Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Combining Application Layer and Network Layer Filtering in Pub/Sub Systems

Fadi Sakkal

Course of Study:	M. Sc. Information Technology
Examiner:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Supervisor:	Dr. rer. nat. Sukanya Bhowmik
Commenced:	Mars 10, 2018
Completed:	October 10, 2018

Abstract

Content-based Publish/Subscribe systems deliver notifications from publishers to subscribers based on the content of the notifications. Low latency and bandwidth efficiency are two major concerns in these systems. Hybrid Publish/Subscribe systems were developed to take advantage of the new trend of Software Defined Networking (SDN). In these systems, notifications can be filtered on two layers, namely the application-layer and the network-layer. Since each one of these two layers has advantages over the other, it was important to have a selection algorithm to decide on which layer a certain notification has to be filtered. When choosing to filter notifications on the application layer, notifications have to be sent to software filters (servers) located in the topology. The placement of these servers is important for the overall performance of the system. In this thesis, we propose three different placement algorithms, each of which considers a different aspect of the system and tries to place the servers in a way that improves this aspect. The *K-Center* placement algorithm aims at minimizing the maximum distance between the sending node and the destination server. This will give an upper bound on the worst-case scenario regarding the latency. The*K-Median* placement algorithm is designed to minimize the average distance that the sent notifications have to pass to reach the server. The *Utilitarian* placement algorithm focuses on providing faster service to the packets which are intended for many subscribers, giving better overall service to the majority of the users at the expense of a worse service for some others. We have evaluated the proposed placement algorithms and compared them to each other according to different categories using two different topologies and two different distributions for the published notifications. As expected, the K-Center algorithm proved to be better in minimizing the maximum distance required for a notification to reach a server, while the K-Median and the Utilitarian algorithms showed similar results in most of the cases and were better in minimizing the average distance.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

API Application Programming Interface. 22

ASIC Application-Specific Integrated Circuit. 20

BFS Breadth-First Search. 32

CLI Command-Line Interface. 22

CORBA Common Object Request Broker Architecture. 15

EIGRP Enhanced Interior Gateway Protocol. 22

FPS Frames Per Second. 20

IP Internet Protocol. 19

JEDI Java Event-Based Distributed Infrastructure. 15

JMS Java Message Service. 15

LAN Local Area Network. 20

MAC Medium Access Control. 19

NBI Northbound Interface. 23

NMS Network Management System. 20

ONF Open Networking Foundation. 22

OS Operating System. 20

SBI Southbound Interface. 22

SDN Software Defined Networking. 16

SNMP Simple Network Management Protocol. 22

SSA Switch Selection Algorithm. 25

SSH Secure Shell. 22

TCAM Ternary Content-Addressable Memory. 16

1 Introduction

A growing amount of information is being exchanged on a daily basis over networks. Applications nowadays require more flexibility and real-time ability which demands a new communication paradigm. *Publish/Subscribe systems* realized this different paradigm by providing an application-independent *middleware* that offers loose coupling [1] and allows for asynchronous message exchange between involved parties.

In a publish/subscribe system, a participant does not have to maintain information about other publishers/subscribers in the network. It can instead rely on the system which provides many-to-many communication. This makes pub/sub systems flexible and scalable, in contrast to the Client/Server communication paradigm which revolves around the request/response model. In Client/Server paradigm, as the number of clients requesting a service increases, the time spent by the server for handling messages increases as well. This makes the Client/Server paradigm unsuitable for hard or soft real-time applications [ZSL08]. Some of the existing publish/subscribe systems include: Common Object Request Broker Architecture (CORBA) Notification Service [GCSO01], Java Message Service (JMS) [HBS+02], Java Event-Based Distributed Infrastructure (JEDI) [CDF01], and SIENA [CRW01]

In Publish/Subscribe systems, there are three types of actors: The *Publishers* (producer), *Subscribers* (consumer), and a *Communication Middleware*. Publishers provide information about the type of events they are going to publish (topics) and then publish these generated events to the middleware. Subscribers subscribe to specific topics or contents and expect to start receiving events that match their subscription. The middleware's goal is to determine the set of subscribers that should receive a particular event, route the event message as quickly and efficiently [2] as possible to the interested subscriber(s).

There are many variations of Publish/Subscribe systems like Channel-Based, Topic-Based and Content-Based. *Content-Based* is one variation which is notably bandwidth efficient. In these systems, As the name suggests, the content of the message determines its recipients. Rather than having the destination address in the header as an indication of the message's recipient(s), components placed inside the network called software routers (brokers) mediate between publishers and subscribers performing message filtering based on installed filters (subscribers interests).

The advantage of having broker-based Publish/Subscribe solution is that software brokers is able to perform perfect filtering. This means that the bandwidth efficiency is maximized and the *false positives* [3] is minimized. The disadvantage is that the filtering at the brokers happens at the

[1] In space and time
[2] Regarding bandwidth usage
[3] Notifications that are falsely forwarded to unintended subscribers

application layer. Application layer filtering cannot match up the performance of communication protocol implemented at the network layer, thus resulting in a higher end-to-end latency and lower throughput rates [Bho13].

Many applications suitable for the publish/subscribe paradigm are delay sensitive and require quick notifications delivery. E.g., online gaming, smart grid, traffic control, and financial applications. Therefore, recent efforts have been made to tackle the performance issue of content-based Publish/Subscribe systems by integrating *Software Defined Networking (SDN)* with them [BTK+17] [TKBR14]

When using SDN, with the help of protocols such as OpenFlow [Fun12], content filters are installed in a special type of memory called the Ternary Content-Addressable Memory (TCAM) inside the routers. TCAM filtering is line-rate filtering. Therefore, solving the end-to-end latency and low throughput issues that non-SDN content-based pub/sub systems usually suffer from.

Before installing a filter, the content has to be represented in a binary string which can then be placed in one of the message's header fields (content propagation). The number of bits available in the message's header is limited; thus, an efficient way of propagating the content to the header has to be implemented [CMT+11; KARW16]. The filter can then be installed on the router's TCAM via an SDN controller(s) [4] [Bho13; BTK+15]. *Spatial Indexing* [BTK+17], *Workload-Based Indexing* and *Dimension Selection* [BTG+18; BTGR16] are used by Bhowmik et al. in PLEROMA [TKBR14] as efficient mechanisms for representing message's content as a binary string that can then be put into one of the message's header and used in TCAM filtering.

TCAM is an expensive resource and therefore is limited in size [KLRW13]. A router that is SDN-compatible has usually TCAM size that can host up to a couple of thousands of filters. Bhowmik et al. introduced a solution to this limitation based on aggregating some of the content filters to reduce TCAM space usage while keeping the false positives as minimal as possible [BTBR17].

Back to the technology that made it all possible, one crucial component in SDN is the *controller*. An SDN controller is a logically centralized component responsible for installing and modifying filters in routers. To ensure consistent flow tables update in the network, the controller can only process one subscription request at a time, introducing a bottleneck in the system. Improvement can be made by using multiple controllers capable of processing multiple simultaneous requests in disjoint network partitions. With one controller present in the system, every request will be sent to that only one available controller. In contrast, with many controllers available, the requests will be sent to the controller responsible for the partition from which the request was generated.

From all this, we conclude that, although a network of software brokers provides accurate filtering, it suffers regarding responsiveness to event delivery. On the other hand, an SDN-based solution provides line-rate performance but falls short in terms of bandwidth efficiency. Bhowmik et. Al. Combined the benefits of the two worlds and provided a *hybrid solution* [BTHR16] that can perform filtering in software (application layer) using brokers and in hardware (network layer) using filters installed on TCAM. They have also provided mechanisms to determine the layer on which each event is filtered while minimizing false positives and considering delay requirements of the network.

[4] Some SDN architectures use multiple controllers

Thus, according to application requirements, some events will be filtered at the network layer while others will be filtered in software by a centralized server capable of concurrent processing of publications.

1.1 Thesis Overview

In this thesis, we will explore the spectrum between having one server responsible for filtering events at the application layer and a number of servers distributed across the network. The goal is to examine the difference between a purely centralized and completely decentralized application layer solution. We will identify the best possible placement of servers inside the network in an SDN Publish/Subscribe system implementing hybrid filtering of events at both application layer as well as the network layer. In chapter 3, we will introduce and discuss three placement techniques: the *K Center Placement*, the *K Median Placement*, and the *Utilitarian Placement* techniques which will optimize the system performance considering different factors. The K center placement technique will consider minimizing the maximum end-to-end delay as a placement factor. The K median placement technique will minimize the average end-to-end delay under different loads. The utilitarian placement technique is based on the philosopher Jeremy Bentham's ethical theory [Gra06], and its target is having a lower delay for the events that are interested for the larger number of subscribers, resulting in lower 'all in all' delay in the system. Following that, in chapter 4, we will introduce example scenarios and the results of the evaluation against a random placement of servers will be shown. Then, we will conclude and present our ideas regarding possible future work.

2 Background

The main objective of this chapter is to provide a good understanding of the key concepts that form the basis of this thesis. In the following sections we discuss the basic principles of the publish/subscribe paradigm, the basics of Software-Defined Networking and we introduce the notion of optimization problem and the usage of approximation algorithms to solve them.

2.1 Software Defined Networks

The goal of a network[1] is to deliver data from a source to a destination. It does so based on a set of protocols that regulate its operation. Software Defined Networks (SDN) is a networking architecture which was developed to tackle the fact that traditional networks architecture is decentralized and therefore complicated to manage while current networks require more flexibility and easy troubleshooting.

2.1.1 SDN Planes

To understand how things are arranged in an SDN architecture, it helps to take a look at the three planes that a networking device operates on. Everything that a networking device does can be related to one particular plane of the following three:

The Data Plane

The data plane contains the tasks that a networking device does to forward a data message. For example receiving, processing and forwarding data, de-encapsulating and re-encapsulating messages, matching the Medium Access Control (MAC) address to the MAC table entries or matching the Internet Protocol (IP) addresses to the routing table entries. Any action that is performed on the data message or directly aims at forwarding it is categorized under the data plane.

The Control Plane

Networking devices depend on their control plane to make a decision regarding the data they are handling. For example, routers need IP routes that are stored in the routing table; switches use entries in the MAC address table so that they can forward layer-two (L2) data frames. This information

[1] In most cases

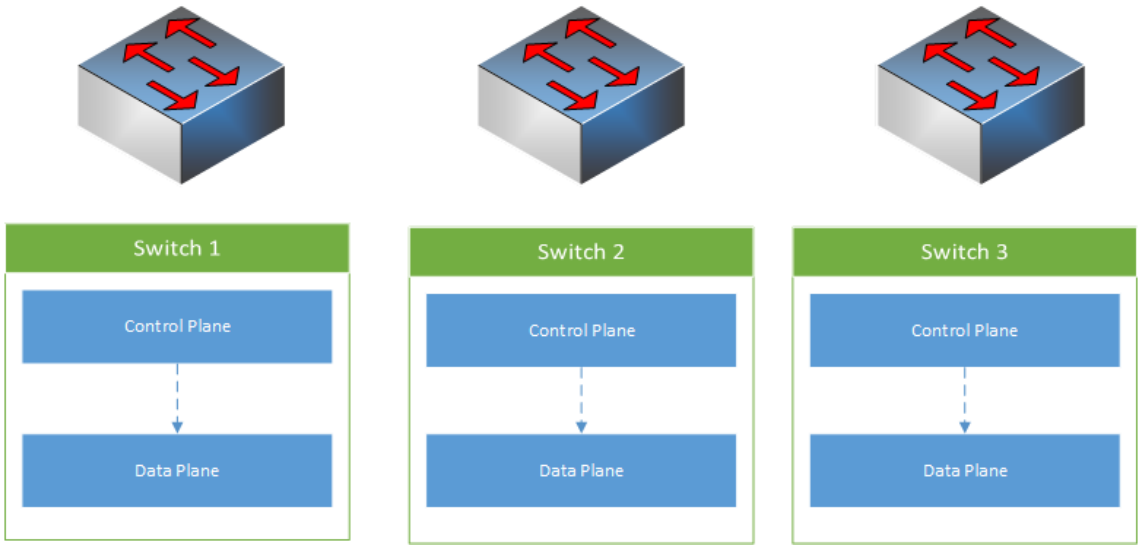


Figure 2.1: conceptual figure showing the distributed control plane in traditional networks

supplied by the control plane controls the work of the data plane. A traditional networking environment (non-SDN architecture) have the functionalities of the control plane distributed into each device. This means that the control plane logic is distributed among all devices. Figure 2.1.

The Management Plane

In contrast to the control plane, the management plane contains the overhead tasks that do not directly impact the data plane. Instead, this plane contains the protocols and tasks that allow managing networking devices. For example, Telnet [NKY+04] and SSH [VSP17] are two popular network management protocols that operate in the management plane. Network management protocols allow a Network Management System (NMS) to manage or monitor a networking device. Figure2.2

2.2 Data Plane Internals

Local Area Network (LAN) switches have to process and forward a large number of Frames Per Second (FPS). Today, even low-end switches are expected to handle millions of frames per second on each of their ports and the switch's general purpose CPU is not fast enough to cope with this huge volume of frames. Instead, switches use specialized hardware circuits to process and forward frames at the data plane [OW17]

An *Application-Specific Integrated Circuit (ASIC)*, As the name suggests, is a chip built to perform a specific purpose, like message processing in a switch. ASICs in switches need to perform MAC address table lookup. To make this lookup operation fast, the switch uses a particular type of memory to store the MAC address table. This memory is called the *Ternary Content-Addressable Memory (TCAM)*. This memory can be considered as an ordered array with parallel look-up ability [QYZ+18]

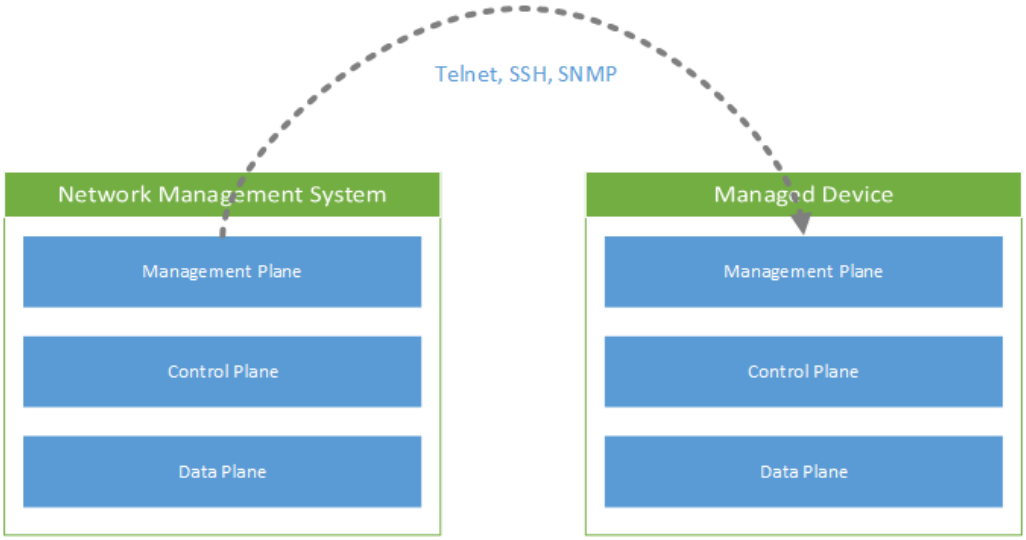


Figure 2.2: SDN Data, Control and Management Planes

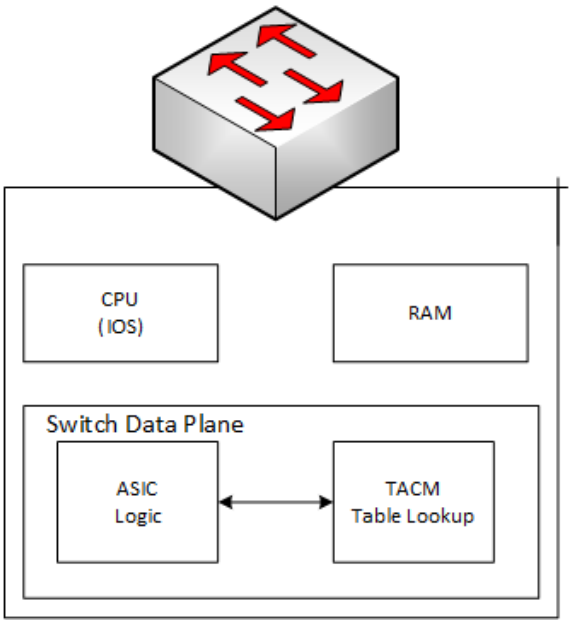


Figure 2.3: Key internal processing points in a typical switch

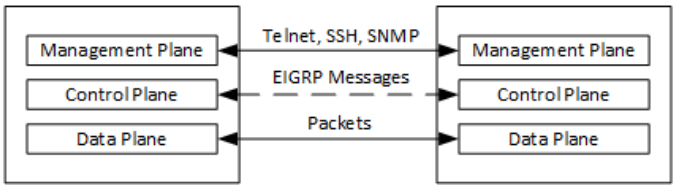


Figure 2.4: Management Plane for Configuration of Control and Data Plane

By having ASICs in a switch, the CPU will handle the switch's Operating System (OS) general operations, while the ASICs and the TCAM will handle the Switch's data plane operations as shown in figure2.3. Figure 2.4 connects the ideas of the three planes in one figure.

2.2.1 Network Architecture When Using SDN

SDN has many distinctions compared to traditional networking approach, one of which is where the control plane functions occur. In particular, SDN moves parts of the control plane tasks into software that runs on a centralized component [2] called a controller.

The Controller

As depicted in figure2.1, in non-SDN architectures, networking devices use a distributed control plane architecture, i.e., each device has its own control plane. To cooperate, the distributed control planes use messages to exchange information and communicate with each other. An example is when a router running Enhanced Interior Gateway Protocol (EIGRP) sends a multicast "query" message. Hence we say that the traditional networks use a distributed control plane. On the other hand, a controller in SDN plays the role of a centralized control plane in which all critical control plane functions run. A program running on the controller directly programs the data plane of the networking devices and can populate their forwarding tables via a Southbound Interface (SBI).

The Southbound Interface

In an SDN architecture, the controller has to communicate with the networking devices. It has become the norm in network drawings to draw the controller above the networking devices, just like in Figure 2.5. Hence the name of the interface between the controller and the networking device came to be known as the Southbound Interface or SBI. In other words, we can say that an SBI is an Application Programming Interface (API) between the controller (a program) and another program (on the networking device).

Several options exist to be used as an SBI: OpenFlow from the Open Networking Foundation (ONF) [Fou], OpFlex from Cisco [Cis], and more traditional Command-Line Interface (CLI) tools like Telnet, Secure Shell (SSH), and Simple Network Management Protocol (SNMP) [KKKK11].

[2] Some SDN architectures have multiple controllers

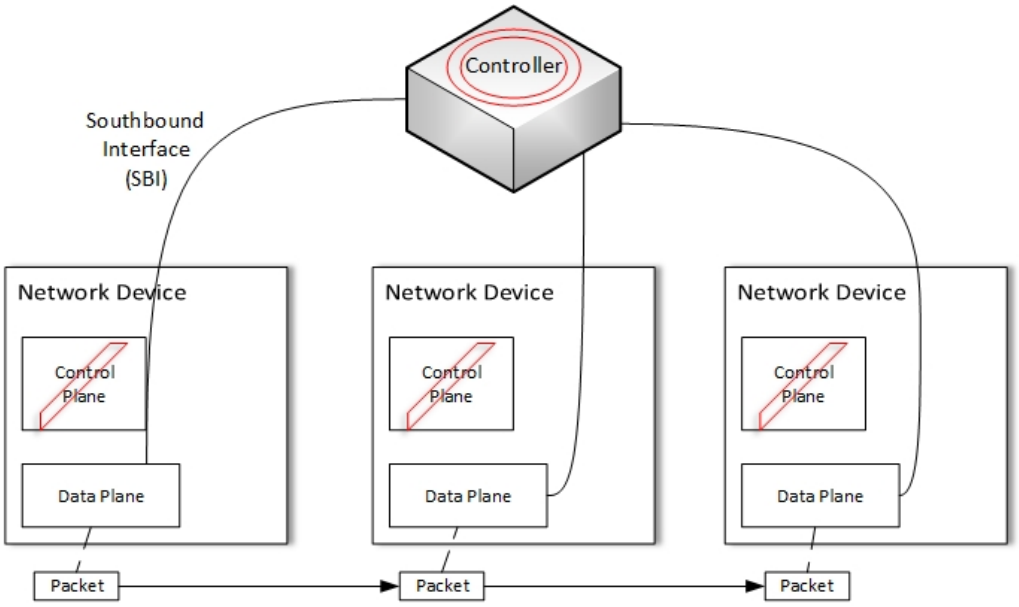


Figure 2.5: Centralized Control Plane and Distributed Data Plane

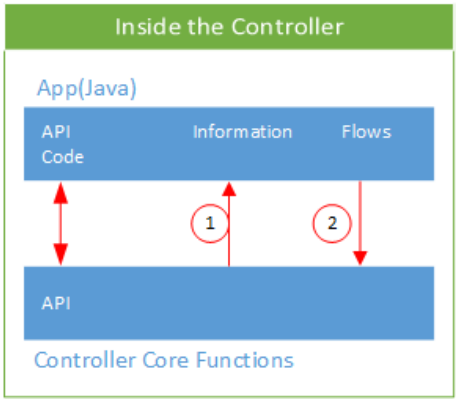


Figure 2.6: Java API, Java application communicates with controller

The Northbound Interface

The controller needs to know information about the networking devices before it can start modifying their control plane. Some of this information include: (i) A list of the devices comprising the network, (ii) Devices models and their capabilities, (iii) The ports/interfaces of each device, (ix) The topology and over which ports are the devices connected to each other.

A Northbound Interface (NBI) allows the use of the controller's collected data and functions by other programs. Programs can pull (via NBI) information from the controller. NBIs also allows the programs to use the controller's abilities to program flows into the devices via SBI. Like the case with SBI, the NBI gets its name from its drawing location. NBI is typically drawn above the controller, in what would be north on a map. Figure 2.6 shows the NBI as red arrows.

Figure 2.6 happens to depict the case where the controller and the application are running on the same machine. The program can as well run on a different machine. In both cases, the program would use an NBI as an API so that both programs can communicate.

2.3 Publish/Subscribe Systems

A Publish/Subscribe system is a communication paradigm that provides asynchronous message exchange between loosely-coupled parties [Gre].

Participants

In the Publish/Subscribe paradigm we have three involved actors: Publishers, Subscribers, and Communication Middleware. A publisher is an information producer that wants to share information with interested subscribers using messaging. A subscriber is an information sink that is interested in a subset of the information produced and published by the publishers. The middleware is a mean to communicate and provide loose coupling between publishers and subscribers [EFGK03]. Neither publishers nor subscribers need to have any pre-knowledge about each other; they can instead rely on the middleware to achieve the intended many-to-many communication. The middleware consists of a set of elements called routers or brokers.

Content-Based Publsih/Subscribe Systems

Rather than having the destination address in the header as an indication of the message's recipient(s), in content-based publish/subscribe systems, as the name suggests, the content of the message determines its recipients.

Paths between publishers and subscribers are determined on the basis of content filters (subscriber interests) installed on the routers [HW04]. These routers examine the content of the packets they receive and forward them [3] specifically to the interested subscribers, effectively filtering unneeded packets and avoid having them flowing through the network. The publishers and subscribers interact with the middleware using operations provided by it [Bho13]. This is depicted in figure 2.7

Publishers can either *advertise* (i.e., declare the nature of information they intend to publish), or *publish* information. Subscribers can perform two operations. These are: *subscribe* i.e., defining their interests and its reverse operation *un-subscribe*. The published information, generally called notifications, can be described as attribute-value pairs, $e =< attr0, value0; attr1, value1; attr2, value2; ..... >$ For example, a publisher *(P1)* can advertise that it intends to publish the reading of two sensors, one is for temperature *(T)*, and the other is for power consumption *(P)*. *P1* can then publish a notification like: $e1 =< T = 30, P = 100 >$

A subscribe operation yields a subscription and is also in the form of attribute-value pairs. In the case of Content-Based Publish/Subscribe system, the subscriber will announce its interests in the form of filters $s =< f1, f2 >$ where *f1, f2* denote advertisement filters. For example, a subscriber

[3] According to the installed filters

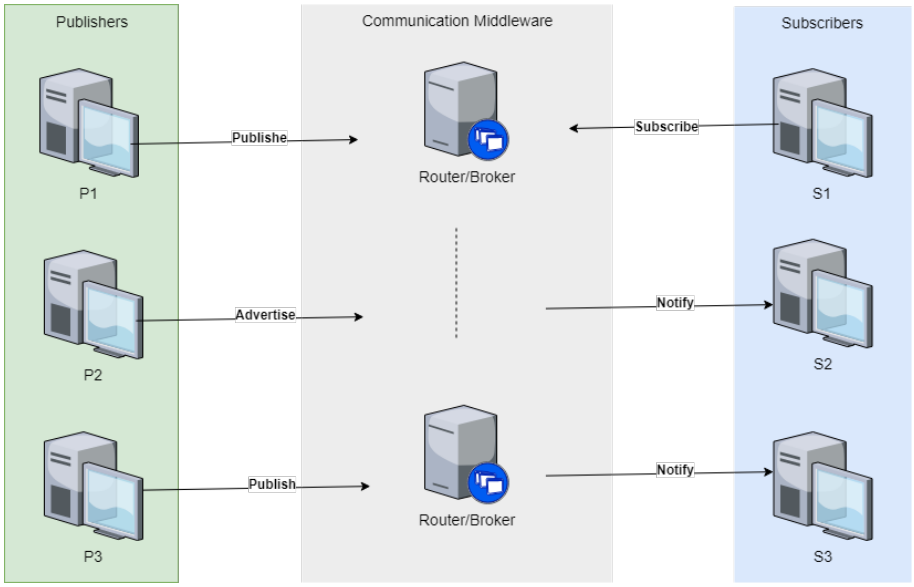


Figure 2.7: Overview of the Components and Operations in a Publish/Subscribe System

S1 can issue a subscription request in the form of $s =< T > 10, P > 30 >$ which is translated to an interest in the T values which are greater than 10 and P values which are greater than 30. We say that a notification e matches a subscription s if e satisfies a filter f from the subscriber's filters.

Many important factors determine the quality of the middleware. Bandwidth efficiency [4] and end-to-end delay [5] are two important performance measures. False positives are closely related to bandwidth efficiency, so it is beneficial to have an understanding of two terms, namely, false positives and false negatives. *False Positives* are events which are delivered to a subscriber that is not interested in receiving them. Whereas, *False Negatives* are events that were not delivered to a subscriber which had expressed its interest in receiving them [Bho13]. It is clear that, the less false positives that a Publish/Subscribe system deliver, the better the bandwidth efficiency will be. However, with filtering performed in software, this kind of implementation falls short (in terms of throughput and latency) compared to communication protocols implemented on the network layer. Recently, this has led to the realization of content-based publish/subscribe systems that utilize the capabilities of Software Defined Networking (SDN) [KDT13] [TKBR14]. Moreover, Bhowmik proposed a solution [Bho13] for the scalability issue that a single controller in an SDN-based Publish/Subscribe system may suffer from. This was achieved by introducing multiple controllers and having each controller responsible for a disjoint part of the network.

Whereas, application layer filtering has an advantage over network layer filtering, in terms of reduced false positives, it loses regarding end-to-end latency and throughput. In [BTHR16], Bhowmik et al. use a combination of application layer filtering and network layer filtering in a hybrid content-based SDN solution. They introduced the *Switch Selection Algorithm (SSA)* to identify the switches that need to have their packets filtered at the application layer. This thesis is concerned with introducing

[4] Minimizing the unnecessary bandwidth consumption
[5] The time needed for a published event to reach the interested subscribers

placement algorithms for servers capable of filtering events at the application layer. The placement strategy greatly impacts the delay of the system, since sending events from SSA-switches [6] to a server incurs more delay as the distance between the server and the switch increases.

2.4 Optimization Problems and the Need for Approximation

Optimization problems are common in many domains and disciplines. In an optimization problem, one has to find an optimal or near-optimal solutions with respect to some goal(s). An optimization problem can be a minimization problem or a maximization problem, depending on the objective function describing the problem [Rot11]. Usually, this kind of problems cannot be solved in one step; the solution instead requires multiple iterations. The number of iterations may vary depending on the nature and difficulty of the optimization problem. To find an exact solution to the problem, one has to design what is called an optimal algorithm. Choosing to apply an optimal algorithm to solve an optimization problem that requires lots of iterations might render the execution time unfeasible.

As an example, we will see that the placement problem we are trying to solve requires finding the best combination of server positions in a certain topology. Assuming that we have a relatively small topology consisting of 30 switches ($n = 30$) and that we have to place 15 servers ($r = 15$), then we will have according to the equation:

$$C(n, r) = \frac{n!}{r!(n-r)!}$$

more than 155 millions different combinations to examine. It is clear that a brute force algorithm will not scale well with the problem size. Thus, for such problems, we utilize what we call approximation algorithms or heuristic algorithms.

2.4.1 The NP-Hard Problems

In theoretical computer science, it is widely believed that P ≠ NP. To prove or falsify this conjecture is considered to be the most important open question in computer science today [For09]. Under the conjecture that P ≠ NP, a broad class of optimization problems cannot be solved exactly in polynomial time [Joh74]. The facility location problem is one such problem. We used the facility location problem as a starting point to develop the placement algorithm we will introduce in the next chapter.

Approximation Algorithms and Heuristics Algorithms

For an approximation algorithm, we need a mathematical proof certifying the quality of the solution produced by an approximation algorithm. This proof should consider the produced result in the worst-case scenario and then compare it to the optimal solution. The ratio between the two solutions is what is called the *approximation ratio*.

[6] Switches that have been chosen by SSA algorithm to send their events to the servers

More formally: For an optimization problem X we call an approximation algorithm A a γ-*approximation* if for all problem instances I of X the algorithm A produces a result R for I with $C(R) \leq \gamma * C(Ropt)$ for a minimization problem and $C(R) \geq \gamma * C(Ropt)$ for a maximization where $Ropt$ denotes the optimum solution and $C(.)$ the cost associated with a solution.

In contrast, a heuristic algorithm does not require a mathematical investigation regarding the produced solutions quality or any proof regarding the approximation ratio [Hro13]. Computational complexity theorists evaluate heuristic algorithms by considering average-cases and measure the required complexity of the execution. They also consider how well can heuristic algorithms perform on average for instances of the problem where the input is supplied by some specific distribution.

3 Server Placement Algorithms

3.1 Defining the Problem

Let $G = (\mathcal{V}, \mathcal{E})$ be an undirected graph with edge costs satisfying the triangle inequality, where G represents the topology of a hybrid Publish/Subscribe SDN-based system [BTHR16], $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. Let $\mathcal{R}_{\mathcal{SSA}} \subseteq V$ be the result of the Switch Selection Algorithm (SSA) [BTHR16], and $1 \leq k \leq |\mathcal{V}|$, is the number of the application-layer filtering servers to be placed in the topology. The goal is to place k servers on V and for any $s_i \in \mathcal{R}_{\mathcal{SSA}}$, Assign s_i to its closest server such that one of the following objectives is achieved:

- The maximum distance between any node $s_i \in SSA$ and its assigned server is minimized.
- The average distance that a packet sent from an SSA node to its correspondent server has to cover is minimized.
- Minimize the sum of the delays incurred on packets interested to the subscribers.

The nature of the application using the Publish/Subscribe system will determine the goal we have to achieve:

If the goal is to bind the worst case scenario of the end-to-end notification delay, we have to achieve the first objective. This objective is achieved by applying the *K-Center Placement algorithm* discussed later.

In case we want to achieve the second objective, we will favor SSA nodes [1] with higher traffic at the expense of SSA nodes with lower traffic. This is achieved by applying the *K-Median Placement Algorithm*.

If we decided that a packet which has to be delivered to a higher number of subscribers should be delivered faster than a packet subscribed to [2] by a smaller number of subscribers, we can use the *Utilitarian Placement Algorithm*.

Lastly, we have implemented a *Random Placement Algorithm* than will merely place the k servers randomly on the topology. This was done to be able evaluate the quality of each of the algorithms against the random placement. The results of the evaluation are available in chapter 4

[1] $s_i \in \mathcal{R}_{\mathcal{SSA}}$
[2] Satisfies a subscriber's filter

The Proposed Solutions

3.2 The K-Center Placement Algorithm

Having the presented general definition of the placement problem from the previous section and the objective of minimizing the maximum distance between an SSA-selected node and its responsible center [3], we can formally define the K-Center problem as follows:

Given an undirected tree $G = (\mathcal{V}, \mathcal{E})$, a subset $\mathcal{R}_{\mathcal{SSA}}$, and natural number k, the K-Center problem is to select a subset $C \subseteq \mathcal{V}$ with $|C| = k$ such that, $\forall s_i \in \mathcal{R}_{\mathcal{SSA}}$, $max\ d(s_i, c_i)$ is minimized. Where c_i is the closest point from C to s_i.

Remarks

- The facility location problem [SKBJ17] is one famous variation of the K-Center problem.
- Since the problem is NP-hard [HS85], approximation techniques have to be used. In the following we will introduce an algorithm based on the greedy approach.

3.2.1 The Greedy Approach to Solve the K-Center Problem

The idea of this algorithm is to start by placing the first server on the node with the highest degree and is selected from the SSA. Then for the placement of the rest of the servers, in each iteration, the algorithm searches for the node which is the farthest from all previously placed servers between all the SSA selected nodes. This will effectively shorten the maximum distance between an SSA node and the closest server to it. The algorithm terminates when there are no more servers to be placed. See algorithm 3.1.

The input of the algorithm is the set of nodes $\mathcal{V}$, the set of lines $\mathcal{E}$, and the number of centers to be placed K. In line 2, the function *sortDegree* sorts $\mathcal{V}$ in ascending order according to the degree of the nodes. In lines $3 \rightarrow 7$, the algorithm tries to place the first server on the node which has the highest degree and was selected from the SSA. In line 10, the algorithm enters in a loop until K reaches 0. In each iteration, the algorithm tries to find the farthest node from all the previously placed centers. This is done through the function *findFarthest* which assign the found node to the variable *newCenter*. The placement phase is done at this point and the next phase of assigning each SSA-node to its closest center can start. In line 15, the algorithms loops the nodes and for each SSA-node, it searches for its closest center via the functioon *findClosestCenter*.

3.2.2 Walkthrough for the K-Center Placement Algorithm

In the following, we will use figure 3.1 as a walkthrough example for the K-Center Placement algorithm and see how the algorithm will place $K = 3$ servers on the topology. For the sake of simplicity and without the loss of generality, we will assume that all switches have been selected by

[3] The terms 'center' and 'server' are used interchangeably in this context

Algorithm 3.1 K-Center Placement Algorithm

Require: The set of nodes ($\mathcal{V}$), the set of links ($\mathcal{E}$) and k are available as input

```
1:  function KCenter(V, E, k)
2:      V ← sortDegree(V)
3:      for i ← V.size() to 1 do
4:          if V_i ∈ R_SSA then
5:              V_i.isCenter ← true
6:              k − −
7:              Break
8:          end if
9:      end for
10:     while k > 0 do
11:         newCenter ← findFarthest(V, E)
12:         newCenter.isCenter ← true
13:         k − −
14:     end while
15:     for s_i ∈ V do
16:         if s_i ∈ R_SSA then
17:             s_i.responsibleCenter ← findClosestCenter(s_i)
18:         end if
19:     end for
20:     return V
21: end function
```

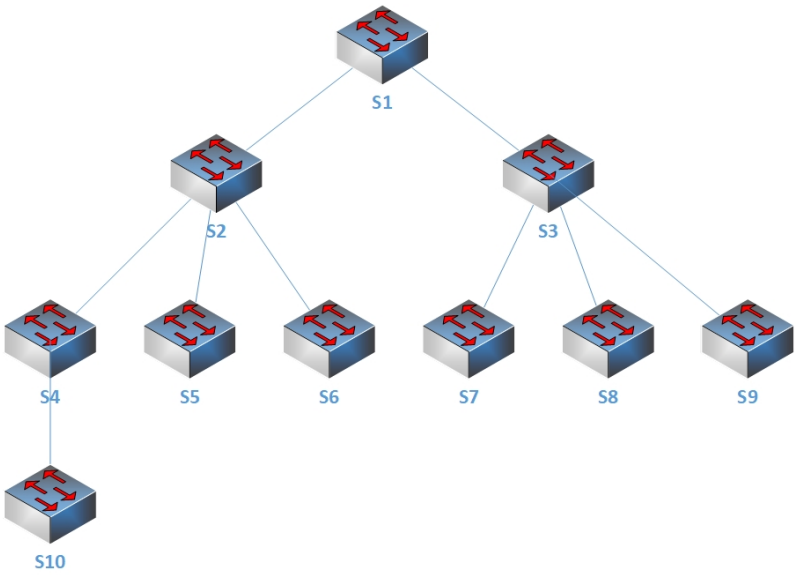


Figure 3.1: Example Topology for the K-Center Placement Algorithm

Node	S2	S3	S1	S4	S5	S6	S7	S8	S9	S10
Degree	4	4	2	2	1	1	1	1	1	1

Table 3.1: Nodes After Sorting

Node	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
Closest Server	S2	S2	S2	S2	S2	S2	S2	S2	S2	S2
Distance to Closest Server	1	0	2	1	1	1	3	3	3	2

Table 3.2: Finding Farthest Node After the First Server (S2) Has Been Placed

the SSA. The algorithm works in 2 phases, namely: the placement phase (lines $2 \rightarrow 14$) in which the K servers are placed on the topology, and the assignment phase (lines $15 \rightarrow 19$) in which we assign every SSA-node to its closest server.

The input of the algorithm will be: The set of nodes $\mathcal{V}$ which contains all the nodes ($S1 \rightarrow S10$), the set $\mathcal{E}$ will be comprised of all the links between the switches, and K will be 3.

The Placement Phase

The algorithm starts by an ascending in-place sorting of the nodes set $\mathcal{V}$ according to the degree of the node. The sorting result is shown in table 3.1. After the set $\mathcal{V}$ is sorted, we can start with the placement phase which in turns can be split into two phases.

First Server Placement

S2 has the highest degree[4] between all nodes; therefore the first server should be placed on S2. The reason behind placing the first server on the node with the highest degree is that a higher degree means more edges from/to the node which will increase the reachability of the node. This means that a node with a higher degree is a more 'strategical' node than one with a lower degree.

At this point the first server has been placed and the value of K is decreased by one ($K = 2$).

Second and Third Servers Placement

Now the algorithm enters in a loop (lines $10 \rightarrow 11$ in the algorithm) as long as $K > 0$. By using Breadth-First Search (BFS) [LS10] and the sets $\mathcal{V}$ and $\mathcal{E}$, the algorithm can find the distance between each node and its closest center.

Table 3.2 shows that all of S7, S8, and S9 have the largest distance to reach their closest server (S2). Hence, the new server could be placed on any one of the three nodes. We will assume that S9 has been chosen and the second server can now be placed on S9.

The value of K is decreased once again by one ($K = 1$) since we have placed a new server.

[4]The same can be said about S3

Node	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
Closest Center	S2	S2	S9	S2	S2	S2	S9	S9	S9	S2
Distance to Closest Center	1	0	1	1	1	1	2	2	0	2

Table 3.3: Finding Farthest Node After two servers (S2 and S9) Have Been Placed

Node	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
Responsible Server	S2	S2	S9	S10	S2	S2	S9	S9	S9	S10
Distance to Responsible Server	1	0	1	1	1	1	2	2	0	0

Table 3.4: The Responsible Server for Each of The Nodes

The execution continues by recalculating the distance between each node and the closest placed server (S2 or S9).

Table 3.3 shows that S7, S8, and S10 are the farthest nodes from the two placed servers. with a distance of two to reach the closest server. Therefore, the third and final server could be placed on any one of S7, S8, and S10. We will assume that S10 has been chosen for the server placement.

The Assignment Phase

After that all three servers have been placed, the placement phase is done. The assignment phase can start. In the algorithm, lines $12 \rightarrow 15$ loops the nodes and assign for each SSA node its closest center. This is done by the function *findClosestCenter*, which execute a BFS starting from the node until a server is found. In our example, the assignment phase will give the values shown in table 3.4.

3.3 The K-Median Placement Algorithm

Having the presented general definition of the placement problem from the previous section 3.1, and the goal of minimizing the average distance between an SSA-selected node and its responsible center, we can formally define the K-Median problem as follows:

Given an undirected tree $G = (\mathcal{V}, \mathcal{E})$, a subset $\mathcal{R}_{SS\mathcal{A}}$, and natural number k, the K-Median problem is to select a subset $C \subseteq \mathcal{V}$ with $|C| = k$ such that the average distance that a packet sent from a SSA-node to its server has to travel is minimized.

Unlike the K-Center placement algorithm where all SSA-nodes are treated equally, in the K-Median placement algorithm we differentiate between the nodes, and we give some nodes are more importance than others. The importance is initially established based on the packets handled by a node in a defined time interval. For example: in SDN, the controller can periodically query the switches' counters and get the number of packets they have handled so far. According to this information, we can assign an importance value to each node. This value should reflect that the node that handles more packets is more important than one that handles fewer packets. If we think how the large corporations geographically place their branches, we notice that they tend to open branches in/or close to places where a large number of their customers reside. This is logical since they want the majority of their customers to spend less time to reach their branches. The K-Median placement algorithm retains the same concept. In contrast, the K-Center Placement Algorithm does not take into consideration the handled packets on each node (the number of customers in the example). For the K-Center, all cities are equal, and the goal was to minimize the maximum distance from a city to the closest opened branch.

The algorithm starts by querying how many packets each node has handled in a unified interval. This can be done through the function *getHandledPackets*. This will then be used by the function *initializeImportance* to assign an importance value to each node. At this point, the importance value of the node is equal to the number of packets it has handled which will give a busier node a higher importance value than a more idle one. After having the importance values, we can calculate the cost values. The cost value in this context refers to the cost of placing a server on a certain node and is calculated according to the equation 3.1.

$$Cost(S_j) = \sum_{S_i \in R_{ssa}} d(S_i, S_j) * importance(S_i) \tag{3.1}$$

Equation 3.1 ensures that the cost of each node will be proportional to its distance from all SSA nodes. The algorithm then sorts the nodes according to the cost and chooses the one which has the lowest cost to place a server on it. Each time a server is placed, the importance values should be modified to reflect the changes that have been made in the topology, namely that a server has been placed and that the importance values of the nodes should be reduced proportionally with their distance to the placed server. This is done according to equation 3.2

$$importance(S_j) = importance(S_j) - \frac{1}{d(i,j)+1} * importance(S_j) \tag{3.2}$$

Algorithm 3.2 K-Median Placement Algorithm

Require: The set of nodes ($\mathcal{V}$), the set of links ($\mathcal{E}$) and k are available as input

```
1:  function KMedian(V, E, k)
2:      for i ← 1 to V.size() do
3:          V_i.hanledPackets ← getHandledPackets(V_i)
4:          initializeImportance(V_i)
5:      end for
6:      while k > 0 do
7:          for j ← 1 to V.size() do
8:              for i ← 1 to V.size() do
9:                  if V_i ∈ R_SSA then
10:                     V_j.cost ← V_j.cost + d(V_i, V_j) * V_i.importance
11:                 end if
12:             end for
13:         end for
14:         V ← sortCost(V)
15:         for i ← 1 to V.size() do
16:             if V_i.isCenter = false then
17:                 V_i.isCenter ← true
18:                 index ← i
19:                 k − −
20:                 Break
21:             end if
22:         end for
23:         for j ← 1 to V.size() do
24:             if j = index then
25:                 V_j.importance ← 0
26:             else
27:                 V_j.importance ← V_j.importance/(d(V_index, V_j) + 1)
28:             end if
29:         end for
30:     end while
31:     return V
32: end function
```

Equation 3.2 calculates the new importance values of the nodes. With these modified importance values, the algorithm is now ready for a new iteration. The cost values are calculated once again, and a new server is placed accordingly.

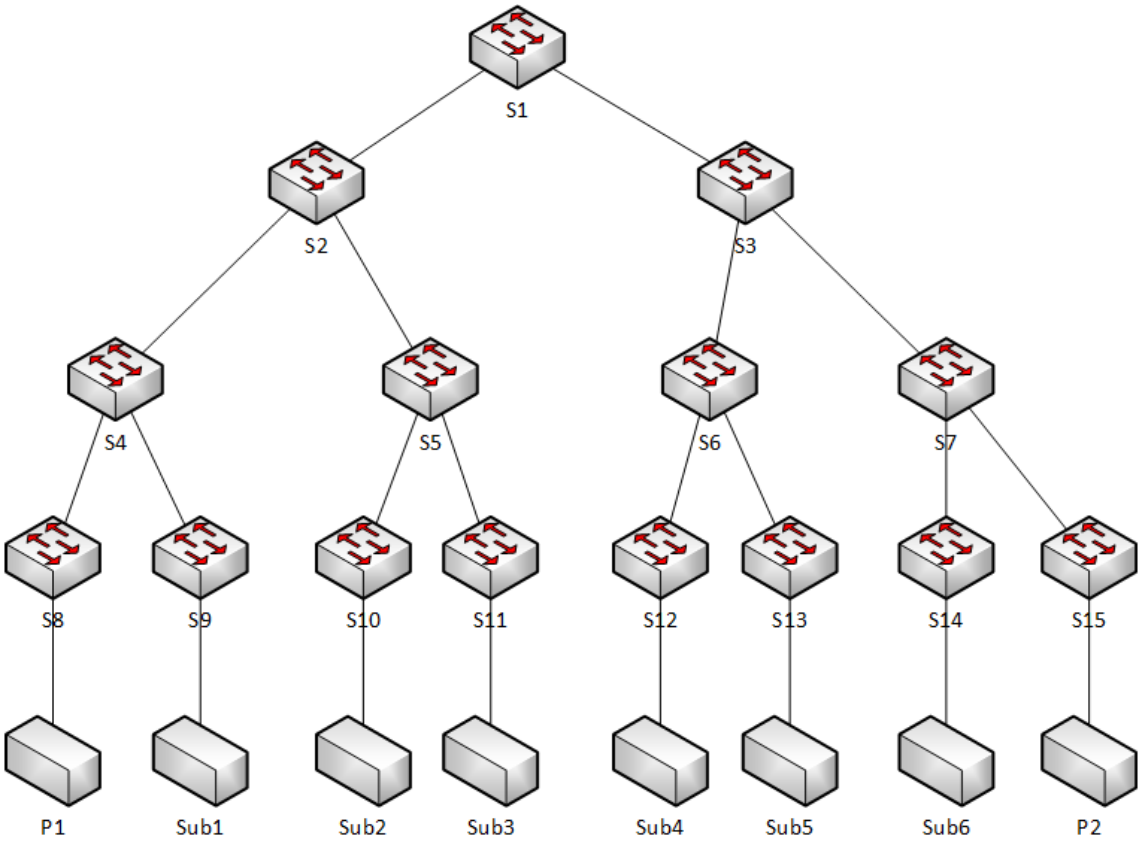


Figure 3.2: The Topology for the K-Median Placement Algorithm Walkthrough Example

Node	Type	Published/Subscribed values
P1	Publisher	Publishes P values
Sub1	Subscriber	Subscribed to P = [25,30]
Sub2	Subscriber	Subscribed to P = [50,100]
Sub3	Subscriber	Subscribed to T >40
Sub4	Subscriber	Subscribed to P >100
Sub5	Subscriber	Subscribed to T <100
Sub6	Subscriber	Subscribed to T [10,30]
P2	Publisher	Publishes T values

Table 3.5: Information about publishers and subscribers in figure 3.2

3.4 Walkthrough for the K-Median Placement Algorithm

Assume having the topology shown in figure 3.2, where (P) denotes a publisher, and (Sub) denotes a subscriber. The information regarding the publishers/subscribers is given in table 3.5. Having the paths installed between the publishers and subscribers so that packets generated by P1, and P2 can reach the intended subscribers, We want to place K servers according to the K-Median placement algorithm.

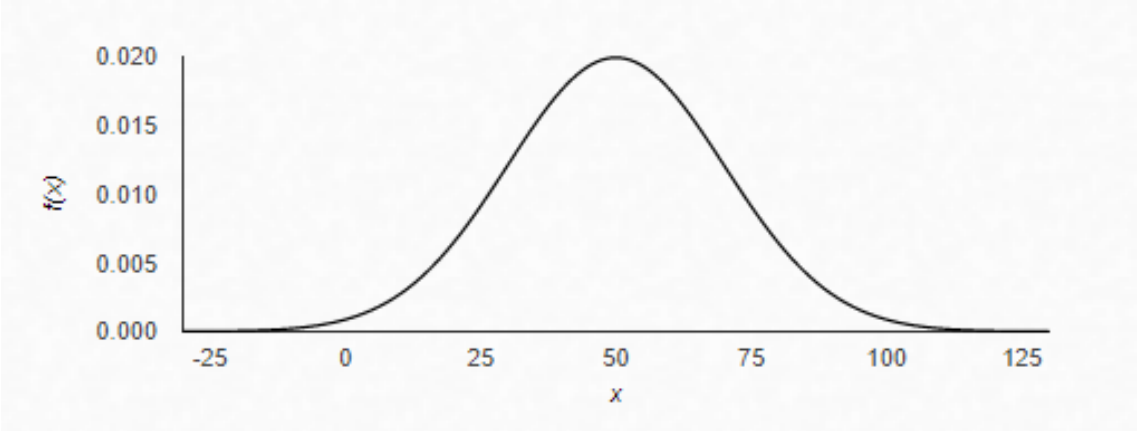


Figure 3.3: The Normal Distribution of the Generated P Values

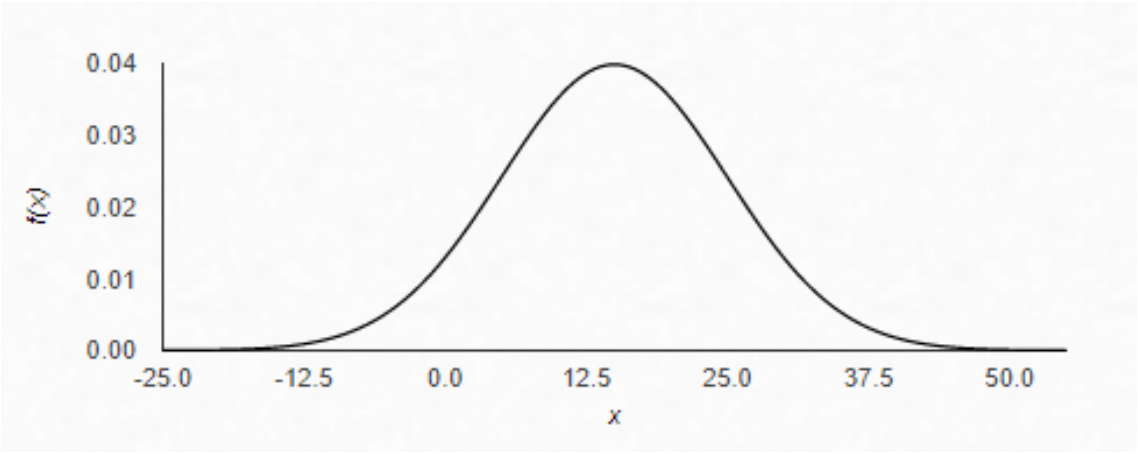


Figure 3.4: The Normal Distribution of the Generated T Values

We will generate 1000 packets at P1 distributed normally with a mean ($\mu = 50$) and a standard deviation ($\sigma 1 = 20$). The distribution is depicted in figure 3.3. Also, 1000 packets are generated at P2 according to the normal distribution with a mean ($\mu = 15$) and standard deviation ($\sigma = 10$) as shown in figure 3.4.

The packets will then flow in the network from publishers to interested subscribers. As the packets flow through the network, the incoming packets counters on each switch ($S1 \rightarrow S10$) will store the number of packets handled by each node. When the execution of the algorithm starts, it can then query these counters and get the number of handled packets as shown in table 3.6. This is done through the function *getHandledPackets* in line 3 of the algorithm 3.2. Here, we need to explain the term 'importance value' which is initially equal to the node's handled packets value. The importance value will be used as an indication of how important is it to place a server close to the node. Remember that only SSA-nodes will send their packets to servers for filtering, therefore as we will see, only these SSA-nodes will affect the calculation of the 'cost' directly influencing the placement of the servers. As the algorithm progresses and servers are placed, the importance value

Node	Handled Packets	Importance	SSA-node
S1	2	2	Yes
S2	143	143	Yes
S3	168	168	Yes
S4	973	973	No
S5	141	141	Yes
S6	168	168	Yes
S7	1000	1000	No
S8	1000	1000	No
S9	132	132	Yes
S10	141	141	Yes
S11	0	0	Yes
S12	2	2	Yes
S13	166	166	Yes
S14	812	812	Yes
S15	1000	1000	No

Table 3.6: The Importance Values

Node	S1	S2	S3	S4	S5	S6	S7	S8
Cost	4688	5449	3931	7060	6760	5134	4182	8935
Node	S9	S10	S11	S12	S13	S14	S15	
Cost	8671	8353	8635	7005	6677	4433	6075	

Table 3.7: The Cost Value for Each of The Nodes

of each node will be modified to reflect how important is it to place the next server closer to each of the nodes. To make the example interesting, we will assume that the SSA has not selected S4, S7, S8, and S15.

From the importance values in table 3.6 we can then calculate the cost of placing a server on a certain node using equation 3.1.

The cost of a node is a score value that indicates how suitable is the node to host a server where a lower cost indicates a better suitability. The cost is proportional to the sum of the importance values and the distance between the nodes. For example, assume that we want to calculate the cost of placing a server on S1. S1 is one hop away from S2, and S3. It is two hops away from S5 and S6 and it is three hops away from the rest of the nodes. To calculate the cost of placing a server on S1, we will have:

$$Cost(S_1) = 1{*}importance(S_2, S_3){+}2{*}importance(S_5, S_6){+}3{*}importance(S_9, S_{10}, S_{11}, S_{12}, S_{13}, S_{14})$$

Note that the nodes: S4, S7, S8 and S15 were not included in the cost calculation because they are not SSA-nodes.

By calculating the costs for all nodes, we will have the values shown in table 3.7.

Node	Importance	Host a Server	SSA-node
S1	1	No	Yes
S2	96	No	Yes
S3	0	Yes	Yes
S4	730	No	No
S5	106	No	Yes
S6	84	No	Yes
S7	500	No	No
S8	800	No	No
S9	106	No	Yes
S10	113	No	Yes
S11	0	No	Yes
S12	2	No	Yes
S13	111	No	Yes
S14	542	No	Yes
S15	667	No	No

Table 3.8: The Modified Importance Values After That The First Server Has Been Placed on S_3

Since S3 has the lowest cost value of 3931, the first server should be placed on S3. The k value should be decreased to become 2. Before starting with a new iteration, the importance values should be decreased proportionally according to the distance between the node and the newly placed server. The node's importance where the server has been placed will become 0, while the other nodes' importance values will be calculated according to equation 3.2.

Where i is the index of the node where the last server has been placed (3 in our example).

As a result, the new calculated importance values are shown in table 3.8. At this point an iteration of the algorithm is completed, the execution goes back to line 6 in algorithm 3.2. The algorithm repeats until all K servers have been placed. The set $\mathcal{V}$ is returned and the execution terminates.

3.5 The Utilitarian Placement Algorithm

Introduction to Utilitarianism

Before starting with the algorithm's explanation, it is beneficial to briefly introduce the ethical theory of utilitarianism. Utilitarianism suggests that the best action is the one that maximizes utility. 'Utility' is usually defined as the well-being of sentient entities. Jeremy Bentham, the founder of utilitarianism, described utility as the sum of all pleasure that results from an action, minus the suffering of anyone involved in the action [Ben96].

As discussed in section 2.3, in content-based publish/subscribe systems, subscribers subscribe to specific content. The middleware is then responsible for delivering the notifications which match subscribers' interests to them. Here, we need to differentiate between popular contents which are interested and subscribed to by many subscribers from the unpopular contents which are less subscribed to. If we follow the utilitarian theory, it is better to quickly deliver the notifications that hold popular content at the expense of slower delivery for the notification that holds unpopular content. By doing so, many subscribers will receive their notifications faster, and only a few will have slower notifications delivery. This will maximize the sum of the satisfaction in the system, which is what utilitarian aims for.

In section 3.3, for the implementation of the KMedian-Placement algorithm, we query the value of the packet counter of each node. We then use this value to calculate the initial importance value of the node. Here, in the utilitarian placement algorithm, it is not sufficient to only query the packet counters to calculate each node's importance value; instead, we need to know how many subscribers are interested in each packet. This can be done at the controller which has a global view of all the subscriptions. This is only done at the beginning of the execution and before the placement so that the overhead of this procedure does not affect the controller.

Formal Definition

Given an undirected tree $G = (\mathcal{V}, \mathcal{E})$, a subset $\mathcal{R}_{\mathcal{SSA}}$, and natural number k, select a subset $C \subseteq \mathcal{V}$ with $|C| = k$ such that the sum of the delay incurred on all subscribers from sending notifications for application-layer filtering is minimized.

The only modification that should be done to the K-Median placement algorithm shown in 3.2 has to do with the function getHandledPackets in line 3. Instead of merely getting the number of handled packets, this function should now register the number of packets multiplied by the number of subscribers interested in each of them. For example: if a packet handled by a node $\mathcal{S}_i$ is subscribed to by 5 subscribers, then this single packet should be calculated as 5. The algorithm can then continues as explained in the K-Median placement algorithm section with the function initializeImportance benfiting from the output of the function getHandledPackets to assign the importance value for each node. This will increase the overhead of the Utilitarian placement algorithm compared to the K-Median placement algorithm.

4 Analysis and Results

4.1 The Experiments Setups

For the evaluation of the placement algorithms introduced in chapter 3, we will use two sample topologies. The first one is called '*Tree Topology*', it is made up of 15 nodes, two publishers and six subscribers as shown in figure (3.2). The second topology is called '*Mixed Topology*', it is composed of 25 nodes, three publishers and 13 subscribers. The mixed topology is shown in figure (4.7).

For each of the algorithms introduced in chapter 3, we ran the execution on both of the topologies six times. Three executions when the publishers are using normal distribution[Bry12] and three executions when the publishers are using uniform distribution[1][JVV86]. The collected results are averaged and organized in tables. Moreover, the figures comparing the placement algorithms with random placement of servers are provided as well.

In the experiments, we will assume that the publishers generate two types of published information, namely the 'P' values and the 'T' values.

4.2 'Tree Topology' Experiment's Results

In this experiment, we will use the topology shown in figure (3.2) and the information about the publishers/subscribers from table (3.5). Publishers can publish integer values according to two different distributions, namely the normal distribution and the uniform distribution. We will show the results when using the normal distribution first and then we will move on to show the results when using the uniform distribution.

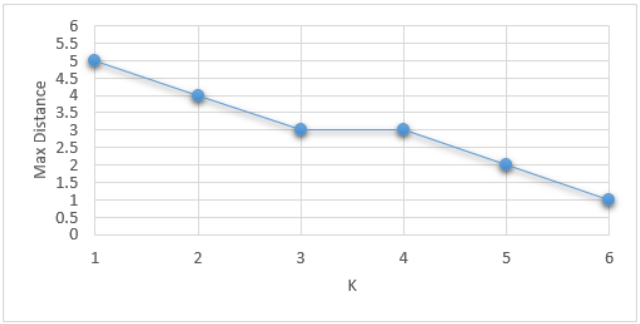
4.2.1 Case 1: Published Information Follow the Normal Distribution

In this case, P1 publishes 'P' values with a mean ($\mu = 50$) and a standard deviation ($\sigma = 20$). The publisher P2 publishes normally distributed 'T' values with a mean ($\mu = 15$) and a standard deviation ($\sigma = 10$).

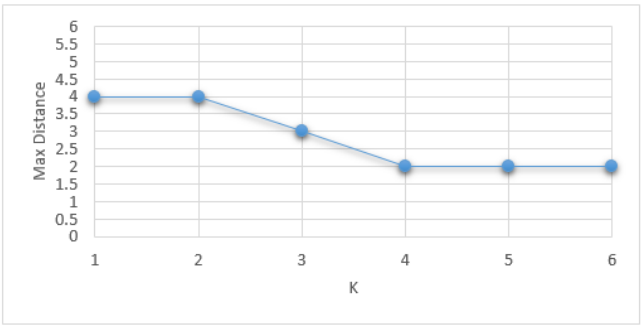
Figure (4.1) shows the maximum distance between a SSA-node and its responsible server. We see that the K-Center starts with a worse result when $K = 1$, but then manages to provide the best result at $K = 6$. The random placement performed the worst. The results are not surprising as the K-Center algorithm consider minimizing the maximum distance. Figure (4.2) shows the average

[1]Details regarding the distributions will be clarified later

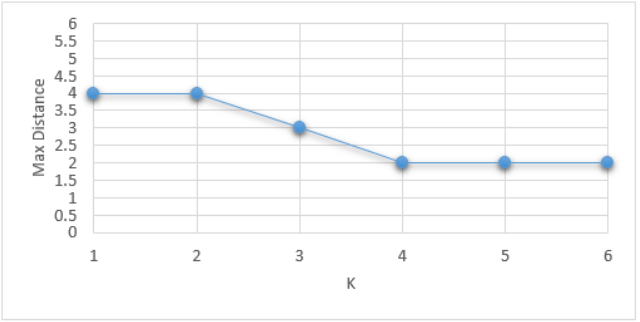
distance between a SSA-node and its responsible server. We recorded almost a tie between the K-Median and the Utilitarian algorithms. This is due to the similarities between the two algorithms. The K-Center started poorly when $K = 1$, but quickly recovered at higher K values. Figure (4.3) shows the average hops that a packet sent from a SSA-node has to cover to reach the responsible server. We recorded very small differences between the K-Median and Utilitarian algorithms.



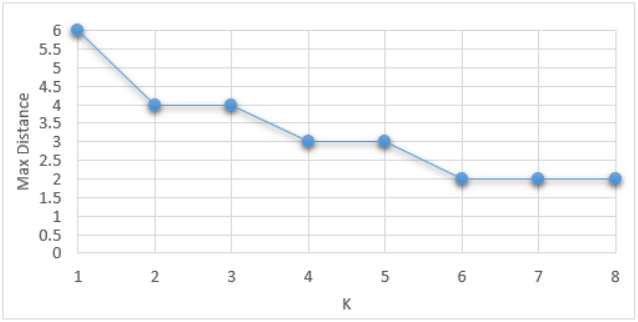
(a) K-Center



(b) K-Median

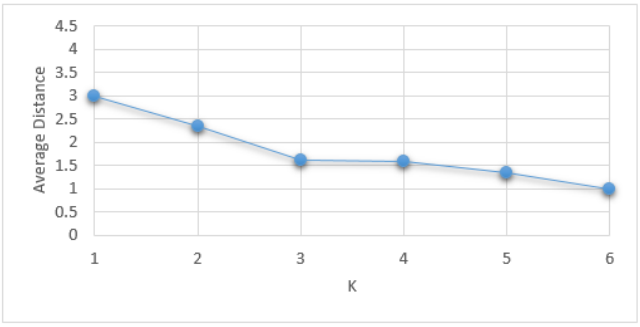


(c) Utilitarian

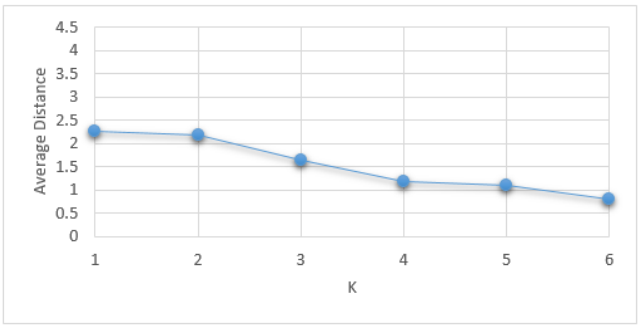


(d) Random Placement

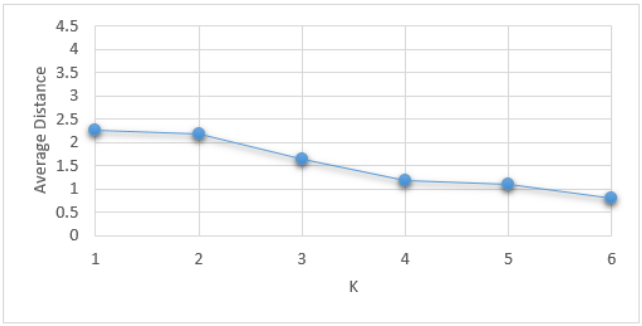
Figure 4.1: Maximum Distance Between a SSA-Node and its Closest Server Using Different Placement Algorithms for Different K Values When Publishers Use Normal Distribution



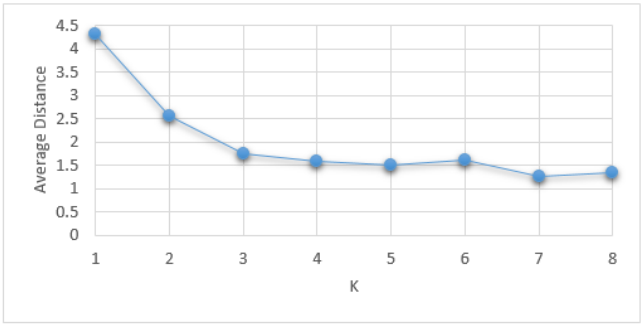
(a) K-Center



(b) K-Median

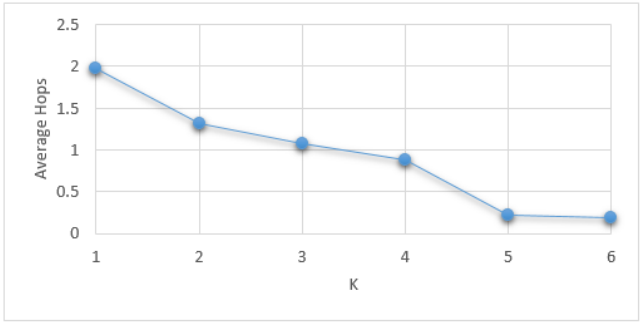


(c) Utilitarian

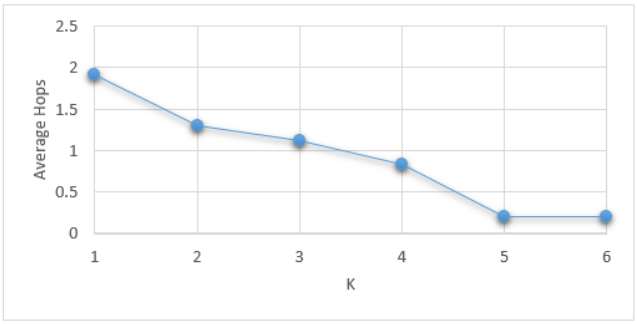


(d) Random Placement

Figure 4.2: Average Distance Between a SSA-Node and its Closest Server Using Different Placement Algorithms for Different K Values When Publishers Use Normal Distribution



(a) K-Median



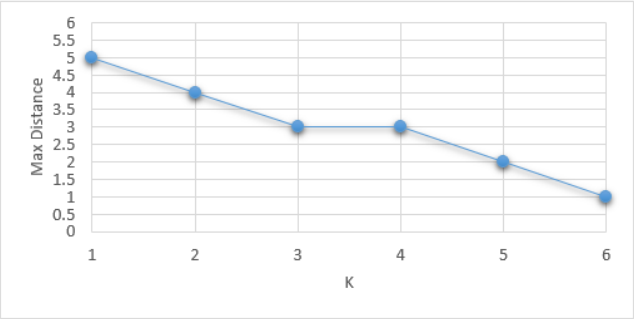
(b) Utilitarian

Figure 4.3: Average Hops That a Packet Sent From an SSA-Node To Its Closest Server Has To Cover Using Different Placement Algorithms For Different K Values When Publishers Use Normal Distribution

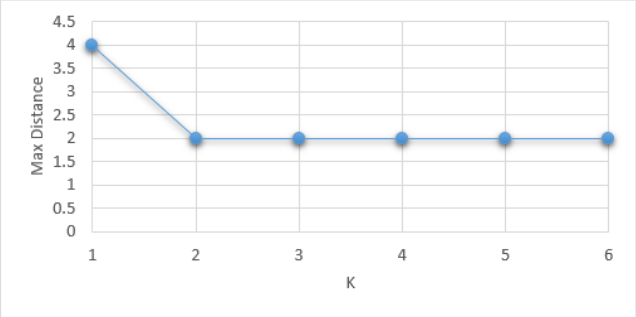
4.2.2 Case 2: Published Information Follow the Uniform Distribution

In this case, P1 publishes (P) values while each of P2 and P3 publishes (T) values. All of the publishers' published information follow the uniform distribution in the range $[0, 50]$. In the following, we will show the plots depicting the results we got from executing each one of the developed placement algorithms on the small topology shown in figure (3.2).

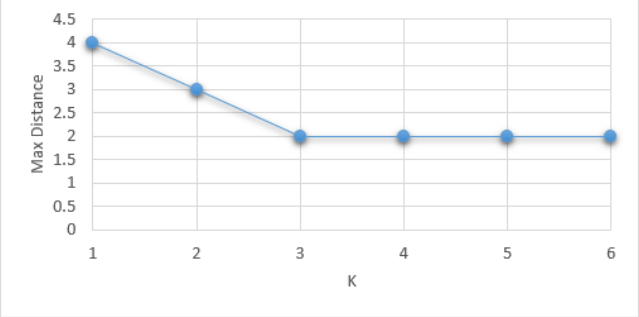
Note that the K-Center placement algorithm and the random placement do not take the handled packets into account when placing servers. Therefore, the results of the placement in these two cases will not be affected by the change of the distribution and are still identical to the results shown in figure (4.1) and figure (4.2) when normal distribution was used. For the sake of completion and ease of comparison they are presented here with the results of the K-Median and Utilitarian algorithms. Figure (4.4) shows the maximum distance between a SSA-node and its responsible server. Once again the K-Center performed better for $K = 6$ registering maximum distance of 1. In figure (4.5), we have the average distance between a SSA-node and the responsible server. Once again the K-Median and the Utilitarian gave the best results in this category. Figure (4.3) shows the average hop-count that a packet has to travel to reach the responsible server.



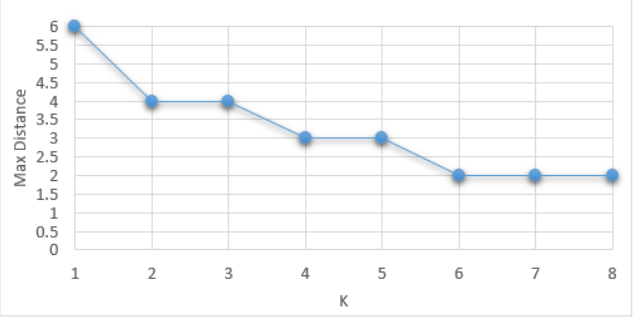
(a) K-Center



(b) K-Median

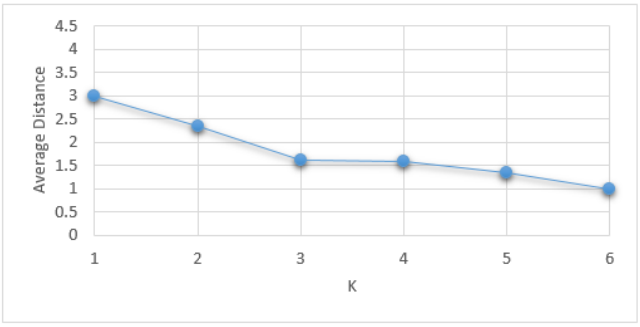


(c) Utilitarian

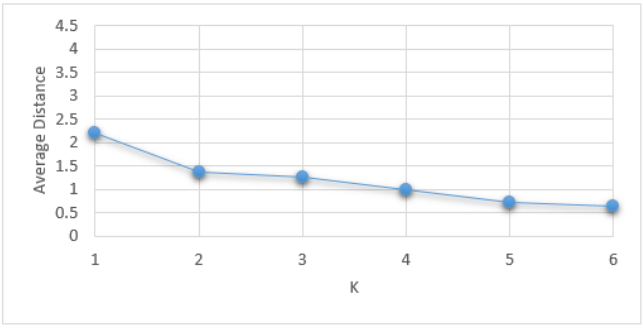


(d) Random Placement

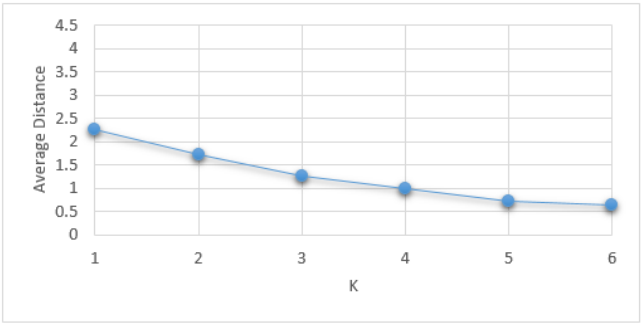
Figure 4.4: Maximum Distance Between a SSA-Node and its Closest Server Using Different Placement Algorithms for Different K Values When Publishers Use Uniform Distribution



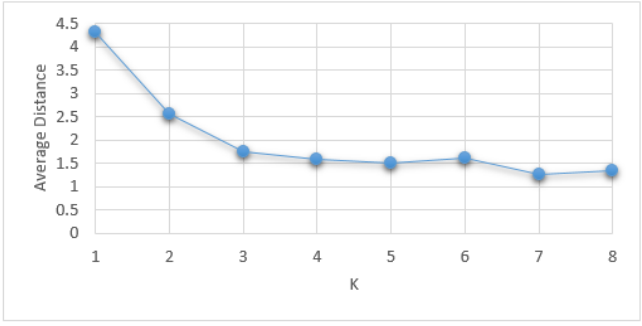
(a) K-Center



(b) K-Median

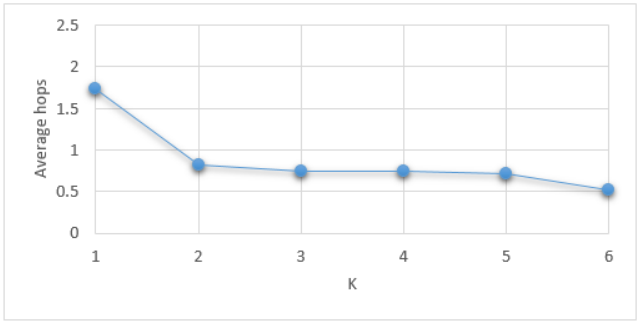


(c) Utilitarian

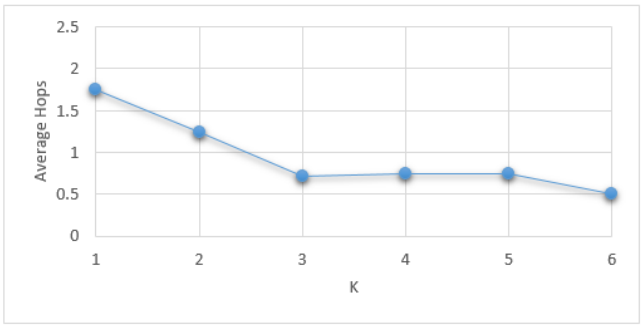


(d) Random Placement

Figure 4.5: Average Distance Between a SSA-Node and its Closest Server Using Different Placement Algorithms for Different K Values When Publishers Use Uniform Distribution



(a) K-Median



(b) Utilitarian

Figure 4.6: Average Hops That a Packet Sent From an SSA-Node To Its Closest Server Has To Cover Using Different Placement Algorithms For Different K Values When Publishers Use Normal Distribution

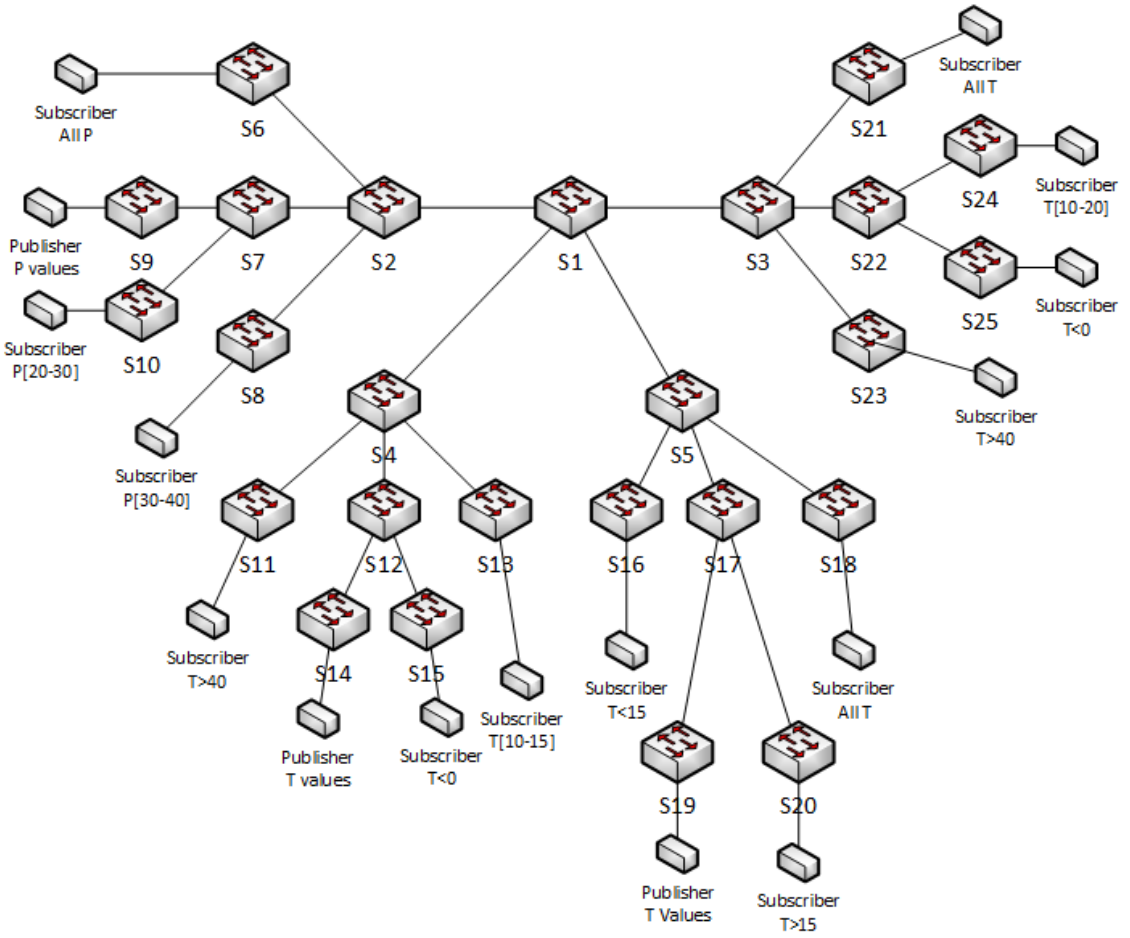


Figure 4.7: The 'Medium Topology' Used for Evaluation

4.3 'Mixed Topology' Experiment's Results

The information regarding the publishers and subscribers in the medium topology is shown in table (4.1). The topology is shown in figure (4.7). We will start by presenting the results when the information published by the publishers is distributed according to the normal distribution; then we will show the results when the published information is distributed according to the uniform distribution.

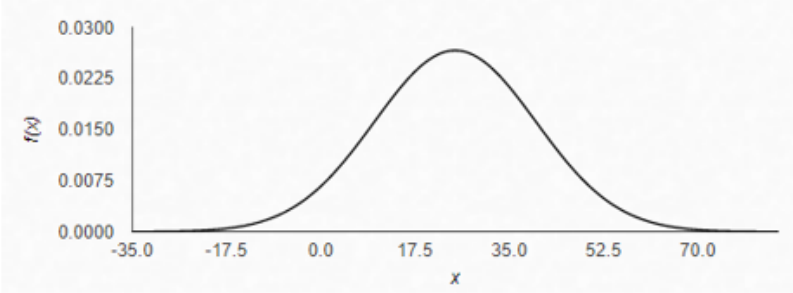
4.3.1 Case 1: Published Information Follow the Normal Distribution

In this case the published information follow the normal distribution. Namely: publisher Pub1 publishes P values that have mean (μ = 25), and standard deviation (σ = 15) as shown in figure (4.8a), Publisher Pub2 publishes T values that have mean (μ = 15) and standard deviation (σ = 10) as shown in figure (4.8b) and Publisher Pub3 publishes T values with the following properties: mean (μ = 25) and standard deviation (σ = 10) as shown in figure (4.8c). Figure (4.9 shows the results of the maximum distance between a SSA-node and its responsible server. The K-Center

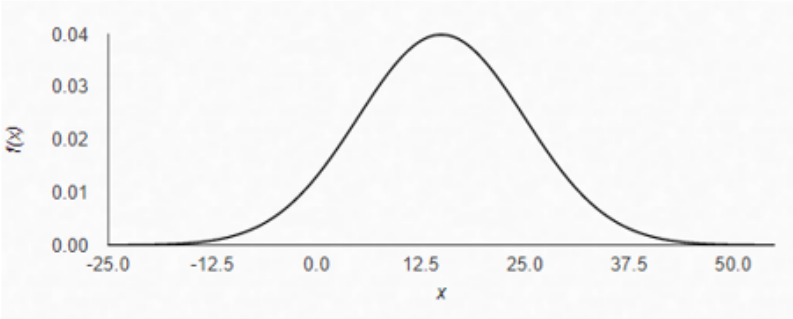
Node	Type	Published/Subscribed values
Pub 1	Publisher	Publishes P values
Pub 2	Publisher	Publishes T values
Pub 3	Publisher	Publishes T values
Sub 1	Subscriber	Subscribed to all P values
Sub 2	Subscriber	Subscribed to P=[20,30]
Sub 3	Subscriber	Subscribed to P = [30,40]
Sub 4	Subscriber	Subscribed to T >40
Sub 5	Subscriber	Subscribed to T <0
Sub 6	Subscriber	Subscribed to T = [10,15]
Sub 7	Subscriber	Subscribed to T <15
Sub 8	Subscriber	Subscribed to T >15
Sub 9	Subscriber	Subscribed to all T values
Sub 10	Subscriber	Subscribed to T >40
Sub 11	Subscriber	Subscribed to T <0
Sub 12	Subscriber	Subscribed to T = [10,20]
Sub 13	Subscriber	Subscribed to all T values

Table 4.1: Publishers and Subscribers Information in the Mixed Topology

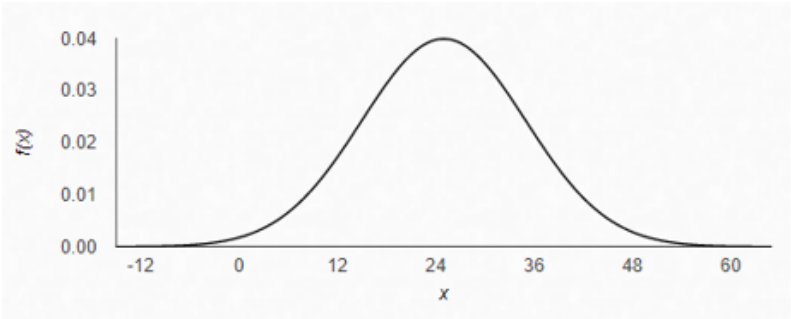
suffers from a bad initial choice of the first placed server. The K-Median and Utilitarian algorithms keep showing strong similarities in their results. The random placement gives unpredicted results. Figure (4.10) shows the average distance between a SSA-node and its responsible server. Here the K-Median and the Utilitarian algorithms give slightly better results than the K-Center. Figure (4.11) shows the average hop-count that a packet has to travel to reach the responsible server.



(a) The Distribution of the P values Published By Pub1

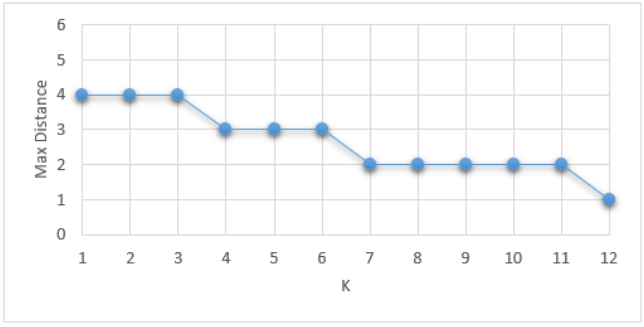


(b) The Distribution of the T values Published By Pub2

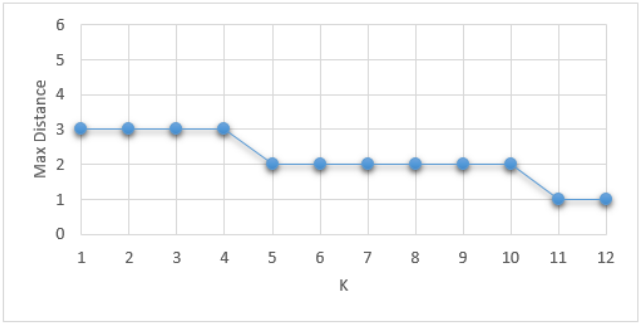


(c) The Distribution of the T values Published By Pub3

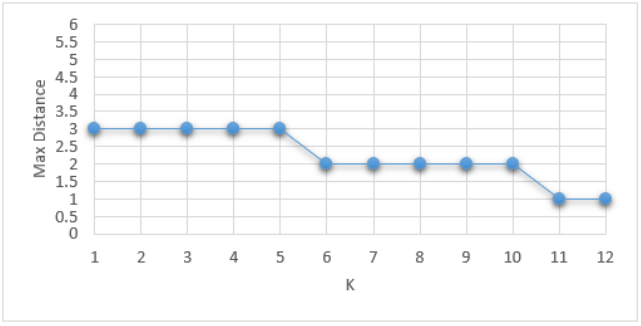
Figure 4.8: The Distribution of the *P* and *T* values Published In the Medium Topology



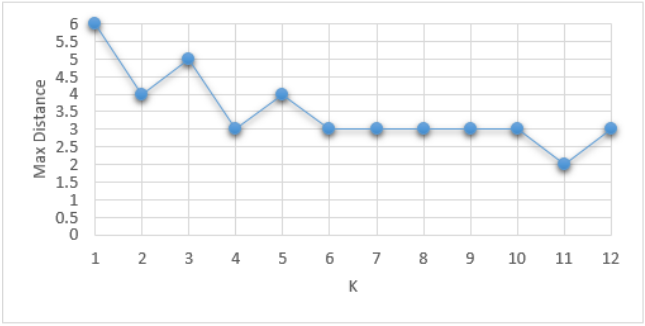
(a) K-Center



(b) K-Median

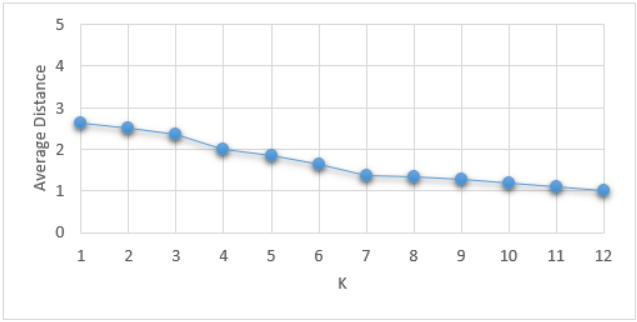


(c) Utilitarian

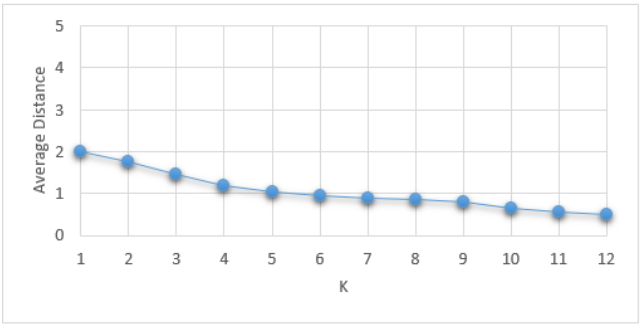


(d) Random Placement

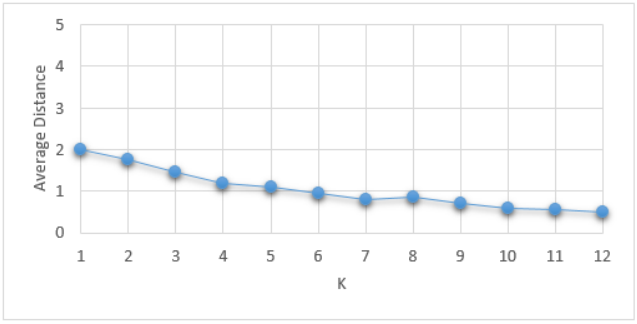
Figure 4.9: Maximum Distance Between a SSA-Node and its Closest Server Using Different Placement Algorithms for Different K Values When Publishers Use Normal Distribution



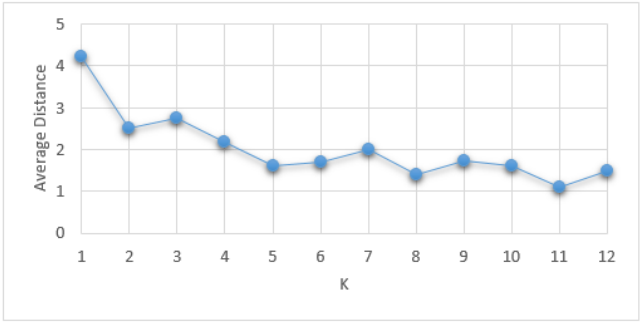
(a) K-Center



(b) K-Median

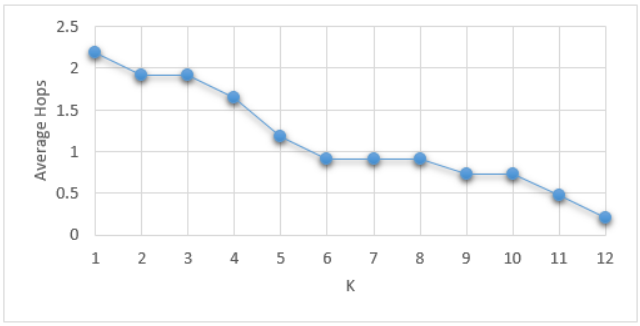


(c) Utilitarian

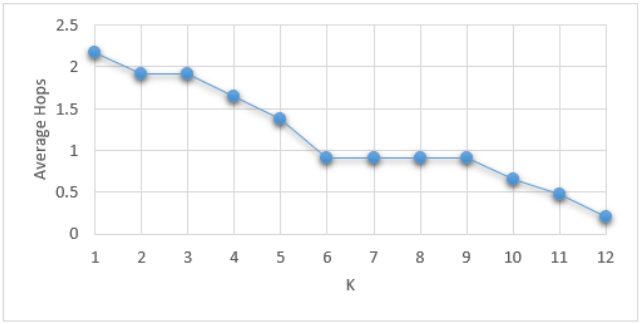


(d) Random Placement

Figure 4.10: Average Distance Between a SSA-Node and its Closest Server Using Different Placement Algorithms for Different K Values When Publishers Use Normal Distribution



(a) K-Median



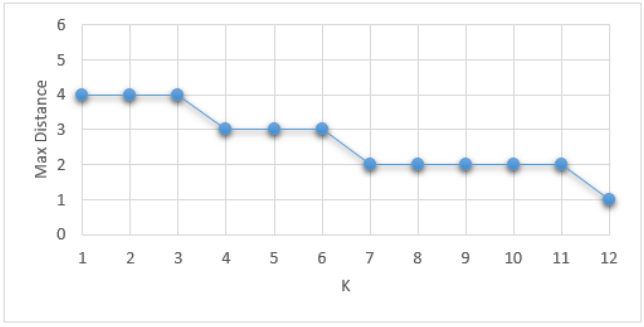
(b) Utilitarian

Figure 4.11: Average Hops That a Packet Sent From an SSA-Node To Its Closest Server Has To Cover Using Different Placement Algorithms For Different K Values When Publishers Use Normal Distribution

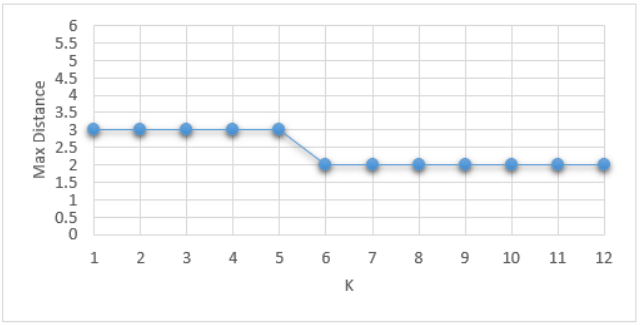
4.3.2 Case 2: Published Information Follow the Uniform Distribution

In this case, neither the random placement of servers nor the K-Center placement will be affected by the change of the distribution. Therefore, the results are identical to the previous case and are shown in figures (4.9), and (4.10). Figure (4.12) shows the maximum distance between a SSA-node and its responsible server. The K-Center reaches 1 at $K = 12$, while the K-Median and the Utilitarian algorithms get to only 2. Figure (4.13) shows the average distance between a SSA-node and its responsible server. The K-Median and the Utilitarian algorithms produce slightly better results in this category. Figure (4.14) shows the average hop-count that a packet is required to go over to reach the responsible server.

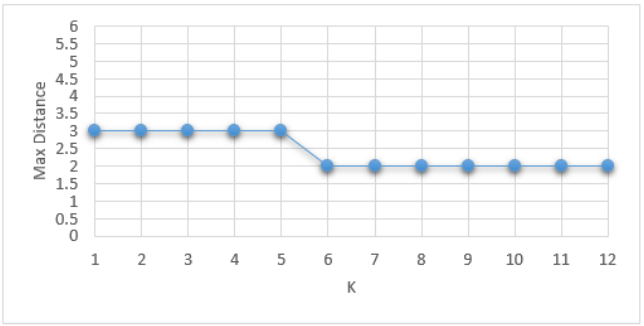
For more precise comparison of the results, you can take a look at appendix (A). The appendix contains the collected results which were used to draw the plots in this chapter.



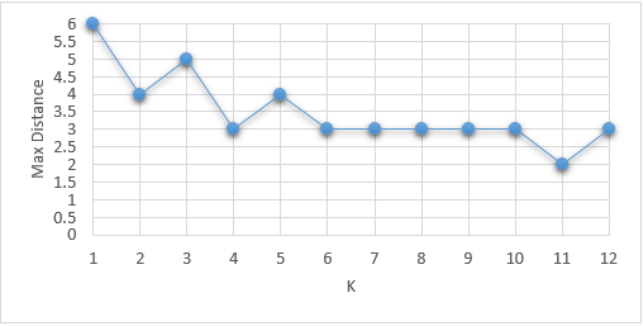
(a) K-Center



(b) K-Median

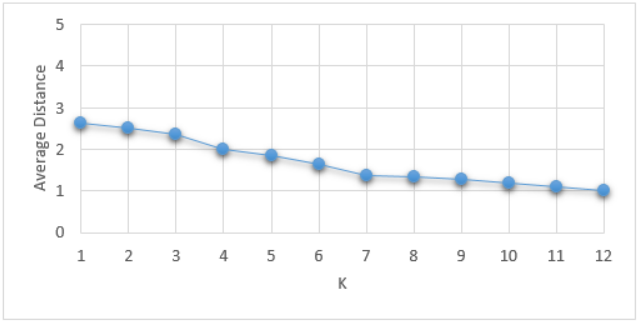


(c) Utilitarian

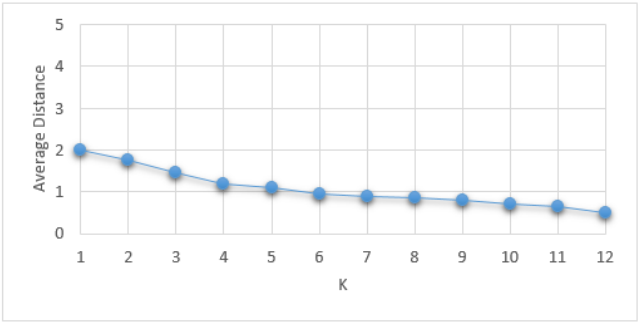


(d) Random Placement

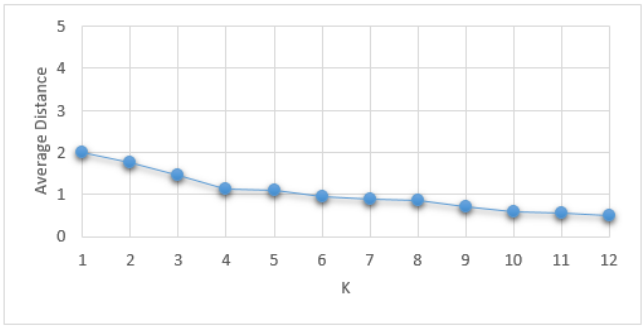
Figure 4.12: Maximum Distance Between a SSA-Node and its Closest Server Using Different Placement Algorithms for Different K Values When Publishers Use Uniform Distribution



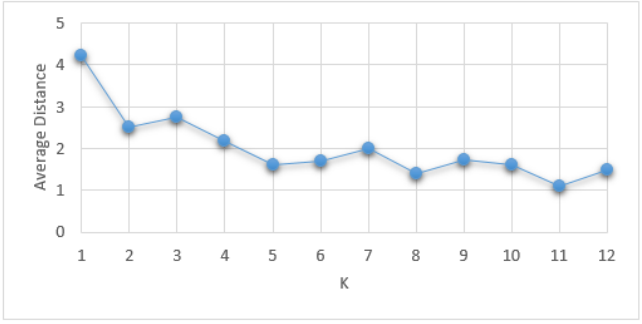
(a) K-Center



(b) K-Median

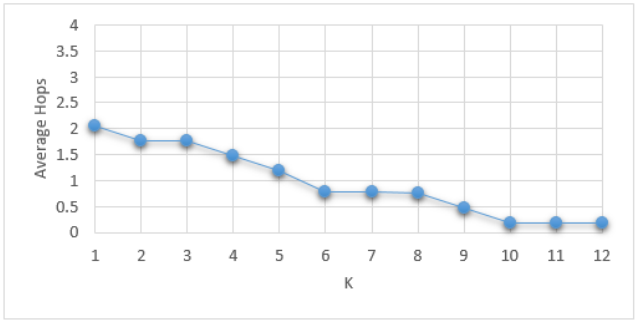


(c) Utilitarian

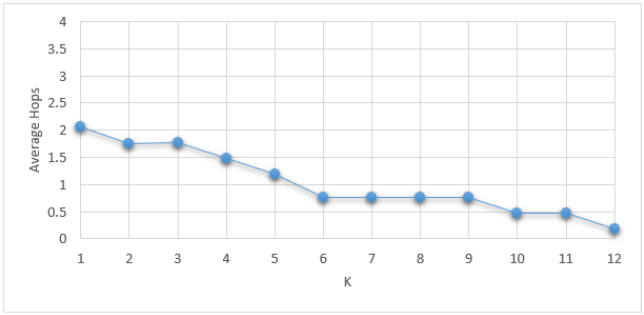


(d) Random Placement

Figure 4.13: Average Distance Between a SSA-Node and its Closest Server Using Different Placement Algorithms for Different K Values When Publishers Use Uniform Distribution



(a) K-Median



(b) Utilitarian

Figure 4.14: Average Hops That a Packet Sent From an SSA-Node To Its Closest Server Has To Cover Using Different Placement Algorithms For Different K Values When Publishers Use Uniform Distribution

5 Conclusion and Outlook

In this thesis, we implemented three different server placement algorithms for hybrid SDN-based Publish/Subscribe systems. By experimenting with the number of servers to be placed (K), we explored the spectrum between a centralized and a decentralized application layer solutions. Each one of the three developed algorithms considers a different usage scenario of the system and place the servers in a way that optimize the usage scenario. That it is why it is important to understand the goal of the Publish/Subscribe system in order to choose the placement algorithm which will give the better results. Since the topology can change over time, the algorithms can be ran periodically to cope with the changes and modify the placement decision.

The results of our experiments showed that the K-Center placement algorithm gave better results when the measured value was the maximum distance between a SSA-node and the responsible server. On the other hand, the K-Median and the Utilitarian algorithms gave better results when measuring the average distance between a SSA-node and the responsible server. Unlike the K-Median and the Utilitarian, the K-Center is not affected by the published information and hence it was not affected by the change of the distribution that the published notifications were following. The random placement gave results that ranged between similar and worse (compared to the other algorithms).

In this thesis, we assumed that the distance that a packet has to travel is directly affecting the latency in the system. Despite that this assumption is valid, it would be interesting to compare the results shown in this thesis with ones from a real SDN testing environment that can measure the latency in terms of time.

A Experiments Results Organized in Tables

This appendix includes the results of the conducted experiments from which we have drawn the figures in chapter 4.

K	Max Distance	Average Distance
1	5	3
2	4	2.33
3	3	1.62
4	3	1.57
5	2	1.33
6	1	1

Table A.1: The Results of Applying the K-Center Placement Algorithm on the Tree Topology

K	Max Distance	Average Distance
1	4	2.63
2	4	2.50
3	4	2.35
4	3	2
5	3	1.86
6	3	1.64
7	2	1.38
8	2	1.33
9	2	1.27
10	2	1.20
11	2	1.11
12	1	1

Table A.2: The Results of Applying the K-Center Placement Algorithm on the Mixed Topology

K	Total Packets	Total Hops	Average Hops	Max Distance Between an SSA-Node and Its Center	Average Distance Between an SSA-Node and Its Center
1	1239	2444	1.97	4	2.27
2	1274	1686	1.32	4	2.18
3	1277	1370	1.07	3	1.63
4	1272	1121	0.88	2	1.18
5	1275	292	0.22	2	1.09
6	1267	246	0.19	2	0.81

Table A.3: The Results of Applying the K-Median Placement Algorithm on the Tree Topology When Publishers Use the Normal Distribution

K	Total Packets	Total Hops	Average Hops	Max Distance Between an SSA-Node and Its Center	Average Distance Between an SSA-Node and Its Center
1	7657	16707	2.18	3	2
2	7651	14663	1.91	3	1.75
3	7636	14623	1.91	3	1.45
4	7660	12698	1.65	3	1.20
5	7650	9052	1.18	2	1.05
6	7621	9680	0.91	2	0.95
7	7615	9697	0.91	2	0.90
8	7579	6952	0.91	2	0.85
9	7632	6532	0.73	2	0.80
10	7630	5630	0.73	2	0.65
11	7619	6319	0.47	1	0.55
12	7638	1638	0.21	1	0.50

Table A.4: The Results of Applying the K-Median Placement Algorithm on the Mixed Topology When Publishers Use the Normal Distribution

K	Total Packets	Total Hops	Average Hops	Max Distance Between an SSA-Node and Its Center	Average Distance Between an SSA-Node and Its Center
1	1543	2670	1.73	4	2.20
2	1521	1252	0.82	2	1.36
3	1508	1136	0.75	2	1.27
4	1527	1150	0.75	2	1
5	1518	1096	0.72	2	0.72
6	1532	806	0.52	2	0.63

Table A.5: The Results of Applying the K-Median Placement Algorithm on the Tree Topology When Publishers Use the Uniform Distribution

K	Total Packets	Total Hops	Average Hops	Max Distance Between an SSA-Node and Its Center	Average Distance Between an SSA-Node and Its Center
1	6870	14182	2.06	3	2
2	6864	12158	1.77	3	1.75
3	6830	12085	1.76	3	1.45
4	6906	10245	1.48	3	1.20
5	6860	8172	1.19	3	1.10
6	6874	5338	0.77	2	0.95
7	6875	5307	0.77	2	0.90
8	6878	5294	0.76	2	0.85
9	6900	3361	0.48	2	0.80
10	6889	1345	0.19	2	0.70
11	6857	1278	0.18	2	0.65
12	6863	1283	0.18	2	0.50

Table A.6: The Results of Applying the K-Median Placement Algorithm on the Mixed Topology When Publishers Use the Uniform Distribution

K	Total Packets	Total Hops	Average Hops	Max Distance Between an SSA-Node and Its Center	Average Distance Between an SSA-Node and Its Center
1	1263	2428	1.92	4	2.27
2	1259	1637	1.30	4	2.18
3	1271	1427	1.12	3	1.36
4	1288	1090	0.84	2	1.18
5	1254	256	0.20	2	1.09
6	1285	268	0.20	2	0.81

Table A.7: The Results of Applying the Utilitarian Placement Algorithm on the Tree Topology When Publishers Use the Normal Distribution

K	Total Packets	Total Hops	Average Hops	Max Distance Between an SSA-Node and Its Center	Average Distance Between an SSA-Node and Its Center
1	7644	16650	2.17	3	2
2	7667	14738	1.92	3	1.75
3	7636	14620	1.91	3	1.45
4	7605	12548	1.64	3	1.20
5	7630	10583	1.38	3	1.10
6	7638	6996	0.91	2	0.95
7	7615	6930	0.91	2	0.80
8	7641	7008	0.91	2	0.85
9	7632	7002	0.91	2	0.70
10	7641	5020	0.65	2	0.60
11	7641	3641	0.47	1	0.55
12	7632	1632	0.21	1	0.50

Table A.8: The Results of Applying the Utilitarian Placement Algorithm on the Mixed Topology When Publishers Use the Normal Distribution

K	Total Packets	Total Hops	Average Hops	Max Distance Between an SSA-Node and Its Center	Average Distance Between an SSA-Node and Its Center
1	1509	2788	1.75	4	2.27
2	1531	1904	1.24	3	1.72
3	1511	1094	0.72	2	1.27
4	1517	1150	0.75	2	1
5	1523	1150	0.75	2	0.72
6	1541	794	0.51	2	0.63

Table A.9: The Results of Applying the Utilitarian Placement Algorithm on the Tree Topology When Publishers Use the Uniform Distribution

K	Total Packets	Total Hops	Average Hops	Max Distance Between an SSA-Node and Its Center	Average Distance Between an SSA-Node and Its Center
1	6900	14246	2.06	3	2
2	6869	12136	1.76	3	1.75
3	6915	12268	1.77	3	1.45
4	6872	10181	1.48	3	1.12
5	6851	8156	1.19	3	1.10
6	6897	5365	0.77	2	0.95
7	6866	5306	0.77	2	0.90
8	6882	5338	0.77	2	0.85
9	6882	5305	0.77	2	0.70
10	6930	3381	0.48	2	0.60
11	6904	3362	0.48	2	0.55
12	6855	1300	0.18	2	0.50

Table A.10: The Results of Applying the Utilitarian Placement Algorithm on the Mixed Topology When Publishers Use the Uniform Distribution

K	Max Distance	Average Distance
1	6	4.30
2	4	2.55
3	4	1.75
4	3	1.57
5	3	1.50
6	2	1.60
7	2	1.26
8	2	1.33

Table A.11: The Results of Applying a Random Placement on the Tree Topology

K	Max Distance	Average Distance
1	6	4.21
2	4	2.50
3	5	2.76
4	3	2.18
5	4	1.60
6	3	1.71
7	3	2
8	3	1.41
9	3	1.72
10	3	1.60
11	2	1.11
12	3	1.50

Table A.12: The Results of Applying a Random Placement on the Mixed Topology

Bibliography

[Ben96] J. Bentham. *The collected works of Jeremy Bentham: An introduction to the principles of morals and legislation*. Clarendon Press, 1996 (cit. on p. 40).

[Bho13] S. Bhowmik. "Distributed control algorithms for adapting publish/subscribe in software defined networks". MA thesis. 2013 (cit. on pp. 16, 24, 25).

[Bry12] W. Bryc. *The normal distribution: characterizations with applications*. Vol. 100. Springer Science & Business Media, 2012 (cit. on p. 41).

[BTBR17] S. Bhowmik, M. A. Tariq, A. Balogh, K. Rothermel. "Addressing TCAM limitations of software-defined networks for content-based routing". In: *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*. ACM. 2017, pp. 100–111 (cit. on p. 16).

[BTG+18] S. Bhowmik, M. A. Tariq, J. Grunert, D. Srinivasan, K. Rothermel. "Expressive Content-Based Routing in Software-Defined Networks". In: *IEEE Transactions on Parallel and Distributed Systems* (2018) (cit. on p. 16).

[BTGR16] S. Bhowmik, M. A. Tariq, J. Grunert, K. Rothermel. "Bandwidth-efficient content-based routing on software-defined networks". In: *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*. ACM. 2016, pp. 137–144 (cit. on p. 16).

[BTHR16] S. Bhowmik, M. A. Tariq, L. Hegazy, K. Rothermel. "Hybrid content-based routing using network and application layer filtering". In: *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*. IEEE. 2016, pp. 221–231 (cit. on pp. 16, 25, 29).

[BTK+15] S. Bhowmik, M. A. Tariq, B. Koldehofe, A. Kutzleb, K. Rothermel. "Distributed control plane for software-defined networks: A case study using event-based middleware". In: *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*. ACM. 2015, pp. 92–103 (cit. on p. 16).

[BTK+17] S. Bhowmik, M. A. Tariq, B. Koldehofe, F. Durr, T. Kohler, K. Rothermel. "High performance publish/subscribe middleware in software-defined networks". In: *IEEE/ACM Transactions on Networking (TON)* 25.3 (2017), pp. 1501–1516 (cit. on p. 16).

[CDF01] G. Cugola, E. Di Nitto, A. Fuggetta. "The JEDI event-based infrastructure and its application to the development of the OPSS WFMS". In: *IEEE transactions on Software Engineering* 27.9 (2001), pp. 827–850 (cit. on p. 15).

[Cis] Cisco. *OpFlex: An Open Policy Protocol White Paper*. URL: https://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731302.html (cit. on p. 22).

[CMT+11] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee. "DevoFlow: scaling flow management for high-performance networks". In: *ACM SIGCOMM Computer Communication Review*. Vol. 41. 4. ACM. 2011, pp. 254–265 (cit. on p. 16).

[CRW01] A. Carzaniga, D. S. Rosenblum, A. L. Wolf. "Design and evaluation of a wide-area event notification service". In: *ACM Transactions on Computer Systems (TOCS)* 19.3 (2001), pp. 332–383 (cit. on p. 15).

[EFGK03] P. T. Eugster, P. A. Felber, R. Guerraoui, A.-M. Kermarrec. "The many faces of publish/subscribe". In: *ACM computing surveys (CSUR)* 35.2 (2003), pp. 114–131 (cit. on p. 24).

[For09] L. Fortnow. "The status of the P versus NP problem". In: *Communications of the ACM* 52.9 (2009), pp. 78–86 (cit. on p. 26).

[Fou] O. N. Foundation. *OpenFlow*. URL: https://www.opennetworking.org/ (cit. on p. 22).

[Fun12] O. N. Fundation. "Software-defined networking: The new norm for networks". In: *ONF White Paper* 2 (2012), pp. 2–6 (cit. on p. 16).

[GCSO01] P. Gore, R. Cytron, D. Schmidt, C. O'Ryan. "Designing and optimizing a scalable CORBA notification service". In: *ACM SIGPLAN Notices*. Vol. 36. 8. ACM. 2001, pp. 196–204 (cit. on p. 15).

[Gra06] C. Grau. "There is noÏin"Robot": Robots and utilitarianism". In: *IEEE Intelligent Systems* 21.4 (2006), pp. 52–55 (cit. on p. 17).

[Gre] B. W. Gregor Hohpe. *Integration Patterns*. URL: https://www.enterpriseintegrationpatterns.com/ (cit. on p. 24).

[HBS+02] M. Hapner, R. Burridge, R. Sharma, J. Fialli, K. Stout. "Java Message Service Specification Version 1.1. Sun Microsystems". In: *Inc. o. Document Number* (2002) (cit. on p. 15).

[Hro13] J. Hromkovič. "Algorithmics for hard problems: introduction to combinatorial optimization, randomization, approximation, and heuristics". In: Springer Science & Business Media, 2013, pp. 431–433 (cit. on p. 27).

[HS85] D. S. Hochbaum, D. B. Shmoys. "A best possible heuristic for the k-center problem". In: *Mathematics of operations research* 10.2 (1985), pp. 180–184 (cit. on p. 30).

[HW04] G. Hohpe, B. Woolf. *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley Professional, 2004 (cit. on p. 24).

[Joh74] D. S. Johnson. "Approximation algorithms for combinatorial problems". In: *Journal of computer and system sciences* 9.3 (1974), pp. 256–278 (cit. on p. 26).

[JVV86] M. R. Jerrum, L. G. Valiant, V. V. Vazirani. "Random generation of combinatorial structures from a uniform distribution". In: *Theoretical Computer Science* 43 (1986), pp. 169–188 (cit. on p. 41).

[KARW16] N. Katta, O. Alipourfard, J. Rexford, D. Walker. "Cacheflow: Dependency-aware rule-caching for software-defined networks". In: *Proceedings of the Symposium on SDN Research*. ACM. 2016, p. 6 (cit. on p. 16).

[KDT13] B. Koldehofe, F. Dürr, M. A. Tariq. “Tutorial: event-based systems meet software-defined networking”. In: *Proceedings of the 7th ACM international conference on Distributed event-based systems*. ACM. 2013, pp. 271–280 (cit. on p. 25).

[KKKK11] S. Kontogiannis, A. Karakos, G. Kokkonis, P. Kitsos. “Snmp for Ethernet networks SETH: A network benchmark toolkit for managing routers statistical information”. In: *Informatics (PCI), 2011 15th Panhellenic Conference on*. IEEE. 2011, pp. 175–179 (cit. on p. 22).

[KLRW13] N. Kang, Z. Liu, J. Rexford, D. Walker. “Optimizing the one big switch abstraction in software-defined networks”. In: *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM. 2013, pp. 13–24 (cit. on p. 16).

[LS10] C. E. Leiserson, T. B. Schardl. “A work-efficient parallel breadth-first search algorithm (or how to cope with the nondeterminism of reducers)”. In: *Proceedings of the twenty-second annual ACM symposium on Parallelism in algorithms and architectures*. ACM. 2010, pp. 303–314 (cit. on p. 32).

[NKY+04] H. Nakano, T. Kita, H. Yokosawa, H. Akiyama, K. Nagamino. “Remote sensing course, beginning with Telnet and ending with network programming”. In: *Information Technology Based Higher Education and Training, 2004. ITHET 2004. Proceedings of the FIfth International Conference on*. IEEE. 2004, pp. 297–301 (cit. on p. 20).

[OW17] W. Odom, S. Wilkins. “CCENT ICND1 100-105 Official Cert Guide and Network Simulator Library”. In: Cisco Press, 2017, pp. 20–50 (cit. on p. 20).

[QYZ+18] K. Qiu, J. Yuan, J. Zhao, X. Wang, S. Secci, X. Fu. “Fast Lookup Is Not Enough: Towards Efficient and Scalable Flow Entry Updates for TCAM-Based OpenFlow Switches”. In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2018, pp. 918–928 (cit. on p. 20).

[Rot11] F. Rothlauf. “Design of modern heuristics: principles and application”. In: Springer Science & Business Media, 2011, pp. 7–44 (cit. on p. 26).

[SKBJ17] B. Sabarish, B. Kailassh, K. Baktha, Y. Janaki. “Recommendations of location for facilities using domination set theory”. In: *Communication and Signal Processing (ICCSP), 2017 International Conference on*. IEEE. 2017, pp. 1540–1544 (cit. on p. 30).

[TKBR14] M. A. Tariq, B. Koldehofe, S. Bhowmik, K. Rothermel. “PLEROMA: A SDN-based high performance publish/subscribe middleware”. In: *Proceedings of the 15th International Middleware Conference*. ACM. 2014, pp. 217–228 (cit. on pp. 16, 25).

[VSP17] R. Vinayakumar, K. Soman, P. Poornachandran. “Evaluating shallow and deep networks for secure shell (ssh) traffic analysis”. In: *Advances in Computing, Communications and Informatics (ICACCI), 2017 International Conference on*. IEEE. 2017, pp. 266–274 (cit. on p. 20).

[ZSL08] L. Zhai, L. Sun, Y. Liu. “Modeling and evaluation of high-performance publish-subscribe system”. In: *Computational Intelligence and Design, 2008. ISCID’08. International Symposium on*. Vol. 1. IEEE. 2008, pp. 457–460 (cit. on p. 15).

All links were last followed on October 7, 2018

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