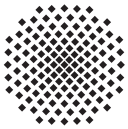


Universität Stuttgart

Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master's Thesis Nr. 0202-0011

Workload-enabled Content-based Routing in Software-defined Networks

Himanshu Sarmah

Course of Study:	INFOTECH
Examiner:	Prof. Dr. rer. nat. Kurt Rothermel
Supervisor:	Msc. Sukanya Bhowmik
Commenced:	2016-5-23
Completed:	2016-11-22
CR-Classification:	C2.1,C2.4

Abstract

Publish Subscribe Paradigm is a recent development in the area of loose coupling communication among distributed components. The communication can be facilitated with the help of content filtering within the network, here the content being the published messages. Using content filtering in the underlying network routers(also known as brokers in publisher/subscriber paradigm)enables to save valuable bandwidth by forwarding only those contents which are of interests to the subscribers.

With the introduction of software defined networking, it is now possible to directly install content filtering rules in the Ternary Content Addressable Memory(TCAM) of the routers in the network. The content filtering rules are created by combining binary representation of the contents and IPv4/IPv6 addresses from a predefined range of addresses. However, the expressiveness of the filtering rules is limited by the small number of bits available in the IP addresses. With increasing amount of published/subscribed messages, it is also required to have proper filtering to avoid unnecessary traffic in the network. Therefore, initial work was proposed to create filtering rules based on the workload i.e. amount of published and subscribed messages, although these techniques are based on a simulated publisher/subscriber filtering environment.

To explore the feasibility of the initially proposed techniques on an actual Software defined networking environment is the primary purpose of this thesis. This thesis discusses the proposed techniques with respect to a software defined networking environment and also implements the mapping of these techniques to work with a software defined publisher/subscriber network and evaluates the scalability and efficiency of the proposed techniques and the cost of doing so, in terms reconfiguration overhead.

Acknowledgements

I would like to express my gratitude to Prof. Kurt Rothermel for giving me the opportunity to write my thesis in the Institute of Parallel and Distributed System. A big thank you to my supervisor Msc. Sukanya Bhowmik for giving me guidance and helpful advices at each step, without which it would not have been possible to complete the thesis. I would also like to thank Dr. Muhammad Adnan Tariq for his valuable guidance during the evaluation step of the thesis. Thanks to Jonas Grunert for letting me work on his code-base for the dimension selection and workload indexing part. I would also like to thank the Floodlight and Mininet discussion groups who helped me to resolve some problems that I had while working on my thesis. I am also grateful to my friends for being with me and being helpful in every ways possible. Last but not least, I am very grateful to my parents and my family for being with me all along and helping me in every way possible to complete my thesis.

Contents

List of Figures

Chapter 1

Introduction

Distributed systems these days have become greatly scalable to the extent that the entities involved in exchanging information may not always be capable of doing a direct or point to point communication using each other's addresses. They may be located around different parts of the globe and have a different kind of behaviour but still should be able to exchange information among them reliably over a wide area network like the Internet. Such asynchronous and loosely coupled behaviour of sender/receiver has motivated development of the publish/subscribe model for distributed systems[16].

In a publish/subscribe paradigm, also known as *pub/sub* model, instead of sender and receiver it uses the concept of publisher and subscriber. The publisher can advertise the type of content that it wants to publish and the subscribers can express interest to receive contents by subscribing for them. Later when the publisher publishes some contents, the matching between the published content and the subscriptions are checked and the content is delivered to the subscriber if there is a match. This kind of communication can be found in for example news-feed, RSS feeds, electronic auctions, system and network management,business process management, service discovery, sensor networks etc[6][18]. This type of communication can not be provided by traditional distributed system messaging paradigms; which involve synchronous end to end communication, rather there was a need to develop a middleware that could abstract the distributed nature of the publisher/subscriber and can establish reliable communication among them. In such a situation, the middle-ware would handle the abstraction of the end hosts i.e. publisher or subscriber so that the end hosts do not need to be aware of the location or state of each other. The middleware provides so-called Notification Service, where the publisher can advertise and publish, and the subscribers can subscribe and receive published contents[9]. As different from traditional distributed system communication paradigm e.g. message passing, RPC, shared space, Message Queuing etc.; with the help of the notification service the publish/subscribe paradigm offers significant advantage in terms of many to many, and asynchronous mode of communication[16].

Publish/subscribe concept can be implemented in different modes, for example topic based or content based. In topic based model, the subscriptions/advertisements happen based on interest to receive/send information on a particular topic, and also the matching and routing of events are done by checking topic similarity. In content based mode, subscribers/publishers can be more specific about their interest to receive/send information by using conditions on

contents instead of topics. Also the published contents are matched with the subscriptions by actual content checking and comparison. In this way content based pub/sub offers more expressiveness compared to topic based pub/sub. In other words, in content based pub/sub the published contents can be filtered in the network based on information available from the advertised/subscribed content, which therefore reduces unnecessary bandwidth consumption and also increases accuracy for the subscribers since filtering in the network means the subscribers will receive contents only for which they expresses interest[6].

There have been a lot of research done on content based routing for pub/sub system as discussed in [18],[28],[9] where the content filtering is performed by entities called *brokers*. Brokers work on the overlay network and uses the underlying network information to create routing plans for published contents. But, since brokers do not operate in the underlying network, using broker based content filtering result in significant delay for end to end content delivery as a result of the extra processing that the brokers need to do to match and filter the content[40]. Recent advancement in the networking field like software defined networking(referred as SDN hereafter) now makes it possible to install content filters directly on the network routers without involving any extra processing. In software defined networking, the whole network is divided into two parts, the control plane and the data plane; a centralized *controller* has global view of the underlying network and can control the multilayer switches with protocol like OpenFlow[47]. OpenFlow makes it possible to install filtering *rules* also known as *flows*[17] on the TCAM memory of switches to enable end to end routing of packets. The filtering rules that are used for OpenFlow are similar to content filtering rules needed for a pub/sub system. Therefore using the concept of SDN, the content filtering now can be performed without any application layer overhead; directly at the switch, it is just a simple memory lookup operation[40]. In many parts of the thesis, the term *switch* is used to refer a multilayer switch which is capable of layer 3 based routing and not just layer 2 packet switching.

As discussed above, content filtering in an SDN based pub/sub can be done with the help of match fields on the Ternary Content Addressable Memory(TCAM) of the switches and middle ware PLEROMA[40] provides an efficient way to install content filtering rules using which the published events can be matched and forwarded to the subscribers at line rate performance. Even though PLEROMA provides efficient line rate content filtering, it still suffers from problem due to limited bits available to represent content. The content filtering technique used in PLEROMA first converts the contents to binary bit streams and then append those streams to a set of predefined IP addresses to generate new IP addresses which are then used as match fields on the switches. The expressiveness of this technique is dependent upon the number of bits available in the IP addresses where the bit stream can be appended. A limited number of bits available create less expressive rules and thus results in consumption of extra bandwidth by unnecessary traffic in the network. This problem has been identified and initial solutions have been proposed in [7] to represent contents efficiently and also to eliminate redundant data to make the content filtering more expressive. The proposed solutions try to represent contents based on the workload of the system i.e. amount of subscriptions and events

published; although the solutions have been proposed in a simulated pub/sub environment and are yet to be explored in an actual SDN based pub/sub network. This thesis tries to solve this problem and maps the proposed solutions to an SDN environment and explores the feasibility and scalability of the proposed techniques.

1.1 Thesis Organization

The thesis has been organized as below:

Chapter 2 provides the necessary background topics and understanding needed for the thesis. It gives an overview of the pub/sub concepts and software defined networking and also provides details about how pub/sub can be realized in an SDN.

Chapter 3 provides the problem statement of this thesis and also discusses some related work which are of relevant for the thesis like content filtering in SDN and how it is realized in PLEROMA.

Chapter 4 provides understanding of the techniques and algorithms that have been proposed in [7],this thesis being an extended work of [7] it is important to know how the techniques solves the problem and how they can be mapped to an SDN environment. This chapter also provides the implementation details of the techniques proposed in [7] for an SDN environment.

Chapter 5 provides analysis and evaluation of the implemented techniques including the test environments used and results of the conducted experiments.

Chapter 6 concludes the thesis with the summary of the work done and including scope of future work that can be performed.

Chapter 2

Background

The objective of this chapter is to provide adequate background knowledge for the theoretical concepts needed to understand the thesis further. Specifically it gives detailed introduction of pub/sub principles, software defined networking concepts and also how pub/sub concept is realized in software defined networking.

2.1 Classical Communication Paradigms in Distributed Systems

A distributed system is a collection of computing systems distributed across geographically different regions, but appears as a single computing system to the end user. For a system to be called a distributed system some characteristics[39, Chapter 1] that it needs to posses,

- Remote resources like computing servers, printers etc. should be easily accessible and can be shared by multiple users at same time.
- A distributed system should hide it's distributed nature from the end users in terms of location of computing systems, or remote resources should be accessible the same way a local resource is accessed by an user.
- A distributed system should be easily scalable i.e. it can cope with increasing number of users without any failure.

Keeping the above characteristics in mind, facilitating communication among multiple entities involved in a distributed system is definitely a challenging task. Before moving onto publish/-subscribe paradigm, some of the other popular communication mechanisms in distributed systems are discussed in the following sections.

2.1.1 RPC

Remote Procedure Call(RPC) is a very popular communication principle in distributed systems. The main principle of RPC is that, a process residing in a client machine can call a procedure which is implemented on a server machine[39, Chapter 4] via a network in which the client and the server machines are connected. To the client process, calling a remote procedure seems no different than calling a local procedure, but there is another process involved known as the *stub* which converts the local procedure call into a transportable message and sends it to its operating system which sends the message to the the server machine's operating system. Similarly for the server, it also has a server stub process which listens for incoming

messages, on receiving the message from it's operating system it converts the message into a method call which is actually implemented on a server's process running on top of the stub. The process returns the result of the method to the server's stub which then packs it into a response message, sends it over to the client and the client's stub unpacks and sends the result to the client process that called the method as a local method. Figure 2.1 shows the

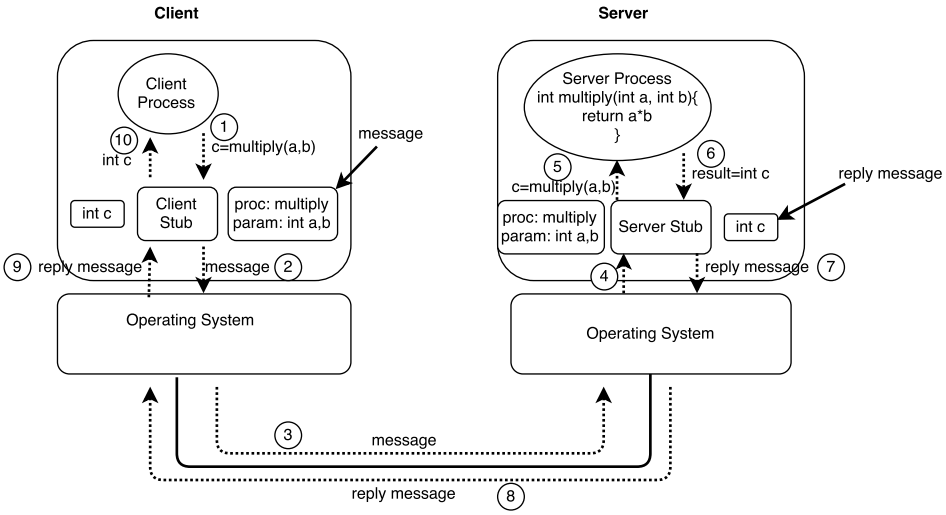


Figure 2.1: Remote Procedure Call

complete process of RPC where the client process calls a procedure to multiply two number which is actually implemented on a server's process. The numbers represent each of the steps followed to complete the remote procedure call. The example shown here is a simple one with both client and server having similar data representation. The client and server machines may have different data representation e.g. one might work on big endian and other might work on little endian, or might need a particular type of message that it understands, in such cases additional implementations are needed so that both the client and server agrees on a similar data representation and a message structure understandable by the respective client/server. An RPC can be synchronous where the client calls the remote server and waits for reply or it can be asynchronous mode where the client sends the request and proceeds with some other task and gets interrupted when the server replies.

2.1.2 RMI

Remote Method Invocation(RMI) is a variant of RPC specifically for Java where a remote object is referenced which can in turn call local procedure and return the results. The difference with RPC being, in RPC additional actions are needed for converting the data into local data structure and also for sending the response back to the client, but in RMI the client invokes a remote method directly using a reference of a remote object on the server. The process of RMI with respect to Java can be explained in the following steps[29],

- The server has one or more remote interfaces which extends java.rmi.Remote interface and the interfaces declare all the methods that can be called remotely by a client. For a

remote object to be able to invoke any method, the object's class must implement one or more of these interfaces.

- The server also has a stub which also implements the remote interfaces. In this way all the remotely invokable methods are also available for the stub and it acts as a proxy for all the objects that are available to the client as remote object.
- There exists a registry service on both server and client where the server can bind objects to the registry service to make them available to clients and clients can look up the registry service to get the reference of a remote object. In reality, after look up instead of the actual object reference, the registry service returns reference of the stub of the remote object which has all the methods available on it as well. The client then call the remote method using the stub reference.

2.1.3 Message Oriented Systems

As discussed above, RPC and RMI enable distributed communication by hiding the details of transporting message over a network. However, they still need both the client and the server processes to be executing at same time for a communication to happen. If there is a failure at any side or there is a failure of the network, then the communication fails, also synchronous mode of communication blocks the client until it receives a reply. To avoid these kind of issues, message oriented communication paradigm was introduced. The following types of message oriented communication paradigms are relevant,

Message Passing Interface Message passing interface provides an interface between the sender and receiver entities using which they can communicate independently of each other. The interface provides message buffers where the messages from a sender are stored and the receiver can read the messages from the buffers. MPI provides different primitives for synchronous and asynchronous modes of communication for both sending and receiving purposes. It also has primitive for passing a reference to a message. The sender and receiver entities in MPI are identified by using a tuple of group id and a process id, where the group id specifies the group in which the entity belongs to and the process id identifies the entity uniquely within a group.

Message Queuing Message queuing system provides queues for each of the distributed entities involved in communication where any data can be stored and received by the entity when it is ready. Because of presence of queues, it is not necessary for the sender and receiver to execute in parallel to successfully exchange message between them. However, the queues of each involved sender/receiver needs to be identified uniquely where a message can be addressed.

Now that we know how traditional communication paradigms work in a distributed system, in the next section the thesis explores the publish/subscribe paradigm and how it is different and advantageous than the existing modes of communication.

2.2 Publish/Subscribe Principles

Publish/Subscribe paradigm enables communication among distributed entities that want to send and receive information without explicitly specifying name or address of any receiver/sender. In Publish/Subscribe paradigm the communicating entities do not have to be aware of each other's existence, they may be located in geographically different positions, and can initiate a information send/receive process independent of each other. Instead of traditional way of distributed communication e.g. client server model[45]; in publish/subscribe principle there exists a set of clients known as publisher and subscriber that communicate to a centralized notification service using a predefined message structure. The notification service then takes care of the involved complexity to establish message delivery among the clients. In the following parts, this thesis explores the details of the key elements involved in pub/sub principle and also discusses the specific message structure used in this thesis.

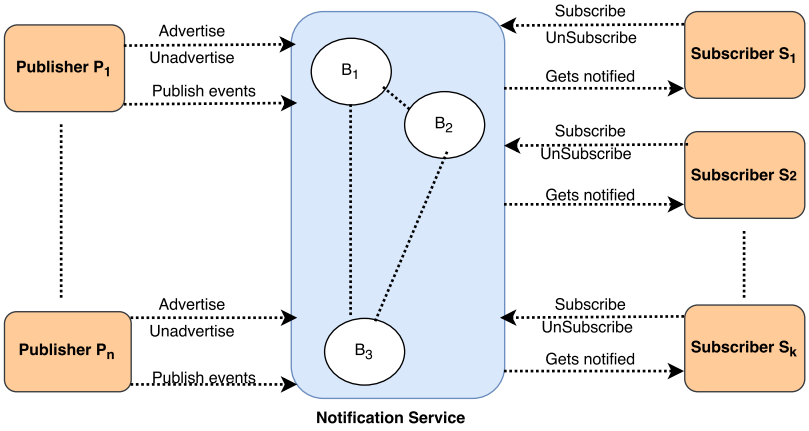


Figure 2.2: A publish/subscribe system

Publisher A publisher is a participant involved in pub/sub system which is also described as the source of data. It can send advertisements about the type of data that it wants to publish, and publishes data which are commonly known as *events*, based on the advertisements it made.

Subscriber A subscriber is the second participant in pub/sub system which is also described as the data consumer. A subscriber can send subscriptions describing the type of information that it wants to receive to the notification service and listens for incoming events. Based on the subscriptions the notification service delivers events published by the publisher to the interested subscriber(s). A subscriber can have multiple subscriptions and therefore the notification service has to deliver it all events matching its subscription set. Figure 2.2 shows structure of a pub/sub system with multiple publishers and subscribers.

Notification Service Since publishers and subscribers are not communicating directly, there exists another entity known as notification service which is responsible for all the logical calculations needed to deliver published events to the interested subscribers. In pub/sub paradigm the notification service consists of entities known as *broker*, the notification service may consist of one or more brokers. As shown in Figure 2.2 all the publishers and subscribers communicate with the notification service, if there are more than one broker involved, then the publishers/subscribers may communicate with any broker from the set and the brokers have to communicate among themselves for establishing rules to filter out events for corresponding subscriptions.

Operations and message structure The publishers and subscribers must communicate to the notification service in a predefined message structure understandable for the notification service. The publishers can *advertise* to the notification service declaring the type of content that it wants to *publish* in future. In general an advertise message can be of structure *Advt(X,F)*[42] where X identifies the publisher and F identifies the criteria that will be used by the publisher while publishing events. Publishers can also perform *un-advertise* using *UnAdvt(X,F)* which will remove the advertising criteria for the publisher. Similarly for a subscriber, it can subscribe for events that it is interested in receiving using *Sub(Y,S)* where Y identifies the subscriber and S identifies the criteria that it has subscribed for. Similar to publishers, subscribers can also *UnSub(Y,S)* which will remove its subscription and the notification service will not notify it of any event matching the criteria identified by S. Events published by the publisher can be simple attribute,value pairs[6] , for example an event can be e=*{*$Attr_0$,$Value_0$;$Attr_1$,$Value_1$;$Attr_2$,$Value_2$*}*. An event e will be delivered to a subscriber if the values of the events match to the criteria S specified by the subscriber.

Spatial Indexing for multidimensional publisher/subscriber data The previously mentioned message structure is a general structure, which means the messages used by the publisher/-subscriber can be adapted to some specific data structure in order to solve a problem. Such a structure has been proposed in PLEROMA middleware[40] that uses a concept called *dz-expression*. Since this thesis uses PLEROMA as the middleware, therefore it is important to know about the message structure used by publisher/subscriber in PLEROMA. It uses the concept of *spatial indexing* to create dz-expressions which are bit string representation of each subscription or advertisement contents with multidimensional data. Spatial indexing is an indexing technique mostly used in database systems to make querying for data faster in a multidimensional environment[26].

In the model described in PLEROMA, the content space is represented as an N-dimensional space where each dimension can refer to an attribute like Pressure or Temperature. The advertisements or subscriptions can be a sub space in the whole content space. Figure 2.3 shows a two dimensional content space w with dimension A in X-axis and dimension B in Y-axis, the range of each axis being the maximum and minimum values for each dimensions A and B. Each subscription or advertisement can have dimension A and B as attributes and a range specifying their values. For example a subscriber can have two subscriptions like this, s_0=(A=[50,100];B=[0,50]) and s_1=(A=[25,75];B=[0,100]), and similarly for advertisements.

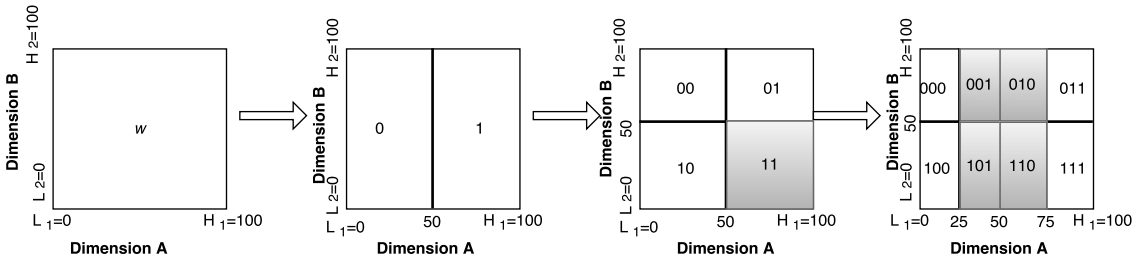


Figure 2.3: Using spatial indexing for two dimensions A and B

An event can be represented as a point in the whole content space with a specific value for each dimension, for example an event can be e=(A=60,B=30). With recursive binary decomposition of the content space as shown in Figure 2.3 each subspace can be represented by a bit string known as the *dz-expression*(in short just dz), and it is quite obvious that as the subspace for each dimension gets smaller, the dz-expressions needed to represent that space gets longer. The subscriptions mentioned earlier i.e. s_0 and s_1 can be represented by dz-expressions {11} and {001,010,101,110 } respectively. The dz-expressions thus generated will be used in later parts of the thesis.

2.2.1 Variations of pub/sub model

Based on the way that subscribers express interest to get notified for events, a publish/subscribe system can be broadly categorized into topic based or content based publish/subscribe model.

Topic based Publish/Subscribe In a topic based pub/sub model, each subscriber subscribes for a particular *topic*, and gets notified when a published event matches with the same topic. It is quite common to refer the topics as *tag*[32] or *group*[11], the common thing being that a logical channel is created among the participating subscribers and publishers when they match on a particular topic/tag and then subsequent events falling into the same topic can be broadcast in that logical channel and thus ensuring that all subscribers can receive that event. Topic based pub/sub enables efficient communication when the subscribed and published data have a natural tendency to fall into a particular topic[48]. Lot of research have already been done on the topic based pub/sub, some of which are [32], LIPSIN[22], SCRIBE[11]. Even though topic based pub/sub provides efficient communication for a fixed set of topics, it however has limitations regarding the expressiveness of subscribers and for dynamically changing topics. A subscriber may not be interested in the whole set of data related to a particular topic, but without more granular expressiveness the topic based event notification system might result in a lot of unnecessary events being delivered to the subscriber[6]. Also, classical topic based pub/sub systems are not designed to work with dynamically changing subscriptions, which has been attempted to solve in DYNATOPS[48].

Content based Publish/Subscribe In a content based pub/sub model, the subscribers can be more expressive regarding their interests, also unnecessary traffic is reduced in the network since matching and delivery of events happen based on the properties of the actual content.

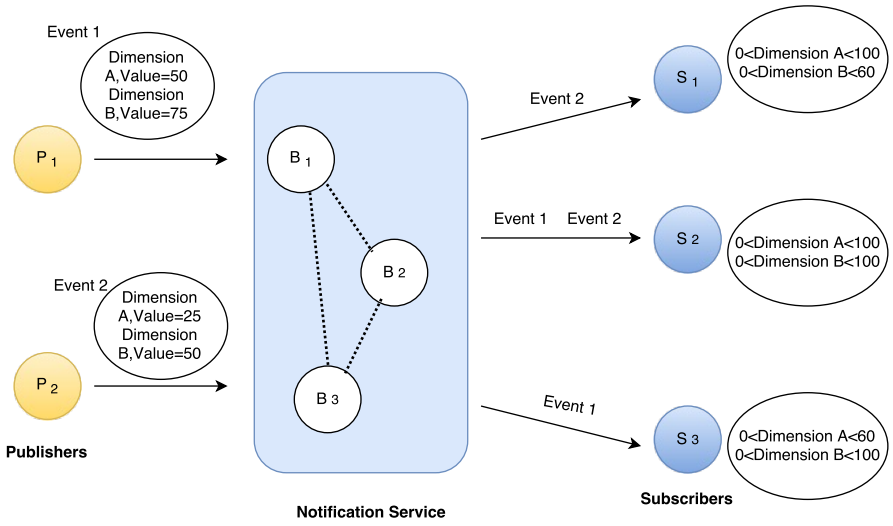


Figure 2.4: A Content based pub/sub model

The subscribers can specify specific criteria for attributes that they are interested in and get notified when an event matching that criteria is published. For attributes having *string* type values, the criteria can includes operators like *equal*,*prefix*,*contains*,*suffix* and for attributes with numeric values the criteria can include operators like $=,<,>,\leq,\geq$[14]. Of course the notification service will need much more sophisticated implementations as compared to topic based pub/sub to efficiently deliver contents to the subscribers. Figure 2.4 shows a content based pub/sub model with two attributes A and B with a notification service consisting of three brokers. The subscriptions of the three subscribers are expressed as specific range of values for attributes A and B. So when a published event's content satisfy the subscription criteria, it is delivered to the subscriber. Subscribers S_1 and S_3 are receiving the events Event 2 and Event 1 respectively, because each of these event's values along dimension A and dimension B fall within the specific range that was specified by the corresponding subscriptions. While, subscriber S_2 is receiving both of these events following the same logic. Some of the most popular content based pub/sub systems being Gryphon[12], SIENA[9], JEDI[15]. In contrast to topic based pub/sub, not much research has been done regarding channelization in a content based pub/sub system with brokers. All the brokers in the notification service have to do per event based matching calculation, resulting in high overhead for the notification service. A lot of research have been done to make matching or filtering of events more efficient and scalable for pub/sub model, the details of which are being discussed in the next section.

2.2.2 Event delivery mechanism in pub/sub model

For a topic based pub/sub system, matching of event with subscriptions is easier because the subscriptions and events already fall into predefined topics and matching an incoming event

is just a look up operation[5]. But since content based filtering relies on the actual content matching operation of each published event with the subscriptions, it introduces overhead for the notification service which makes achieving scalability for a large number of subscriptions or events a challenge. In this context it is important to mention that if matching of events are not efficient, the subscribers will receive event notification which they did not subscribe, resulting in *false positives*. False positives waste important network bandwidth, and therefore it is a goal of any pub/sub system to keep false positive at minimum level. The most naive way of filtering is to check all subscriptions against a published event. This is not efficient because the run time of this algorithm increases linearly with increasing subscribers[5] also this algorithm is based on the assumption that the notification service has a global view of all subscriptions. If there are multiple brokers in the notification service, creating a global view for each of the brokers creates a lot of unnecessary traffic in the system and also introduces overhead for the brokers in terms of storage memory and event matching operation. There have been a lot of research on this topic and it has been established that the most efficient way of filtering in a notification service with brokers is to create multicast groups based on subscription clustering[34],[10],[31]. In the middleware PLEROMA, this problem is solved using a prefix based matching of advertisement and subscriptions and then creating corresponding routing tree between publisher and subscriber, the details of which will be discussed in a later part of this thesis.

Besides filtering, events also have to be routed properly to the interested subscribers by the notification service. There can be two types of routing for a content based pub/sub model, *simple routing* and *covering based routing*[28].

- In simple routing all the events are flooded to every broker in the notification service, ensuring that any subscriber connected to a broker receives the events that it is interested in. Obviously this introduces lot of unnecessary traffic in the system and also this routing scheme assumes that every broker has a global knowledge about all the subscribers of the system. With dynamically changing subscriptions, it will involve huge overhead to maintain the routing tables of every broker.

- In covering based routing, the brokers look for similarities among subscriptions. It is very likely that a lot of subscribers are interested in a similar set of events. Therefore, instead of flooding all the subscriptions to all the brokers, a broker will forward a subscription to its next broker only if the subscription generates a new interest that was not already covered by some subscriber already known to the broker. Similarly for an event, it will be forwarded by a broker to its next broker only if the event is covered by an existing subscription in the broker. In this way network bandwidth is reserved as well as storage space is reserved at the brokers. When the broker receives an event, then it can either deliver it to a connected subscriber or forward it to its next broker based on the subscriptions that it has. The amount of unnecessary messages can be further reduced by use of advertisements before a publisher actually publishes.

2.3 Contemporary pub/sub Systems

As discussed before, there can be different varieties of a pub/sub model. To have a general overview of different attempts of realizing a scalable and efficient pub/sub model, in this section some of the state of the art publish/subscribe system are being discussed.

2.3.1 SIENA

SIENA[9] presents a wide area event notification service for content based pub/sub system with event filtering implemented at application layer. Besides subscriptions, SIENA uses the concept of advertisements that the publishers can use to avoid sending of unnecessary notifications. Also SIENA is the first to introduce the concept of *unsubscribe* and *unadvertise* in the publish/subscribe paradigm. The notification service implements the covering based routing that was mentioned in the previous section, following the similar technique for propagation of advertisements. The notification service defines two kind of filtering techniques, which are subscription based and advertisement based.

- Subscription based filtering defines the condition under which a notification will be delivered to the interested subscriber. For a notification to be delivered to an interested subscriber, all its attributes must be covered by the corresponding subscriber. Figure 2.5 shows an example for the same. In this example the second notification is not

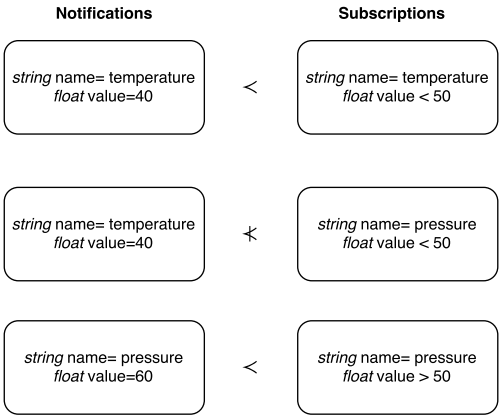


Figure 2.5: Covering relation of notification and subscriptions

  covered by the subscription because the field *name* of the notification does not match with the corresponding field of the subscription. In this case only the first and the third notifications will be delivered to the subscribers.

- Advertisement based filtering technique defines the conditions under which a notification is allowed to be published by the publisher. A publisher can have multiple attributes that it can use while advertising, and for a notification to be published, each individual attribute of the notification needs to be covered by an advertisement. In contrast to subscription based filtering, the advertisement and and notification need not have an exact match but the notification attributes should be a subset of the advertised attributes.

Figure 2.6 shows an example for the same. In this example the second notification is not

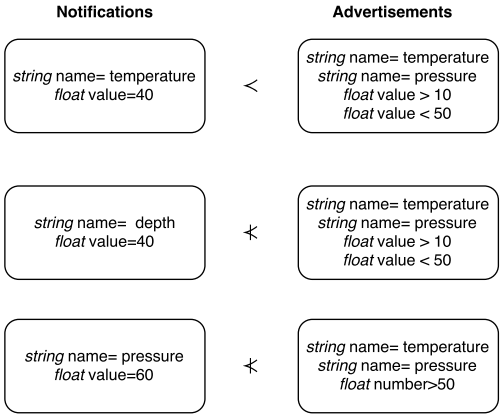


Figure 2.6: Covering relation of notification and advertisements

covered by the advertisement because the value of the attribute *name* is not included in the advertisement. The third notification is not covered because the attribute *value* is missing in the advertisement.

SIENA further handles unsubscribe and un-advertise messages using similar covering relation. If an unsubscribe message covers existing subscriptions, then all those subscriptions are removed from the notification service. Similarly for un-advertise, if an un-advertisement message covers existing advertisements, then those advertisements are removed and disabled from publishing further notifications in future. The notification service presented in SIENA offers a scalable and expressive content based pub/sub system, but the drawback being that it is implemented on the application layer and therefore it suffers from additional end to end delay for matching and forwarding of notifications.

2.3.2 JEDI

Java event based distributed infrastructure(JEDI)[15] is an object oriented implementation of a notification service for a content based pub/sub system. It is based on the concept of *Active Objects*[43], where the publisher and subscribers take the role of active objects and another component called the *Event Dispatcher* has the role of the notification service. All active objects have to establish a connection to event dispatcher using *open* operation. Supported operations by the subscribers in JEDI are Subscribe, Unsubscribe, Receive and publishers can publish or generate events. A subscription in JEDI can be of structure *name(parameter1,parameter2..)* where *name* is the name of the subscription and *parameter1* and *parameter2* are the parameters specified by that subscription. The subscribers can subscribe for one particular event with specific values for name and parameter or it can use a pattern in its subscriptions to match with multiple events. For example, a subscription with structure *Temp*(*,high)* will match with any event whose name starts with *Temp* and has a second parameter called *high*. JEDI also supports mobility of the publishers/subscribers

using the concept of Java Aglets[25]. To support the mobility feature subscribers/publishers can perform two additional operations which are; *moveOut* to disconnect itself from the event dispatcher and *moveIn* to reconnect to the event dispatcher at a later time after it has performed *moveOut*.

2.3.3 GRYPHON

GRYPHON[12] is a content based pub/sub system designed to work with brokers running on the overlay network. GRYPHON in particular takes special care for scalability, availability and security for a distributed pub/sub system, and therefore it has been used in real time score distributions for Grand Slam Tennis and Ryder Cup. Some of the most important concepts involved in GRYPHON are described below,

GRYPHON uses the concept of information flow graph[37] to model the flow of events from publishers to subscribers. An information graph can do operations like *transform*, *merge*, *select* on the existing data to produce new variance of data and to select data of a specific attribute respectively. Using this concept GRYPHON can operate over a distributed set of brokers over a wide area network like the internet. Figure 2.7, shows a information flow graph

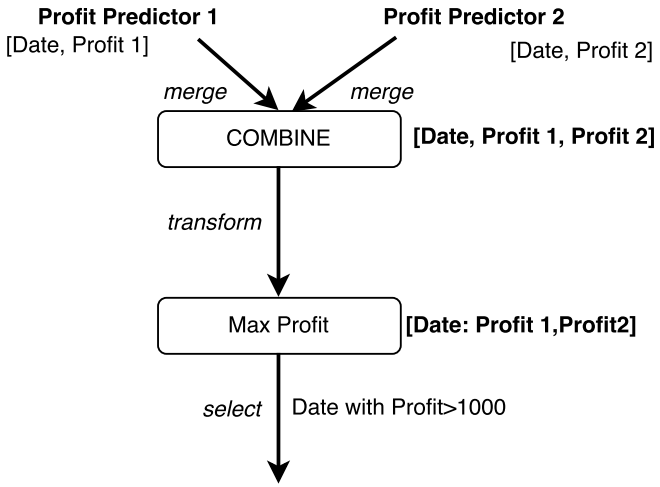


Figure 2.7: Information flow graph model in GRYPHON

with two data sources which predict different profits on a date and the data published by them contain two attributes date and profit. This data is combined and put in the COMBINED information space using the *merge* operation, and in the next step *transform* operation is applied on it which transform the data to a form where date is the key and the prices are the values for it. By using the *select* operation, a date can be selected with a minimum threshold value of profit. Some other operations that can be performed are *interpret*, *expand*, *collapse*.

The matching of events with subscriptions in GRYPHON is solved by using a parallel search tree where the subscriptions are the leaf nodes and they can be traversed by the events from the root nodes using some tests which are represented as the non leaf nodes in the tree[1]. A subscription in GRYPHON can contain the attributes from the information space that was discussed before, for example in Figure 2.7 a subscription can be $Subscription_1$=*[date=22.07.16, profit > 1000]*. Figure 2.8 shows the matching tree that is needed to match an event with

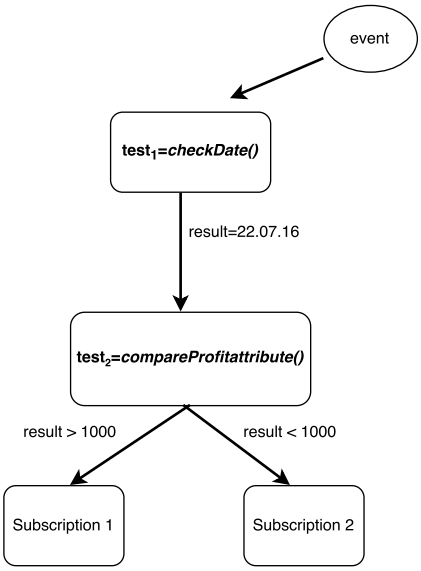


Figure 2.8: Content matching in GRYPHON

this subscription. The root node of this tree being the $test_1$, *checkDate()* which returns a result represented by the edge to the next node $test_2$ which tests for the profit attribute and based on the result of this test the event can follow any of the two edges pointing to two leaf nodes which are the subscriptions. In this example the event matches with $Subscription_1$. GRYPHON provides even more flexible tree structures including don't care edges when some subscriptions have a wild-card matching for a particular attribute.

GRYPHON produces impressive performance using their matching algorithms and parallel search tree technique, even though this still requires the brokers to have a complete knowledge of all the subscriptions involved in the system.

2.3.4 LIPSIN

LIPSIN[22] is a topic based pub/sub system which offers line rate performance by implementing forwarding on the network layer. It uses a network structure that consists of three layers, *forwarding layer* for forwarding of events from publisher to subscriber,and above that *data plane* that takes care of the the traditional transport layer functionalities and in addition introduces some new functions like caching and error correction, on top of it lies the *control*

plane which consists of the topology manager to create view of the underlying network topology and the rendezvous system which actually takes care of the matching between publisher and the subscriber based on topics. LIPSIN uses the concept of bloom filter[44] to identify each link in the network topology. Each link has a link id which is represented by a bloom filter and the topology manager has a global view of all the links and link ids that it can use to determine the path that an event published by a publisher needs to follow. When a publisher publishes an event, the packet containing the event also has the result of OR operation of all the link ids in the form of one vector called *zFilter* in its header. When the packet arrives at one node, it checks its routing table that has all the links and their corresponding link ids, and does AND operation of the zFilter with the individual link ids, and if the result matches one particular link id in its routing table, then the node forwards the event to that link.

LIPSIN achieves a line rate performance using a topic based pub/sub system which is a first of its kind. However, it does suffer from problem of false positives due to usage of bloom filter and has a low expressiveness because it is a topic based system. It would be interesting to see the combination of content based routing done on a network layer which will offer both expressiveness and line rate performance. With this motivation in mind this thesis discusses the next section on software defined networking.

2.4 SDN

Software defined networking is the recent advancement in the computer networking sector which makes it easy to control the switches with the help of a centralized controller from application level software, instead of manually programming each and every switch. When there are a lot of switches involved in a network and also huge amount of information exchange is happening for example in a datacenter network; it demands quick programmability of the the network for routing without any information loss and as well as for failure recovery purpose. In such a situation, manually programming switches would not be so effective, and hence the rise of software defined networking which makes it possible to treat the network like a software with pluggable modules. In software defined networking, there is a centralized controller with a global view of all the underlying switches that it is connected to, and developers can write applications on top of the controller to control the underlying switches.

Figure 2.9 shows the structure of a typical software defined networking model, the switches are connected to a specific port of the controller via the *southbound interface*, the communication protocol between the switches and the protocol is defined in OpenFlow protocol[17]. Therefore the switches must be capable of understanding OpenFlow protocol. The applications can use the services provided by OpenFlow to program the switches. Since the controller provides access to all the underlying switches, it is easier for the developer to write applications involving logic which considers all of the switches together. In this context, below are some of the important terms and concepts involved in an SDN.

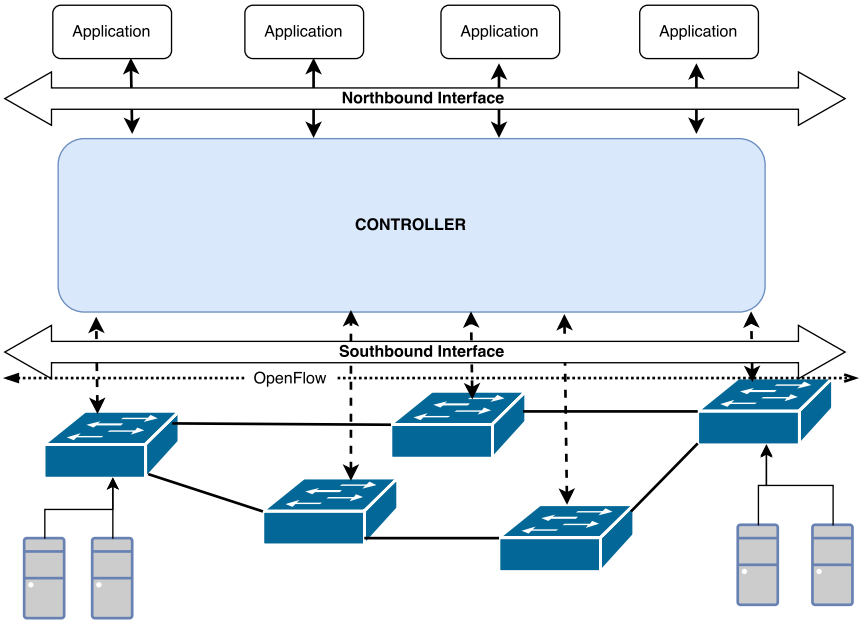


Figure 2.9: SDN Architecture

Proactive and Reactive SDN In a proactive model, the controller would install flow rules in all the underlying switches before any packet arrival at the switches. That is not a practical approach since it is not possible to know about all the packet headers beforehand. In reactive model, by default all the switches are instructed to send a packet to the controller when it does not know what to do with a packet. The controller then checks the header of the packet and uses the logic implemented as an application to create end to end path and pushes flow rules on all the switches involved in the end to end path, so the next time a packet with the same header arrives at the switch it can just follow the flow rules already set by the controller.

A flow A flow is installed in the switch's hardware by the controller as a rule that the switch will follow for all incoming packets matching the entry *match field*.

Match Field	Priority	Counters	Actions	Timeout	Cookie

Figure 2.10: A flow table with different entry fields

- A match field is an entry in a flow rule, that can be used to match incoming packets. A match field can be destination/source IP address,MAC address, VLAN id, ether type of the packet(IPv4, ARP, IPv6 etc.), incoming port of the switch, TCP/UDP port and so on. Based on the match field of an incoming packet, the switch decides what to do with the packet or to which port the packet has to be sent out. There can also be *wildcard* matching rule which can match with any incoming packet.

- Priority field to specify the priority of this flow rule, since there can be multiple flow rules in a switch.
- Counters to keep track of count of packets that has been matched.
- Time-out field that can be specified if the flow needs to be removed after a certain idle time.
- Action is another entry in a flow that specify the actions that a switch need to perform when an incoming packet matches on the corresponding match fields. Actions can specify to send the packet out via a particular port, drop the packet or to modify existing headers of the packet. Multiple actions can be set together, for example the switch can modify some header fields of a packet before sending it out via a particular port.
- Cookie field to uniquely identify a flow.

Once the flows are put into the switch's hardware, the switch can make decisions about routing the incoming packets. Figure 2.11 shows the matching procedure when packets arrive at the switch. In this example, a flow table is put on the switch where the match field is set as ether type IPv4, and action is to send out such packets at port 3. Therefore when an IPv4 packet matching the match field criteria arrives at the switch, it is sent out via port 3. But when an IPv6 packet arrives at the switch, the match field does not match with it and hence the switch sends the packet to the controller to decide about the packet. As discussed, with introduction

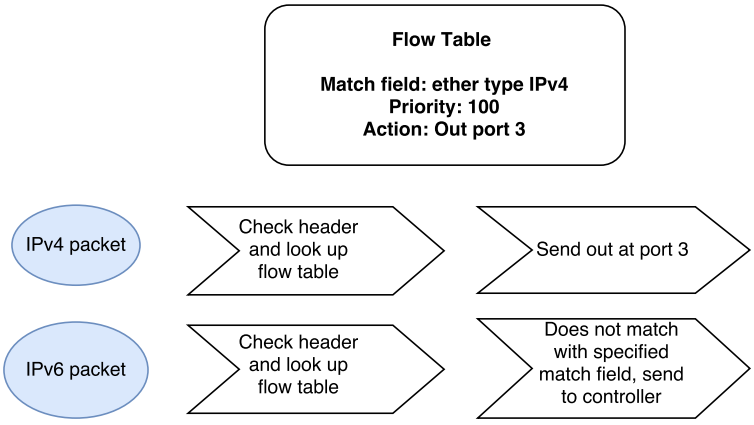


Figure 2.11: Matching procedure in switch's hardware

of SDN it provides endless possibilities to inject application level logic into the network layer and therefore can be applied in different use cases. Its application in the publish/subscribe paradigm has been first proposed in [23] and in the next section of this thesis it is discussed in details.

2.5 Pub/Sub Model on SDN

As mentioned before in section 2.1, there have been few successful implementation of large scale distributed content based pub/sub systems like [9],[18],[30]. However, each of them does

event matching and filtering at the application level and rely on getting correct underlying network information on the overlay network. Approaches like [24],[20],[13] explore different methods to infer underlying networking information, using which filtering can be applied for a broker based pub/sub at the overlay network. But from the performance perspective, it would be more efficient to implement a content based pub/sub model on the underlying network layer instead of overlay network[23]. With introduction of SDN, instead of using event filtering techniques at the application layer, filtering rules can be installed as flows in the TCAM memory of the switches which can offer line rate content filtering. In the proposed technique, the publishers and subscribers are connected as end hosts to the OpenFlow enabled switches. A fixed IP address is used in the packet header by the publishers/subscribers when they send any advertisements/subscriptions, and no flows are installed in switches against that IP address. So each time a publisher/subscriber initiates advertisement/subscription requests, they get forwarded to the controller by the switches. After receiving the subscriptions and advertisements data the controller acquires a global view of all the publishers and subscribers, specifically the switch and the port to which they are connected. The controller then uses the dz-expressions of the advertisements and subscriptions created using spatial indexing, to find covering relation among them to install corresponding flows on the switches. Events also have corresponding dz-expression which is converted to IP address and used as destination IP address of each packet containing an event. So when the packets are published, they are filtered in the network by the switches by matching the IP address on the packet headers with the match field of the installed flows. In this way, network bandwidth is reserved by filtering out unnecessary events. Also, since there is no extra processing involved for the switches to match and filter events, filtering at underlying network offers performance benefits by reducing end to end delay which was a major drawback for broker based application level filtering.

In the next chapter the thesis looks into details about how an efficient pub/sub model is implemented in SDN using PLEROMA which is the middleware used in this thesis.

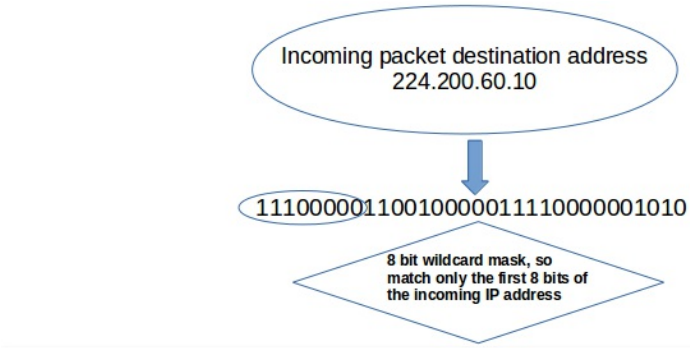
Chapter 3

Content Filtering at Line Rate

The objective of this chapter is to provide the detailed description for content matching and filtering in the switch's memory and how this feature is used for implementing an efficient pub/sub model in PLEROMA. At the end of this chapter, the problem statement of the thesis is stated.

3.1 Content Matching in Switch

To achieve line rate performance, event filtering must be performed at the network layer as filtering at overlay network introduces additional delay. For a content based pub/sub system, it needs a way to match and filter event contents at the network layer. Every network switch has a small inbuilt memory that it uses for high speed lookup operation by matching of binary content. This memory is called Content Addressable Memory, in short CAM. In CAM all filtering conditions are represented as binary values and any search criteria is matched by its corresponding binary values[46]. TCAM is a special type of CAM, that allows for a third condition other than 1 or 0, which is called don't care and represented by * in the entry and it can match with any value 0 or 1. The third condition allows to ignore some bits of the search criteria, and therefore only a part of the whole binary content needs to match with the condition. This makes it possible to represent wild-card masked IP addresses as filtering conditions in which only the masked number of bits are needed to be matched with binary representation of IP address of an incoming packet. Figure 3.1 shows the matching operation in TCAM. In the TCAM entry table some wild-card mask IPs in their binary format are set as match conditions, and on matching with the IP address of an incoming packet, it returns the port number to which the packet has to be sent out. As shown in the figure, the first entry is binary representation of a 8 bit wild-card mask IP address, which means only the first 8 bits of a packet needs to match with this entry, and then the packet will be sent out to port 4. Therefore, this matching can also be called as prefix matching since if the bits set as the condition to match, appear as a prefix of the binary representation of the search criteria, then the condition is always fulfilled. It is also important to mention that a packet can match multiple conditions in the TCAM entry table. This property of prefix matching is extensively used in PLEROMA for matching of event values with subscriptions, that will be discussed in the next section.



TCAM Entry	Destination Address to match	Output Port
1	11100000**********	4
2	001100**********	2
3	110**********	1
4	01**********	3

Figure 3.1: TCAM matching operation

3.2 PLEROMA

The challenges to create a scalable middleware to handle large number of dynamically changing publisher/subscriber and also to make content based filtering more efficient by using the features provided by SDN, have motivated the development of PLEROMA. It uses the same pub/sub model for SDN as proposed in [23] which means publishers/subscribers send their advertisements/subscriptions with a fixed destination IP address in their packet headers that the switches then forward to the controller. The controller uses covering relation between advertisements and subscriptions to create and maintain end to end path between publishers and subscribers and install the required flows on the TCAM of the switches. The published events are matched and forwarded to the interested subscribers by prefix based matching feature of TCAM. Considering the relevance of the methods used in PLEROMA for this thesis, some of them are discussed in the next few sections.

3.2.1 Transforming contents to match fields

PLEROMA[40] uses the concept of spatial indexing as described in Chapter 2 to generate dz-expressions of multidimensional subscription and advertisement data. The dz-expressions thus generated can be used to represent the value of a subspace in which the content has been subscribed or advertised. Some properties[23] of the dz-expressions:

- The subspace represented by a dz-expression is inversely proportional to the length of

the dz-expression i.e. longer dz-expression represents a smaller subspace.

- With increasing number of dimensions, length of dz expressions increases to represent a fine grained advertisement/subscription.
- The subspace of an advertisement/subscription with dz-expression dz_1 is covered by another advertisement/subscription with dz-expression dz_2, if dz_2 happens to be a prefix of dz_1.

By making use of dz-expressions it is possible to represent advertisement/subscription contents in a coarsely grained or very finely grained manner[41], also since the covering of sub spaces can be determined by just prefix matching of dz-expressions it enables accurate matching for the purpose of covering based routing. For example the sub space of subscription with dz-expression s_1={10} will cover the sub space represented by another subscription having a dz-expression s_2={1011}. An advertisement or subscription can be represented by multiple number of dz-expressions depending upon the range of sub space that the advertisement/-subscription includes. On the contrary, since an event is a point in the content space with particular values for each dimension, therefore it can be represented with one dz-expression. An event matches a subscription when it lies within the sub space covered by a subscription, in dz-expression notation that will mean that an event matches a subscription when the subscription dz is a prefix of the event dz. Now to match a subscription to an event, the subscription dz-expressions need to be converted to a match field compatible with OpenFlow, which can be put in the switch's TCAM. For this purpose IPv4 addresses from a range of multicast IPv4 addresses (225.128.0.0-255.255.255.255)[6] is used, where the dz-expressions can be appended. For example in Figure 3.2 a dz={0100} is appended to the IPv4 address after the first 9 bits and converted to a match field which can be represented as a 13 bit wildcard masked IP address in the TCAM of the switch. In this example an IPv4 address is used , but this procedure can be applied for IPv6 addresses too.

3.2.2 End to end path maintenance and flow installation

To match and route the published events to the interested subscribers in a software defined networking environment, flows need to be installed on all the switches that are involved in the path between a publisher and a subscriber. As mentioned before, PLEROMA uses covering relation between advertisements and subscriptions to install necessary flows. To handle large number of dynamically changing advertisements and subscriptions and also to reduce end to end latency and redundant event forwarding, PLEROMA maintains a set of spanning trees which spans onto every switch in the network following a shortest path. Each advertiser is associated to one or more spanning tree(s) to which it can publish events and all the subscribers interested in the advertised set of data also join the same tree, thus for successful event matching and delivery; flows need to be installed only on the switches which are on the tree. In this way it reduces the total number of flows which are needed to be installed and also reduces multiple forwarding of same events. Using multiple set of spanning trees also helps in load balancing, enabling the system to handle more number of event publishers in parallel. Each spanning tree is represented with a dz-expression that is used for finding out

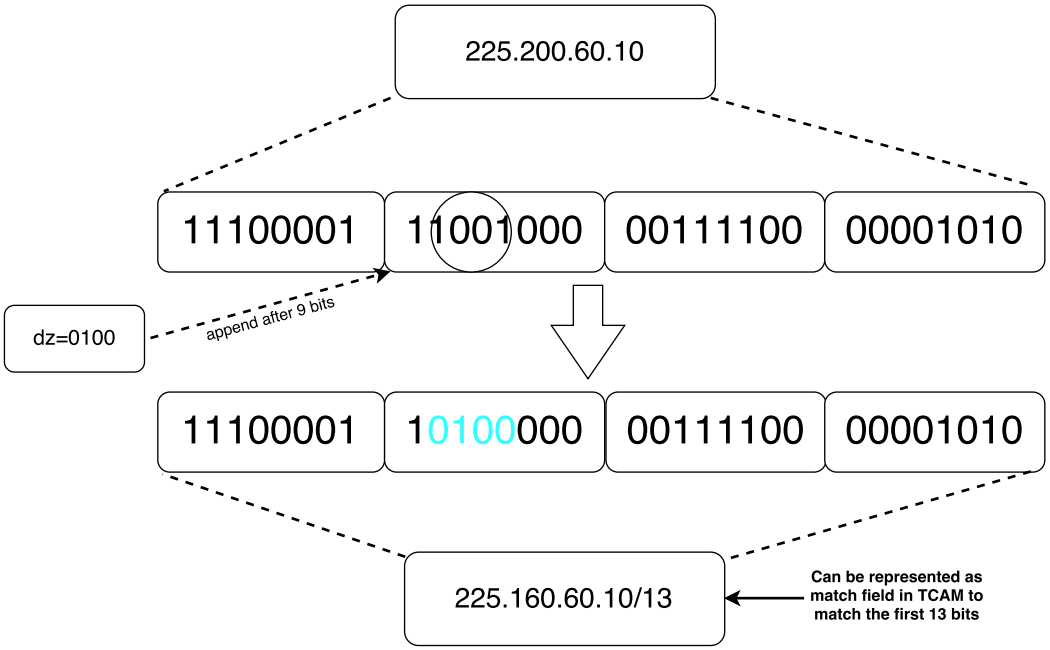


Figure 3.2: Converting dz expression to match field

the responsible tree to which a particular advertisement and subscriptions should join. The details of these methods are discussed in the following sections.

Spanning Tree Creation and Advertisement Handling A publisher sends advertisements including the dz-expressions in its packet header to the switch that it is connected to, with a fixed IP address as the destination address. The switch then forwards those packets to the controller. The controller on receiving an advertisement starts to check to which spanning tree the publisher can join. In particular, the controller checks every dz-expression in the set of dz-expressions that an advertisement has to see if any of them is covered by the dz-expression of any existing tree. If the tree dz-expression covers any of the dz-expression of the advertisement, then the publisher joins that tree. Figure 3.3 shows an example, where there are three publishers involved in the topology. Assuming that these three publishers advertise in a chronological order, i.e. in the order P_1,P_2,P_3. When P_1 advertises there are no existing tree before, therefore a new spanning tree is created with the switch R_1 as the root and having the dz of {10}. Next when P_2 advertises with the dz-expression {100}, the controller sees that this dz is covered by the existing tree by prefix matching operation, and thus P_2 joins the existing tree. The advertisement published by P_3 covers the existing tree, therefore it is split into two parts {10}, and {11}, with the first part it joins the existing tree and with the second part a new spanning tree is created with R_4 as the root of the tree. Subscriptions are treated in a similar way which is described in the next section.

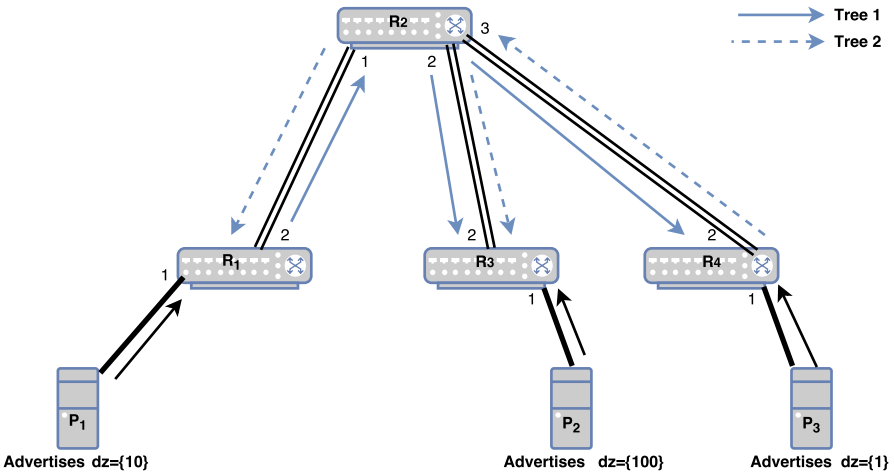


Figure 3.3: Handling advertisements in PLEROMA

Handling subscriptions and installing flows On receiving a subscription the controller checks every dz expression of the subscription and searches to which trees the subscriber can join. If a dz-expression of the subscriber is covered by the dz-expression of an existing tree, then the subscriber joins the tree. In this way overlapping advertisements and subscriptions join the same tree. If a subscriber dz is not covered by any of the existing trees, then the controller stores it and checks again when a new advertisement triggers a new tree creation process or the dz of an existing tree changes. Now there can be three conditions under which a path between a publisher and a subscriber needs to be considered,

1. The dz-expression of an advertisement covers the subscription dz, $dz_{sub} \prec dz_{advt}$
2. The advertisement dz and subscription dz are equal, dz_{sub}=dz_{advt}
3. The subscription dz covers an advertisement dz, $dz_{advt} \prec dz_{sub}$

In any of the three cases, the controller needs to find a path between the publisher and the subscriber so that events can be routed to the subscriber. To install flows for a pair of publisher and subscriber, the controller first calculates the path from the publisher to the subscriber in terms of the switches that an event needs to travel and also the output port in each of the switches. The complete path in the tree can be represented by a set of tuples where each tuple consist of a switch and the output port. After selecting the switches and the output ports, the flows are installed on the corresponding switches using the IP addresses as the destination address in match field, which are obtained by appending the subscription dz-expression to the IP address from the multicast range. Figure 3.4 shows an example for flow installation considering one publisher and two subscribers. Again, we assume that the advertisements and the subscriptions happen in a chronological order i.e. P_1,S_1,S_2. So when the publisher advertises, the spanning tree is created with R_1 as the root node. Next, the subscriber S_1 subscribes with the dz-expression {100}, being covered by the tree dz it joins the tree and the controller calculates the path from the publisher to the subscriber and install the corresponding flows on the the switches R_1,R_2,R_3 with destination IP 225.192.70.10/12

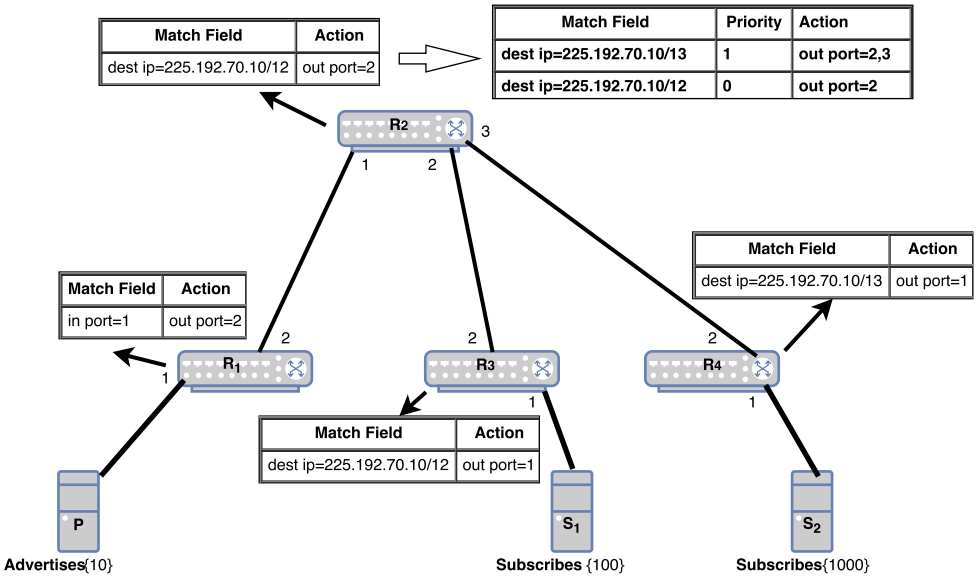


Figure 3.4: Flow installation in PLEROMA

in the match fields of R_2 and R_3. Subscriber S_2 then subscribes with a dz-expression {1000} which is also covered by the same tree. Therefore, a flow with destination IP 225.192.70.10/13 in the match field of switch R_4 is installed. To install the corresponding flow on the switch R_2, the controller finds that an event matching the dz-expression {1000} of S_2 will always match with the dz {100} of S_1. So such an event should be forwarded to both of the subscribers, but an event matching only the shorter dz {100} should be forwarded only to subscriber S_1. Therefore it changes the priority field of the existing flow to a lower level and install a new flow with higher priority to forward an event both of the subscribers. Making use of the priority field ensures that an event arriving at R_2 will be checked for prefix matching for both subscriptions first, and if there is no matching then prefix matching will be checked only for S_1. Although not shown in the Figure, there needs to be additional actions in the flows of R_3 and R_4 to overwrite the destination mac address and destination IP address of a packet containing an event to that of the subscribing hosts.

As discussed in the previous sections, the content filtering technique used in PLEROMA depends upon the expressiveness offered by the dz-expression representation of the content. As the IP address length is limited to a certain number of bits, therefore only a certain length of dz-expression can be appended to it. Length of dz-expressions increases with increasing number of attributes/dimensions and after a certain length, the content filtering rules start to overlap and fail to provide the expressiveness needed to efficiently match and filter contents. This introduces false positives in the network and wastes valuable bandwidth. This problem has been identified and initial solutions have been proposed in [7] to represent contents based on workload on the system and also to remove irrelevant attributes so that relevant attributes can be represented more expressively. However, the solutions have been proposed on a simulated publish/subscribe filtering environment and their performance on an actual SDN based

pub/sub system are yet unknown.

3.3 Problem Statement

As explained in the previous section, the solutions to make content filtering more expressive for a pub/sub system are yet to be explored for an actual SDN environment. This thesis tries to solve this problem and maps the proposed solutions in [7] to an SDN environment and explores the feasibility and scalability of the proposed techniques. This thesis uses PLEROMA as the middle ware and integrates the proposed techniques to it, while also evaluating the effectiveness of the techniques used and measuring the overhead of the controller in order to reconfigure the existing rules in the switches.

Chapter 4

Workload Enabled Indexing

The objective of this chapter is to introduce the techniques which can be used to make content filtering in an SDN based pub/sub model more expressive and also to provide the implementation details of mapping the techniques to a centralized controller based SDN environment. The term workload refers to the amount of subscriptions and events in the system, and the effectiveness of the techniques used vary according to the type and distributions of the workload used.

4.1 Limitations of Existing Content Filtering Method

Till now we have discussed how content filtering in an SDN based pub/sub system is achieved by spatially indexing the multi dimensional content space and representing the contents in the form of binary strings called dz-expressions. With increasing number of connected devices and depending on the type of information exchanged there can be many attributes/dimensions for a pub/sub system. Also we have seen in the previous chapter that the length of the dz-expressions to represent a fine-grained subspace increases with increasing number of dimensions in the content space. Due to the limited number of bits available in IP address for appending the dz-expressions, it becomes a challenge to maintain the same level of expressiveness in the match fields required for efficient content filtering. For example if there are two subscriptions S_1,S_2 with dz-expressions {10} and {100} respectively, by property of the dz-expression S_2 has a subscription for a finer grained subspace as compared to S_1. But, if only 2 bits are available in the IP address where the dz-expression can be appended, then S_1 and S_2 both will be represented by {10}. As a result of this,S_2 will receive events corresponding to {10} which is not what it actually subscribed for. This introduces unnecessary traffic in the network and increases the amount of false positives for the end subscribers.

To reduce the amount of false positives and save network bandwidth, the solutions for the above mentioned problem have been stated in [7]. The solutions improve the existing spatial indexing method by selecting the most relevant space to index based on the subscriptions and also removes irrelevant attributes based on the different parameters like event distribution, subscription overlapping, correlation among attributes, so that the relevant attributes can be represented more expressively. The methods are discussed in details in following sections.

4.2 Minimizing Content Space

To reduce the amount of false positives in a content based pub/sub, there has been previous attempt like [33] where the subscriptions are clustered based on their similarity to determine multicast groups where the published events can be broadcast. Similar concept is applied in [7] to divide the subscriptions into clusters and then perform spatial indexing only on those clusters and thus minimizing the content space to be indexed which in turn reduces the number of bits required to represent a subscription. To understand this process better we need to understand how is clustering done for a dataset and what is achieved by clustering.

4.2.1 Data Clustering

Cluster analysis is done to find patterns for data points where there is little or no prior knowledge about the data points. It divides the data points into groups or clusters based on their similarity. A clustering technique is efficient when points within the same clusters have greater level of similarity and points in different clusters have greater level of difference. Cluster analysis finds its application in many fields like data mining, document retrieval, pattern recognition[19] etc. Figure 4.1 shows an example of clustering in two dimensional content space where the similar data points are being clustered into groups. One of the most

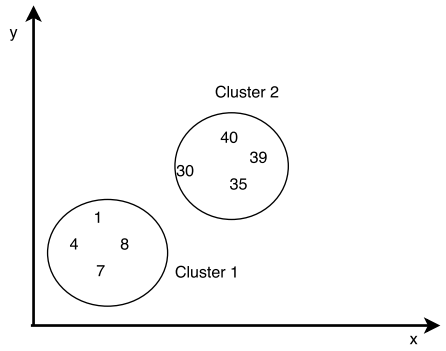


Figure 4.1: Clustering of data in two dimensions

popular clustering technique used is *K-Means* clustering. Figure 4.2 shows the algorithm for K-Means clustering, it starts with a user assigned number of clusters K. Then it initiates K number of centres or centroids for the clusters. In the next step it forms K number of clusters by assigning the centroids to its nearest data points. K-Means clustering achieves the similarity based grouping by calculating the minimum euclidean distance of the point and the centroid of the cluster. In the next step the algorithm recalculates the centroids of all the clusters, which can be expressed as an optimisation problem to minimize the sum of squared error[38, Chapter 8]. Specifically for each cluster it tries to solve for,

$$Minimize(\sum_{i=1}^{k}\sum_{x\in c_i}(x-\alpha_i)^2) \tag{4.1}$$

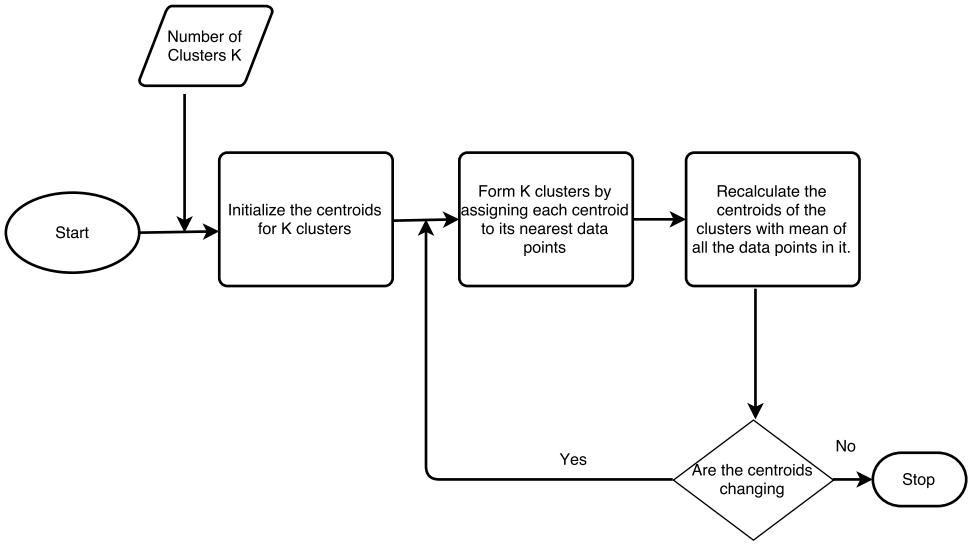


Figure 4.2: K-Means clustering algorithm

where k is the number of clusters, and x is the set of data points belonging to the cluster c_i, and α_i is the centroid of the cluster. It picks up the data points which returns the minimum value after solving the problem and recalculates the cluster centroid by taking mean of the data points. This process continues till the centroids have a constant value and do not change any further. The algorithm then terminates, having divided the data into K clusters based on their similarity.

K-Means clustering is one of the approaches used in [7] to cluster the subscriptions, although there can be other clustering approach like the R-Tree as well. After dividing the subscriptions into clusters, the clusters are bounded by minimum closing approximations which can be represented by rectangular structures called minimum bounding rectangles(MBR). Minimum bounding rectangle is a concept that can be used to bound geographic data with many vertices into a minimum area rectangular space[8] which can be represented in terms of two coordinates X_{min},Y_{min} and X_{max},Y_{max}. Figure 4.3 shows an example how the the subscriptions are clustered and bounded using the MBRs. Now that the MBRs contain all the subscriptions, it is possible to do the spatial indexing only within the MBRs and not in the whole content space. Leaving out the unnecessary parts of the content space in this way is advantageous because it enables to represent the meaningful spaces with more expressiveness.

In addition to performing spatial indexing within MBRs another approach mentioned in [7] to increase expressiveness is to remove attributes/dimensions which are not important in performing spatial indexing. Since with increasing number of dimensions the length of dz-expressions increases therefore leaving out unnecessary dimensions will help to represent the more important ones with more expressiveness. To find out the importance of a dimension in its capability to filter events or to minimize false positives, many factors need to be considered.

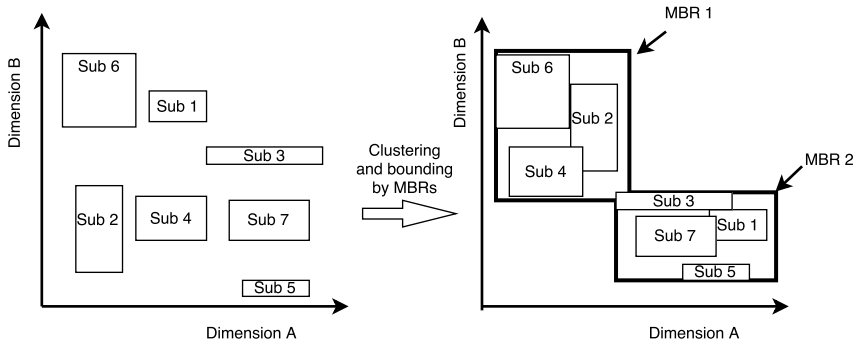


Figure 4.3: Clustering of subscriptions and bounding the clusters with MBRs

Therefore [7] introduces few algorithms to evaluate the importance of a dimension and consider only those dimensions to calculate the dz-expressions. Since this thesis maps and evaluates these techniques in SDN platform, therefore it is necessary to know the details about the underlying algorithms.

4.3 Dimension Selection Algorithms

To identify dimensions which are more relevant for event filtering, factors like event distribution in the content space, subscriptions overlapping, correlation among attributes needs to be considered. Therefore three algorithms have been proposed in [7] to select relevant dimensions, which are event variance based selection, subscriptions match based selection and correlation based selection.

4.3.1 Event Variance Based Selection

In Event variance based selection(EVS), dimensions are selected based on the distribution of events along the dimensions. A dimension having a greater distribution of events is more capable in reducing false positives than a dimension with lesser event distribution . Figure 4.4

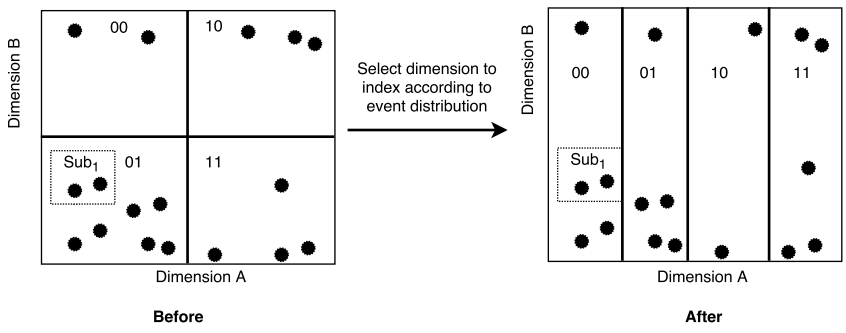


Figure 4.4: Event variance based dimension selection

shows an example for the same. Initially when indexing is performed along both dimensions,

the subscription Sub_1 is represented by the dz-expression {01} and it receives all the events which lie in that space even though the subscription did not subscribe for the whole space. Therefore it receives some events which are false positives for it. In the next step indexing is performed along dimension A having a higher event distribution, Sub_1 is then represented by the dz-expression {00}. In this case also there are some false positives that Sub_1 receives but it is lesser than what it received before when indexing was performed along both dimensions. For data with multiple dimensions, EVS assigns *selectivity factor* to each dimension based on the event variance along that dimension, higher event variance means a higher selectivity factor for a dimension. Then it selects a predefined number of dimensions having the highest selectivity factors from the complete set of dimensions. Spatial indexing is then performed considering only the selected dimensions.

4.3.2 Subscription Overlap Based Selection

When subscriptions are overlapping along one particular dimension, then events along that dimension will match with most of the subscriptions. Therefore if such a dimension is selected for indexing, then false positives are high because events will match with multiple subscriptions along that dimension, whereas an event should be received by a subscriber only when it matches to an event along all dimensions. Figure 4.5 shows a situation when indexing is performed along a dimension where the subscriptions are overlapping, which shows that the false positives for both of them increases. To avoid this problem another algorithm was

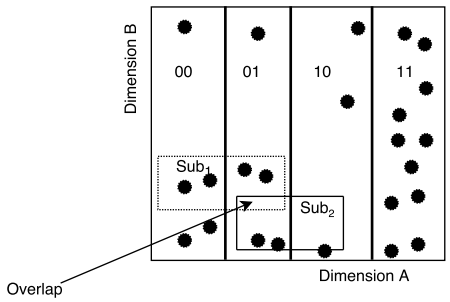


Figure 4.5: Subscriptions overlapping along Dimension A

proposed in [7] which is known as *Event Match Count-Based Selection*. This algorithm assigns selectivity factor to a dimension based on the number of subscriptions that an event matches along that dimension. If an event matches multiple subscriptions along a dimension, then that dimension is marked as less important for indexing purpose. After counting the total number of event to subscription matches along a dimension, this algorithm assigns higher selectivity factor to the dimensions with a lower count. Then it selects the predefined number of dimensions having the highest selectivity factor and performs spatial indexing considering those dimensions. Since this algorithm considers both subscription overlap and event variance to calculate selectivity factor, it offers better performance than EVS .

4.3.3 Correlation Based Selection

Very often, there is some form of correlation among different physical attributes/dimensions[21] which makes selecting all of them for spatial indexing redundant. Due to correlation among dimensions, an event which matches a subscription in one dimension will also match the subscription in the other dimension which is correlated to the first dimension. Because of this reason, selecting both the dimension for spatial indexing becomes redundant when the subscriptions can be represented expressively just by indexing along one of the dimensions. Therefore another algorithm was proposed in [7] that in addition to considering event variance and subscription overlap,also takes the correlation among dimensions into account while selecting dimensions for spatial indexing. To find correlation among dimensions, the concept of covariance[27] matrix is used. For example if the data is 3 dimensional, then a 3x3 covariance matrix is generated where an element at the position (1,2) will represent the covariance between dimension 1 and dimension 2. If there is some form of correlation between those two, then the covariance is a non zero element. The diagonal elements of the covariance matrix holds the selectivity factor of each of the dimension. To find the covariance between two dimension pair, the algorithm calculates the number of subscriptions that an event matches along a dimension and assigns a *similarity factor* to each dimension pair based on the number of subscriptions that an event matches along both of the dimensions. The similarity factor is then aggregated for all the events and the covariance of the dimension pair is calculated by reversing the similarity factor. This means a higher covariance value for a dimension pair having a lower similarity factor. After calculating the covariance matrix, the dimensions with highest covariance or lowest similarity are selected by applying the concept of principal component analysis[36] on the covariance matrix. Then spatial indexing is performed on the set of the selected dimensions. The correlation based selection algorithm offers better performance than EVS and EMCS in terms of its capacity to reduce false positives[7], however with the cost of increased computational complexity.

Another algorithm called *Greedy Selections* is introduced in [7] which calculates false positive rates at each step with a particular set of dimensions and removes a random dimension at each step. It returns the set of dimensions which provides the lowest false positive. Although this approach performs even better than CS in its capacity to reduce false positive, it has a very high time complexity and therefore may not be feasible in a live network.

Till now this thesis has discussed how a content based pub/sub system can be made more expressive by performing spatial indexing only on selected content space and by removing redundant attributes. Now to make these techniques work on an actual SDN platform is the main contribution of this thesis, and the next section discusses the implementation details of mapping the techniques to work with an SDN based pub/sub system.

4.4 Implementing Workload Enabled Indexing in SDN

For implementing the workload enabled indexing techniques in SDN, Floodlight was used as the controller and Mininet was used to emulate the underlying network. The PLEROMA middleware runs as a module in the floodlight controller, another module was added to the controller to handle the functionalities for enabling the workload based indexing.

4.4.1 Floodlight Controller

Floodlight[2] is an OpenFlow based controller written in Java. Floodlight provides static flow entry pusher service for installing and deleting flows on the underlying switches via Java applications. It also provides thread pool services for running multi-threaded applications in parallel. The application modules communicate to the controller modules via the Java API exposed by the controller. Figure 4.6 shows the structure of the controller, the coloured modules are default modules provided by Floodlight controller which are loaded when the controller starts up.

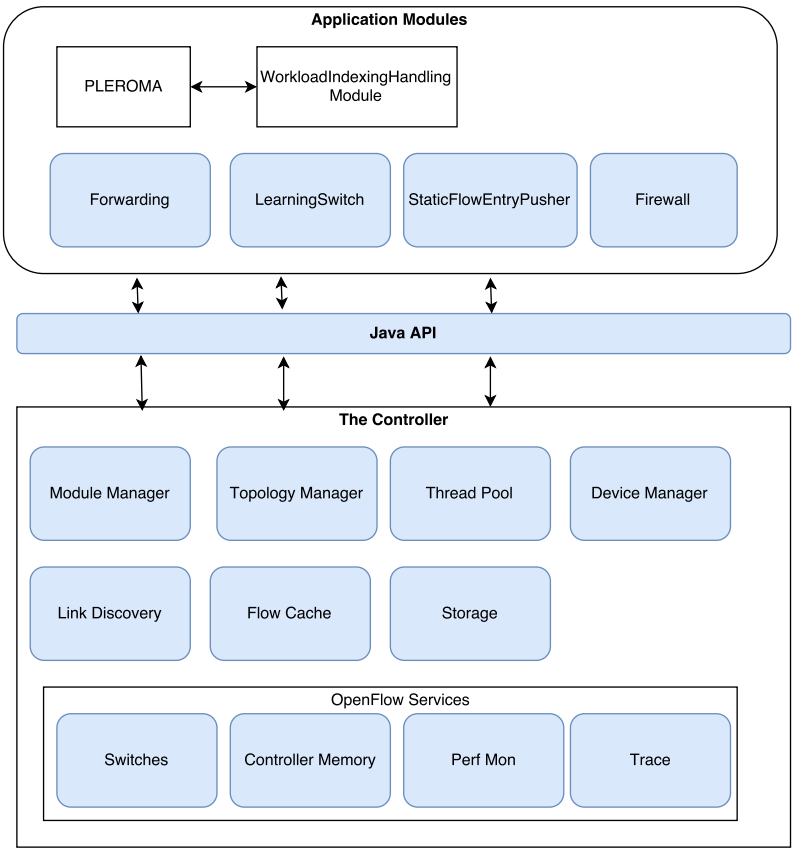


Figure 4.6: Controller and Module Structure

4.4.2 Mininet

Mininet[4] is a network emulator which can emulate a real network with OpenFlow enabled switches having kernels similar to hardware switches. Mininet uses OS virtualization concept to emulate OpenFlow enabled switches and it is equivalent to using a hardware switch, so all the experiments and simulations that are done by using Mininet can be deployed on a real network as well.

Combining the Floodlight controller's OpenFlow services and Mininet's OpenFlow capable switches, it makes it very easy to set up a software defined networking environment with all the features and functions of a real network. The Floodlight modules are written using Java, while the publisher/subscriber run using Python scripts. The dimension selection and workload indexing runs as a separate process and the communication between it and the Floodlight module is facilitated using sockets. Figure 4.7 shows the complete overview of the system with all the entities involved. To enable the workload based indexing on the data

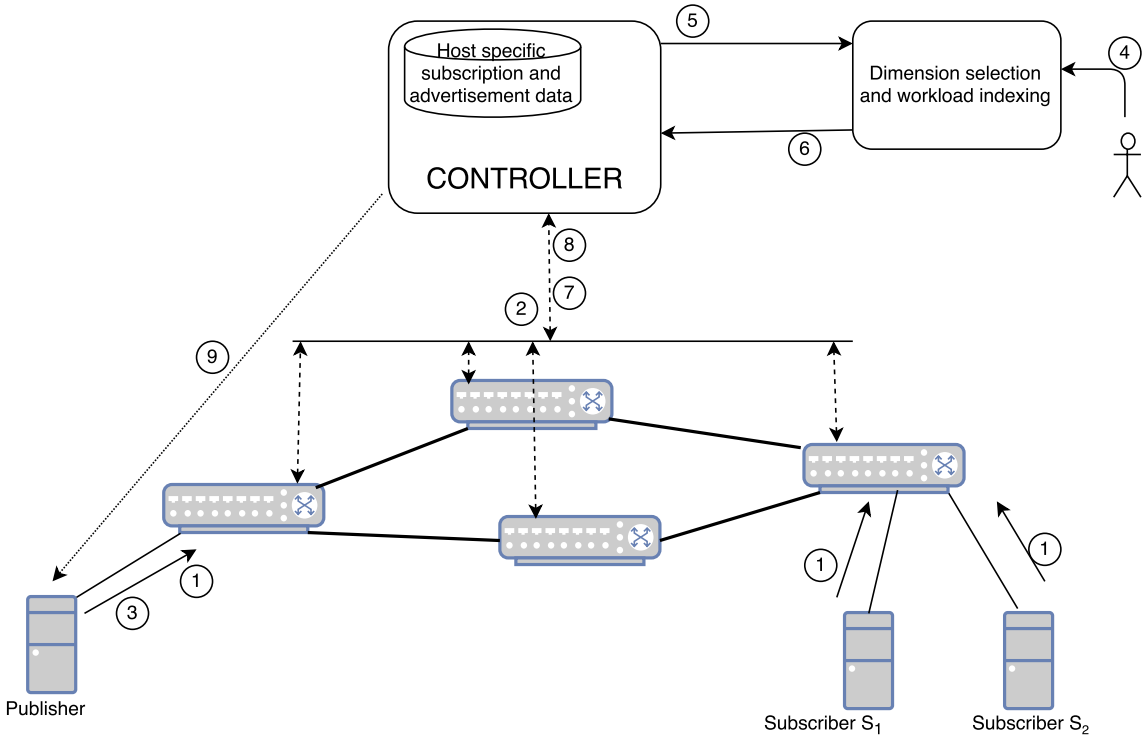


Figure 4.7: Implementation of Workload Enabled Indexing Techniques

plane of the network, the controller has to take care of the following items,

- Preserving the subscription and advertisement values for each host so that they can be used in future with the newly generated dz-expressions using workload based indexing.
- Mapping the advertisement/subscription and event values to the data structure needed for the purpose of workload indexing and dimension selection.

- Deleting the old flows from all the switches in order to install new flows using the new dz-expressions.
- Installing new flows on the switches with the newly generated dz-expression in a small amount of time to make the system ready for future communications.

To explain the process better, from Figure 4.7 each of the steps involved can be explained as below,

1. Each host acting as publisher/subscriber send the respective advertisements/subscriptions by using a Python script. The advertisements/subscriptions are encoded into IPv4 packets with a fixed multicast IP address as the destination address and sent using UDP sockets. This is explained using the algorithm 1. The parameter *values* contain the lower and higher values of this particular advertisement/subscription along each dimension e.g. low_0,$high_0$,low_1,$high_1$ for 2 dimensions and DZs contain all the dz-expression corresponding to this advertisement/subscription. The publisher sends its advertise-

Algorithm 1 Sending advertisements and subscriptions

```
procedure SUBSCRIBEORADVERTISE(values, DZs)
    sock ← createSocket()
    for each value ∈ values do
        for each dz ∈ DZs do
            packet ← createPacket(value, dz, IP_multicast)
            sock.sendPacket()
```

ments and the subscribers send their subscriptions to the corresponding switches that they are connected to. The switches being unaware about the destination address of such packets, send them to the controller. A multiple number of spanning trees can be set in the controller which have their own independent threads running to calculate paths and install flows on the switches.

2. The controller on receiving the packets, checks its destination address and if it matches to the multicast IP address that an advertisement/subscription should have, it starts with the calculations for finding path and flow installation process on the switches as described in Chapter 3.

3. Once all the flows are installed the publisher publishes the events by creating IPv4 addresses from the dz-expression of each event and encoding it as the destination address in the packet containing the event so that they can be matched and forwarded on the switches using the flows installed. Since the controller also needs the events for carrying out dimension selection and indexing process, another thread was added to the publisher that sends the events to the controller using the same multicast address that was used for advertisements/subscriptions. Before publishing any event the publisher also checks if the event is covered by the set of advertisement that it already advertised, if only the event is covered by an advertisement then it is published. This is done in order to reduce unwanted traffic in the network.

Once the subscriber receives the events and false positive is calculated, they can inform to the controller about the false positives that they have. Therefore the controller decides to perform dimension selection and workload indexing to reduce the false positive. The controller now has all the advertisements, subscriptions and events. For workload based indexing and dimension selection it has to find the corresponding dimensions from the data that it has and also has to map the advertisements,subscriptions and events to the corresponding data structure needed for workload indexing and dimension selection. The workload based indexing and dimension selection treats advertisement/subscriptions as Participant objects, events as Event objects and dimensions as Dimension object. Although it does not need advertisements for workload indexing and dimension selection, the advertisements are sent to it because the new dz-expressions for advertisements are also required for reinstalling the flows after dimension selection and workload indexing. The controller also stores the advertisements/subscriptions data for each host so

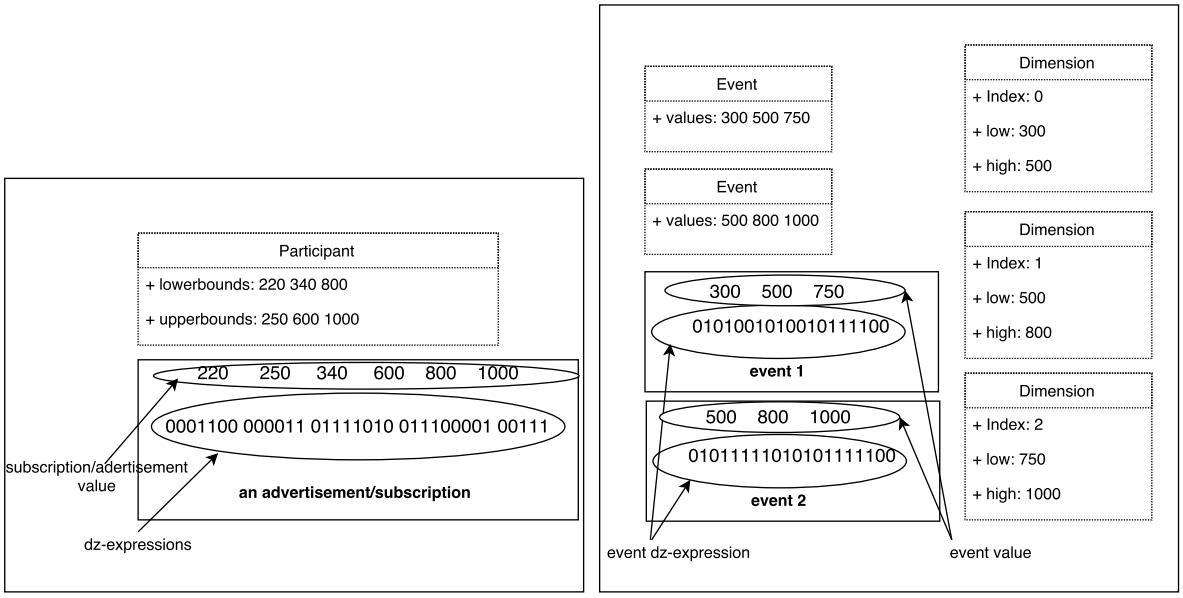


Figure 4.8: Mapping advertisement/subscription to Participant class and Events to Event class and finding Dimension from events

that they can be used in future for reinstalling the new flows with new dz-expressions generated by workload based indexing. Figure 4.8 shows the structure of advertisement/subscription and event message, and how they are mapped to Participant class, Event class and Dimension class respectively for workload indexing and dimension selection. Three dimensional data is used in this example, and therefore three dimensions with indices 0,1 and 2 are constructed out of it. Even though only two events are used in this example, the lower and upperbounds of the dimensions are extracted after all the events have been received and the controller can sort and select maximum and minimum values of each dimension from all of the event values.

Algorithm 2 Mapping Subscription to Participant

```
1: procedure CreateParticipant(values, host)   ▷ values containing the subscription values and host is the publisher/subscriber
2:     for (int i=0;i<values.length;i++) do
3:         If(i%2==0)
4:         lowerbounds[i/2] ← values[i]
5:         Else
6:         upperbounds[i/2] ← values[i]
7:         end If
8:     end for
9:     Participant participant ← new Participant(lowerbounds, upperbounds)
10:    subToHostMap.put(participant, host)
11:    return
```

A subscription is converted to a Participant object following the algorithm 2 and stored in the controller. This algorithm also creates a hash-map of host and subscriptions to store all the corresponding subscriptions of a host. The reason for storing this information is that, after workload based indexing we need to construct new advertisement/subscription request for each host keeping everything else same except adding the new dz-expressions. For advertisements it follows the same algorithm for mapping the advertisements to a participant. When the controller receives any event, it maps those to the Event objects using the algorithm 3 and also creates Dimension objects from the event values. For creating dimensions, an array of lists is created where each list can hold values of each event along that dimension, and on receiving an event the values along each dimensions are added to the corresponding lists as shown in the line 4 of algorithm 3. The procedure *CREATEDIMENSIONS* then finds the minimum and maximum from each lists and create the Dimension objects accordingly.

Algorithm 3 Event handling

```
1: procedure CreateEvents(eventvalues)
2:     for (int i=0;i<eventvalues.length;i++) do
3:         values[i] ← eventvalues[i]
4:         dimension[i].add ← event values[i]
5:     end for
6:     Event event ← new Event(values)
7: procedure Createdimensions
8:     for (int i=0;i<values.length;i++) do
9:         dimensionlow ← minimum(dimension[i])
10:        dimensionhigh ← maximum(dimension[i])
11:        Dimension d ← new Dimension(i, dimensionlow, dimensionhigh)
12:    end for
```

4. Then the dimension selection and workload based indexing process is started with an

user specifying the dimension selection algorithm that it needs to run and also which indexing method should be followed while generating new dz-expressions. Also two sockets are added, one for listening so that it can receive the subscription, event, dimension data that it needs for the process of dimension selection and workload indexing and another for sending the data back to the Floodlight module after completing the process.

5. Once the controller has all the subscriptions, events, dimensions and advertisements it sends them to the workload based indexing and dimension selection process via a socket and also listens for incoming data.

6. Figure 4.9 shows the complete work-flow of dimension selection and workload indexing. Upon receiving the data from the floodlight module, it reads the properties set by

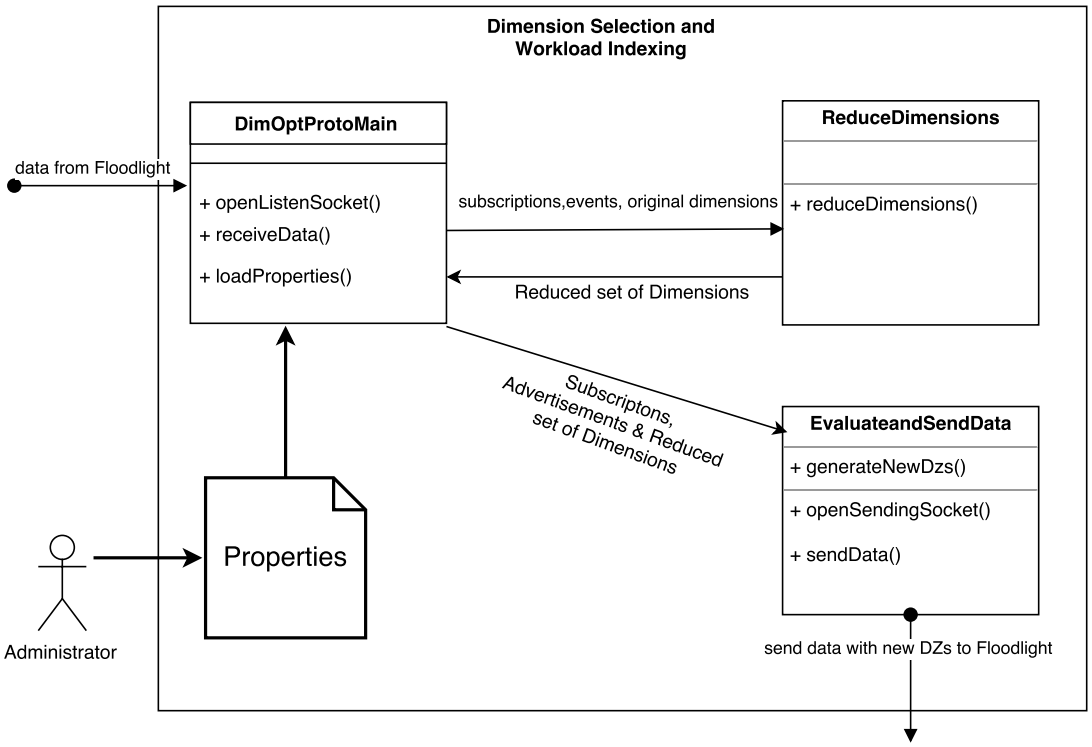


Figure 4.9: Workflow of dimension selection and workload indexing

administrator and performs dimension reduction following a particular algorithm(EVS, EMCS or CS, GS) which returns the set of selected dimensions from the set of complete dimensions. Then considering the selected dimension it generates new dz-expressions by applying spatial indexing after K-Means clustering or spatial indexing without clustering. The data is returned in the form of hash-maps containing the advertisement/subscription as the key and the corresponding newly indexed dz-expressions as the value. The controller on receiving the data, constructs new dz-host tuples using the host information saved before. The steps 5 and 6 can be explained using the algorithm 4. The parameter *data* contains all the subscriptions, events, dimensions and advertisements.

In this algorithm, the line numbers 6 and 11 give the subscribing and advertising hosts

Algorithm 4 Send and receive data

```
1:  procedure SendandReceive(data)
2:      socket ← createSendingSocket()
3:      socket.send(data)
4:      openlistensocket()
5:      On Subscription Receive(<Participant,DZs>)
6:      hosts ← getallSubscribingHosts(Participant)
7:      for each host ∈ hosts do
8:          for each dz ∈ DZs do
9:              subTuples ← newdzhosttuple(dz, host)
10:     On Advertisement Receive(<Participant,DZs>)
11:     advthosts ← getallAdvertisingHosts(Participant)
12:     for each host ∈ advthosts do
13:         for each dz ∈ DZs do
14:             advtTuples ← newdzhosttuple(dz, host)
```

from the map used in algorithm 2. The dzhosttuple is an internal data structure used in PLEROMA for the purpose of associating every unique dz expression and the corresponding host that sent it. In the line number 9 and 14 of algorithm 4 we construct the tuples considering the new dz-expressions received after dimension selection and workload indexing.

7. After constructing new dzhosttuples the controller needs to delete the old flows so that it can install the new flows with the new DZs. The algorithm 5 describes this process,where T_s is the set of all the existing spanning trees in the network. For deleting the old flows, it iterate over all the spanning trees in the existing topology and remove all the existing advertisers and subscribers that joined the tree, as shown in the line numbers 6 and 8. Also all the flow objects that were added to the tree are deleted. After that flows are deleted from the switches using the static flow entry pusher service of Floodlght as shown in the line number 10.

8. After deleting the old flows, the controller is ready to install the new flows. But now since we do not have any publisher/subscriber sending any data, so the controller needs to take care of creating a request that seems like it came from an actual publisher/subscriber so that the PLEROMA middleware can continue with the flows installation process. This process is done using the algorithm 6. The parameters of this algorithm are the subscription and advertising tuples where each tuple consists of one dz-expression and the corresponding host whose subscription is represented by that dz-expression. These tuples have been created in the algorithm 4 after receiving the new dz-expressions from dimension selection and workload indexing module. The new flows installation process should happen as fast as possible to avoid any information loss if the publisher publishes any event. Therefore two threads were created using Java's ScheduledExecutorService[3] that can create and send advertisements/subscriptions in parallel, as shown in the line

Algorithm 5 Delete advertisement/subscriptions and flows

```
1: procedure deleteOldFlows
2:     for each t ∈ T_s do
3:         Tuples_sub ← t.getSubTuples()
4:         Tuples_advt ← t.getadvtTuples()
5:         for each subTuple ∈ Tuples_sub do
6:             t.removeSubscriber(subTuple)
7:         for each advtTuple ∈ Tuples_advt do
8:             t.removeAdvertiser(advtTuple)
9:         t.removeAllFlows()
10:    StaticFlowEntryPusherService.deleteAllFlows()
```

Algorithm 6 Install new flows

```
1: procedure installNewFlows(subTuples,advtTuples)
2:     timertask[] = new TimerTask(2)
3:     TIMERTASK1.start()
4:     TIMERTASK2.start()
5:     procedure timertask1
6:         for each advtTuple ∈ advtTuples do
7:             host ← advtTuple.getHost()
8:             dz ← advtTuple.getDz()
9:             host.parameters ← getParameters(host)   ▷ find parameters like switch port,
   host mac, host ip from previously stored information
10:            T_s ← findResponsibleTrees(dz)          ▷ finds all trees such that dz ≺ dz_tree
11:            for each t ∈ T_s do
12:                t.process(dz, host)
13:    procedure timertask2
14:        for each subTuple ∈ subTuples do
15:            host ← subTuple.getHost()
16:            dz ← subTuple.getDz()
17:            host.parameters ← getParameters(host)
18:            T_s ← findResponsibleTrees(dz)
19:            for each t ∈ T_s do
20:                t.process(dz, host)
```

number 2. For constructing a new advertisement/subscription request, the required parameters like host IP, host mac, switch port that the host is connected to etc. are obtained by using the saved information for the host in algorithm 2. And then it searches for all the trees where the publisher/advertiser can join based on the same prefix check that was done before, and submits the dz and host to the tree for further processing. The flow installation is taken care of after that as part of the PLEROMA middle-ware.

9. In the last step, the publishers must also be informed about the change of the schema so that the publishers can also use dz-expressions considering only the reduced set of dimensions returned after dimension selection and workload indexing. Only in this way it would be possible to match the event dz-expressions with the newly created subscription dz-expressions. Therefore the controller informs the publisher about the change and ask it to consider the reduced set of dimensions for dz-generation when it publishes events in future.

Till now this thesis has described the implementation process for enabling workload based indexing techniques on an actual SDN environment. In the next chapter, this thesis evaluates the feasibility and effectiveness of the proposed workload based indexing techniques to reduce false positives in a software defined networking environment.

Chapter 5

Evaluation of Workload Enabled Indexing in SDN environment

The goal of this chapter is to provide the details about the evaluation platform and evaluation results of workload enabled indexing techniques in a software defined network. Specifically, evaluations are performed to see how scalable and efficient the proposed solutions can be in terms of their capacity to reduce false positives, and also to know the overhead on the controller in order to reconfigure the system by deleting old flows and installing new ones.

5.1 Evaluation platform

As discussed before, Mininet is used to emulate a network with OpenFlow enabled switches and Floodlight is used as the controller. For all the evaluations, Mininet and the controller are hosted in a Linux system having a processor of 3.2GHz×4 and 8GB memory. As shown in Figure 5.1, a simple Mininet network can be created by the command *sudo mn* with one switch and two host and having a default controller. To integrate Mininet with an external

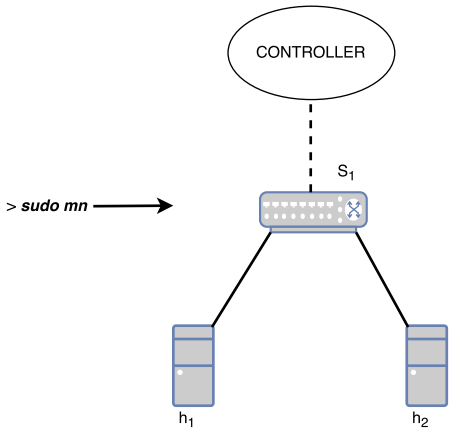


Figure 5.1: Simple Mininet Topology

controller like the Floodlight controller additional options can be specified in command line. For the evaluation of this thesis, a tree topology is used consisting of 63 switches and having

64 hosts, where any host can have the role of a publisher or a subscriber. To create the given topology with all switches connected to Floodlight controller, the below command is used,

```
$ sudo mn --topo tree,depth=6,fanout=2 --controller=remote,
  port=6634,protocols=OpenFlow13
```

Here, the depth means the number of level of switches in the topology and fanout means how many child nodes can one parent node have. Figure 5.2 shows the evaluation topology generated by using the above command. Using such a network with large number of hosts can

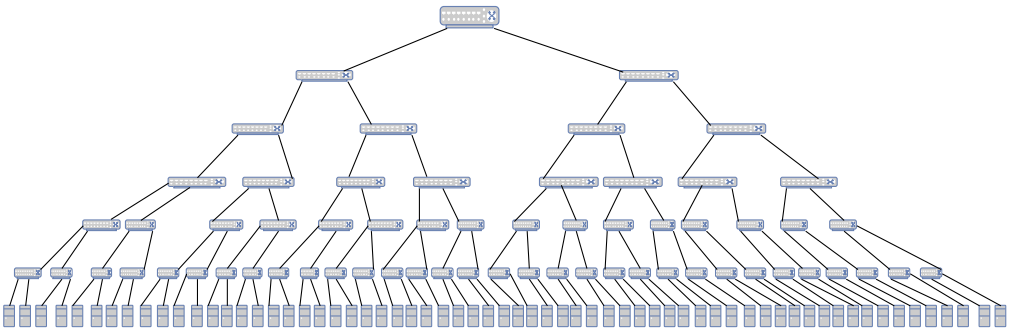


Figure 5.2: Evaluation Topology

prove that the implemented techniques are scalable. The advertisements and subscriptions are divided among different hosts.

This thesis evaluates both the false positive rate and false positive count for each of the subscribing hosts. A false positive occurs when the host has received an event whose value is not covered within the bounds by any of the subscriptions made by that host, so the only way it has received this event is because a dz-expression of the subscriptions must have a prefix match with the dz-expression of the event. For each of the received events by a host, this check is done by comparing it to the subscriptions it made. As explained in the algorithm 7, each event is checked if its covered within the subscription bounds, and then the event dz-expression is checked with all the subscription dz to check if there is a prefix match. If the event is covered and as well as there is a prefix match with a subscription dz, then the event is a true positive, otherwise if there is only a prefix match but the event is not actually covered within the subscription bounds, then it is registered as a false positive. After calculating false positive and true positive for each host, the total amount of false positives and true positives are counted by summing them. False positive rate is then calculated by using following equation,

$$FalsePositiveRate = \frac{FalsePositiveCount}{FalsePositiveCount + TruePositiveCount} \times 100 \qquad (5.1)$$

For the experiments Zipfian distributed data with 8 dimensions have been used. In Zipfian distributed data, subscriptions and events are located in certain number of *buckets*, and the distribution can be changed by changing properties like number of buckets, bucket size etc. In the following sections, the results of different experiments carried out will be discussed.

Algorithm 7 False Positive Calculation

```
procedure CALCULATEFALSEPOSITIVE(event, eventDz, subscription, subDZs)
    eventCovered ← subscription.covers(event)
    for each subDz ∈ subDZs do
        If(eventDz ≺ subDz)
        {
        eventDzCovered ← true
        break
        }
    If(eventCovered)
    {
     If(eventDzCovered)
    truepositive++
    }
    Else
    If(eventDzCovered)
    falsepositive++
```

5.2 Evaluation Results

5.2.1 False Positive

To evaluate the effectiveness of the techniques in their ability to reduce false positives on an SDN platform, the experiments have been conducted in a particular way,

- First, dimension selection and workload indexing is performed on a set of Zipfian distributed data to find the set of dimensions that produce the minimum amount of false positives, and then flows are installed using the dz-expressions created considering those dimensions.
- False positive is measured with increasing number of published events.
- New set of data is generated by changing properties of Zipfian distribution. After publishing 5000 events, the publisher publishes events having a different distribution than the previous one, then the subscribing hosts receive events matched by the flows that were installed using the previous distribution of data and false positive is measured.
- The controller notices the change of distribution and after 8000 events, workload indexing and dimension selection is performed again considering the new distribution of subscriptions and events, new flows are installed and false positive is measured with increasing number of events.

Considering the evaluations done in [7], for all experiments correlation based dimension selection algorithm was used to select dimensions. The experiments have multiple runs and then the average has been taken of all the runs to arrive at the final results. In each run, different dataset is used by changing a random seed used while generating data.

To evaluate the effect of workload based spatial indexing compared to normal spatial indexing, two experiments have been conducted with same set of data,

1. Using CS to select dimensions and then performing normal spatial indexing on those dimensions to generate new dz-expressions, i.e CS with SI
2. Using CS to select dimensions and perform K-Means clustering of the subscription on those dimensions and then generate dz-expressions by spatial indexing only within the MBRs as explained in Chapter 4, i.e. CS with K-Means+SI.

The result of these two experiments are compared in Figure 5.3.

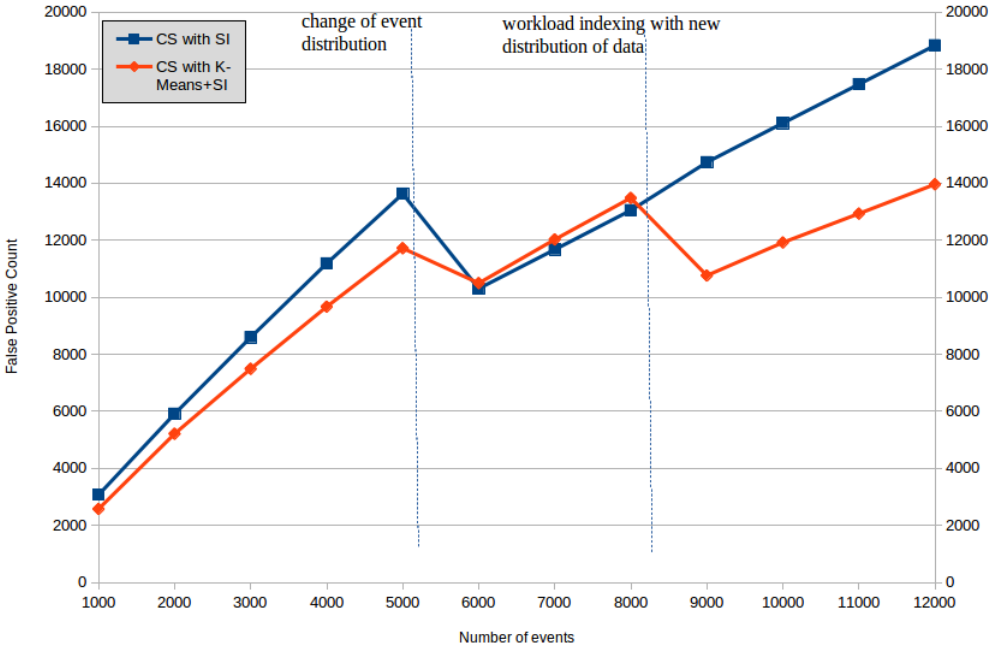


Figure 5.3: False Positive Count Comparison

The graph shows that with increasing events, the number of false positives are increasing, and changing the distribution of events causes a decrease in number of false positives as received by the end hosts. As seen from the graph, after performing workload based indexing considering the new distribution of data, CS with K-Means+SI performs significantly better in terms of its capability to reduce false positive count, whereas performing normal spatial indexing is not able to reduce the count of false positives. So it can be said that using workload based indexing to generate dz-expressions performs better than using normal spatial indexing in terms of ability to reduce false positives.

However, only false positive count can not reflect the performance of using a particular schema. It also depends upon the count of true positives, so false positive rates were also

measured for this experiment and Figure 5.4 shows the false positive rate when CS with SI was used. As seen from the graph, the change of distribution introduces a huge increase in

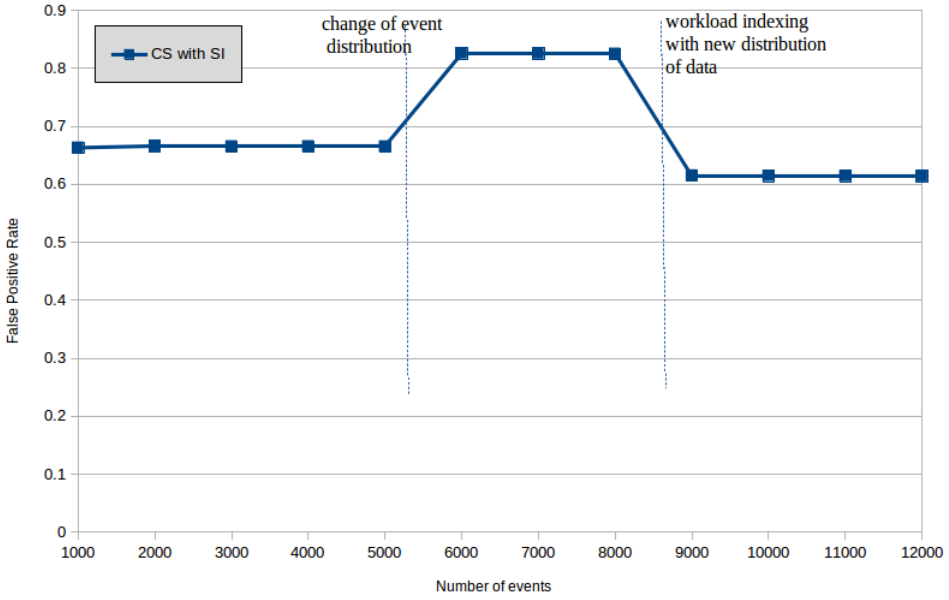


Figure 5.4: False Positive Rate for CS with spatial indexing

false positive rate. Although the count of false positive decreased as seen in the previous graph, the increase in false positive rate is because of huge decrease in number of true positive counts. Performing CS with SI on the new distribution of data is however able to reduce the false positive rate by increasing number of true positives because of considering only a reduced set of dimensions to generate dz-expressions. So CS with SI can decrease the false positive rate even though it could not reduce the actual false positive count.

Considering the performance of CS with K-Means+SI in its ability to reduce false positive counts, another experiment was conducted using different sets of data than the previous experiment. Figure 5.5 shows the false positive counts and Figure 5.6 shows the false positive rate when CS with K-Means+SI was used. The graphs show that changing the distribution of events resulted in increase of false positive count as well as rate. Comparing the result in Figure 5.3 and this result, it can be said that the change of false positive count i.e. increase or decrease after change of event distribution is dependent on the dataset used, however, the false positive rate always increases. After performing workload based indexing on the new distribution of data it reduces both the count and the rate. The false positive rates and counts are both lesser as compared to performing CS with SI.

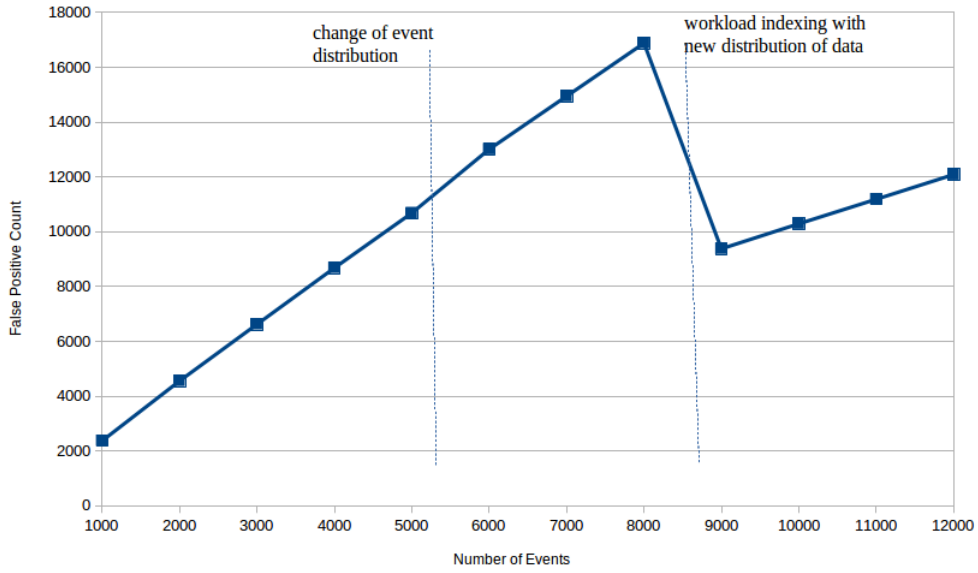


Figure 5.5: False Positive Count for CS with k-means clustering and spatial indexing

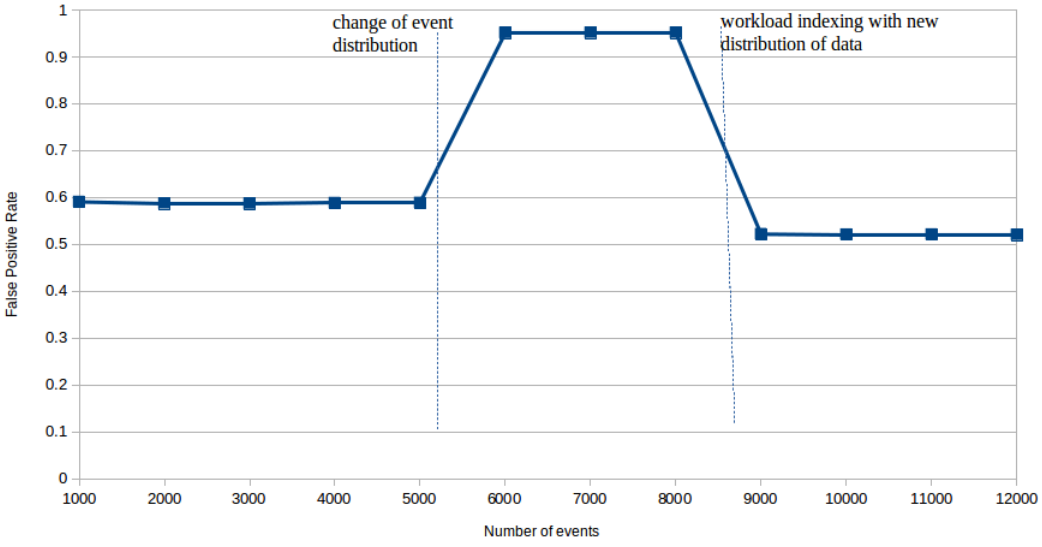


Figure 5.6: False Positive Rate for CS with k-means clustering and spatial indexing

5.2.2 Controller Overhead

Since the controller needs to delete the existing flows and install new flows considering the newly generated dz-expressions, therefore it is expected that this should happen as fast as possible to avoid any loss of published events. To measure the overhead on the controller in order to deploy a new configuration, the time in performing each step was measured. The total time to deploy a new configuration can be calculated as below,

$$T_{total} = T_{wi} + T_{delete} + T_{install} \tag{5.2}$$

where T_{wi} is the time needed by the controller to send the data to the dimension selection and workload indexing module and get the new data back, T_{delete} is the time needed by the controller to delete the previous flows and $T_{install}$ is the time needed by the controller to install new flows by populating the advertisements and subscriptions considering the new set of dz-expressions.

Table 5.1: Reconfiguration Time Measurement

	T_{total}(in Seconds)
CS with SI	143
CS with K-Means+SI	102

Table 5.1 shows the evaluation results obtained for reconfiguration time measurement for both schema i.e. CS with SI and CS with K-Means+SI. It was observed that that the dimension selection and workload indexing process does not take much time, the maximum amount of time is taken by the controller to populate the advertisements, subscriptions and to install corresponding flows. The time taken to reconfigure will change according to change of the topology, number of hosts, number of subscriptions, number of spanning trees used in PLEROMA.

Chapter 6

Summary and Conclusion

This thesis has implemented the workload enabled indexing process in the publish/subscribe middleware PLEROMA for a software defined networking environment. Also the effectiveness and scalability of the techniques for workload indexing have been evaluated in terms of their ability to reduce false positives and also measured the overhead of the controller in order to deploy a workload based indexing schema by deleting previous flows and installing new flows.

Specifically some of the questions that have been answered by this work are,

- How can the controller implement a scalable model for enabling workload based indexing techniques in the data plane of SDN?
- How is the performance of the workload based indexing techniques on an SDN based pub/sub system in their ability to reduce false positives?
- What is the cost of performing workload enabled indexing in SDN in this proposed way?

The implementation and evaluation parts of the thesis have tried to answer each of these questions, and it has been observed that it is possible to reduce false positives for a pub/sub system in SDN by utilizing the workload indexing techniques. The experiments have been run multiple times to arrive at a confident result by averaging the results of all runs.

The techniques have also been made scalable for handling a large number of publishers and subscribers, and the cost of performing workload enabled indexing is measured in terms of amount of time taken by the controller to deploy a workload enabled indexing schema on the data plane.

However, there are some limitations of the work performed in this thesis. After performing workload enabled indexing, the controller deletes the old flows and installs the new flows. While deleting the old flows, there might be still some publishers publishing events, as a result the subscribers will not receive those events resulting in loss of information.

In the evaluation chapter, it was seen that changing the distribution of events results in increased false positive rate, therefore the controller needs to perform workload enabled indexing again considering the new distribution of subscription and events. To do so, the existing system needs to be stopped and new distribution of subscriptions,advertisements and events need to be sent to the controller from respective subscriber/publisher.

As future work of this thesis, there can be a more sophisticated way to handle the flow deletion and installation process so that no loss of published events happen. Also in case the topology is changed, the advertisements and subscriptions need to be sent to the controller again to enable workload indexing on the new topology. Additional research can be performed to find a better solution to handle such a situation. Also the false positive in this thesis has been measured at the end hosts which tells us the capability of workload indexing to reduce false positives as seen from the end hosts. However, an approach to measure false positives in the network switches after performing workload based indexing would tell also about how bandwidth efficient the techniques really are.

In this thesis, the evaluations to measure false positives have been conducted using IPv4 addresses, where 23 bits are available for appending the dz-expressions. Another evaluation can be performed to see the performance when IPv6 addresses are used, since IPv6 addresses will offer more than 23 bits to append the dz-expressions. Also the reconfiguration time has been measured using Mininet's emulated network, the performance of which is dependent upon the performance of the hosting machine and the controller used. Research has been conducted before to compare the performance of different controllers, and it has been established that different controllers have different performance with respect to number of OpenFlow messages that they can handle or number of flows that they can install per second[35]. Therefore, the reconfiguration time measurements that have been evaluated in this thesis are specific only for Floodlight controller, and using a different controller may offer a better result. Also, to avoid the dependency on the performance of the hosting machine, hardware switches can be used to measure the reconfiguration time instead of using Mininet's emulated switches.

The data that was used for conducting the experiments in this thesis has been generated by using a data generation algorithm that generates the data in Zipfian buckets. Even though, the results of the experiments using such a data set show that the techniques are capable of reducing false positives for Zipfian distributed data, not always real life data is Zipfian distributed. Therefore, additional research can be performed in order to find the efficiency of the techniques when real publish/subscribe data is used.

Bibliography

[1] An efficient multicast protocol for content-based publish-subscribe systems. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, ICDCS '99, pages 262–, Washington, DC, USA, 1999. IEEE Computer Society.

[2] Flooglight openflow controller. `https://floodlight.atlassian.net/wiki/display/floodlightcontroller/The+Controller`, [Accessed : November, 2016].

[3] Java scheduledexecutorservice. `https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ScheduledExecutorService.html`, [Accessed : November, 2016].

[4] Mininet-the network emulator. `http://mininet.org/`, [Accessed : November, 2016].

[5] Marcos K. Aguilera, Robert E. Strom, Daniel C. Sturman, Mark Astley, and Tushar D. Chandra. Matching events in a content-based subscription system. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '99, pages 53–61, New York, NY, USA, 1999. ACM.

[6] Sukanya Bhowmik. Distributed control algorithms for adapting publish/subscribe in software defined networks. Master's thesis, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, November 2013.

[7] Sukanya Bhowmik, Muhammad Adnan Tariq, Jonas Grunert, and Kurt Rothermel. Bandwidth-efficient content-based routing on software-defined networks. In *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, DEBS '16, pages 137–144, New York, NY, USA, 2016. ACM.

[8] Douglas R. Caldwell. Unlocking the mysteries of the bounding box. `http://purl.oclc.org/coordinates/a2.htm`.

[9] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst.*, 19(3):332–383, August 2001.

[10] Emiliano Casalicchio and Federico Morabito. Distributed subscriptions clustering with limited knowledge sharing for content-based publish/subscribe systems. *2013 IEEE 12th International Symposium on Network Computing and Applications*, 00(undefined):105–112, 2007.

[11] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC)*, 20(8):1489–1499, 2002.

[12] IBM TJ Watson Research Center. Gryphon : Publish/subscribe over public networks. `https://www.research.ibm.com/distributedmessaging/gryphon.html`, [Accessed : October, 2016].

[13] Chen Chen, Hans-Arno Jacobsen, and Roman Vitenberg. Algorithms based on divide and conquer for topic-based publish/subscribe overlay design. *IEEE/ACM Trans. Netw.*, 24(1):422–436, February 2016.

[14] Alex King Yeung Cheung and Hans-Arno Jacobsen. Load balancing content-based publish/subscribe systems. *ACM Trans. Comput. Syst.*, 28(4):9:1–9:55, December 2010.

[15] Gianpaolo Cugola, Elisabetta Di Nitto, and Alfonso Fuggetta. The jedi event-based infrastructure and its application to the development of the opss wfms. *IEEE Trans. Softw. Eng.*, 27(9):827–850, September 2001.

[16] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, June 2003.

[17] Open Networking Foundation. Openflow switch specification. `http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf`, [Accessed : October, 2016].

[18] Hans-Arno Jacobsen, Alex Cheung, Guoli Li, Balasubramaneyam Maniymaran, Vinod Muthusamy, and Reza Sherafat Kazemzadeh. The PADRES Publish/Subscribe System. In *Principles and Applications of Distributed Event-Based Systems*, pages 164–205. IGI Global, 2010.

[19] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, September 1999.

[20] X. Jin, W. Tu, and S. H. G. Chan. Scalable and efficient end-to-end network topology inference. *IEEE Transactions on Parallel and Distributed Systems*, 19(6):837–850, June 2008.

[21] Apoorva Jindal and Konstantinos Psounis. Modeling spatially correlated data in sensor networks. *ACM Trans. Sen. Netw.*, 2(4):466–499, November 2006.

[22] Petri Jokela, András Zahemszky, Christian Esteve Rothenberg, Somaya Arianfar, and Pekka Nikander. LIPSIN: line speed publish/subscribe inter-networking. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 195–206, New York, NY, USA, 2009. ACM.

[23] Boris Koldehofe, Frank Dürr, Muhammad Adnan Tariq, and Kurt Rothermel. The power of software-defined networking: Line-rate content-based routing using openflow. In *Proceedings of the 7th Workshop on Middleware for Next Generation Internet Computing*, MW4NG '12, pages 3:1–3:6, New York, NY, USA, 2012. ACM.

[24] Minseok Kwon and Sonia Fahmy. Path-aware overlay multicast. *Comput. Netw.*, 47(1):23–45, January 2005.

[25] Danny B. Lange, Mitsuru Oshima, Günter Karjoth, and Kazuya Kosaka. *Aglets: Programming mobile agents in Java*, pages 253–266. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[26] Yannis Manolopoulos, Yannis Theodoridis, and Vassilis J. Tsotras. *Spatial Indexing Techniques*, pages 2702–2707. Springer US, Boston, MA, 2009.

[27] Wolfram MathWorld. Covariance. `http://mathworld.wolfram.com/Covariance.html`, [Accessed : November, 2016].

[28] Gero Mühl. *Large-Scale Content-Based Publish-Subscribe Systems*. PhD thesis, Technische Universität, Darmstadt, November 2002.

[29] Oracle. An overview of rmi applications. `https://docs.oracle.com/javase/tutorial/rmi/overview.html`, [Accessed : November, 2016].

[30] Peter R. Pietzuch. *A scalable event-based middleware*. PhD thesis, University of Cambridge, June 2004.

[31] Maria Gradinariu Potop-Butucaru, Silvia Bianchi, and Pascal Felber. Stabilizing distributed r-trees for peer-to-peer content routing. *IEEE Transactions on Parallel and Distributed Systems*, 21(undefined):1175–1187, 2009.

[32] R. Rajkumar, M. Gagliardi, and Lui Sha. The real-time publisher/subscriber interprocess communication model for distributed real-time systems: Design and implementation. In *Proceedings of the Real-Time Technology and Applications Symposium*, RTAS '95, pages 66–, Washington, DC, USA, 1995. IEEE Computer Society.

[33] Anton Riabov, Zhen Liu, Joel L. Wolf, Philip S. Yu, and Li Zhang. Clustering algorithms for content-based publication-subscription systems. In *Proceedings of the 22 Nd International Conference on Distributed Computing Systems (ICDCS'02)*, ICDCS '02, pages 133–, Washington, DC, USA, 2002. IEEE Computer Society.

[34] Anton Riabov, Zhen Liu, Joel L. Wolf, Philip S. Yu, and Li Zhang. New algorithms for content-based publication-subscription systems. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, ICDCS '03, pages 678–, Washington, DC, USA, 2003. IEEE Computer Society.

[35] Alexander Shalimov, Dmitry Zuikov, Daria Zimarina, Vasily Pashkov, and Ruslan Smeliansky. Advanced study of sdn/openflow controllers. In *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*, CEE-SECR '13, pages 1:1–1:6, New York, NY, USA, 2013. ACM.

[36] Jonathon Shlens. A tutorial on principal component analysis. *CoRR*, abs/1404.1100, 2014.

[37] Robert Strom, Guruduth Banavar, Tushar Chandra, Marc Kaplan, Kevan Miller, Bodhi Mukherjee, Daniel Sturman, and Michael Ward. Gryphon: An information flow based approach to message brokering. In *IN PROCEEDINGS OF THE INTERNATIONAL SYMPOSIUM ON SOFTWARE RELIABILITY ENGINEERING*, 1998.

[38] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[39] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[40] Muhammad Adnan Tariq, Boris Koldehofe, Sukanya Bhowmik, and Kurt Rothermel. Pleroma: A sdn-based high performance publish/subscribe middleware. In *Proceedings of the 15th International Middleware Conference*, Middleware '14, pages 217–228, New York, NY, USA, 2014. ACM.

[41] Muhammad Adnan Tariq, Boris Koldehofe, Gerald G. Koch, Imran Khan, and Kurt Rothermel. Meeting subscriber-defined qos constraints in publish/subscribe systems. *Concurr. Comput. : Pract. Exper.*, 23(17):2140–2153, December 2011.

[42] Sasu Tarkoma. *Publish / Subscribe Systems: Design and Principles*. Wiley Publishing, 1st edition, 2012.

[43] Wikipedia. Active object- wikipedia. `https://en.wikipedia.org/wiki/Active_object`, 2016.

[44] Wikipedia. Bloom filter- wikipedia. `https://en.wikipedia.org/wiki/Bloom_filter`, 2016.

[45] Wikipedia. Client–server model. `https://en.wikipedia.org/wiki/Client%E2%80%93server_model`, 2016.

[46] Wikipedia. Content-addressable memory- wikipedia. `https://en.wikipedia.org/wiki/Content-addressable_memory`, 2016.

[47] Wikipedia. Software-defined networking – wikipedia. `http://en.wikipedia.org/wiki/Software-defined_networking`, [Accessed : October, 2013].

[48] Ye Zhao, Kyungbaek Kim, and Nalini Venkatasubramanian. Dynatops: A dynamic topic-based publish/subscribe architecture. In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS '13, pages 75–86, New York, NY, USA, 2013. ACM.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature