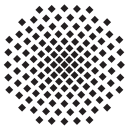

Universität Stuttgart

Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master's Thesis

Ensuring Data Plane Consistency in SDN-Based Publish/Subscribe Systems

Deepak Srinivasan

Course of Study:	InformationTechnology/INFOTECH
Examiner:	Prof. Dr. Kurt Rothermel
Supervisor:	Dr. Adnan Tariq & M.sc. Sukanya Bhowmik
Commenced:	2016-11-01
Completed:	2017-05-03
CR-Classification:	C.2.1,C.2.4

Abstract

Content-Based publish/subscribe paradigm is a widely used paradigm which enables applications to share events, and allow the integrated applications to remain loosely coupled. The importance of the paradigm increases further with the emergence of IoT applications, a micro service approach to application development, etc. With the rise in the amount of data being shared between applications, there is a need to provide high data rate and bandwidth efficiency, and the introduction of Software Defined Networking(SDN)-based pub/sub, it becomes possible to achieve both.

Content-based pub/sub built on SDN uses the IP address match fields of the flows in Ternary Content-Addressable Memory(TCAM) as content filters. While such an in-network filtering offers high data rate, it also involves frequent updates in the data-plane to keep the system running optimally. Although pub/sub using SDN is efficient, to use it in real-time, it needs to ensure some level of consistency when updating content-filters in the data plane so that events are not subjected to black holes, duplicates, loops, etc.

Keeping the requirement of ensuring consistency in mind, the goals of this thesis include an analysis of the data plane consistency issues related to SDN-based pub/sub, to apply existing algorithms or implement new algorithms to preserve consistency of the data plane, and to provide an evaluation comparing all the implemented solutions.

Acknowledgements

Working on this thesis with the Institute of Parallel and Distributed Systems has been an incredible experience. My thanks to Prof. Kurt Rothermel for giving me the opportunity to work in this department on an interesting and challenging topic.

My deepest gratitude to M. Sc. and soon-to-be Dr. Sukanya Bhowmik for her insights, patience, and balanced guidance during the duration of this work. It was an enjoyable experience working under her guidance. I would also like to thank Dr. Muhammad Adnan Tariq for making time in his busy schedule and providing me with insightful perceptive on pub/-sub architectures. Special thanks to Jonas Grunert, Himanshu Sharma and for letting me work on their codebase for workload based indexing. It would be remiss of me not to thank the developers of floodlight for helping me understand some of the undocumented floodlight modules.

I cannot put into words the debt I owe my family and friends without whom my life has no meaning. I thank them for being there for me.

Contents

List of Figures

List of Algorithms

1 Introduction

The last two decade has seen an emergence of an ecosystem with a large number of applications, each integrated with several other applications. These integrations are extremely inflexible, not fault tolerant if the event sources and consumers need to know each other's physical location(referential coupling) and communicate using synchronous protocol(temporal coupling). Referential decoupling with asynchronous communication provides the necessary flexibility for such integrated applications to run independent of each other. Publish-Subscribe(called pub/sub from here onwards) paradigm is one such solution which makes this loose coupling possible.

In pub/sub architecture publishers are senders of events and subscribers are receivers of events they are interested in. An event sent by a publisher can be received by zero or more subscribers. A subscriber can receive events sent by multiple publishers. In pub/sub architecture both, publishers and subscribers, are unaware of the other's network location and the events are received asynchronously unlike synchronous architectures such as client/server architecture where a client sends a request to the server and then it waits for the server to respond. This enables the subscribers to remain free from the burden of needing to maintain the location of all data sources and publishers from the burden of having to respond to individual requests from different subscribers, interested in the same data, by sending the same data over the network multiple times. Moreover, pub/sub also enables a publisher or subscriber to dynamically join and leave the system without informing the other publishers and subscribers. Hence pub/sub architectures provide a dynamic many to many asynchronous communication pattern with referential and temporal decoupling which is used in several event-driven applications such as an electronic auction, online trading, IoT applications.

In any pub/sub architecture for publishers and subscribers to remain unaware of the other's location and yet route events to interested subscriber there needs to be an intermediary. This role is fulfilled by an intermediary called the *Notification service* [1]. Subscribers send *subscriptions* to the Notification service to make it aware of their interests. While the Notification service is logically centralized, it can be physically partitioned & replicated to make it scalable and fault tolerant. For instance, it can consist of multiple nodes called brokers which are distributed over the network. Hence Notification service forms a logical overlay network over the existing physical network.

The pub/sub architecture is classified, based how events are matched to subscriptions at these brokers, into topic-based and content-based[2] pub/sub. In topic-based pub/sub, the events are tagged with their topic which is used for routing. On the other hand, in content-based routing the actual content of events are used for routing. The bandwidth efficiency in the network depends on the effectiveness(called expressiveness) of the filter provided in the subscriptions and on how expressiveness the filters are at the brokers. A high bandwidth efficiency can be achieved if the expressiveness of the filters is high and vice versa. Since in content-based routing the content of the events is directly used for filtering, it is possible to achieve greater expressiveness with it than with topic-based routing. Another quality factor which is an indicator for better performance is the speed of filtering[3]. This factor needs to be comparable to line rate. Both these factors can be optimized only if the filtering happens in network layer rather than at the application layer and the pub/sub overlay network has knowledge of the underlying physical network which consists of switches and routers. Achieving this has become a possibility with the emergence of content-based pub/sub systems which make use of Software Defined Networking(SDN)[4].

Traditional network is a black box whose switches are closed to any change of its pre-defined protocols. Each of the switches in the network takes routing decision only based on its own state and any information it can deduce about other switches & devices connected directly to it. In SDN all the switches in a network are connected to a central entity which plays the role of a controller which can gather data about the switches and has the unified view of the network. Further, the controller can add routing rules and push new protocols to the switch based on its view of the entire network. For installing a new protocol it makes use of memory in the switches called Ternary Content Addressable Memory(TCAM). The new rules that are added are called flows. A Flow consists of two parts one of which is the match field which is used for matching packets to flow rules and the other is the action which describes the actions to be preformed on the packet for routing.PLEROMA [5][6] is a pub/sub system which uses the controller as the notification service and the physical network for actual routing using rules provided by the controller.

PLEROMA uses the Ternary Content Addressable Memory(TCAM) of the switches to install a content filter in the form of a flow whose destination IP match field corresponds to the filter. This enables filtering of events in the physical network and not at the application layer. Moreover, since the events are handled at the network layer it becomes possible to achieve line rate performance. Even though content-based pub/subs are supposed to provide high expressiveness in the case of PLEROMA the expressiveness of the filters were limited since the number bits in the IP address are limited. This limitation was addressed in [7] which provides a solution to efficiently convert event content to IP address based workload in the network. The proposed solution uses the current distribution of events in event spaces & the subscriptions and then provides a new method for translation of event content to IP address. Such a change in the SDN data plane needs a deployment plan which ensures the quality of service required of PLEROMA are not violated (are referred to as network invariants from here) during the transition. In addition to content address optimization, there are instances

such as high network traffic due to which the deployed content filters need to be moved to other switches. All these changes also need to ensure the necessary quality of service. In generic terms, these updates need to be consistent. Existing solutions for such consistent updates are of two categories stateless and stateful algorithm. In stateless algorithms, the flow changes associated with the update from one consistent state to another consistent state are ordered so as to ensure that network invariants are not violated. In stateful algorithms, packets are injected with a version or state identifier to ensure that they are routed with either old or new state but never a mixture of the two[8].

Most of the existing solutions for consistent updates in the data plane generally deal with unicast routes which cannot be applied directly(except [8]) to PLEROMA because its content filtering routes are IP prefix-based multicast routes. The aim of this thesis is to provide consistent update solution for IP prefix-based multicast routes based on the above-stated cases in PLEROMA.

1.1 Thesis Organization

The thesis has been organized as described below:

Chapter 2 provides an overview of topics which are necessary for understanding to the thesis. It includes an overview on pub/sub, SDN, unicast consistent update strategies.

Chapter 3 provides an overview of topics PLEROMA, Workload-based indexing.

Chapter 4 provides the definition and description of this thesis's problem statement.

Chapter 5 provides a description of algorithms and an observation of the proposed solutions.

Chapter 6 provides an overview of the implementation, evaluations and an analysis of results.

2 Background

This chapter endeavors to provide an adequate background of key concepts that forms the basis for this thesis. It provides a fundamental overview of pub/sub paradigm, Software-Defined Networking(SDN), existing pub/sub systems built using SDN and existing consistent data plane update strategies in SDN.

2.1 Inter Process Communication paradigms

Inter-process communication is a term which covers a wide range of communication paradigms. In principle, two different processes can communicate with each other as long as there is a channel between them for them to do so. A channel can be files, pipes, shared memory to complex middlewares which abstract out the complexities of communication between processes in remote machines[9]. Each of following paradigms aims of providing some or all of the following four criterion [10]

- Access Transparency - Hiding of the heterogeneities that arise due to OS, hardware and programming language specific differences between the communicating processes. It also refers to keeping the communicating processes oblivious to the fact that they are communicating to remote processes.
- Location Transparency - The actual network specific addresses of the processes involved in communication are hidden from the communicating processes.
- Concurrency Transparency - Several processes may be communicating with the same process. But the process which initiates such a communication should not be aware of this concurrency
- Replication Transparency - Called or calling methods can be replicated without breaking each other

This section is dedicated to such paradigms one of which is pub/sub.

2.2 Remote procedure call

A local method/function call is one in which the *called method* resides in the same process address space as the calling method, as a result, both share the same variables, memory space and system specifications. In contrast, Remote Procedure Call(RPC) provides a paradigm in which the called method(server) resides in a remote process and may be in a remote machine

to the calling process(client). The idea of this paradigm is to make RPC behave in a manner similar to local calls by making method signatures and invocation procedures more akin to local calls. This follows the typical client/server architecture. Several middlewares such as Java RMI provide this paradigm. These middlewares provided location transparency - as both client and server are oblivious to the other's location. Some these middlewares also provide asynchronous request-response architecture.[11][12]

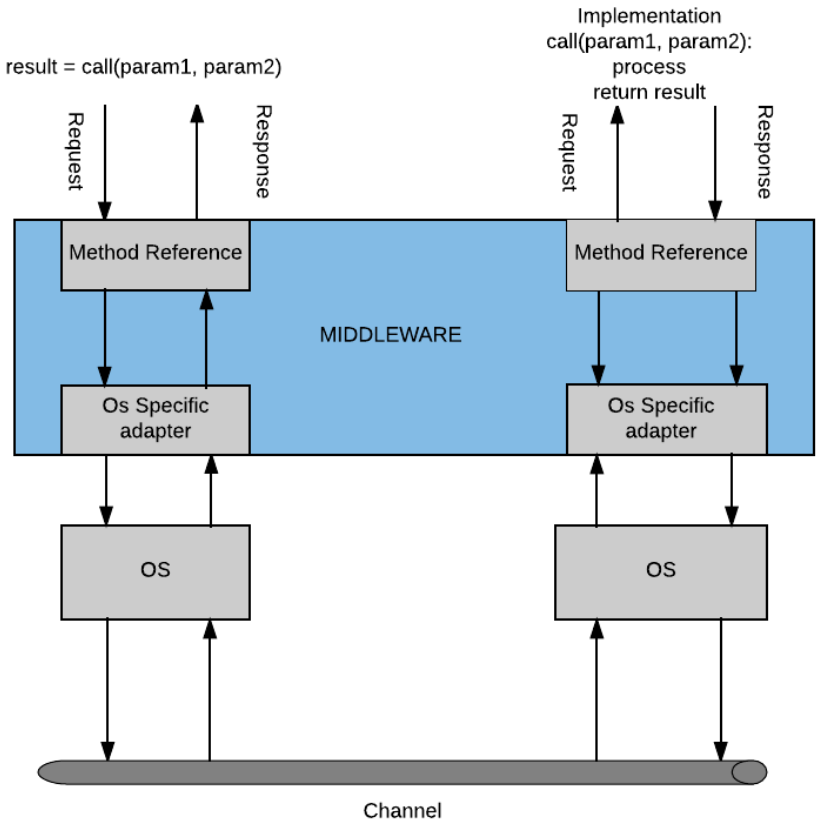

Figure 2.1: RPC Architecture

The disadvantage with this paradigm is the violation of access transparency. One main reason for this is that the latency caused due to communication delays is much higher for remote calls than with local calls and for remote calls, there are several exception scenarios such as communication failure, remote process failure, etc., as well. It is not possible or even prudent to abstract such errors from the calling processes. Another reason for the loss of transparency is that if the process communicating with each other run on different OS or use different programming language semantics, it becomes difficult to completely hide the heterogeneity [10]. Moreover, in RPC the client and server are tightly coupled to the remote

method signature. This makes it difficult for the server to change the remote method signature. There are scenarios where it is not desirable to use RPC especially when there is many to many communication involved. For example in information dissemination applications, where a client may be interested in similar data from multiple servers. If that's the case, then it must invoke a remote method from each of them to get the data it desires. This is a burden for the client because it needs all the servers to be available at the same time and it becomes a burden for the server because several clients may request separately for the same data. Moreover, A server cannot start a new conversation as well. Hence, on the whole, the interface of RPC is not flexible enough to allow the integrated application to change independent of each other and it is not suitable for many to many communication.

2.3 Message Oriented Middlewares

Message oriented middlewares(MOM) represents a class of middlewares which makes information dissemination in large heterogeneous networks possible. The communication between the nodes in such networks have the following three attributes

- **Asynchronous communication or Event-driven communication** - This the opposite of request-response communication. Here the receivers of information do not initiate a request and wait for a response.
- **Referential Decoupling** - The event-senders and event-receivers are unaware of the network address of the other. This is made possible by using an intermediary which takes care of routing packets to correct destination. Sometimes, these intermediates are just simple buffers or queues.
- **Generic Data Type Formats** - The data type of the events sent in these middlewares is independent of sender's & receiver's environment. This makes it easier for the middlewares to handle the events routed by it.

2.4 Principles of Publish/Subscribe Paradigm

A pub/sub is a pattern/flavor of MOMs. It is used in distributed information dissemination applications in which the membership of the entities in the system are dynamic and many-many communication is involved. The various components in this pattern are discussed in the following section 2.4.1.

2.4.1 Components in pub/sub

Publishers

Publishers are event senders/event sources. Every publisher will just send events without addressing it to any specific destination.

Subscribers

Subscribers are event receivers/event sinks. A subscriber only receives data which it is interested in. But it has no idea about the source of a received event.

Notification Serivce

It is an intermediary between the publishers and subscribers. It is responsible for receiving events and for handing it over to the correct subset subscribers. Notification service also handles the membership of publishers and subscribers. It is also responsible for maintaining their respective addresses. In general, a notification service can comprise of a network of distributed nodes called the brokers which can span a wide area network. This broker network forms an overlay network on the physical network by handling the actual routing of events based on pub/sub rules. The exact method used for routing the packets depends on two factors, namely the type of pub/sub architecture and the quality of service offered for event delivery by the same. The various classes of pub/sub are described in 2.4.3 and the quality of service of a pub/sub based delivery of events are - at-least once, atmost-once , or exactly once semantics. The implementation of these semantics require the use of queues and may require the queues to maintain a history of past event in notification service. 2.2

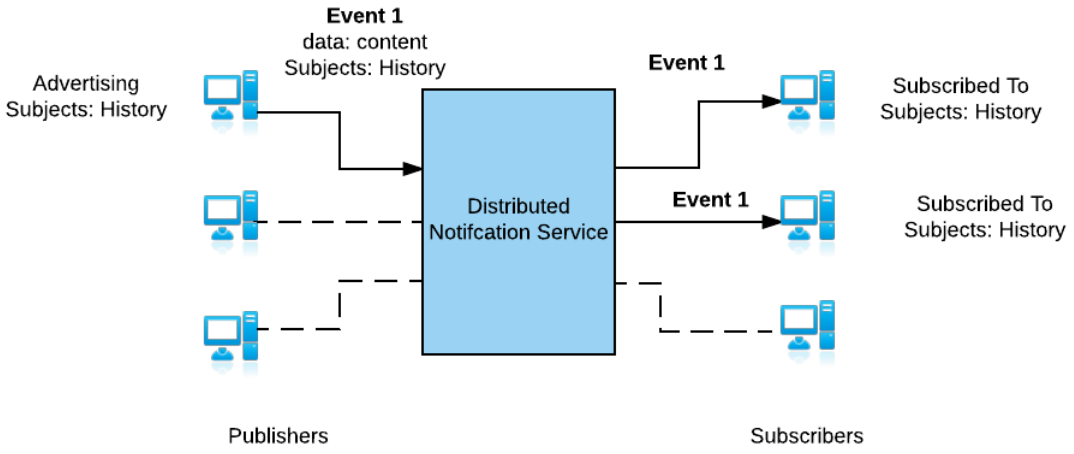

Figure 2.2: Pub/Sub Architecture

2.4.2 Operations in pub/sub

This list of operations in pub/sub are listed below:-

- **A new publisher entering pub/sub** - A new publisher enters the pub/sub system by sending a special message called the advertisement to the notification service or the nearest broker. The content of advertisements must contain information regarding the type of data it wishes to publish. The structure of the advertisement can be a simple string label, a range of values with multiple dimensions, or a complex expression depending on the type of pub/sub system. Only after a publisher sends an advertisement the notification service will handle events sent by the publisher.

- **A new subscriber entering pub/sub** - A subscriber can receive events only after it subscribes to the events. A subscriber uses the subscription messages for this purpose. The structure of a subscription event can be a simple label or a condition similar to SQL condition depending on the type of pub/sub system.

- **An existing publisher leaving pub/sub** - A publisher can withdraw itself from the pub/sub system by sending an unadvertisement message to the notification service or nearest broker.

- **An existing subscriber leaving pub/sub** - A subscriber can stop receiving events by sending an unsubscription message to the notification service or to its nearest broker.

2.4.3 Categories in pub/sub

A pub/sub system can be classified based on how packets are filtered and routed by the notification service. The events can be filtered either based on the actual content of the packets or based a label attached to the packet by the publishers [13]. The first is called a topic-based pub/sub and the second is called a content-based pub/sub.

Topic-Based Pub/Sub

Topic-based pub sub can be further subdivided into 2.3

- **Channel-based** - Here all the publishers and subscribers belong to one or more channels. A published event is published to exactly one of the channels and will be received by all the subscribers of that channel. In channel-based pub/sub, there is no relationship between two channels. There is a one-one correspondence between publishers and subscribers. This can be directly used with IP-multicast where each multicast address corresponds to a channel. In this case, no advertisement are required. Example :- CORBA [14].

- **Subject-based** - The topics of a subject-based pub/sub are related to each other. The relationship diagram can be represented by a tree. There is more decoupling between publishers and subscribers in subject-based pub/sub than in channel based pub/sub. Example :- Apache TIBCO [15].

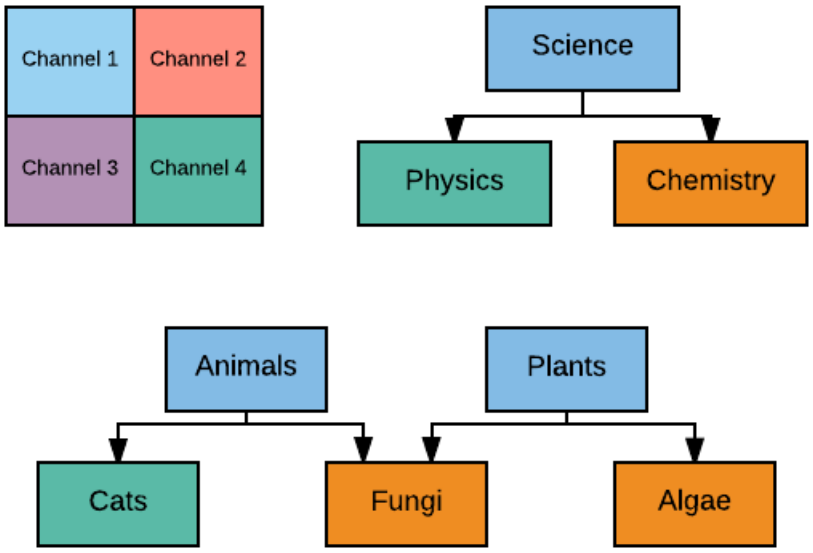

Figure 2.3: Topic-Based pub/sub - On top left is an example of channel-based topics-space, on top right is an example of subject-based topics tree and On the bottom is an example of type-based topics-graph

- **Type-based** - Type-based pub/sub is based on the idea of inheritance of an object oriented approach. Each type can be inherited and/or inherit from one or more types. It is more flexible and more decoupled than both channel and subject based architectures.

The advertisements of a publisher in topic-based pub/sub will be of the format : {topics:[topic1,topic2...]}. While subscriptions will be of the format : {topics:[topic2,...]} and events of the format {data:{}, topic:topic1}.[13].

Content-Based Pub/Sub

Content-based pub/sub performs routing by using the actual content of the events. This enables the subscribers to be more granular about their subscriptions and advertisements. This allows subscribers and publishers to be completely decoupled. However, the actual granularity of the subscriptions depends on the expressiveness and flexibility allowed for by the intermediate filters at the brokers. The filters can be a simple equality checking condition or can be as complex as a SQL statement. As the expressiveness of the filters increases so does the corresponding complexity. In general, more complex a filter higher is the latency for handling each packet. This is especially true for distributed brokers as the actual set of subscribers for each event is dynamic depending on the content of the packets. As a result, it becomes difficult to achieve as high a performance in content-based pub/sub as compared to topic-based architectures. 2.4

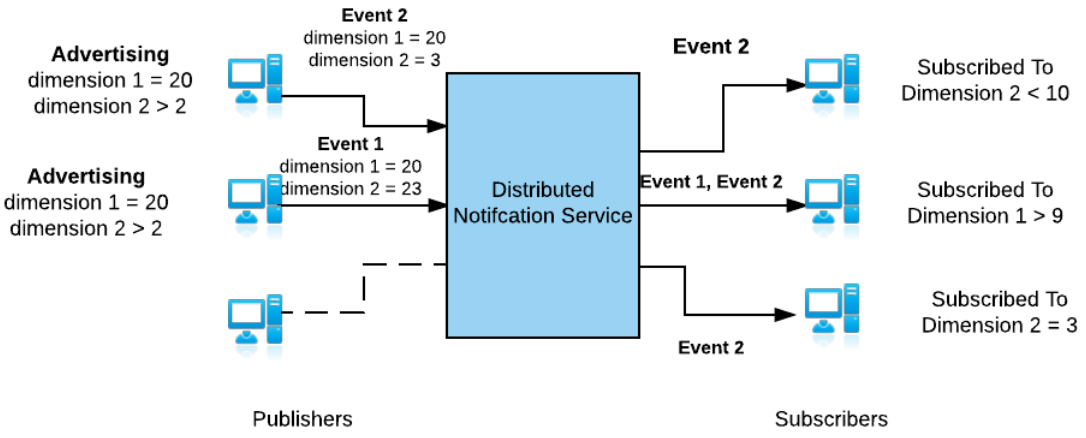

Figure 2.4: An example of a content-based pub/sub system

One method of filtering in content based pub/sub is to calculate a content address from the entire content. This is similar to labeling an event where the label is calculated from the content of the event. A concrete methodology to do this is spatial indexing proposed by Tariq et al. in [16]. In this methodology, the assumption is that each event belongs to an n-dimensional event space. The entire event space is represented by *. When this event space is divided into two parts along one of the dimensions each of the resulting partition will be 0 and 1 respectively. On further division of, say partition 0, into two sub-partitions they will be prefixed with label of their parent partition and suffixed with an additional 0 and 1 at the end respectively. The partitioning can then proceed along any of the n-dimensions. The binary address calculated with this method is called DZ from here on. DZ of these partitions can be used to represent an event space used in advertisements and subscriptions. The granularity of these subscriptions depends on the maximum number of bits per DZ that can be used in the particular pub/sub architecture. A DZ can be calculated for each of the published events, using which each an event can be routed. One thing of note about forming the DZ address is that there are several ways to partition the event space even along just one dimension as depicted in 2.5. The specific method of partitioning is chosen based on the distribution of events and subscriptions in event space to achieve the necessary performance parameters2.4.4 of the pub/sub system.[17] 2.4

2.4.4 Performance parameters of pub/sub

Performance of a pub/sub can be measured with the following parameter [16]

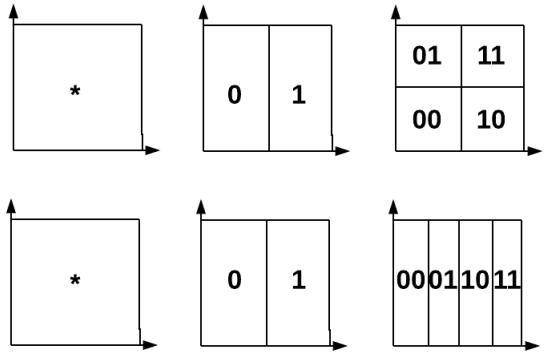

Figure 2.5: An example of partitioning event space in two different ways.

Bandwidth efficiency

Any pub/sub system has some unnecessary traffic in its network links. This can be due to multiple reasons, some of which are listed below.

- Events can be passed around in the physical network where it may traverse the same link multiple times because the brokers are an overlay network and have knowledge of the underlying physical network. Hence two logically neighboring brokers can be in different parts of a wide area network causing events to be pinged back and forth.
- Duplicate events may be delivered to subscribers multiple times.
- Events which are delivered to a subscriber may be contained in the subscription of a subscriber but is not actually needed by the subscriber. This happens if the subscriptions are not expressive enough. These events are called *false positives*.

Due to the above reasons, it becomes necessary to measure bandwidth efficiency. It is the ratio of useful bandwidth which is characterized by the number of true positive (opposite of false positives) to the totally utilized bandwidth which is characterized by the total number of events. In general, network false positives is good indicator of bandwidth inefficiency and subscriber false positives a good indicator of the expressiveness of subscriptions.[16]

Linerate performance

This is a measure of how fast the events are transmitted from publisher to subscriber. An important observation about the complexity of filtering method is that it affects scalability and line-rate performance of a pub/sub architecture. In general, more complex the filtering process less scalable the pub/sub architecture. Hence, the complexity of filtering process is directly proportional to the expressiveness of the filters and inversely proportional to line-rate performance. This result in a scenario where a compromise is necessary between expressiveness and scalability.[16]

2.4.5 Related Pub/Sub architectures

SIENA

Scalable Internet Event Notification Architecture(SIENA)[1] is a wide area content-based pub/sub. The *Notification Service* of SIENA is made up of a distributed set of brokers. An event in SIENA is a set of attribute-value pairs. An attribute has a type and value. To receive events, a subscriber has to send a subscription message. A subscription message is a conjunction of predicates. A predicate is a condition on an attribute which is compared to a value in comparison conditions. Similarly, an advertisement is also a conjunction of predicates and it is a message used by publishers to let the brokers know that a publisher publishes a certain group of events which satisfies the condition sent as part of the advertisement.

A version of SIENA works without advertisements, in this case, when a subscription is received by a broker it is flooded to all the brokers. In the other version, with advertisements, SIENA floods advertisement to all brokers and the advertisement information is used to send the new subscriptions only to a subset of the brokers. This method leads to a more cleaner content filtering approach because information on publishers is also available. Once a subscription/advertisement is received by a local broker, it forwards the subscription or advertisement if and only if it is not already covered by a previous subscription/advertisement. A sub A content filter is added to a broker if it receives a subscription/advertisement not covered before. A subscription/advertisement covers another subscription/advertisement if the condition of the former is completely covered by the condition of the latter. For example, a>10 is completely covered by a>9. This is represented as a>10 $\succ$ a>9. A content filter is a conjunction of predicates. A publisher sends an event to its local broker. The local broker checks if the event is covered by a content filter. If so, then the event is passed along to the next broker or subscriber. If not, it is dropped.

SIENA provides an example pattern of components and techniques to create a scalable content-based pub/sub. Its obvious disadvantage is that the content filtering takes place on the application layer and it supports only small set predefined types and predicates.[18]

LIPSIN

LIPSIN[19] is a topic-based pub/sub which perform in-network filtering. The major challenge of performing in-network filtering is that the applications running on the application layer are unaware of the network topology in traditional networks. To make this a reality, it divides the notification services into two layers controller plane and data plane. The function of the control plane includes topology system which obtains a distributed knowledge of the network topology at the routers and another module called the rendezvous system which contains a cache of topics of subscriptions, and advertisements.

When the topology system collects the information of links, each link is associated with two Ids - one for the forward direction and another for the reverse direction. These link ids is a unique binary string for each link. Once all the links are associated with the Link Id, LIPSIN gets ready to handle subscriptions and advertisements. When there is a new publisher with a topic to publish events in, the rendezvous system checks if there are active subscribers for that topic. If there are active subscribers, a multicast path connecting publisher to all active subscriber is computed. The resulting tree path is represented by a concatenation of all the link Ids in the path. The concatenated result is hashed using bloom filters called zFilters [20]. The hashed result is sent to the publisher. The publisher now uses this hashed result as a label for the events it sends. An event with the bloom filter value as the label received by any node between the subscribers and publisher ANDs the filter value with all the outgoing link Ids of that node. If the result of this AND operation is the Link Id then the event is sent out along that link.

While LIPSIN provides in-network filtering, it suffers from the obvious problem of higher false positive rate inherent to most topic based systems. Moreover, the network knowledge obtained by the topology system is only as dynamic as any routing protocol. Hence, the system cannot react to changes in network traffic. The following section describes a well-known approach of making the network more flexible.[19]

2.5 Software Defined Networking

In the traditional network, the protocols supported by the network depends on the protocols defined by the individual components of the network, namely, the switches and routers. The traditional network has no centralized component and it is a distributed network of devices. As a result of its distribution, it is not easy to introduce a new protocol in this network because if a new protocol is to be implemented then all firmware in the devices or the devices themselves need to be replaced.

Software Defined Networking(SDN) is an approach to bring a degree of flexibility into this network by adding a higher plane called the control-plane to the so-called data-plane in which switches, hosts, and links between them exist. The control-plane contains one logical entity called the controller. The switches are connected to this controller. They send can packets they receive, statistics about themselves to the controller using the connection. As a result, the controller obtains knowledge of the switches, topology of the network and various statistics about the switches. It can use this combined knowledge to deploy new protocols in the form flows to the switches.[21][22][23]

2.6 The communication protocol format between data-plane and control-plane is defined by Openflow standards [21]. The various standards defined by open flow includes,

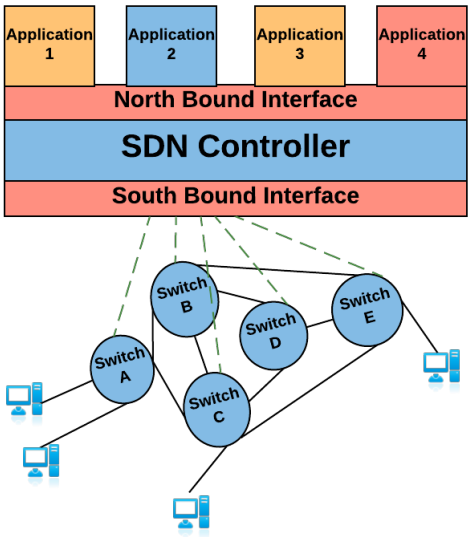

Figure 2.6: SDN Architecture

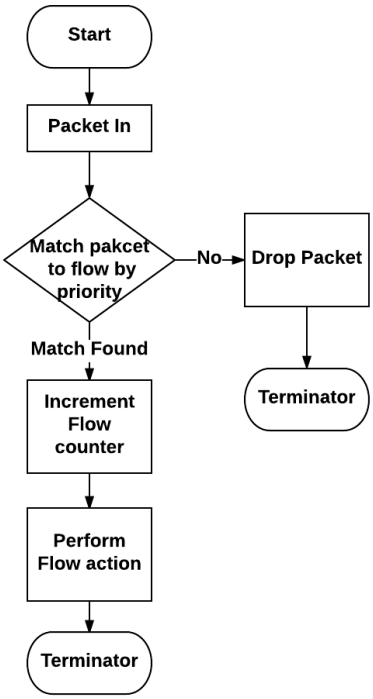

Figure 2.7: Packet Matching

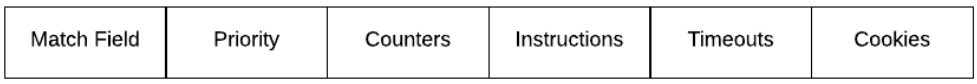

Figure 2.8: Flow Structure

1. An interface between the controller and the switches which is called the southbound interface.

2. An interface between the controller and application for running business specific protocols is called the northbound interface.

3. A definition of the data format and messages used in the northbound interface and southbound interface.

The decoupling provided by the Openflow standards enables the use of controllers from different vendors and environments. One thing to note is that the exact implementation details of the various messages and stats are vendor specific. Hence, for a switch to be part of SDN it should be implementing one of the Openflow standard versions(OF 1.0,1.2 or 1,3 etc).

One can view what SDN does as network visualization where we can create logically separate networks which lie in the same physical network. The networks so created are not an overlay on the existing network from which the physical network was abstracted out but a logical separation of the same network layer. The topologies of the logical networks can be altered from the controller in real time.

2.5.1 Flow Structure

The new protocols that are pushed by the controller on to the switches are in the form flows. These flows reside in the flow table in a memory called the TCAM memory or Ternary Content Addressable Memory. These flows constitute the rules for handling an incoming packet[21]. Parts of a flow are shown in 2.8.

- **Match Field** It is used for matching the data in this field to incoming packets. It can have ingress & egress ports, source & destination IP addresses, ports, VLAN Ids, etc., as the matching criterion.
- **Priority** This field defines the order in which the incoming packets are matched to flows. Greater priority flows are matched to incoming packets before lesser priority flows.
- **Counters** Counters are incremented each time a specific event occurs. An event can be the matching of a packet to a flow, dropping of a packet, etc.. These meters are useful to obtain flow specific stats which can be used for further processing at the controller.
- **Instructions** Specifies the set of actions to be performed when a packet is matched to a flow.
- **Timeouts** Time to live for flows either in terms of maximum time to live or idle timeout.
- **Cookies** Opaque headers for use of the controller

2.6 Content-Based pub/sub with SDN

Distributed pub/sub solutions such as [24], [25] are implemented at the application layer. The middlewares implemented at the application layer are unable to provide line rate performance because

- they are unaware of the underlying physical network. This means that two distributed brokers of a pub/sub middleware could be on different ends of the globe and packet may need to be bounced around a lot. The reason for this is that the Internet was meant to transmit a packet over link based only on the end hosts and the network, paths were meant to be transparent. Or in other words, the abstraction of the network was by design rather than a flaw in the traditional internet. But this in the case of network unaware pub/sub results in higher network traffic

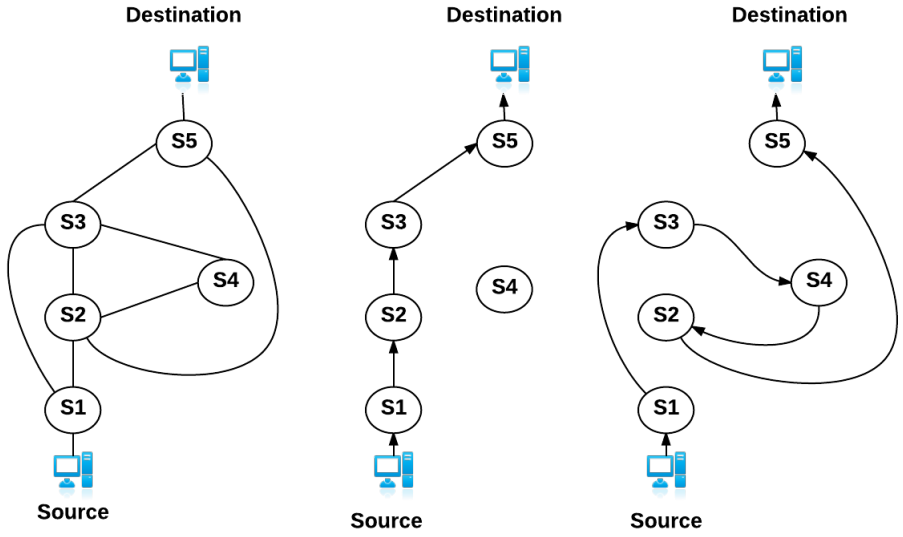

Figure 2.9: The figure on the left provides the network topology graph. The figure in the middle provides rules with existing State A. The figure on the right depicts the rules for the new State B

- they operate at the application layer, which means a packet has to be handled by the sessions, presentation layers before reaching application layer which leads to processing related latency.

Several topic-based pub/sub solutions which depend on network knowledge for routing have already been introduced. For example, LIPSIN [19] is a topic-based pub/sub which uses network topology knowledge & bloom filters to route events. The problem with most of these solutions is a lower selectivity for subscribers 2.4.3 which is inherent to all topic-based pub/subs. A network aware content-based pub/sub would rectify this problem and with the introduction of SDN it became possible to create such solutions. Such pub/sub solutions have an SDN application on the control plane playing the role of the notification service and the network switches themselves playing the role of brokers. Hosts connected to switches can send advertisements and subscriptions to the controller which can add the necessary flows between the end hosts. Now the events are filtered and routed at the network layer and the link between the hosts are created by a network aware entity(the controller). One such a solution is PLEROMA [5] it uses spatial indexing described in 2.4.3 for generating content address called DZs. PLEROMA is discussed in detail in future chapters.

2.7 Consistency in SDN

Every network needs to ensure some quality of service(QoS) factors at all times. The various quality factors or *network invariants* in the network may include the following parameters

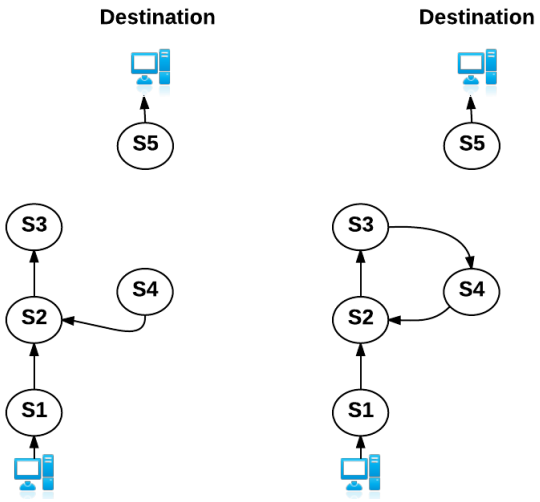

Figure 2.10: The figures depict intermediate states when transitioning from State A to State B described in 2.9. The figure on the left shows a blackhole which results when The outgoing rule in S3 is deleted. The figure on the right shows loop which is caused because the new state rules are added to S3 and S4.

- **Loop Freedom** - There is should be no transient loops in the network path when moving from one consistent state to another. An example of a transient network loop is depicted in 2.7.1
- **Packet Drops Freedom & Blackholes** - Packets are sent out through a link to a switch which has no information as to how to handle a packet. An example of a blackhole is depicted in 2.7.1
- **Memory Limit** - The TCAM memory of a switch can hold only a limited number of flow rules. Hence some SDN controllers may add a limitation on the switch flow count.
- **Packet Coherence** - It refers to a requirement which states that a packet should see only the rules of only one of the consistent states and never a mix of two.
- **Others** - Additional factors can be added or existing factors can be relaxed depending upon the requirement of the applications. For example, pub/sub can provide additional factors such as durability and non duplication of events.

SDN controller sometimes needs to change flow rules(from one state to another) in switches for reasons such as high traffic in some links. The network needs to ensure that the necessary *network invariants* are preserved at all times even when changing from old rules(old state) to new rules(new state). Even if the old and new states ensure necessary Quality, it does not follow that the intermediate states also will provide the necessary Quality. Example how network invariants are violated are is shown in 2.7.1.

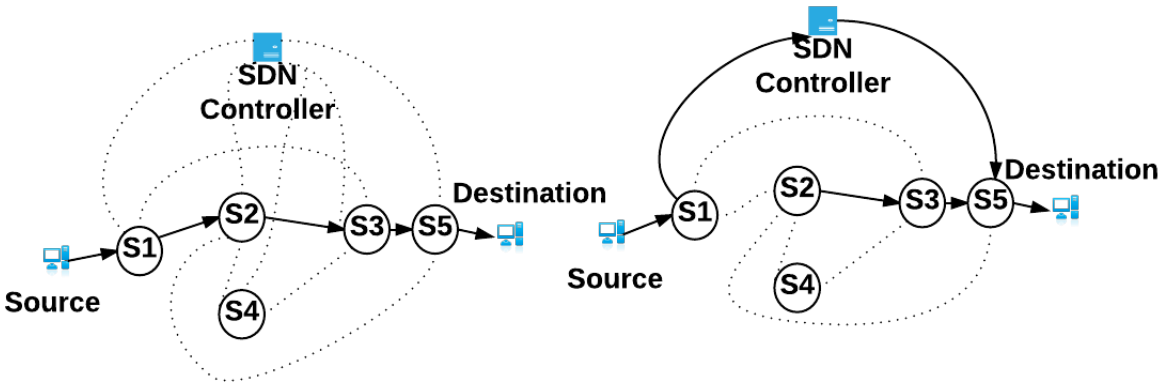

Figure 2.11: The figure depicts adding intermediate controller bridge steps

Sometimes specific strategies are needed for updating from one state to another to ensure QoS. Any update strategy which does not violate network invariants is called a consistent update strategy example [26], [8]. The complexity of preserving each of the network invariants is dependent on the particular strategy used. In general, these update strategies fall under two categories - Stateful and Stateless strategies. The following section discuss the various strategies in detail.

2.7.1 Controller bridge

This is a naive solution for providing a consistent update which uses the controller as the intermediary for routing when moving from one consistent state to another. When the new rules for routing are available - randomly removing the old rules and adding new ones could lead to the network invariants being violated . First, in this strategy, the controller adds intermediate rules to all ingress port connecting hosts to switches 2.11. This intermediate rule sends all the incoming packet to the controller and the controller then sends the packets to the destination switch port using a command called the packetout command. This process is depicted in 2.12. Thus all the intermediate switches between the source and destination switches are replaced by the controller instead.

Next, all the old rules can be expired once the controller has taken over routing and old flows can be made to expire with an idle timeout to make sure that all packets that are routed with old rules have left the network 2.12. Once the old rules have expired new rules are added and then intermediate rules are removed 2.13.

This strategy provides packet drop freedom, loop freedom, and packet coherence. The obvious disadvantage of the approach is that the controller becomes overloaded and line rate performance cannot be achieved.

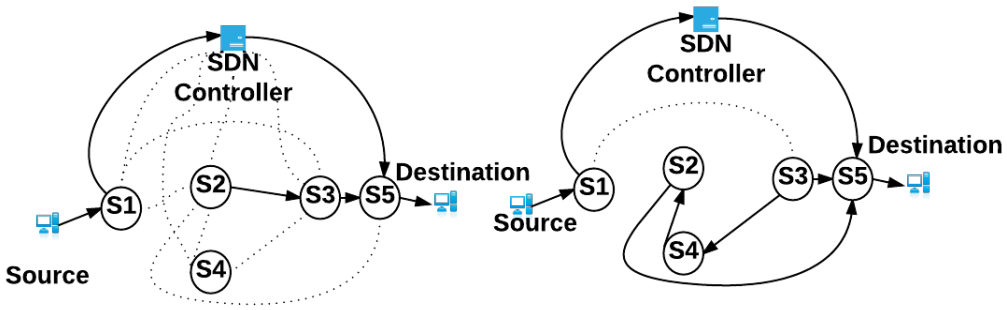

Figure 2.12: The figure depicts removing old state flow rules

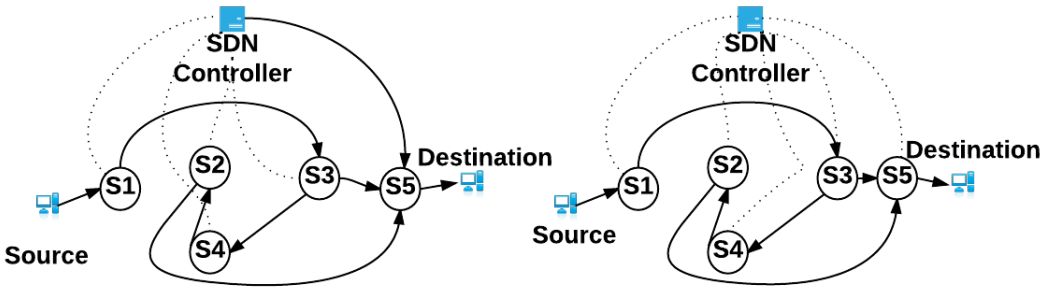

Figure 2.13: The figure depicts adding new state and removing intermediate state flow rules

2.7.2 Stateful Strategies

These strategies inject a state into each of the packets which enter the network at the border switches and the routing rule applied are based on the state of the packet.

Versioning

The strategy described in [8] ensures that all the network invariants are preserved. In this method, all the rules associated to ingress ports connecting host(called ingress gateway ports) and switches inject a VLAN Id to the packets when they are forwarded through these flow rules. All the other flow rules in the network match the packets with the VLAN Id in addition to other match field criteria. Next, new flow rules of the new state are edited have a different VLAN Id in the match field and are added to the switches. Note that at this point no packet would match the flows with new VLAN Id as the packets are tagged with the old VLAN Id. Next, the rules associated with ingress gateway ports are modified to inject the new VLAN Id. This step is atomics to individual rules. Hence the packets which enter the networks with new VLAN Id are routed only with new rules and packets with old VLAN Id with old rules.

This strategy works for both unicast, multicast paths and provides packet coherence property during the transition. However, it may require more than twice the number of flows per

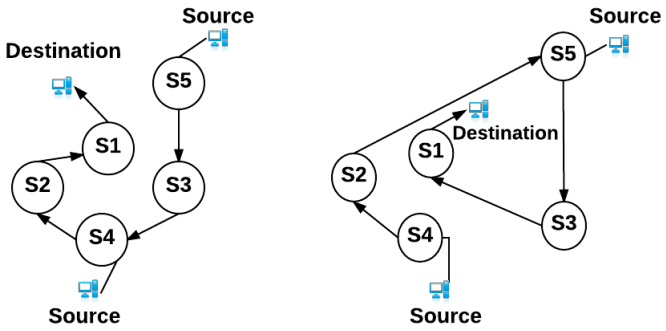

Figure 2.14: The figure on the left depicts the initial State A and one on the right final State B.

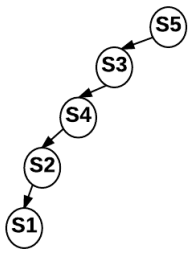

Figure 2.15: Dependency Tree is calculated as described in 2.7.3

switch hence its can be overload on the limited TCAM space. A solution for TCAM space constraints have been addressed in suggested in [27] using iterative update strategy. However, this comes with the disadvantage of increased time required for updating. This strategy also does not provide any lesser level of consistency than packet coherence.

2.7.3 Stateless Strategies

The stateless strategies update the network flow rules from the old state to the new state by deploying the flows in a specific order so that the network invariants remain unaffected. Unlike with the Versioning strategy, the packets are not tainted with a state for the purpose of updates in this strategy.

Ordered Updates

The approach described in [26] has provided a stateless update strategy which can ensure non-violation of different levels and combinations of network invariants. 2.14 represents the first consistent state A and 2.14 represents the new consistent state B. To move from state A to state B, first a destination rooted in-tree as depicted in 2.15 is constructed. Then the updates are executed from destination switch to the source switches updating in parallel along the branched paths. This ensures loop freedom because a switch is updated with new rules only after all the downstream switches are updated.

[26] describes a minimalistic solution using multiple dependency trees(called dependency forest) where multiple nodes can be updated in parallel. The algorithm for loop freedom is given below

- The old state is taken up as the initial state for this algorithm. Then old rules are removed, new rules are added to each switch and a test is run to check if loop results.

Network Invariants	Knowledge required
Eventual Consistency	Always guaranteed
Packet-Drop Freedom	Add flows before removing ensures packet-drop freedom
Loop Freedom	Dependency tree is requried for ordered updates
Packet Coherence	Impossible without versioning

Figure 2.16: The figure depicts adding new state and removing intermediate state flow rules

If not, then the switches are taken as a root of dependency tree and moved to a state called limbo. If yes, then the switch is added to a state called old.

- Next, all children of this dependency tree nodes discovered in the previous step are taken up at the next iteration and the again new rules are added to these switches and it is checked if adding the new rule results in a loop. If yes, then the node is labeled as limbo and added as a child to the corresponding parent. If not, it is labeled as old.
- The above step is run until all the switches with new rules are in limbo. This generates a forest of dependency trees.
- Once the dependency forest has been generated all the new rules can be added and old rules removed in parallel in individual dependency trees.

2.16 also provides different levels of complexity associated with providing different levels of consistency. Notice that 2.16 mentions that this algorithm cannot provide a solution for a maintaining packet coherence.

Dionysus

Dionysus is a consistent update strategy with a goal to achieve high consistent update speeds. The ordering of updates into the following phases - dependency graph based on the current state of the network, then the update scheduler schedule the flow updates and recompute the dependency graph based on the current state of the network. The dependency graph of Dionysus is an extension of the dependency graph provided by [26] as described in the previous section. But instead of ordering update to achieve consistency it uses versioning approach for providing packet-coherence, while the dependency graph is formed based on network traffic, memory utilization during transitions. The scheduling of the updates is based on the dependency graph generated. [28]

All the stateless strategies outlined above ensures the consistent update of unicast paths but not multicast paths or for IP-prefix based routing. The aim of this thesis is to provide a consistent update solution for such multicast paths which use IP-prefix based routing in a content-based pub/sub built on SDN. The following chapter provides a deeper description of this content-based pub/sub, various categories of updates in relation to this application.

3 In-Network Content Filtering And Multicast Path Consistency

The goal of this chapter is to provide a brief description of content-matching, routing in PLEROMA and different case of network updates which arise in its operations and optimization[5]. This chapter provides an overview of the environment and immediate background for the problem statement discussed in the next chapter.

3.1 In-Network Content Filtering

Content filtering at switches requires two factors- namely

- A storage unit which can persist this filtering information. The filtering logic is described in the form of flows in tables called flow tables as mentioned in 2.5. The storage unit which contains the flow tables is called the TCAM or Ternary Content Addressable Memory. All SDN-compatible switches have TCAM as mentioned in 2.5. An important property of any Ternary memory unit is that it can contain three states 0, 1, and X. The 'X' state of TCAM represents a wild card entry ie., it can refer to any of the states.
- A strategy to express content filters in the form of flows. One such strategy to achieve this goal is described in 2.4.3. This strategy provides a method to express multidimensional event spaces as a binary string called DZs. The DZ of an event-space/event can be appended to an of IP address with a subnet mask. When a DZ string is appended to an IP address, it is prefixed with the IP subnet corresponding to an IP multicast address. An IPv4 multicast address can be anything between 224.0.0.0 to 239.255.255.255. Thus, when a DZ is appended to an IPv4 address, atmost the first 8 bits correspond to the multicast subnet and is common for any resulting IP address formed after DZ is appended, while the remaining 24 bit can be reserved for the DZ string. For example, the IP multicast subnet can be 224.0.0.0/8. When the event space {00} is appended to this IP multicast address the resulting IP address is 224.0.0.0/9. This process is depicted in 3.1

The IP address formed by appending DZ to IP multicast subnet is added as the match field(called the destination IP field) of a flow. This flow can then be added to the flow table of a switch. An incoming event is matched to the flows of the flow table based on the logic provided in 2.7 and each event can be output through multiple ports. However, for the above content filtering to work, all the events should have a destination IP address which is calculated

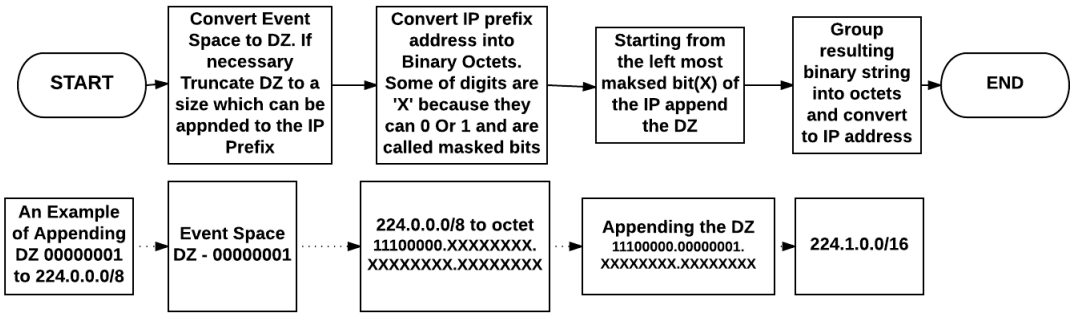

Figure 3.1: The process for appending DZ to an IP address with subnetmask

the same way the destination IP match field of a flow is calculated. Line rate performance of such in-network filtering is very high because the content filters are in the switches at the network layer and because the TCAM memory is specialized to find a matching flow for events at packet rate.[5]

PLEROMA described in 3.2 uses the above approach for specifying its content filters in the data plane. An important point to mention here is that the event space can be partitioned in more than one way 2.4.3 and PLEROMA uses this flexibility to optimize its performance parameters such as bandwidth efficiency.

3.2 PLEROMA

PLEROMA[5][6] is a content-based pub/sub which provides in-network content filtering. The model of PLEROMA is based on the model described in [4]. It consists of a controller playing the part of the notification service and switches playing the part of distributed brokers whose function is to route and filter events. In PLEROMA, the DZ representation of event space is used by advertisers in their advertisements to indicate the event-subspace to which events they will publish belongs and subscribers also use the DZ in their subscriptions to indicate their regions of interest in the event space. These subscriptions and advertisements are sent to pre-defined IP address which corresponds to the controller's location in this network. On receiving the advertisements and subscriptions, the controller uses its knowledge of the network topology to compute the content filters(flow rules) for the switches so that there is a path between each of the advertisers and interested subscribers so the events can reach all interested subscribers. Once advertisements are sent, advertisers can publish events whose destination IP address is calculated from DZ of the event content. The events are routed to interested subscribers using the flow rules deployed by the controller in switch Flow tables. The following section describes the functionality of PLEROMA in more detail.

3.2.1 Covering relationship of DZ

The most important property of DZ which is necessary for IP prefix based routing is - a shorter DZ 'A' covers a longer DZ 'B' if and only if 'A' is the prefix of 'B'. For example, DZ {0} covers DZ{01} which in turn covers {011}. This covering relationship is maintained even when a DZ is converted to an IP address. For example, consider a DZ{0} 'A1' with a corresponding IP address 224.0.0.0/9 and DZ01 'A2' with an IP address 224.64.0.0/10. Now a DZ 'A3' 010000000000000100000001 with an IP 224.64.1.1 is covered by both DZs A1, and A2 and in prefix based routing, an event with event address as A3 has its destination IP set to A3's IP will be matched to both IP addresses of A1 and A2. Hence this covering relationship is extremely useful for routing.

Another important property of DZ is granularity. A shorter DZ covers more event space than longer DZ. Hence the granularity of a DZ increases with its length. However, the expressive of DZs cannot be utilized to their maximum potential as the number of bits of a DZ that can be appended to an IP address is limited.[5]

3.2.2 Spanning Trees In PLEROMA

In PLEROMA, the network controller divides all available event space based on DZ into *Partitions* and assigns a spanning tree of the network graph to each of the partitions. Since partitions are formed based on DZ, each of the partitions is associated with a DZ, say 0 or 01, etc. The entire event space should be covered by the union of the DZs of all the partitions, and none of the DZs assigned to partitions can overlap. For example, 0, and 01 cannot be DZs assigned to different partitions of PLEROMA. Now, when an advertisement or subscription is received by the controller, one of the following cases is true - either DZ of the advertisement/subscription is completely covered by the DZ of a partition or the DZ of the advertisement/subscription covers DZs of several partitions. In case of the former, advertisement/subscription is handled by the partition which covers its DZ. In case of the latter, the DZ of advertisement/subscription is split up so that it is covered by the individual partitions independently and the resulting set of advertisements/subscriptions are handled as separate advertisements/subscriptions by the individual partitions.

Having logically separate partitions has several advantages. For instance, the paths connecting publishers to interested subscribers are along the spanning tree of the partition to which the subscribers and publishers belong. Hence, the flows of different partitions can never match to the same event even if all the partitions use the same spanning tree in the network. Another advantage is that the flows would never result in loops when using a tree topology. A disadvantage of this approach is that when there is high network traffic along some links in network some partitions may need to be moved to a different spanning tree. Obviously, the network invariants must not be affected when such move takes place. We provide a consistency algorithm for this case as part of the thesis in subsequent chapters.[5]

The following sections describe how these partitions handle advertisements and subscriptions.

3.2.3 Handling Advertisements, Subscriptions and Events

Any publisher must declare its intent to publish events by sending an advertisement message. The advertisement messages are addressed to a pre-defined controller IP address and it contains the DZ of the event sub-space to the events the publisher intends publish will belong. For understanding how the controller processes advertisements let us consider a Network of 4 switches with a topology graph as shown in fig 3.2. As shown in the figure, this topology is divided into two partitions P1, P2 having DZs {0},{1} each with a spanning tree. Now, when the controller receives advertisement messages A1{00}, and A2{01} from hosts H1, H2 respectively, as shown in 3.2, it starts searching for the partition that can handle the advertisements with the strategy described in 3.2.2. In our example, both A1 and A2 are handed over to P1. Next, it checks if there are any matching subscriptions. For our example, there are no subscriptions yet, so it stores the advertisements in respective partitions. Now the publisher can publish data since it has sent its advertisement.[6]

To receive interested events a subscriber has to send a special message called subscription to the same pre-defined IP like the publishers. On receiving a subscription, the controller again assigns it to one of the partitions based on the DZ of the subscription as described in 3.2.2. 3.2 depicts hosts H3 , and H4 which sends subscriptions S1{000}, and S2{0} respectively. In both cases, the controller lets P1 handle the subscriptions. Next, the controller finds advertisers which match the subscriber using the DZ of advertisement and subscription messages. For this purpose, it needs to find overlapping DZs. A DZ, say DZ1, overlaps with another DZ, say DZ2, if DZ1 equals DZ2 or DZ1 covers DZ2 or DZ2 covers DZ1. So, In our example, S1 is matched to A1 and S2 to both A1 and A2. In case there are overlaps found, the next step is the calculation of flows for connecting publishers to subscribers. This step is used to handle only one subscription at a time. For each subscription with overlapping advertisements, the controller fetches the list of switches in the path from publishers to the subscriber. For each switch in the calculated path, the controller checks if there is a flow with the same ingress port as in the path and a DZ(destination IP address) which overlaps with the DZ of the subscription. The overlap can cause one of the following cases

- The DZ of a flow completely covers or is equal to the DZ of the subscription, has the same input port and output ports as in the current path. In this case, there are no flows to be added to the switch.
- The DZ of a flow completely covers or is equal to the DZ of the subscription and has the same input port but not the same set of output ports as in the current path. In this case, a new flow with a destination match criteria as the IP address formed from subscription DZ, ingress port as the source port match criterion, a priority higher than the matched flow's priority and output ports as the union of the output port of the matched flow, and the path is created. The new flow is then added to the switch.

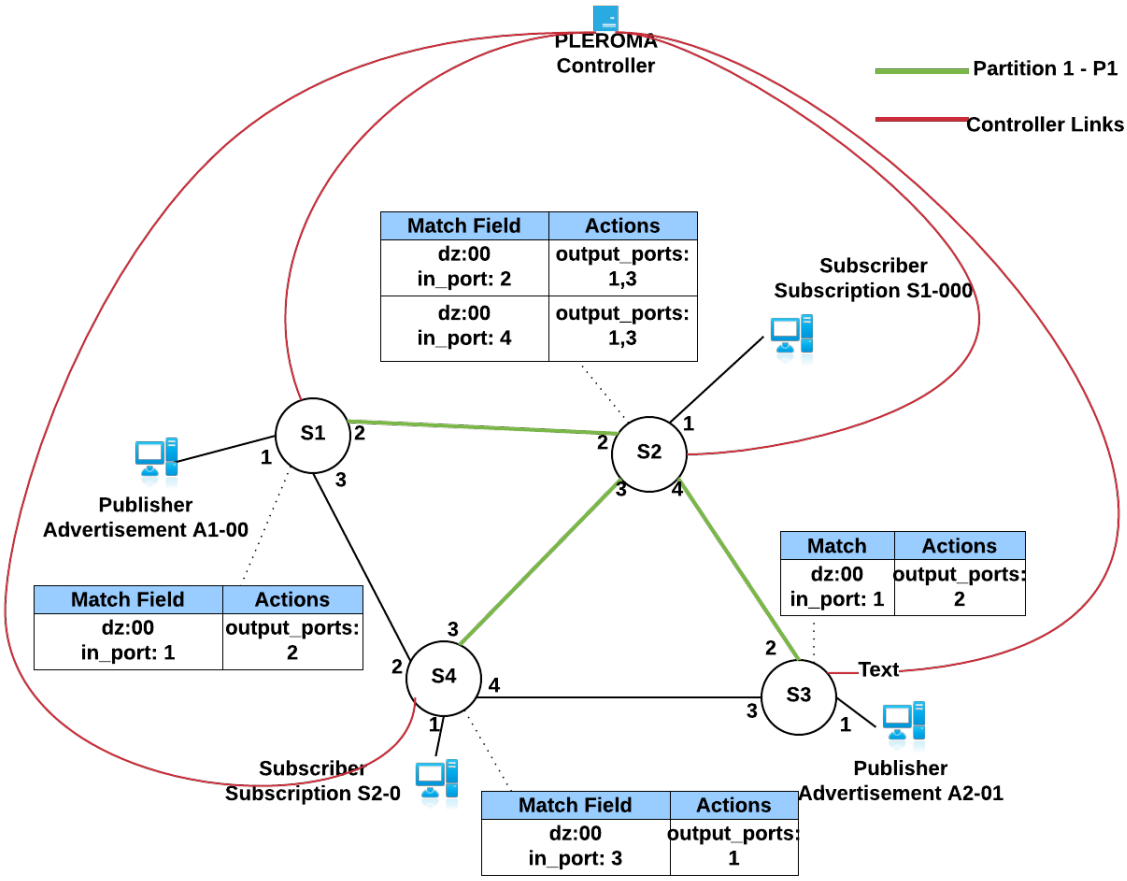

Figure 3.2: The figure Depicts PLEROMA data plane with 2 publishers, 2 subscribers, 4 switches. Assume that, it is possible to only append 2 bits to IP address in the data plane.

- The DZ of a flow only partially covers the DZ of the subscription and has the same ingress port as in the path. Then, the DZ of the flow is subtracted from DZ of the subscription. This results in a list of DZs. Each of these DZs is sent through the same algorithm with the ingress port. For example if DZ of the flow is {01} and DZ of subscription is {0} then {0} is subtracted from {01} which results in [{00}]

- The DZ of the subscription completely covers the DZs one or more flows, it has the same ingress port for as the flows, and all the output ports of all the matched flows are contained in the path. In this case, all the old flows are removed from the switch. Then, a new flow is created with subscription's DZ(in the form of IP address) as the destination IP match criterion, ingress port as source port match criterion, and output ports as the outgoing port of the paths. The new flow is then added to the switch.

- The DZ of the subscription completely covers the DZs of one or more flows, it has the same ingress port for as the flows but not all the output ports of all the matched flows are contained in the outgoing ports of the current path. In this case, a new flow is created with destination IP address formed from subscription DZ as the source IP match criterion, ingress port as the source port match criterion, a priority lower than all the matched flows, and output ports as the outgoing ports in the path. The new flow is then added to the switch and all the matched flows are edited to also output along the outgoing ports of the current path.

- There is no overlap with any of the flows found because there are no flows with same ingress port or there is no match to DZ of any of the flows. In this case, a new flow is created with a subscription DZ as the source IP match criterion, ingress port as the source port match criterion, a priority lower than all the matched flows, and output ports as the outgoing ports in the path.

The above cases are checked for each of the switches in the calculated path. In case there are no overlaps then the subscription is stored in the partition for future advertisements. 3.2 depicts the paths calculated for A1,A2,S1,S2 using the above algorithm. One thing of note here is that, although the above algorithm is matching subscription to advertisements the opposite is also possible in case there are already subscriptions in the system. After subscriptions have been processed by the controller and the corresponding flows are pushed to the switches, subscribers will start receiving events. 3.2 depicts an example of an event being transmitted through the network using prefix based routing.

[5] proves that the rate of packet handling is extremely high. But PLEROMA suffers from bandwidth inefficiency as described in [29] [7]. This bandwidth inefficiency is observed as false positives in the network and at the subscribers as described in 2.4.4. The following section provides a brief overview of a strategy to improve the bandwidth efficiency as described in [29].

3.3 Bandwidth Efficient Indexing In PLEROMA

The objective of this section is to provide a brief overview of the solution proposed in [7], [29] to improve bandwidth efficiency of PLEROMA and describe the scenario why this solution is a case which requires a consistent update strategy in PLEROMA.

3.3.1 Limitation Of Content Filter Expressiveness In PLEROMA

The expressiveness of a content filter in PLEROMA depends on the length of the DZ that can be appended to an IP address which is used as the filter at the switch. This limitation of DZ length leads to a limited expressiveness of content filters. The actual granularity of the content filter then depends on the number of dimensions and the range of each dimension in the event space. Greater the number or range of dimensions lesser is the effective granularity of content filters. For example, consider a 2-dimensional event space as shown in 3.3. The figure on the left(called Case A) has range 0-100 for both dimensions and figure on the right(called Case B) has range 0-200 for both dimensions. The partitioning method used for both the event spaces is to allocate the same number of bits, ie 1, for each dimension and within each dimension to partition the range equally. Hence the event space of the resulting partitions is shown in 3.3. The figures 3.3 also depicts a subscription S (dim1:[10-20], dim2:[10-20]) which is interested in the same event-subspace in both Case A and Case B. Of note here is the fact that the content filter DZ which is used at the switches in both the cases is {00} because the length of content filters is limited 2 bits. Hence, a subscriber subscribing to S will receive all events from bigger event space in Case B than in Case A. This in turn results in higher false positive rate for Case B that Case A. From the above stated example, it is clear that granularity of the content filters and bandwidth efficiency depends on the size of event-space because of the fixed limitation on DZ length.

As another example, consider if in Case A the length of DZ that can be appended to an IP address is reduced from 2 bits to 1 bit. Then for the same subscription S the events space covered by the content filter will be the event space represented by {0}. This results in a higher false positive rate because a subscriber subscribing to S will receive all events from bigger event space. This example makes it clear that there is a direct correlation between length of DZ and bandwidth efficiency.

In PLEROMA, the bounds of all the dimensions of the event space in PLEROMA is a constant and the length DZ that can be appended to IP address is also constant. With the above preconditions in place, [7] suggests the following possibilities to improve bandwidth efficiency.

- Minimizing the size of the event space available for partitioning through some method
- Using the flexibility of partitioning event space in more than one way as described in 2.4.3.

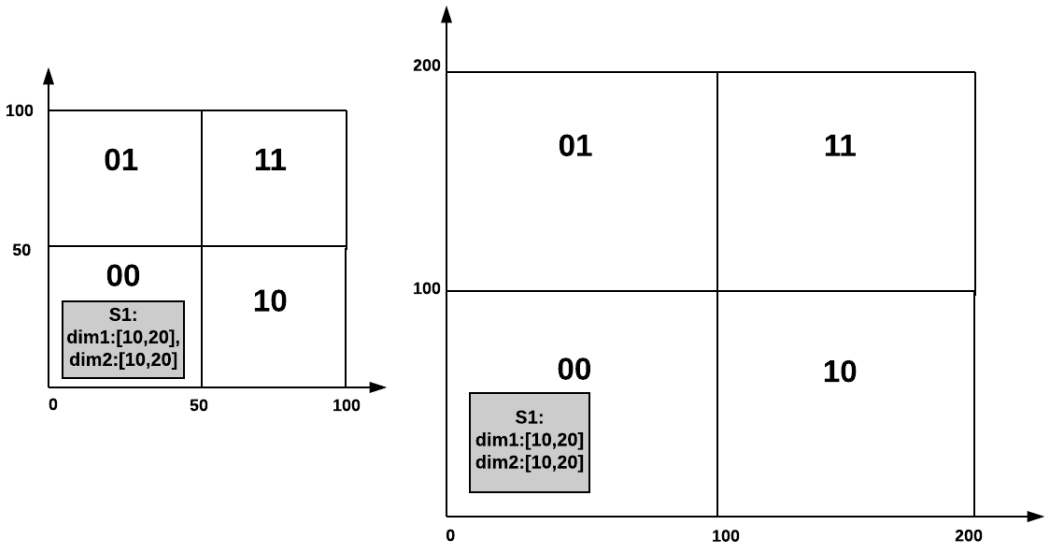

Figure 3.3: The figure on the left depicts an event space with both dimensions ranging from 0-100. The figure on the right depicts an event space with both dimensions ranging from 0-200

[29], [7] suggests strategies to improve bandwidth efficiency based on the above stated possibilities. However, the solutions proposed in [7] require further information about subscriptions, and event distribution in event space. The following section discusses those solutions in detail.

3.3.2 Workload-Based Indexing Of Event Space

Workload based indexing as described in [7] proposes to improve bandwidth efficiency by reducing the event space available for partitioning. This strategy proposes to partition only the part of event space covered by subscriptions. This consists of two steps - clustering the subscriptions in events space and partitioning within the clusters. The algorithm is as follows.

- Let event space be represented by ES.
- ES is populated with subscriptions.
- The subscriptions are grouped using similarity based clustering as described in [30]. Such clustering algorithms are used for grouping individual data points for different purposes. The actual clustering algorithm used for this purpose is k-mean clustering. In K-means clustering, K is a user defined value which corresponds to the number of clusters the data points are grouped into. First, K number of means are initialized as centroids. Then, the data points for the cluster are calculated based euclidean distance of the data points from the centroid. The actual centroid for each cluster is re-calculated.

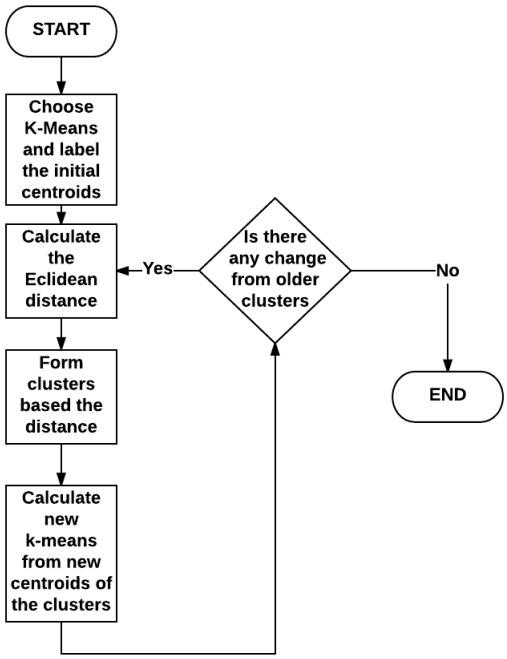

Figure 3.4: K-Means Clustering Algorithm

The calculated centroid of each cluster is then is made the new set of k-means. The above steps of grouping data points into clusters and recalculating centroids are repeated until k-means between two successive iterations remains constant. 3.4 depicts k-means algorithm.

- Once the subscriptions are clustered, they are enclosed into one or more hyperrectangles(rectangles for higher order of dimensions). The set of hyperrectangles enclose all the subscriptions and are called minimum bounding rectangles(MBRs) in [7]
- Next, each MBR is associated with a unique binary id and the partitioning now takes place only within the MBRs and DZs of each MBR is prepended with the unique binary id of the MBR to make the DZs of different MBRs unique. The MBRs are depicted in 3.5.

Hence the overall size of events space is reduced to a much smaller region which covers all the subscriptions. Thus the granularity of content filters is increased because the size of event space considered for partitioning is effectively reduced.[7]

3.3.3 Dimension Selection Algorithms For Events Space Partitioning

Dimension selection algorithms proposed in [7] are based on the observation that not all dimensions are equally important when it comes to content filtering ie., it not necessary to allocate the same number bits to each dimension when partitioning or rather, greater

granularity can be achieved if more bits are assigned to some dimensions than others and some dimensions need not be allocated with any bits at all.

Event Variance-Based Selection

This algorithm decides the dimensions which are necessary to have better granularity for content filtering than just using all the dimensions based on event distribution in the event space. The algorithm is as follows

- Let event space be represented by ES.
- Let D be the set of all Dimensions in ES. DS be the subset of D.
- Populate ES with all the events and let E represent the set of all events.
- The sum of the variance of the value of each event along each of the dimensions is calculated with the formula $\Sigma(x_i - x_m)^2/|E|$ where x_i is the value of the event along i^{th} dimension. The calculated value is called the selectivity factor of that dimension. This selectivity factor is higher for a dimension if the events are distributed more along the entire length of that dimension.
- Only those dimensions with higher selectivity factor are added to DS.
- Then, Event space partitioning takes place only with DS instead of D which means that more bits are allocated now for the dimensions in DS than they were before. This increases the granularity of content filter because dimensions with higher selectivity factor have more events distributed along its length. Hence, increasing the number of partitions along the length of a dimension by increasing the number of bits allocated to it also increases the granularity of that dimension as depicted in 3.6.

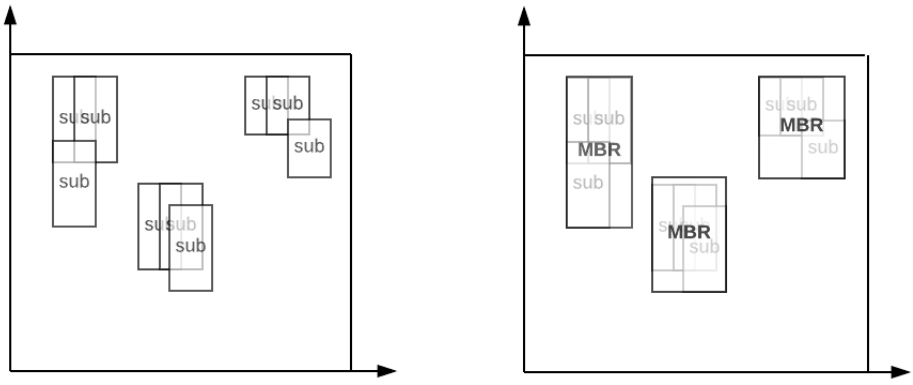

Figure 3.5: Workload-Based Indexing and calculation of MBR

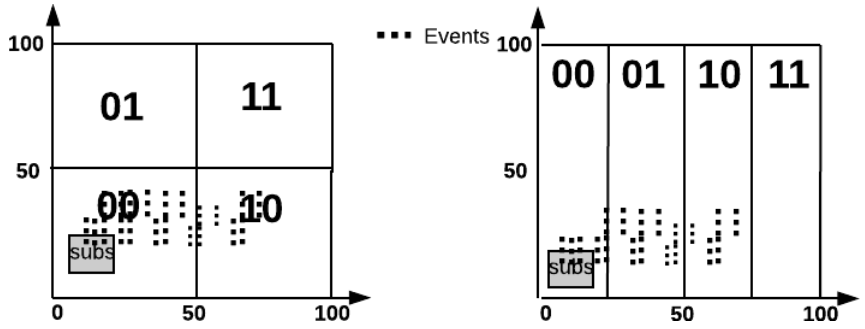

Figure 3.6: A depiction of dimension selection with Event Variance-based Selection algorithm

Both the figures in 3.6 depicts a 2-Dimensional event space with event distribution in which the maximum number of bits that can be appended to an IP address is 2 bits. 3.6 also depicts a subscription sub1 in both the figures. The figure on the left depicts a partitioning strategy in which an equal number of bits are allocated to both dimensions and the one on the right depicts event space partitioned using the Event Variance-Based Selection algorithm. In the latter case, the 2 available bits are allocated to the horizontal dimension which increases selectivity because sub1 now match an event subspace which causes lesser number false positives as the number of unnecessary events is obviously lesser.

Subscription Matching and Event Count-Based Selection

An interesting observation noted in [7] is that, if a large number of subscriptions overlap along one of the dimensions then an event matching a subscription along that dimension has a higher probability of matching several more subscriptions. If this happens to be the case, then that dimension has lesser granularity for content filtering and hence its ability to reduce false positive is low. This concept is depicted in 3.7. The following algorithm decides the dimensions to which bits need to be allocated for calculating DZs(content filter) based subscription overlap and the total number events which are matched to these subscriptions. The algorithm is as follows

- Let the event space be represented by ES. Let E be the set of all events and S be the set of all subscriptions.
- Then, for each event e in E, the subset of S which matches the event along each dimension d is added as an element to the set S_d^e
- Next, the importance of a dimension d is calculated with the formula $\Sigma(S_d^e)/(|E| * |S)$. This formula gives a value between the interval [0,1] which denotes how important a dimension is for granularity of content filters. Greater the value greater is its importance. This value is called the selectivity factor.
- Then, event space partitioning takes place only with a subset of D whose selectivity factor is high. This means more bits are allocated for the selected dimensions than they were before.

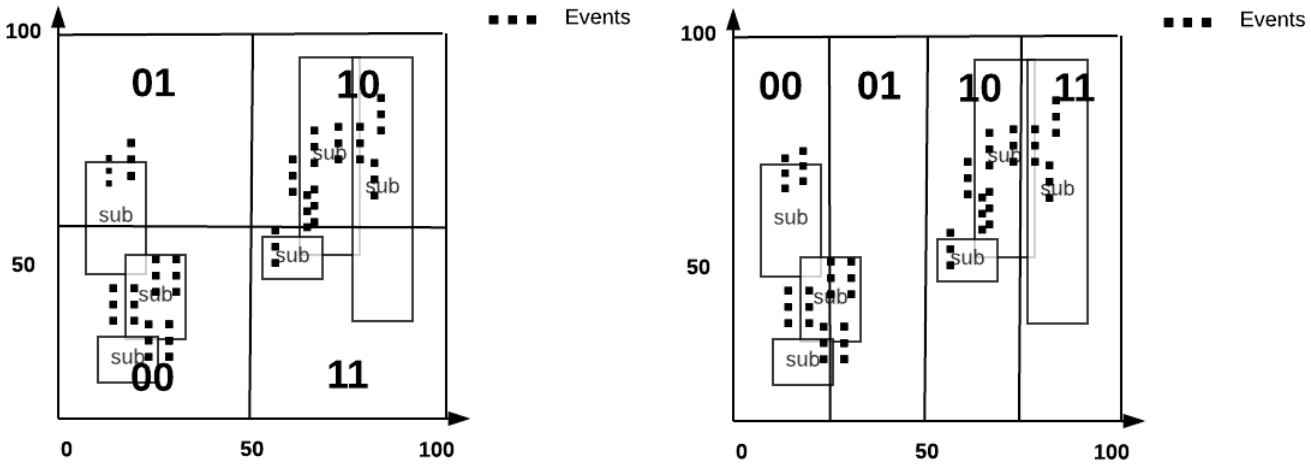

Figure 3.7: A depiction of dimension selection using Subscription matching

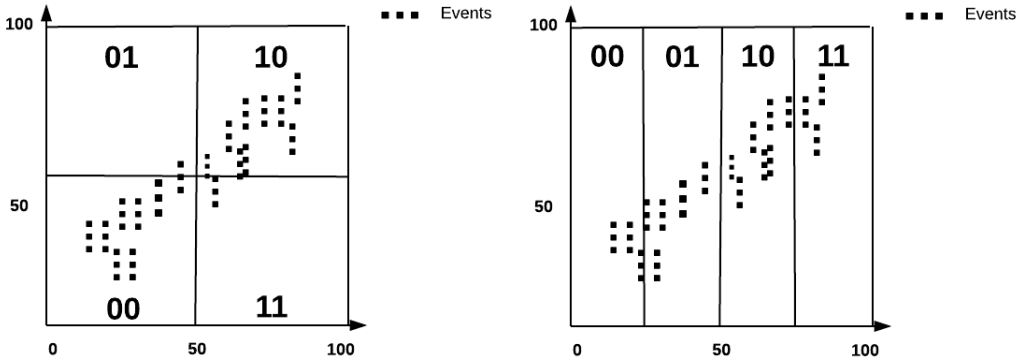

Figure 3.8: A depiction of dimension selection using correlation based selection

Correlation-Based Selection

This strategy is based on the fact that often multi-dimensional dataset has several dimensions which are related to each other. For example, consider a sensor which collects temperature and pressure in a room. It can be shown that the dimensions are positively(in case of sealed room) or negatively (in case of rooms with lots of windows) correlated to each other. When the dimensions are related to each other in this manner, it becomes redundant to allocate the same number of bits to all the dimensions. Allocating the redundant bits to a subset of unrelated dimensions increases the overall granularity of the content filters.[7]

3.8 depicts a 2-Dimensional event space where both the dimensions are positively correlated and event space is allocated with 2 bits.for appending with IP address. The figure on the left depicts an event space where all dimensions are partitioned using the same number of bits and the one on the right with an event space where only one dimension is allocated with both

bits. It can be seen that the subscription in the latter case has more granularity than the one in the former case.

3.3.4 Workload-Based Indexing As Implemented In PLEROMA

[29] implements the solutions suggested in [7] for improving the bandwidth efficiency in PLEROMA. The implemented solution samples the current subscriptions and events over a time interval in the PLEROMA controller. Whenever the controller detects a change in the distribution of the sent events or subscriptions in the event space, it tries to optimize the allocation of bits to the dimensions(called Dimension Optimization or DO from here on) using a combination of above stated algorithms. Once the dimensions have been optimized the DZs of advertisements and subscriptions are recalculated with the new partitioning technique. An example of how the dimension optimization affects the flows in a switch is depicted in 3.9.

As shown in 3.9, the resulting DZs are invariably different than they were before, the flows whose destination IP addresses are calculated from old DZs are also no longer valid and new flows are calculated with new DZs. Next, they are deployed to switches by the controller. However, this transition as implemented in [29] is not consistent with the Quality of Service promised by PLEROMA which requires that the PLEROMA should not introduce loops, blackholes, and event duplicates in the network layer when deploying the new DZs. The following chapter discusses the need for consistent update solution for the implemented Workload Based Indexing.

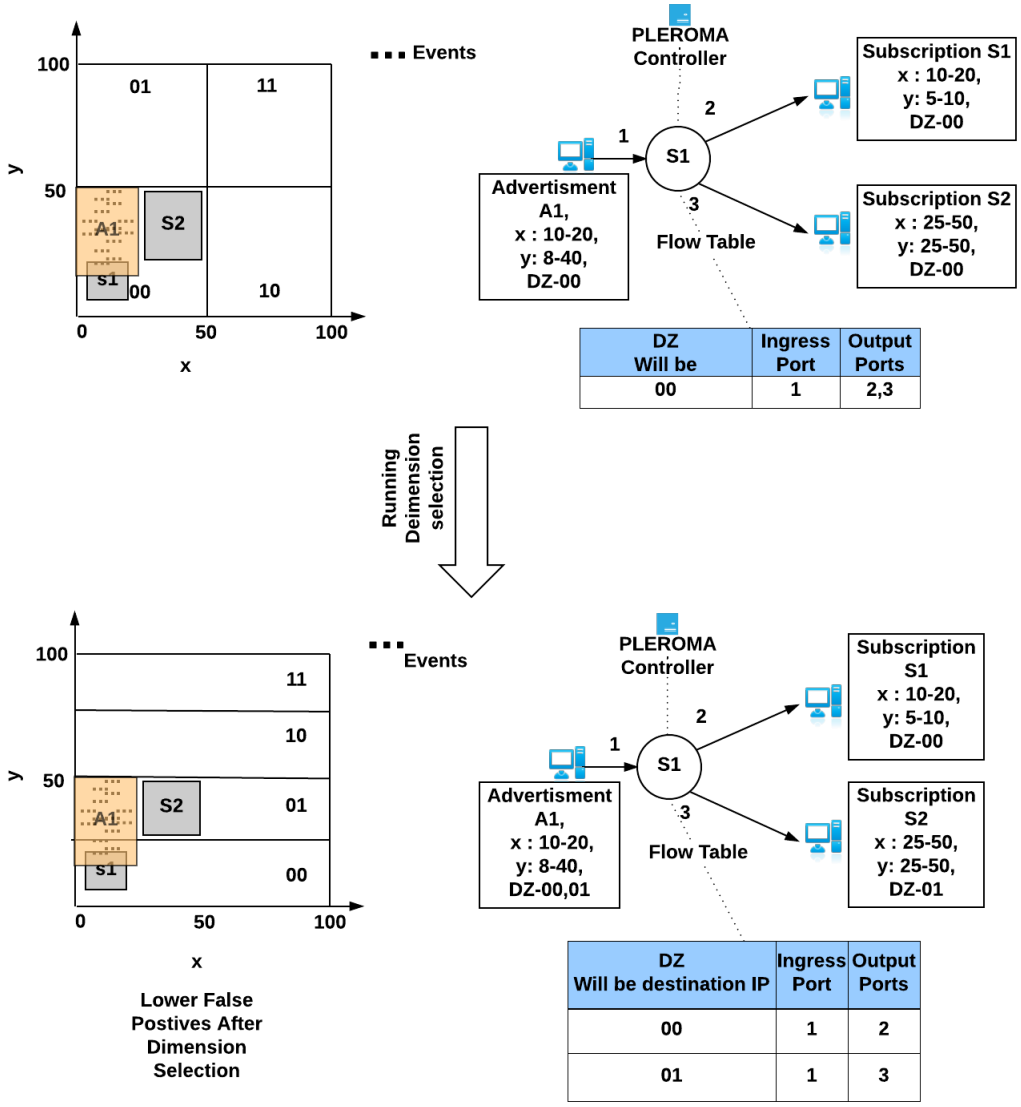

Figure 3.9: An example of dimension selection in the network with one switch, one publisher and 2 subscribers where PLEROMA has 2 bits available for appending to an IP address for content filtering.

4 Problem Statement

Before we can provide the problem statement for this thesis which deals with consistent update strategies in PLEROMA. We need to answer the following questions

- What is the QoS promised by PLEROMA or rather, what are the network invariants that need to be protected from violation?
- Are there scenarios which would require a consistent update in PLEROMA?
- Why not use generic consistent update strategies described in 2.7.2, 2.7.3?.

4.1 QoS Requried in PLEROMA

PLEROMA promises the following QoS in the data-plane.

- Loops freedom - refers to freedom from transient loops in the data plane when transitioning from one consistent state to another. For examples refer to the section 2.7.
- Blackhole Freedom - During network updates some switches do not have information of how to handle events and drops the events. Under normal circumstance, the events would have been output along one of the switch's outgoing links. This scenario is called black hole freedom. Such scenarios should be avoided
- Duplication freedom - An event should not be duplicated in the network.

4.2 Scenarios which require a Consistent Update

The various scenarios in PLEROMA which require a consistent update strategy in the data plane are

- when a spanning tree of a partition needs to be moved to a different partition - The Network Invariants to be preserved by PLEROMA during this update are - no loops, blackholes, and no event duplication.
- when the flows for new subscriptions/new advertisements are to be added - The Network Invariants to be preserved by PLEROMA during this update are - loop freedom and no event duplicates. This case is not discussed in detail since using spacing trees for individual partitions when calculating paths precludes loops and duplication.

- when the DZs are changed as a result of dimension optimization - The Network Invariants to be preserved by PLEROMA during this update are - no loops, blackholes, and no event duplication.

4.3 Other criteria to consider for consistent data-palne updates

Before we discuss any consistent update algorithms, we need to understand the nature of the paths formed by the flows in the data-plane by PLEROMA. PLEROMA, like any other pub/sub architecture, provides many-many communication pattern and since it does in-network filtering, the paths from publishers to subscribers are multicast paths. In addition, these paths use IP-prefix based routing in the data plane. This is another important criterion to consider when choosing a consistent update strategy for PLEROMA because it adds additional constraints to some consistent update strategies. [5][31]

4.4 Generic Consistent Update Strategies for PLEROMA

4.4.1 Limitations Of Ordered Update Strategy

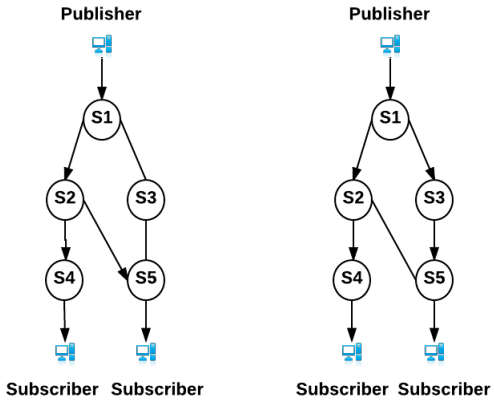

Figure 4.1: Current and new multicast paths.

The ordered update strategy discussed in 2.7.3 provides a generic update solution for unicast paths. However, with this strategy it is impossible to provide the QoS required by PLEROMA because for some multicast paths, it is impossible to find an ordering of flow updates which can ensure duplication freedom and packet drop freedom at the same time. This observation was made in [31]. An example for this claim is provided in 4.1, 4.2, 4.3.

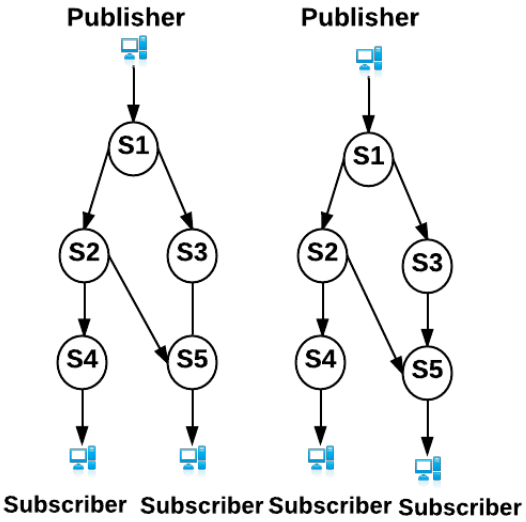

Figure 4.2: One of the two ordered update solutions for the depicted multicast path in 4.1 causes event duplication

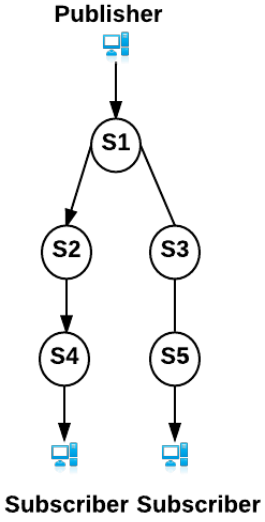

Figure 4.3: One of the two ordered update solutions for the depicted multicast path in 4.1 causes events to be dropped

In 4.1 let the figure on the left represent the old state and let the figure on the right represent the new state. 4.2, 4.3 and depicts how irrespective of the order in which the old state is updated to the new state, it results either in the events being duplicated or event being dropped. It becomes impossible to find a solution which offers both packetdrop-freedom and duplication if and only if there is a shift of the switch which multicasts events. Such a shift is called replicator shift. Only a replicator shift would result in the impossibility solution. As result of the above scenario we cannot use an ordered update solution in PLEROMA for scenarios such as moving to a new tree.

4.4.2 Versioning Strategy

The versioning strategy discussed in 5.1.1 provides a strong consistency of packet coherence. Hence, it can be used for all the update scenarios of PLEROMA. But the disadvantage of using this approach is that the number flows per switch is nearly doubled during transition. This can prove burdensome when TCAM memory usage needs to be kept to a minimum. Hence solutions which require lesser TCAM memory usage are required.[27][8]

4.5 Problem Statement Definition

The objective of this thesis is to provide a consistent update strategy for various update scenarios in PLEROMA such as moving the tree of a partition and change of DZ during

dimension optimization. Having taken into consideration the QoS required by PLEROMA and the generic update strategies already available, we proceed to describe the consistent update algorithms for different scenarios in PLEROMA in the following chapters. Future chapter described the advantages and disadvantages associated with each of the strategies.

5 Consistent Update Algorithms For PLEROMA

The consistent update algorithms proposed for the different scenarios are not the same. The algorithms described in this chapter are optimized for the individual needs of each case. They fall into two categories - one set of algorithms for the consistent update of dimensions during dimension optimization and another set of algorithms for moving the trees of a partition.

5.1 Consistent Update Algorithms For Dimension Optimization

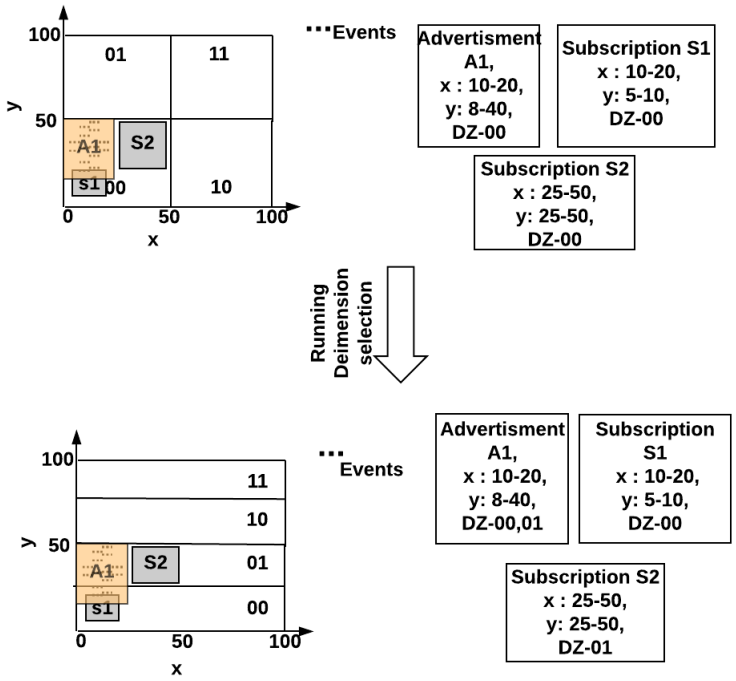

Figure 5.1: An example of change of DZs which results from dimension optimization

When the dimension optimization algorithm described in 3.3.4 is executed, it returns a more optimal partitioning technique for the current distribution of subscriptions and events in event space. Hence a new set of DZs is available to PLEROMA controller as depicted in 5.1.

A change of DZs which results because of dimension selection/optimization can be understood better using a translation analogy. For example consider a small region in an n-dimensional event space represented with a DZ which is a binary representation calculated using the partitioning represented in the figure on the top in 5.1. This DZ(called old DZ) can be considered to be a content address written in a particular language, say Lang A. Another DZ(called new DZ) can represent the same region of the same event space when a different partitioning technique, as shown in the figure on the bottom in 5.1, is used. The new DZ is an address of the same location described using another language which has same alphabets as the original language. The alphabets, in this case, is 0 and 1. Since the content filters in the switches are represented using the IP addresses formed from DZs, the IP address for an old and new DZ will be different even though logically they may be representing the same region in the event space. What all this means to the publishers and subscribers is that while the publishers are publishing the same content they did before but the content address should be formulated with the new partitioning method, subscribers are interested in the same event space, and the filters(Flows) in the SDN switches between the publishers and subscribers will be different after dimension optimization. Hence, when the controller moves from one partitioning technique to another, at the very minimum the publishers need to be notified of the change in the partitioning method(called a change of DZs from here onwards) because they need to calculate the DZ of event-content with the new partitioning method. Hence, any algorithm which proposes a consistent change of DZs must be able to

- Calculate the flows which comply with new DZs
- Send notification of the change in partitioning technique to publishers.
- Preserve the network invariants such as loop freedom, blackholes, and event duplication freedom during the transition from the old state with old DZs to the new state with new DZs.

The following algorithms either describe an existing solution(versioning) or proposes new ones to fulfill the three above criteria during a change of DZs.

5.1.1 Versioning

This method is based on versioning algorithm described in 2.7.2. The steps in this algorithms are explained below [8]

1. The publishers send events injected with version id in VLAN field of the IP packets which corresponds to events. The version id corresponds to the old partitioning technique(or rather old DZ version)
2. All the flows deployed to a switch match all incoming events with VLAN Id in addition to the destination IP address(content filter) and ingress port. Hence the routing and filtering of the events are based on the version of the DZ in addition to their content address.

3. Once the DZ optimization module of PLEROMA described in 3.3.4 performs a dimension optimization based existing distribution of subscriptions, and events and returns the optimized partitioning of the event space.

4. Using the optimized partitioning method, new Flows with IP address corresponding to the new DZs for existing advertisements, and subscriptions are calculated by creating a new logical partition for the same event space partition in the controller with the same spanning trees as before and re-advertising/re-subscribing using existing advertisements, and subscriptions of the original partition with new DZs. The calculated set of flows with new DZs is not deployed to the switches.

5. A new version Id is designated for the new partitioning technique. All the new flows are tagged with a VLAN Id match fields using the new version Id as the match criterion.

6. Next, all the flows with the new DZs are pushed to the switches.

7. No event sent by any publisher at this point will match the new flows because of VLAN field of the new flows have a different value than the ones used by the events published by the publishers.

8. A *change of DZ version notification* is sent to all publishers so that they can calculate the event content address(DZ) with the new partitioning scheme and tag the events with the new version Id in their VLAN field.

9. Once all the events published with the old VLAN id are out of the network, the old flows can be removed. This is achieved by using the idle timeout property of the flows which instructs the switches to delete a flow if the flow has not been matched to an event for the specified period.

This procedure ensures that there is a seamless transition between the two consistent states. Unlike in [8] we cannot inject VLAN tag at the borders switches. Instead, the publishers tag the events with the VLAN Id because otherwise, when DZ version is changed the following two actions would need to take place atomically

- The publishers would need to change their content address calculation technique to use the new partitioning methodology.
- The flows at border switches would then need to inject the new VLAN tag to the events with new DZs. But it is impossible for the border switches to tell whether these events have the new or the old DZ.

It is impossible for both the actions to occur simultaneously and hence, whichever order we choose for the change there could always be events sent with old DZs from publishers which will be routed in new DZ paths or vice versa. Hence there is always a state associating the flows and publishers in PLEROMA when Dimension Optimization is used and in this algorithm, we use this to our advantage to tag the event with VLAN Ids at the publishers.

The obvious drawback with this approach is that the number flows at each of the switches during the transition is the sum of number old flows with the old VLAN Id and number new flows with the new VLAN Id at these switches. Hence it causes a burden on TCAM memory. The following two approaches does not place a such a huge burden on TCAM memory.

5.2 IP Multicast

The idea behind this approach is to use channel based pub/sub like approach as the intermediate state. Here, each partition can be considered to be a channel in channel based pub/sub approach, and they are assigned an IP multicast address each. IP multicast is described in RFC 1112 [32]. It is a many-many to communication standard used to send IP packets to multiple hosts using a single pre-defined destination IP address. According to this standard, there are groups to which hosts can join and each one of the groups is associated with an IP address and packets with this IP as the destination address will be routed to all the members of the group. Senders of the packets need not belong to the group.

A consistent transition from one partitioning technique(old DZs) to another(new DZs) can be achieved through the use of IP multicast as an intermediate state. As stated previously in 3.2.2, the event space in PLEROMA is divided into several non-overlapping partitions. Each of these partitions is associated to a DZ, and a spanning tree. Since the trees are associated to a DZ through the partition they are called prefix trees. In this approach, we assign each of these prefix trees with an IP multicast address and all subscribers of a partition are added as members of the multicast group of that partition. So, any event sent addressed with a multicast IP address will be routed to all the subscribers of that partition. The algorithm described below uses the IP multicast address assigned to a prefix tree to provide a consistent update during DZ change.

1. The prefix tree of each of the partitions is assigned to a multicast IP address. All the subscribers belonging to a partition is a member of the multicast group of the IP address of that partition.
2. When the DZ optimization module of PLEROMA described in 3.3.4 performs a dimension optimization based existing distribution of subscriptions, and events and returns the optimized partitioning of the event space this algorithm is triggered.
3. Using the optimized partitioning method, new Flows with IP address corresponding to the new DZs for existing advertisements, and subscriptions are calculated by creating a new logical partition for the same event space partition in the controller with the same spanning trees as before and re-advertising/re-subscribing using existing advertisements, and subscriptions of the original partition with new DZs. The calculated set of flows with new DZs is not deployed to the switches.
4. Then a *publish using multicast address message* is sent to all the publishers. Once publishers receive this message they will start publishing events using their designated multicast address instead of the content address. If there is a path from a publisher to

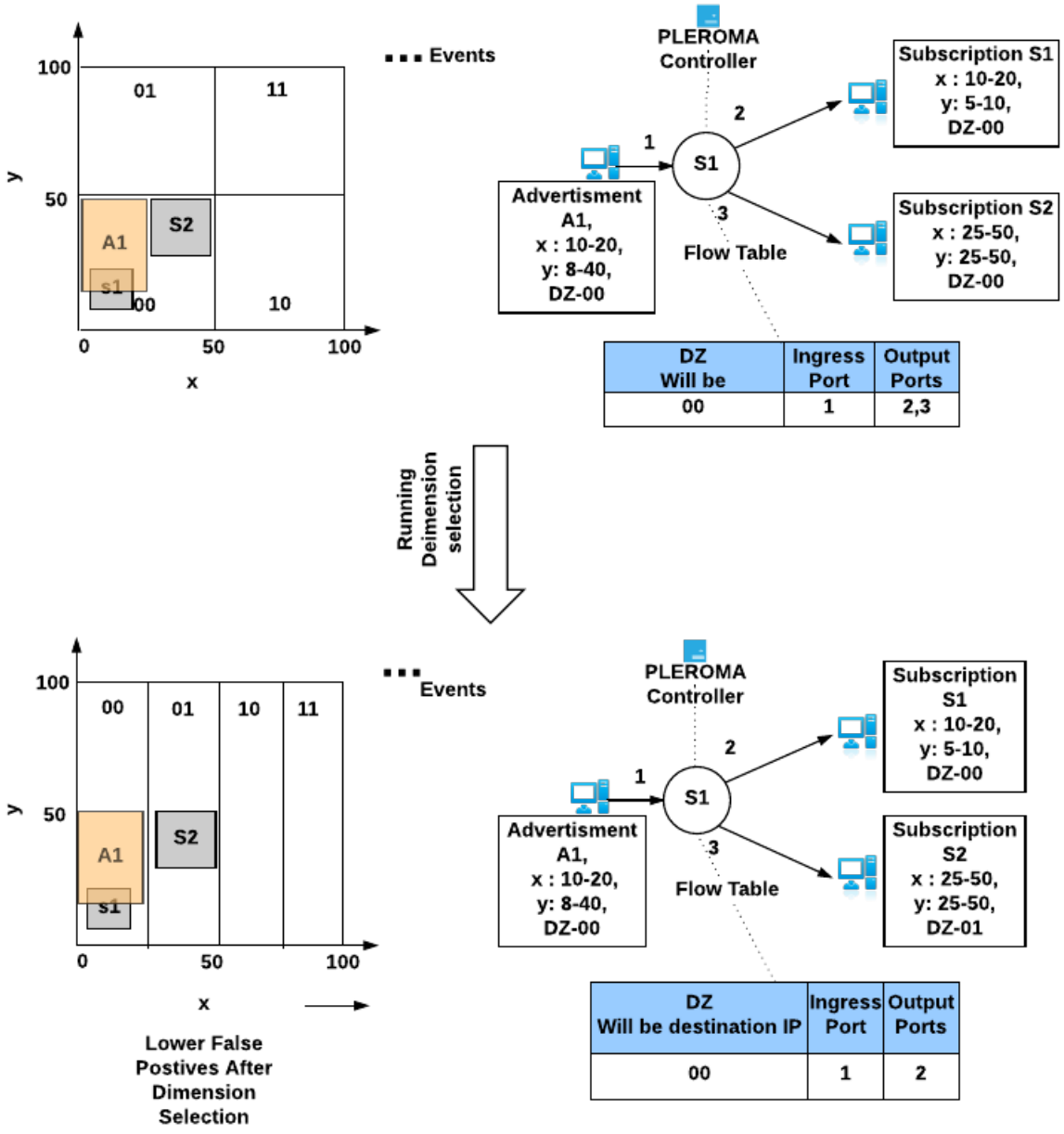

Figure 5.2: Publisher and Subscriber which have true overlap of advertisement and subscription event space will lie in the same spanning tree

a subscriber and if such a path is not caused due to the loss of granularity stemming from the limitation on the length of DZ. Then, such a publisher and subscriber will always belong to the same partition in PLEROMA even if the partitioning technique is changed. For example, consider the figure 5.2 it depicts an event space with 2 bits that can be appended to IP address. The figure shows a subscriber S1 which subscribed to {x[0,10], y[0,10]}, subscriber S2 subscribed to {x[25,50], y[0,10]} and advertiser advertising {x[0,10], y[0,10]}. In partitioning technique shown on the left S1, S2, and A1 all belong to the same partition {00} but with the partitioning technique shown on the right, S1 and A1 belong to {00} partition while S2 belongs {01} partition. This is because S1, and A1 truly overlap with each other while S2 has no overlap but sometimes belongs to the same partition because of the loss of granularity. Hence it is always true that, if there is true overlap between a subscriber and publisher they will belong to the same partition even when the DZs are optimized.

5. Keeping the above observation in mind, we can be sure that using the IP multicast address will ensure that, all the subscribers of the same partitions will receive all the events they are interested in.
6. Next, The old flows are deleted and new flows are added.
7. A *change of DZ version notification* is sent to all the publishers so that they can publish events with the new content address(new DZs) calculated using the new partitioning scheme.

The disadvantage of this approach is a high rate of network false positives caused in the network. There is a higher overhead of maintaining IP multicast groups. But on the otherhand, the number of extra flows in a switch is the sum of number multicast group flows and the number of old flows with old DZ, which is orders of magnitude less than 5.1.1 approach.

5.3 Customized Multicast

The previous approach has an extremely high network false positive rate during transition period because in stable states (old or new DZ states) not all subscribers of a partition are connected to all publishers of that partition. The goal of this approach to reduce the network false positives while also maintaining the advantage of reduced number of flows over the 5.1.1 approach.

The solution described in this approach is based on the following behavior of a PLEROMA. PLEROMA's correctness requires that if a publisher publishes events in which a subscriber is interested in, and the events are not entirely false positives, then there will be a path between such a publisher and subscriber even if the DZs are changed. System correctness also requires that if there is no path between a publisher and subscriber with one partitioning technique and there is a path with another partitioning technique of the same event space, then such

a path contributes entirely to false positives at the subscriber. This is depicted in 5.2. The path to S2 in the figure on the top contributes to false positives entirely.

Keeping the above observations in mind, we propose the following solution 2.

1. This algorithm is triggered when the DZ optimization module of PLEROMA described in 3.3.4 performs a dimension optimization based on existing distribution of subscriptions, and events and returns the optimized partitioning of the event space.

2. The following steps are applied iteratively to all the partitions one at a time.

3. For each switch in the prefix tree of the partition, we fetch all the source ports specified in the flows of that switch and the corresponding set of out-ports for each ingress-ports. We then add one flow for each ingress port with match fields for ingress port and destination IP address as the IP address with a subnet mask, which is calculated from the DZ of the prefix tree. We then deploy these multicast flows in per switch basis and immediately remove all the old flows of that switch.

4. With the above step, all the subscribers will receive all the events they are interested in, but the false positive will be higher than normal but lesser than for 5.2. Because unlike, 5.2 not all the subscribers of the prefix tree receive all the events published into the prefix tree because of how the output ports of the flows are decided in the above step.

5. Next, the controller deploys the new DZ flows, and then the controller sends a *change of DZ version notification* request to all the publishers to use the new DZs when sending events.

6. Once all the publishers have complied, the multicast flows can be removed after a time-out. The order in which the multicast flows are removed impacts the network false positives. There three different algorithms we have tried for this removal. The evaluation section describes this ordering in detail and provides an analysis of which of the approaches provide a better result.

Procedural representation of the above algorithm is described in 2. The disadvantage of this approach when compared to versioning 5.1.1 is the false positive rate. But the number extra flows per switch required by this approach is orders of magnitude less than versioning approach 5.1.1, because if there are N source ports, there are at most N multicast flows extra per switch. This approach is comparable to IP-Multicast5.2 with regards to the number of flows and the false positives rate of this approach is much lower than that of IP Multicast5.2. This approach has better false positive performance than IP-Multicast because unlike IP-Multicast in this approach not all subscribers receive all the events during this multicast period. This is depicted in 6.4.

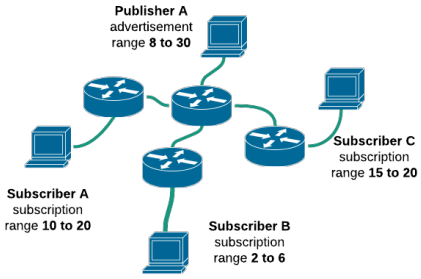

Figure 5.3: Paths formed (in green) on using one IP Multicast Tree. Subscriber B gets message it does not need which results in an increase of false positives

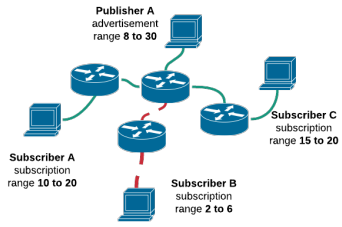

Figure 5.4: Paths formed (in green) using customized multicast algorithm. Subscriber B is not in the path, hence reduces unnecessary traffic

Algorithm 1 Consistent Update of Dimension Optimization Results

0: **procedure** CHANGEDZ(newAdvtHostTuples,newSubHostTuples,mode)
switches ← *getSwitchesWithFlows*()
mulitcastFlows ← CALCULATEMULTICASTFLOWS(*switches*)
newDZFlows ← CALCULATEFLOWSWITHNEWDZ(*newAdvtHostTuples*, *newSubHostTuples*)

newDZFlowsPerSwitch, *multicastPerSwitch* ← FLOWSGROUPBYSWITCHES(*newDZFlows*, *mulitca*

if mode is sourceQunich **then**
sortedSwitches ← SORTSWITCHESBYPUBLISHERDISTANCE(*switches*, *newAdvtHostTuples*)
else if mode is filterRate **then**
sortedSwitches ← *sortSwitchesByFilterRate*(*switches*)
else if mode is transmissionRate **then**
sortedSwitches ← *sortSwitchesByTransmissionRate*(*switches*)
else if mode is random **then**
sortedSwitches ← *shuffel*(*switches*)
end if
DEPLOYMULTICASTFLOWS(*switches*, *multicastPerSwitch*){*Deployingmulticastflows*}
DEPLOYNEWDZFLOWS(*switches*, *newDZFlowsPerSwitch*)
sendChangeDZNotifcationToPublishers(*newAdvtHostTuples.getHosts*())
onAcknowledge :
DELETEMULTICASTFLOWS(*sortedSwitches*, *multicastPerSwitch*)

Algorithm 2 Calculation Of Multicast Flows

```
0: function calculateMulticastFlows(switches)
1: multicastFlows ← []
2: for each switch in switches do
3:    srcPorts ←switch.getSroucePorts()
4:    for each srcPort in srcPorts do
5:       flow.match.srcIp ← 227.0.0.0/24
6:       flow.match.srcPort ← srcPort
7:       flow.action.outPoPorts ←switch.getOutportPorts(srcPort)
8:       multicastFlows.add(flow)
9:    end for
10: end for
11: return multicastFlows
```

Algorithm 3 Calculate Flows With New DZ

```
0: function calculateFlowsWithNewDZ(advtHostTuples,subHostTuples)
1: newFlows ← []
2: for each advtHostTuple in advtHostTuples do
3:    dz, host ← advtHostTuple.dz, advtHostTuple.host
4:    trees ← findResponsibleTrees(advtHostTuple.dz)
5:    for each tree in trees do
6:       newFlows ← newFlows + tree.processAdvt(dz, host)
7:    end for
8: end for
9: for each subHostTuple in subHostTuples do
10:   dz, host ← subHostTuple.dz, subHostTuple.host
11:   trees ← findResponsibleTrees(dz)
12:   for each tree in trees do
13:      newFlows ← newFlows + tree.processSub(dz, host)
14:   end for
15: end for
16: return newFlows
```

Algorithm 4 Group New Flows, Multicast Flows By Switches

```
0: function flowsGroupBySwitches(newFlows,multicastFlows)
1: groupedNewFlows ← [][]
2: groupedMulitcastFlows ← [][]
3: for each newFlow in newFlows do
4:     switch ← newFlow.getSwitch()
5:     groupedNewFlows[switch] ← groupedNewFlows[switch] + newFlow
6: end for
7: for each multicastFlow in multicastFlows do
8:     switch ← multicastFlow.getSwitch()
9:     groupedMulitcastFlows[switch] ← groupedMulitcastFlows[switch] + multicastFlow
10: end for
11: return groupedNewFlows, groupedMulitcastFlows
```

Algorithm 5 Sort Switches Based on Distance From Publishers

```
0: function sortSwitchesByPublisherDistance(switches, advtHosts)
1: sortedSwitches ← []
2: numberOfHops ← 0
3: while size(sortedSwitches) != size(switches) do
4:     tempSwitches ← getSwitchesByDistance(advtHosts, numberOfHops)
5:     unAddedSwitches ← tempSwitches − sortedSwitches
6:     sortedSwitches ← sortedSwitches + unAddedSwitches
7:     numberOfHops ← numberOfHops + 1
8: end while
9: return groupedNewFlows, groupedMulitcastFlows
```

Algorithm 6 Deploy Multicast Flows To Data Plane

```
0: function deployMulticastFlows(switches, multicastFlows)
1: for each switch in switches do
2:     flowsToDelete ← switch.getFlows()
3:     flowsToAdd ← multicastFlows[switch]
4:     switch.addFlows(flowsToAdd)
5:     switch.deleteFlows(flowsToDelete)
6: end for
7: return
```

Algorithm 7 Deploy New DZ Flows To Data Plane

```
0: function deployNewDZFlows(switches, newDZFlows)
   for each switch in switches do
       flowsToAdd ← newDZFlows[switch]
       switch.addFlows(flowsToAdd)
   end for
   return
```

Algorithm 8 Remove Multicast Flows From Data Plane

```
0: function deleteMulticastFlows(switches, multicastFlows)
   for each switch in switches do
      flowsToDelete ← multicastFlows[switch]
      switch.deleteFlows(flowsToDelete)
   end for
   return
```

5.4 Consistent Update Algorithm For Moving Trees

There are scenarios where the spanning trees of a partition needs to be moved to a new tree. This can be because of link failure, high traffic in certain links of the spanning tree, etc. Such a change entails that all the paths between the publishers and subscribers belonging to a partition needs to be moved to the new tree. It becomes impossible to use an ordered update strategy to preserve network invariants for shifting tree when it involves a replicator move 4.4.1 and using versioning 5.1.1 approach can cause a strain of the TCAM memory. So we propose a strategy which utilizes a stateful approach which does not place such a heavy strain on the TCAM memory. This algorithm is described in the following section.

5.4.1 Customized Versioning

This strategy falls under the category of stateful update strategies. It does not require all the non-border switches to do a check on VLAN Id of the events unlike [8]. Consider a multicast path A moving to a new multicast path B. There are three categories of switches during such a transition -

1. Switches which are linked to the same neighbors in the initial and final tree are called overlapping switches.
2. Switches which are linked to some of the same neighboring switches but not all neighbors are the same in both the tree. These switches are called partially overlapping switches.
3. Switches which have no neighbors which are the same in both the trees. These switches are called non-overlapping switches.

Duplicates or loops are caused during a transistion if and only if an event which was routed initially using the flows of one tree is routed to a path which is not a part of the original tree but is a part of the other tree. To prevent the above case, we need to be sure that events would not leave the bounds of a tree which it belongs to. An event could move from the bounds of one tree to another tree only in the case of the second category of switches(partially overlapping switches). Hence, In this algorithm, we need to make sure events are kept within the bounds of the trees it belongs to at the switches. To know which event belongs to which tree we propose to use VLAN Id as a versioning field match field at partially overlapping switches. These VLAN Ids are injected at the border ports of switches when an event enters the network

and the VLAN Ids are removed when the events are sent to subscribers and at the partially overlapping switches, the events are kept within the bounds of one tree or the other based on the VLAN Id of the events. The exact algorithm is as follows

1. Whenever the tree of a partition needs to be moved, a new spanning tree is calculated.
2. All the flows are calculated for the new tree by adding advertisers and subscribers of the old tree to the new tree by re-advertising/re-subscribing in the new tree using the algorithms described in 3.2.3.
3. The overlapping, non-overlapping, and partially overlapping switches between the trees are calculated. The old and new trees are each associated with a unique version Id of their own.
4. All the flows which have an ingress port directly connected to a publisher are edited to have an additional action which tags the incoming events with a VLAN Id corresponding to the version Id of the old tree.
5. At this point, the VLAN Id has no meaning as it is not used for routing/filtering in any of the subsequent flows.
6. A sufficient amount of time is allowed to pass so that all events with no VLAN Id to leave the network.
7. For each new flow of the new tree for partially overlapping switches, we fetch all the source ports specified in the flows of that switch and the corresponding set of out-ports for each ingress-ports. We then compute one flow for each ingress port with match fields for ingress port and destination IP address as the IP address with a subnet mask, which is calculated from the union of DZ of the flows for this ingress port, and a VLAN match field where the match value is version Id of the new tree. The flows are deployed no to the switches. The flows in partially overlapped switches which have ingress ports directly connected to publishers are left out of the step.
8. All the old flows of the old tree that are in the partially overlapping switch are replaced with a multicast flows, calculated for each ingress port which belongs to the old tree like in the previous step. The flows with ingress ports which are connected directly to publishers or subscribers are not taken into consideration for this multicasting. All the old flows which are covered by the multicasted flows are deleted from the switch.
9. The multicasting in the above two steps cause some false positives in the network. But the maximum number extra flows at any point of time in a partially overlapping switch is at most equal to the number ports in the old/new tree for that switch.
10. All the flows for non-overlapping switches are then deployed to the switches. There can be no port in nonoverlapping switches connected to a publisher because all ports of the switches connected to a publisher or subscribers can only be partially or completely overlapping switches. The reason for this is that all the switches directly connected to a publisher/subscriber have at least the link connecting publisher/subscriber to that switch in common in both the spanning trees.

11. The new flows of the new tree for completely overlapping switches is combined with the existing flows of the old tree. This union would not always cause twice the number of flows because some of the existing flow's IP address and output ports already would cover the new flows of the new tree. Such new flows are not added. In some cases, new flows which completely cover the older flows would also exist. Such older flows are immediately removed after adding the new flows. Hence this step does not cause twice the number of flows in TCAM memory like in versioning strategy. But this step causes a slightly increased false positive rate.
12. Next, the flows corresponding to border ports directly connected to publishers in the new tree are added, and they inject new VLAN Id. All the flows which inject old VLAN Id are deleted.
13. After allowing sufficient time for the events with old VLAN Id to leave the network we can remove the flows with old VLAN Id from partially overlapping switches using the idle timeout expiration value of the flows. In the case of completely overlapping switches, first a single flow which covers the IP address of all the flows in that switch for each ingress port which outputs through all the output link in the new tree is added and all existing flows of the switch are removed and new flows of the new tree are added. This is because the complete overlapping switches have a mix flows from old and new trees.
14. Any flow having VLAN match field is edited so that the VLAN match field is removed.
15. All the border flows injecting and removing VLAN Id is divested of that action.
16. In the controller, now, the new tree takes the place of the old tree in the partition in handling all incoming advertisements and subscriptions.

The compromise between complete versioning(reitblatt [8]) and this algorithm is that this algorithm causes a lesser number of flows per switch than versioning while having a slightly higher number of false positives rate.

6 Implementation and Evaluation

For evaluating the performance of consistency algorithms described in the thesis, a working implementation of PLEROMA, workload based indexing, deployment module which implements the consistency algorithms, and a network with OpenFlow-enabled switches to run the evaluations is required. The following sections provide a brief description of the evaluation setup and a discussion of the performance of the proposed algorithms based on the results of the evaluations.

6.1 Overview Of Implementation

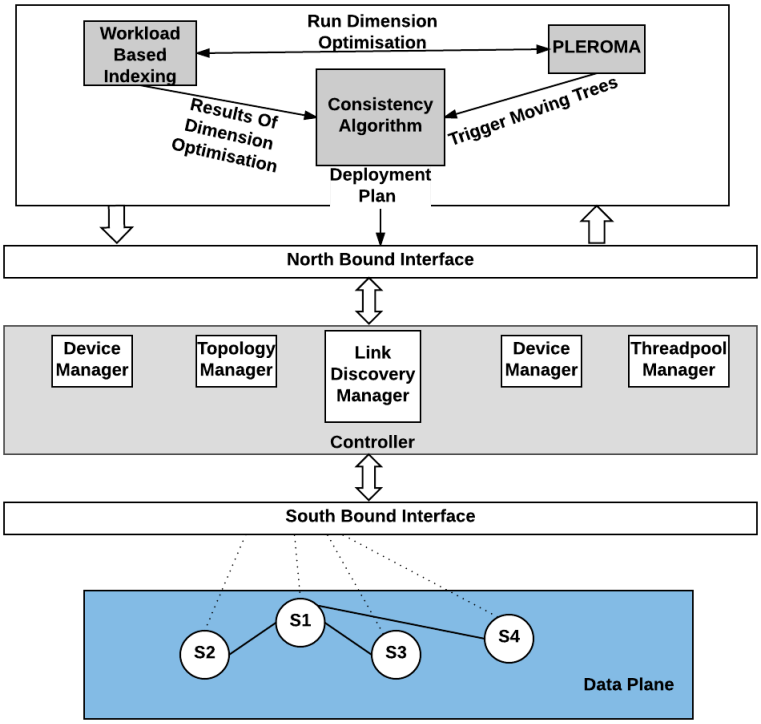

Figure 6.1: PLEROMA Module Structure

Floodlight is an opensource, and OpenFlow based SDN controller implementation. It is implemented in Java. Floodlight allows new modules and new services to be added to the floodlight controller which extends the features provided by the controller. A floodlight module can receive events passed along to the Floodlight controller from the dataplane. It can deploy new flows onto switches using a service called the static flow pusher service. It is also possible to provide new services. For instance, a new service which collects statistics from the network switches has been implemented by us for the purpose of evaluations. PLEROMA and Workload-based Indexing(Dimension Optimizer) is implemented as separate modules in Floodlight. The PLEROMA module is responsible for listening to advertisements, subscriptions, and creating paths between publishers and subscribers. The Dimension Optimizer module is responsible for listening to changes in event/subscription distribution in the dataplane and triggering dimension optimization. Consistent deployment algorithms for changing DZs, and moving to a new spanning tree is implemented as consistent deployment service which uses static flow pusher service for deploying/deleting flows in the network switches and a custom statistic service for collecting information about the switches. Dimension Optimizer and PLEROMA uses the consistent flow pusher service to deploy the changes in a manner which can preserve the network invariants. A simplified overview of PLEROMA on Floodlight controller is depicted in 6.1

6.2 Overview of Evaluation Environment

To validate the efficiency of the algorithms proposed in the previous sections and to compare the various algorithms, an evaluation of the algorithms were carried out on a working PLEROMA implementation, and a network of OpenFlow-enabled switches emulated using Mininet. The following section discusses Mininet and network topology setup used for evaluation.

6.2.1 Mininet

Mininet creates a virtual network emulating a network of switches, and links. The emulated switches can be connected to SDN controller either on the same or remote machine. Mininet provides a python API to create a network with various topologies with differing number hosts, and switches. The APIs are used to set-up certain percentage of hosts as publishers and the remaining as subscribers by starting the publisher/subscriber application at the hosts which listens on a port for commands sent in UDP packets. It is possible to send UDP packets with a string command using mininet python API. This ability is used for controlling a publisher/subscriber by commanding it to send advertisements/subscriptions, send/listen to events, etc. For our evaluations, we use Mininet API for setting up the datapalne with different amount of switches, publishers, and subscribers.[33]

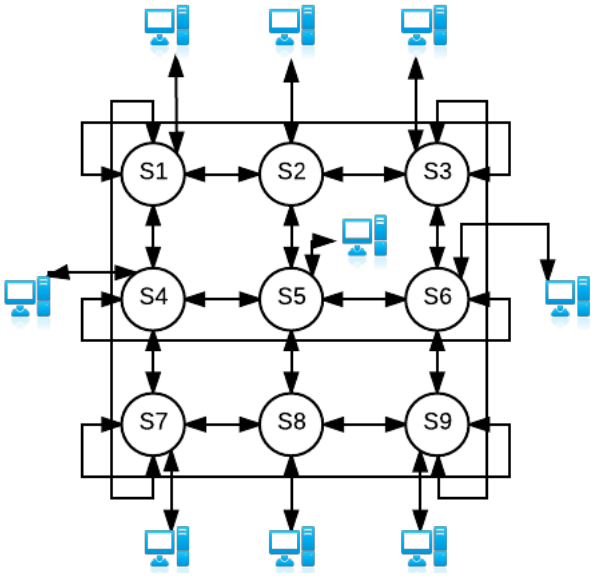

Figure 6.2: A Torus topology with 9 switches, 9 hosts

6.2.2 Network Topology

For the purpose of evaluating loop-freedom during an update, a topology with loops is necessary. While an evaluation of packetdrops or blackholes in a multicast network would require a topology with atleast a single branched path. Hence for all our evaluation we have chosen a 2 dimensional torus interconnect topology. An example of 2-Dimensional torus topology is depicted in 6.2.

6.2.3 Parameters To Evaluate

To compare the efficiency of the various update algorithms, the effect each algorithm has on the system can be used for comparison. In our case, the algorithms have different requirements of flows required per switch and cause different levels of unnecessary traffic in the network. Hence for comparing different algorithms we monitor the follow parameters when the algorithms are executed

1. **Number of flows** - This parameter gives an overview of the number flows in the dataplane when transitioning from one state to another. Lesser the number of flows caused by an algorithm lesser is the burden on TCAM space.

2. **True Positives** - If a subscriber was interested in the event that it received, then such an event is called a true positive. This parameter corresponds to the necessary traffic in the network.

3. **False Positives** - If a subscriber was not interested in an event that it received, then such an event is called a false positive. This parameter corresponds to the unnecessary

traffic in the network in PLEROMA. It is generally caused due to a lesser granularity of the content filters. However, during updates higher false positive rate is caused by the multicasting algorithms. Hence a measure of this parameters gives an inference of the unnecessary traffic caused in the network by different algorithms.

4. **False Negatives** - If a subscriber does not receive an event that it is interested in, such an event is called a false negative. This parameter provides the validity check for the implementation of the proposed algorithm. This parameter should be zero in the network.

6.2.4 Evaluation Execution

For our evaluations, the evaluation platform is setup as follows,

1. The floodlight controller with PLEROMA, workload based indexing module and consistent deployment service are started.

2. The controller collects switch statistics of the number packet transmitted/received during each 30 milliseconds interval.

3. Next, the Mininet script for setting up the network is run. This script starts up the publishers and subscribers. The network can have 8,16,32 or 64 switches and as many hosts. The publisher to subscriber ratio is around 1:1. For example, with a network of 8 switches, there are four publishers and four subscribers.

4. Once the network is set up, the publishers and subscribers are commanded to send advertisements and subscriptions respectively.

5. Once publishers have sent advertisements, they start publishing events and subscribers listen to events after subscribing to a region of the event space. The events sent by publishers have dimensions that are positively correlated with each other.

6. The publishers and subscribers have logs of events sent by publishers and received by subscribers. The false positives received by subscribers are measured for each interval of 3 seconds.

7. To evaluate the dimension optimization and consistent update of the change of DZ, the existing distribution of events is changed to be negatively correlated in some dimensions. Then dimension optimization is performed. The results of optimization are pushed onto the switches using one of the proposed algorithms described in 5.2.

8. The spanning tree of one of the partitions is then forced to change when the traffic in link reaches a certain threshold. The change is executed first by using Reitblatt [8] and next by using the algorithm proposed in 5.4. All the above steps are repeated 3 times.

9. Next, The experiment is repeated using a different consistent DZ change algorithm to make the deployment.

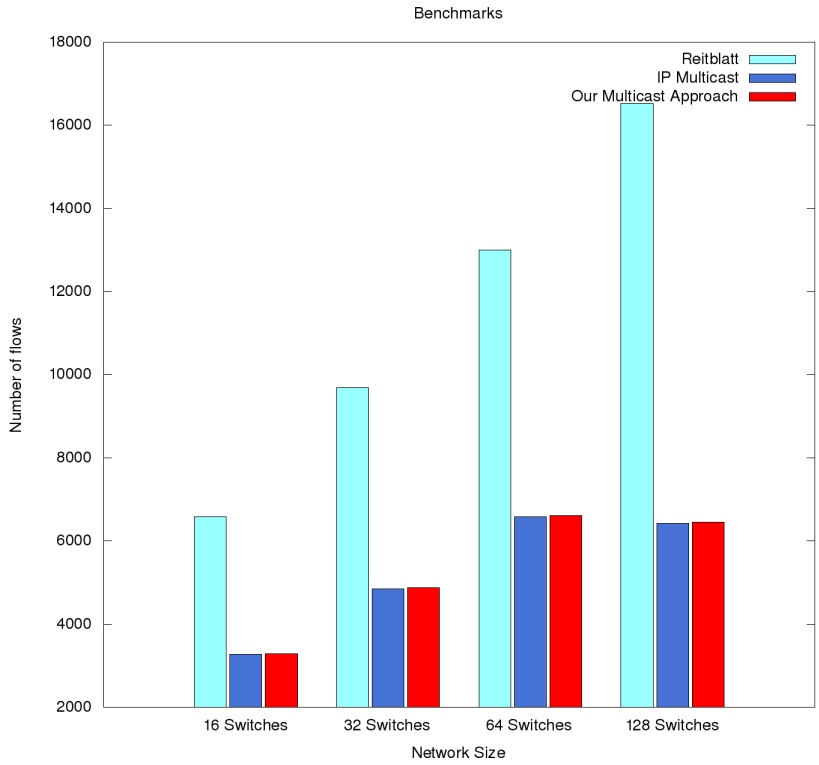

Figure 6.3: Number flows when updating with versioning and multicast approaches

The above experiment is repeated for networks of different size, 8, 16, 32, 64 switches. Once the above steps are carried out, the information of false positives at the subscriber with respect to time, total number events in the entire network with respect to time, and number flows in the network with respect to time is available to us for comparing and contrasting all the algorithms.6.2.3.

6.3 Evaluation Results

6.3.1 Consistent Update Algorithms For Dimension Optimization

The following section compares different *Consistent Update Algorithms For Dimension Optimization* using the results of evaluation.

TCAM requriement - Reitblatt Vs IP-Multicast Vs Our Multicast

6.3 depicts the number of flows in the entire network during the transition from one partitioning method to another after Dimension Optimization. Since the TCAM requirements directly correspond to the number of flows in a switch, lesser the number of flows lesser is the burden

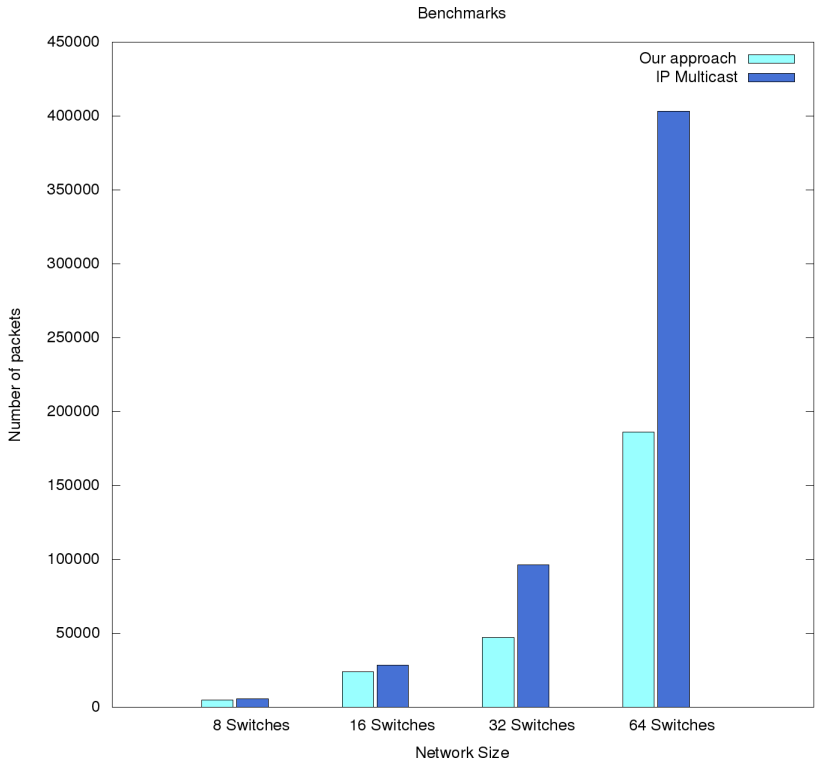

Figure 6.4: IP-multicast vs custom-multicast

on TCAM memory. It is clear that the number flows required during the transition is much higher for versioning (reitblatt)[8] as compared to using the multicast algorithms. However, the performance of both the multicast algorithms is similar in this aspect.

False Positives - IP-Multicast Vs Custom Multicast

6.4 depicts the number of packets in the network during the transition from a change of DZs for networks of different sizes. Since the same event distribution is used for both the multicast algorithms and the events are published at the same rate, the number of packets sent into the network is comparable during the change of DZ using each of the two algorithms separately. Any appreciable difference in the traffic of the network for the different approaches is caused due to the fact that the number false positives introduced by the different algorithms are different. Hence, from 6.4 it clear the performance of topology-aware custom multicast described in 5.3 is much better that using out of the box IP-multicast described in 5.2.

Impact of order of removal of multicast flows in Custom Multicast

One of the final steps of using the custom Mutlicast Algorithm(5.3) for consistently changing from one partitioning technique to another is the removal of intermediate multicast flows from

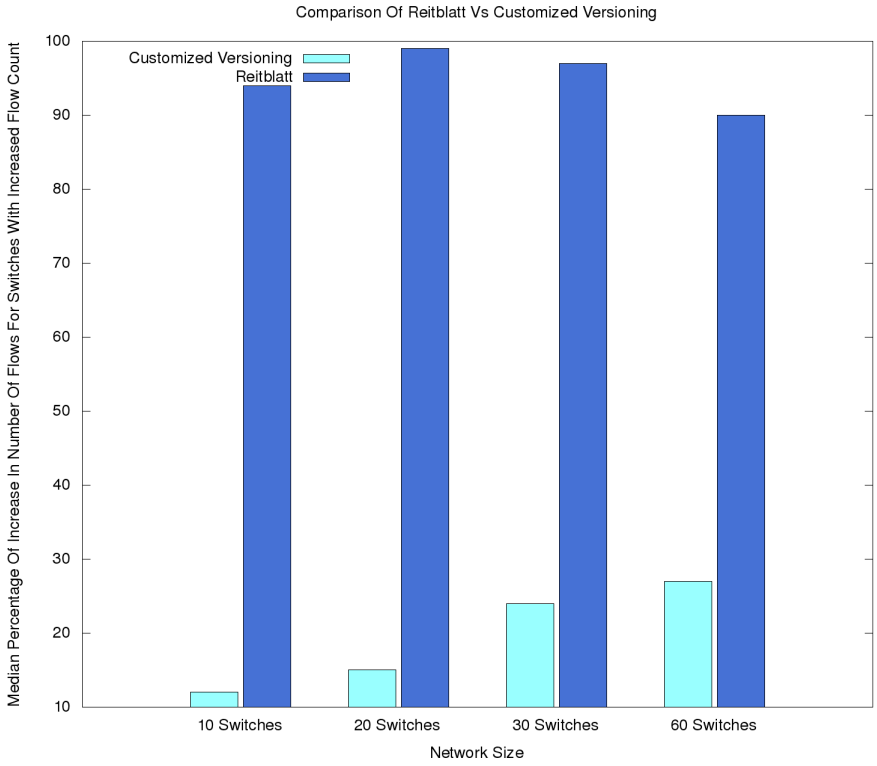

Figure 6.5: Median Increase in the number flows per switch when updating with Reitblatt and Customized Versioning Approach

the switches. Does the order of removal of multicast flows from switches have an impact on unnecessary the traffic in the network? To answer this question we performed *consistent update of dimension optimization* using custom multicast 5.3 but on the final step of the algorithm where multicast flows are deleted from the switch, we ordered the deletion of the flows using based on one one of the following approaches

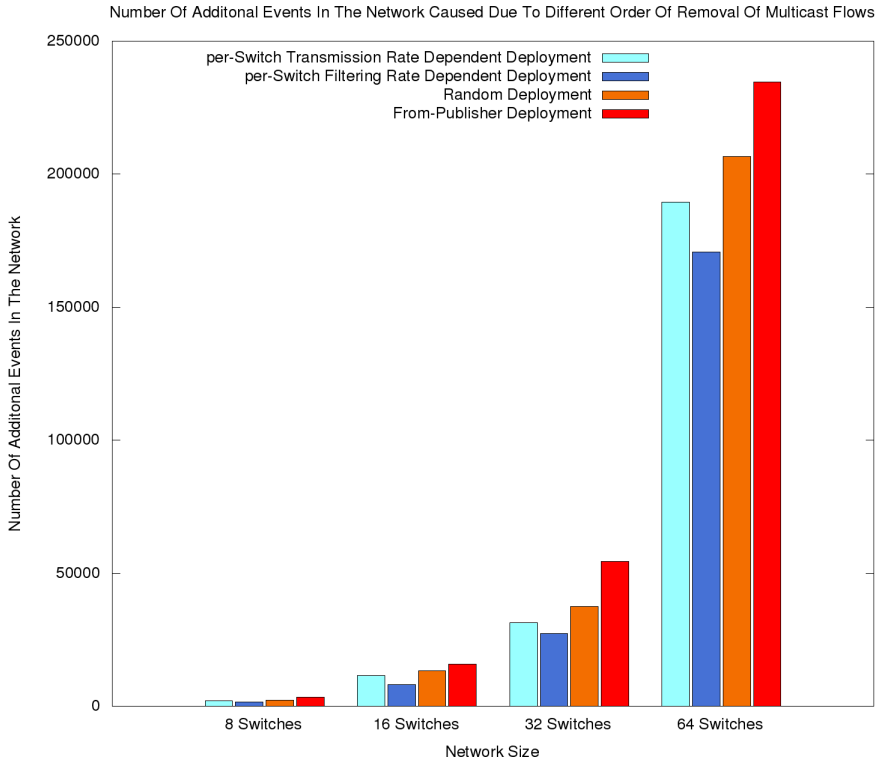

Figure 6.6: Number of Packets in the network with different order of removal of multicast flows

- **Transmission rate of the switches** - From the switch statistics collected at the controller, we sort the switches based on the number of the packets transmitted by the switch in descending order. The multicast flows are removed based on the transmission rate of the switches.
- **Filter rate of the switches** - This rate corresponds to the ratio of the number packets dropped to the number of packets handled by the switch. Higher this value higher, higher is the filtering rate of the switch. Hence during the change of partitioning technique, we use this value to order the deletion of multicast flows from switches with the highest value to the lowest.
- **Source quenching** - Start removing multicast flows from the switches directly connected to publishers(called border source switches) and move to the other switches based on their distance from the border source switches.

- **Random** - Remove multicast flows in random order.

From the figure 6.6, it is clear that transmission and filter rate based approach provides much better performance than a random strategy or source quenched approach because the actual filtering of events takes place rarely at the border switches in the case of networks with large number switches and subscribers.

Subscriber False Positives During Dimension Optimization

The subscriber false positives rate is given by

$$false\_postive\_rate = false\_postives\_count\_per\_10\_ms / total\_events\_per\_10\_ms \tag{6.1}$$

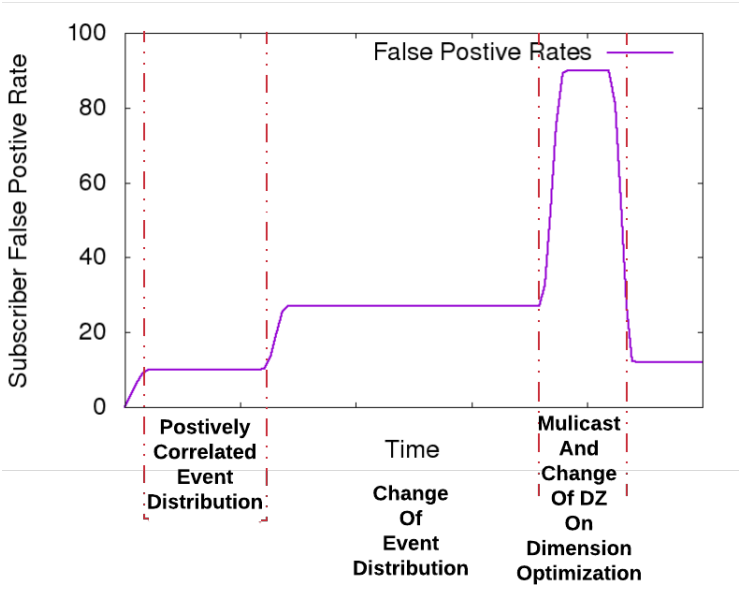

Figure 6.7: Average Subscriber False Positive Rate vs. Time in seconds before, during, and after the consistent update of dimension optimization with custom multicast

We monitored the false positives at the subscriber before, during, and after the consistent update of dimension optimization with custom multicast. The figure depicted in 6.7 shows the increase in false positive rate during the transition period. This spike is due to intermediate multicast flows added during the transition. A similar result is obtained when using IP multicast during the transition as well but the total number of false positive in IP multicast is much higher. Hence the compromise between a multicast approach and versioning approach is between the number of flows and the false positive rate introduced in the network.

6.3.2 Consistent Update Algorithm For Moving Trees

The following section compares two stateful strategies - reitblatt [8] and customized versioning 5.4 when consistently moving the spanning tree of a partition to another spanning tree.

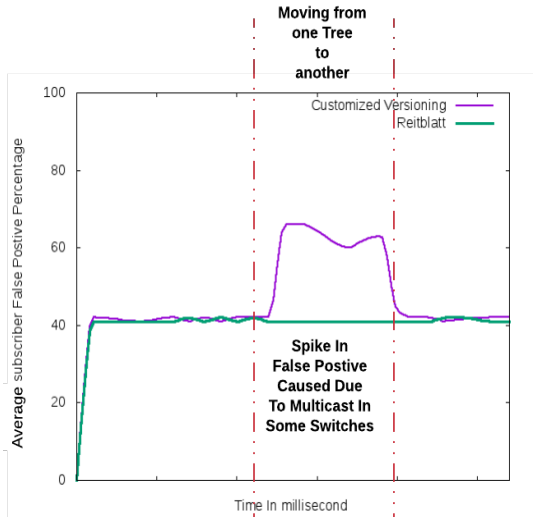

Figure 6.8: Subscriber False Positive percentage when moving spanning tree of a partition

Subscriber False Positives - Customized Versioning vs Reitblatt

The graph 6.8 depicts subscriber false positive percentage when moving the spanning tree of a partition using versioning [8] and customized versioning (5.4). It can be seen that rietblatt causes no new false positives in the network but on the other hand customized versioning causes a slightly higher positive rate during the update than in the consistent state. The reason for the higher false positive rate is the multicasts used in the algorithm. The false positive rate of customized versioning depicted in 6.8 is the worst case false positive rate we encountered during our evaluations. So why should we use customized versioning even if it could have a slightly higher false positive percentage? The following section provides the answer to the above question.

TCAM requirement - Customized Versioning vs Reitblatt

The TCAM requirements directly correspond to the number of flows in a switch. Lesser the number of flows lesser is the burden on TCAM memory. The graph depicted in 6.5 shows that reitblatt places a much higher burden on TCAM memory than customized versioning

(5.4). Hence the comprise between the two algorithms is the number flows per switch and the false positive rate introduced in the network

7 Conclusion and Future Work

Middleware is an application which acts as a communication channel between two/more applications. Such an entity must assure a certain level of Quality to the applications which connects to it. Hence, PLEROMA being a pub/sub middleware needs to ensure a certain quality of services or rather ensure a certain level of consistency to its end-users during its operation. This thesis has proposed and implemented algorithms to maintain one such promised quality of services of PLEROMA during optimization operations.

Specifically the questions answered by this thesis are

- What is the nature of the data-plane consistency problem in PLEROMA - a content-based pub/sub environment build on SDN?
- What are scenarios which require consistency when PLEROMA is dynamically optimized based event traffic, subscription distribution?
- How well would existing "SDN data plane consistent update solutions" fit the for the scenarios specific to PLEROMA?
- On what basis can we compare different solutions and hence, understand the compromises involved?
- Are there other algorithms which better fit the specific scenarios in PLEROMA than generic consistent update solutions?

The chapters on the existing update solutions, algorithms and evaluations have been written with the aim of answering those questions. The chapter on in-network filtering discusses the nature of data-plane consistency in PLEROMA and the problem statement provides a brief description of what the existing solutions are lacking. The proposed multicast algorithms for consistent update of Dimension Optimization , and customized versioning algorithms for shifting trees both make use of the IP-prefix based routing property of PLEROMA to propose a consistent update solution which reduces the strain on TCAM memory and evaluations show that the TCAM memory usage is indeed reduced but with an increase in network and subscriber false positives. The evaluation chapter also provides a basis to compare the different solutions based on the resource which affects the performance, correctness, and efficiency of PLEROMA to understand the factors which could shed light on the compromises involved when choosing a specific solution to provide consistency.

However, there are some limitations to the implemented solutions. It does not provide different levels of consistency for PLEROMA. Also, the proposed algorithm for providing consistency are for the most part context-specific, i.e., the algorithm for different scenarios is different. The proposed solutions also do not take control plane scalability into consideration. Hence a possibility for the extension of this thesis work can include work on how the update consistency can be decoupled from the pub/sub context to sufficiently make sure that PLEROMA scenarios can change independently of consistency algorithms. To achieve a true decoupling between data plane consistency and pub/sub optimization operations, there should be a way for the PLEROMA modules to generically communicate the required level of consistency and scenarios. Another avenue for development is to consider actual resource utilization in the data plane based on traffic and memory utilization into consideration in the algorithms.[5] [33]

An important point to consider before inferring the evaluation is that the results of the evaluations in this thesis have been carried out using Mininet with the basic assumption that there are no link failures between the switches and the controller. The evaluation does not cover the time taken for executing updates because actual latency can have different causes most of which are environment dependent. For instance, controller response times is dependent on the hosting machine and the latency of communication between switches, and controller. Hence, any such evaluation needs to be performed on a real network infrastructure with OpenFlow-enabled switches.

Bibliography

[1] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and evaluation of a wide-area event notification service," in *ACM Trans. Comput. Syst.*, pp. 332–383, Aug 2001.

[2] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, pp. 114–131, June 2003.

[3] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Challenges for distributed event services: Scalability vs. expressiveness," in *Engineering Distributed Objects '99*, pp. 155–166, May 1999.

[4] B. Koldehofe, F. Dürr, M. A. Tariq, and K. 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, (New York, NY, USA), pp. 3:1–3:6, ACM, 2012.

[5] M. A. Tariq, B. Koldehofe, S. Bhowmik, and K. Rothermel, "PLEROMA: a sdn-based high performance publish/subscribe middleware," in *Middleware '14 Proceedings of the 15th International Middleware Conference*, pp. 217–228, December 2009.

[6] M. A. Tariq, B. Koldehofe, S. Bhowmik, and K. Rothermel, "High performance publish/-subscribe middleware in software-defined networks," pp. 3–169, IEEE/ACM Transactions on Networking, December 2016.

[7] S. Bhowmik, M. A. Tariq, J. Grunert, and K. Rothermel, "Bandwidth-efficient content-based routing on software-defined networks," in *10th ACM International Conference on Distributed and Event-based Systems DEBS '16 New York, NY, USA*, p. 137–144, May 2016.

[8] M. Reitblatt, "Abstractions for network update," in *SIGCOMM*, MAY 2012.

[9] R. H. Arpaci-Dusseau, Arpaci-Dusseau, and C. Andrea, *Introduction to Distributed Systems (PDF)*. Arpaci-Dusseau Books, pp, 2014.

[10] A. S. Tanenbaum and M. V. Steen, *Distributed Systems - Principle and Paradigms*. Prentice Hall, 2014.

[11] Wikipedia, "Remote procedure call - wikipedia." `http://en.wikipedia.org/wiki/Remote_procedure_call`, 2013.

[12] T. O. Group, "DCE 1.1: Remote procedure call. technical standard C706." The Open Group, Cambridge, MA, USA, 1997.

[13] P. Eugster, R. Guerraoui, and J. Sventek, *Type-based publish/subscribe.* Technical Report DSC ID:200029, EPFL Lausanne., 2000.

[14] Object Management Group, *CORBA Event Service Specification, version 1.0.* OMG Document formal, 2000.

[15] Apache, "Tibco." `https://docs.tibco.com/?_ga=1.88791323.1035077876.1490629495`, 2013.

[16] M. A. Tariq, B. Koldehofe, G. G. Koch, I. Khan, and K. Rothermel, "Meeting subscriber-defined QoS constraints in publish/subscribe systems," *Concurrency and Computation: Practice and Experience*, vol. 23, pp. 2140–2153, Dec. 2011.

[17] G. Mühl, *Large-Scale Content-Based Publish/Subscribe Systems.* PhD thesis, Darmstadt University of Technology., June 2002.

[18] H. Holbrook and B. Cain, "Source-specific multicast for ip," 2006. RFC 4607.

[19] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander, "LIPSIN: line speed publish/subscribe inter-networking," in *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, (New York, NY, USA), pp. 195–206, ACM, 2009.

[20] Z. Jerzak and C. Fetzer, "Bloom filter based routing for content-based publish/subscribe.," in *In DEBS '08*, 2008.

[21] OpenFlow, "SDN standards." `https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf`, 2015.

[22] Cisco, "Internetworking basic." `https://www.cisco.com/cpress/cc/td/cpress/fund/ith/ith01gb.htm`, 2015.

[23] Wikipedia, "Software-defined networking – wikipedia.." `http://en.wikipedia.org/wiki/Software-defined_networking`, [Accessed : October, 2017].

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

[25] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "SCRIBE: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 20, no. 8, pp. 1489–1499, 2002.

[26] R. Mahajan and R. Wattenhofer, "On consistent updates in software defined networks," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, pp. 1–7, NOVEMBER 2013.

[27] N. P. Katta, J. Rexford, and D. Walker, "Incremental consistent updates." `https://www.cs.princeton.edu/~dpw/papers/incremental-updates-hotnets-2013.pdf`, 2013.

[28] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic scheduling of network updates.," in *In Proceedings of the 2014 ACM Conference on SIGCOMM, SIGCOMM'14*, p. 539–550, 2014.

[29] H. Sarhma, "Workload-enabled content-based routing in software-defined networks," Master's thesis, Universität Stuttgart, Germany, November 2016.

[30] A. Riabov, Z. Liu, J. L. Wolf, and L. Zhang, "Clustering algorithms for content-based publication-subscription systems.," in *In Proc. of the 22nd Int. Conf. on Distributed Computing Systems*, 2002.

[31] T. Kohler, F. Dürr, and K. Rothermel, "Update consistency in software-defined networking based multicast networks.," in *IEEE Conference On Network Function Virtualization and Software Defined Networks*, pp. 177–183, NOV 2015.

[32] IETF, "Ipv4 multicast." `https://www.ietf.org/rfc/rfc1112.txt`, [Accessed : April, 2017].

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

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