Hybrid Content-based Routing Using Network and Application Layer Filtering

Sukanya Bhowmik, Muhammad Adnan Tariq, Lobna Hegazy, Kurt Rothermel
University of Stuttgart
{first name.last name}@ipvs.uni-stuttgart.de

***Abstract*—Over the past few decades, content-based publish/subscribe has been primarily implemented as an overlay network of software brokers. Even though such systems provide the possibility of bandwidth efficient expressive filtering in software, they cannot match up to the performance (in terms of end-to-end latency and throughput) of communication protocols implemented on the network layer. To exploit network layer performance benefits, recently, content-based publish/subscribe was realized using the capabilities of Software-defined Networking (SDN). While SDN allows line-rate forwarding of events by content filters directly installed on switches, it suffers from inherent hardware limitations (w.r.t. flow table size, limited availability of bits in header fields) that adversely impact expressiveness of these filters, resulting in unnecessary network traffic.**

In this paper, we strike a balance between purely application-layer-based and purely network-layer-based publish/subscribe implementations by realizing the first hybrid content-based middleware that enables filtering of events in both layers. Moreover, we provide different selection algorithms with varying degrees of complexity to determine the events to be filtered at each layer such that unnecessary network traffic can be minimized while also considering delay requirements of the middleware. Our hybrid middleware offers full flexibility to configure it according to the performance requirements of the system. We provide a detailed performance evaluation of the proposed selection algorithms to determine their impact on the performance of the designed hybrid middleware which we further compare to state-of-the art solutions.

***Index Terms*—Content-based Routing, Publish/Subscribe, Software-defined Networking, Network Virtualization, Hybrid Routing**

I. Introduction

Content-based publish/subscribe (pub/sub) has evolved as a universal paradigm for bandwidth-efficient interactions between loosely coupled producers (publishers) and consumers (subscribers) of content. Its main advantage is that it provides mechanisms to use disseminated content and subscriber interests to take routing decisions. So, effectively, the paths between publishers and subscribers are determined on the basis of content filters (subscriber interests) installed on content-based routers (brokers) that ensure bandwidth-efficient forwarding to only interested subscribers.

Various middleware implementations (e.g., [11], [18], [32]) of the pub/sub paradigm have efficiently supported content-based routing at overlay networks of software brokers. However, with routing realized at the overlay and filtering performed in software, these broker-based implementations are unable to exploit performance benefits (in terms of latency and throughput) of communication protocols implemented on the network layer. This has, recently, led to the realization of content-based publish/subscribe that exploits the capabilities of the very popular software-defined networking (SDN) technology [24], [31].

With the help of standards like Openflow [13], SDN allows software to flexibly configure the network. By extracting all control logic from hardware switches and hosting it on a logically centralized *controller*, SDN manages to establish a clear separation of the control plane and the data (forwarding) plane. The logically centralized controller has a global view of the network and is capable of both reading as well as modifying the state of the network (data plane). Content-based pub/sub greatly benefits from this as it uses the controller to establish optimal paths between publishers and subscribers by installing forwarding rules (content filters) on the Ternary Content Addressable Memory (TCAM) of SDN-compliant switches. Published events get directly filtered against these forwarding rules on the TCAM, resulting in line-rate forwarding and high throughput rates.

More specifically, the match fields of flows on TCAM are used to express content filters and header-based matching of packets dictates the forwarding of published events to interested subscribers. Note that efficient mapping of content to expressive filters that can be deployed as flows on switches may prove to be very challenging. However, expressiveness is an extremely important factor as the effectiveness of the filters installed on TCAM impacts the amount of unnecessary traffic in the network, directly impacting bandwidth efficiency.

Even though, in many cases, the matching operations performed by TCAM are comparable in expressiveness to filtering operations performed in software, it has some inherent limitations. One of these limitations is the number of flow table entries available for content-based filtering in the TCAM of switches. Considering the criticality of the cost for TCAM in the design of a switch, vendors offer only a limited number of flow table entries, which is currently in the order of thousands to hundreds of thousands flow table entries per switch [15]. Interestingly, the growth of routing table is a common problem faced by most routers and switches today. For example, in August 2014, Microsoft Cloud, Ebay, Lastpass along with some others were hit by outages as a result of full BGP routing tables [1]. The TCAM in affected Cisco routers had a default limit of 512k entries for IPv4 routes which was exceeded, causing a spillover effect. So, with a growing demand in connectivity, the limited TCAM resources must be judiciously utilized. In fact, the number of available flow entries largely depend on the match fields used. For example, in an Openflow-enabled switch such as NEC PF5240, the TCAM size is further reduced to a few thousands for IPv6 traffic. To abide by the limit on number of available flow entries on TCAM, expressiveness of content filters may need to be compromised to allow for limited flows representing multiple aggregated filters.

Along with the switch design limitation, the expressiveness of a content filter is further impeded by the limited number of bits available for filter representation at the match fields (e.g. destination IPv6 address, VLAN tag) of flows. For example, even an IPv6 address can offer only a maximum of 128 bits for filter representation. Additionally, the entire range of these IP addresses have to be shared among many applications, thus reducing the available address range for content-based traffic. Besides, it should be noted that IPv6 is yet to be fully deployed at large. In fact, LIPSIN [22], implementing hardware-enabled forwarding for pub/sub, suffers from up to 10% unnecessary traffic in a moderately small topology despite using 248 bits in packet headers for path encoding.

Where on one hand event forwarding in overlay networks provides possibilities of accurate filtering but suffers in terms of responsiveness to event delivery, on the other hand an SDN-based middleware provides line-rate performance but suffers in terms of bandwidth efficiency. So, while considering these two state-of-the-art implementations independently, we are tempted to ask the question : can we do any better? Is it possible to make these two radically different filtering approaches meet in the middle? And this is where we attempt to combine the benefits of both application layer filtering and network layer filtering in realizing a content-based pub/sub middleware that provides hybrid filtering of events. Therefore, in this paper, we focus on designing an SDN-based pub/sub that not only aims at line-rate performance but also bandwidth efficiency by providing a mechanism to filter events both in software (application layer) and on hardware (network layer). We provide selection mechanisms to determine the layer in which each event gets filtered in order to minimize unnecessary traffic in the network while also considering delay requirements of the middleware. Our hybrid approach offers complete flexibility to control the amount of filtering to be performed at each of the layers where the two extreme cases are pure software filtering and pure hardware filtering, thereby providing a complete degree of freedom to select the performance of the system in terms of latency and bandwidth efficiency. In summary, the contributions of this paper are the design, implementation, and detailed performance evaluation of a hybrid SDN-based pub/sub middleware, the first of its kind, that provides event selection techniques to enable filtering of events both on the application layer as well as on the network layer in a latency and bandwidth efficient manner.

II. Preliminaries and System Architecture

The content-based middleware implemented on SDN consists of primarily two participants, i.e., publishers and subscribers. The publishers specify content they intend to publish (i.e., advertisements) and subscribers specify content they are interested in receiving (i.e., subscriptions) to the controller. On receiving advertisements and subscriptions, the controller, who has a global view of the network, establishes paths between each publisher and its interested subscribers. This is achieved by using the widely accepted Openflow standard to install content filters embedded in the match fields of flows on Openflow-enabled switches in the data plane. The deployed flows enable header-based matching of packets from publishers to subscribers at line-rate [31].

The primary requirement of content-based routing involves an efficient mechanism to represent content in an expressive manner and this may be achieved by following a content-based subscription model where events are represented by attribute-value pairs. So, a subscription/advertisement (content filter) is represented as a conjunction of filters on these attributes. Moreover, to ensure the aforementioned packet-header-based filtering of events on the network layer, an efficient mapping between a content filter and match fields on flows of switches is crucial. There are many candidates (e.g., MAC Address, IP Address, VLAN tags) in the match fields of the flows that may be used as content filters. To this end, we use a special range of IPv6 multicast addresses for our middleware traffic. Since header-based matching of packets involves matching of bit strings in the header fields of a packet with the bit strings that constitute the IP addresses on flows, content has to first and foremost be represented in the form of binary strings. Also, these binary strings need

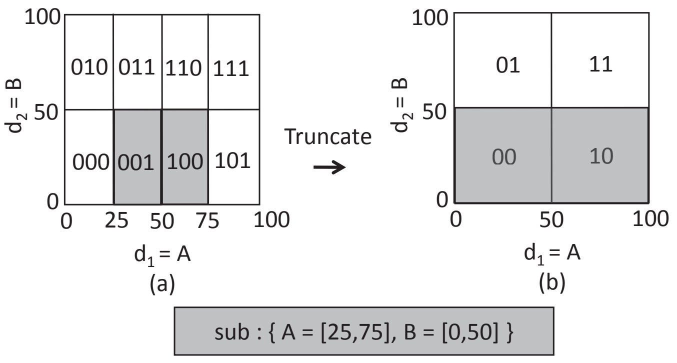


Fig. 1: Spatial Indexing

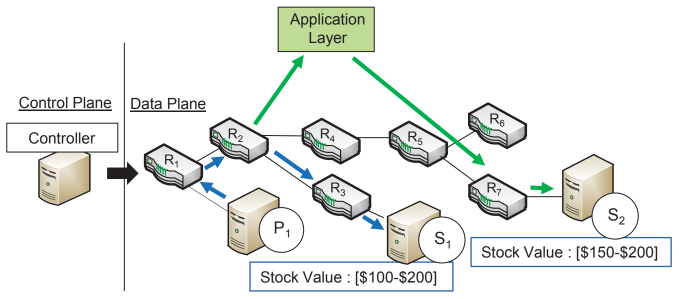


Fig. 2: Hybrid Content-based Routing

to possess certain characteristic properties depending upon the subscription/advertisement/event they represent. So, two identical content filters (subscription/advertisement) should have identical binary representations. Also, containment relation between two subscriptions should be reflected in their respective binary representations. For example, if there are two subscriptions, sub_1 = stock value : [100 – 200] and sub_2 = stock value : [150 – 200], where sub_1 contains ($\succ$) sub_2, then the binary strings representing them should also reflect this relation. This, in turn, ensures matching of events to subscriptions as the binary string representing an event will successfully reflect its containment within all binary strings representing subscriptions interested in it. Mapping content to binary strings can be performed using different approaches, e.g., spatial indexing [32], bloom filters [8].

The concepts discussed in this paper are generic and can be applied to any mapping strategy. However, in our middleware implementation, we use spatial indexing [31] which successfully maps content to binary strings while preserving the aforementioned containment relations. Spatial indexing may generate a set of binary strings (denoted by BS where $bs_k \in BS$) for the same subscription to attain the desired level of expressiveness of content filters which would result in multiple flows for the same subscription. We explain the introduction of unnecessary traffic in the network due to the aforementioned limitations with a simplified example illustrated in Figure 1. Let us assume that a subscription, sub: $\{$A=[25,75], B=[0,50]$\}$, has to be represented as binary string using spatial indexing. As depicted in Figure 1(a), for two attributes A and B, two binary strings $\{001, 100\}$ consisting of 3 bits each are required to accurately represent this subscription. However, let us assume that only 2 bits are available to represent this filter. In such a case, the new filters will be $\{00, 10\}$ and all events matching the entire highlighted area in Figure 1(b) will now be forwarded by these filters, resulting in unnecessary traffic. Also, to reduce TCAM requirements, there may be a restriction on the number of filters (i.e., binary strings) that are allowed to represent a subscription. If only one filter is allowed to represent the subscription in the example in Figure 1, then we can see that only a filter forwarding all event traffic to the subscriber can be used as, in this particular case, two filters must be deployed to even achieve a minimum level of expressiveness. Of course, the above scenario produces a significant amount of unnecessary traffic. It should be noted that, irrespective of the mapping scheme, the length of the binary strings required to accurately represent content (subscriptions/advertisements/events) will increase with the increase in number of attributes in the system, which is a problem considering the limitations on match field length. Also, the number of binary strings needed to represent a single subscription will increase with the increase in number of attributes which can be of crucial importance considering TCAM is a scarce resource. So, the aforementioned limitations will adversely affect the expressiveness of the mapped binary strings and subsequently the amount of unnecessary traffic in the network. In the remaining part of this paper, we refer to unnecessary traffic as false positives. More specifically, in the context of our middleware, false positives on a link or at a subscriber are those events which should have already been filtered out but got forwarded due to the limited expressiveness of the installed content filters on TCAM.

In this paper, we strike a balance between application and network layer filtering while considering both their advantages and disadvantages. Figure 2 illustrates a hybrid approach to content-based filtering where events are filtered both on the network as well as the application layer. We realize the application layer in our middleware as a pub/sub cloud service similar to BlueDove [26]. We perform multi-dimensional subscription space partitioning and distribute them among multiple servers (or matchers) that parallelize event filtering. In Figure 2, the filters for subscriber S_1 are completely installed on the network layer, whereas the filter corresponding to the subscription of S_2 at R_2 sends all events matching this filter to the application layer which enables accurate filtering. Only matched events are then injected back to the network at R_7 and forwarded to S_2, resulting in no false positives for this subscriber and subsequently no false positives along the path from R_2 to S_2 in the network. However, there may be false positives along the path between P_1 and S_1 depending on the expressiveness of the filters for S_1 on the switches. Where on one hand, application layer filtering has a distinct advantage over network layer filtering, in terms of reduced false positives, it loses out in matters of end-to-end latency/delay incurred for the delivery of events. Forwarding of events to S_1 occurs at line-rate, whereas that to S_2 is delayed due to filtering in software. Thus, there is a trade-off between reduction in false positives and end-to-end latency in the network as the improvement in one adversely affects the other.

III. Filter Selection Problem

Due to the aforementioned trade-off between end-to-end latency and bandwidth efficiency, the selection of filters that forward events to the application layer is very crucial. In fact, the main problem that we tackle in our hybrid approach to filtering is the selection of filters in the network layer that forward events to the application layer for more accurate filtering in the attempt to reduce the overall false positives in the network. However, we do so while ensuring that the average end-to-end latency of events in the system stays within the application-specified threshold. More formally, let F be the set of all filters on all switches in the network where $f_i \in F$. Also, let rfp_i be the number of false positives reduced in the system if filter f_i is chosen to send matched events for further filtering in the application layer. Let $\mathbb{S}$ be the set of all subscribers in the system where $S_k \in \mathbb{S}$. Again, let δ_k be the average end-to-end latency at subscriber S_k. Finally, let Δ be the average end-to-end latency threshold to be maintained in the system.

Our objective is to determine the subset $SF \in F$ that forwards events to the application layer such that the combined effect of the filters $\in SF$ results in maximum reduction of false positives in the network while staying within a given average end-to-end latency threshold, i.e.,

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

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

This is an optimization problem. Let there be a total of m filters on n switches constituting the network, where m ranges from 0 to $\sum_{j=1}^{|\mathbb{S}|} |BS|_j$. Then, to arrive at the optimal solution, all combinations of filters, i.e., 2^m possible subsets SF have to be calculated and considered. Also, it should be noted that the value of m can be in the order of hundreds of thousands, making the optimal solution impractical and not scalable in a realistically large network.

The above problem may look seemingly like the Knapsack problem [23] where the value (i.e., benefit) of each item in the Knapsack problem may be compared to the false positives reduced by each filter and the weight (i.e., penalty) of each item may be compared to the increase in average end-to-end latency on selecting a filter. The goal in the Knapsack problem is to maximize the total value of items selected for the knapsack while the total weight of the knapsack remains within a given threshold which is similar to our problem where the goal is to maximize the total false positives reduced by selected filters while the total delay penalty incurred by them remains within a given threshold.

However, there is a major difference between the two problems that sets them apart. If an item gets selected for the knapsack, this selection has no influence on the values and weights of the remaining items to be considered for selection. This is where our optimization problem differs. In our optimization problem, if a filter gets selected, this may influence the false positives reduced by the remaining filters and the increase in average end-to-end delay on selecting each of these filters. Due to this significant difference, approaches for solving the Knapsack problem cannot be directly employed to our problem. As a result, in this paper, we propose two selection algorithms with varying degrees of complexity and benefits, in terms of bandwidth efficiency, to solve the filter selection problem. Till now, we have made the assumption that the false positives reduced and the increase in end-to-end latency on selection of every individual filter is already known. However, the process of determining these values is not straightforward.

So, to arrive at a scalable solution to our optimization problem, we have to tackle three subproblems – (i) detect false positives due to each filter on each link of the network such that rfp_i for each filter f_i, i.e., benefit, can be determined (Section IV-A), (ii) determine the increase in the average end-to-end latency of the system, i.e., penalty, on selecting each filter (Section IV-B), and (iii) with the knowledge of these calculated benefits and penalties for all filters in the network, design efficient filter selection algorithms (Section V).

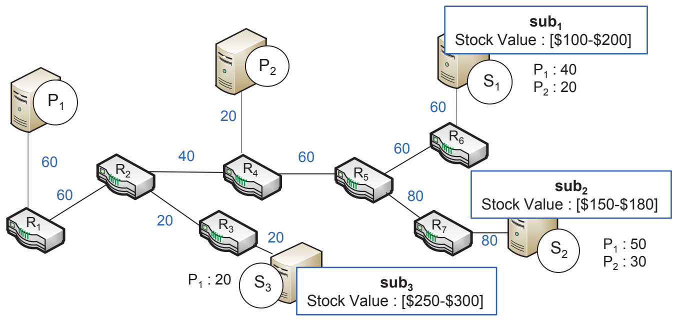
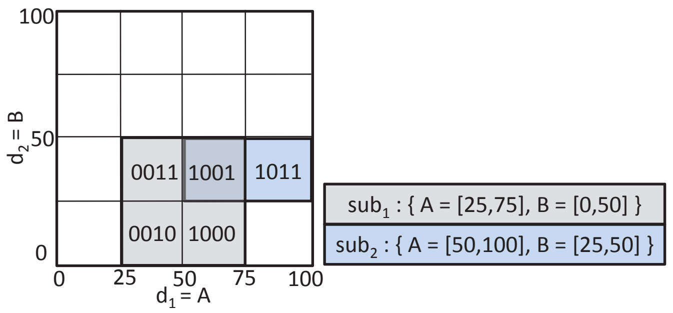


Fig. 3: False Positive Detection



100
50
0
$d_2 = B$
0011
1001
1011
0010
1000
0
25
50
75
100
$d_1 = A$
sub$_1$: { A = [25,75], B = [0,50] }
sub$_2$: { A = [50,100], B = [25,50] }

Fig. 4: Partial Overlap

IV. Filter Benefit and Penalty Calculation

In this section, we provide the means of calculating benefit and penalty associated with the selection of each filter. These metrics form the basis of the filter selection algorithms.

A. *Benefit*

To determine the false positives reduced on selection of each filter, it is imperative to first calculate false positives on each link of the network due to each filter. To do so, each subscriber needs to periodically send the false positives received by it from all its associated publishers to the controller such that the controller can determine false positives along each path between publishers and subscribers. More specifically, the controller calculates false positives on each link of the network for each filter by backtracking on the paths from subscribers to publishers while aggregating false positives on links. However, containment relations between subscriptions, i.e., (i) disjoint, (ii) complete overlap, and (iii) partial overlap, need to be considered during aggregation.

In the scenario where false positives of two subscriptions that are completely disjoint are disseminated over a link, the aggregation over this link will be a sum of the false positives of both subscribers.

However, if one of the subscriptions is contained by the other or both are equal, then a simple sum will account for more than the actual false positive count over the link. In this case, false positives for the broader subscription (or one of the subscriptions in case of equality) should only be considered over the link as all other false positives are either already accounted for in the broader subscription or are events that should be forwarded along this link as they match the broader subscription.

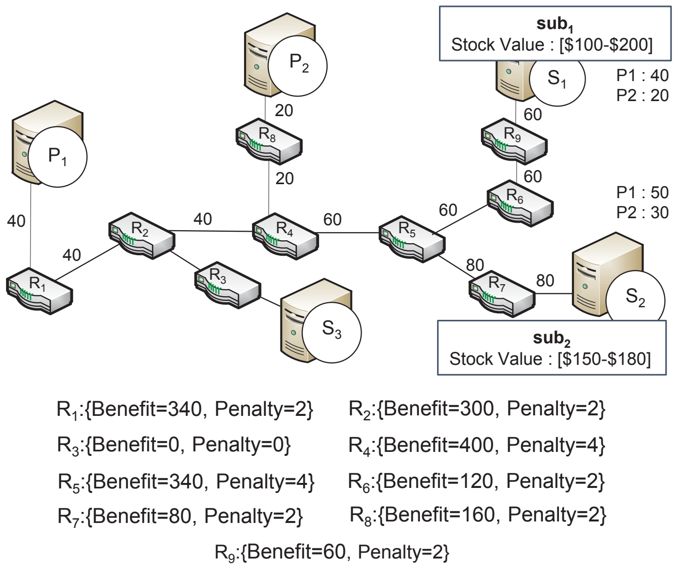
This can be understood with an example in Figure 3 where subscribers S_1 and S_2 subscribe for sub_1 and sub_2 respectively and $sub_1 \succ sub_2$. Let us start the backtracking process from S_1 by aggregating the false positives along the paths to publishers P_1 and P_2. This is straightforward, until we reach switch R_5 as R_5 also forwards false positives to S_2 which might need to be aggregated for the link between R_4 and R_5. However, since $sub_1 \succ sub_2$, only the false positives of the broader subscription sub_1 will be considered for this link.

In the case of a partial overlap between two subscriptions, we mainly identify 3 subspaces, i.e., the overlapping subspace and the 2 disjoint subspaces. Now, if false positives for the overlapping parts and the disjoint parts for each subscription can be identified, then the aforementioned mechanisms can be employed to detect false positives on network links. For example, Figure 4 depicts two subscriptions sub_1 and sub_2 with a partial overlap. Now, if the two subscribers divide the subscriptions into subspaces of finer granularity (as depicted in the figure) and locally detect false positives corresponding to each subspace, then the controller would have to only deal with complete overlaps and disjoint relations. So, while calculating false positives over a link delivering events to sub_1 and sub_2, for the subspace $\{1001\}$ (i.e., overlapping subspace), false positives for only one subscription are counted. Again, the false positives for the other disjoint subspaces can be simply aggregated. Of course, here, there is a trade-off between the accuracy of the false positive count and the granularity at which detection occurs at the subscriber. Finer the granularity, greater is the accuracy as well as the overhead of management at both the subscriber and the controller. Analyzing this trade-off has been the subject of previous research [19] and is not the focus of this paper. Instead, we focus on the challenging issue of performing efficient hybrid content-based filtering. With the knowledge of the number of false positives on each link of the network due to each filter, we can calculate the benefit of a filter, i.e., the false positives reduced by it in the network, by aggregating all false positives forwarded by it along its downstream paths.

B. Penalty

The delay penalty incurred by a filter on its selection primarily deals with the number of paths between publishers and subscribers along which it forwards events. On its selection, a filter, say f_i, will forward events to the application layer increasing the end-to-end latency for these events along all paths that f_i is associated to. This means that while calculating the new average end-to-end latency for the system on selection of f_i, the specific network delays along each path that f_i affects have to be replaced with application delays. Naturally, there is an increase in the average end-to-end latency and this increase is the calculated penalty for f_i.

For the sake of simplicity and without loss of generality, to explain our selection algorithms, we represent delay penalty in terms of the number of affected paths between publishers and subscribers as this number directly affects the average end-to-end latency. Let us assume that the average end-to-end latency of events in the network is N_d, average end-to-end latency of events when the application layer is involved is A_d, and total number of paths between publishers and subscribers in the network is TP. Now, if x is the number of paths involving application layer filtering, then the average end-to-end latency in the network is $[(x * A_d) + ((TP - x) * N_d)]/TP$. Also, the calculation of Δ (i.e., average end-to-end latency threshold) in terms of the maximum number of paths that can be allowed to be affected by application layer filtering delay, AP_{Th}, follows directly from the previous formulation, and can be calculated as $AP_{Th} = (TP * (\Delta - N_d))/(A_d - N_d)$. Note that we calculate penalty and the penalty threshold here w.r.t. average delays and represent them as affected paths just for the sake of better understandability of the following sections. However, in reality, while calculating the penalty, our system can consider the exact network delay incurred along each path between publishers and subscribers and can calculate penalty as the exact increase in average end-to-end latency of the system as described earlier.



sub$_1$
Stock Value : [$100-$200]
P$_2$
S$_1$
P1 : 40
P2 : 20
P$_1$
20
60
R$_8$
R$_9$
20
60
R$_6$
P1 : 50
P2 : 30
40
40
60
60
R$_2$
R$_4$
R$_5$
40
80
80
R$_1$
R$_3$
R$_7$
S$_2$
S$_3$
sub$_2$
Stock Value : [$150-$180]
R$_1$:{Benefit=340, Penalty=2}
R$_2$:{Benefit=300, Penalty=2}
R$_3$:{Benefit=0, Penalty=0}
R$_4$:{Benefit=400, Penalty=4}
R$_5$:{Benefit=340, Penalty=4}
R$_6$:{Benefit=120, Penalty=2}
R$_7$:{Benefit=80, Penalty=2}
R$_8$:{Benefit=160, Penalty=2}
R$_9$:{Benefit=60, Penalty=2}

Fig. 5: Switch Selection

V. Selection Algorithms

After determining the benefits and penalties for each individual filter in the network, we proceed to propose two selection algorithms—the Switch Selection Algorithm and the Cluster-based Selection Algorithm—that differ in time complexity as well as in quality w.r.t. reduction in false positives.

A. Switch Selection Algorithm

We can simplify our problem by selecting switches in place of filters that forward events to the application layer. However, even in this case, we must consider 2^n possible subsets of the entire set of n switches in the network for an optimal solution w.r.t. switches. Can we do something better to reduce this complexity? The main idea behind the Switch Selection Algorithm (SSA) is to iteratively select switches, such that the most beneficial switch gets selected in each iteration, till the given delay threshold is reached and eventually obtain a subset of switches $SR \in \mathbb{R}$ that forward incoming events to the application layer.

SSA starts by considering the set $\mathbb{R}$ consisting of all switches in the network and calculates the benefit and penalty of each switch. The benefit and penalty of each switch is the sum of the benefits and penalties of all filters on it. Next, all switches whose penalty violates the average end-to-end latency (or available path) threshold are removed from $\mathbb{R}$. From the remaining switches, the switch with the highest benefit within the delay penalty threshold is selected and added to the subset SR and removed from $\mathbb{R}$. If the delay penalty threshold has not been reached, then all remaining switches $\in \mathbb{R}$ are again considered for the next cycle. Please recall that the selection of a switch for application layer filtering may change the number of false positives reduced and the additional delay incurred by other switches due to the filters on the selected switch. Assume, a switch R_i having a filter f_i, is added to SR. Now, for another switch $R_j \in \mathbb{R}$ having a filter f_j, the benefit and penalty additionally offered by f_j need to be recalculated. Otherwise, the same false positives already reduced by f_i will again be considered for f_j. Also, the same path already considered for application layer delay may again be counted in the delay penalty for f_j. So, after determining the benefit and penalty for f_i, all false positives for filters corresponding to f_i on subsequent switches along the downstream paths of f_i must be set to zero and the paths marked as already considered. Now, while calculating the benefit and penalty of f_j, none of the false positives and paths already considered for f_i will be reconsidered. So, for each cycle, the benefits and penalties of all filters on the remaining switches in $\mathbb{R}$ are recalculated based on the filters on switches in SR. The cycles continue until the delay penalty is reached or would be potentially exceeded with any further selection or if $\mathbb{R}$ is empty.

To explain the algorithm, we use an example from Figure 5, where the initially calculated benefits and penalties for each switch $\in \mathbb{R}$ are depicted. Let us assume that the average latency threshold Δ when mapped to the affected paths threshold AP_{Th} (cf. Section III) has the value 3. Of all the switches $\in \mathbb{R}$, R_4 and R_5 get removed as they violate the threshold. Also, R_3 can be removed as it has 0 benefit. According to the algorithm, R_1 gets selected as it has maximum benefit within the given threshold. As a result, at the end of this cycle $\mathbb{R}$=$\{R_2, R_6, R_7, R_8, R_9\}$ and SR=$\{R_1\}$. Since, AP_{Th} has not been reached yet, another cycle will commence. Now, since R_1 has already been selected, it will send all events received by it to the application layer, resulting in no or lesser false positives on its downstream paths. As a result, the benefits and penalties of the remaining switches need to be recalculated. In this case, the recalculated benefits and penalties for R_2 are 0 and 0, R_6 are 40 and 1, R_7 are 30 and 1, R_8 are 160 and 2, and R_9 are 20 and 1 respectively. As per the algorithm, in this cycle, R_8 gets removed from further consideration as it violates the threshold, while R_6 gets selected and added to SR. Finally, since the delay threshold of 3 is reached, the algorithm terminates with the final set of selected filters returned as SR=$\{R_1, R_6\}$. The switch selection algorithm has a complexity of O(n^2).

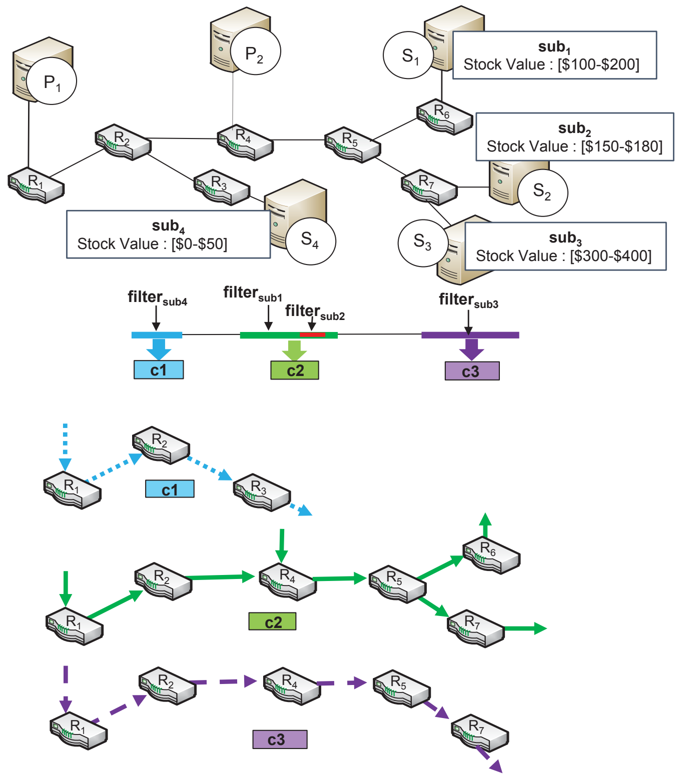


Fig. 6: Cluster-based Selection

B. Cluster-based Selection Algorithm

In SSA, we selected switches instead of filters as a solution considering individual filters is impractical. However, a solution which is in the middle of these two, would be interesting to analyze. As a result, in our next algorithm called Cluster-based Selection Algorithm (CSA), we select filters rather than switches but this time we consider a group of filters based on the subscriptions they represent. More specifically, we first cluster all binary filters (representing subscriptions on the network layer) based on their similarity into spatially disjoint groups. There are many subscription clustering techniques proposed in literature and any one of them may be selected for the clustering of filters [16], [30]. Since the filter clusters (i.e., $\mathbb{C}$) are spatially disjoint, each cluster disseminates a disjoint set of events in the network, thus giving the notion of separate event dissemination trees embedded in the network for each cluster. Therefore, in the following description, we consider each cluster to have its own dissemination tree disjoint from those of other clusters such that an event gets disseminated along only a single cluster's tree and can only affect the false positive count along the links of this tree. For example, Figure 6 illustrates a scenario where 4 subscribers S_1, S_2, S_3, and S_4 subscribe for *sub*$_1$, *sub*$_2$, *sub*$_3$, and *sub*$_4$ respectively where the containment relations between subscriptions and consequently the filters they represent are depicted. Here, we consider a very simple case with 3 clusters, c_1, c_2, and c_3, that are disjoint in space as can be seen in the figure. Also, there are three dissemination trees embedded in the network for each of these clusters.

After clustering of filters, in each cluster $\in \mathbb{C}$, we identify the switch with maximum benefit within the penalty threshold. So, if a switch R_i gets selected in the cluster c_j, we represent this switch-filter_cluster as $R_i\_c_j$. This process of identifying the most beneficial switch within each cluster is identical to calculating the benefits and penalties of each switch in $\mathbb{R}$ and selecting the most beneficial and feasible switch as discussed in details in Section V-A. In this approach, all filters of a switch do not get selected but only a filter set representing a cluster on the switch gets selected for application layer filtering. As a result, we get a total of $|\mathbb{C}|$ switches from all the clusters and add them to a switch-filter_cluster set RC. Let R_1, R_4, and R_1 be selected in c_1, c_2, and c_3 respectively such that $RC = \{R_1\_c_1, R_4\_c_2, R_1\_c_3\}$. Note that even though the same switch R_1 gets selected for two clusters, the switch-filter_cluster makes each pair unique. Now, we try to find the subset $SRC \in RC$ that maximizes the combined reduction of false positives in the network due to all selected switch-filter_cluster pairs $\in SRC$ while ensuring the average end-to-end latency of the system within Δ.

If all combinations of switch-filter_clusters are considered for the solution, then the complexity is $O(2^{|\mathbb{C}|})$. It should be noted that unlike our original optimization problem, selection of a particular switch-filter_cluster pair for forwarding events to the application layer does not affect the reduction in false positive count and the delay penalty of the other switch-filter_cluster pairs as the clusters are disjoint. So, no recalculation of benefit and penalty need to be done at the switches. This problem can now be solved by directly mapping it to the Knapsack problem.

The aforementioned steps produce the subset SRC that maximizes reduction in false positives while staying within Δ. However, if the threshold has not yet been reached, then the entire cycle has to be repeated. In the next cycle, again the benefits and penalties of switches have to be recalculated for clusters that are part of SRC. This is because the selection of a switch from a cluster will affect the benefits and penalties of switches within the same cluster. Based on the new values, again, $|\mathbb{C}|$ switch-filter_clusters are selected from all the clusters and the cycle progresses as before with a new set RC. The cycles continue till the threshold is reached. As a result, CSA has a complexity of O($n^2 + n * 2^{|\mathbb{C}|}$).

C. Network Updates

Once the filters for forwarding events to the application layer are selected, the controller makes the necessary changes to the network by modifying the action field of each flow representing the selected filters. As a result, all events that match these filters get forwarded to the application layer as dictated by the action field of the flows. Clearly, the event distribution and the current subscriptions in the system might change over time degrading the performance of our deployed solution to forward events to the application layer. So, in order to adapt to changes, the controller periodically collects information about the false positives (in the recent time window) from the subscribers, recalculates the most beneficial set of filters, and deploys the changes in the network.

VI. Further Optimizations

The complexity of both the proposed algorithms depend on the number of switches in the network, i.e., n. So, these algorithms may be further optimized if we reduce the search space, i.e., reduce the number of switches on which they operate. In fact, we identify those switches that would add value to the solution and will be candidates for the desired solution while neglecting all other switches. A switch is selected as a candidate if no other switch in the network reduces more false positives than this one for the same set of paths that this switch affects. In doing so, we identify 3 types of switches as candidates for selection—a leaf switch connected directly to a publisher, a switch with two or more ingress ports, a switch connected directly to a switch with two or more egress ports. All other switches in the network may be ignored. The reason why these switches are the only ones that make a difference to the solution is because, due to their ingress ports, they are the starting points of new combinations of paths and therefore will always reduce the most false positives on these path combinations. This can be understood in the following example depicted in Figure 5.

Figure 5 shows false positives on each link of the network when two subscribers S_1 and S_2 subscribe and two publishers P_1 and P_2 publish events. Let us focus on switch R_1 which is directly connected to P_1. If R_1 is selected to forward events for further filtering, the number of false positives it will decrease in the network is 340. Also, selection of R_1 introduces application filtering delay along two paths, i.e., between P_1—S_1 and P_1—S_2. For the same two paths, we need to check if any other switch reduces more false positives. In fact, R_2 reduces 300 false positives while incurring a delay penalty of 2 for the same two paths. As R_1 is the starting point of the path combination consisting of these two paths, it will always reduce more false positives than R_2 for the same penalty. As a result, R_1 gets selected as a candidate while switches like R_2, which are guaranteed to have less benefit for the same penalty, can safely be ignored for further consideration as they will never be a part of the desired solution. A switch like R_4 with two or more ingress ports, however, must be considered as it is a switch where multiple paths join. As a result, for this new combination of paths (P_1—S_1, P_1—S_2, P_2—S_1, and P_2—S_2), this switch will always have the most benefit as it is the starting point of this path combination in the switch network. So, a switch like R_5, with two or more egress ports makes no difference to the solution as its benefit is less than R_4 while it affects the same paths as R_4. Again, a switch like R_6 that is directly connected to a switch with two or more egress ports that splits paths, poses as the source of a new combination of paths, i.e., P_1—S_1, and P_2—S_1, and therefore must be considered for further processing. The example shows how the aforementioned three types of switches should only be considered for further processing without adversely affecting quality of the solution. Pruning the network has a complexity of O(n) and can reduce the runtime of SSA and CSA without degrading their quality. Of course, the effectiveness of this optimization depends largely on the paths between publishers and subscribers.

VII. Performance Evaluations

This section is dedicated to an analysis of the proposed hybrid pub/sub middleware and its comparison to purely network-based and purely application-based implementations. In fact, we compare the hybrid middleware with two network-based implementations. The first implements pure in-network filtering of events while its variant implements R-Tree-based clustering along with in-network filtering. A series of experiments are conducted to understand the effects of the design on performance metrics such as end-to-end latency for event dissemination and bandwidth efficiency in terms of false positives disseminated in the network. We further compare the performances of the two proposed selection approaches, SSA and CSA, in terms of benefit and complexity.

A. Experimental setup

The following experiments have been evaluated under two test environments—1) an SDN-testbed (*SDN-t*) comprising a hardware whitebox Openflow-enabled switch from Edge-Core and commodity PC hardware, and 2) an emulated network running on a single machine using Mininet (*SDN-m*). The latency-related experiments were conducted on *SDN-t* where we created a hierarchical fat-tree topology consisting of 10 switches and 8 end-hosts. The 10 switches are hosted on the hardware whitebox switch from Edge-Core running the network operating system PicOS (version 2.6) [2], [3]. The 8 end-hosts are hosted on commodity rack PCs and perform the role of publishers and subscribers. The SDN controller and application layer reside on a 3.10 GHz machine with 40 cores. All end-hosts are synchronized using the IEEE 1588 Precision Time Protocol (PTP). We used a separate network infrastructure for PTP traffic using a second NIC on each host dedicated to PTP synchronization to counter the possibility of inaccuracies in clock-synchronization.

Besides the testbed, experiments have been conducted on *SDN-m* consisting of a prominent tool for emulating software-defined networks, namely, Mininet [25]. Based on the concept of OS-level lightweight virtualization for network emulation, Mininet enables users to experiment with various topologies and application traffic. We use Mininet to experiment with up to *337* switches and *729* end-hosts on different topologies.

We use a content-based schema that contains up to *6* attributes, where the domain of each attribute varies in the range *[0,1023]*. We use both real-world workload as well as synthetic workload to conduct our experiments. For synthetic data we use two different models for the distributions of subscriptions and events. The uniform model generates subscriptions and events independent of each other. Meanwhile, the interest popularity model chooses up to *8* hotspot regions around which subscriptions/events are generated using the widely used zipfian distribution. For real-world workload, we use data in the form of stock quotes procured from Yahoo! Finance containing a stocks daily closing prices [12]. Such real world data further highlights the performance and importance of the hybrid approach under realistic scenarios.

B. Comparing with State-of-the-Art

The first set of experiments, compares the performances of the hybrid middleware (*HYB-M*), a purely network-based middleware (PLEROMA), and a purely application-based middleware (*APP-M*). PLEROMA [31], as mentioned earlier, implements SDN-based in-network filtering. Moreover, we implemented the purely application-based middleware as a parallelized matching pub/sub service. We divided the event-space into 16 partitions and assigned them to 16 matchers running on 16 cores to enable one-hop forwarding of events as performed in Bluedove [26]. All measurements in the application layer have been performed corresponding to this configuration. We used multiple event rates and datasets to benchmark the performance of the implemented application layer. Please recall that the performance of the hybrid middleware can be regulated by adjusting the value of Δ. In the following experiments we represent this threshold value in terms of a factor of the application layer filtering delay, such that a factor of 0 implies pure network filtering and a factor of 1 implies pure application layer filtering. Also, HYB-M uses SSA for switch selection such that we can compare the performance of a pessimistic hybrid approach with state-of-the-art solutions rather than CSA which outperforms SSA w.r.t. reduction in false positives as can be seen later in this section.

Figure 7(a) depicts the performance of HYB-M and PLEROMA w.r.t. total false positives in the network, i.e., the sum of all false positives on all links, with increasing number of subscriptions when 10,000 events are disseminated. Since APP-M performs accurate filtering of events in software, we do not plot its performance in this graph. For a threshold factor of 0.6, the figure shows that the false positives for HYB-M are much less in every case than those for pure in-network filtering. Even though the hybrid middleware performs better as compared with PLEROMA in terms of bandwidth efficiency, it comes with a price. Figure 7(b) depicts the plots for average end-to-end latency with increasing subscriptions for all 3 systems. The figure shows that pure network-layer filtering has minimum latency in the order of a few microseconds. Also, the increase in number of subscriptions has no influence on latency. On the other hand, APP-M has the worst performance with latency in the order of milliseconds which increases with increasing number of subscriptions because, in software, more the number of subscriptions, more will be the time needed to match events. The figure shows that hybrid filtering results in latency less than that of APP-M but greater than that of

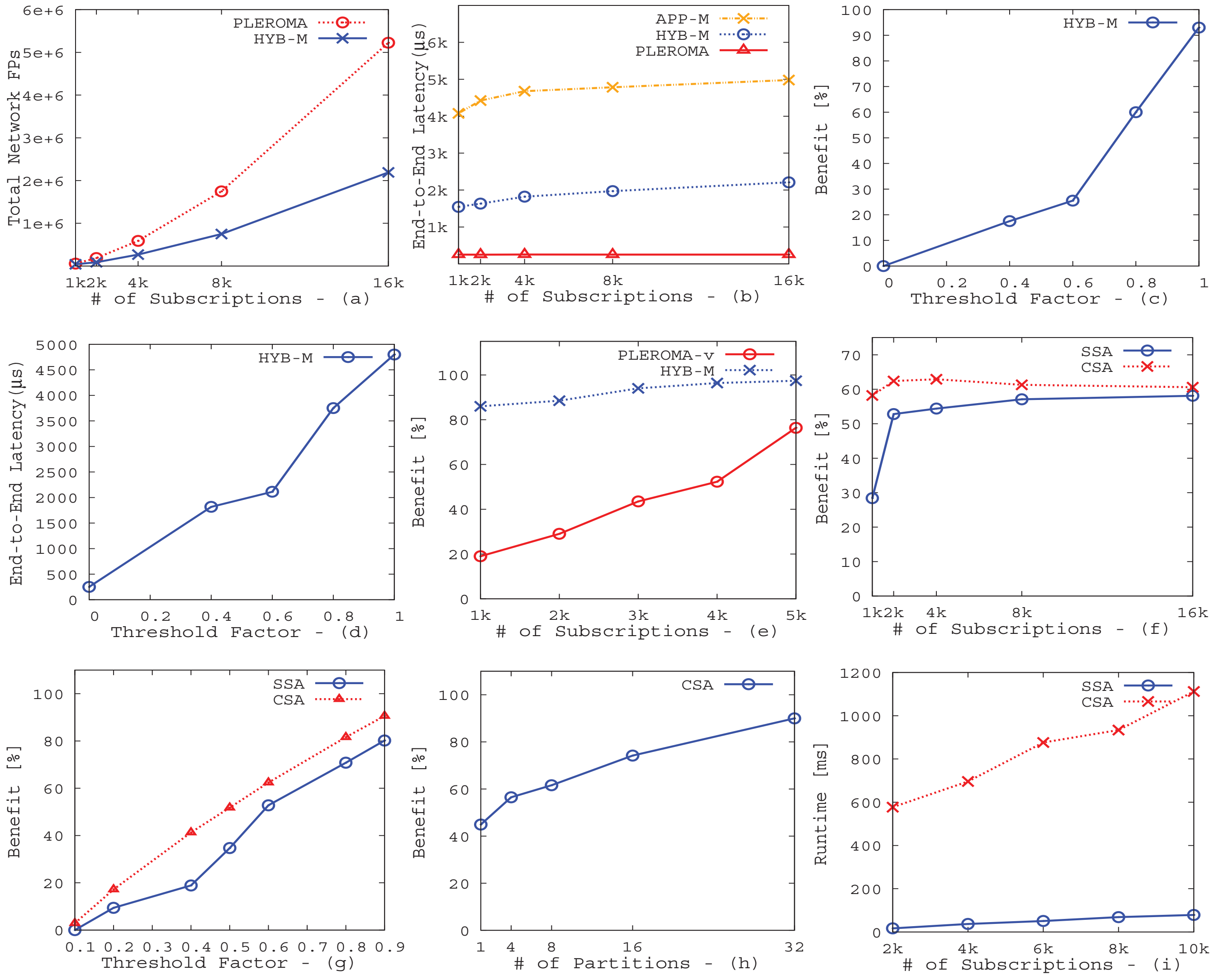


Fig. 7: Performance Evaluations

PLEROMA as a certain percentage of events are now affected by application layer filtering delay. However, both bandwidth efficiency and latency of the hybrid approach may be regulated by adjusting the threshold factor, i.e., Δ, which is clearly visible in the following set of experiments.

Figure 7(c) and Figure 7(d) show the effects on bandwidth efficiency and latency in a system with 8000 subscriptions when the threshold factor is increased from 0 to 1. In Figure 7(c), the term benefit signifies the % of false positives reduced by HYB-M w.r.t. the false positives occurring in PLEROMA. With the threshold factor set to 0, HYB-M has no benefit as it is comparable to pure in-network filtering. With increasing threshold factor, the benefit gradually increases. However, this also implies increase in average end-to-end latency as depicted in Figure 7(d). It should be noted that a factor of 1 is comparable to APP-M w.r.t. latency. However, even in this case the benefit will not be 100% as the false positives on the links between the publishers and the switches they are directly connected to will remain in the system.

In literature, we often encounter the technique of clustering of subscriptions to enhance bandwidth efficiency of pub/sub [12], [33]. As a result, we compare bandwidth efficiency of our hybrid system to a variant of PLEROMA called PLEROMA-v that implements R-tree-based clustering along with in-network filtering. More specifically, PLEROMA-v, first, clusters all subscriptions in the system using the very popular R-Tree [16] clustering approach into multiple clusters. Each cluster has a Minimum Bounding Rectangle (MBR) that represents its ranges along each attribute and consequently the subspace it covers in the entire event space. If we assume that the total number of bits available for filter representation is 100, then, in PLEROMA, spatial indexing will be employed on the entire range for each attribute and the resulting binary strings will be truncated to 100. However, in PLEROMA-v, for each cluster, spatial indexing will be employed only within the ranges of the MBR. So, if there are 8 clusters in the system, the first 3 bits of the filter will represent the cluster id and the remaining 97 bits will be available for spatial indexing within the cluster subspace. Clustering is popular at overlays but in this paper we evaluate PLEROMA-v implemented on SDN. So, our next set of experiments compares the performance of PLEROMA-v with 6 clusters with that of HYB-M when Zipfian distribution is used to generate subscriptions and events with 5 dimensions around 8 hotspots. Figure 7(e) shows that even with the threshold factor set to just 0.4, HYB-M has a higher benefit than PLEROMA-v with increasing number of subscriptions. This is because, no matter how good the clustering technique, PLEROMA-v will always be limited to a specific number of bits, whereas this limitation can be mitigated in HYB-M by involving filtering in software. Our experiments also showed that when subscriptions were generated using uniform distribution, PLEROMA-v performed worse than PLEROMA. In this case, the MBRs of each cluster covered almost the entire range for each attribute making spatial indexing in PLEROMA-v comparable to that in PLEROMA. Moreover, PLEROMA-v lost additional bits to represent the cluster ids. Note that PLEROMA-v will have an advantage over HYB-M w.r.t. end-to-end latency as it performs in-network filtering.

C. *SSA vs CSA*

The next set of experiments evaluates and compares the performance, in terms of benefit and complexity, of the two proposed selection algorithms – SSA and CSA. Figure 7(f) depicts the benefit of SSA and CSA with increasing number of subscriptions. For these experiments, we used 16 clusters for CSA. The figure shows that, in each case, CSA has higher benefit than SSA. This is because CSA has higher flexibility w.r.t. selection of filters as it chooses groups of filters within a switch rather than all filters on it as is the case in SSA. We also conducted experiments to see the behavior of both methods when the threshold factor is gradually increased in a system with 2000 subscriptions. Figure 7(g) shows that with increasing threshold, the benefit increases in both approaches and CSA performs consistently better than SSA. Please note that the performance of CSA largely depends on the number of clusters used by it and as a result our next set of experiments is conducted to analyze the effect of increasing clusters on CSA. Figure 7(h) clearly depicts that, with increasing number of clusters, the performance of CSA improves further as its flexibility of filter selection increases manifold. Of course, when a single cluster is used, CSA is essentially reduced to SSA. We conducted the above experiments using real-world workload which clearly highlights the bandwidth efficiency achievable by the hybrid middleware under realistic scenarios even for a threshold factor of just 0.4.

Even though CSA offers higher benefit than SSA, it loses out to SSA in terms of time complexity. We conducted experiments to compare the runtime of both algorithms. Figure 7(i) shows that with increasing number of subscriptions, the runtime of both approaches increases. This is because, higher the number of subscribers in the system, higher will be the number of paths and filters on switches to be considered, thus increasing the runtime. Also, the runtime of CSA is consistently higher than that of SSA, as in each iteration of CSA, not only does the most beneficial switch get determined in each partition, but also all combinations of switch-filter_clusters are considered to achieve a high quality solution. So, while choosing between SSA and CSA, one needs to consider the trade-off between quality and complexity.

VIII. Related Work

Content-based pub/sub is a much researched area enriched by various contributions to literature [4], [9], [10], [14], [18], [20], [21], [27]–[29], [33], [34]. However, even though these systems can be highly expressive, w.r.t. content representation, most of them are implemented on an overlay network of software brokers and are unable to provide performance, in terms of throughput and end-to-end latency, similar to network layer implementations of communication protocols.

The importance of a scalable and elastic pub/sub providing high throughput and low end-to-end latency has always been impressed upon. In recent times, significant contributions in this respect have been made possible with the emergence of cloud computing which has driven the idea of realizing pub/sub middleware as a cloud service. Li et al. present BlueDove [26], an attribute-based pub/sub service, that targets parallelism of the event filtering process by organizing multiple servers into a scalable overlay as candidates for one-hop forwarding of events. Based on a multi-dimensional subscription space partitioning technique for the distribution of subscriptions between servers, BlueDove exploits skewness in data distribution for performance-aware event filtering at the least loaded servers. In fact, we use similar techniques to implement our application layer. Similarly, Barazzutti et al. also focus on parallelizing the event filtering process by designing StreamHub [5], a scalable pub/sub service based on a tiered architecture. StreamHub comprises a set of independent operators that take advantage of multiple cores on multiple servers to perform pub/sub operations which include subscription partitioning, event filtering, and event dispatching. Scalability of StreamHub is further supported with elasticity in e-StreamHub [6], which is capable of scaling in and scaling out depending on load observations of the system to improve system throughput. Although the performance of these services is ahead of traditional broker-based overlay implementations, they are curbed by the limitations of filtering in software.

In the recent past, networking technologies, such as SDN and NetFPGA, have attracted much attention of the research community, resulting in efforts towards realizing a pub/sub middleware that performs event filtering and routing within the network. LIPSIN [22] proposed an efficient multicast strategy to route events on the network layer using bloom filters. LIPSIN performs routing similar to source-routing where the set of links to be traversed by the packet is encoded in the packet header. This encoding is restricted by the available bits in the packet header, resulting in the generation of fixed length Bloom filters. The use of fixed length Bloom filters to encode links implies the presence of false positives in the network impeding bandwidth efficiency of the system. Likewise, middleware such as DDSFlex [17] and PLEROMA [31] exploit the capabilities of SDN to realize routing and filtering of events on the network layer. However, even though, Bhowmik et al. in [7] address the concerns of the control plane in an SDN-based publish/subscribe middleware, the concerns of the data plane w.r.t. the inherent limitations of hardware switches remain. These data plane limitations result in false positives in the network further highlighting the cost that network layer implementations need to bear in order to achieve line-rate performance. Thus, the need to combine filtering at both layers to strike a balance between their performances is quite apparent.

IX. Conclusion

In this paper, we propose, implement and thoroughly evaluate the performance of a hybrid content-based pub/sub middleware. To the best of our knowledge, we are the first to combine filtering of events in application and network layers in the context of content-based pub/sub. We provide algorithms with various associated complexities and benefits to determine the layer in which each event gets filtered such that the overall false positives in the system can be minimized while staying within an average end-to-end latency threshold. The evaluation results show that our hybrid middleware can be configured by an application to various settings ranging from pure network layer filtering to pure application layer filtering by adjusting the average end-to-end latency threshold in order to achieve desired performance.

References

[1] 512K-Day. http://tech.firstpost.com/news-analysis/internet-outages-possible-next-week-routers-hit-512k-limit-229087.html.

[2] Hardware Switch Edge-Core AS5712-54X. http://www.edge-core.com/.

[3] PicOS Version 2.6. http://www.pica8.com/documents/pica8-datasheet-picos.pdf.

[4] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *Proc. of the 19th IEEE Int. Conf. on Distributed Computing Systems*, 1999.

[5] R. Barazzutti, P. Felber, C. Fetzer, E. Onica, J.-F. Pineau, M. Pasin, E. Rivière, and S. Weigert. Streamhub: A massively parallel architecture for high-performance content-based publish/subscribe. In *Proc. of the 7th ACM Int. Conf. on Distributed Event-based Systems*, 2013.

[6] R. Barazzutti, T. Heinze, A. Martin, E. Onica, P. Felber, C. Fetzer, Z. Jerzak, M. Pasin, and E. Rivière. Elastic scaling of a high-throughput content-based publish/subscribe engine. In *Proc. of 34th IEEE Int. Conf. on Distributed Computing Systems*, 2014.

[7] S. Bhowmik, M. A. Tariq, B. Koldehofe, A. Kutzleb, and K. Rothermel. Distributed control plane for software-defined networks: A case study using event-based middleware. In *Proc. of the 9th ACM Int. Conf. on Distributed Event-Based Systems*, DEBS '15, 2015.

[8] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 1970.

[9] C. Cañas, E. Pacheco, B. Kemme, J. Kienzle, and H.-A. Jacobsen. Graps: A graph publish/subscribe middleware. In *Proc. of the 16th Annual Middleware Conference*, 2015.

[10] F. Cao and J. P. Singh. Efficient event routing in content-based publish-subscribe service networks. In *Proc. of 23rd IEEE INFOCOM*, 2004.

[11] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[12] A. K. Y. Cheung and H. Jacobsen. Green resource allocation algorithms for publish/subscribe systems. In *2011 International Conference on Distributed Computing Systems, ICDCS*, 2011.

[13] O. M. E. Committee. *Software-defined Networking: The New Norm for Networks*. Open Networking Foundation, 2012.

[14] G. Cugola, D. Frey, A. L. Murphy, and G. P. Picco. Minimizing the reconfiguration overhead in content-based publish-subscribe. In *Proc. of ACM Symp. on Applied Computing (SAC)*, 2004.

[15] F. Dürr and T. Kohler. Comparing the Forwarding Latency of OpenFlow Hardware and Software Switches. Technical report, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, 2014.

[16] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. of the 1984 ACM SIGMOD Int. Conf. on Management of Data*, SIGMOD '84, 1984.

[17] A. Hakiri, P. Berthou, P. P. Patil, and A. Gokhale. Towards a publish/subscribe-based open policy framework for proactive overlay software defined networking. (ISIS-15-115), 2015.

[18] H.-A. Jacobsen, A. K. Y. Cheung, G. Li, B. Maniymaran, V. Muthusamy, and R. S. Kazemzadeh. The PADRES publish/subscribe system. In *Principles and Applications of Distributed Event-Based Systems*. 2010.

[19] H. Jafarpour, S. Mehrotra, N. Venkatasubramanian, and M. Montanari. MICS: An Efficient Content Space Representation Model for Publish/Subscribe Systems. In *Proc. of the 3rd ACM Int. Conf. on Distributed Event-Based Systems*, 2009.

[20] K. R. Jayaram, C. Jayalath, and P. Eugster. Parametric subscriptions for content-based publish/subscribe networks. In *Proc. of 11th Int. Conf. on Middleware*, 2010.

[21] S. Ji, C. Ye, J. Wei, and H.-A. Jacobsen. Merc: Match at edge and route intra–cluster for content-based publish/subscribe systems. In *Proc. of the 16th Annual Middleware Conference*, 2015.

[22] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander. LIPSIN: line speed publish/subscribe inter-networking. *ACM SIGCOMM Computer Communication Review*, 2009.

[23] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, Berlin, Germany, 2004.

[24] B. Koldehofe, F. Dürr, and M. A. Tariq. Tutorial: Event-based systems meet software-defined networking. In *Proc. of the 7th ACM Int. Conf. on Distributed Event-based Systems*, DEBS '13.

[25] B. Lantz, B. Heller, and N. McKeown. A network on a laptop: Rapid prototyping for software-defined networks. In *Proc. of 9th ACM Wshop. on Hot Topics in Networks*, 2010.

[26] M. Li, F. Ye, M. Kim, H. Chen, and H. Lei. A scalable and elastic publish/subscribe service. In *Proc. of IEEE Int. Parallel & Distributed Processing Symp.*, 2011.

[27] G. Mühl, H. Parzyjegla, and M. Prellwitz. Analyzing content-based publish/subscribe systems. In *Proc. of the 9th ACM Int. Conf. on Distributed Event-Based Systems, DEBS '15, Oslo, Norway*.

[28] N. K. Pandey, K. Zhang, S. Weiss, H. Jacobsen, and R. Vitenberg. Minimizing the communication cost of aggregation in publish/subscribe systems. In *35th IEEE Int. Conf. on Distributed Computing Systems, ICDCS*, 2015.

[29] P. R. Pietzuch and J. Bacon. Hermes: A distributed event-based middleware architecture. In *ICDCSW '02: Procs. of the 22nd Int. Conf. on Distributed Computing Systems*, 2002.

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

[31] M. A. Tariq, B. Koldehofe, S. Bhowmik, and K. Rothermel. PLEROMA: A SDN-based high performance publish/subscribe middleware. In *Proc. of 15th Int. Middleware Conference*, 2014.

[32] 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*, 2011.

[33] M. A. Tariq, B. Koldehofe, G. G. Koch, and K. Rothermel. Distributed spectral cluster management: A method for building dynamic publish/subscribe systems. In *Proc. of the 6th ACM Int. Conf. on Distributed Event-Based Systems (DEBS)*, 2012.

[34] S. Voulgaris, E. Rivire, A.-M. Kermarrec, and M. V. Steen. Sub-2-sub: Self-organizing content-based publish subscribe for dynamic large scale collaborative networks. In *the 5th International Workshop on P2P Systems*, 2006.