant connections give this topology an advantage when compared with the previous two topologies as it overcomes the problem of having single points of failure. But it should be taken into consideration that these redundant paths require the implementation of special algorithms to avoid the creation of cycles. So in order to tackle this, SIENA's content-based routing algorithm first executes a distance-vector protocol to create a spanning tree before routing events.

4. *Hybrid Architecture*: The hybrid architecture is offered for networks falls between local and wide area networks categories. It tries to compromise the benefits of the three basic architecture in a trade-off with the architecture complexity.

For the routing tasks, in the hierarchical client/server architecture, each server keeps a partially ordered set of all the subscriptions received by this server. When a server receives a new notification, it goes through all stored subscriptions and performs the matching operation. After that, it sends a copy of this event to all interested subscribers. Interested subscribers information is stored with each subscription. In a peer-to-peer architecture, received events are treated in the same manner as in the hierarchical architecture. However, each server has to maintain additional information about the peers its connected to in order to be able to create the routing paths successfully. Sending a new notification towards servers that have interested subscribers attached to it is the main routing strategy adopted by SIENA.

In the context of content-based Pub/Sub systems, no one can deny that SIENA contributed to an evolution by providing a comprehensive solution. On the other hand, it did not pay much attention to system response time towards different received subscription requests. This comes as a consequence of the fact that it builds an overlay broker network over the existing underlying physical network infrastructure which results in extending the path between the end-points, and adds more time delay for the matching operations that occur at the application layer. This makes SIENA model not adequate for most today's demanding applications.

2.2.2 Hermes

Hermes [20] can be presented as a distributed, event-based middleware that utilizes peer-to-peer techniques to implement a scalable overlay network of event brokers for event distribution. It is built on the theory that building this network of event brokers on top of a peer-to-peer routing substrate enhances the properties of Pub/Sub systems.

A system that is built using Hermes middleware has two main components:

1. *Event Brokers*: Event brokers are those found at the overlay broker network and used to implement all the functionality of the middleware. *Local event broker* is that one responsible for maintaining connections between network end-points. Event brokers exchange four main messages types: (1) Type messages that are described below. (2) Publishers' advertisements. (3) Interested clients subscriptions. Finally, (4) events published by publishers.

2. *Event Clients*: Event clients are system publishers and subscribers. They first have to be linked to an event broker through a connection to be able to use the middleware services.

Hermes adopts a feature called *event typing*. This means that any published event must be an instance of a predefined event type that is characterized by a name and a list of attributes. So when a new event arrives, first it must be checked against available events types. For the routing logic, it supports two forms of routing: *type-based routing* and *type/attribute-based routing*. In type-based routing, all events classified under a certain type are forwarded to all interested clients that are subscribed to this event type. While in type/attribute-based routing, subscribers are allowed to filter on the list of attributes associated with each event.

When compared with SIENA presented in 2.2.1, experiments showed that Hermes has better performance in terms of routing efficiency. But again it has the inherited drawbacks of any broker-based Pub/Sub system with filtering operations happen at the application layer.

2.2.3 REBECA

Rebeca Event-Based Electronic Commerce Architecture (REBECA) [19] is a distributed object-oriented event notification system that adopts the content-based routing model. It discusses the potential of utilizing the Pub/Sub paradigm for large-scale event-based business applications. Its focus is directed towards the scalability of the routing algorithms as it was shown that the use of advanced routing algorithms has a significant advantage when compared with common approaches. Rebeca implements the Pub/Sub paradigm by a set of connected event brokers that form an acyclic graph. Each broker is responsible for managing a set of locally connected components that represent data producers and consumers. For the purpose of exchanging published events between connected brokers, each broker has to maintain the subscriptions of neighbor brokers and their directly-connected subscribers by storing them in a content-based routing table. Based on that, a published event can be forwarded stepwise originating at the broker that host the publisher over intermediary brokers to all brokers hosting interested subscribers.

One of the interesting characteristics of REBECA its feature-based modular design. The original design was built on top of a base system architecture that can be extended by adding more modules to the systems, each module representing a new feature. This offers the flexibility of customizing the system and enable a large degree of configuration freedom. However, it imposes new challenges on how to make the correct interaction between different modules. The newly imposed challenges led to the evolution of *plugins*. As presented by REBECA, each feature is implemented using a pluggable component that is called a plugin, this component can be inserted to a REBECA system. A plugin consists of two parts: a *broker engine* and a *connection sink*. Plugins are supported by event brokers to extend their features.

This new design allowed REBECA's developers to introduce new features for the basic Pub/Sub systems. The mandatory plugins offer the generic routing mechanism with different filtering based algorithms besides the traditional event flooding approach. One of the matching and routing features is *simple routing* which allows events to be filtered at the producers. Another routing feature is the *identity routing* that avoids the dissemination of an already forwarded subscription. It also supported *covering routing* that avoids forwarding subscription covered by another disseminated subscription. A way of reducing the size of filters tables kept by the brokers is the *merging routing*. It gives brokers the ability to merge existing subscriptions and forward only matched events to the generally merged subscriptions. A feature for structuring large-scale applications is *event scopes*. It is restricting the visibility of published events to only a subset of subscribers in the system. So when crossing scope boundaries, an event may be transformed according to an event mapping, in a similar manner to gateways in event federations. Although the use of plugins succeeded in extending the basic Pub/Sub systems features, reducing the end-to-end delay imposed by the use of brokers for filtering operations was out of focus in this model.

2.2.4 JEDI

The Java Event-Based Distributed Infrastructure (JEDI) [10] is a Java-based implementation of a distributed content-based Pub/Sub system. A JEDI system consists of two main components: *active points* that acts as publishers or subscribers, and *dispatching servers* that are responsible for routing the events from publishers to subscribers. Following the common approach, publishers publish events that are characterized with a name and an ordered set of event parameters values, and subscribers subscribe using an *event pattern* which is simple pattern matching language represents event parameters values.

The main feature of this system is the organization of event dispatchers in a tree structure where the routing is performed according to a hierarchical subscription strategy. Starting from an active point, subscriptions propagate upwards is the tree and its state is maintained by the dispatchers. Events also propagate in the same manner, but whenever a dispatcher received a matched event against a subscription, it forwards it through downward branches to be delivered to interested subscribers. The hierarchical tree structure of event dispatchers results in an increased processing overhead on dispatchers higher in the hierarchy, as the overhead increases by getting closer to the root node. This makes the system prone to the breakage of the tree into isolated regions caused by the overhead bottlenecks.

Based on the information presented in [9], the system has been extended to support clients mobility. This is supported by providing event dispatchers by the new *moveIn* and *moveOut* operations. A client uses moveIn operation if he wants to reconnect from a different location inside the network. Similarly, moveOut is used for the disconnection. When a client connects to a new server it contacts the old server for obtaining this subscriber past subscriptions. This gives clients the flexibility to connect to new dispatching server while all notification are being kept. Although this feature may be useful for many real-life applications, it imposes new challenges to the system. The back-and-forth migration of clients from one server to another cause load changes inside dispatchers tree and this requires the deployment of advanced dynamic load balancing algorithms.

2.2.5 BlueDove

BlueDove [18] is an attribute-based Pub/Sub system. The main aim of this project is to tackle the problem of scalability when Pub/Sub systems are used as cloud services. Attribute-based Pub/Sub systems are considered to be one kind of the content-based systems. Messages in topic-based systems are associated with strings that express the topic of the message. While in attribute-based systems messages are provided with various attributes that serve the aim of describing the message content, which makes it more expressive when compared to topic-based. Content-based systems are considered to be more general as they allow subscriptions to be in the form of arbitrary boolean functions not limited only to attributes. Attribute-based systems try to achieve the balance between the expressiveness level of content-based systems and the simplicity of the topic-based ones.

BlueDove supports the Pub/Sub paradigm by depending on a two-tier architecture built on servers:

- *Dispatchers*: Front-end servers that are open to the internet. Publishers and subscribers can connect directly to the dispatchers using their publicized IPs. Those servers are used to send different received messages to the servers at the back-end.
- *Matchers*: Matching servers found at the back-end that is used to match incoming events against received events. After matching operations are performed, messages are routed to subscribers using those matchers.

This architecture is said to be suitable for cloud services as the events are matched and delivered to the subscribers using only one matcher. Even though this significantly increases system responsiveness when compared with those systems that use an overlay network of brokers, in practice the matcher is similar to the concept of implementing brokers, and hence, the model suffers from the same inherited limitations.

2.3 Conclusion

This chapter highlighted SDN technology and Pub/Sub systems by providing their general specifications. In order to provide a solid background, a number of real implementations that relates by a way or another to this thesis work were discussed. Even though many solutions exploited SDN to provide more efficient Pub/Sub systems implementations, no promising solution can be found in the literature that covers the real-world applications requirements represented in the reduced end-to-end delay while keeping the efficient use of the available network bandwidth.

Chapter 3

PLEROMA: SDN-based Publish/Subscribe Middleware

The building block of this thesis work is the middleware presented in this chapter. PLEROMA integrates SDN technology with the traditional Pub/Sub communication model offering content-based routing. It exploits network layer elements capabilities for performing the content filtering operations. Model specifications start with the implemented Pub/Sub paradigm and the system structure. Other topics of concern are discussed in subsequent sections including how the content space is represented, matching and forwarding incoming events from publishers. The problem that the thesis tries to tackle is stated by the end of this chapter after discussing some design issues related to the presented model.

3.1 Publish/Subscribe Paradigm

Like most Pub/Sub systems implementations, the system primarily consists of a number of hosts that may act as *publishers* or *subscribers*. Hosts are directly connected to OpenFlow-compliant switches. Publishers and subscribers are decoupled and they have no information about each other existence. The link between them exists through the *controller* based on the following scenario:

1. Before publishing events, publishers first send *advertisements* to the controller about the content they intend to publish.

2. Subscribers send *subscriptions* requests to the controller showing their interest in receiving event notifications about specific published content.

3. Based on received advertisements from publishers and subscribers requests, the controller creates network paths between them as it has an end-to-end logical view of the whole underlying network. This is achieved by the communication of the controller with the switches using the standard protocol OpenFlow to install content filters in their match tables as shown in figure 3.1.

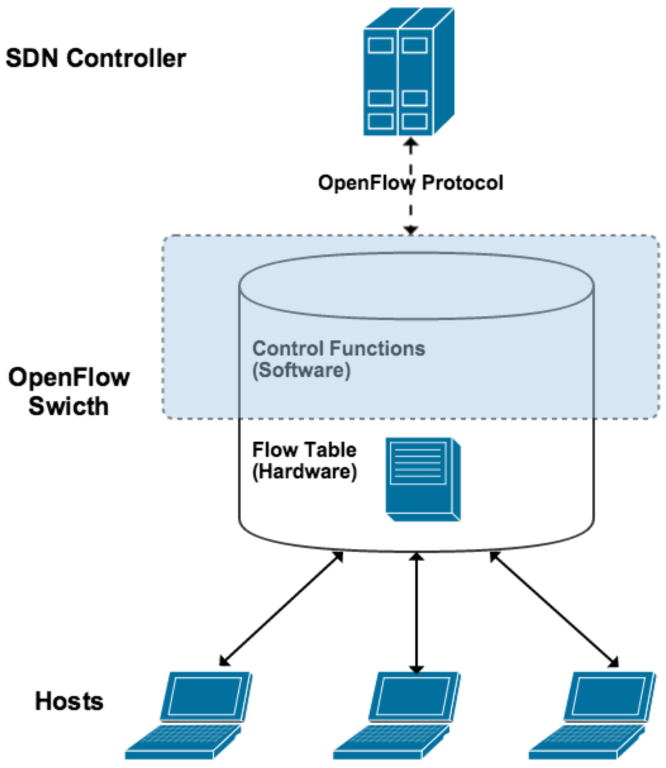

Figure 3.1: SDN-Compliant Switch Structure

The previous scenario incorporates the pre-mentioned two planes: *control plane* and *data plane*. The control plane consists of one or more SDN controllers that are responsible for the routing operations by creating paths between publishers and subscribers. The controller can receive two types of requests from subscribers namely *subscribe* and *un-subscribe*, and it receives *advertise*, *un-advertise* requests, and the *events* from publishers. The data plane consists of the intermediate connected OpenFlow switches where filtering operations take place. The difference between traditional switches and SDN-compliant switches is visible in figure 3.1. It shows an OpenFlow switch where the control functions were transferred from the hardware to a software running on an SDN controller while the flow tables are still part of the hardware. The communication between an OpenFlow Switch and an SDN controller happens by exchanging OpenFlow messages through a secure channel between them. The basic structure of PLEROMA is shown in figure 3.2.

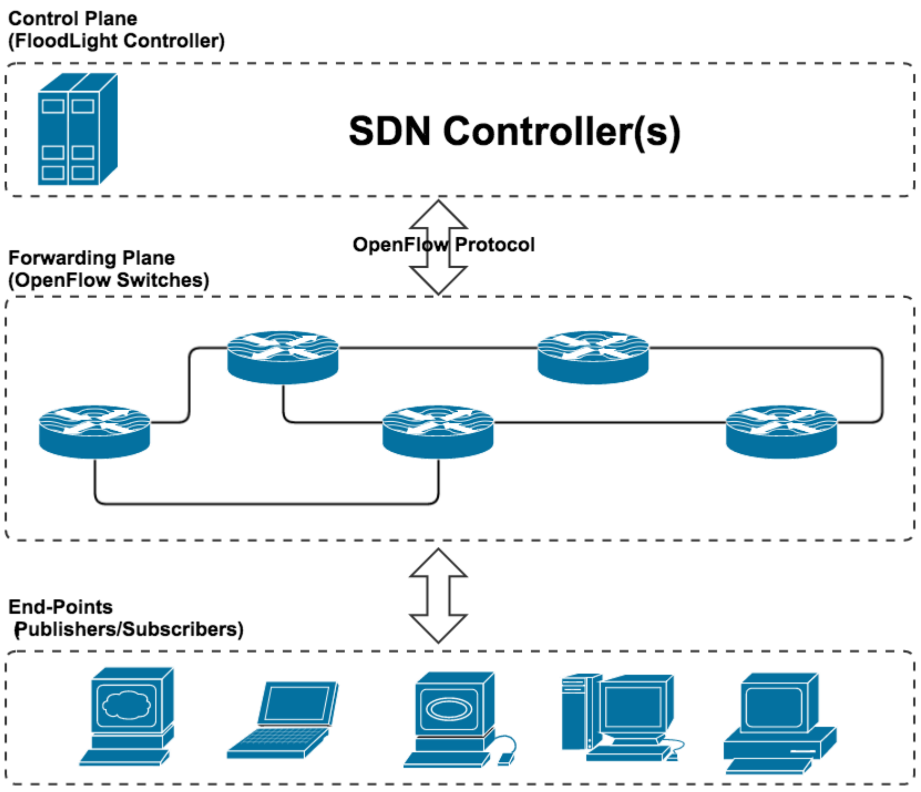

Figure 3.2: PLEROMA Basic Structure

3.2 Content Space Representation

The technology of SDN was primarily integrated with content-based Pub/Sub systems to switch filtering operations from the application layer to the network layer switches for better performance in terms of system responsiveness. Network switches are designed to forward incoming packets based on matching values from the packet header fields, possibly *IP-address*, *MAC address*, *port number* etc., with the installed forwarding rules. And as previously mentioned about the content-based model in section 2.1.2, events must be filtered based on the content of the message for accurate forwarding. So this imposes the question of how to make the network switches able to filter incoming packets based on the content instead of header fields values. To resolve this issue, message content must be embedded in the packet header in a format similar to this one used to represent the header fields, the binary string format, to make it understandable by the switches. PLEROMA uses the following described mechanism from [22] to reach such a representation.

First, the entire content space is represented as a multidimensional space where each dimension corresponds to attribute values. Following the concept of spatial indexing, the event space for each dimension gets partitioned recursively yielding regular subspaces, each assigned a binary number using recursive binary decomposition. Figure 3.3 assumes an example of a simplified event content space with only two attributes. As the content space includes two attributes, it can be represented as a 2-dimensional plane, each dimension represents an attribute, i.e., *Height* on the x-axis, and *Width* on the y-axis. As shown, the content space got partitioned three times resulting in eight subspaces, each represented by a binary number. The decomposed binary number now, known as *dz-expression*, can be used as an approximation for events, subscriptions, and advertisements. For example, a subscription request $s1$ for events with attributes values {Height=[50, 70], Width=[0, 100]} will be represented as $s1 = \{100, 110\}$, which is the binary representation of this area of the subspace. As described here, a subscription/advertisement may require more than one dz-expression for representation as it traverses multiple areas, however, it should be noted that an event always has one dz-expression representation as it is a well-defined point in the content space.

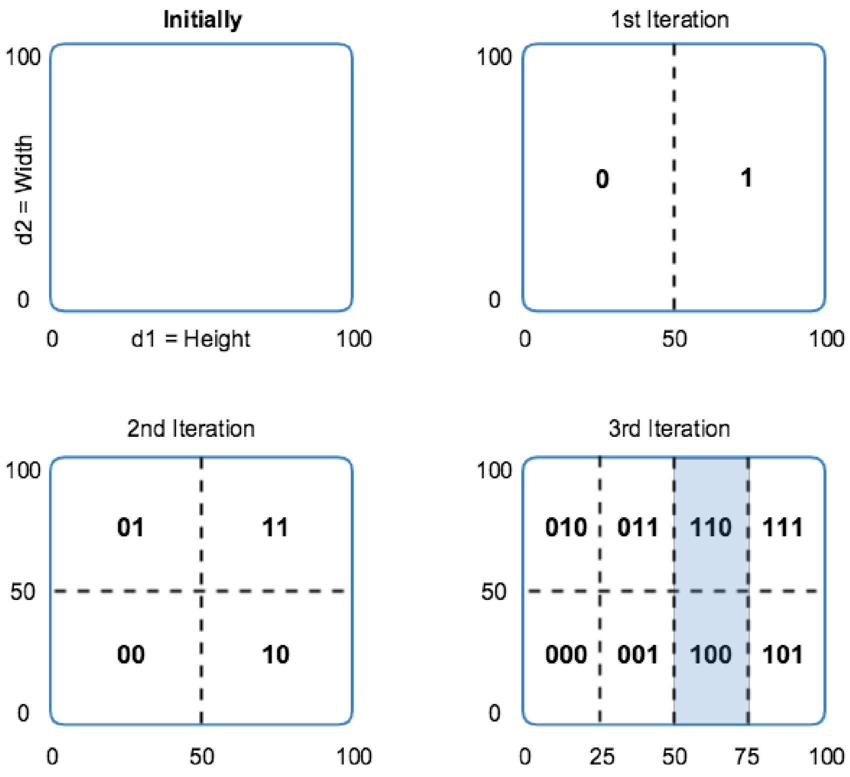

Figure 3.3: Spatial Indexing Example

3.2.1 Containment Relationship and Events Matching

In practice, there are many possible algorithms that can convert different data into binary strings like using a kind of hash functions or bloom filters [24], but the ease of identifying containment relationships between subscriptions is the main motive of using the previously discussed representation. A subscription *s0* is said to be contained by another subscription *s1* if and only if the dz-expression of *s1* is a prefix for the dz-expression of *s0*. For example, a subscription with dz-expression '0' covers the two dz-expressions '00' and '01'. This feature is very useful when installing flows to switches forwarding tables as it enables the installment of the most general filters. This significantly reduces the memory requirements for storing flow tables on switches knowing that it has limited storing capabilities.

By preserving containment relationships using such a representation, matching events has become an easy task. The matching is performed on a prefix basis, such that, an event matches a subscription if the dz-expression of the subscription is a prefix for the dz-expression of the event. If this condition is true, the subscription is said to cover the event and it is forwarded to the corresponding subscriber(s). In a typical scenario, the dz-expression is mapped to a packet header field, and this field gets mapped against installed flows on switches. If no match is found to an incoming packet, it is sent directly to the controller. PLEROMA embeds an event dz-expression as part of an IPv4-Multicast address and uses it as the match field, the scheme is detailed more in the next section.

3.2.2 IPv4 Adress Formulation

The middleware reserved a fixed range of IPv4-Multicast, 225.128.0.0 - 225.255.255.255 specifically for pub/sub traffic. Total of 32 bits are divided as follows: the first 9 fixed as the multicast address, leaving the remaining 23 bits for representing the content in the form of a dz-expression. So in order to form the IP destination address for a packet using its content, simply, the content is converted to a maximum of 23 bits in length dz-expression, and gets concatenated to the fixed multicast 9 bits address, and the length of the dz-expression is attached to the address as a subnet mask. By taking the example from figure 3.4, it is shown that the first 9-bits represent the fixed 225.128 part of the address, the next 23-bits are used for the dz-expression. In this example, it is assumed that this is the address of a message its content represented by the 3-bits dz-expression $\{010\}$. By padding the rest of available bits to '0' and convert it to the decimal IP representation, it corresponds to 225.160.0.0. Using the same methodology, another dz-expression of $\{0101\}$ would be converted to 225.168.0.0. If noted, the dz $\{0101\}$ is covered by $\{010\}$, and hence, any matched event to the address 225.160.0.0, is considered to be a match for 225.168.0.0 as well.

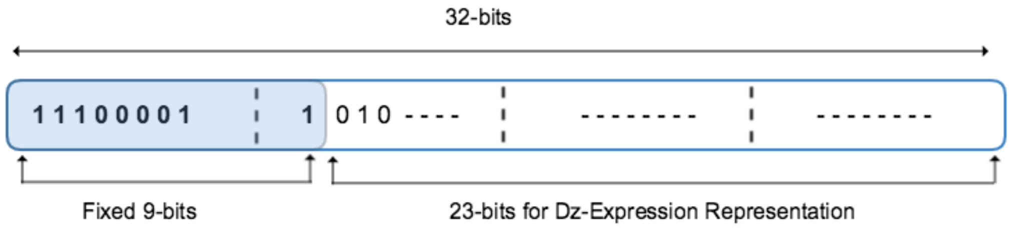

Figure 3.4: IPv4-Multicast Address Formulation

3.3 Content-Based Routing

As discussed in previous sections of this chapter, in order to perform matching operations by network OpenFlow switches, the content is mapped to an IPv4-Multicast address and used as the matching field. This section discusses how flows are being installed to switches, and the routing of events to interested subscribers after being matched against installed flows.

3.3.1 Publishers/Subscribers Spanning Trees

Forwarding events from publishers to subscribers is done via spanning trees. The controller, that is aware of the underlying network switches structure, organize those switches in a form of a *loop-free* spanning tree. Loop-free means that there is only one path from one switch to any of the switches, thus, it is ensured that no event is forwarded twice to the same subscriber. Figure 3.5 shows the spanning tree representation for an example switches structure, where $R2$ was chosen as root by the controller randomly. By considering real-world large-scale Pub/Sub systems, maintaining a single tree limits its scalability. To tackle this issue and provide a scalable solution, the controller maintains a set of spanning trees, such that each tree is responsible for the dissemination of events covered by a subset area of the event space. The number of spanning trees in a system depends on the number of subspaces created out from partitioning the content space. Recall the event space representation after the 2^{nd} iteration from figure 3.3, 4 spanning trees shall be maintained by the controller, each covers a subspace represented by its binary dz-expression. More precisely, all covered events by $\{00\}$ will be disseminated through its corresponding spanning tree.

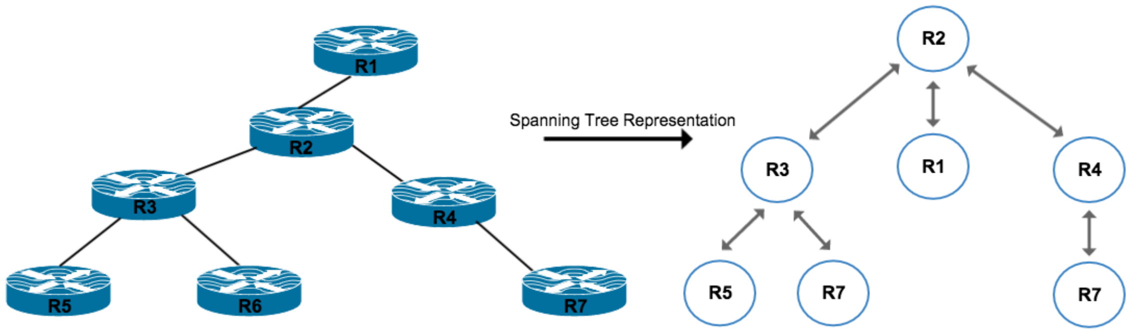

Figure 3.5: Loop-free Spanning Tree Representation

The creation of spanning trees is driven by received advertisements requests from publishers. An advertisement may trigger the creation of a new tree, or it may join an already existing tree. Assume that the tree from figure 3.5 was initially created after receiving an advertisement request from one of the publishers intending to publish events with dz $\{00\}$. Another publisher that advertises $\{000\}$ will join the same spanning tree. In a case of another advertisement that is not covered by any of the existing trees, a new spanning tree is created. Subscribers are added to the tree based on the same logic, however, a subscription request cannot trigger the creation of a new tree.

3.3.2 Publishers/Subscribers Paths Creation

Upon receiving a subscription (advertisement) request, the controller searches for all advertisements (subscriptions) requests that cover this subscription (advertisement). For every advertisement that covers a subscription, the controller creates a path as follows:

1. It starts by calculating the route from the publisher to the subscriber.
2. The route is defined as a set of physically interconnected switches from the publisher to the subscriber based on the corresponding spanning tree structure.
3. Once the route is defined, the path is created by installing, or possibly modifying flows of switches found on the defined route. Installing flows is explained in detail in the next section.

3.3.3 Flows Installment

As mentioned earlier, each switch maintains a flow table of forwarding rules that are used for matching operations. These flow tables are manipulated by the control plane controller. For example, when a subscription request is received, it installs new flow or modify a pre-installed flow. A flow on a switch consists of main three fields:

- *Match Field*: The match field defines the value of the header field that is used to match incoming packets. As previously described, PLEROMA middleware uses an IPv4-Multicast address for matching.
- *Instruction Set*: It represents the action that should be taken by a switch towards a matched incoming packet. For forwarding matched events, this set will be in the form of a list of outports on this switch that the event has to be forwarded to, in case it is a middle switch. Setting more than one outport enables forwarding events to multiple destinations. If it is a terminal switch, it changes the destination IP of the packet to the IP of the linked subscriber(s). If no match is found for an incoming event, it is sent directly to the controller for information about how to deal with this packet.
- *Priority Order*: A value that is used if a packet has more that one match. In such a scenario, the instruction set of the matched flow that has higher priority order will be followed.

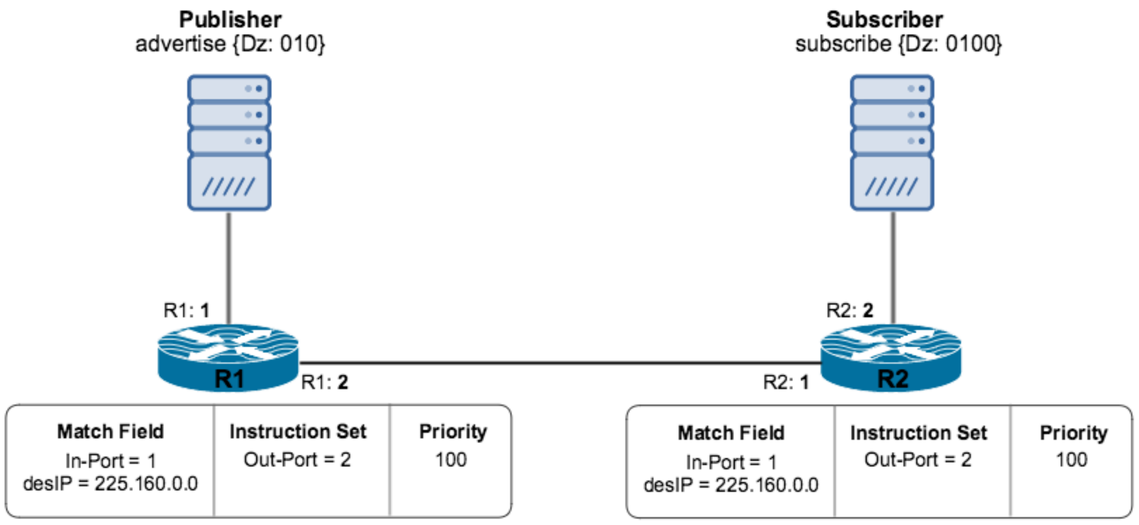

Figure 3.6: Flows Installment Scenario.1

Before the installment of a new flow, the controller first ensures that there is no containment relationship between this flow and the pre-installed flows. If a relation is found between the two flows, the controller only modifies the installed flow based on the type of relationship between the two flows. For clarifying, figure 3.6 shows a scenario where switch $R2$ is connected to a subscriber who sends a subscription request with dz $\{0100\}$. Another host acts as a publisher connected to switch $R1$ sent an advertisement request to the controller for content with dz $\{010\}$. The controller found that this advertisement covers the subscription request so a path should be created from the publisher to the subscriber for this content. As the controller is aware of the underlying switches topology, the route is defined as $\{R1 - R2\}$. Based on the defined path, the controller modifies flows tables of switches on that path. Assuming that the two switches initially have empty flow tables, the controller will install flows on each switch as shown.

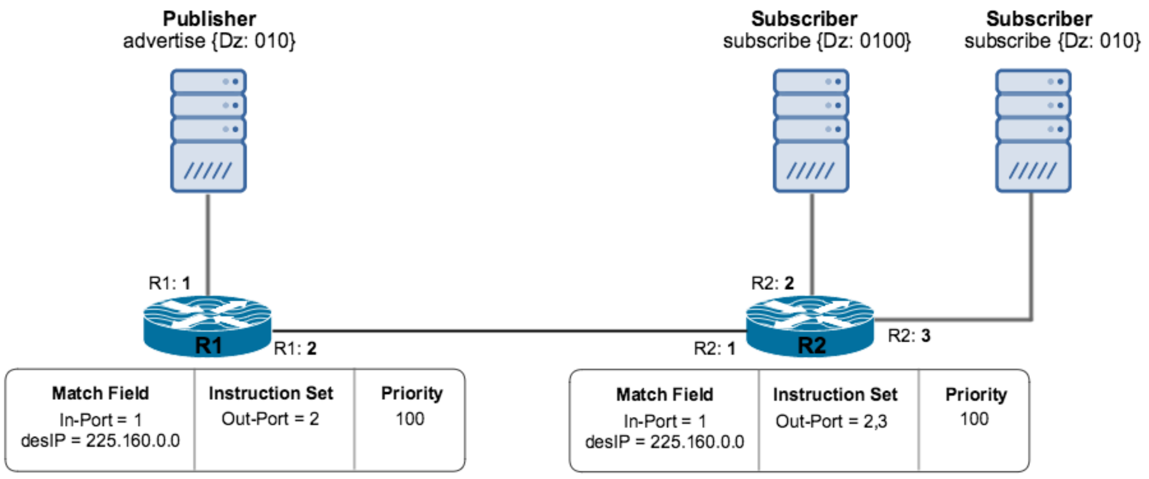

Figure 3.7: Flows Installment Scenario.2

Figure 3.7 shows the same example with another subscriber connected to switch $R2$ with dz $\{010\}$. As the installed flow on $R2$ covers this subscription, there is no need for the installment of a new flow. The controller just updates the installed flow to forward matched packets to this subscribers by including its connected port to the instruction set.

3.4 Problem Statement

The content-based model presented by PLEROMA takes advantage of SDN data and control planes separation to implement network layer filtering solution. The logically centralized controller installs the content filters directly to switches TCAM with the help of OpenFlow protocol. Incoming events get matched instantly against those filters. This described scenario guarantees line-rate performance and achieves high levels of system responsiveness, however, it has a downside. Network switches are limited-memory elements, this design issue affects the expressiveness of installed content filters. Content filters expressiveness is a crucial factor that affects available bandwidth usage. Reduced filters expressiveness results in more matched and forwarded events to subscribers through network physical paths, which requires more bandwidth.

In the context of content-based publish/subscribe systems, application layer filtering using brokers is characterized with advanced filtering capabilities that can highly utilize available bandwidth, but it fails when it comes to overall system responsiveness due to additional delay caused by the extended end-to-end path between publishers and subscribers with the existence of brokers, and the time needed for the matching operations at the application layer. On the contrary, network layer filtering in middlewares that make use of SDN, e.g. PLEROMA, achieve significant system responsiveness improvement as it offers line-rate performance due to the separation of data and control planes, but it suffers when compared with application layer filtering bandwidth efficiency. This thesis objective is to develop a middleware that combines the benefits of the two different filtering approaches. So the contributions of this thesis can be concluded in the design and implementation of a content-based publish/subscribe system exploiting the capabilities of SDN for providing hybrid filtering operations, and the analysis of the effect of such approach on system overall performance in terms of latency and bandwidth efficiency.

Chapter 4

Hybrid Content-based Publish/Subscribe Middleware

As briefly described in the introduction chapter 1, modern publish/subscribe systems pay a close attention to exploiting the SDN technology. In one implementation, like PLEROMA [21][6][16], matching (filtering) operations are performed using the processing capabilities of the TCAM on the network SDN-compliant switches. An event is said to be matched if a flow entry inside switch's flow tables is found to match this event. Although this scheme takes advantage of the pre-installed underlying hardware infrastructure without the need for installing specialized hardware components, it has its own downside. Before investigating this approach limitations three terms must be defined: *system responsiveness, filter expressiveness* and *false positive.*

System responsiveness is usually defined with the system capabilities of serving subscribers requests in terms of speed. So it can be evaluated based on the time between a subscription request is received and the successful delivery of matched events to subscribers. System responsiveness is a crucial metric while evaluating a publish/subscribe system performance, especially those employed for real-time applications. If it takes too long to deliver notifications to the subscribers, the data may become obsolete and hence the information is not reliable.

Filter expressiveness is the complexity level of used language by subscribers to express their interests. Expressiveness highly affects the bandwidth efficiency of a publish/subscribe system. The lower degree of expressiveness, the more matched events and forwarded, and hence increased network unnecessary traffic. In the context of publish/subscribe systems, unnecessary traffic is a notation of the events that being forwarded through network paths that do not match any of the received subscribers requests. Another performance evaluation metric that is affected by the used language is the number of false positives the system produces. A **false positive** is detected when a subscriber receives an event notification that does not match any of the subscriptions requests. Getting back to filtering operations on switches TCAM, filters expressiveness is restricted by the limited number of bits available for filter representation at the match fields of tables flows.

As a conclusion, it is investigated that network layer filtering adopted by SDN-based middlewares provides line-rate performance but they have poor bandwidth efficiency caused by the unnecessary traffic. On the other hand, application layer filtering introduces bandwidth efficient systems but suffers in the area of system responsiveness. As an attempt to provide as optimum solution, this thesis presents a hybrid solution between network and application layer filtering that combine the two approaches. Theory is simple but for implementation a question arise, what is the scheme of selecting events that will be filtered at the application layer. For the purpose of addressing this question, this chapter focuses on introducing the implemented two different selection algorithms.

4.1 System Model and Architecture

4.1.1 Data and Control Planes

System data and control planes are identical to the same planes in PLEROMA middleware presented in chapter 3. The data plane consists of interconnected SDN-compliant switches that are responsible for forwarding matched incoming events to interested subscribers. The control plane includes one or more SDN controllers that maintain network topology to create paths between publishers and subscribers. This is done by means of installing forwarding rules (content filters) on data plane switches TCAM. Once an event is received by one of those switches, it searches all installed filters to find a match. If a match is found, the actions in the *instruction set* field is performed. Packets that do not have a match are forwarded to the control plane controller.

4.1.2 Application Plane

The hybrid model involves an additional plane which is the *application plane*. It is used for exploiting application layer capabilities of performing events matching operations more accurately. Filtering events at the application layer significantly decreases the total number of false positives. This is because the difference between matching operations in the network layer and in the application layer is that matching incoming events in the application layer happen on the exact values of subscription requests and events instead of using their dz-expressions. Events being filtered by the application layer are selected using different selection algorithms that are investigated later in this chapter. The system structure is shown in figure 4.1.

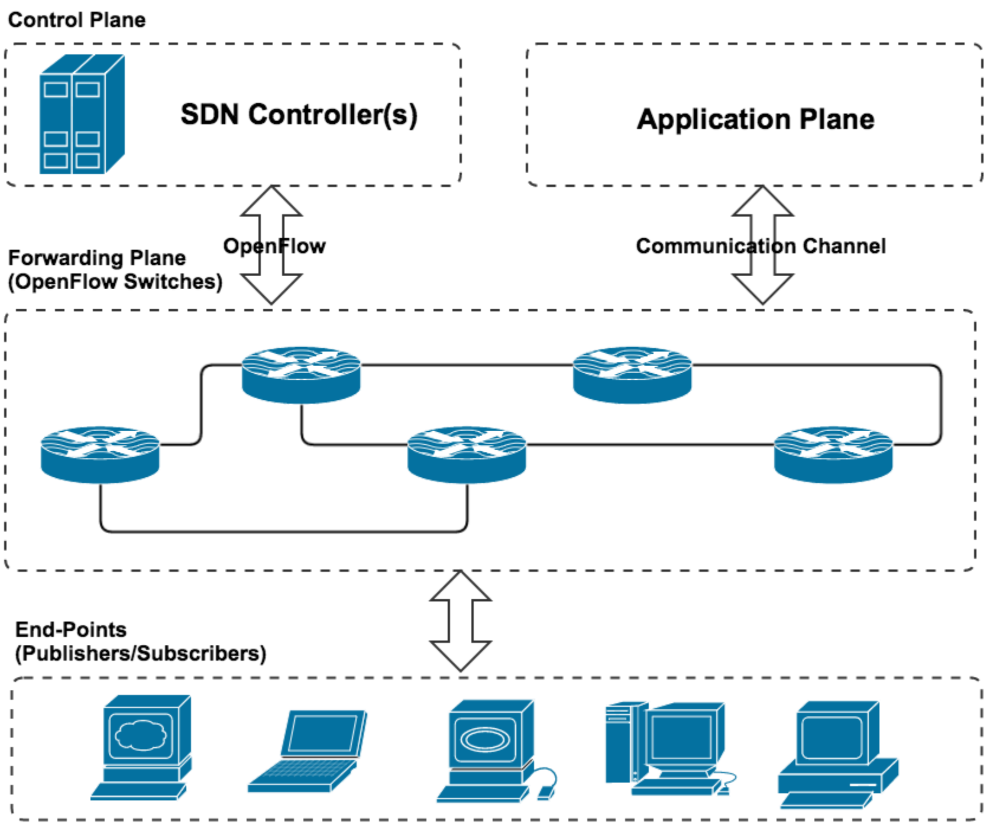

Figure 4.1: Hybrid Filtering Pub/Sub System Structure

4.2 Filter(s) Selection Problem Formulation

The main challenge of implementing the hybrid solution is selecting a subset of filters from all filters installed on all network switches, that forward matched events to the newly added application plane for more accurate filtering, and hence reduced false positives. The implemented mechanism tries to compromise between the responsiveness of systems that use network layer filtering while preserving the bandwidth efficiency of application layer filtering. This is achieved by giving the application the flexibility to inform the system its tolerance level for the end-to-end delay. As a constraint on the system, the delay must be less than or equal to this value specified by the application. This can be formed as an optimization problem by letting the following variables be:

F

All installed filters on all network switches, where filter $f_i \in F$

Δ

Average end-to-end delay threshold tolerated by the application

rfp_i
: Number of reduced false positive if filter f_i is chosen to forward matched events to the application layer

$\mathbb{S}$
: All subscribers set where $s_k \in \mathbb{S}$

δ_k
: Average end-to-end delay at subscriber s_k

So the objective now is to provide the system with a combination of selected filters $SF \in F$ that forward matched events to the application layer for achieving maximum false positives reduction, while maintaining the end-to-end latency by the given threshold Δ. This can be mathematically formulated as follows:

$$\mathit{Maximize} \quad \sum_{i \in SF} rfp_i$$

$$\mathit{subject\ to} \quad (\sum_{k=1}^{|\mathbb{S}|} \delta_k)/|\mathbb{S}| \leq \Delta$$

By considering an application with m total number of filters installed on n network switches, the number of all possible filters combinations is given by 2^m. In order to find the optimal solution, each filter inside each combination must be considered and evaluated separately. This is because selecting one filter has a significant impact on calculations for other filters in the same combination subset. Given that the value of m is usually in the order of hundreds of thousands, it makes achieving the optimal solution impractical and stands as an obstacle to implementing realistically scalable systems. After demonstrating the downside of achieving the optimal solution, the rest of this chapter discusses the proposed two selection algorithms for tackling filter(s) selection problem.

Based on the implementation experience, solving the selection problem was not found to be really straightforward as it introduces other three sub-problems. In order to come up with a solution, the following questions must first be investigated:

1. How many false positives will be saved upon selecting a specific filter?

2. How the selection of one filter affects the average end-to-end delay?

3. How to design efficient selection algorithm in terms of responsiveness and bandwidth efficiency?

Sections 4.2.1 and 4.2.2 discusses how the first two questions were addressed respectively, while 4.3 introduces the logic behind the implemented selection algorithms.

4.2.1 Filter Benefit Calculation

Before addressing the first question, the term filter *benefit* must be defined. It is the number of reduced false positives if this filter is selected to forward its matched events to the application layer. Calculating this value involves two sub-tasks: (i) obtaining the total number of false positives on each network link due to this filter, and (ii) aggregating all values obtained from (i) to get the benefit value upon selecting this filter. For calculating the number of false positives on each link, subscribers periodically send the number of received false positives corresponding to a specific filter to the controller. More formally, this means that the controller backtracks all network paths from subscribers to publishers. For the first thought, aggregating single links false positives may be perceived as trivial mathematical addition, but in fact, a close attention must be paid to containment relations between subscriptions before adding values up. Three forms of containment relations can be found between two (or more) subscriptions:

Disjoint Two subscriptions are said to be disjoint if there is no overlapping between their subspace rectangular area of interest. In this case, the aggregation over this link will be the sum of false positives of both subscribers.

Complete Overlap This occurs between two equal subscriptions, or when one of them is completely contained by the other. The simple addition of their number of false positives obviously will count more false positives over the link than the actual number. If they are equal, the total number of false positives over the link is the single value of any of them. If one is contained by the other, the false positives of the broader subscription must be taken as it accounts the false positives of the contained subscription.

Partial Overlap When two subscriptions areas of interest are partially overlapped, they are said to have partial overlap relationship. For calculating the total false positives of two partially overlapped subscriptions, the controller first must identify the overlapping area between them, and the two disjoint areas. And then it has to obtain the number of false positives for each area separately.

4.2.2 Filter Penalty Calculation

Filter delay *penalty* is the increased end-to-end latency caused by selecting this filter for application layer filtering. For better understanding, the difference between network delay and application delay should be highlighted here. If matching events of a specific filter is only bounded to the network layer controller, the end-to-end delay here is simply the network delay. But if filter f_i, as an example, is selected for application layer filtering, network delays along each path affected by f_i have to be replaced by the application layer delays. This extra delay comes as a result of the increased path length to reach the application layer, and the time it needs for accurately filtering forwarded events. The number of network paths between publishers along which it forwards events to subscribers is used as the measurement for the penalty. Based on the previous clarification, the penalty value is directly affected by the number of these paths.

By following the same approach, the value of application threshold Δ described in section 4.2, can be represented by, AP_{TH}, the maximum number of paths that the application layer is allowed to affect. This variable provides a good representation of the delay as when the number of affected paths increases the delay proportionally increases.

4.3 Selection Algorithms

This section discusses the design of two implemented algorithms for filters selection. After determining the effect of selecting filters on the end-to-end delay and the total number of the false positives, the algorithms try to keep the effect on the delay minimum, while maximizing the effect on the number of false positives. To give more flexibility to the application, the algorithms have different time complexity and performance in terms of the number of reduced false positives.

4.3.1 Switch Selection

Overview

After investigating why the optimal solution of selecting a combination of filters is not the most intelligent one for systems scalability, as a simplification, selecting a subset of filters can be replaced by selecting a subset of network switches that forward events to the application layer. But again, for a system with n switches, 2^n subsets must be evaluated for providing switches optimal solution. So it does not solve the complexity problem that selecting filters subsets imposes. The solution presented here tries to compromise between the optimal solution efficiency in terms of reduced false positives, and its time high complexity.

Algorithm Logic Design

SSA is an iterative process of selecting switches one-by-one, in such a way, a switch that achieves maximum false positives reduction gets selected in each iteration. The process stops when the specified application threshold is reached. The output of the algorithm is a selected subset of all network switches that will forward matched events to the application layer while maintaining the threshold value. For a more concrete presentation, let us consider the algorithm in terms of sequential steps:

1. **Switch Benefit and Penalty Calculation**: The algorithm starts with calculating benefit and penalty values for all network switches. For one switch, these values are simply the sum of benefits and penalties of all filters installed on this switch. Calculating single filters benefit and penalty is accomplished as described in section 4.2.1 and section 4.2.2 respectively.

2. **AP_{TH} Calculation**: As the system represents delays in terms of network paths, the value of AP_{TH}, which is the maximum allowed application delay in terms of paths, is calculated as described in section 4.2.2.

3. $\mathbb{R}$ **Set Definition**: At the first iteration, the algorithm considers all switches for selection by defining the $\mathbb{R}$ as the set of all network switches.

4. $\mathbb{R}$ **Set Filtration**: As the system must maintain the specified penalty threshold Δ, it performs a filtration operation for switches inside $\mathbb{R}$, such that, any switch that has penalty value greater than the threshold will be excluded from the set. If $\mathbb{R}$ is empty, the process terminates.

5. ***SR* Set Definition**: At the very first beginning, *SR* is defined as an empty set as no switches are selected yet.

6. **Switch Selection**: From $\mathbb{R}$, the switch that has maximum benefit is selected, i.e., added to *SR* and excluded from $\mathbb{R}$.

7. **Available Paths Calculation**: Upon selecting a switch, the penalty it imposes on the system is deducted from AP_{TH}. This gives an indication of the number of available paths that can be utilized for further selection, and hence false positives reduction. The algorithm stops when there is no available paths, or the number of available paths cannot be utilized, i.e., all switches in $\mathbb{R}$ has penalty greater than this value.

8. **Switch Benefit and Penalty Recalculation**: If the number of available paths calculated from the previous step allows more cycles, benefit and penalty values for all switches inside $\mathbb{R}$ must be recalculated before making a selection. This is because the selection of one switch may significantly affect the benefit and penalty of other switches.

   For more clarification, let us recall how benefit value is obtained for single filters from section 4.2.1. As it is described, calculating filter benefit is done by means of backtracking links from subscribers to publishers. In this context, assume that switch R_i is selected and added to *SR*, and one of its filters is f_i. Another switch $R_j \in \mathbb{R}$ having a filter f_j, such that f_i and f_j affects the same, one or more, network links based on a relation between the two filters subscriptions. In this scenario, if the algorithm just considered the old benefit and penalty values for f_j after selecting R_i, the same false positives reduced by f_i will again be considered by f_j. Also, the delay added by f_i affected paths, may again be counted in the delay penalty for f_j.

   As a mean of prevention mechanism, for all filters of a selected switch, after determining the benefit and penalty for f_i, all false positives for filters corresponding to f_i on subsequent switches along the downstream paths of f_i is set to zero, and these paths marked as already considered.

Algorithm Typical Scenario

Consider the simple example from figure 4.2, it includes two publishers labeled as $P1$ and $P2$, one subscriber S, and three switches act as the forwarding plane. Assume that S is originally subscribed to all published events by $P1$ and $P2$. Initially, benefit and penalty values are calculated per each switch $\in \mathbb{R}$. The values are shown, **B** represents benefit and **P** represents penalty. Assume that the average delay threshold Δ after being mapped to a number of available paths AP_{T} is found to be 2. Based on this value, switches will be filtered from $\mathbb{R}$. In this example, all switches will be considered as the penalty value is less than or equivalent to the threshold AP_{TH}. Out of all switches $\in \mathbb{R}$, $R2$ will be selected as it yields the maximum benefit while maintaining the threshold. The selection is in the form of removing it from $\mathbb{R}$ and adding it to the initially empty set SR. So by the end of this cycle $\mathbb{R} = \{R1, R3\}$ and $SR = \{R2\}$. After that the new values of benefit and penalty are calculated for all switches $\in \mathbb{R}$ and the value of AP_{TH} is updated to 0 after subtracting the selected switch penalty. If one of the available switches can yield more reduced false positives without affecting new paths it will be chosen, otherwise, the algorithm terminates and all events matched to filters installed in $R2$ will be forwarded to the application layer for accurate filtering. Note that even though all switches forward events to the application layer, the false positives on links between publishers and switches will remain inside the network.

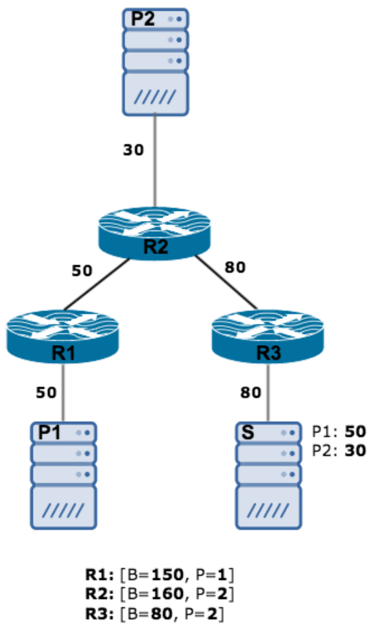

Figure 4.2: Switch Selection Algorithm Example

4.3.2 Cluster-based Selection

Overview

SSA was designed after showing the impracticality of single filters selection optimal solution due to its high level of complexity. However, implementing an intermediate solution between them may be a compromise between complexity level and performance in terms of network reduced false positives. Instead of selecting single filters, or switches, CSA proposes considering a group of filters based on similarities between the subscriptions they represent. So before the selection, all filters, on all switches, must undergo a grouping process in which they will be clustered into spatially disjoint trees. There are many filters clustering algorithms can be found in literature and in practice, but they are not presented here as it is out of this thesis scope.

Filters Clustering

By clustering filters in a spatially disjoint trees, each cluster $\mathbb{C}$ becomes responsible for forwarding disjoint set of events inside the network. This forms separate event dissemination trees in the network for each cluster. As a result, a matched event gets disseminated along only a single cluster's tree and thus affects the false positives count along the links of this tree and has no effect on other trees, so clusters are independent units in this scenario.

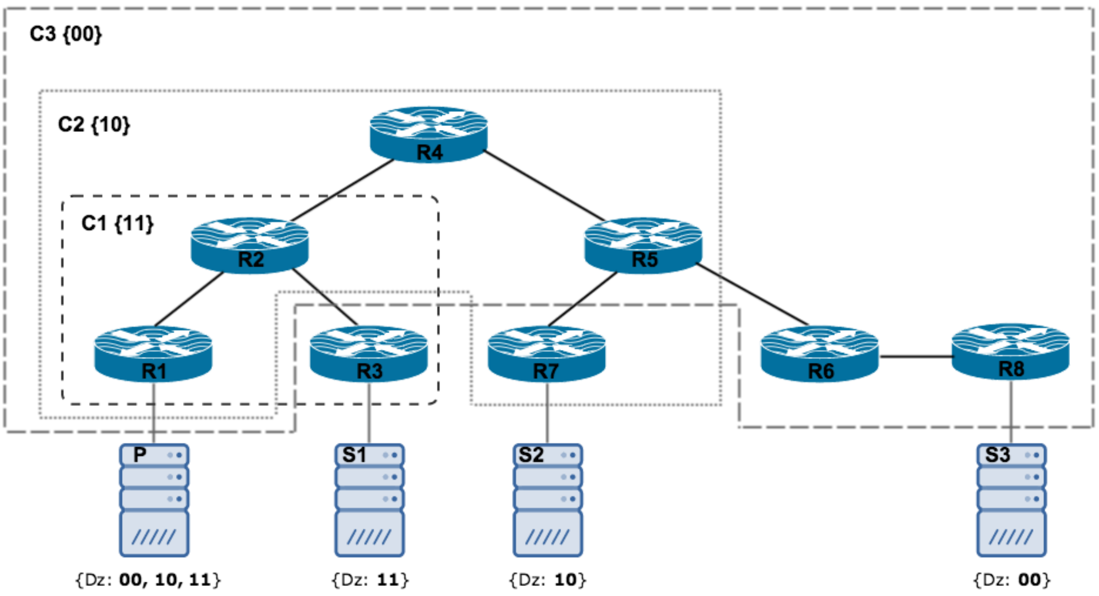

Figure 4.3: Logical Clusters Creation

Consider an example from figure 4.3 with 3 subscribers with subscriptions requests corresponding to dz-expressions $\{11\}$, $\{10\}$, and $\{00\}$ respectively. As there is no containment relationship between and of those subscriptions, 3 different clusters will be created as shown. The creation of these clusters will trigger the creation of 3 dissemination trees inside the network. For example, any published events match dz-expression $\{10\}$ will be forwarded through cluster $\mathbb{C}2$ dissemination tree that includes $\{R1, R2, R4, R5, R7\}$. Following the same approach, $\mathbb{C}1$ dissemination tree considers only $\{R1, R2, R3\}$. And finally $\mathbb{C}3$ filters matched events will traverse

$\{R1, R2, R4, R5, R6, R8\}$. If another subscriber sends a request with dz-expression $\{000\}$, the controller will recognize that this subscription is covered by $\{00\}$ and will automatically join the already-set dissemination tree corresponds to $\mathbb{C}3$.

Algorithm Logic Design

CSA uses the same calculations for benefit and penalty values, all other used parameters and variables are described below. Algorithm steps can be sequentially ordered as follows:

1. **Filters Clustering**: As previously mentioned, all filters on all switches are clustered in a spatially disjoint trees. Each cluster disseminates a set of disjoint events.

2. **$AP_{\mathbf{TH}}$ Calculation**: Again, this is the threshold Δ in terms of a number of affected paths. It should be noted that the number of total paths in the system now has increased. This is because there is a logical separate link between publishers and subscribers for each group of clustered filters.

3. ***RC* Set Definition**: This is the set of selected switches within each cluster after each iteration. Before the selection process starts, it is just an empty set.

4. **Per Cluster Switch Penalty and Benefit Calculation**: This is identical to calculating the benefits and penalties for switches in SSA algorithm. However, it is calculated for each group of filters that belongs to a cluster on one switch.

5. **Per Cluster Switch Selection**: Within each cluster, the switch that has the maximum benefit while maintaining the threshold is selected. The difference between SSA and CSA should be noted here. In SSA, selecting a switch means that matched events against all filters on this switch will be forwarded to the application layer. However, in CSA, by selecting a switch within a cluster, only matched events towards the group of filters that belong to this particular cluster on this switch will be forwarded to the application layer.

   The output of this step is a total of $|\mathbb{C}|$ selected switches from all clusters added to RC set. In a system with 3 filters clusters and m switches, if switches R_2, R_1, and R_2 selected respectively in clusters c_1,c_2, and c_3, RC is represented by the set $\{R_2\_c_1, R_1\_c_2, R_2\_c_3\}$. It should be noted that the same switch can be selected more than once inside different clusters as pairing one switch with a cluster gives a different set of filters when pairing the same switch with another cluster.

6. **Pairs Combination Benefit and Penalty Calculation**: The benefit of selecting n combined switch_cluster pairs from RC is the sum of single benefit values of each pair. The penalty is calculated by following the same scheme. Because the algorithm deals with disjoint clusters, it is not needed to recalculate benefit and penalty values for switch_filter pairs after one selection.

7. **Maximum Benefit Combination Selection**: The task now is to select the subset $SRC \in RC$ such that the combined reduction of network false positives is maximum while maintaining the average end-to-end delay within the specified threshold Δ. Out of

values calculated from the previous step, the best combination of switch_cluster pairs is chosen for application layer filtering.

8. **Available Paths Calculation**: Number of available paths is calculated as discussed in SSA, section 4.3.1. If there is a number of available paths that can be utilized, the entire cycle repeats.

4.4 Algorithms Complexity

By comparing the two algorithms logic design, CSA proves to have higher time complexity than SSA. This is because each cycle requires recalculating switches benefit and penalties values for all clusters $\in SRC$, as selecting a subset of filters on one switch affects these values. However, this increase in time complexity comes with more reduced false positives inside the network as the algorithm considers the selection of a group of filters on one switch instead of selecting switch all filters.

As it can be deduced, the complexity of the two implemented algorithms highly dependent on the number of switches in the underlying forwarding plane, as the search space size increases with increasing switches number. Based on this, reducing the number of switches in the search space can improve the algorithms time complexity without affecting its performance in terms of the number of reduced false positives. To achieve this, a scheme is developed to only consider a number of candidate switches out of the complete set while neglecting the others. Technically speaking, a switch is selected for application layer filtering if no other switch in the network reduces more false positives while maintaining the application threshold. In doing so, the proposed scheme tries to identify the switches that will save more false positives compared to the others.

4.5 Conclusion

This chapter was dedicated to the presentation of hybrid-filtering middleware solution for content-based Pub/Sub systems that exploit the technology of SDN. Recall that the development of this middleware was motivated by real-world applications need for a system that compromises between the system end-to-end delay and the bandwidth efficiency. As it was investigated, network-layer filtering provides the best results when it comes to the end-to-end delay, while application-layer filtering offers the most efficient solution in terms of the bandwidth usage. Theoretically speaking, a hybrid solution that combines both filtering approaches would achieve the desired results. In doing so, the proposed solution is built on PLEROMA presented in chapter 3 that follows the in-network filtering approach. An additional plane is integrated with PLEROMA system to perform the application layer tasks. To prove the credibility of the theory, the next chapter introduces the results out of different evaluations.

Chapter 5

Testing and Performance Evaluations

This chapter is dedicated to testing and analyzing the proposed hybrid-filtering solution performance while considering the other two solutions, i.e., pure network-layer and pure application-layer filtering. It also provides a technical comparison between the two implemented selection algorithms. Before presenting the results, conducted experiments setup is explained first, followed by the specifications of the used data sets for testing purposes.

5.1 Experimental Setup

As discussed previously in section 4.1, the hybrid system combines three different planes, i.e., control plane, data plane, and the application plane. Control plane is built on the FloodLight controller mentioned while introducing SDN technology in section 2.1.1. FloodLight implements SDN-controller functions using the famous Java API. For the data plane, it consists of a number of SDN-compliant switches, OpenFlow is used for the work of this thesis. Publishers/subscribers are normal hosts at the end-points. For the purpose of this chapter, the experiments have been evaluated under two different environments, i.e., hardware SDN-testbed, and mininet emulated network incorporating different data sets.

5.1.1 Hardware SDN-Testbed

In order to provide reliable results, a real-world hardware testbed was used for the purpose of testing. The setup is built on a number of commercial PC hardware and a SDN-compliant OpenFlow switch from Edge-Core. A hierarchical tree topology was created from 10 switches and 8 end-hosts connected as shown in figure 5.1. The used hardware white box switches from Edge-Core act as the data plane responsible for forwarding tasks from publishers to subscribers. While the connected end-hosts, that act as publishers\subscribers, are hosted on a PCs rack. Each machine in this rack has 16 cores working at 3.50 GHz. Another separate powerful machine was incorporated to perform both control and application planes controllers tasks. The used machine has 40 cores, each works at 3.10 GHz.

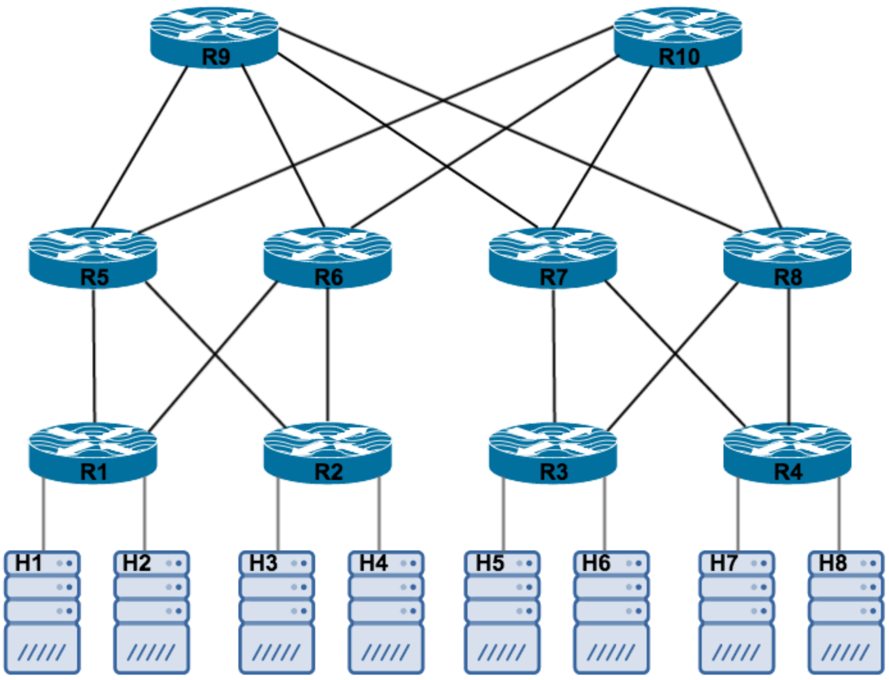

Figure 5.1: Hardware SDN-Testbed Topology

5.1.2 Mininet Emulated Network

Mininet [2] is a dominant tool among the available network emulation tools that support SDN technology. It supports the connection of external SDN-controllers to virtually-created programmable networks. This was very useful while working on the initial evaluations to verify the performance and behavior of the two used Floodlight controllers. Mininet was used to conduct experiments on different topologies that included up to 337 switches and 729 hosts. Although the use of mininet offers a prominent platform for testing tasks, the calculations of the time delay cannot be applied to real-world scenarios. Hence, all latency-related experiments were run using the hardware testbed described in the previous section.

5.1.3 Data Sets

Out of the keenness on introducing comprehensive evaluations, both synthetic data and real-world data were used to conduct the experiments. It should be noted that data here is a notation for *subscriptions*, *advertisements*, and *events*. The entire content-space was represented using 6 different dimensions, each dimension represents an attribute. All attributes domains vary in the range from the lower value 0 to the upper value 1023. For the generation of the synthetic data, two models were used, i.e., *uniform* and *zipfian* As it is implied by the name, the uniform model generates independent data forming a uniform distribution. The other Zipfian model generates dependent data around 8 specified hotspots. To reflect the performance of the hybrid proposed solution in real-world scenarios, stock quotes about daily closing prices procured from Yahoo Finance were used.

5.2 Experimental Results

The hybrid middleware was put into comparison with PLEROMA model described in chapter 3 that implements pure in-network filtering, and a pure application-filtering middleware that was implemented as a parallelized matching Pub/Sub service. In all covered scenarios, the content space was divided into 16 partitions, each partition has its own events matcher running on a separate core for better forwarding efficiency.

5.2.1 Total Network False Positives

The objective of the first experiment is to evaluate the performance of the hybrid-filtering solution in terms of the in-network generated false positives. Basically, network false positives is the sum of false positives on all network links. For running this experiment, an increasing number of subscriptions was used, while keeping the number of disseminated events from publishers constant to 10,000 events. By examining the graph from figure 5.2, it is shown that the hybrid solution saves a significant number of false positives over PLEROMA in-network filtering. These values were the result of using an application delay threshold factor of 0.6. The threshold value represents Δ, the average end-to-end delay that the application can tolerate. Based on this definition, using threshold factor of 1 is an indication for pure application filtering, and a factor of 0 implies pure network filtering.

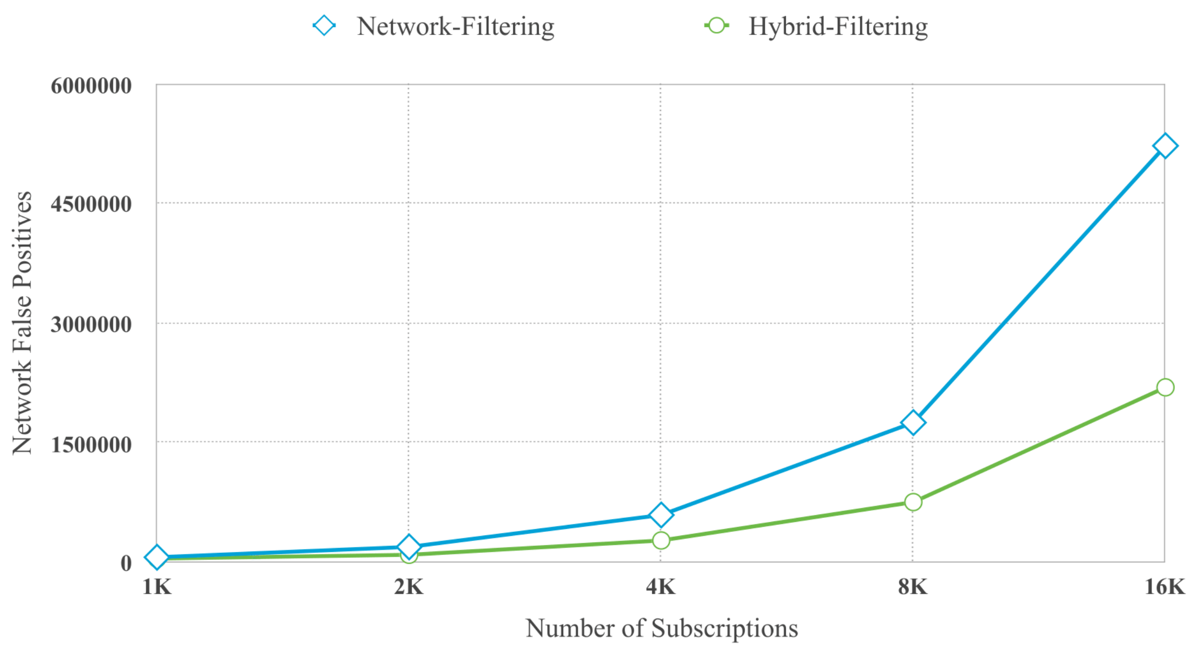

Figure 5.2: Hybrid Vs. PLEROMA False Positives

5.2.2 End-to-End Delay

Although the hybrid middleware is capable of improving the bandwidth efficiency by reducing the network false positive, this comes in a trade-off with the end-to-end latency. The delay increase happens as a consequence of shifting the matching operations for a subset of the events to the application layer. The graph presented in Figure 5.3 shows the effect of this shifting on the average end-to-end delay under 10,000 events get matched with a different number of subscriptions. As it can be deduced from the graph, PLEROMA proved that it operates at a line-rate performance with the minimum latency in the order of few microseconds. On the contrary, a pure application-filtering approach has the worst end-to-end delay which makes it not suitable for real-time applications. As a compromise, the hybrid-filtering approach shows less latency than the application layer, but greater than the network layer. As both the hybrid solution and the application-layer solution involves the use of software for matching incoming events, the delay increase with increasing the number of subscriptions because more incoming events have to be matched by the software.

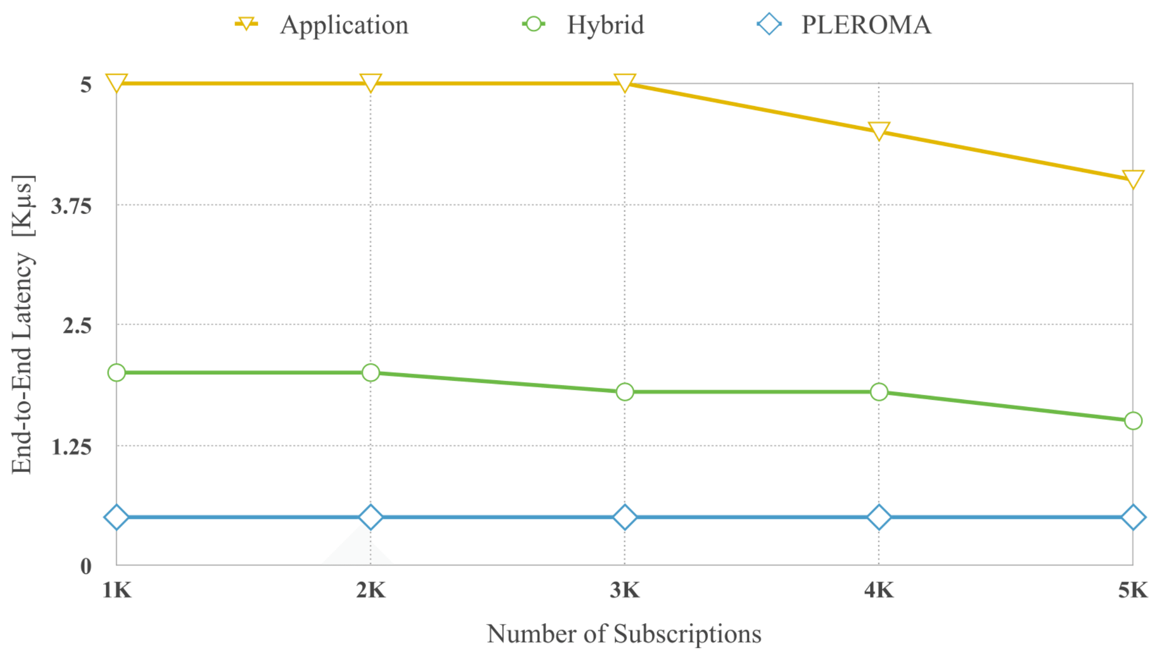

Figure 5.3: End-to-End Delay Comparison

5.2.3 Hybrid-Middleware Performance Adjustment

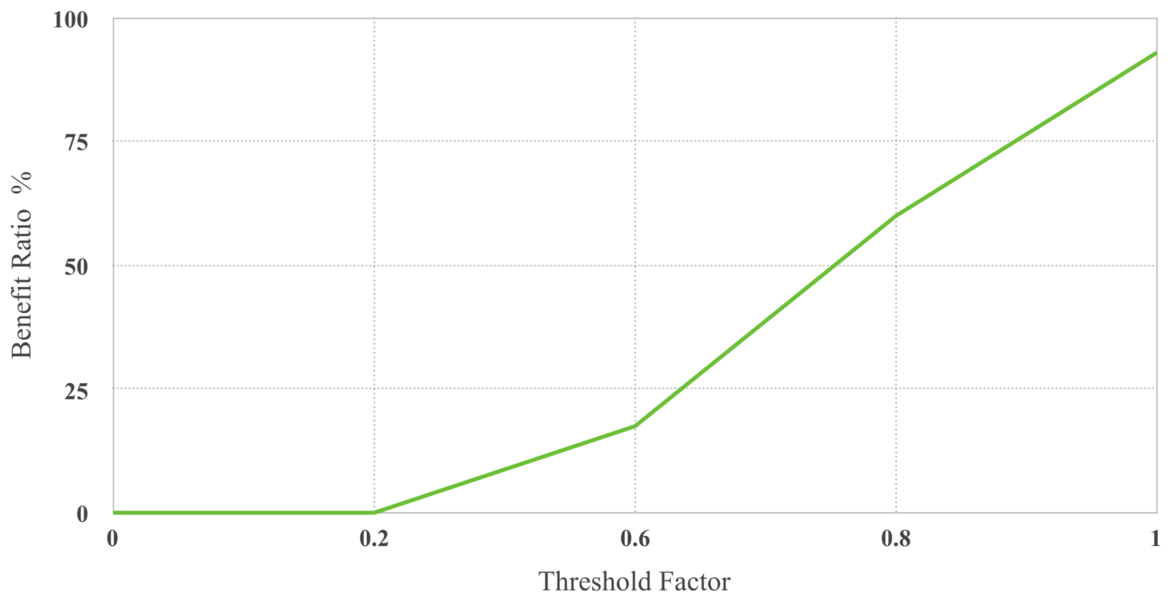

Figure 5.4: The Effect of Threshold Factor on Bandwidth

As described in section 5.2.1, the value of the allowed application delay Δ is represented as a factor from 0 to 1. A threshold factor of 1 means that it is allowed to forward all events to be matched by the application layer resulting in pure application-layer filtering while 0 means that the application cannot stand any extra end-to-end delay and no events will be forwarded to the application layer. In this case, the system has the same performance of PLEROMA in-network filtering. Consequently, adjusting the value of the threshold factor is a mean of regulating the hybrid middleware performance giving more flexibility to the application. The effect of adjusting the threshold value was tested in a system with 8000 subscriptions while gradually increasing the threshold from 0 to 1 with a step of 0.1. In doing so, figure 5.4 and figure 5.5 shows this effect on the bandwidth efficiency and the end-to-end delay respectively. The effect on the bandwidth efficiency is represented on the y-axis as the *benefit ratio*. It is the percentage of reduced false positives with respect to the total network false positives generated out from using pure network filtering. As it can be noticed, the benefit ratio increases with increasing the threshold factor. On the other hand, the average end-to-end delay increases.

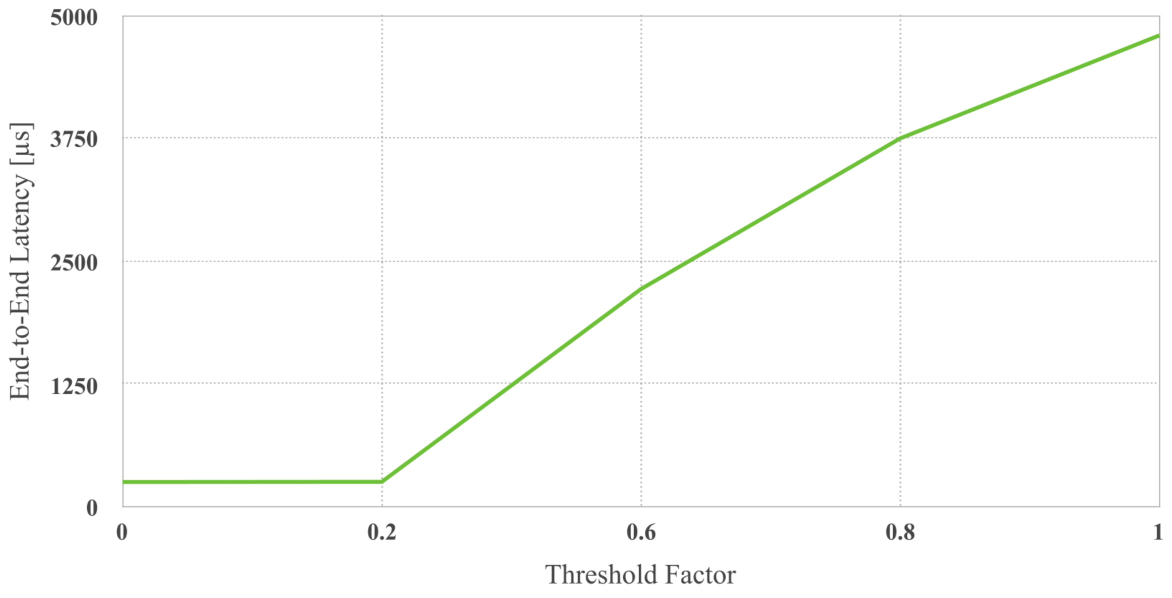

Figure 5.5: The Effect of Threshold Factor on Latency

5.2.4 Hybrid-Middleware Vs. PLEROMA-v Model

As a mean of improving the bandwidth efficiency, PLEROMA-v is a variant model of PLEROMA in which similar subscriptions are clustered in trees before any filtering operations. PLEROMA-v follows the R-Tree [14] clustering approach for the creation of subscription clusters. Subscriptions are clustered such that each cluster is represented with a Minimum Bounding Rectangle (MBR). As the name implies, cluster MBR is the minimum values along each attribute that encloses this cluster. Thesis contributions in this area are the implementation of PLEROMA-v on SDN, and evaluating its performance. To compare between PLEROMA-v and the hybrid middleware, 8 subscriptions clusters were created out from synthetically generated data based on the Zipfian distribution. The content space was represented using 5 dimensions, and the values were generated around 8 hotspots. Figure 5.6 shows that with threshold factor 0.4 the hybrid solution achieves higher benefit than PLEROMA-v. Also, the experiments showed that the hybrid middleware has better performance when uniform data was used. This is because the independence of the data set made the MBR of each cluster roughly covered the 5 attributes entire range. As PLEROMA-v still performs in-network filtering, it has better latency compared with the hybrid-filtering.

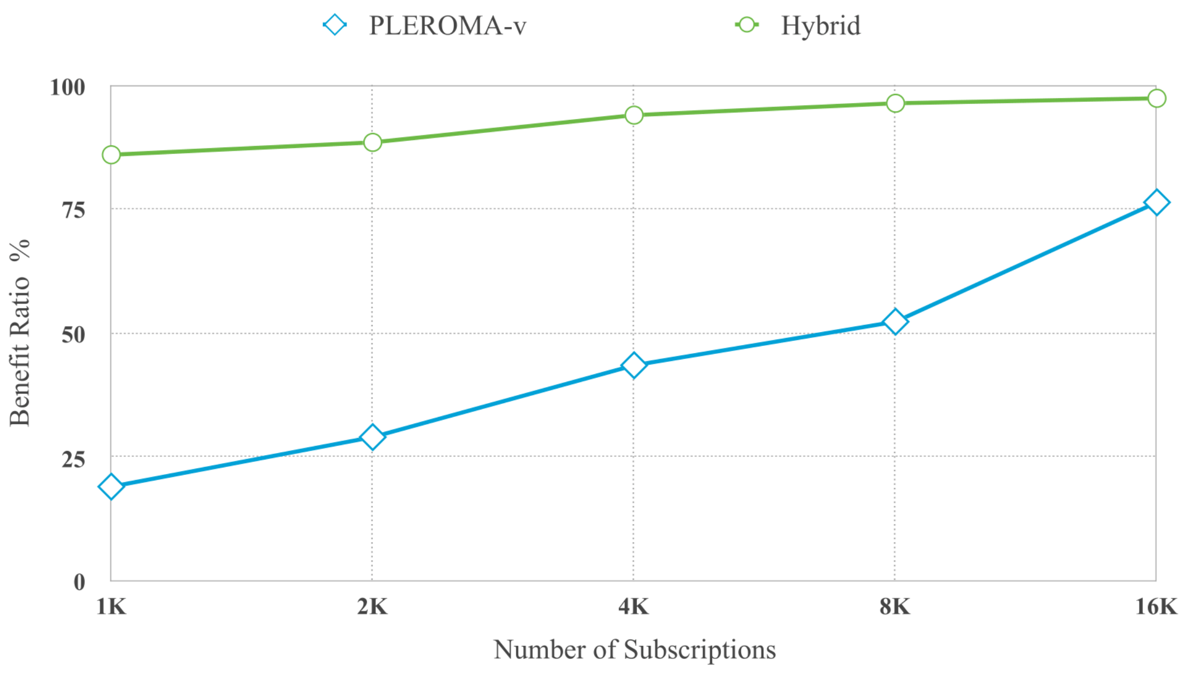

Figure 5.6: Hybrid Vs. PLEROMA-v Benefit

5.2.5 SSA Vs. CSA

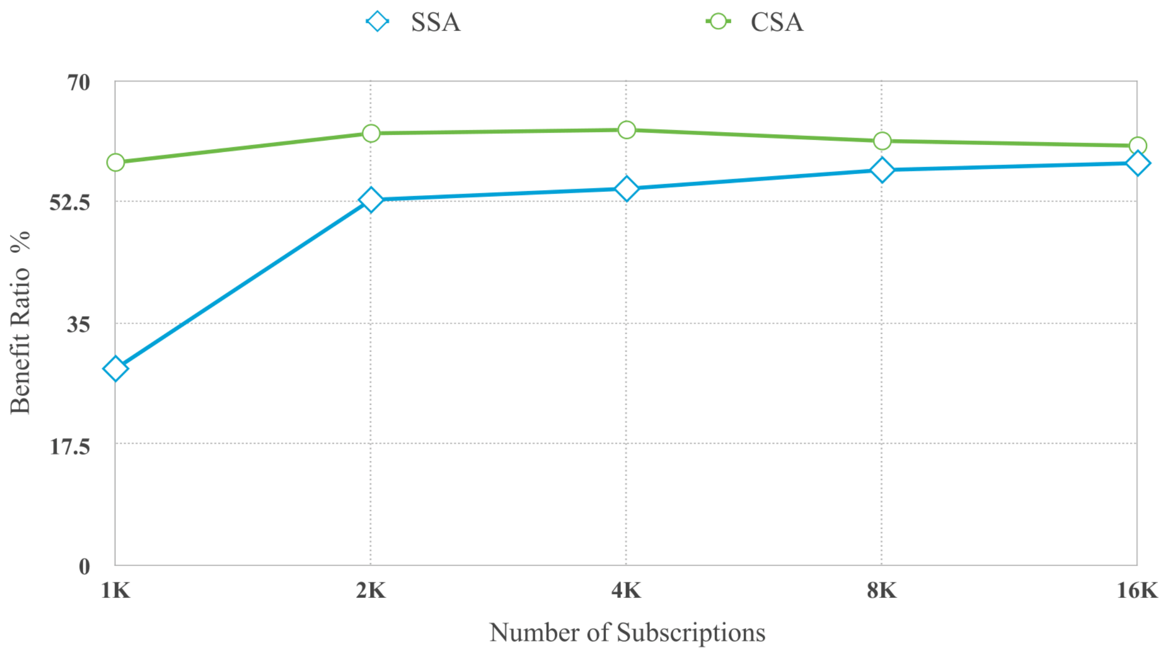

Figure 5.7: SSA Vs. CSA Benefit

The two proposed selection algorithms are evaluated in this section. As a result of the flexibility that CSA provides by examining a group of filters on one switch instead of the selection of all filters, figure 5.7 shows that CSA achieves better false positives reduction with increasing the total number of subscriptions, compared with SSA using the same number of subscriptions. Figure 5.7 proves that CSA has the same behavior in comparison with SSA for increasing the threshold factor gradually. By recalling from section 4.3.2 that CSA uses separate dissemination tree for each created cluster of filters, the number of those filters significantly affects its performance. Figure 5.9 depicts that increasing the number of clusters improves the algorithm performance. Figure 5.10 shows that this improvement comes in a trade-off the time complexity. Increasing subscriptions number increases the runtime for both algorithms. It is shown that CSA has a higher runtime that SSA for the same number of subscriptions. This is because each iteration considers all switch-filters combinations for clusters, compared with SSA that only considers switches.

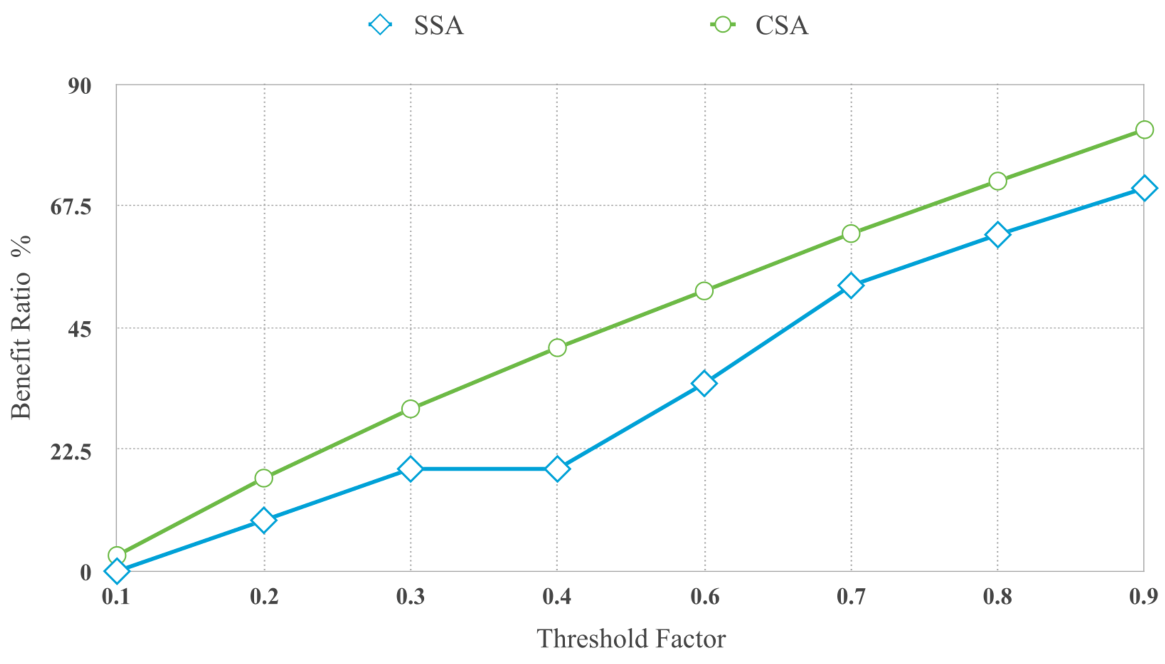

Figure 5.8: Adjusting Threshold Effect on SSA and CSA

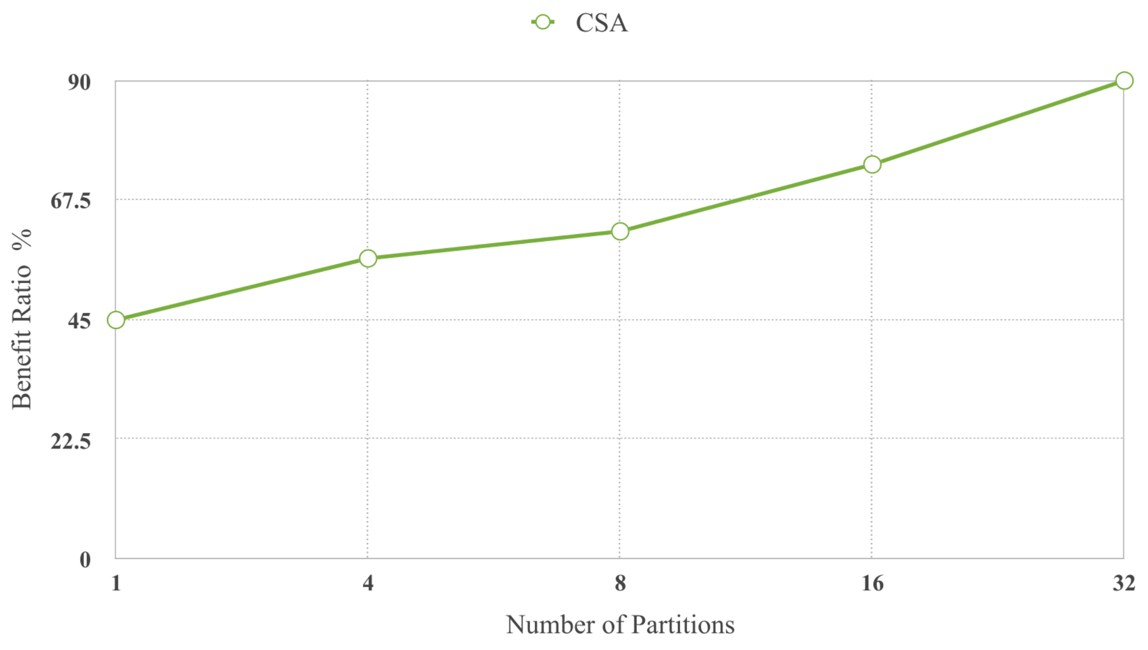

Figure 5.9: Number of Partitions Effect on CSA

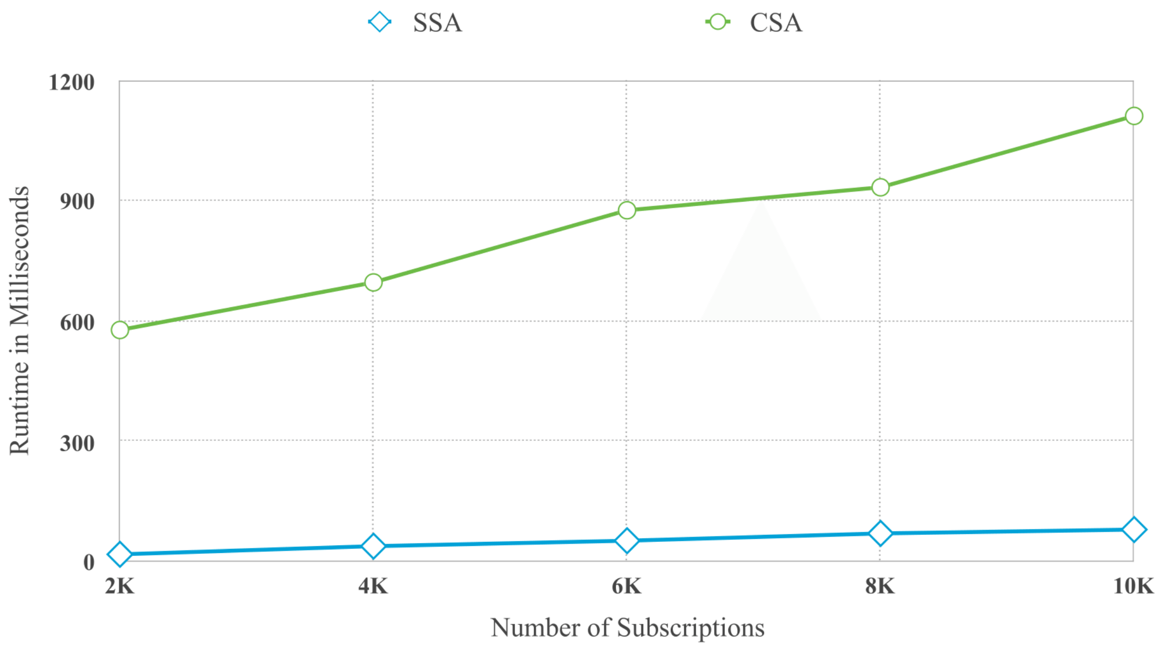

Figure 5.10: SSA Vs. CSA Runtime

Chapter 6

Conclusion & Future Work

This thesis work proposed a hybrid content-based Pub/Sub system built on top of PLEROMA [21] that exploits the evolving SDN technology. The idea of developing a hybrid-filtering system was motivated by the inherited problem of wasting the network bandwidth when in-network layer filtering is deployed, and the increased end-to-end delay if filtering operations are performed in the application layer. Real-world applications need for systems that can operate with a high level of bandwidth efficiency while keeping minimalist delay has led to the development of the proposed solution.

Theoretically speaking, incorporating filtering operations in both network and application layer can provide a compromise between the efficient bandwidth usage when application layer is involved, and line-rate performance of the systems that use in-network layer filtering like PLEROMA. In order to make a coherent picture about the theory, events that cause high rates of false positives can be forwarded to the application layer for accurate filtering to reduce bandwidth requirements, and the rest can be filtered in the network layer in order not to highly increase the end-to-end delay. In practice, two algorithms were developed for the purpose of selecting filters in which matched events will be forwarded to the application layer, e.g., SSA and CSA discussed in sections 4.3.1 and 4.3.2 respectively. The two algorithms have different time complexity and provide different rates of reduced false positives to give more choices based on the application needs.

A number of different experiments were run on the implemented middleware to prove the credibility of the theory. The system was evaluated in comparison with PLEROMA as a representation for pure in-network layer filtering, and another system that uses pure application layer filtering. The results showed that the hybrid solution significantly reduces the bandwidth requirements compared with PLEROMA while keeping the delay in the middle between the two filtering approaches. It was also investigated that the performance of the hybrid solution can be adjusted by altering the value of Δ, the application threshold. This gives the application the flexibility of adjusting the trade-off between the bandwidth efficiency and the end-to-end path delay imposed when dealing with Pub/Sub systems.

As the system is built on top of PLEROMA[21] middleware, all proposed future enhancements apply to the solution that this thesis presents. Apart from the model presented by PLEROMA, the area of selecting filters for application layer filtering needs further research for a more optimized solution. As it was investigated that the complexity of the two implemented selection algorithms is highly dependent on the number of deployed switches in the underlying hardware infrastructure, optimization techniques can be researched in the future on how to reduce the search space relative to the total number of switches. Another open area for research is how to reduce network traffic by making the switches aware of all possible types of received packets, as the systems follows the approach of forwarding to the network layer controller for any packet that does not find a match. Some of these packets may be out of the context of the Pub/Sub middleware.

Appendix

Appendix A

Lists

SDN	Sotware Defined Networking
TCAM	Ternary Content Addressable Memory
ASICs	Application Specific Integrated Circuits
SIENA	Scalable Internet Event Notification Architecture
REBECA	Rebeca Event-Based Electronic Commerce Architecture
JEDI	Java Event-Based Distributed Infrastructure
SSA	Switch Selection Algorithm
CSA	Cluster-based Selection Algorithm
API	Application Programmable Interface
Pub/Sub	Publish/Subscribe
ICN	Information-Centric Networking
CPs	Content Providers
CDNs	Content Distribution Networks
MBR	Minimum Bounding Rectangle

List of Figures

Bibliography

[1] Floodlight. [online]. available: http://www.projectfloodlight.org/floodlight/.

[2] Mininet. [online]. available: http://mininet.org/.

[3] Bengt Ahlgren, Christian Dannewitz, Claudio Imbrenda, Dirk Kutscher, and Börje Ohlman. A survey of information-centric networking. *Communications Magazine, IEEE*, 50(7):26–36, 2012.

[4] Kapil Bakshi. Considerations for software defined networking (sdn): Approaches and use cases. In *Aerospace Conference, 2013 IEEE*, pages 1–9. IEEE, 2013.

[5] Guruduth Banavar, Tushar Chandra, Bodhi Mukherjee, Jay Nagarajarao, Robert E Strom, and Daniel C Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *Distributed Computing Systems, 1999. Proceedings. 19th IEEE International Conference on*, pages 262–272. IEEE, 1999.

[6] Sukanya Bhowmik, Muhammad Adnan Tariq, Boris Koldehofe, André Kutzleb, and Kurt Rothermel. "Distributed Control Plane for Software-defined Networks: A Case Study Using Event-based Middleware". *DEBS: Distributed Event-based Systems Conference*, pp. 92–103, 2015.

[7] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems (TOCS)*, vol. 19, no.3, pp. 332-383, 2001.

[8] Cisco. Cisco visual networking index: Forecast and methodology, 2014-–2019. Technical report, May 2015.

[9] C Gianpaolo Cugola and H.-Arno Jacobsen. Using publish/subscribe middleware for mobile systems. *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 6, no.4, pp. 25–33, 2002.

[10] Gianpaolo Cugola, Elisabetta Di Nitto, and Alfonso Fuggetta. The jedi event-based infrastructure and its application to the development of the opss wfms. *IEEE Transactions on Software Engineering*, vol. 27, no.9, pp. 827–850, 2001.

[11] Patrick Eugster. Type-based publish/subscribe: Concepts and experiences. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 29(6), 2007.

[12] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The road to sdn: An intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.

[13] Albert Greenberg, Gisli Hjalmtysson, David A Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. A clean slate 4d approach to network control and management. *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 41–54, 2005.

[14] Antonin Guttman. *R-trees: a dynamic index structure for spatial searching*, volume 14. ACM, 1984.

[15] Fei Hu, Qi Hao, and Ke Bao. "A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation". *IEEE Communications Surveys and Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.

[16] Boris Koldehofe, Frank Dürr, and Muhammad Adnan Tariq. "Tutorial: Event-based Systems Meet Software-Defined Networking". *DEBS: Distributed Event-based Systems Conference*, pp. 271–180, 2013.

[17] Adrian Lara, Anisha Kolasani, and Byrav Ramamurthy. "Network Innovation using OpenFlow: A Survey". *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, pp. 493–512, 2014.

[18] Xingkong Ma, Yijie Wang, Qing Qiu, Weidong Sun, and Xiaoqiang Pei. Scalable and elastic event matching for attribute-based publish/subscribe systems. *Future Generation Computer Systems*, vol. 36, pp. 102–119, 2014.

[19] Helge Parzyjegla, Daniel Graff, Arnd Schröter, Jan Richling, and Gero Mühl. Design and implementation of the rebeca publish/subscribe middleware. In *From active data management to event-based systems and more*, pages 124–140. Springer, 2010.

[20] Peter R Pietzuch and Jean Bacon. Peer-to-peer overlay broker networks in an event-based middleware. In *Proceedings of the 2nd international workshop on Distributed event-based systems*, pages 1–8. ACM, 2003.

[21] M. Adnan Tariq, Boris Koldehofe, Sukanya Bhowmik, and Kurt Rothermel. "PLEROMA: A SDN-based High Performance Publish/Subscribe Middleware". *ACM Middleware Conference*, pp. 217–228, 2014.

[22] Muhammad Adnan Tariq, Boris Koldehofe, Gerald G Koch, Imran Khan, and Kurt Rothermel. Meeting subscriber-defined qos constraints in publish/subscribe systems. *Concurrency and Computation: Practice and Experience*, 23(17):2140–2153, 2011.

[23] Sasu Tarkoma. *Publish/subscribe systems: design and principles*. John Wiley & Sons, 2012.

[24] Sisi Xiong, Yanjun Yao, Qing Cao, and Tian He. kbf: A bloom filter for key-value storage with an application on approximate state machines. In *INFOCOM, 2014 Proceedings IEEE*, pages 1150–1158. IEEE, 2014.

[25] George Xylomenos, Christopher N. Ververidis, Vasilios A. Siris, Nikos Fotiou, Christos Tsilopoulos, Xenofon Vasilakos, Konstantinos V. Katsaros, and George C. Polyzos. "A Survey of Information-Centric Networking Research". *IEEE Communications Surveys and Tutorials*, vol. 16, no. 2, pp. 1024–1049, 2014.

Declaration