Institut für Parallele und Verteilte Systeme

Abteilung Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Studienarbeit Nr. 2240

# Support for QoS in Publish/Subscribe systems by partitioning the event space
# Unterstützung von Dienstgüte in Publish/Subscribe Systemen durch Partitionierung des Ereignisraums

Daniel Biliniewicz

| | |
|---|---|
| Studiengang: | Informatik |
| Prüfer: | Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel |
| Betreuer: | MSc. Muhammad Adnan Tariq |
| begonnen am: | 2. November 2009 |
| beendet am: | 4. Mai 2010 |
| CR-Klassifikation: | C.2.1, C.2.4 |

## Abstract

In content-based publish/subscribe systems it is difficult to satisfy end-to-end delays for each subscriber in the presence of bandwidth constraints. To tackle this problem, we arrange the subscribers in a tree structure such that the individual latency requirements are fulfilled and for scalability divide the publishers into clusters. Here we address the clustering of publishers and to organize the clusters, we look at the advertisement and the degree constraint of each publisher. In this thesis we developed different clustering strategies, analysed them and verified our results experimentally.

# Contents

# 1 Introduction

In the context of information-driven applications like news services, monitoring systems or financial services, publish/subscribe is a popular and well-known communication paradigm. The problem addressed by a publish/subscribe system is the decoupling of producers of information, called publishers, from consumers of information, called subscribers.
Subscribers can specify their interest by the use of subscriptions and they will then, without knowing the source, receive all published notifications which match their subscription. That way a publish/subscribe system realizes a many to many communication. One publisher can provide notifications to multiple subscribers and subscribers can receive notifications from multiple publishers.
Publish/Subscribe systems can be divided into two types based on their expressiveness. For the type of publish/subscribe systems that are based on topics (also called subjects), notifications are categorized in a hierarchical manner. A notification could for example be categorized as *news.politics.germany*, if our hierarchy would be structured accordingly. The drawback is the expressiveness of the notification selection mechanism and that is why we here consider the other type of publish/subscribe systems which are content-based.
A content-based publish/subscribe system is the most expressive type of publish/subscribe systems, because it is possible to specify constraints on the notifications. In this model the notification are attribute-value pairs and subscriptions are expressed as conjunctions over ranges of attribute values.
If we want to provide QoS constraints in the sense of end-to-end delays between publishers and subscribers in a content-based publish/subscribe system it is very difficult due to the decoupling of publishers and subscribers. But there are a lot of applications like for example stock exchange, streaming services or online games where a certain end-to-end latency must not be exceeded for the clients. To address the problem the idea is to organize the subscribers in a spanning tree, where the notifications are forwarded starting a the root and to arrange the subscribers within this tree such that as many as possible of the individual latency requirements are fulfilled.
If only a single tree would be used then it would not be scalable, because a single root would represent a bottle neck. Also arranging all subscribers within a single tree would not scale, because there would be large sets of subscribers that have completely different subscriptions and the result would be a lot of false positives, where we define false positives as the unnecessary messages which are received by a node only for forwarding and the node is not interested in. So we either can't fulfill the latency constraints of the subscribers or would have to do a not scalable amount of local filtering.
Another approach might be to build a separate dissemination tree for each publisher,

but this also would not scale, because subscribers then need to be placed with respect to multiple publishers and there would be a lot of control overhead.
The idea here is to group publishers together into clusters such that subscribers only need to connect to one cluster for each subscription they posses and each clusters maintains a dissemination tree that respects the QoS requirements of the individual subscribers. Therefore we want to develop a clustering strategy that creates clusters which are scalable for publishers as well as subscribers.
The organization of the thesis is as follows. Chapter 2 deals with some related work settled in the field of publish/subscribe systems and clustering of subscriptions. In chapter 3 we specify our system model with special regard to the modelling of publishers and subscribers and we also relate to an existing solution for support of QoS constrains in publish/subscribe system. Furthermore we define the system model on which we build our clustering strategies. Chapter 4 deals with the developed clustering strategies and they are characterized and explained in detail. In chapter 5 the clustering strategies are evaluated with the help of an implementation that emerged from this thesis. Chapter 6 summarizes the results and gives an outlook on possible improvements.

# 2 Related Work

This chapter introduces some related work and here we look at representatives of publish/subscribes systems like SIENA and SemCast. We will also look at some existing clustering algorithms where in the end the clusters are mapped onto multicast groups and that way form a publish/subscribe system. In the following, we will describe the functionality and principles for each of these systems.

## 2.1 SIENA

In [1] an event notification service called SIENA is introduced and because the two main activities of a notification service are notification selection and notification delivery, it is designed to maximize expressiveness in the selection mechanism without losing scalability in the delivery mechanism in the context of a wide-area setting. This work is really very fundamental and can be regarded as basis for content-based publish/subscribe systems.
The service is intended to be best-effort and it does not provide means to specify delay constraints. SIENA extends the the common publish/subscribe protocol that consists only of the functions 'publish', 'subscribe' and 'unsubscribe', with an additional function called 'advertise', which a publisher can use to advertise the notifications it will create in the future. They also define the fundamental elements of a content-based publish/subscribe system, namely notifications and filters and how they interact. Notifications are sets of typed attribute and filters consist of constraints on the values of attributes. Here an attribute is a tuple $\alpha = (type_\alpha,\ name_\alpha,\ value_\alpha)$ and it matches an attribute constraint $\Upsilon = (type_\Upsilon,\ name_\Upsilon,\ operator_\Upsilon,\ value_\Upsilon)$, denoted by $\alpha \prec \Upsilon$, if and only if $type_\alpha = type_\Upsilon \wedge name_\alpha = name_\Upsilon \wedge operator_\Upsilon(value_\alpha,\ value_\Upsilon)$. If a filter is used in a subscription then multiple constraints on the same attribute are seen as a conjunction. Therefore they say that a notification $n$ matches a filter $f$, denoted by $n \prec_S^N f$, if and only if $\forall \Upsilon \in f :\ \exists \alpha \in n : \alpha \prec \Upsilon$. The statement notification $n$ matches filter $f$ is equivalent to the statement filter $f$ covers notification $n$.
Another important construct are advertisements, because they are intended to direct the propagation of subscriptions and are also specified as filters. The covering relation between an advertisement $a$ and a notification $n$ is defined as $n \prec_A^N a \Leftrightarrow \forall \alpha \in n : \exists \Upsilon \in a : \alpha \prec \Upsilon$. It follows that a filter $a$, which is interpreted as an advertisement, covers a filter $f$, which is interpreted as a subscription, if and only if $\exists n :\ n \prec_A^N\ \wedge\ n \prec_S^N f$.
For the architecture of the event notification service they in general consider three basic network topologies, more specific hierarchical, acyclic and general peer-to-peer struc-

tures and they also take a look at hybrid architectures.
To make the routing efficient they say that it is necessary to send notifications only towards servers that have subscribers that are interested in the notification and to avoid sending notifications into regions of the network where no appropriate subscriber is found. To achieve this they formulate the principles of downstream replication and upstream evaluation.
Downstream replication means to keep a notification during the routing in one piece as long as possible and only replicate it near to the subscribers, so if you think about a tree it should be very thin at the root and only get wide when reaching the leaves.
Upstream evaluation is related to filters and it says that the filters should be assembled as close as possible to the publishers.
For SIENA there are two different semantics, one is subscription based and the other is advertisement based.
In the subscription based variant no advertisements are used and the routing paths for notifications are set by subscriptions. To build these paths a subscription s is forwarded through the network until a node is reached that already has seen another subscription s' and s' cover s. The spanning tree that is built by the forwards of subscriptions is then used to route the notifications back to the subscribers along the reverse path.
For the advertisement based variant additionally to subscriptions advertisements are used and they determine the nodes where subscriptions are forwarded to. So subscriptions are only sent towards publishers that announced with their advertisement that they will generate notifications that match the subscription. Advertisements are propagated through the network, forming a spanning tree and subscriptions are only forwarded along the reverse paths and thereby they "activate" these paths, so notifications are only forwarded along activated paths.

## 2.2 Clustering Algorithms

In [3] a procedure to construct a limited number of multicast groups with the help of different clustering algorithms is presented. The focus here lies on the comparison of well created multicast groups in contrast to the use of broadcast and unicast.
They model a subscription as a set of aligned rectangles in the event space and its number of rectangles, i.e. the dimension, is equal to the number of attributes of the subscription. Subscribers are then modelled by sets of subscriptions.
The efficiency of their algorithms is based on some assumptions, namely that the subscribers are regional in the network topology, i.e. the subscribers are unevenly distributed, and if there are a lot of subscriptions in some area, it follows that there will also be a lot of notifications.
They divide the clustering algorithms into two categories, namely grid-based clustering and the other category only contains an algorithm called No-Loss. The grid-based algorithms have the disadvantage that it is possible that subscribers get messages they are not interested in, whereas for the No-Loss algorithm it is guaranteed that this situation doesn't arise.

The grid-based algorithms work with clustering heuristics that operate on cells of a regular grid in the event space. For these cells membership vectors over subscribers are defined, i.e. the vectors state which subscribers are included in the cell, and they use these vectors together with a distance function for the "expected waste", which is defined, to group the cells into clusters such that the expected waste is minimized. Then they introduce several grid-based clustering algorithms like K-Means, pairwise grouping or minimum spanning tree clustering. In the end the subscribers in each cluster are mapped to a multicast group.
They also sketch how the matching for both categories of clustering algorithms can be done in real time with regard to the different data structures the algorithms produce.
The main problem here is that the multicast groups are precomputed and depend on the set of subscriptions, so as presented here it is not usable in a dynamic system, because the subscribers have to be known in advance.

## 2.3 SemCast

SemCast [2] is an overlay network based system that implements a publish/subscribe system by using channels. Each channel represents an individual dissemination tree and there are in principle two type of trees, low-cost trees and delay-bounded trees. For the low-cost trees the idea is to have a minimal number of inner nodes within the tree to reduce bandwidth and the delay-bounded trees offer the possibility to subscribers to specify latency requirements.
The focus here lies on the creation of channels concerning their number and also which messages are forwarded through them, i.e. for which content they are responsible. SemCast only requires filtering at so-called source and gateway brokers, which represent the predecessors of the root of a dissemination tree and the leaves of a dissemination tree. Within the separate channels there is no filtering and the idea behind this is to reduce the filter overhead for the interior nodes.
The source brokers are the entry point for the publishers, so publishers connect to them and each of the source brokers is connected to a certain number of roots of dissemination trees representing a channel. When a message arrives at a source broker it determines the set of channels where the message needs to be forwarded by looking at an expression that is linked with each channel. Thereby the expressions determine the content of the channel and an expression is basically a combination of conjunctions and disjunctions of profiles. Here the term "profile" is just the name for their model of subscriptions. When a message reaches the end of a channel it is again filtered. Subscribers connect to the end of a channel via gateway brokers and these are the nodes that perform the mentioned filtering.
To create the channels a channelization algorithm is provided which mainly collects informations about the covering relation between different profiles with the support of statistical data and calculates a division into channels out of it. For the statistical data at each source broker for every message the set of matching subscriptions is stored in a sliding window manner.

By collecting this data an overlapping matrix for each pair of subscriptions is created and with the additional help of a cost function the structure and amount of channels is determined. The hereby start with containment hierarchies of subscriptions and then merge different overlapping roots of hierarchies. To decide whether to merge or not for both resulting situations the costs are calculated. After this computation process the channels are reorganized according to the result and all this is done periodically.
The problem is that the information has to be collected at a so-called coordinator that performs the calculations and informs the nodes to build the channels as calculated. Hereby a central component is introduced and even though they claim the central coordinator can be replaced by a distributed solution, in the end one node must be elected to perform the calculations and they also admit that.

# 3 System Model und Problem Description

## 3.1 System Model

For the system model we stick to the model presented in [5] which means that we have a content-based publish/subscribe model for the communication between peers and a distributed system with an unbounded and dynamic set of peers. In general peers can arbitrarily join and leave the system. Each peer has got a unique identity, so it can establish a logical point-to-point connection to form an overlay network. We also assume that these logical links are maintained and created by a probabilistic membership service as described in [5] and [4], thus a partitioning of the network is not assumed to happen. On top of the membership service peers are forming a spanning tree in which all are connected, so they can communicate with each other.
There are two types of peers in the system, namely publishers and subscribers. Publishers represent the source of information and they do so by creating notifications according to their attribute set. Subscribers are interested in only a certain part of all these created messages and in the best case receive only the messages they are interested in, i.e. the ones they subscribed for. As it is usual for a content-based publish/subscribe model, an event notification consists of a set of (typed) attributes and a value for each attribute contained in the notification. Here we assume a set $\Omega = \{A_1,\ A_2,\ A_3, ...\}$ representing all attributes in the system and a notification can be represented as $n = \{A_1 = v_1,\ A_2 = v_2,\ ...\}$.
From this attribute set $\Omega$, every joining publisher first chooses a set of attributes $\Psi_p \subseteq \Omega$. This set $\Psi_p$ is the advertisement of publisher p. From the subscribers point of view, the set of notifications that publisher p can publish is $PS^{-\emptyset}_{Adv}(p) = 2^{\Psi_p} - \{\emptyset\}$ and as we can see, the set $PS^{-\emptyset}_{Adv}(p)$ consists of all possible attribute combinations of $\Psi_p$. If publisher p sends a notification it always contains all attributes in $\Psi_p$. So for example $\Omega = \{A,\ B,\ C,\ D\}$ and $\Psi_p = \{A,\ C,\ D\}$, then publisher p would send one notification with attributes A, B, C and D and for a notification containing these attributes we will just write ABCD.
For the sake of simplicity, subscribers in the system only choose one attribute combination from $\Theta := \bigcup_{p \in P_{sys}} PS^{-\emptyset}_{Adv}(p)$, where $P_{Sys}$ denotes the set of all publisher currently in the system. So a subscriber can only subscribe for an attribute combination for which a publisher already exists. We in general denote with $\Phi_s \subseteq \Theta$ the set of subscriptions of subscriber s and because we will only consider the case where $|\Phi_s| = 1$, we don't

distinguish between a set of subscriptions and a single subscription and therefore also refer to $\Phi_s$ as the subscription of subscriber s. What concerns the subscribers and the notifications they receive, we will here not care about the actual values of the attributes contained in a notification, because our clustering strategies build on an existing algorithm for the construction of dissemination trees as presented in [5] and since filtering regarding the values of the attributes is already done within these trees, for us it is only important to consider the absence of an attribute from $\Omega$. So a notification $n$ will be considered as a false positive for a subscriber $s$ with subscription $\Phi_s$ if and only if $\Gamma_n$ does not contain all attributes of the attribute combination in $\Phi_s$, where $\Gamma_n$ denotes the set of attributes for notification $n$. A possible subscription would be $\Phi_s = \{AB\}$ and the meaning is that subscriber s is only interested in notifications containing the attributes A and B. Of course it is possible that a notification for s contains more than these two attributes according to the definition in [1], so a notification containing attributes A,B and D would also match the subscription. Especially if a publisher p with $\Psi_p = \{A,\ C,\ D\}$ would publish a message, it would contain the attribute combination ABCD and hence be accepted by subscriber s.

## 3.2 Problem Description

In general we want to provide a publish/subscribe system that can guarantee a QoS bound with a probabilistic reliability and to derive this bound we first look at the behaviour of the underlay network and then give guarantees according to what was seen. For this problem there is already a paper [5] which addresses this issue and for a great extent of this description we will stick to it, but nevertheless extent what was developed so far. An advantage of this probabilistic reliability is that it is possible to overstep the QoS bound to a certain extent without violating the agreement with the user, e.g. we can say that a notification will be delivered within 100ms with a probability of 95% and for 5% of the messages the user has to deal with a latency greater than 100ms, but he knows about it. Of course the reliability of the given QoS bound depends on the values chosen for the bound and its probability distribution.
For the special case that the publishers can be partitioned into disjoint groups $G_p := \{p_1, p_2, ...\}$ with $\forall p_i, p_j \in G_p : \Psi_{p_i} \subseteq \Psi_{p_j} \vee \Psi_{p_j} \subseteq \Psi_{p_j}$, so 2 publishers are either completely overlapping or disjoint, there is already an algorithm to provide these bounds and maximize the number of subscribers whose latency specifications can be satisfied without violating their traffic constraints, i.e. their bandwidth. If we assume that a subscriber has only 1 subscription as we said in the previous chapter 3.1 (and the paper also does), a subscriber only needs to be connected to a single group of publishers, because the notifications it is interested in are only generated and spread there.
The mentioned algorithm works for this case and creates a dissemination tree for each pair of publisher group and associated subscribers. This trees are dynamically adapted and optimized while the system is running to make sure that the given bounds in latency are fulfilled if it is possible.

### 3.2.1 Multi-Publisher Problem

We want to generalize this approach and remove the restriction on the publishers to get a generic publish/subscribe system. Now it is possible that a subscription can be satisfied by more than one group of publishers, because the advertisements $\Psi_p$ of different publishers may overlap, i.e. $\exists p_i, p_j : \Psi_{p_i} \cap \Psi_{p_j} \neq \emptyset \wedge \Psi_{p_i} \not\subseteq \Psi_{p_j} \wedge \Psi_{p_j} \not\subseteq \Psi_{p_i}$. We refere to this problem as the multi-publisher problem.

#### Clustering as Solution

To tackle the problem the idea is to group the publishers into clusters such that we can apply the existing algorithm from [5] for every single cluster. So it is necessary to develop a distributed algorithm which does an appropriate clustering of the publishers. After having built the clusters, each publisher is connected to all clusters $C_i$ where the intersection of $PS^{-\emptyset}_{Adv}(p)$ with the set of attribute combinations of cluster $C_i$ is not empty. In the following we denote a cluster of publishers with C and its set of attribute combinations that are published in cluster C as $CS_C$. Let for example for a cluster $C_1$ be $CS_{C_1} = \{A,\ B,\ AB\}$, then subscribers with a subscription for A,B or AB will connect to this clusters, e.g. if there is a subscriber $s_1$ with $\Phi_{s_1} = \{B\}$ it would connect to $C_1$. Furthermore we want this attribute combination to occur only in at most one cluster, because then the subscribers also have to connect to only one cluster and that way we can preserve scalability and use the already existing algorithm. For the clustering algorithm we will look at the intersection of $PS^{-\emptyset}_{Adv}$ of all publishers and more specific the clustering strategies will calculate the intersection of $PS^{-\emptyset}_{Adv}(p)$ of a joining publisher p and $CS_{C_i}$ of all already existing clusters. With this information we will then decide if we will restructure some clusters or build a new cluster and so on, so this will be the basis for our strategy.
To achieve this goal we will preserve an invariant, which is as follows:
Let M:=$\{C_1, C_2, ...\}$ be the set of all currently existing clusters in the system. Then our clustering algorithm has to preserve for every point in time the invariant that the attribute sets of all clusters $C_i \in M$ are disjoint, what means:

$$\forall C_i, C_j \in M : i \neq j \Rightarrow CS_{C_i} \cap CS_{C_j} = \emptyset \tag{3.1}$$

#### Solution in the paper

The paper also describes an algorithm for the clustering and which cases might be distinguished in terms of intersections between publishers and clusters. But it addresses a problem that arises, namely the publishers would have to connect to too many clusters, if every publisher p chooses its set of attributes $\Psi_p$ uniform at random from

$\Omega$, because the algorithm presented would create too many clusters, each only serving a small fraction of $PS^{-\emptyset}_{Adv}(p)$ for publisher p. On the other hand the false positives rate is considered to be small.
Another approach was to generalize the advertisements of a publisher p if the number of clusters p has to connect to would be too large, so clusters can be merged. But the problem here is the resulting false positives rate.

### Development of Strategies

Our effort here will be to think about different clustering strategies, see how the system behaves and try to improve them to deal with the problem of scalability in terms of the number of clusters a publisher has to connect to and also the upcoming false positives rates for the subscribers. In the end we would like to have an "optimal" strategy to build clusters in a way that we minimize the false positives rates and also preserve scalability for the publishers, i.e. the number of clusters stays within a limit we can deal with. Once we got the clustering we can then use the algorithm in [5] for each cluster and group of connected subscribers to additionally preserve latency bounds. As we tried to sketch, the clusters are in fact just groups of publishers, but clearly a publisher can be in more than one cluster. In the next chapters these strategies are presented and analysed.

# 4 Cluster Building Strategies

In this chapter we will introduce the developed clustering strategies and for each strategy we will explain the algorithm and give some examples to show how it works. For strategy 1 and strategy 2, we also provide a formula for the maximum number of clusters and explain how we derived it.

## 4.1 Strategy 1

This strategy is a very simple approach to build a system with a false positive rate of 0, for every cluster or subscriber respectively. It ensures, together with the assumption that a publisher only publishes notifications containing all attributes a publisher has, that publishers don't send any unwanted notifications to the clusters they are connected to.

**Algorithm 1** Strategy 1

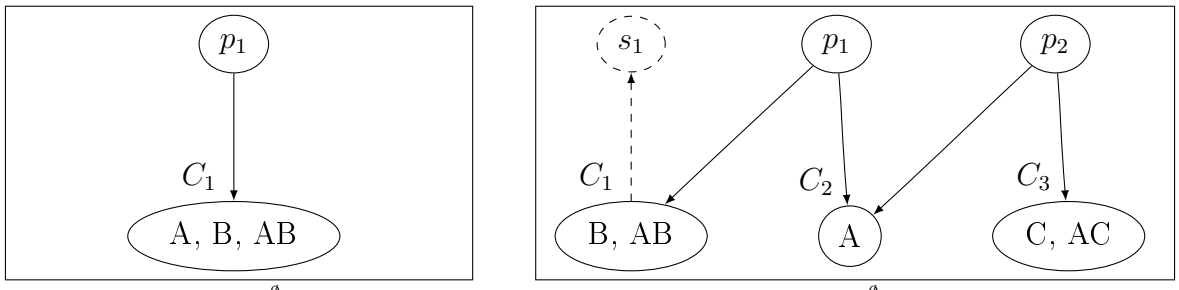
**Require:** A new publisher p enters the System

1: S := $PS_{Adv}^{-\emptyset}(p)$
2: **for all** $CS_{C_i}$ **do**
3: **if** $S \cap CS_{C_i} \neq \emptyset$ **then**
4: **if** $CS_{C_i} \subseteq S$ **then**
5: join $C_i$
6: S := S - $CS_{C_i}$
7: **else**
8: create new cluster $C_{new}$ with $CS_{C_{new}} = S \cap CS_{C_i}$
9: S := S - $CS_{C_i}$
10: **end if**
11: **if** S = $\emptyset$ **then**
12: break
13: **end if**
14: **end if**
15: **end for**
16: **if** $S \neq \emptyset$ **then**
17: create new cluster $C_{new}$ with $CS_{C_{new}} = S$
18: **end if**

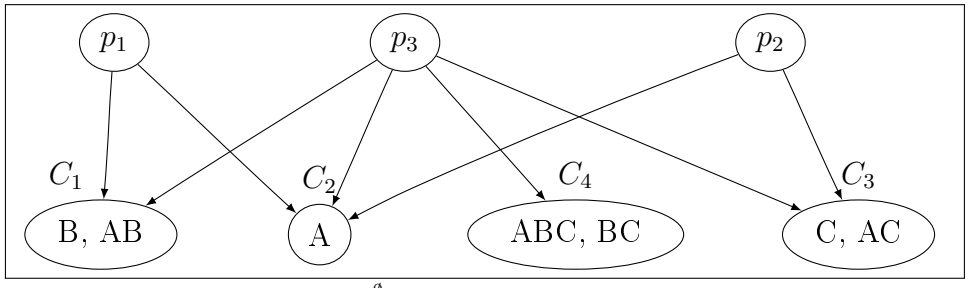
### 4.1.1 Algorithm Description

The strategy works as follows (cf. Algorithm 1):
Whenever a new publisher p enters the system, it calculates the overlap with all clusters that exist at this point and if there is an overlap with a cluster $C_i$ such that not all attribute combinations $CS_{C_i}$ of the cluster are in the intersection, the attribute combinations of the intersection $CS_{C_i} \cap PS^{-\emptyset}_{Adv}(p)$ are taken out of $CS_{C_i}$, a new cluster with attribute combinations $CS_{C_i} \cap PS^{-\emptyset}_{Adv}(p)$ is created and the publisher joins this new created cluster. The second case for an intersection is that all attribute combinations of the cluster are contained in $PS^{-\emptyset}_{Adv}(p)$. In this case the publisher simply joins the cluster for this attribute combinations. After having calculated the intersection with all existing clusters, there might be some attribute combinations in $PS^{-\emptyset}_{Adv}(p)$ for which there is no cluster yet. So in the last step the publisher creates a new cluster with these leftover attribut combinations if they exist and joins this new created cluster. In the algorithm this case arises if the set S isn't empty after having gone through all existing clusters.

### 4.1.2 Example




(a) publisher with $PS^{-\emptyset}_{Adv}(p_1) = \{A,\ B,\ AB\}$ (b) publisher with $PS^{-\emptyset}_{Adv}(p_2) = \{A,\ C,\ AC\}$




(c) publisher with $PS^{-\emptyset}_{Adv}(p_3) = \{A,\ B,\ C,\ AB,\ AC,\ BC,\ ABC\}$

Figure 4.1: publishers $p_1$, $p_2$ and $p_3$ using strategy 1

The example in figure 4.1 shows how the system evolves if all publishers joining use the algorithm for strategy 1 (cf. Algorithm 1). Here we consider 3 publisher $p_1$, $p_2$ and $p_3$ that join the system one after the other. In figure (a) publisher $p_1$ is the first node

that joins and $p_1$ has chosen to publish notifications containing attributes A and B, so $PS_{Adv}^{-\emptyset}(p_1) = \{A,\ B,\ AB\}$. There are no clusters yet and therefore there are no intersections with already existing clusters. The result is one cluster in which notifications with all attribute combinations in $PS_{Adv}^{-\emptyset}(p_1)$ are published. When publisher $p_1$ decides to publish a notification it creates a message containing all attributes, i.e. a notification containing AB.
In figure (b) another publisher $p_2$ joins with $PS_{Adv}^{-\emptyset}(p_2) = \{A,\ C,\ AC\}$. When it executes the algorithm for strategy 1, there is an intersection with the attribute combination set $CS_{C_1}$ of cluster $C_1$ which is non empty, because $PS_{Adv}^{-\emptyset}(p_2) \cap CS_{C_1} = \{A\}$. According to the strategy a new cluster $C_2$ is created with $CS_{C_2} = \{A\}$, $p_2$ joins this cluster and also $p_1$ has to connect to $C_2$ because A is an element of $PS_{Adv}^{-\emptyset}(p_1)$. Note that the creation of cluster $C_2$ here prevents the system from false positives, because if $p_2$ would just join cluster $C_1$ and let $CS_{C_1}$ unchanged a subscriber $s_1$ interested only in attribute B, i.e. $\Phi_{s_1} = \{B\}$ would connect to cluster $C_1$ and recognize all messages published by $p_2$ as false positives, because the messages would only contain the attribute combination AC. Besides $C_1$ there is no other already existing cluster, but $p_2$ still has the attribute combinations C and AC which don't occur in any cluster yet. So $p_2$ creates another cluster $C_3$ with $CS_{C_3} = \{C,\ AC\}$ for the leftover attribute combinations and joins $C_3$.
In figure (c) publisher $p_3$ joins with $PS_{Adv}^{-\emptyset}(p_3) = \{A,\ B,\ C,\ AB,\ AC,\ BC,\ ABC\}$. It again calculates the intersection with all existing clusters and $PS_{Adv}^{-\emptyset}(p_3)$ has an intersection with every cluster $C_1$, $C_2$ and $C_3$. For all 3 intersection it is the case that $CS_{C_i} \subseteq PS_{Adv}^{-\emptyset}(p_3)\ , i \in \{1,\ 2,\ 3\}$ , so $p_3$ just connects to these $C_i$ and there is no danger of false positives for subscribers that might connect to one of these clusters. Publisher $p_3$ also creates a new Cluster $C_4$ with attribute combinations $CS_{C_4} = \{BC,\ ABC\}$, because there are no clusters which contain these attribute combinations.
When we look at the result then we can see that our invariant (3.1) is still fulfilled, so every attribute occurs in at most 1 cluster and furthermore there will be no false positives when subscribers connect to clusters where notifications are published they are interested in, because we split the clusters or more precisely we take the attribute out of a cluster if there is only a partial overlap with a new publisher. What we can also recognize is that we only split and never merge and therefore the number of clusters a publisher has to connect to grows (see 4.1.3).

### 4.1.3 Analysis

By analysing the algorithm an upper bound for the maximum number of clusters for the system could be found. If there are **n** attributes in the system and every publisher chooses **m** attributes out of them, the maximum number of clusters $c_{max_1}$ is

$$c_{max_1} = \begin{cases} 2^n - \sum_{i=m+1}^{n} \binom{n}{i} & ,\ m < n \\ 1 & ,\ m = n \end{cases} \tag{4.1}$$

The explanation is as follows. In general the strategy tends to build a separate cluster for every possible combination of attributes and if there is a publisher for each of the $\binom{n}{m}$ possible combinations, this worst case really shows up. So the question is how many clusters containing one attribute combination can be created.

As we know from set theory, for a set with n elements there are $2^n$ subsets. But because publishers only choose m elements, we have to subtract the number of subsets with more than m elements, e.g. for m+1 elements there are $\binom{n}{m+1}$ different subsets and this leads to formula (4.1). The case were m=n follows from the algorithm, because there is no partial overlap and we get exactly one cluster.

### 4.1.4 Summary

What can be seen is that the maximum number of clusters and i.e. the maximum number of connections per publisher depends on the number of attributes each publisher chooses and in the worst case it is $2^n$. Clearly this is not acceptable, although we have a guaranteed false positives rate of 0, but the number of connections per publisher grows exponential what is not scalable.

## 4.2 Strategy 2

After having seen the results of strategy 1, the next idea was to think about a strategy which doesn't have an exponential growth in the number of attributes a publisher chooses because it doesn't scale. So in contrast to the first strategy, merging of clusters is in the foreground here. In strategy 1, clusters were only split and never merged. In fact in this strategy, a publisher either joins a cluster or merges existing clusters. New clusters are only created if some attributes (or attribute combinations) of the joining publisher never occurred before.

### 4.2.1 Algorithm Description

As you can see in the pseudo code (cf. Algorithm 2) a new publisher p joining calculates the overlap with all clusters currently in the system. If there is an overlap, we distinguish 2 cases. In the first case the set of attribute combinations of an existing cluster is included in $PS^{-\emptyset}_{Adv}(p)$, so we remember this cluster to get merged by using the set *clustersToMerge*. In the other case we only have a partial overlap and so the publisher just joins this cluster and removes the overlapping attribute combinations from the (temporary) set $S$, which contains the attribute combinations we don't have a cluster for yet. For the case where an existing cluster is included in $PS^{-\emptyset}_{Adv}(p)$, the attribute combinations aren't taken out of S, because we will create a new cluster in the last step. If the set $S$ isn't empty we create a cluster $C_{new}$ with $CS_{C_{new}} = S$ and then

we merge all clusters we remembered in the set *clustersToMerge* with $C_{new}$. In practice this means informing all publishers of all clusters in the set *clustersToMerge* to migrate to $C_{new}$, because the other clusters are removed.

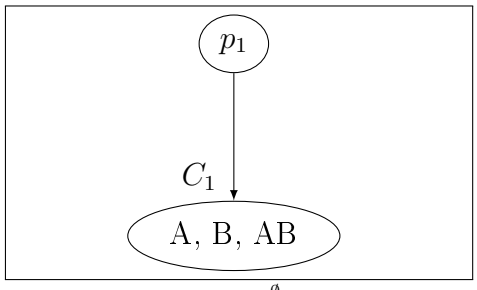
**Algorithm 2** Strategy 2

**Require:** A new publisher p enters the System

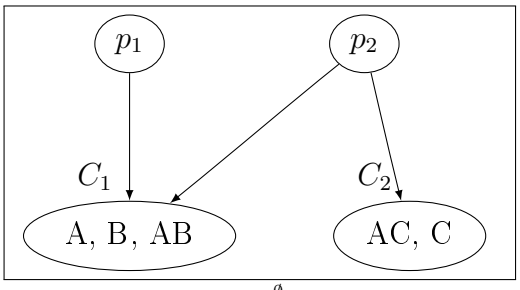
```
1:  S := PS^{-∅}_{Adv}(p)
2:  clustersToMerge := ∅
3:  for all CS_{C_i} do
4:    if S ∩ CS_{C_i} ≠ ∅ then
5:      if CS_{C_i} ⊆ S then
6:        clustersToMerge := clustersToMerge ∪ {C_i}
7:      else
8:        join C_i
9:        S := S - CS_{C_i}
10:     end if
11:     if S = ∅ then
12:       break
13:     end if
14:   end if
15: end for
16: if S ≠ ∅ then
17:   create new cluster C_new with CS_{C_new} = S
18:   merge all clusters in clustersToMerge with C_new
19: end if
```

### 4.2.2 Example

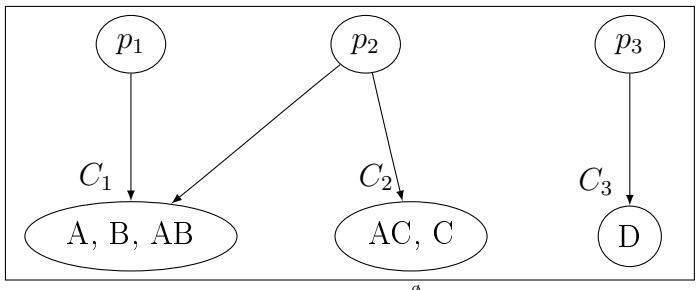
The example in 4.2 shows how the system behaves if the publishers joining use strategy 2 (cf. 4.2.1). We take the publishers $p_1$, $p_2$, $p_3$ and $p_4$ and let them one after the other join the system. In figure (a) publisher $p_1$ is the first node that joins and so there is no intersection of $PS^{-\emptyset}_{Adv}(p_1) = \{A,\ B,\ AB\}$ with an already existing cluster. This implies that $p_1$ creates a new cluster $C_1$ with $CS_{C_1} = \{A,\ B,\ AB\}$ and $p_1$ joins this cluster.
In figure (b) publisher $p_2$ joins, which has chosen $\Psi_p = \{A,\ C\}$ as set of attributes his notifications will contain and therefore $PS^{-\emptyset}_{Adv}(p_2) = \{A,\ C,\ AC\}$. When it calculates the intersections with all already existing clusters there is only one with the attribute set $CS_{C_1}$ of cluster $C_1$ because $PS^{-\emptyset}_{Adv}(p_2) = \{A,\ C,\ AC\} \cap CS_{C_1} = \{A\}$. Following the algorithm $p_2$ just joins cluster $C_1$ and doesn't take the attribute combination in the intersection out of $C_1$ as it would be the case for strategy 1. This is one reason for the false positives that are now present in contrast to the first strategy. If we imagine a subscriber $s_1$ with $\Phi_{s_1} = \{B\}$ then it would receive notifications it isn't interested in,



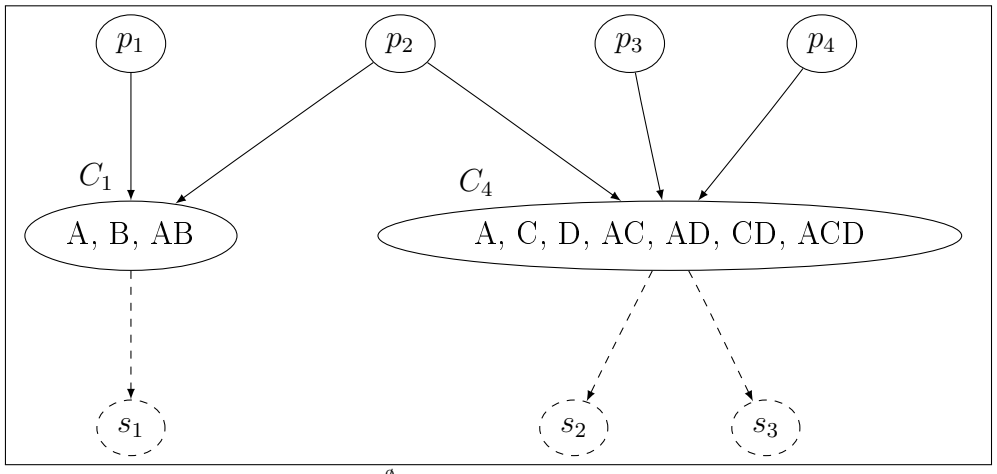

(a) publisher with $PS^{-\emptyset}_{Adv}(p_1) = \{A,\ B,\ AB\}$




(b) publisher with $PS^{-\emptyset}_{Adv}(p_2) = \{A,\ C,\ AC\}$




(c) publisher with $PS^{-\emptyset}_{Adv}(p_3) = \{D\}$




(d) publisher with $PS^{-\emptyset}_{Adv}(p_4) = \{A,\ C,\ D,\ AC,\ AD,\ CD,\ ACD\}$

Figure 4.2: publishers $p_1$, $p_2$, $p_3$ and $p_4$ using strategy 2

because publisher $p_2$ publishes messages containing the attribute combination AC, i.e. the notifications contain only A and C, and therefore every notification published by $p_2$ would be a false positive for subscriber $s_1$.
In figure (c) publisher $p_3$ with $PS^{-\emptyset}_{Adv}(p_3) = \{D\}$ joins and as it was the case for $p_1$, there is no intersection with any already existing cluster, so a new cluster $c_3$ is created with $CS_{C_3} = \{D\}$ and $p_3$ joins this cluster. We let $p_3$ join to show a characteristic behaviour of the strategy in the next figure.
In figure (d) publisher $p_4$ joins the system with $PS^{-\emptyset}_{Adv}(p_4) = \{A,\ C,\ D,\ AC,\ AD,\ CD,\ ACD\}$ and as it goes through the clusters

and calculates the intersection with $CS_{C_i}$ , $i \in \{1,2,3\}$, there are non-empty intersections with all 3 existing clusters. For cluster $C_1$ we have $PS^{-\emptyset}_{Adv}(p_4) \cap CS_{C_1} = \{A\}$ and because $CS_{C_1}$ is not completely included in $PS^{-\emptyset}_{Adv}(p_4)$, publisher $p_4$ joins $C_1$. For clusters $C_2$ and $C_3$ we also got a non-empty intersection, but this time both attribute combination sets $CS_{C_2}$ and $CS_{C_3}$ are included in $PS^{-\emptyset}_{Adv}(p_4)$, e.g. $PS^{-\emptyset}_{Adv}(p_4) \cap CS_{C_2} = \{C,\ AC\} = CS_{C_2}$. In the algorithm we would just remember these clusters on a list to get merged in the last step, but here these were the last ones we looked at and according to the algorithm $p_4$ creates a new cluster $C_4$ in the next step, with $CS'_{C_4} = \{A,\ AD,\ CD,\ ACD\} = PS^{-\emptyset}_{Adv}(p_4) \setminus (CS_{C_2} \cup CS_{C_3})$ and then $C_2$ and $C_3$ are merged with $C_4$. The final attribute combination set is $CS_{C_4} = \{A,\ C,\ D,\ AD,\ AC,\ CD,\ ACD\}$.
If we now imagine a subscriber $s_2$ , which is interested in attributes A and C, i.e. $\Phi_{s_2} = \{AC\}$, it would connect to cluster $C_4$. The messages published by $p_2$ and $p_4$ match the subscription of $s_2$, but the problem is publisher $p_3$ that is also connected to cluster $C_4$. The messages he publishes in cluster $C_4$ contain only the attribute D, because $PS^{-\emptyset}_{Adv}(p_4) = \{D\}$ and are therefore false positives for subscriber $s_2$. This case only arises because we merged clusters $C_2$ and $C_3$ with $C_4$, but we do so to keep the number of connections a publisher has to establish with clusters low. If $s_2$ would choose $\Phi_{s_2} = \{D\}$ instead, we would have the analogous problem what concerns false positives, but this time with publisher $p_2$ instead. Also a subscriber $s_3$ with $\Phi_{s_3} = \{ACD\}$ would experience false positives and more over quite a high rate of them, because every message from $p_2$ and $p_3$ is a false positive for $s_3$. Publisher $p_4$ is the only node that publishes messages $s_3$ is interested in. In fact $s_3$ should get the highest false positives rate here, if we assume that the message from all publishers are evenly distributed and the explanation is that $s_3$ has the most selective subscription with regard to the set of attributes of cluster $C_4$.
If we would add another publisher $p_5$ with $\Psi_{p_5} = \{A,\ B,\ C,\ D\}$ then its set $PS^{-\emptyset}_{Adv}(p_4)$ would have an overlap with all existing clusters and additionally every $CS_{C_i}, i \in \{1,2\}$ would be included in $PS^{-\emptyset}_{Adv}(p_4)$. The result would be one big cluster where all publishers are connected to, because we only merge and never split clusters.

### 4.2.3 Analysis

The system that is created by using this strategy now tends to build one big cluster and is the other extreme to strategy 1. As we shall see by analysing the algorithm one big cluster only occurs if there is a publisher choosing all attributes in the system as his attribute set. By looking a little bit more into the details again an upper bound for the maximum number of clusters could be derived. If we have **n** attributes for the whole system, i.e. $|\Omega| = n$ and each publisher chooses **m** out of these attributes the maximum number of clusters $c_{max_2}$ is

$$c_{max_2} = \binom{n}{m} \tag{4.2}$$

The explanation is as follows:
A new cluster is only created if the attribute combination didn't occur before and for this case an attribute combination consisting of $m$ single attributes can be found in the new created cluster. So we can use this combinations as markers and if another publisher with the same attribute combination consisting of $m$ attributes joins the system it is not a problem because it won't create a new cluster. So this leads to formula (4.2).

### 4.2.4 Summary

What wasn't considered so far is the fact, that we don't have a false positives rate of 0 as this is the case for strategy 1. As we shall see in the evaluation part the false positives rate is really very high and that is the problem of this strategy because there are so to say not enough clusters to get a low false positives rate. On the other hand the number of connections per publisher is very low especially when compared to strategy 1. So the next idea was to find a strategy that is between these 2 extremes, what leads to strategy 3.

## 4.3 Strategy 3

With the results of strategy 1 and strategy 2 the main focus here is to develop a strategy that is between these 2 extremes. An important step here is that the system model was changed what means that one aspect was added which hasn't been considered for strategy 1 and strategy 2. The publishers now have a limited number of connections they can establish with the clusters. We still want to maintain the invariant that every attribute combination occurs at most once in one cluster, so a subscriber interested in only one attribute combination has to connect to exactly one cluster. This constraint in the number of connections per publisher also imposes a constraint on the way how the attribute combinations in the system can be divided into clusters. So as we shall see the number of connections per publisher has a heavy influence to which of the former 2 strategies the system will tend and therefore it influences the number of false positives, too.

### 4.3.1 Algorithm Description

The algorithm to achieve the before mentioned goal works as follows (cf. Algorithm 3). Whenever a new publisher p joins the system it calculates the intersection between S (attributes of $PS_{Adv}^{-\emptyset}(p)$ for which no cluster is found yet) and $CS_{C_i}$ (attribute set for cluster $C_i$) for every cluster $C_i$ currently in the system.
One important thing is the order in which the existing clusters are seen by the algorithm. Here we tested 2 possibilities, one is random order and the other is sorting

**Algorithm 3** Strategy 3

**Require:** A new publisher p enters the System

S := $PS_{Adv}^{-\emptyset}(p)$
clustersIncluded := $\emptyset$
clustersOverlapping := $\emptyset$
shuffle the list of existing clusters
OR
sort clusters by the number of attribute combinations in descending order
**for all** $CS_{C_i}$ **do**
  **if** $S \cap CS_{C_i} \neq \emptyset$ **then**
    **if** $CS_{C_i} \subseteq S$ **then**
      clustersIncluded := clustersIncluded $\cup\{C_i\}$
    **else**
      { partial overlap }
      **if** p.numberOfFreeConnections $> 1$ **then**
        **if** min($C_i$.numberOfFreeConnectionsOfAllPublishers) $> 0$ **then**
          create $C_{new}$ with $CS_{C_{new}} = S \cap CS_{C_i}$
          S:= S - ($S \cap CS_{C_i}$)
        **else**
          join $C_i$
          S := S - ($S \cap CS_{C_i}$)
        **end if**
      **else**
        clustersOverlapping := clustersOverlapping $\cup$ $\{C_i\}$
      **end if**
    **end if**
  **end if**
**end for**
**while** ( $\neg$ isEmpty(clustersIncluded) $\wedge$ p.numberOfFreeConnections $> 1$ ) **do**
  join $C_i$
  clustersIncluded := clustersIncluded $\setminus\{C_i\}$
  S := S - ($S \cap CS_{C_i}$)
**end while**
**if** $S \neq \emptyset$ **then**
  create cluster $C_{new}$ with $CS_{C_{new}}$=S
  merge Clusters in clustersIncluded and clustersOverlapping with $C_{new}$
**end if**

the clusters in descending order according to the number of attribute combinations they contain, so the biggest clusters are seen first and we refer to it as the descending variant (cf. Algorithm 3, line 4).
If the intersection isn't empty 2 cases are in general distinguished.
If $CS_{C_i}$ is a subset of S then cluster $C_i$ is added to a set called "clustersIncluded" and hence remembered for a later step. The idea behind this is to first look at the clusters that are partial overlapping with S (cf. Algorithm 3, line 7).
So in this (second) case we try to apply strategy 1 but we have to consider the constraint in the number of free connections publisher p has. If there is a partial overlap we therefore first look at the free connections of the new publisher p and if it got more than one left, we try to take the attribute combinations contained in $S \cap CS_{C_i}$ out of cluster $C_i$ (cf. Algorithm 3, lines 11-14). This is only possible if all publishers currently connected to $C_i$ (excluding p of course, which isn't connected yet) have at least 1 free connection left and in this case a new cluster with attributes $S \cap CS_{C_i}$ is created, p joins this new cluster and attribute combinations $S \cap CS_{C_i}$ are taken out of cluster $C_i$, what might cause $C_i$ to be removed if the leftover set $CS_{C_i}$ is empty. If cluster $C_i$ can't be split, we try to join cluster $C_i$ to avoid that this cluster is merged with e.g. another cluster $C_j$ whose attribute combinations in $CS_{C_j}$ are included in S (cf. Algorithm 3, lines 16-17). Here the idea is that partial overlapping clusters are not matching so well to $PS^{-\emptyset}_{Adv}(p)$ in contrast to clusters included, so they should be split. If it is not possible to split the cluster it is remembered in a set called "clustersOverlapping".

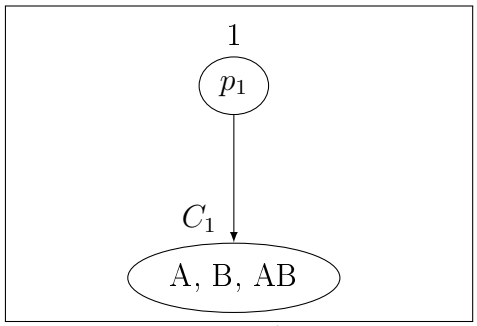
After having calculated all intersections with the existing clusters and reacting accordingly, the algorithm tries to join as many as possible of the clusters that are included in S at this point. They were remembered in a set called "clustersIncluded" and as long as the new publisher p got more than 1 free connection it joins one of these clusters (cf. Algorithm 3, lines 25-29).

In the last step p creates a new cluster $C_{new}$ with $CS_{C_{new}}$=S if there are attributes for which there is no cluster yet (or the cluster is in one of the 2 sets), i.e. S isn't empty. Also if the sets "clustersIncluded" and "clustersOverlapping" aren't empty, because there were not enough free connections, the clusters in these sets are merged with $C_{new}$ (cf. Algorithm 3, lines 30-33).
This part in which the clusters are merged should be avoided for a low false positives rate, but in many cases it can't because we only got limited free connections here. When clusters from the set "clustersIncluded" are merged this shouldn't be so bad as when clusters from "clustersOverlapping" are merged together because they are likely to deviate a lot from the joining publisher's set of attributes, so we try to handle them first when p got all connections free.

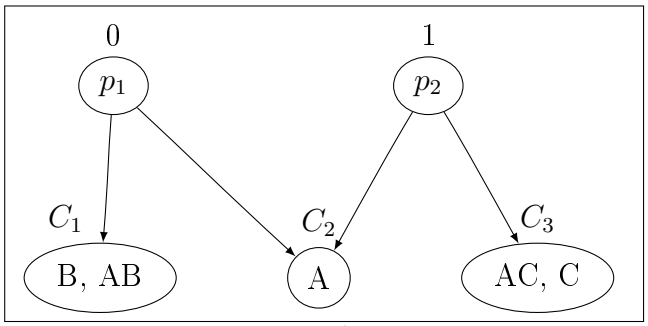
### 4.3.2 Example

In this section we look at some example runs for strategy 3 and show how the attribute combinations in $PS^{-\emptyset}_{Adv}(p_i)$ of our publishers $p_i$ in the system are partitioned into clusters,

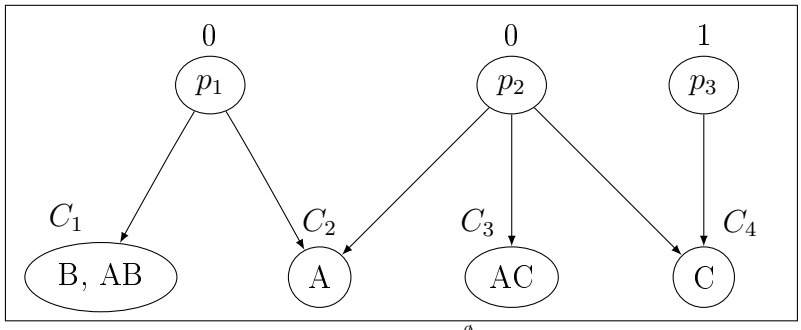
when using the described algorithm (cf. Algorithm 3). As can be derived from the code and was also said in the beginning of chapter 4.3, a certain constraint we didn't look at for strategy 1 and strategy 2 now plays an important role, namely a constraint for the number of clusters a publisher can connect to. This is more towards reality and depending on that individual constraint for each publisher, the system will tend to one of the former strategies. We will therefore attach a number to each node $p_i$, i.e. to each publisher, and this number represents the amount of free connections the publisher got left.



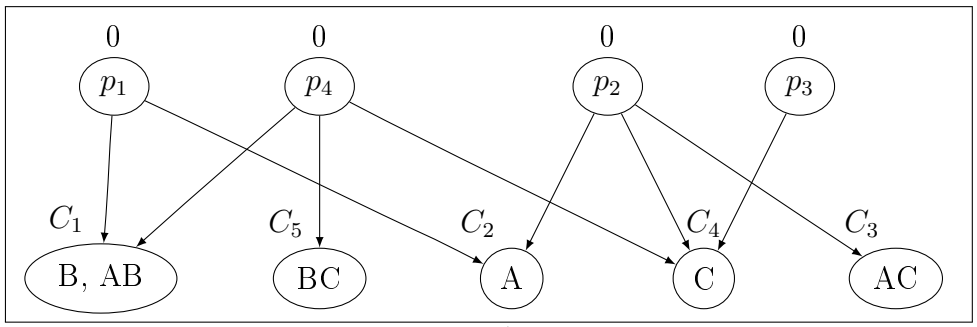

(a) publisher with $PS^{-\emptyset}_{Adv}(p_1) = \{A,\ B,\ AB\}$




(b) publisher with $PS^{-\emptyset}_{Adv}(p_2) = \{A,\ C,\ AC\}$




(c) publisher with $PS^{-\emptyset}_{Adv}(p_3) = \{C\}$



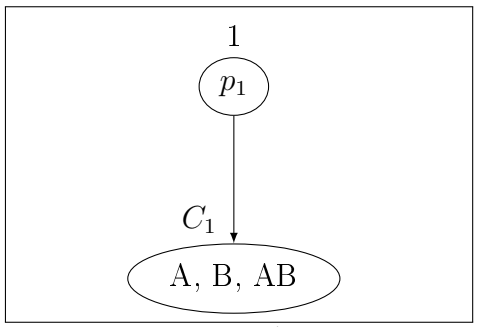

(d) publisher with $PS^{-\emptyset}_{Adv}(p_4) = \{B,\ C,\ BC\}$

Figure 4.3: publishers $p_1$, $p_2$, $p_3$ and $p_4$ using strategy 3

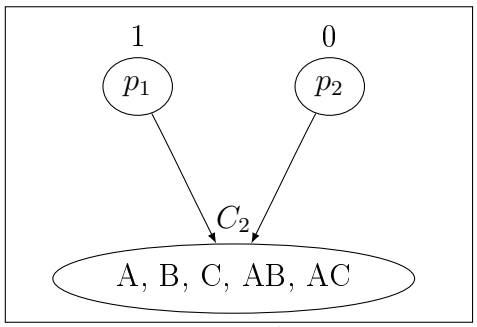
In the example in figure 4.3 we assume 4 publishers that join the system one after the other, starting with a state where no publisher has joined yet, so there are no clusters in the beginning. We also use, if not explicitly declared, the variant where the clusters are

gone through in descending order with regard to the number of attribute combinations a cluster is responsible for. In figure (a) publisher $p_1$ joins with $PS^{-\emptyset}_{Adv}(p_1) = \{A,\ B,\ AB\}$ and his maximum number of free connections is 2. Because there are no existing clusters it creates cluster $C_1$ with $CS_{C_1} = PS^{-\emptyset}_{Adv}(p_1)$ and it connects to $C_1$, what also means that it only got 1 free connection left after this action, since it got a maximum of 2 connections.
In figure (b) we let $p_2$ join which has chosen $PS^{-\emptyset}_{Adv}(p_2) = \{A,\ C,\ AC\}$ and its maximum number of connections is 3. It calculates the intersection with $CS_{C_1}$ and the result is $CS_{C_1} \cap PS^{-\emptyset}_{Adv}(p_2) = \{A\}$. Because only $p_1$ is currently connected to $C_1$ and $p_1$ got 1 free connection left, $p_2$ can split $C_1$ and take attribute A out of it. So a new cluster $C_2$ with $CS_{C_2} = \{A\}$ is created, $p_2$ connects to it and the number of free connections of $p_2$ is now 2. Publisher $p_1$ has no more connection left as it also has to connect to $p_2$.
In figure (c) another publisher $p_3$ comes into the system with $PS^{-\emptyset}_{Adv}(p_2) = \{C\}$ and a maximum number of connections of 2. The only intersection with existing clusters that isn't empty is with $C_3$, because $CS_{C_3} \cap\ PS^{-\emptyset}_{Adv}(p_3) = \{C\}$. Since only $p_2$ is connected to $C_3$ and $p_2$ also got 1 free connection left it is possible to split cluster $C_3$. According to the algorithm $p_3$ creates a new cluster $C_4$ with $CS_{C_4} = \{C\}$ and takes attribute C out of cluster $C_2$. After this action $p_2$ has no connection left and $p_3$ has 1 free connection remaining.
So far we don't have any false positives in all of our clusters, because we followed strategy 1, since as long as there are free connections we split clusters in cases where we have a partial overlap, what means that the intersection $PS^{-\emptyset}_{Adv}(p) \cap CS_{C_i}$ isn't empty and $CS_{C_i} \nsubseteq PS^{-\emptyset}_{Adv}(p)$. Notice that instead of $PS^{-\emptyset}_{Adv}(p)$ we consider the temporary set S, that consists of the attribute combinations from $PS^{-\emptyset}_{Adv}(p)$ for which a cluster hasn't been found yet, while we go through the list of cluster and calculate the intersections. For the intersections we can also use $PS^{-\emptyset}_{Adv}(p)$ instead of S, because of our invariant in formula (3.1). That there are no false positives can be easily seen by looking at the clusters and the corresponding publishers that are connected to them. For example if we focus on cluster $C_2$, we have publishers $p_1$ and $p_2$ connected to it. A subscriber s which is connected to cluster $C_3$ must have a subscription $\Phi_s = \{A\}$, because subscribers with $\Phi_{s'} = \{AB\}$ or $\Phi_{s''} = \{AC\}$ would connect to clusters $C_1$ or $C_3$ respectively and we also assumed in our system model that a subscriber chooses only an attribute combination for which there is already a publisher in the system, what implies that there are no other possibilities. If now $p_1$ publishes a message it contains AB according to its set $PS^{-\emptyset}_{Adv}(p_1)$ and therefore the message in particular contains attribute A, what means that it is not a false positive for s. For publisher $p_2$ connected to $C_2$ we have the analogous case, but here the message contains AC.
When we now look at figure (d) a publisher $p_4$ joins with $PS^{-\emptyset}_{Adv}(p_4) = \{B,\ C,\ BC\}$ and we assume a number of free connections equal to 3. Since we said that we use the variant where the clusters are visited in descending order w.r.t. the number of attributes, it first looks at cluster $C_1$ and recognizes a non-empty intersection that is $PS^{-\emptyset}_{Adv}(p_4) \cap CS_{C_1} = \{B\}$. The problem is that we can't split cluster $C_1$ here due to publisher $p_1$ has no more free connection and for this case the algorithm says that $p_4$

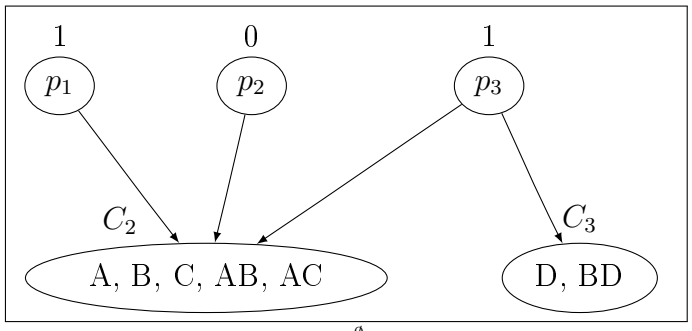
will simply connect to $C_1$. The idea behind it is, that otherwise we would have to merge $C_1$ with a cluster which has nothing in common concerning the attribute combinations, because we only got a partial overlap here. There is also a non-empty intersection with cluster $C_4$ and $p_4$ connects to it because $CS_{C_4} \subseteq PS^{-\emptyset}_{Adv}(p_4)$. These are all non-empty intersections for $p_4$, but we still got attribute BC left and so a new cluster $C_5$ is created with $CS_{C_5} = \{BC\}$.




(a) publisher with $PS^{-\emptyset}_{Adv}(p_1) = \{A,\ B,\ AB\}$




(b) publisher with $PS^{-\emptyset}_{Adv}(p_2) = \{A,\ C,\ AC\}$




(c) publisher with $PS^{-\emptyset}_{Adv}(p_3) = \{B,\ D,\ BD\}$

Figure 4.4: publishers $p_1$, $p_2$ and $p_3$ using strategy 3

To show another behaviour of the strategy we start with the situation in figure 4.4(b). Here we got 2 publishers $p_1$ and $p_2$ which are both connected to cluster $C_1$. The setting arises, because we assume $p_2$ to have only 1 free connection when it joins the system and the result is that a cluster $C_2$ with $CS_{C_2'} = \{A,\ C,\ AC\}$ is created and due to the non-empty intersection with $C_1$ and the lack of free connections, $C_1$ gets merged with $C_2$ and in the end $CS_{C_2} = \{A,\ B,\ C,\ AB,\ AC\}$. Clearly we got false positives for attribute combinations B, C, AB and AC.
If we now let publisher $p_3$ join with $PS^{-\emptyset}_{Adv}(p_3) = \{B,\ D,\ BD\}$ we could in theory take attribute B out of cluster $C_1$, because both $p_1$ and $p_3$ got enough free connections, but our algorithm so far doesn't consider free connections for each attribute combination and instead just takes the minimum of free connections from all currently connected publishers, what is 0 here for $C_2$. It follows that B won't be taken out and $p_3$ just connects to $C_2$. As you can imagine, here is a chance to get a better false positives rate if we would associate an individual number of free connections with each attribute

combination in a cluster. We will follow this approach in the next chapter (4.4). In figure (c) the final result is shown, $p_3$ connects to cluster $C_1$ and also creates a new cluster $C_2$ with $CS_{C_2} = \{D,\ BD\}$

### 4.3.3 Summary

As we shall see in the evaluation part the strategy works as intended, what means that for a high number of free connections per publisher it tends to strategy 1 and the lower it gets the more it tends to strategy 2. The strategy that way adapts to the number of connections a publisher can have and this is the main influence on the resulting false positive rate.

## 4.4 Strategy 3.1

In the last chapter we introduced strategy 3 which is intended to operate between the 2 extremes of strategy 1 and strategy 2. For strategy 1 we got in the worst case too many clusters, but a false positives rate of 0 and for strategy 2, we expect a rather small number of clusters, nevertheless or in fact because of this small number of clusters we also expect a high false positives rate. The strategy presented in this chapter is an extension of strategy 3. In the second example in Chapter 4.3 we saw that there are situations where it would have been possible to split a cluster with regard to the free connections of the involved publishers, i.e. the ones that have attribute combinations in the intersection and more formal these are publisher p' with $PS^{-\emptyset}_{Adv}(p') \cap (CS_{C_i} \cap PS^{-\emptyset}_{Adv}(p)) \neq \emptyset$ for cluster $C_i$ and p the incoming publisher, but strategy 3 doesn't perform the split. This is because for strategy 3 we only look at the minimum of free connections of all publishers currently connected to a cluster. So we attach a value to each attribute combination **a** in cluster $C_i$ which represents the minimum over the free connections of all publisher p' for which it is true that $\mathbf{a} \in PS^{-\emptyset}_{Adv}(p')$ and obviously p' is connected to $C_i$. With these associated values we can find out which attribute combinations in the intersection can be taken out of cluster $C_i$ and we expect a reduction in the false positives rate.

### 4.4.1 Algorithm Description

For the extension of the algorithm for strategy 3, we need to associate a value with each attribute combination **a** in a cluster C, which keeps track of the minimum of connections that are left among all publishers p' with $\mathbf{a} \in PS^{-\emptyset}_{Adv}(p')$ and formally spoken this value is $min(\mathbf{a}) = min(\{\ p'.numberOfFreeConnections \mid \mathbf{a} \in PS^{-\emptyset}_{Adv}(p') \wedge \text{p' connected to C}\ \})$. Because of our invariant an attribute combination occurs in at most one cluster, so this implies that if $\mathbf{a} \in PS^{-\emptyset}_{Adv}(p')$ and $\mathbf{a} \in CS_C$ then p' is connected to cluster C.
So for the rest of the section we assume that we have these values and also if given a subset $T \subseteq CS_C$ we can filter the elements $\mathbf{a}t \in T$ for which min(**a**)>0. The strategy now works as follows (cf. Algorithm 4): If a new publisher p enters the system it as before calculates the intersection with all existing cluster, where it either considers

**Algorithm 4** Strategy 3.1

**Require:** A new publisher p enters the System

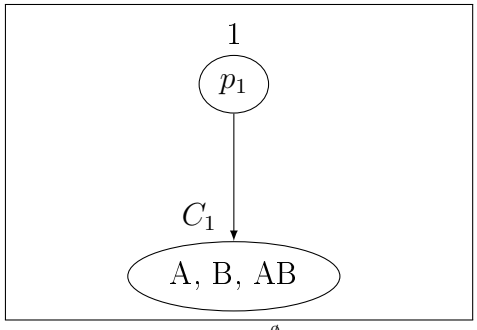
1: S := $PS_{Adv}^{-\emptyset}(p)$
2: clustersIncluded := $\emptyset$
3: clustersOverlapping := $\emptyset$
4: shuffle the list of existing clusters
OR
sort clusters by the number of attribute combinations in descending order
5: **for all** $CS_{C_i}$ **do**
6: **if** $S \cap CS_{C_i} \neq \emptyset$ **then**
7: **if** $\mathrm{CS_{C_i}} \subseteq$ S **then**
8: clustersIncluded := clustersIncluded $\cup\{C_i\}$
9: **else**
10: { partial overlap }
11: **if** numberOfFreeConnections $> 1$ **then**
12: T := $S \cap CS_{C_i}$
13: T := T $\cap$ {**a** $\in CS_{C_i}$ | all publishers for attribute combination **a** have at least 1 free connection}
14: **if** $T = S \cap CS_{C_i}$ **then**
15: create $C_{new}$ with $CS_{C_{new}}$ = S $\cap$ $CS_{C_i}$
16: S := S - (S $\cap$ $CS_{C_i}$)
17: **else**
18: **if** numberOfFreeConnections $> 2 \wedge T \neq \emptyset$ **then**
19: create $C_{new}$ with $CS_{C_{new}}$ = T
20: **end if**
21: join $C_i$
22: S := S - (S $\cap$ $CS_{C_i}$)
23: **end if**
24: **else**
25: clustersOverlapping := clustersOverlapping $\cup$ $\{C_i\}$
26: **end if**
27: **end if**
28: **end if**
29: **end for**
30: **while** ( $\neg$ isEmpty(clustersIncluded) $\wedge$ numberOfFreeConnections $> 1$ ) **do**
31: join $C_i$
32: clustersIncluded := clustersIncluded $\setminus\{C_i\}$
33: S := S - (S $\cap$ $CS_{C_i}$)
34: **end while**
35: **if** S $\neq \emptyset$ **then**
36: create cluster $C_{new}$ with $CS_{C_{new}}$=S
37: Merge Clusters in clustersIncluded and clustersOverlapping to $C_{new}$
38: **end if**

them randomly or in descending order related to $|CS_{C_i}|$, i.e. the number of attribute combinations in each cluster. To recall, the idea behind the sorting is that when we start looking at the clusters a publisher has the maximum of free connections in the beginning and if we start with the "big" clusters it is more likely to split them.
To continue with the algorithm description, if the intersection with $CS_{C_i}$ is such that $CS_{C_i} \subseteq PS_{Adv}^{-\emptyset}(p)$ then there is no change to the former strategy, i.e. p just remembers $C_i$ in a list and will either connect or merge in the last step (cf. Algorithm 4, lines 7-8). But for the case in which $PS_{Adv}^{-\emptyset}(p) \cap CS_{C_i} \neq \emptyset \ \wedge \ CS_{C_i} \nsubseteq PS_{Adv}^{-\emptyset}(p)$ we try to take out as many attribute combinations as possible from $C_i$ with the help of the individual values $min(\mathbf{a})$ for each attribute combination $\mathbf{a} \in CS_{C_i}$.
First we check if p actually got at least two connections left, we reserve 1 connection for the very last step, because p might have to create a new cluster. This could be improved by looking twice at all clusters. If p got at least two free connections then we calculate the set of attribute combinations $\mathbf{a}$ that are in $PS_{Adv}^{-\emptyset}(p) \cap CS_{C_i}$ and for which $min(\mathbf{a}) > 0$, i.e. every publisher p' with $\mathbf{a} \in PS_{Adv}^{-\emptyset}(p')$ has at least one free connection left. In the pseudo code we call this set T (cf. Algorithm 4, lines 11-13). Now we compare T with the intersection $PS_{Adv}^{-\emptyset}(p) \cap CS_{C_i}$ and if both sets are equal, we can take all attribute combinations in the intersection out of $C_i$. So we create a new cluster $C_{new}$ with $CS_{C_{new}} = T$ and are done with cluster $C_i$ (cf. Algorithm 4, lines 14-16). Here we might already gain something in terms of false positives, because strategy 3 wouldn't have done this if there was at least one publisher with no free connection and that is connected to $C_i$.
If T and $PS_{Adv}^{-\emptyset}(p) \cap CS_{C_i}$ are not equal then we have to look at the free connections of the joining publisher p again, since we now need an additional connection, because we want to split $C_i$, but after that operation $C_i$ still has attribute combinations from $PS_{Adv}^{-\emptyset}(p)$ and so we also have to connect to $C_i$. If p has more than 2 free connections (and $T \neq PS_{Adv}^{-\emptyset}(p) \cap CS_{C_i}$), it creates a new cluster $C_{new}$ with $CS_{C_{new}} = T$ and p also joins $C_i$ (cf. Algorithm 4, lines 18-20). Only if p doesn't have enough free connections, we simply connect to $C_i$ and don't take the attribute combinations in T out of $CS_{C_i}$, i.e. don't create a new cluster (cf. Algorithm 4, lines 21-22).
In the situation where $T = \emptyset$ we act like strategy 3 (cf. Algorithm 4, lines 25), but in the other cases it is more likely to split clusters and therefore reduces the number of false positives with regard to the connections each publisher has, what is still the main influence here, because if we got unlimited connections for each publisher, the algorithm behaves like strategy 1 and the result would be a false positive rate of 0.
The last steps are the same like in strategy 3, here we simply try to connect to the clusters where $CS_{C_i}$ is included in $PS_{Adv}^{-\emptyset}(p)$, instead of merging them, what we do in the very last step and naturally we create a new cluster if there are leftover attribute combinations that aren't contained in any cluster yet (cf. Algorithm 4, lines 30-38).

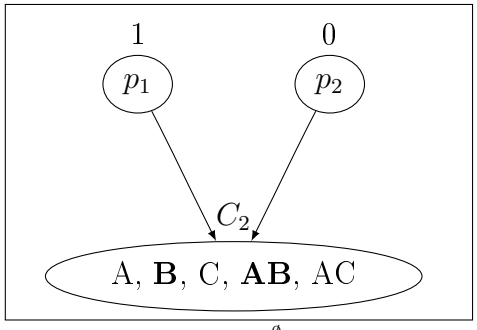
### 4.4.2 Example

To stress the difference in the behaviour of strategy 3.1 as extension of strategy 3, we first look at an example that we have already seen for strategy 3, what means that we

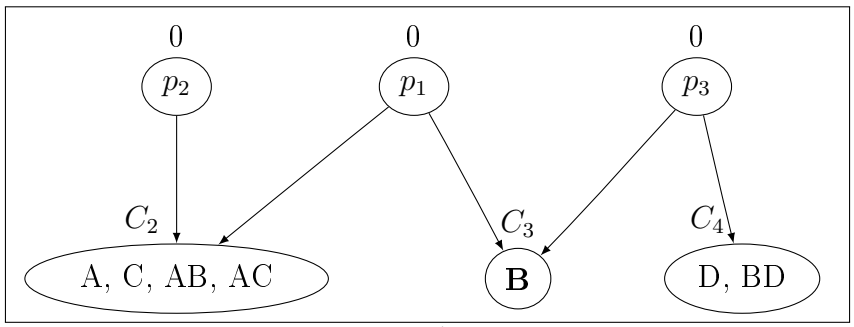
use the same set of publishers and also keep the order in which they join, but as can be expect the result will be different. We again annotate the nodes that represent the publishers with numbers indicating the free connections left for this particular node.




(a) publisher with $PS^{-\emptyset}_{Adv}(p_1) = \{A,\ B,\ AB\}$




(b) publisher with $PS^{-\emptyset}_{Adv}(p_2) = \{A,\ C,\ AC\}$



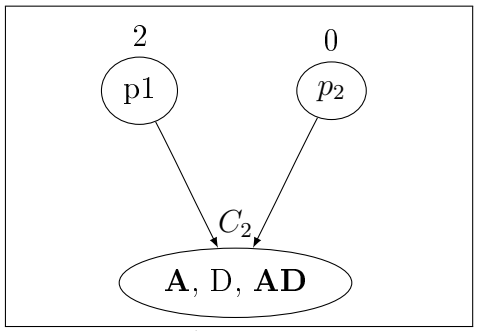

(c) publisher with $PS^{-\emptyset}_{Adv}(p_3) = \{B,\ D,\ BD\}$

Figure 4.5: publishers $p_1$, $p_2$ and $p_3$ using strategy 3.1

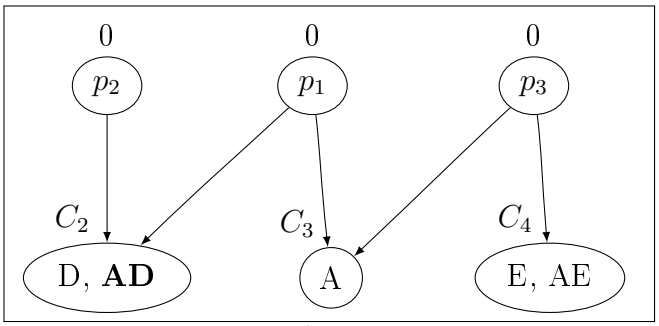
In the example in figure 4.5 we want to show the difference in the behaviour of strategy 3.1 and strategy 3. For completeness the first step, where publisher $p_1$ with $PS^{-\emptyset}_{Adv}(p_1) = \{A,\ B,\ AB\}$ and total of free connections of 2 joins, is drawn in figure (a), but the interesting part starts after $p_2$ has joined the system. In figure (b) there is so far no change to strategy 3, what means that we have publisher $p_2$ entering the system which creates a new cluster $C_2$ and merges $C_1$ with it. This happens because there is a non-empty intersection $PS^{-\emptyset}_{Adv}(p_2) \cap CS_{C_1} = \{A\}$, but $p_2$ has only 1 free connection in total, so from here follows that we merge $C_1$ with our created cluster $C_2$ with $CS'_{C_2} = \{A,\ C,\ AC\}$ and the result is $CS_{C_2} = \{A,\ B,\ C,\ AB,\ AC\}$ as shown in the figure. What you can see is that attribute combinations B and AB in cluster $C_2$ are marked and that's because for these 2 attributes it is still possible to take them out of $C_2$ and thereby split the cluster by creating a new cluster with B or AB.
Now in figure (c) publisher $p_3$ with $PS^{-\emptyset}_{Adv}(p_3) = \{B,\ D,\ BD\}$ joins and same as for strategy 3 we have got an non-empty intersection with cluster $C_2$ that only contains attribute B, i.e. $PS^{-\emptyset}_{Adv}(p_3) \cap CS_{C_2} = \{B\}$. When we now look at the publishers associated with attribute B in cluster $C_2$ we recognize that $p_1$ is the only publisher with

$B \in PS^{-\emptyset}_{Adv}(p_1)$ and thus the minimum number of free connections for attribute B is 1, what implies that we can split $C_2$. The result is a new cluster $C_3$ with $CS_{C_3} = \{B\}$ and both $p_1$ and $p_3$ are connected to it. Additionally $p_3$ creates a new cluster with $CS_{C_4} = \{D, BD\}$.
If we take a closer look at the final outcome it is for this example a nearly optimal case for the clusters and in particular how we have divided the attribute combinations among them. The explanation is as follows: If we look at $p_2$, it is clear that it has to be connected to clusters where attribute combinations A, C and AC occur and since $p_2$ has only 1 free connection in total, these 3 attribute combinations must be in 1 cluster, lets call it C'. The other thing is that $A \in PS^{-\emptyset}_{Adv}(p_1)$, what implies that $p_1$ has to be connected to C', too. So we can't circumvent the false positives for subscribers interested in AC or C and they will arise, because messages from $p_1$ contain only AB, but $p_1$ is always connected to this cluster C'. On the other hand, we have a separate cluster for attribute B, so there are no false positives for it and the same holds for attributes D and BD. The only thing what could be better is that attribute combination AB should perhaps be in the same cluster as B, but subscribers for AD would also experience false positives if we move it to $C_3$ because of $p_3$. What is not possible is to put AB in a separate cluster, since $p_1$ only go 2 connections in total and one of them must be used for a cluster where A occurs and i.e. $p_2$ is connected, as we argued before. So the only other possibility would be to put it together with B. We can also see that no publisher has a connection left and if it indicates that we have divided the attribute combinations into clusters quite good and in particular we don't have a too small number of clusters.

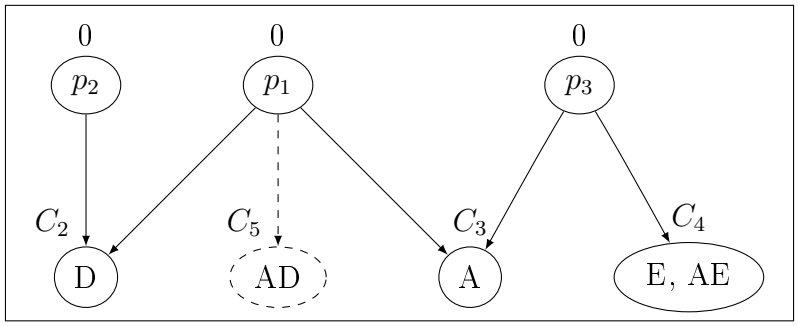
In figure 4.6 we have another example run for strategy 3.1, that is intended to show what can be still improved. We start with the situation in figure (a) where publishers $p_1$ and $p_2$ are already in the system and $p_1$ joined before $p_2$. This constellation appears, because $p_2$ only got 1 free connection in total and when it joins it has an intersection with the attribute set $PS^{-\emptyset}_{Adv}(p_1)$ and clearly also with $CS_{C_1} = \{A,\ D,\ AD\}$.
In figure (b) we let publisher $p_3$ join which has a non-empty intersection with cluster $C_2$ and because we individually count for each attribute the minimum number of free connections from all associated publishers, it is possible to take $\{A\} = PS^{-\emptyset}_{Adv}(p_3) \cap CS_{C_2}$ out of $C_2$. The result are 2 new clusters $C_3$ and $C_4$, whereby $C_3$ contains attribute A only and $C_4$ is for the leftover attribute combinations E and AE.
If we now look at the clusters in (b) then we only have false positives for a subscriber s that is interested in AD, because s would connect to cluster $C_2$ and the messages from publisher $p_2$ contain only attributed D what implies that all these messages are false positives for subscriber s with subscription $\Phi_s = \{AD\}$. If we take a closer look at figure (b) then we recognize that here it would be possible to achieve a false positives rate of 0 for all clusters, i.e. for cluster $C_2$.
The way to do this would simply be to let $p_1$ create a new cluster $C_5$ with $CS_{C_5} = \{AD\}$, because as can be seen, $p_1$ still got 1 free connection left. The question here is when we will do this split, because in the situation where $p_1$ joins, there was no other cluster




(a) $p_1$ with $PS^{-\emptyset}_{Adv}(p_1) = \{A,\ D,\ AD\}$ and $p_2$ with $PS^{-\emptyset}_{Adv}(p_2) = \{D\}$




(b) $p_3$ with $PS^{-\emptyset}_{Adv}(p_3) = \{A,\ E,\ AE\}$




(c) improvement of the situation in (b)

Figure 4.6: possible improvement for strategy 3.1

or publisher and therefore no need to do the mentioned split, i.e. there were no false positives. On the other hand it got 3 free connections in the beginning and it would be possible to change the behaviour of a publisher such that it creates as many clusters as possible when it joins the system, i.e. $p_1$ would create 3 clusters instead of 1 and the consequence would be that the described false positives wouldn't occur. The resulting clusters are shown in figure (c).
Another point where we could do a "reactive split" would be, if we recognize false positives in a cluster, we just look at the attribute combination that causes them and check whether it is possible to take the attribute combination out of the cluster, i.e. create a new cluster with it by looking at the free connections of the associated publishers. The disadvantage in this variant would be the coordinating, because we need to rely on feedback of the subscribers, whereas if we do it at the join of a publisher this wouldn't be needed.

### 4.4.3 Summary

As can be seen in this chapter there are situations where the extension of strategy 3 can really help to reduce the false positives rate for certain clusters, but the cost here is that we have to maintain individual values for each attribute combination in a cluster,

which represent the minimum number of free connections of the associated publishers, as we explained. In fact, it is not very easy to see from here what will be the influence on the overall false positive rate, but as the basic strategy is already adapting to the available free connections of the publishers and the situations where we can gather from the improvement might not appear so often, the reduction in the false positives rate is expected to be of medium size.

# 5 Evaluation

In this part we finally look at concrete implementations of the strategies in chapter 4. With the help of these implementations, we will examine each single strategy and we will focus on the behaviour with regard to the number of clusters that are created and the resulting false positives rate. After explaining the implementation and the modelling of the system in the simulation in chapter 5.1, we will then show the results of our measurement for each strategy and discuss it in the following chapters.

## 5.1 Implementation and Simulation

To evaluate the cluster building strategies for a publish/subscribe system with certain qualities of service, which were developed in this thesis, we implemented the system in Java. Except of the standard libraries, no additional component was used, so the whole implementation was so to say done from scratch. Another possibility would have been to use PeerSim, but because all the algorithms are implemented in Java, it is still possible to use them together with it and our implementation keeps track of all necessary data, so we weren't forced to use it. The only thing that can't be done, concerns the way how the publishers and subscribers join, because clearly the model isn't as complex as PeerSim is, but to show how the system evolves with different strategies, our implementation fits quite well.
For the basics, we model every publisher and subscriber as a separate node, i.e. a Java object, and depending on the type it has certain functions it can execute or more specific all administration of the objects is done by the main procedure. To connect publishers and subscribers, we also model each cluster as a separate node, which keeps a list of all connected publishers and subscribers. A cluster also has a set which represents the attributes that are associated with this particular cluster. Moreover, all publishers and subscribers maintain a list of clusters they are connected to and for the subscribers the list contains exactly one cluster. The publishers and subscribers are additionally kept in a list by the main program (see figure 5.1).

When we initiate the system, first all publishers are created, i.e. they choose randomly a subset $\Psi_p$ from all attributes in $\Omega$, where the size of this subset is determined by a parameter. Then they calculate their individual sets $PS^{-\emptyset}_{Adv}(p)$ by using the chosen set $\Psi_p$. So to recall, if we have 5 attributes named from A to E for $\Omega$, a publisher p might choose $\Psi_p = \{A,\ D\}$ and it follows $PS^{-\emptyset}_{Adv}(p) = \{A,\ D,\ AD\}$.
To be more precise, the publishers choose the attributes from $\Omega$ according to a uniform distribution, so it is truly random and perhaps not realistic, but what concerns the clustering and the resulting false positives, this makes the problem even harder. So if we

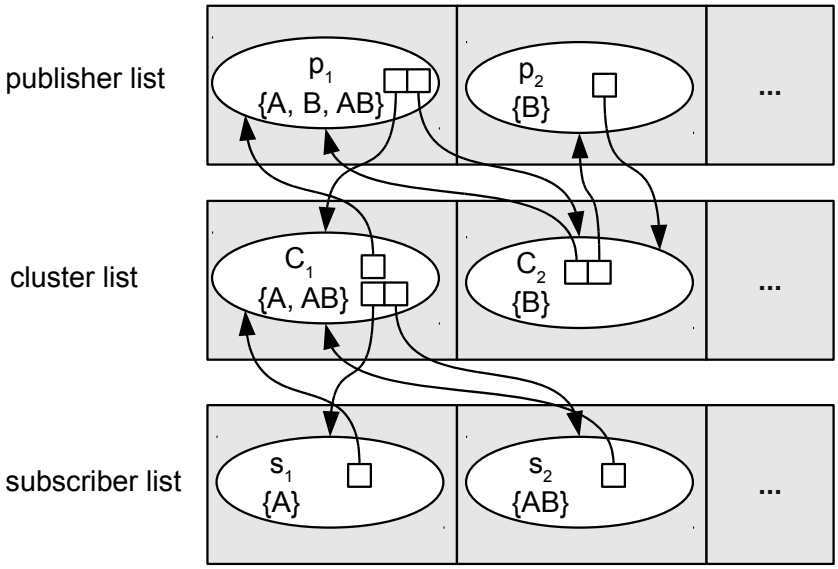



Figure 5.1: interaction of publishers, subscribers and clusters in the implementation

for example would use a Zipfian distribution, the results are very likely to be better, because a Zipfian distribution is more towards clustering and makes it easier in contrast to a uniform distribution.
After we created all publishers we let them join the system in groups of e.g. 10 at a time (but still one after the other) and take our measurement concerning the number of clusters. The clusters are additionally organized by a cluster manager, but here this is only technically, because when publishers for example do a split of a cluster, they can execute the operation via the cluster manager. Here it is important to distinguish, that the cluster manager only calls the corresponding function on the cluster and also does a callback on publisher that invoked it. The real operation is then done by the cluster or publisher itself, so this is not a centralized approach. The cluster manager here only serves to make the (initial) communication of publishers and clusters possible if they don't know each other and coordinates the correct calls for operations like split, merge or join of clusters. It also provides a list of all currently existing clusters, so this could all be done in a distributed manner and we only did it to make the implementation for the simulation as easy as possible.
After all publishers have joined and have built the clusters, we let the subscribers join in groups. We do this because the subscribers choose their subscriptions among the attribute combinations of all publishers currently in the system and in the beginning not all publishers have joined, although they are created. If we e.g. let initially 1 publisher p' join and afterwards 10 subscribers, they would choose their subscription from $PS_{Adv}^{-\emptyset}(p')$ only and we don't want that bias in out simulation. When a group of subscribers has joined, we randomly choose a certain number of publishers and let them publish messages containing all their attributes, e.g. if $\Psi_p = \{A,\ B,\ C\}$ the message from p would contain ABC and it would be sent to all clusters which p is connected to.

The notifications are forwarded from the publishers to the subscribers via our modelled clusters. The individual subscribers then give feedback to the cluster they are connected to and each clusters keeps track of the number of published messages and the number of the resulting false positives. A subscriber can check for a false positive very easily here by matching its subscription to the attributes contained in the message. If we sort the attributes alphabetically it is a simple string match. To calculate the total false positives rate, we collect the data from each cluster and process it, i.e. we collect the number of send messages and the number of the resulting false positives from each cluster and build the quotient of total send messages to total resulting false positives. So we kind of look at the system from the outside and we just see how many messages are created in total and how many of them are false positives.
During the measurement, the data is saved in a file and after the simulation has terminated, several files for gnuplot are created, so the data can be directly transformed into a graph. For the simulation there are various parameters, for example the total number of attributes, number of publishers, minimum and maximum number of attributes a publisher chooses and so on. As we mentioned, it is also possible to simulate different strategies during one run and it is guaranteed, that the set of publishers is identical to make the outcomes comparable. In the next chapters we will now present the results of the simulations.

## 5.2 Strategy 1

In this chapter we will simulate the system by using strategy 1 and what we want to verify are the considerations of chapter 4.1. This means that we want to check formula 4.1 that we derived from our thoughts and show that our considerations are correct.
For formula 4.1 lets recall, that we said that the maximum number of clusters is $c_{max1} = 2^n - \sum_{i=m+1}^{n} \binom{n}{i}$, if we have in total $|\Omega| = n$ attributes the publisher can choose from, every publisher chooses at most $m$ out of those $n$ attributes and we have $m \neq n$.
In the simulation we used 50 publisher that joined the system for all parameter combinations that we will explain in the following.
In the first part we let the value for n fixed at n=10 and vary the minimum and maximum number of attributes a publisher chooses from $|\Omega|$. In the second part we reduce n to n=5 and also vary the number of attributes a publisher chooses to show that the formula is correct and not dependent on some special parameter value. For n=10 we also show the case where n=m=10 to test that the system behaves as we said, i.e. we let every publisher choose $\Psi_p = \Omega$.
If we look at the results in figure 5.2 we can see for all graphs, that the more publishers join, the more clusters we get. So the number of clusters is monotonically non-decreasing. This is plausible, because we only split clusters and never merge them. There is also a maximum value for each graph (except the red one) that is exactly according to formula 4.1. You should also notice, that the scale of the y-axis is logarithmic with base 2, so the growth of the number of clusters is really quite fast.
If we take a closer look at the graph with the green crosses, we see that the maximum is

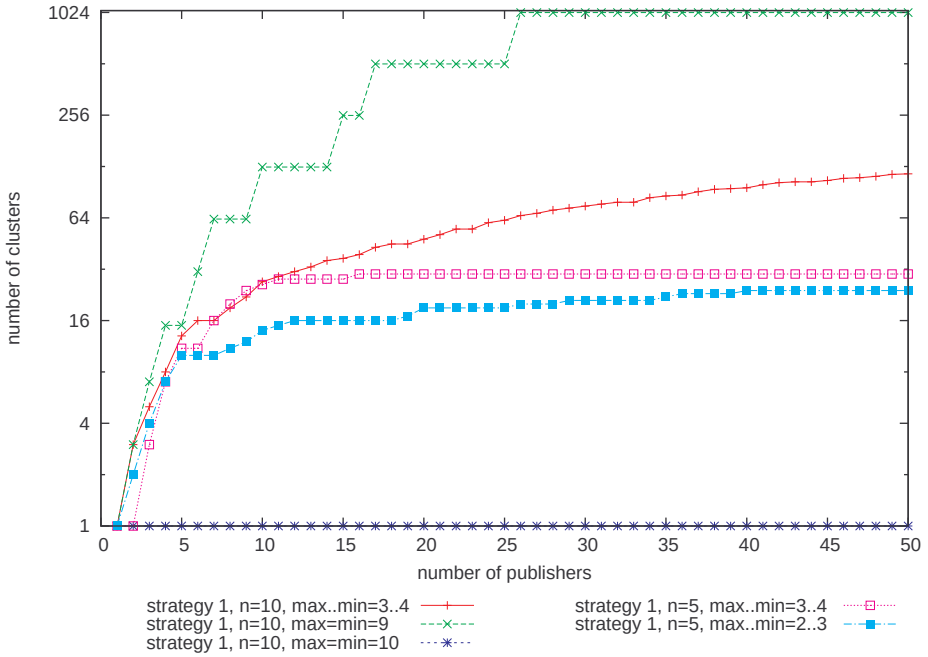



Figure 5.2: growth of the number of clusters for strategy 1, simulated for $|\Omega| = 10$ and $|\Omega| = 5$ and varying minimum and maximum number of attributes a publishers chooses from $\Omega$

at about 1022 (value taken from the data file). Here we have n=10 and m=9, so when calculating $c_{max1}$, we get $c_{max1} = 2^{10} - \sum_{i=10}^{10} \binom{10}{i} = 1023$ what is correct, because we need all $\binom{10}{9} = 10$ possible publishers with regard to $\Psi_p$, to get exactly 1023 clusters and since the publishers choose randomly this can be tolerated, because we only got 50 publishers in the system.
For the graphs with n=5, we got the similar result, so for n=5 and m=4 we get a maximum value of 30 and for n=5 and m=3 it is 24. The calculated values are $2^5 - \sum_{i=5}^{5} \binom{5}{i} = 31$ and $2^5 - \sum_{i=4}^{5} \binom{5}{i} = 26$ and this corresponds to our simulation. We clearly need to take the maximum number of attributes a publisher chooses for m, because the clusters for the publishers choosing a value smaller than this m are completely split (see also the explanation in chapter 4.1.3).
Next we look at the graph with the dark blue crosses for n=m=10 and what we can see is that there is only 1 big cluster and this makes sense, because every publisher chooses $\Psi_p = \Omega$ and therefore we have one cluster $C_1$ with $CS_{C_1} = 2^{\Omega}$ and for every publisher p that joins it holds that $CS_{C_1} = PS^{-\emptyset}_{Adv}(p)$. So it follows that we only get one cluster.
The last graph we look at is the red graph for n=10 and m=4 and if we calculated the maximum according to our formula, we get $c_{max1} = 2^{10} - \sum_{i=5}^{10} \binom{10}{i} = 386$, but in the graph the maximum is only 116. The explanation is that we need $\binom{10}{4} = 210$ different publishers and we only got 50 publishers in the system here, so clearly it is not possible to reach this maximum number of clusters. Therefore we made a run with 3000 publishers for n=10 and m=4 and the result is shown in figure 5.3. The scale is this

time non-logarithmic, because it is only one graph and as can be seen, the maximum is at 385 and fits nearly perfectly.

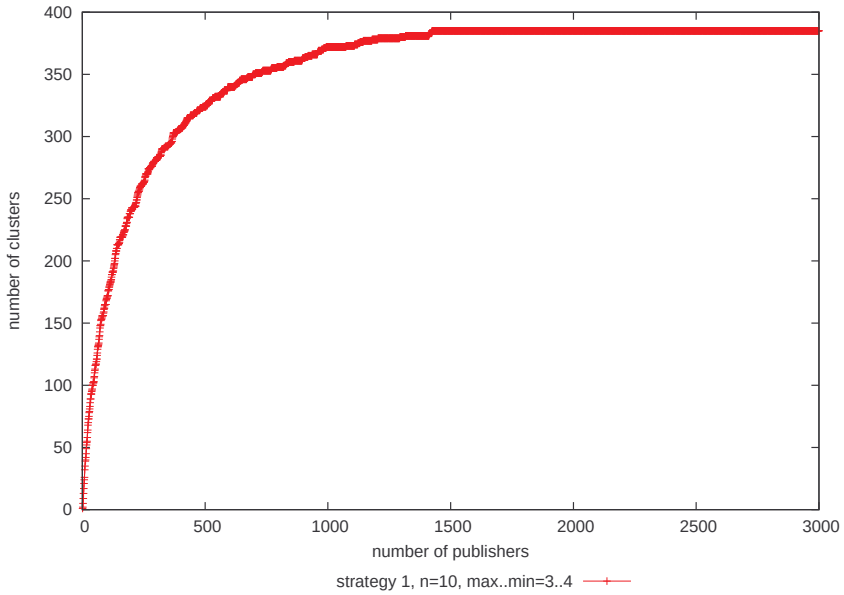



Figure 5.3: growth of the number of clusters for strategy 1, simulated with 3000 publishers for $|\Omega| = 10$ and minimum number of attributes a publishers chooses min=3 and maximum number of attributes a publishers chooses max=4

In figure 5.4 we also plotted the results for the average number of connections a publisher has to establish, i.e. to how many clusters a publisher has to connect to. We can see, that the graph is very similar to figure 5.2, what means that the number of clusters gives a hint to the average number of connections a publisher has to establish. We also use a logarithmic scale for the y-axis and regarding the graphs, they are nearly as monotonic as in figure 5.2.
Nevertheless there are parts where the average number decreases and this can be explained, because the connections a publisher needs to establish when it joins depends on the existing clusters and $\Psi_p$ that the publisher chooses. So if we don't have the maximum number of clusters yet, a publisher p doesn't have to connect to a separate cluster for each element in $PS^{-\emptyset}_{Adv}(p)$. But notice that the more publishers join, the more clusters we get and publishers already in the system also have to establish new connections. From this point of view the number of connections for a publisher only increases and never decreases. It even more depends on the minimum and maximum of attributes a publisher chooses, because the number of connections clearly is determined by $|PS^{-\emptyset}_{Adv}(p)|$.
If we look at the graph with the green crosses in figure 5.4 the number of attributes a publisher chooses from $\Omega$ is 9 and we see that the maximum value we reach is 511, what means that nearly all clusters contain exactly one attribute combination. For the

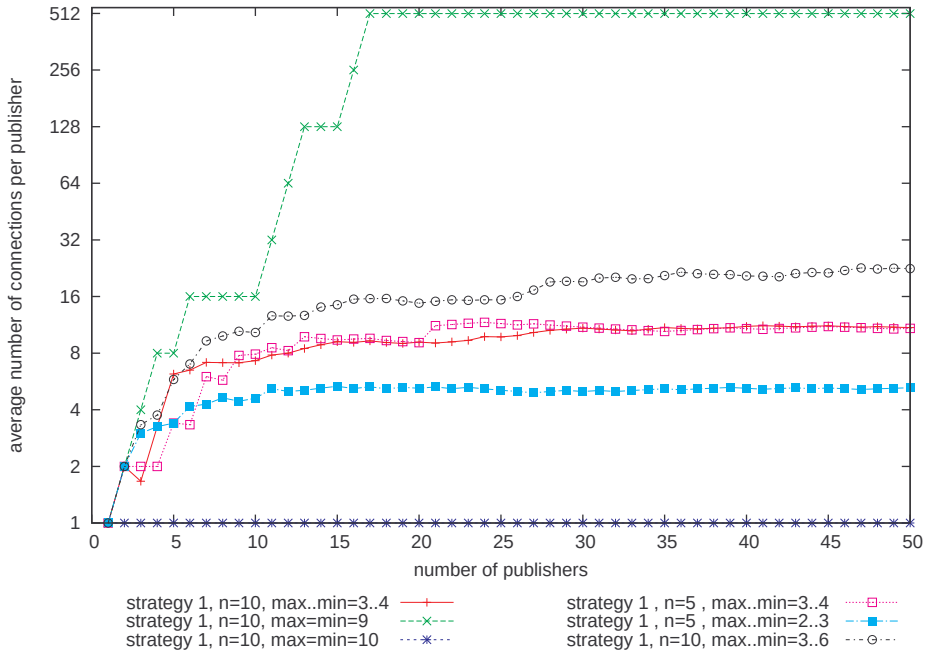



Figure 5.4: average number of connections per publisher for strategy 1 simulated for $|\Omega| = 10$ and $|\Omega| = 5$ and varying minimum and maximum number of attributes a publishers chooses from $\Omega$

publishers it follows, that each of them has to connect to $|PS^{-\emptyset}_{Adv}(p)| \approx 2^9 = 512$ clusters. Here we can see that the upper bound is exponential in the number of attributes $|\Psi_p|$ a publisher chooses from $\Omega$, since $|PS^{-\emptyset}_{Adv}(p)| = 2^{|\Psi_p|} - 1$. We also see, that the graph is monotonically non-decreasing.
For the graphs with n=5, we get nearly the same result. So for $min..max = 3..4$ we expect a value between 8 and 16 and this is exactly what we see and for $min..max = 2..3$ we discover a value between 4 and 8. The non monotonic behaviour of the graph can be explained, if we think about the last characteristic we saw, namely that we said that the average value is between for example 8 and 16. So there are 2 type of publishers in the system, one that chooses 3 attributes from $\Omega$ and the other chooses 4 attributes. Because we determine the type, i.e. how many attributes are chosen, randomly, it is clear, that the average value must vacillate between these 2 values and therefore the graph can't be monotonic.
For n=m=10, i.e. we choose $\Psi_p = \Omega$ for every publisher, we got the confirmation, that there is only one cluster by examining the dark blue graph.
What concerns the red graph, we see that its value also vacillates between 3 and 4 as we have for n=5 and this clearly shows, that the number of connections a publisher needs to establish only depends on the number of attributes he chooses from $\Omega$ and for this strategy they really grow exponential in the worst case. Additionally to figure 5.2 we included a graph for n=10 and $max..min = 3..6$ in figure 5.4, and what we can see here is similar to the red graph, namely that we really need a lot of publishers to reach the

maximum of connections per publisher, what is here $\binom{10}{6} = 210$. The reason is that we need many publishers that choose 6 attributes from $\Omega$, but the probability for them is not so high and therefore we don't get so many clusters. So for small $|\Psi_p|$, i.e. the number of attributes a publisher chooses, this strategy truly might be a good choice.
Nevertheless if we wouldn't have a uniform distribution, which concerns the selection of attributes from $\Omega$, the connections probably wouldn't grow that bad, but still noticeable. So there might be cases, where this strategy is a good choice, especially regarding the false positives rate which is 0. We omitted the graph, because showing a function that is constantly 0 is senseless, but if we look at the graph for the false positives for the example in figure 5.3 and also the log file, we had a total number of 30 000 subscribers and about 4.8 million messages that arrived in total at the subscribers and none of them was a false positive.

## 5.3 Strategy 2

Now we take a closer look at strategy 2 in this chapter and make simulations to verify our considerations from chapter 4.2. Analogous to the previous paragraph we want to verify formula 4.2 that we derived for strategy 2 and look at the behaviour of the system in general, i.e. the growth of clusters. A characteristic that is very important here will be the false positive rate, because as we already considered it is not 0 for this strategy and moreover really needs attention. To recall, a false positive in our world here is a notification received by a subscriber, where the attributes in the message don't cover the whole subscription $\Phi_s$, i.e. not all attributes contained in the attribute combination in $\Phi_s$ occur in the message.
If we look at formula 4.2 from chapter 4.2.3 we said that the maximum number of clusters is $c_{max2} = \binom{n}{m}$, if we have in total $n = |\Omega|$ attributes a publisher can choose from and every publisher chooses at most m out of these n attributes. In contrast to strategy 1, we here do the simulation with 200 publishers that join the system, because the merging of clusters that appears here is a little bit more dependent and we want to make sure that when we look at the false positive rate, it is derived from a system/simulation that has run for some time, so we choose more publishers.
The number of joining publishers is fixed for all parameter combinations that we will use and to make it comparable with the results from strategy 1 we simulate with the same parameters, but we will omit one case for n=5. What means that we will vary $n = |\Omega|$ and for some fixed n, here for n=10, we also vary the minimum and maximum number of attributes a publisher chooses from $\Omega$.
If we look at the results in figure 5.5 we can see that for all graphs we have the situation that the more publishers join, the more clusters we get, but it is not monotonic. Especially for the red graph we see some parts, where the function is decreasing and it makes sense, because if we think about the algorithm we merge clusters and only create a new one if the attribute combination didn't occur so far. This means that the overall grow results from the joining publishers and their different attribute sets and the decrease results from merges where $CS_C \subseteq PS^{-\emptyset}_{Adv}(p)$ for some cluster C and joining publisher p.

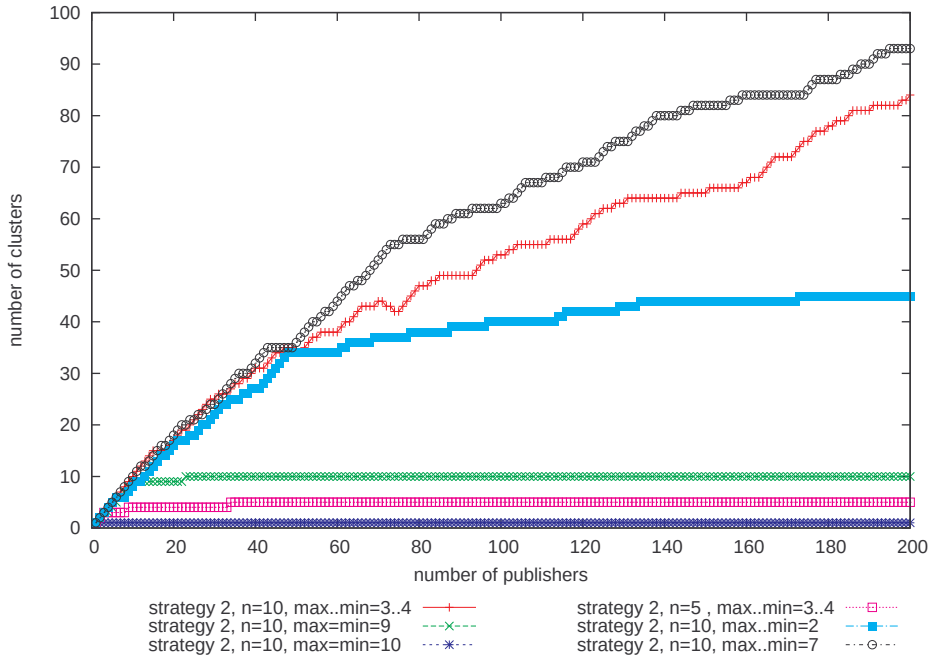



Figure 5.5: growth of the number of clusters for strategy 2, simulated for $|\Omega| = 10$ and $|\Omega| = 5$ and varying minimum and maximum number of attributes a publishers chooses from $\Omega$

We also see, that the number of clusters grows linear or even sub-linear and another observation is that the more attributes a cluster chooses form $\Omega$, i.e. if m grows and if we have nearly m=n, the less clusters we get and this makes sense, because the more we merge the clusters until we only get one big cluster.
If we now take a closer look at the graph with the green crosses in figure 5.5 we recognize that the maximum value for the number of clusters is at 10. Here we have n=10 and m=9 and according to our formula $c_{max2} = \binom{10}{9} = 10$, what fits exactly.
For the graph with the green squares we can read a maximum value of 5 and if we calculate it with our formula we have n=5 and m=4, so $\binom{5}{4} = 5$. Notice that we take here again the maximum number of attributes a publisher chooses from $\Omega$, because the smaller clusters all get merged with the bigger ones and that is how we actually derived our formula in 4.2.3.
For the dark blue graph we have again the special case that we have exactly one clusters and this is plausible, because every publisher chooses $\Psi = \Omega$ and since we only merge or create a new cluster, there must be only one big cluster. Notice that one publisher p with $\Psi_p = \Omega$ is enough for this case to arise.
If we take a closer look at the red graph, we see that we don't really reach a maximum value and that is because we for n=10 and m=4 we need all $\binom{10}{4} = 210$ different publishers and clearly we can't get there with only 200 joining. So we included again another run with 3000 publishers (see figure 5.6) and here we can see that we get a a maximum of 210 clusters which corresponds to our formula.

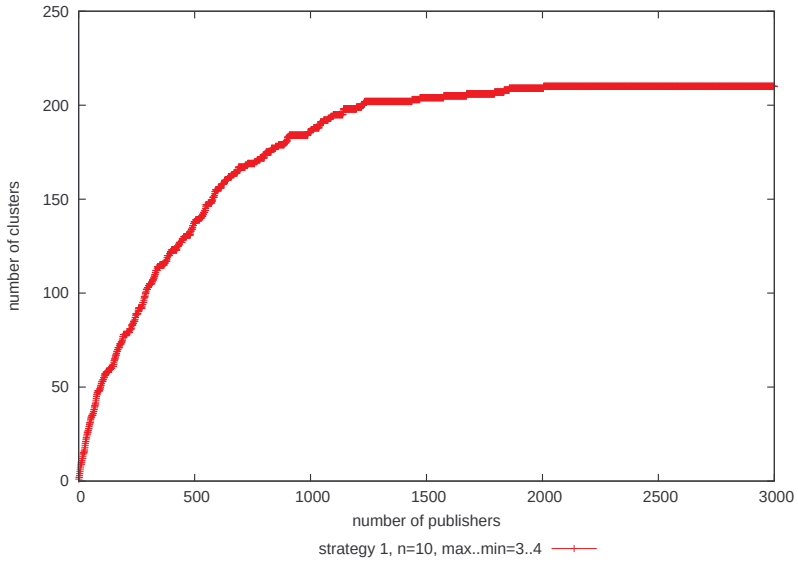



Figure 5.6: growth of the number of clusters for strategy 2, simulated with 3000 publishers for $|\Omega| = 10$ and minimum number of attributes a publishers chooses min=3 and maximum number of attributes a publishers chooses max=4

We also included graphs for m=2 and m=7 and for the case where m=2 we see that if we only have $\binom{10}{2} = 45$ different publishers the number of clusters is clearly lower than as for example for m=4 with 210. For m=7 we have a stronger growth than we have for m=4, but the maximum here will be still $\binom{10}{7} = 120$, whereas the graph for m=4 will grow up to 210.
In figure 5.7 we also have graphs for the average number of connections per publisher, i.e. to how many clusters a publisher has to connect to. We can see that the more attributes a publisher chooses from $\Omega$, the less connections we get, except for m=7, but also here we end up with a maximum of about 14 connections per publishers when simulated with 1000 publisher. Again the argument is, that the more attributes a publisher chooses, the more we merge and for a growing m, $\binom{n}{m}$ decreases and only for values around $\frac{n}{2}$ the connections are "high", but there are never as many as for strategy 1. So if we need to characterise the growth of connections per publisher, it is linear or even sub-linear depending on the maximum number of attributes a publisher chooses and also depending on $n = |\Omega|$.

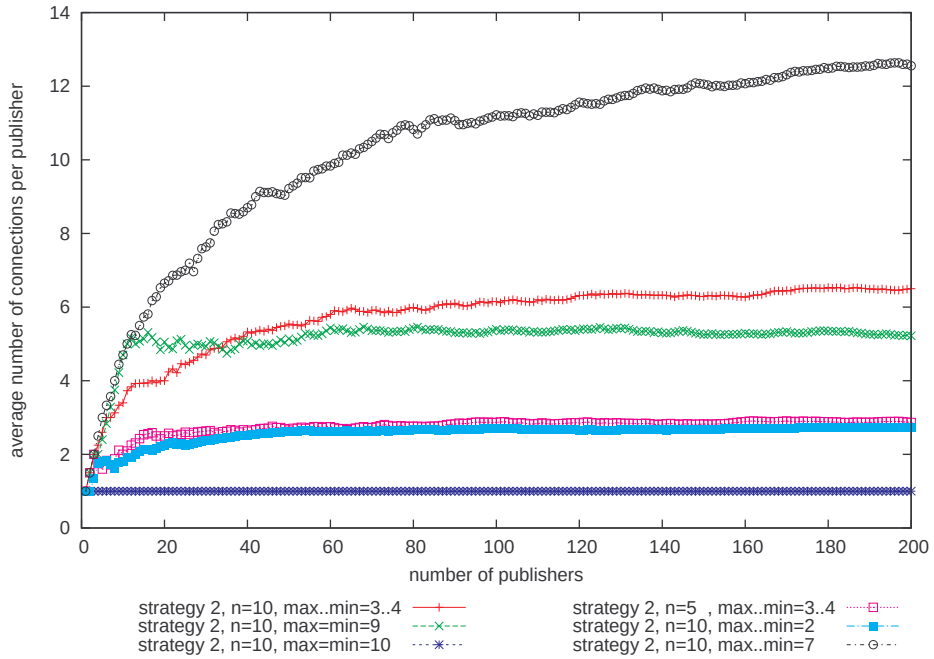



Figure 5.7: average number of connections per publisher for strategy 2 simulated for $|\Omega| = 10$ and $|\Omega| = 5$ and varying minimum and maximum number of attributes a publishers chooses from $\Omega$

We will now discuss the results for the false positives rates in figure 5.8 and what we can see is that the more m is towards $\frac{n}{2}$, we get a higher false positives rate. This can be explained by the fact that for a higher m we only get few different publishers (remember it is $\binom{n}{m}$) and therefore each publisher has a lot of attributes from $\Omega$ and so can server a lot of subscribers. If we consider the dark blue graph we see a false positive rate of 0, but this is a special case where $\Psi = \Omega$ for every publisher, so the result is clear. For a small m we have the analogous case, namely that we have a lot of small clusters.
To find out why we get these false positives rates, we look a little bit more in detail at the simulation for n=10 and m=7, i.e. the graph with the dark circles and for n=5 and m=2, so that is the graph with the light blue squares.
For the first case we get a very high false positives rate and if we look at the histogram in figure 5.9 we can see that there are a lot of clusters with $|CS_C| = 1$ and also a lot with $|CS_C| < 5$, but we also see really big clusters with e.g. 96 attributes and there are a lot of subscribers, too, since we choose the subscriptions uniform at random, so that is where the high false positives rate results from. We just don't have enough clusters for this scenario and the result are too many big clusters with a high false positives rate, resulting in an overall high false positives rate, because many subscribers are connected to the big clusters and there most of the messages are created there.
If we in contrast look at the histogram for n=5 and m=2 in figure 5.10 we see that we have only clusters with at most 3 attributes and a lot of clusters with only one attribute combination, so this clearly implies that we get a quite low false positives rate

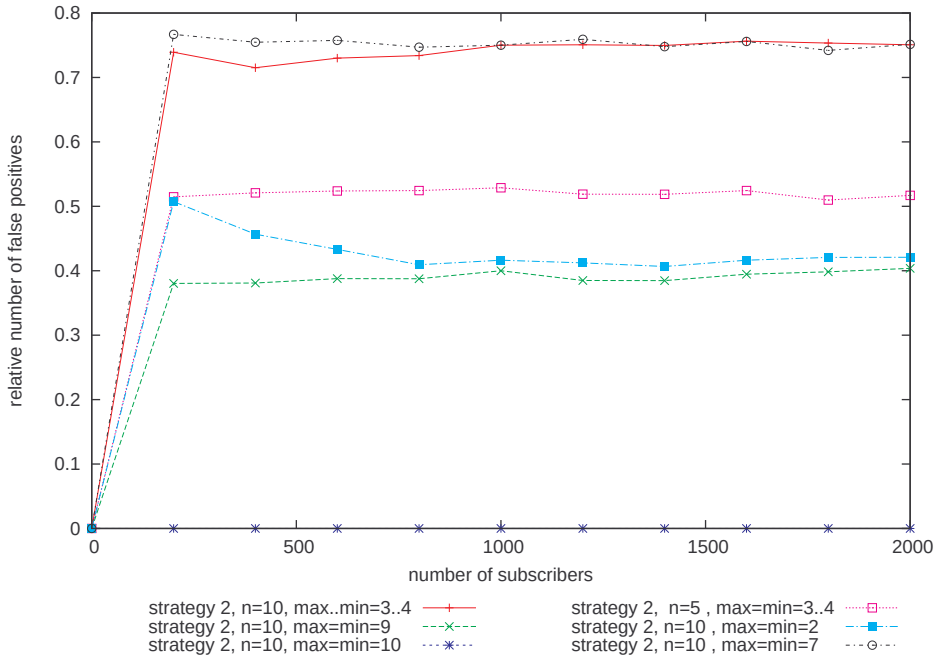



Figure 5.8: false positives rate for strategy 2 simulated with $|\Omega| = 10$ and $|\Omega| = 5$ and varying minimum and maximum number of attributes a publishers chooses from $\Omega$

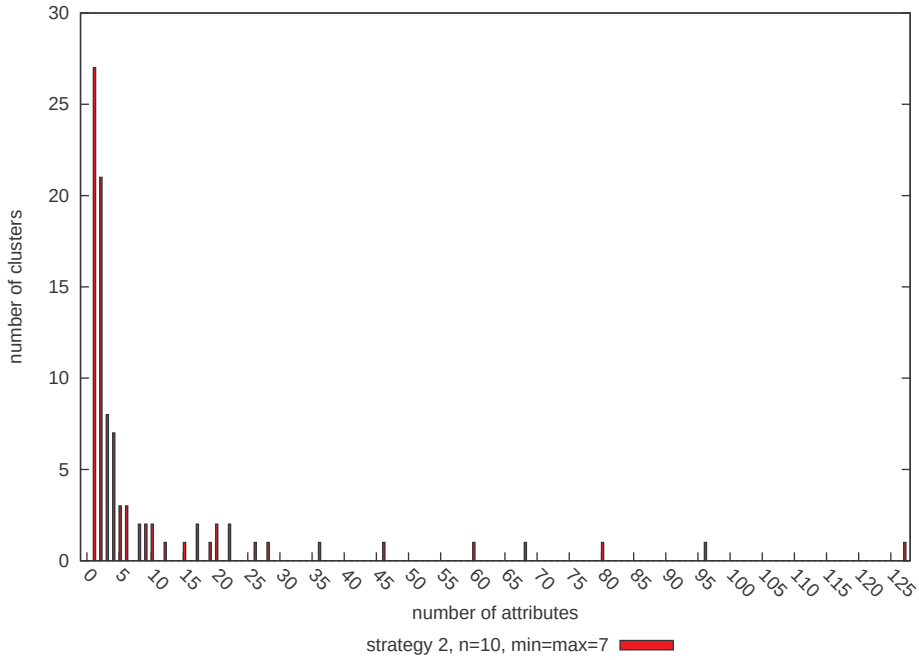



Figure 5.9: histogram for strategy 2 simulated with $|\Omega| = 10$ and minimum and maximum value of attributes a publisher chooses $min = max = 7$

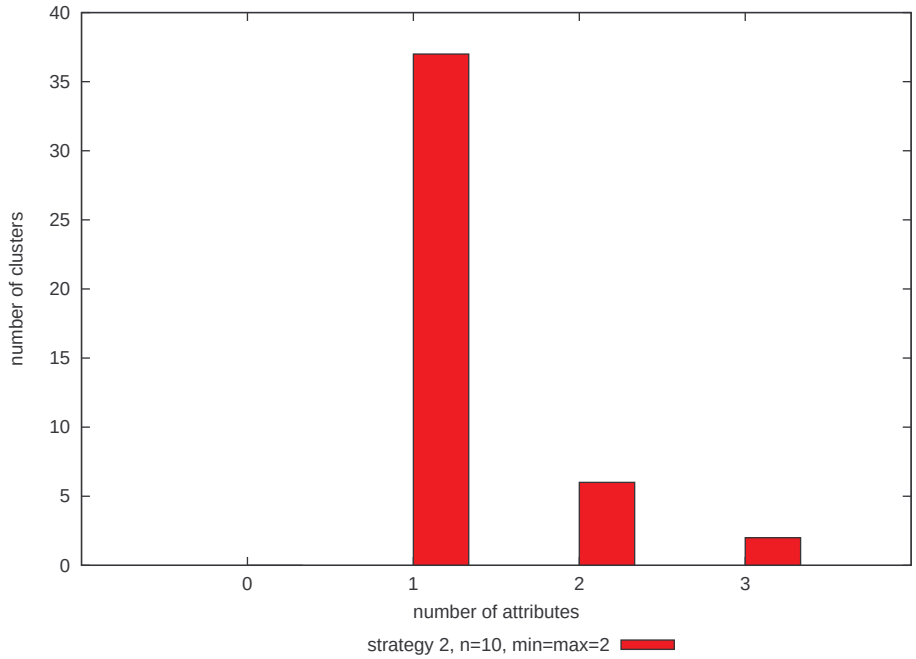



Figure 5.10: histogram for strategy 2 simulated with $|\Omega| = 5$ and minimum and maximum value of attributes a publisher chooses $min = max = 2$

of about 40%, what is still very high, but for every cluster with $|CS_C = 1|$ we got a false positives rate of 0, so the medium false positives rate results from the clusters with 2 or 3 attributes.

What we have seen here is that this strategy really gets rid of the number of clusters we get and therefore also of the connections a publishers has to establish. But the problem here is that for certain constellations we get a very high false positives rate, what might be not tolerable for some applications and it is also not very desirable. The result is a little bit astonishing, but in general it is plausible, because if we always merge we put together all attribute combinations that consist of a single element from $\Omega$ and that way we get high false positives rates for subscribers interested in only these attribute combinations. So as far as we don't care about the connections per publisher, strategy 1 performs a lot better, but only in this case. In the next chapter we will evaluate the results for strategy 3, that is intended to be in-between of the extremes we saw so far and should adapt, depending on the free connections we got at hand.

## 5.4 Strategy 3

In this chapter we will evaluate strategy 3 that is intended to be between the 2 extremes of strategy 1 and strategy 2 what concerns the number of clusters and the false positives rate. One very important thing her is that as already said in chapter 4.3 we have a constraint concerning the number of connections a publisher can establish and it has a

major influence on how the clusters can be created. Moreover we want this strategy to tend, depending on the number of connections each publisher has, to either strategy 1 or strategy 2 and as should be clear, the more connections we have, the more it should go towards strategy 1 and the less we have, the more it should go towards strategy 2. This behaviour is one aspect we want to show and we will also look at the resulting false positive rates here. We will do some simulations for a fixed number of joining publishers, attribute set $\Omega$ and the number of attributes a publisher chooses from $\Omega$. For this fixed parameters we will only vary the minimum number of connections a publisher chooses, because we will fix the maximum of connections, too. Another aspect we want to have a look at is the behaviour of strategy 3 when we use the descending and the random variant and we want to compare them to estimate how much we gain for the false positive rate by sorting the existing clusters in descending order before considering them.

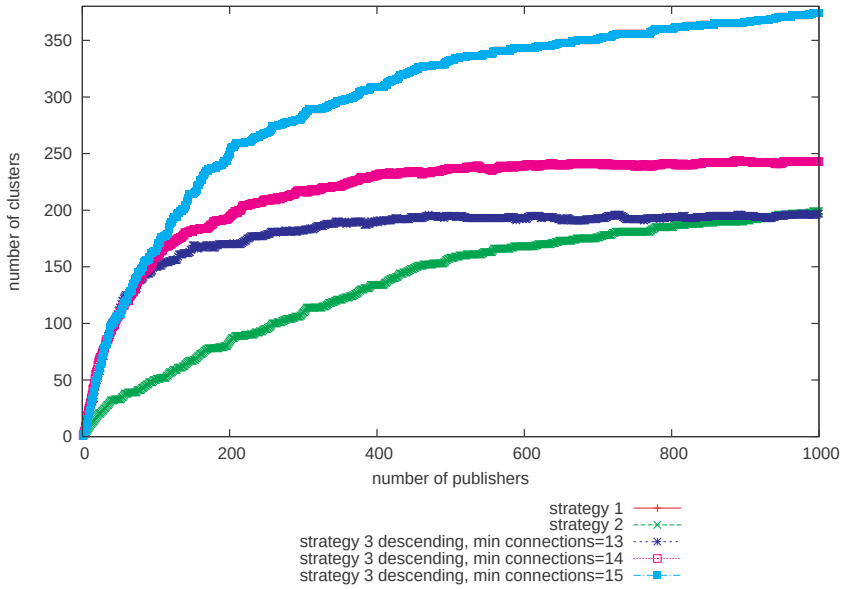



Figure 5.11: growth of the number of clusters for strategy 3, compared with strategy 1 and strategy 2, simulated for $|\Omega| = 10$,minimum and maximum number of attributes a publishers chooses from is 3 to 4 and number of maximum connections per publisher fixed at 15, minimum number of connections varies from 13 to 15

We will now look at the simulation results in figure 5.11. Here we used a total of 1000 publishers that join the system and each of them chooses a minimum of 3 and a maximum of 4 attributes from $\Omega$. For the total number of attributes we have chosen $|\Omega| = n = 10$ and all these parameters stay fixed during the simulation. The maximum number of connections a publisher can establish is also fixed at 15 connections, but what will vary is the minimum number of connections, i.e. it changes from 13 up to 15 connections. Notice that for strategy 1 and strategy 2 these connections are not considered in the algorithms, so the graphs are nearly the same every time. For the results in figure 5.11 we recognize that strategy 3 behaves in the simulations as we expected. The green graph

represents strategy 2 and the red graph should represent strategy 1, but it is identical with the light blue graph for strategy 3 if every publisher has 15 connections and we will explain that.
For a minimum of connections of 13, we get the dark blue graph and as we can see, the growth of the number of clusters is a little bit stronger in the beginning if compared to strategy 2, but when all publishers have joined, the number of clusters is nearly the same or even a little bit less. This can be explained if we think about a reasonable number of splits of clusters for the first part of publishers that join and the more publisher join, the less splits we get and probably the number of merges that occur also rises, so we end up with a number of clusters corresponding to strategy 2.
If we look at the magenta graph, where we simulated with a minimum number of connections of 14, we recognize that the number of clusters goes more towards strategy 1 and this is plausible, since the more connections we have, the more splits are done of course what lead leads to more clusters.
The last graph is the light blue and as we already mentioned it is identical with the graph for strategy 1 and this is because if we keep in mind that a publisher p chooses at most 4 attributes from $\Omega$, we have $|PS^{-\emptyset}_{Adv}(p)| = 2^4 - 1 = 15$ and so every publisher got enough connections for every attribute in $PS^{-\emptyset}_{Adv}(p)$ and it implies that we only have clusters containing 1 attribute.

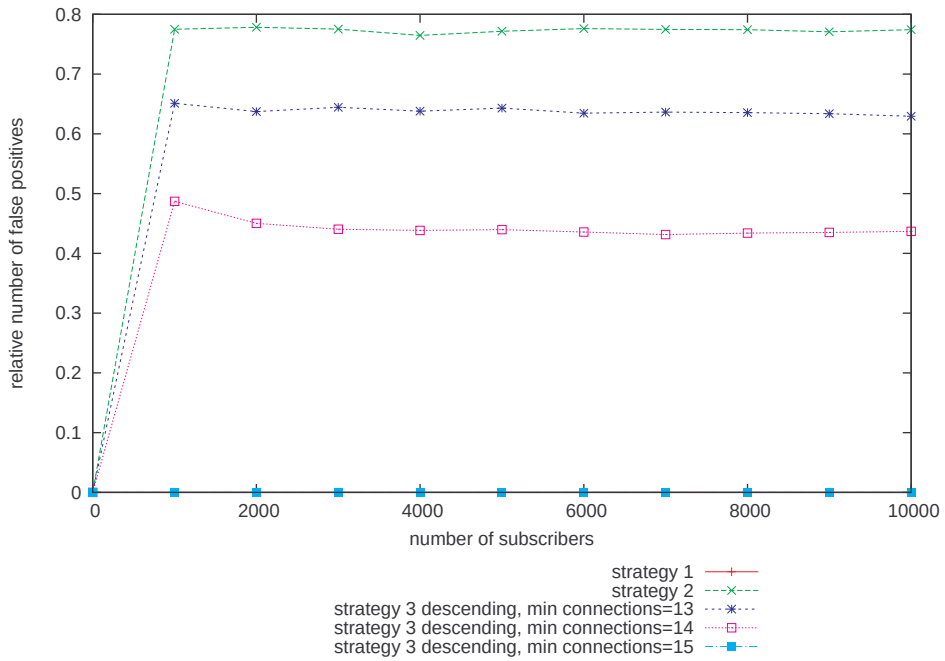



Figure 5.12: false positive rate for strategy 3 compared to strategy 1 and strategy 2 simulated with $|\Omega| = 10$

The graphs in figure 5.12 show the corresponding false positive rates for the graphs of the number of clusters in figure 5.11 and what we can see here is that the false positive rate is always better as for strategy2, independent of the minimum number of connections a

publisher has.
Even for the dark blue graph that belongs to the simulation where the minimum number of connections is 13 and as we saw for figure 5.11 we nearly got the same number of clusters and the false positives rate is about 10 percentage better compared to strategy 2. For the graph with the magenta squares, i.e. the minimum number of connections is 13, we get a moderate false positives rate of 45 percent and for the scenario that was equal to strategy 1, what concerns the number of clusters, we clearly get a false positives rate of 0.
To summarize, we showed with our simulations, that strategy 3 adapts to the number of connections each publisher has and depending on that, we get more or less clusters and a more or less high false positives rate, i.e. it is between strategy 1 and strategy 2 as we wanted.

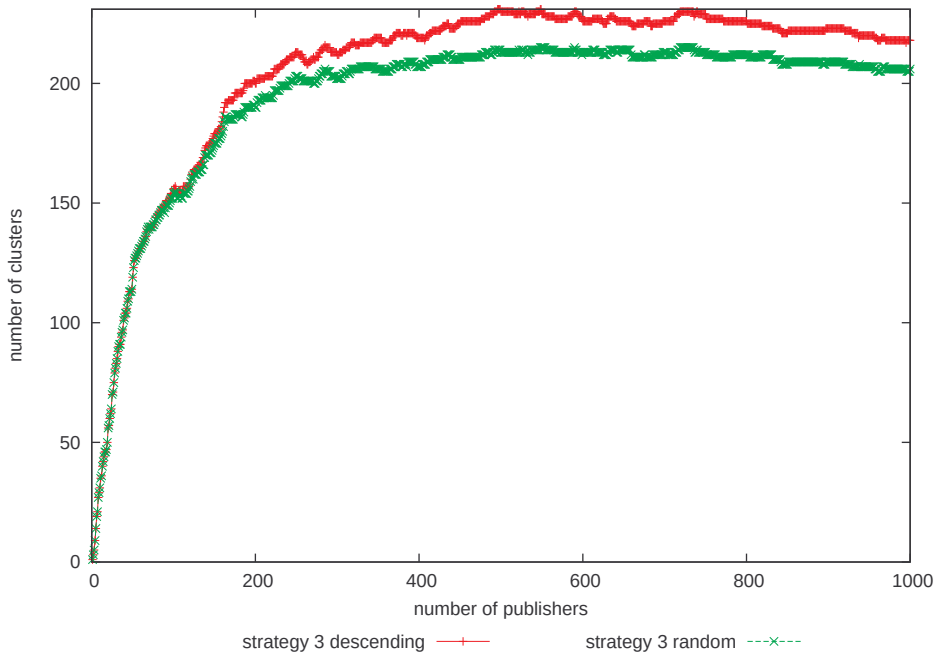



Figure 5.13: comparison of strategy 3 descending and strategy 3 random for the number of clusters, simulated for $|\Omega| = 10$, minimum and maximum number of connections per publisher 13 to 15

In this part we will now compare the 2 variants of strategy 3, namely the one where we look at the clusters in descending order with regard to $|CS_{C_i}|$ and the we used so far for the previous simulations and for the other variant we don't sort the clusters in descending order, but go through them randomly, so this strategy is called strategy 3 random. For the simulation we again use 1000 publishers, $|\Omega| = 10$, every publisher chooses 3 to 4 attributes from $\Omega$ and also chooses randomly from 13 to 15 for the free connections it in principle has.
What we can see in figure 5.13 is that for the descending variant we get more clusters than for the random variant and because we are sure, that we have enough connections

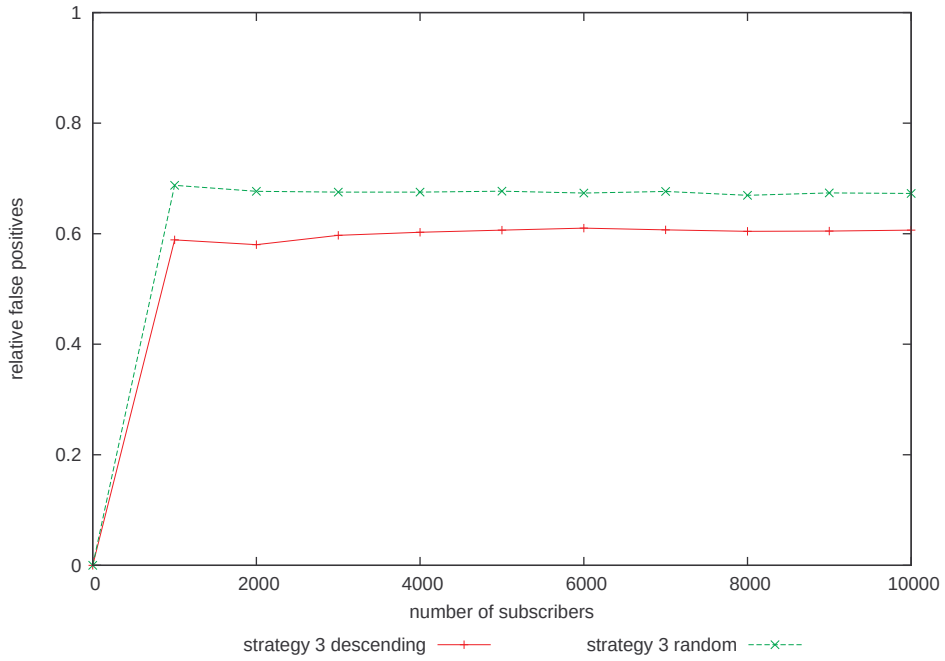



Figure 5.14: comparison of strategy 3 descending and strategy 3 random concerning the false positive rates, simulated for $|\Omega| = 10$, minimum and maximum number of connections per publisher 13 to 15

for each publisher, since they choose them in the beginning and both strategy respect the constraint in the connections, more clusters are better here and so the descending variant is better. The higher number of clusters for the descending variant is plausible, because we look at the big clusters first and so it is more likely to split and that is actually why we do this sorting.
The results in figure 5.14 show the false positive rates for the identical simulation that was done for figure 5.13 and here we see that the descending variant is clearly better and has a false positives rate, that is about 10 percent better than for the random variant. One explanation is that we have more clusters here and so this hints for a better false positives rate. Moreover if we look at the histograms in figure 5.15 and figure 5.16 for this simulation, where we have the distribution of the clusters and their corresponding false positives rate, we see that the descending variant has more clusters with a false positives rate of 0 and what is more important here, there are less clusters with a high false positives rate of about 75 percent and more clusters with a rate of about 45 percent for strategy 3 descending. So this explains the result for the total false positives rate, since the most notifications are sent in the big clusters, because it is more likely that a subscriber chooses a attribute from them if we keep in mind that they select uniform at random and therefore these clusters determine to a great extent our total false positives rate, because we calculate it by dividing the number of false positives by the number of all sent notifications.
What we have seen here in the last part is that the descending variant of strategy 3

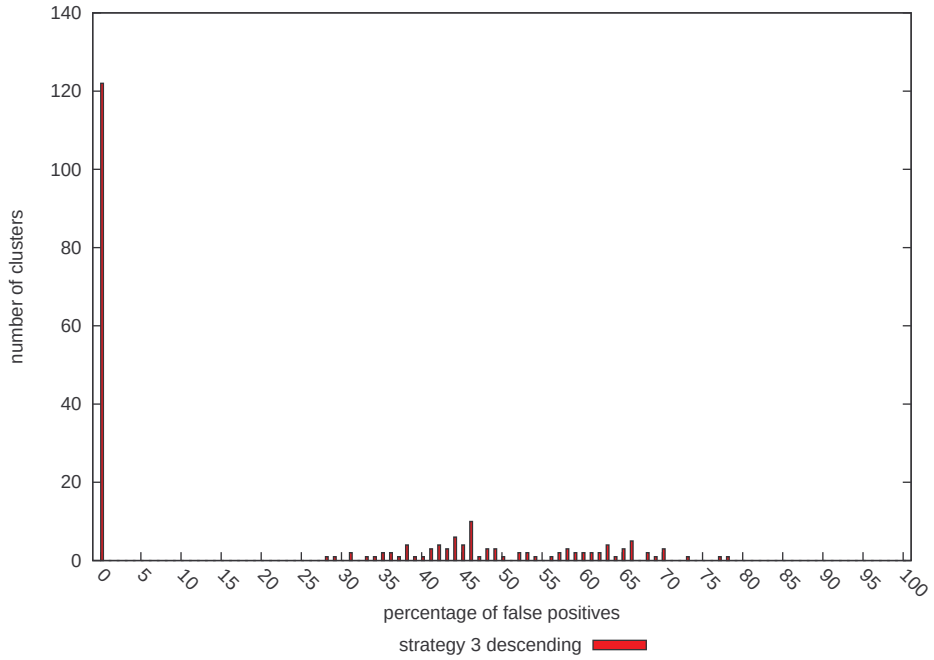



Figure 5.15: histogram for strategy 3 descending, simulated with $|\Omega| = 10$, minimum and maximum value of attributes a publisher chooses $min = 3$ and $max = 4$ and minimum and maximum number of connections per publisher 13 to 15

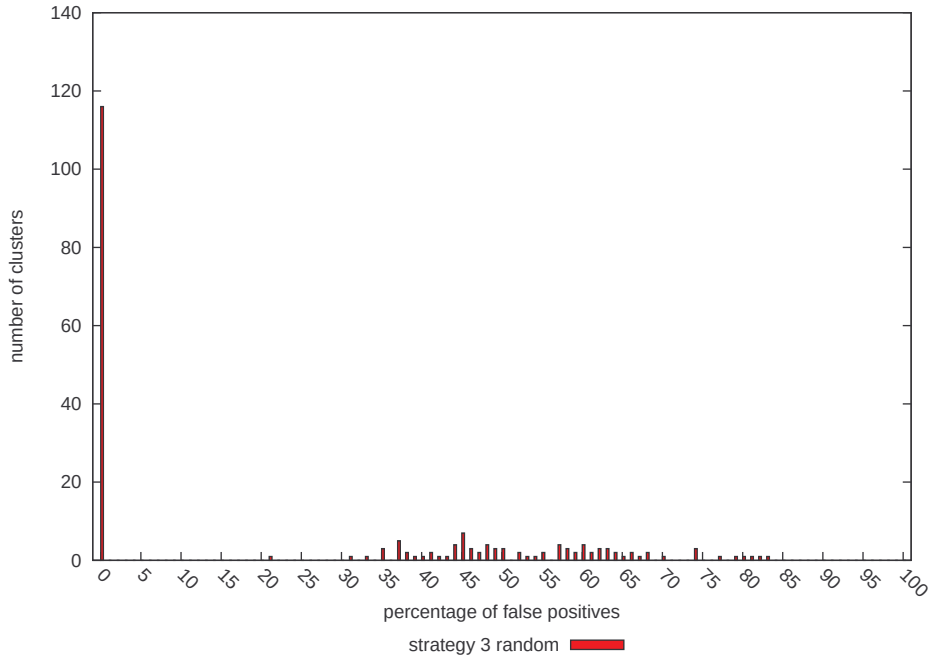



Figure 5.16: histogram for strategy 3 random, simulated with $|\Omega| = 10$, minimum and maximum value of attributes a publisher chooses $min = 3$ and $max = 4$ and minimum and maximum number of connections per publisher 13 to 15

performs better than the random variant concerning the aspect we are interested in, i.e. the number of clusters that got build is higher and we also got a lower false positive rate. The only aspect that can't be estimated here is the effort that we need for the sorting or more particular that we first need to collect the information from all clusters, whereas for the random strategy we can do this on the fly, but it also needs to see all clusters in the end. When thinking about how the system evolves, the descending variant should be worth the additional effort it needs.

## 5.5 Strategy 3.1

In this chapter we will finally evaluate strategy 3.1 which is the extension of strategy 3 and to recall, for this extension we now have for every attribute combination **a** in a cluster a separate value for the minimum of free connections, so we only consider publishers p for which it holds that $\mathbf{a} \in PS_{Adv}^{-\emptyset}(p)$ and we use this to determine whether we can take this attribute out of the cluster, instead of calculating the minimum over the free connections of all publishers that are connected to the cluster. We will compare 2 variants of strategy 3.1. The one where we look at the clusters in descending order concerning $|CS_C|$ and for the other variant we look at the clusters randomly and also compare them with the "descending variant" of strategy 3 which we have already seen in the previous chapter.
For the simulation results that we will now look at, we used again $|\Omega| = 10$ and let 1000 publishers join which choose 3 to 4 attributes uniform at random from $\Omega$ and which also choose their total free connections from 11 to 15 uniform at random. This scenario is interesting, because if we have a publisher that chooses 4 attributes from $\Omega$ and has only 11 connections, chance for false positives rise.

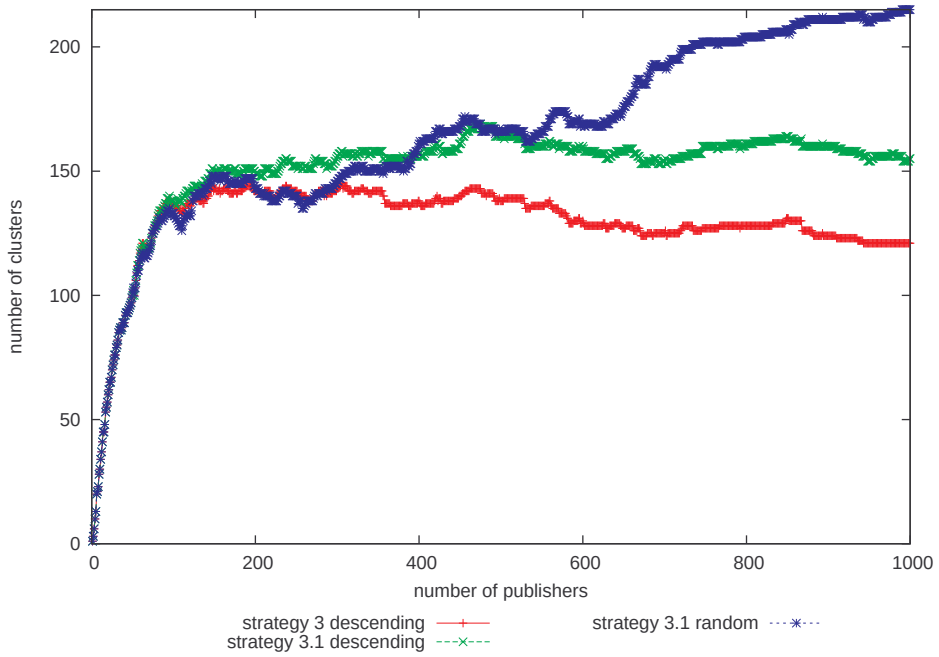



Figure 5.17: growth of the number of clusters for both variants of strategy 3.1 compared with strategy 3

If we look at figure 5.17 we can see that both variants of strategy 3.1 create more clusters than the descending variant of strategy 3 and here again we are sure that we have enough connections for each publisher and since all of the considered strategies respect the maximum number of connections a publisher can establish, the more clusters we get the better is the strategy. This implies that both variants of strategy 3.1 are better than

strategy 3 and the random variant of strategy 3.1 performs best here, because it creates about 60 clusters more. (If we look at the log file there are 215 clusters for the random variant and only 155 clusters for the descending variant.)

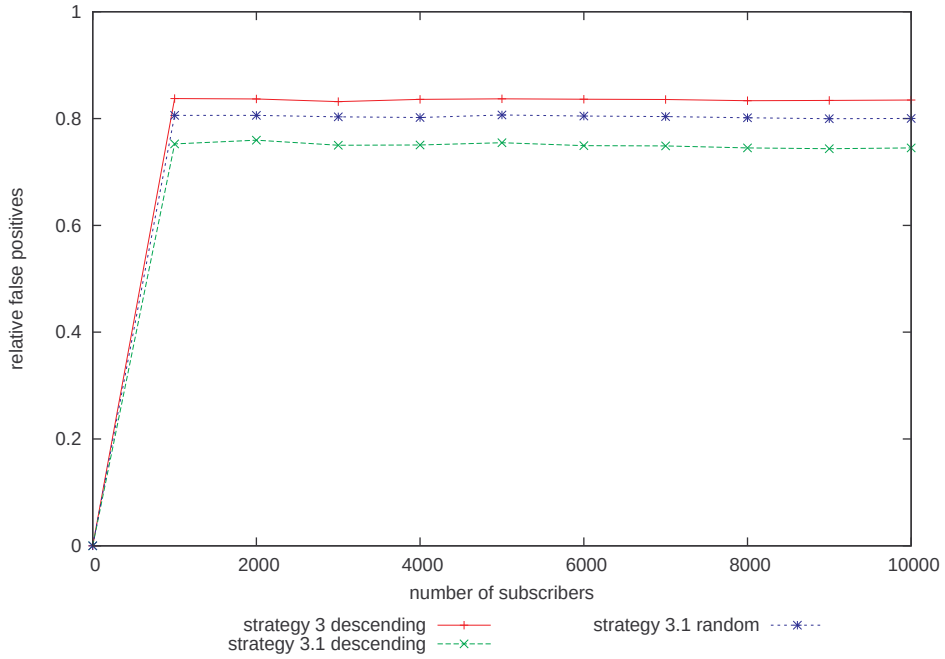



Figure 5.18: false positive rates for both variants of strategy 3.1 compared with strategy 3

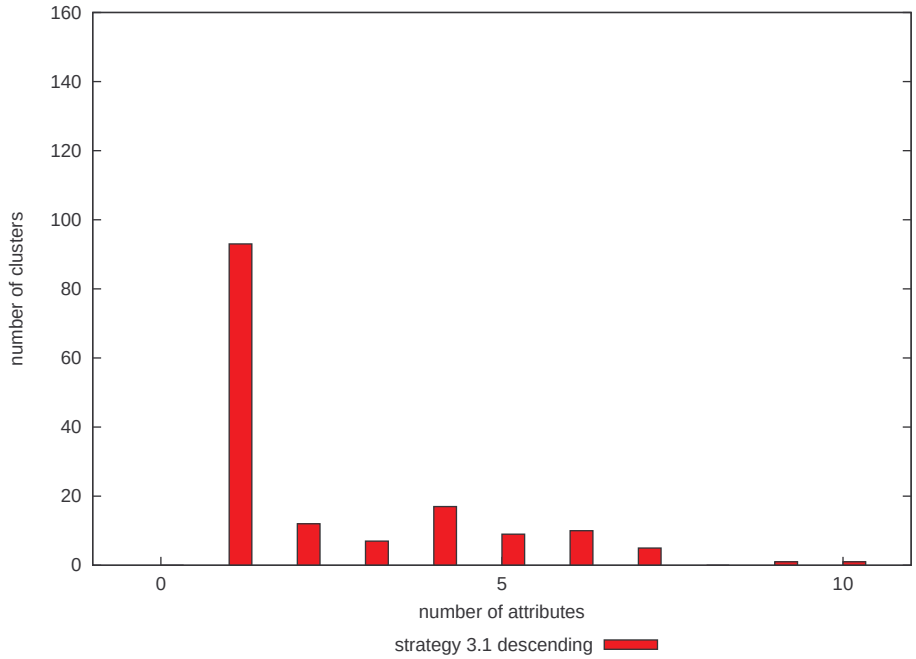
The results for the false positive rates are shown in figure 5.18 and here we again see, that strategy 3.1 is better than strategy 3. One reason for the overall quite high rate is the fact that when looking at our parameters we can have publishers p with 4 attributes and only 11 connections and in this case $|PS^{-\emptyset}_{Adv}(p)| = 15$, so it is not possible to have a separate cluster for all attributes in $PS^{-\emptyset}_{Adv}(p)$ and therefore false positives can hardly avoided. The only aspect that is a little bit strange here is the fact, that despite it produces more clusters, the random strategy has a higher false positives rate, but we will try to explain why.

When we look at the number of attribute combinations each cluster has, as shown in figure 5.19 we see that for the descending variant there are only 2 clusters with 9 and 10 attribute combinations, but the random variant produces 4 clusters that have more than 10 attribute combinations and that is a problem, because here we also get a lot of subscribers that are connected to these big clusters, since they choose their subscriptions uniform at random from $\bigcup_p PS^{-\emptyset}_{Adv}(p)$, i.e. from all attribute combinations in the system. This implies that the most notifications are created in these clusters when we look at the total number of messages and we calculate the false positive rate by dividing the total number of false positives by the total number of sent notifications and because in the big clusters we clearly have a lot of false positives, this is becomes the major influence on the total false positive rate. The random variant only got that high false positive

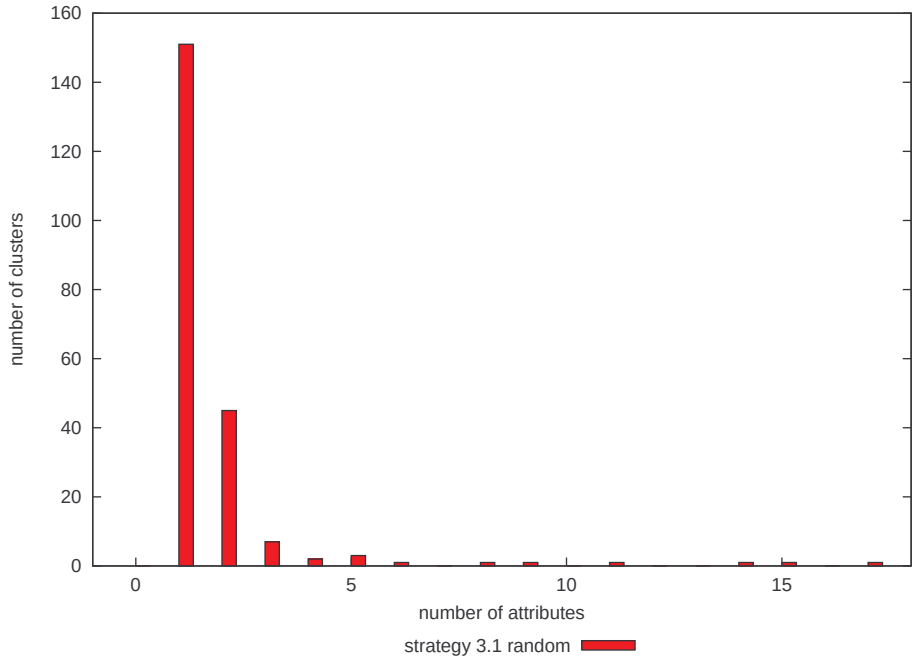
rate, because of these big clusters which the descending variant doesn't posses, since it is programmed to split the big clusters first.

In figure 5.20 we have printed the distribution of the false positive rates separate for each cluster and also calculated the standard deviation. What can be seen here is that the random variant truly is better, because the average, represented by the green line, is at 0.159, so it is smaller as for the descending variant, and also the deviation from the average is smaller, which is 0.258 for the random variant and represented by the blue lines. So there are just a few clusters with a high false positives rate that disturb our total false positives rate.

To summarize, in this part we have compared the descending variant of strategy 3 with both variants of strategy 3.1 and as the results show, both are better than the third strategy. Furthermore the random variant of strategy 3.1 performs better than the descending one if we can tolerate some big clusters with a high false positive rate.

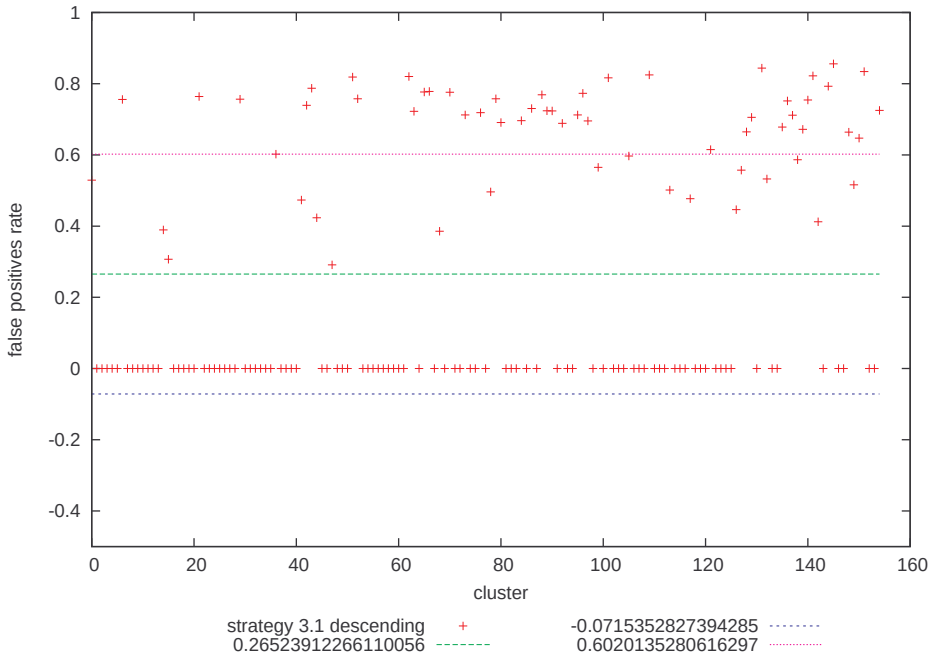



(a) histogram for the attribute combinations per cluster for strategy 3.1 descending

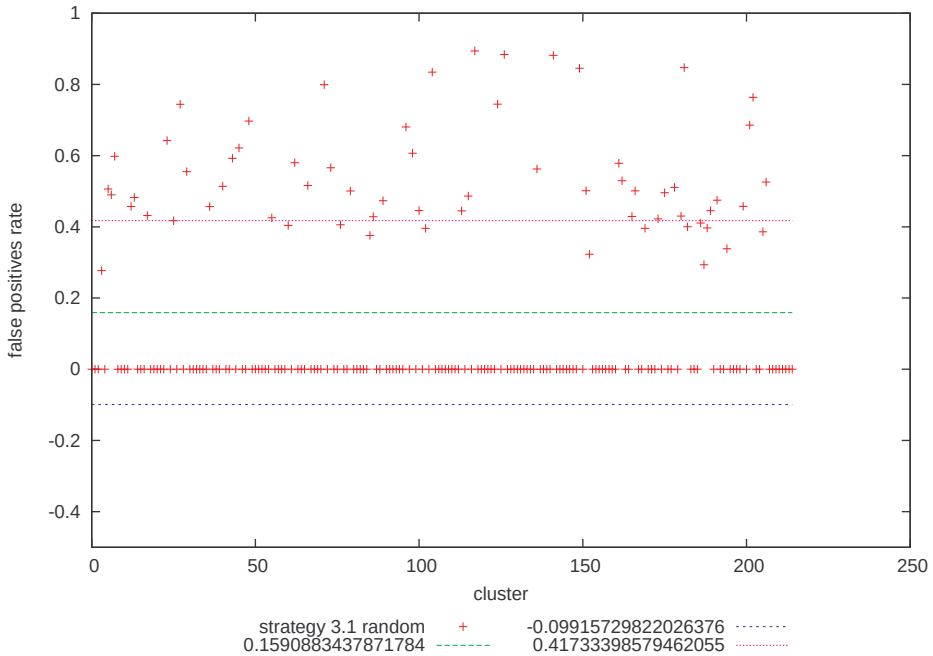



(b) histogram for the attribute combinations per cluster for strategy 3.1 random

Figure 5.19: histograms showing the number of attribute combinations the clusters have




(a) standard deviation for strategy 3.1 descending




(b) standard deviation for strategy 3.1 random

Figure 5.20: false positive rates for every cluster and standard deviation for strategy 3.1

# 6 Conclusion

In this thesis we developed clustering strategies for publishers in the context of a content-base publish/subscribe system. We first looked at 2 extremes that in general are not scalable regarding on the one side the number of clusters and on the other side the rate of false positives. Too many clusters implied that the strategy won't be scalable for the publishers and from too many false positives we can derive that it is not scalable for the subscribers. So we developed a strategy that adapts to the bandwidth constraints of the publishers, what we here modeled with a certain number of connections, and thereby the strategy tries to create clusters in a way that also the rate of false positives is as low as possible.
As the results of the evaluation part show, a publisher needs a certain minimum number of connections that depends on the size of the attribute set of the individual publisher, to be able to build appropriate clusters. Otherwise the result is a too high false positive rate of about 80 percent.
The third strategy we developed was the first into this direction and as shown, depending on the available connections of each publisher, the strategy tends to one of the two extremes, namely either a lot of clusters and a quite low false positives rate or a a medium or small number of clusters and a high false positives rate. Because of the fact that our third strategy respects the number of maximal connections a publisher can establish, it is then desirable to get as many clusters as possible, since it implies a low false positives rate.
Between our 2 variants of strategy 3 the one which sorts the clusters in descending order with regard to their number of attributes, at first glance provides better results than the random variant and again the idea of the sorting was to make it more likely to split large clusters, because the more different attributes a cluster contains the higher is the expected false positive rate.
We then tried to improve this basic strategy by introducing an additional value for each attribute in a cluster that represents the minimum over the free connections of all publishers associated with the attribute and we called this extension strategy 3.1. By examining these values it is possible to take certain attributes out of a cluster despite the fact that there is a publisher connected, which has no more free connections and therefore can not connect to an additional cluster. The introduction of this finer granularity results in a higher number of clusters, because we are now able to split clusters as long as there is an attribute with an associated value greater than 0. For the third strategy there where a lot of cases where we just didn't split the cluster, so here lies the improvement. The better use of the connections in this extension of strategy 3 also results in a better false positive rate. The only thing that changed for strategy 3.1, as could be seen by regarding the results, was the fact that for the extended strategy the random variant

performs better and with regard to the number of created clusters in our scenario it produced 30 percent more clusters than the descending variant, but the overall false positive rate was too high. This could be explained by examining the distribution of the clusters regarding their size and we saw that there were a few too big clusters causing a lot of false positives.
If we now think about further improvements, in general the goal should be to create as many clusters as possible (by obeying to the maximum number of connections for each publisher). This can only be done by exploiting all available connections of the publishers, so for an optimal case all connections are used and as the log file shows, there might be some more room, because there are publishers with a noticeable amount of leftover connections. The only problem are certain publishers that, to make it clear, have for example only one connection in total, what implies that all their attributes must be in one cluster.
Another improvement could be to look at the set of publishers that share attributes with the joining publisher and to reorder the clusters within this set or group of publishers, because that way the connections could also be restructured and perhaps better used than before.

# Bibliography

[1] Carzaniga, Antonio ; Rosenblum, David S. ; Wolf, Alexander L.: Design and Evaluation of a Wide-Area Event Notification Service. In: *ACM Transactions on Computer Systems* 19 (2001), August, Nr. 3, S. 332–383

[2] Papaemmanouil, Olga ; Çetintemel, Uğur: *SemCast: Semantic Multicast for Content-based Data Dissemination.* 2005

[3] Riabov, Anton ; Liu, Zhen ; Wolf, Joel ; Yu, Philipp ; Zhang, Li: *Clustering Algorithms for Content-Based Publication-Subscription Systems.* 2002

[4] Spyros, Voulgaris ; Gavidia, Daniela ; van Steen, Maarten: Cyclon: Inexpensive membership management for unstructured P2P overlays. In: *J. Network Syst. Manage* (2005)

[5] Tariq, M. A. ; Koldehofe, Boris ; Koch, Gerald G. ; Rothermel, Kurt: Providing Probabilistic Latency Bounds for Dynamic Publish/Subscribe Systems. In: *KiVS*, 2009, S. 155–166

## Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben.

Unterschrift:

Offenau, den 2. Mai 2010