Incremental Topology Transformation for Publish/Subscribe Systems Using Integer Programming

Pooya Salehi*, Kaiwen Zhang†, Hans-Arno Jacobsen‡
Middleware Systems Research Group, Technical University of Munich, Germany
Email: *salehip@in.tum.de, †zhangk@cs.tum.edu, ‡jacobsen@eecg.toronto.edu

***Abstract*—Distributed overlay-based publish/subscribe systems provide a selective, scalable, and decentralized approach to data dissemination. Due to the dynamic communication flows between data producers and consumers, the overlay topology of such systems can become inefficient over time and therefore requires adaptation to the existing load. Existing studies propose algorithms to design overlay topologies which are optimized for specific workloads. However, the problem of generating a plan to incrementally transform the current topology to an optimized one has been largely ignored. In this paper, we present IPITT, an approach based on integer programming for the incremental topology transformation (ITT) problem. Given the current topology and a target topology, IPITT generates a transformation plan with a minimal number of steps in order to lessen service disruption. Furthermore, we introduce a plan execution mechanism and evaluate our approach on an existing publish/subscribe system. Based on our evaluation, IPITT can reduce plan computation time by a factor of 10 and generates plans with an execution time up to 55% shorter than those of existing approaches.**

***Index Terms*—Client-server systems; Middleware; Publish-subscribe; Integer linear programming;**

I. Introduction

Publish/subscribe (pub/sub) is widely used as a loosely coupled, asynchronous, and selective communication substrate for the dissemination of event data (publications) between information producers (publishers) and consumers (subscribers) [1]–[3]. The publish/subscribe system is in charge of matching and forwarding incoming publications against registered subscriptions in order to deliver data to interested subscribers. In practice, pub/sub is widely employed for high throughput and low latency data dissemination [4]–[7].

To raise scalability, pub/sub systems are often distributed, where the tasks of matching and forwarding publications to interested subscribers are divided among a network of pub/sub service providers (called brokers), collectively called the broker overlay network. In many systems, the topology of the overlay is organized as a connected tree of brokers [1]–[3], [8]. A routing protocol then dictates how publications traverse the overlay to reach the intended recipients.

In an overlay-based pub/sub system, the communication load for each broker is variable, since message flows between publishers and subscribers vary as subscriptions enter and leave the system. Hence, it possible for some publication to be delivered through a long path of brokers in the current topology, thereby raising the average resource consumption across the brokers and increasing the end-to-end delivery latency. In a large-scale environment, the publish/subscribe system will experience a diverse range of workloads and thus suffer temporary drops in performance when using a static topology. Therefore, brokers should dynamically be migrated to a new topology which provides efficient delivery paths for the current workload. For instance, the Google Pub/Sub System (GooPS), used for the integration of online services across distributed locations, performs a periodic evaluation of the topology quality [4]. If the current topology is deemed unsuitable under the current conditions, a new topology is generated which is optimized given the actual pub/sub workload.

In this paper, we focus on the problem of migrating brokers in an online manner. Modifying the topology while the pub/sub system is still active is a challenging task since the reconfiguration of the broker routing states can induce a temporary partitioning of the overlay network, thereby resulting in a loss of messages in transit during the migration period. A naive, but reliable approach to migration involves stopping all new operations, allowing all messages currently in transit to be delivered, before applying migration updates and resuming the service. While this does guarantee reliable message delivery, such an approach suffers from high delivery latencies during migration. In a scenario where the overlay is re-evaluated every few minutes, as in the GooPS case, the above approach incurs frequent disruptions of the pub/sub service.

There is thus a need to generate an efficient plan which migrates brokers from an initial topology (i.e., the current topology) into a goal topology (i.e., the optimized topology). We define a *transformation plan* as a sequence of steps that must be applied to incrementally transform the topology into the target form, while guaranteeing reliable message delivery during the execution of the entire plan, and minimizing disruption to the pub/sub service. This type of approach is called *Incremental Topology Transformation* (ITT) [9].

By leveraging this incremental approach, a pub/sub system can undergo constant topology changes while minimizing the impact of transformations as perceived by the clients (i.e., the publishers and subscribers). Furthermore, execution of an incremental transformation plan can always be partially halted, since every intermediate topology produced by the plan is valid and will guarantee reliable delivery.

Existing works address the problem of optimal overlay design for a particular workload [10], [11], or define primitive operations for reconfiguration of pub/sub topologies[1] [12]–[14]. However, the problem of generating a plan that transforms an initial topology to a goal topology using these operations has been largely ignored. Yoon *et al.* formulate the ITT problem as an automated planning problem [9]. However, they only provide heuristics to calculate a plan which may not be optimal. In other words, the plan is not guaranteed to contain the minimum number of steps needed to perform the transformation.

In this paper, we present IPITT: an integer programming-based (IP) approach to the ITT problem. IPITT calculates a sequence of steps that transform an initial topology to a goal topology. Executing each step of the generated plan results in a valid topology with correct routing information. Furthermore, the plan contains the minimum number of steps required to perform the transformation. IPITT uses an integer linear programming formulation of the automated planning problem to solve the ITT problem. An IP-based approach provides a formulation of the problem that is easy to extend in order to add new constraints to the solution, leverages existing highly optimized commercial IP solvers, and generates optimal transformation plans. The contribution of this paper are as follows:

1) We present an IP-based formulation of the ITT problem, which facilitates integration of custom plan constraints and enables existing IP solvers to generate optimal ITT plans.
2) We provide an ITT planner that calculates transformation plans using different operations while minimizing disruptions to the pub/sub system. Furthermore, we propose two heuristics to improve scalability of the ITT planner.
3) We describe a mechanism to perform and coordinate plan execution in order to prevent an incorrect routing state or invalid topology.
4) We evaluate a complete implementation of IPITT on an existing pub/sub system, demonstrate the scalability of the presented approach, and study the effect of ITT on the pub/sub system.

Section II follows with a survey of related works. Section III explains automated planning, the ITT problem, and its IP formulation in detail. Next, we present our proposed approach in Sections IV and V. Section VI presents our evaluation. Finally, we conclude in Section VII.

II. Related Work

In this section, we present the related work on overlay design, topology reconfiguration, and topology transformation.

A minimum cost topology minimizes the overall cost of routing messages in the overlay. Minimum cost topology construction is a NP-hard problem [15], [16]. Nonetheless, there exists many algorithms which construct overlay topologies given different constraints while minimizing routing cost [10], [15]–[17]. Elshqeirat *et al.* provide different heuristics to design minimum cost overlay topologies subject to reliability constraints [15]. Chen *et al.* present algorithms for designing topic-connected overlays with fast implementations [10]. Weighted overlay design incorporates underlay information and is used to design overlay topologies that maintain their performance in the presence of underlay changes [17]. These works are orthogonal to this paper and can be used to generate goal topologies which are then provided to IPITT for generating efficient transformation plans while minimizing disruptions.

Dynamic reconfiguration techniques for pub/sub systems [14], [18]–[20] allow brokers to exchange performance information and make local decisions to improve the overlay performance by changing their neighbors. These approaches focus on reducing delivery latency and minimize the routing cost in self-organizing overlays which are more common in peer-to-peer systems. In comparison, IPITT focuses on topology transformation of structured overlays. Furthermore, the combination of different overlay design algorithms and IPITT provides a general framework that allows topology maintenance of pub/sub systems subject to a wide range of constraints.

Changing the overlay topology of a running system requires clearly defined operations in order to prevent message loss, invalid routing state, and service disruption. Yoon *et al.* propose a set of operations used to transform an overlay while maintaining delivery guarantees [12]. Nitto *et al.* also propose operations for self-adaptive overlays [13]. While the proposed set of operations between these two works are similar, they provide different delivery guarantees and can handle different failure models. IPITT can produce plans using any operation (including those in the above sets) that can be defined in terms of preconditions and effects on the overlay. For example, the *swapLink* [13] and *shift* [12] operations are equivalent but with different delivery guarantees. Therefore, depending on the delivery guarantees that the plan must maintain, the corresponding operations can be used in our planner.

ZUpdate provides a plan-based approach to migrate the topology of SDN-based datacenter networks (DCN) [21]. Similar to IPITT, ZUpdate uses integer programming to find a transition plan, consisting of *zUpdate* operations, to migrate the topology of DCNs. However, the proposed operations and plans are specific to OpenFlow switches in DCNs. Furthermore, rather than migrating to a new topology, the focus of the work is updating the switches without overloading existing links when specific switches are down.

Yoon *et al.* address incremental topology transformation in pub/sub systems [9]. They formulate the ITT problem as an automated planning problem [22] and use existing planners [23], [24] to find a solution. However, as existing planners are not scalable, the authors propose heuristics for the ITT problem. Similarly, IPITT uses an automated planning formulation, but uses integer programming to solve the automated

[1] Throughout this paper we simply refer to these primitive reconfiguration operations as *operations*.

planning problem. The IP-based approach of IPITT leverages commercial solvers such as Gurobi [25] and CPLEX [26] to solve the problem. These solvers are heavily optimized and can efficiently exploit multicore and cluster architectures [27]. Furthermore, IPITT can customize the plan search by changing the objective function of the IP model or adding constraints to the plan search. Section VI presents experiments which compares our solution (IPITT) to the state of the art heuristic proposed by the aforementioned paper.

GooPS [4] is an internal pub/sub system used for online service integration at Google. GooPS uses a controller that periodically recomputes a cost-minimal tree based on the existing workload, network underlay, and utilization information that it collects. GooPS uses a central routing approach which relies on Chubby to maintain all the subscriptions and routing information. Routing updates resulting from overlay changes are addressed with versioning of the routing information on Chubby. Unlike GooPS, IPITT focuses on overlay-based pub/sub systems where the routing information is distributed across brokers.

III. Background

In this section, we first provide a brief background on overlay-based pub/sub systems. Then, we explain the (automated) planning problem, and our formulation of the ITT problem. Lastly, we describe an IP-based approach to solve automated planning problems.

A. Overlay-Based Pub/Sub

In an overlay-based pub/sub system, brokers are connected together in a topology. Clients can connect to brokers and publish data or subscribe to data. Subscriptions are routed through the overlay towards publishers and establish communication paths between publishers and interested subscribers. Publications are then routed across the network towards matching subscriptions using these established paths. Each broker knows only of its direct neighbors in the overlay and can route subscriptions and publications using its local routing information to the next hops.

In topic-based systems, each publication is associated to a topic (class) ID, and each client can subscribe to one or more topics and receive all publications published on these topics. In content-based systems, subscribers additionally specify filters defined over the publication contents. For example, a subscriber with the subscription $s : \{class = temp, value > 20\}$ receives only temperature publications that have a value higher than 20. Content-based pub/sub allows more fine-grained subscriptions which reduce bandwidth usage compared to that of topic-based systems.

B. Automated Planning

A planning problem Π is defined as a tuple $<P, \Sigma, O, s_0, g>$ where: P is a set of predicates used to describe the problem state, Σ is a set of objects, O is a set of operations, s_0 is an initial state and g is a goal state. Each predicate $p \in P$ and each operation $o \in O$ takes one or more objects $\sigma \in \Sigma$ as parameters. A proposition is a predicate $p \in P$ applied to a subset of objects $\Sigma' \subseteq \Sigma$. A state s is a set of true propositions that uniquely defines a problem state. An action a is an operation $o \in O$ applied to a subset of objects $\Sigma' \subset \Sigma$. Applying an action on a state $\gamma(s, a)$ transitions the state s to a new state s'.

Each operation $o \in O$ is defined using three sets of predicates $pre(o), add(o), del(o)$. $pre(o)$ is the set of preconditions of operation o. For an action a to be applicable to a state s, s must satisfy all preconditions of a. $add(o)$ and $del(o)$ are the sets of predicates which are added to or removed from the state as a result of applying an action to s. Therefore, each action a changes the state s by changing the set of propositions defining s.

A solution to the planning problem Π is a sequence of actions $(a_1, a_2, \ldots, a_k)$ that corresponds to a set of state transitions $(s_0, s_1, \ldots, s_k)$ such that $s_1 = \gamma(s_0, a_1), \ldots, s_k = \gamma(s_{k-1}, a_k)$, where s_k is a goal state [22]. The synthesized sequence of actions that transitions the initial state to the goal state is called a plan.

C. ITT as Automated Planning

In order to define the ITT problem as a planning problem, we need to define the set of predicates, objects, and operations of an ITT problem. In this paper, we adopt the same ITT formulation as presented by Yoon *et al.* [9].

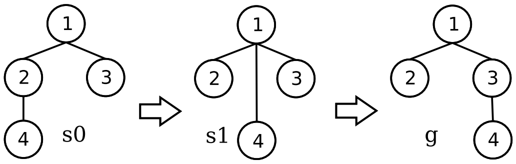


Fig. 1: Initial and goal topology

The set of objects, Σ, is the set of brokers comprising the overlay (e.g., in Figure 1, $\Sigma = \{1, 2, 3, 4\}$). We use the following predicates to uniquely describe any ITT problem state: $P = \{conn(i, j), rem(i, j), eq(i, j)\}$. $conn(i, j)$ is a predicate indicating whether there is a direct link between brokers i and j in the topology. In order to define all existing links in the topology, we use $conn$ propositions. In Figure 1, $C = \{conn(1, 2), conn(1, 3), conn(2, 4)\}$. In an overlay with undirected links, $conn(i, j) = T \implies conn(j, i) = T$.

$rem(i, j)$ indicates whether the link between i and j can be removed (i.e., (i, j) is a removable link). During the topology transformation, any link which is not part of the goal topology is removable. In Figure 1, the link $(2, 4)$ is not part of the goal topology, therefore $rem(2, 4) = T$. Lastly, $eq(i, j)$ indicates whether brokers i and j are identical. There exists one $eq(i, j)$ proposition for each broker in the overlay. Therefore, using the defined P and Σ, we can describe the initial and goal state of Figure 1 as the following set of true propositions: $s_0 = \{conn(1, 2), conn(1, 3), conn(2, 4), rem(2, 4), eq(1, 1), eq(2, 2), eq(3, 3), eq(4, 4)\}$ and $g = \{conn(1, 2), conn(1, 3), conn(3, 4), eq(1, 1), eq(2, 2), eq(3, 3), eq(4, 4)\}$.

In order to transform any initial topology to any given goal topology, we require the following operations: *append(i, j)*, *detach(i, j)*, and *shift(i, j, k)*. These operations have been shown to safely enable any arbitrary transformation [12] [28]. *append* creates a new broker i and attaches it to the broker j. *detach* removes broker i from the overlay by disconnecting it from its neighbor j. *shift(i, j, k)* removes the link between i and j and establishes a link between i and k. These three operations are defined as follows [9]:

$$append(i,j):\begin{cases} pre(O)=\{\neg eq(i,j),\neg conn(i,j)\} \\ add(O)=\{eq(i,i),conn(i,j)\} \\ del(O)=\{\} \end{cases}$$

$$detach(i,j):\begin{cases} pre(O)=\{\neg eq(i,j),conn(i,j),rem(i,j)\} \\ add(O)=\{\} \\ del(O)=\{eq(i,i),conn(i,j),rem(i,j)\} \end{cases}$$

$$shift(i,j,k):\begin{cases} pre(O)=\{conn(i,j),conn(j,k),rem(i,j), \\ \qquad\qquad \neg eq(i,j),\neg eq(j,k)\} \\ add(O)=\{conn(i,k), \\ \qquad\qquad rem(i,k) \text{ if } (i,k)\notin g\} \\ del(O)=\{conn(i,j),rem(i,j)\} \end{cases}$$

For example, in Figure 1, *shift(4, 2, 1)* is applicable to the initial topology because there is a link between $(4,2)$ ($conn(4,2)$) and $(2,1)$ ($conn(2,1)$) and the link $(4,2)$ is removable ($rem(4,2)$). Applying this action results in: $s_1 = \{conn(1,2), conn(1,3), conn(1,4), rem(1,4), eq(1,1), eq(2,2), eq(3,3), eq(4,4)\}$. The link $(1,4)$ is required, because only after transitioning the overlay to the state s_1, can we apply *shift(4, 1, 3)* to reach the goal topology.

Note that *shift* requires three distinct brokers. Therefore, we need *eq(i, j)* in order to represent the existence of a broker in the overlay and prevent the generation of invalid *shift* actions such as *shift*(1, 1, 2).

In order to prevent message loss, we use the *synchronous shift protocol* [12]. In this protocol, i, j and k first start buffering publications and then perform changes on their routing tables. Once all participants of a *shift* have finished updating their routing tables, processing of publications (including buffered publications) is then resumed.

D. Integer Programming for Automated Planning

There exists general planners that can be used to solve any planning problem encoded using the Planning Domain Definition Language (PDDL) [23], [24], [29]. Yoon *et al.* have studied the applicability of such general planners to the ITT problem [9]. However, these planners fail to scale to overlays of 50 or more brokers. This is due to the fact that, given the PDDL encoding of the ITT problem, the general planners consider all possible actions. This can result in a search space of $O(n^3)$ for all combinations of *shift(i, j, k)*.

In this paper, we adopt an integer programming approach to planning for the following reasons:

- An IP-based approach specifies the set of actions (A) that the planner should consider. Consequently, we can reduce the search space by providing a smaller set of actions to the planner rather than all possible actions.
- It has been shown that IP is more general than propositional logic and allows for a more concise representation of constraints [30]. This results in a rich modeling formalism which enables customization of the plan search [31].
- An IP-based approach lends itself to being used with existing commercial solvers. Due the wide applicability of IP, such solvers are highly optimized and can even be easily configured to utilize computer clusters [27], [32].

We use the general IP approach for solving planning problems as presented in *OptiPlan*, and customize it by defining ITT-specific predicates and operations to create an ITT planner [31], [33]. To formulate the planning problem of transforming an initial state to a goal state in T steps, we define the binary variable $x_{a,t} = 1$ if action a is executed in step t and 0 otherwise. Using the following objective function, the IP formulation finds a plan that minimizes the number of actions required to perform the transformation:

$$\min \sum_{a\in A} \sum_{t\in\{1,\ldots,T\}} x_{a,t}$$

Furthermore, the formulation defines other binary variables and constraints that link actions and propositions in each planning step, guaranteeing mutual exclusion of the chosen actions, restricting parallel state changes, and ensuring backward chaining. Appendix A presents a more detailed explanation of the IP-based planning formulation.

IV. Incremental Topology Transformation Using Integer Programming

In this section, we first introduce the different components of the IPITT system. Next, we explain how our ITT planner functions and describe its IP model. Lastly, we explain how plans are deployed and executed.

A. System Overview

Figure 2 presents the different components of IPITT. We assume there exists a topology manager which knows the identity of all brokers in the overlay. The topology manager does not need to know the routing table nor subscriptions located on each broker, only the overlay topology. The four main components of IPITT are the overlay evaluator, ITT planner, plan deployer, and ITT agent.

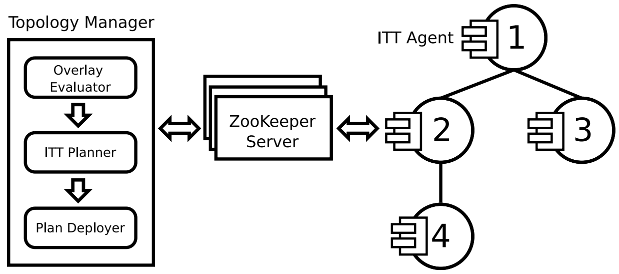


Fig. 2: IPITT components

The overlay evaluator is responsible for periodically evaluating the overlay performance and designing a new topology if necessary. The topology manager periodically provides the overlay evaluator with the current topology and performance metrics. If the current topology is deemed inefficient by the overlay evaluator, it calculates a new topology. Note that the focus of this paper is on the generation of a transformation plan given initial and goal topologies. The strategy taken to evaluate the current topology and design a new one is considered out of the scope of the paper. Works on designing the overlay and evaluating the efficiency of pub/sub and overlay-based systems are orthogonal to our work, and their techniques can be integrated in our system [4], [11], [17], [34].

If a new topology is returned by the overlay evaluator, IPITT calls the ITT planner with the current topology and the new topology. The ITT planner is responsible for finding a plan to perform the transformation. The plan calculation takes into consideration time requirements. In other words, a new plan must be returned before the next topology evaluation. The calculated plan is passed to the plan deployer which is responsible for executing the transformation on the overlay. The deployer ensures each step of the plan is performed successfully before moving to the next step, since each step may depend on intermediate links previously established in the plan. Furthermore, in order to prevent any message loss or an invalid topology, the steps and operations within each step need to be synchronized. To do so, the deployer communicates with the ITT agent located on each broker.

B. ITT Planner

The planner is responsible for finding a plan that transforms the initial topology to the goal topology using a minimum number of steps. Algorithm 1 shows the operation of the ITT planner. First, it generates the set of all actions to be considered for the plan. The generated set A does not describe any order between the actions but simply provides a superset of actions that can form a viable plan. In order to create the IP model of the ITT problem, the planner needs to determine the length T of the plan. This is required because the IP formulation of the planning problem requires the plan length to create a decision variable for each action in each step ($x_{a,t}$). In order to ensure a plan with a minimum number of steps, the planner starts with $T = 1$. In our implementation, we speed up the calculation of the algorithm by starting with a higher value of T. The minimum number of steps required to transform a topology is proportional to the topology size and the number of brokers facing a link change. Therefore, the result of previous plan calculations can be stored in a table where each entry records the starting value of T for a given topology size and percentage of overlay links that are changed in the goal topology.

Given the initial and goal topologies, the action set, and plan length, the planner creates the IP model of the ITT problem instance. This is done by first creating the set of all propositions $F = \{f \in pre(a) \mid a \in A\} \cup \{f \in add(a) \mid a \in A\} \cup \{f \in del(a) \mid a \in A\}$. The IP model consists of a set of binary variables and constraints. Besides creating a variable for each action on each step, we create five variables for each proposition for each step. These variables link the propositions to the actions in each step of the plan.

Algorithm 1 ITT planner

```
function PLANNER($s_0, g$)
    $A \leftarrow$ generateActions($s_0, g$)
    $T \leftarrow 1$
    do
        $M \leftarrow$ createModel($s_0, g, A, T$)
        $M$.optimize()                ▷ Pass to solver
        if $M$ is infeasible then
            $T \leftarrow T + 1$
    while $M$ is infeasible
    plan $\leftarrow$ extractSteps($M$)
    return plan
```

After adding constraints over the defined variables, the IP model is given to the IP solver. If it is not possible to find a plan with the input T, the solver determines that the model is infeasible. In this case, we increment T, create a new model, and pass it to the solver. Once the solver returns the optimized model, the planner extracts the plan based on the values of the $x_{a,t}$ variables. The result is a sequence of steps, where each step contains one or more actions. For example, the plan for the ITT problem in Figure 1 is the following:

```
1:[shift(4,2,1)], 2:[shift(4,1,3)]
```

In this case, a plan in one step is not possible since an intermediate state with the link $(1, 4)$ is required to achieve the goal topology.

The size of the IP model, and consequently the time required to solve it, is related to the number of variables and constraints defined in the model. Since the defined number of variables and constraints for each proposition and action is fixed, decreasing the number of actions is the only way to reduce the total number of variables and shorten the time required to find a plan. Since *append* and *detach* can only add or remove leaf brokers [12]–[14], finding the set of *append*s and *detach*es is trivial. First, all brokers which will be removed from the overlay are added as leaf brokers to the goal topology. Next, any broker which will be added to the overlay, are added as leaves to the initial topology. Consequently, both the initial and goal topologies have the same set of brokers. Once a plan is calculated, the set of *append*s is added to the beginning of the plan and the set of *detach*es are added to the end of the plan. We separate calculation of the set of *append* and *detach* operations in order to reduce the IP model size which does not impact the correctness of the planner. Therefore, we consider only initial and goal topologies which have the same set of brokers and focus on generating a set of *shift* actions. Due to this property, there is no need to use the $eq(i, j)$ predicate, since we consider only valid *shift* actions (where i, j, and k are distinct brokers). Excluding *append* and *detach* from the IP formulation reduces the model size and hence the time required to find the plan. Note that migrating clients connected to a removed broker is out of the scope of this paper and is addressed in client mobility studies [35], [36].

A naive approach to generate the set A is to consider all possible subsets of size 3 of the brokers. This results in

$P_{n,3} = n!/(n-3)! \approx O(n^3)$ actions, where n is the overlay size. In medium to large overlay sizes (e.g., more than 40 brokers), this results in a very large action set. Although generating all possible actions guarantees that the optimal solution can be found, the resulting set can include *shift* actions which involve brokers facing no change in the transformation. Based on this observation, we propose two heuristics which can considerably reduce the size of the generated action set.

The first heuristic, shown in Algorithm 2, starts by identifying links that must be established in the goal topology. For example, in Figure 3, $(1,3)$ and $(6,9)$ are removable links and $(4,7)$ and $(8,9)$ are goal links. Next, for each goal link, the path between its source and destination in the initial topology is calculated. We call these *transformation paths*, highlighted in red in Figure 3. The intuition behind this heuristic is that any *shift* action that can contribute to the transformation of the initial topology to the goal topology must involve brokers located on these paths. Finally, at line 7, we generate all possible actions that can happen among the set of brokers located on a transformation path.

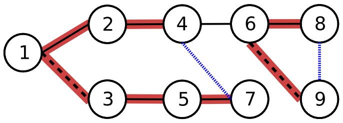


Fig. 3: Transformation path

Algorithm 2 Generate actions for all paths

```
1: function GENERATEACTIONSFORALLPATHS($s_0, g$)
2:     goalLinks ← { $\ell \in g \mid \ell \notin s_0$ }
3:     brokers ← {}
4:     for $\ell \in$ goalLinks do
5:         path ← getPath($\ell_{src}$, $\ell_{dest}$, $s_0$)
6:         brokers ← brokers ∪ path
7:     actions ← perm(brokers, 3)
8:     return actions
```

The size of the action set generated by this heuristic (Algorithm 2) is no longer a function of the overlay size, but rather a function of the number of brokers located on a transformation path. Therefore, this algorithm takes into account the size of the set of brokers which are affected by the transformation. Furthermore, in an ITT problem where brokers affected by the transformation are located close to each other, the generated set of actions is smaller. Algorithm 3 further reduces the search space by considering each transformation path separately. Therefore, rather than first collecting all affected brokers, the algorithm computes all possible *shift* actions for each path.

The main difference between Algorithms 2 and 3 is that the former considers any *shift* action that can happen among different transformation paths. For example, in Figure 3, the set of actions generated by Algorithm 3 does not include any *shift* among brokers 4, 6 and 7 because these three brokers are located on two different transformation paths. In contrast, Algorithm 2 generates the *shift*s among these three brokers as well. This may result in shorter sequence of actions, if they exist. Nonetheless, in scenarios where two different parts of the overlay undergo transformation (e.g., Figure 3), this can generate unnecessary actions. Algorithm 3 ensures that only *shift* actions among brokers located on at least one transformation path are considered. Since there exists always at least one sequence of *shift* actions to transform a removable link located on the transformation path of a goal link, Algorithm 3 generates an action set which contains enough actions to create at least one viable plan.

Algorithm 3 Generate actions for each path

```
function GENERATEACTIONSFOREACHPATH($s_0, g$)
    goalLinks ← { $\ell \in g \mid \ell \notin s_0$ }
    actions ← {}
    for $\ell \in$ goalLinks do
        path ← getPath($\ell_{src}$, $\ell_{dest}$, $s_0$)
        actions ← actions ∪ perm(path, 3)
    return actions
```

C. Plan Deployment

The task of the plan deployer is to execute a given plan on the topology. Furthermore, the plan deployment must be synchronized and performed step-by-step, in order to prevent invalid routing states and an incorrect topology. Each broker has an ITT agent which is listening for *shift* actions, performing them on its host broker, and reporting back. The plan deployer uses ZooKeeper to communicate with the ITT agents, to deploy a plan, and to coordinate its execution [37]. ZooKeeper allows complete decoupling of the plan deployer from the pub/sub system and is scalable to large overlays. Furthermore, it facilitates reactions to plan execution failures. Figure 4 shows the ZooKeeper tree used by IPITT.

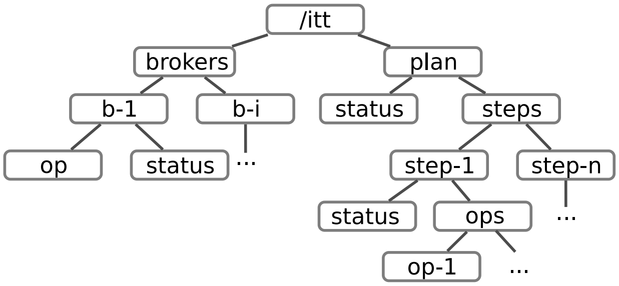


Fig. 4: IPITT ZooKeeper tree

Upon receiving a plan to execute, the deployer writes the plan to */itt/plan/steps*. Each step in the plan is mapped to a node in ZooKeeper and each action is mapped to a child node of its step. Furthermore, each broker is mapped to a node. Upon broker start, each ITT agent establishes a watcher on */itt/brokers/brokerID*, waiting for *shift* actions to be issued by the deployer. In order to execute a plan step, for each step action, the deployer writes the *shift* action to the data node of each involved broker and creates watchers on the *status* node of each broker. Upon receiving a *shift* command, the ITT agents trigger a shift by directly contacting other involved brokers and exchanging routing table entries if necessary and changing overlay links. After successful completion of the *shift* command, each ITT agent updates the status of the received *shift* action on ZooKeeper as *successful*.

In case of failure or timeout on any broker, the *shift* action is aborted and the status of the corresponding broker is set to *error*. Once all ITT agents have updated the status of the assigned action, the deployer updates the status of the *shift* action in the ZooKeeper tree. The status of a step is updated to successful only if all actions for that step have completed successfully. The deployer moves on to the next step for execution only if the previous step is successful. If a step fails and cannot be finished within a timeout, the plan execution is aborted. Aborting the plan is necessary in order to prevent invalid routing information or an invalid topology from occurring if further steps are taken. However, since *shift* actions of each step are independent from each other, a step can be partially successful without resulting in an incorrect state. Furthermore, by keeping the plan state on ZooKeeper, plan execution can recover from the failures of the deployer or the topology manager. Using ZooKeeper allows the ITT agents to use the ZooKeeper tree to send back statistics about the broker. This information can be used as input to evaluate whether it is necessary to create a new topology. This allows IPITT to create a feedback loop between the overlay and the topology manager.

While the deployer ensures fault tolerance during plan execution, it relies on the reliability of the operations to ensure fault tolerance during the execution of each action. IPITT requires each action to either succeed or fail without leaving the local routing table and links invalid. Consequently, any intermediate state of the topology during plan execution is valid and capable of correct routing of publications. Existing operations assume different fault models [12]–[14] which are applicable in different scenarios.

V. IPITT as a Framework

IPITT can be used as a framework for the incremental topology transformation of pub/sub systems. The ITT planner is able to find a plan consisting of operations that can be defined in terms of precondition, add, and delete effects on the overlay state. The choice of transformation operations to use in the plan depends on the guarantees that the transformation must maintain. While *SwapLink* [13] (or *shift*) performs small changes which only involve three brokers, there exists composite operations that perform larger transformations in one step. *Move* [12] and *LinkExchange* [14] are two examples of operations which directly replace a removable link with a goal link. Compared to *shift*, these operations require synchronization among a larger number of brokers. Thus, they require a longer time to execute and have a larger impact on the clients. Nonetheless, in scenarios where the larger delay is tolerable, they can be used for building a plan.

The IP formulation of ITT allows changing the objective function to define a new cost model for planning. In this paper, we assume all actions have an equal cost. Therefore, we minimize the number of actions to improve plan quality. However, it is possible to assign each action a cost depending on the involved brokers and find a plan which minimizes the total cost of the plan. If each action a has a cost C_a, the objective function minimizing planning cost is $min \sum_{a \in A} \sum_{t \in \{1,\dots,T\}} C_a \times x_{a,t}$.

It is also possible to define new constraints for the plan. For example, if we want to limit the maximum number of links of each broker during the transformation, we can define a variable $D_{b,t}$ which is the degree of broker b in step t of the plan. The maximum degree can be defined using the set of constraints $D_{b,t} \leq D_{max}\ ,\ \forall\ b \in B$, where B is the set of overlay brokers. Furthermore, the *add* and *delete* effect of the operations must be extended to consider $D_{b,t}$. Lastly, the ITT planner and deployer can be used in combination with any overlay-based pub/sub system, as long as each broker implements an ITT agent which can access the ZooKeeper tree of IPITT and can execute actions on the host broker.

VI. Evaluation

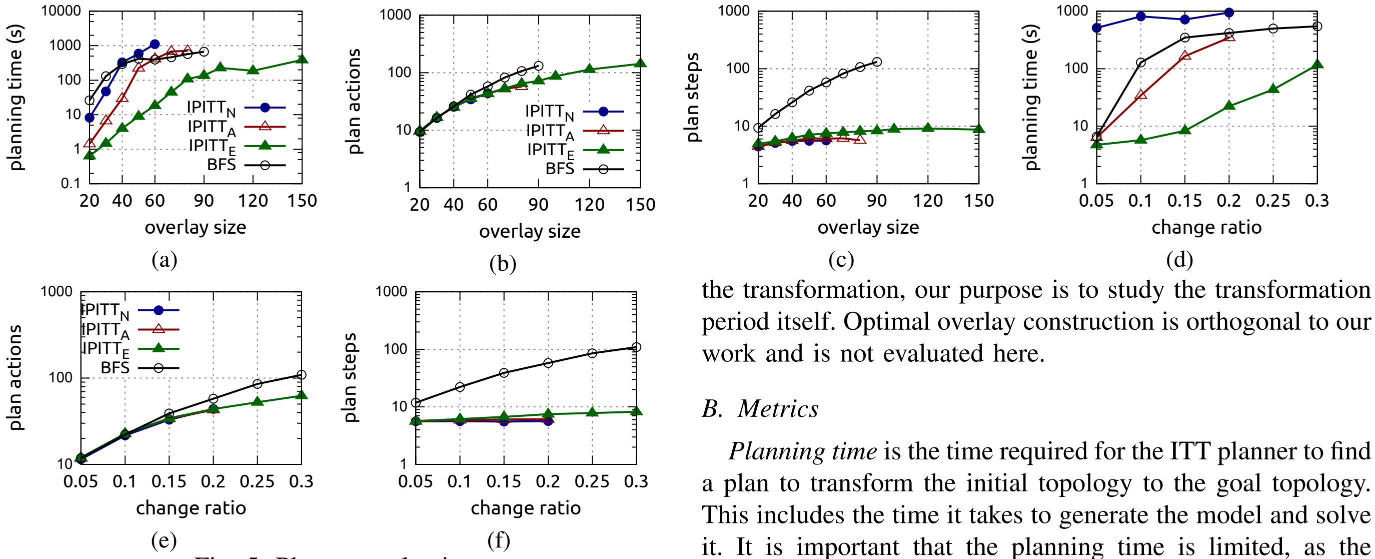
In this section, we evaluate IPITT operating with a working pub/sub system. We implemented our ITT planner using Python and the Gurobi Optimizer [25], and implemented the *shift* operation [12] and ITT agent in Java on the PADRES pub/sub system [1]. We use a cluster of 20 Dell PowerEdge R430 servers for our evaluation.

Note that PADRES uses the content-based subscription model, which allows fine-grained filters to be expressed. Although IPITT is not tied to any subscription model, we choose to evaluate our approach using a content-based system since it generates larger routing tables. Consequently, broker migration incurs more overhead and is therefore more challenging for ITT, which performs frequent routing table updates. Furthermore, we compare our approach with the best-first search (*BFS*) heuristic to solve ITT, presented by Yoon *et al.* [9]. *BFS* uses an algorithm similar to those underlying state of the art planners, utilizes domain specific heuristics, and avoids generating all actions upfront. *BFS* models the planning problem as a graph with each node representing a different state of the problem (i.e., a set of propositions) and each edge representing an action. Given a time limit and depth limit, *BFS* searches for a plan and returns the best result (lowest number of actions). The depth limit stops the heuristic from following a branch of the graph for an excessive period of time. Upon reaching the depth limit, *BFS* starts a new search on a different part of the graph. This is repeated until the time limit is reached.

A. Workload

We evaluate IPITT using acyclic hierarchical topologies which are common in high-throughput pub/sub systems [3], [38]. The generated tree topologies used for the evaluation have a maximum node degree of 5, 6 or 7 and each have two balanced and unbalanced variations. The degree of each broker is selected based on a random uniform distribution and the maximum degree of the tree. Each topology has two variations generated with different seeds. The link latencies are between 10 and 50 ms and selected based on an uniform distribution. We run our experiments using 12 topologies covering a wide variety of trees.

We evaluate IPITT and *BFS* using a synthesized workload with the following parameters: We use 20 publishers all



(a)

(b)

(c)

(d)

(e)

(f)

Fig. 5: Planner evaluation

connected to the root of the tree topology each publishing 5 publications per second. The content space has 10 different class of publications with two attributes each between 0 and 1000. Each publisher generates publications based on an uniform distribution from the content space. Each subscriber subscribes to one class of publication with the popularity of classes based on a Zipf distribution with $a = 1.1$ [39] and the attribute range of each subscription based on a random uniform distribution. Subscribers choose their edge brokers based on a random uniform distribution among the leaf brokers of the overlay. Clients are located on different VMs than the brokers, in groups of at most 80 clients per VM. Publishers publish messages for 10 minutes. Starting from the 120^{th} second of each experiment, an incremental topology transformation is performed on the overlay. Each experiment is run with 12 different topologies and 2 different workloads (24 runs) and the metric values are averaged.

We evaluate the impact of three variables on our metrics. *Overlay size* is the number of brokers in the overlay and *change ratio* is the percentage of links changed in the goal topology. These variables influence the IP model size and the size of the generated plan in terms of number of actions and steps. Furthermore, we change the number of *subscribers* to study the scalability of plan execution in terms of client size.

Approach	Mem. (MB)	Actions	Variables	Constraints
IPITT$_N$	11 352	205 320	1 355 820	3 810 515
IPITT$_A$	2 750	35 904	392 928	919 799
IPITT$_E$	405	1 860	154 620	211 973
BFS	230	-	-	-

TABLE I: ITT planner memory footprint and model size

In order to be able to control the change ratio between the goal and initial topologies, we do not employ an overlay design algorithm. The goal topology is calculated by changing N% of the links in the initial topology. While a randomly generated topology may degrade the performance of the overlay after the transformation, our purpose is to study the transformation period itself. Optimal overlay construction is orthogonal to our work and is not evaluated here.

B. Metrics

Planning time is the time required for the ITT planner to find a plan to transform the initial topology to the goal topology. This includes the time it takes to generate the model and solve it. It is important that the planning time is limited, as the objective is to perform transformations periodically. Systems such as GooPS perform re-evaluation and transformations frequently, requiring a planning time in order of minutes [4].

Plan quality is defined in terms of the number of steps and number of actions in the plan. A high quality plan minimizes disruption to the pub/sub system. Since the number of link changes directly influences service disruption in the overlay [34], we consider a plan with a minimal number of actions as high quality. Furthermore, a higher number of steps can result in a longer plan execution time which can impact service disruption.

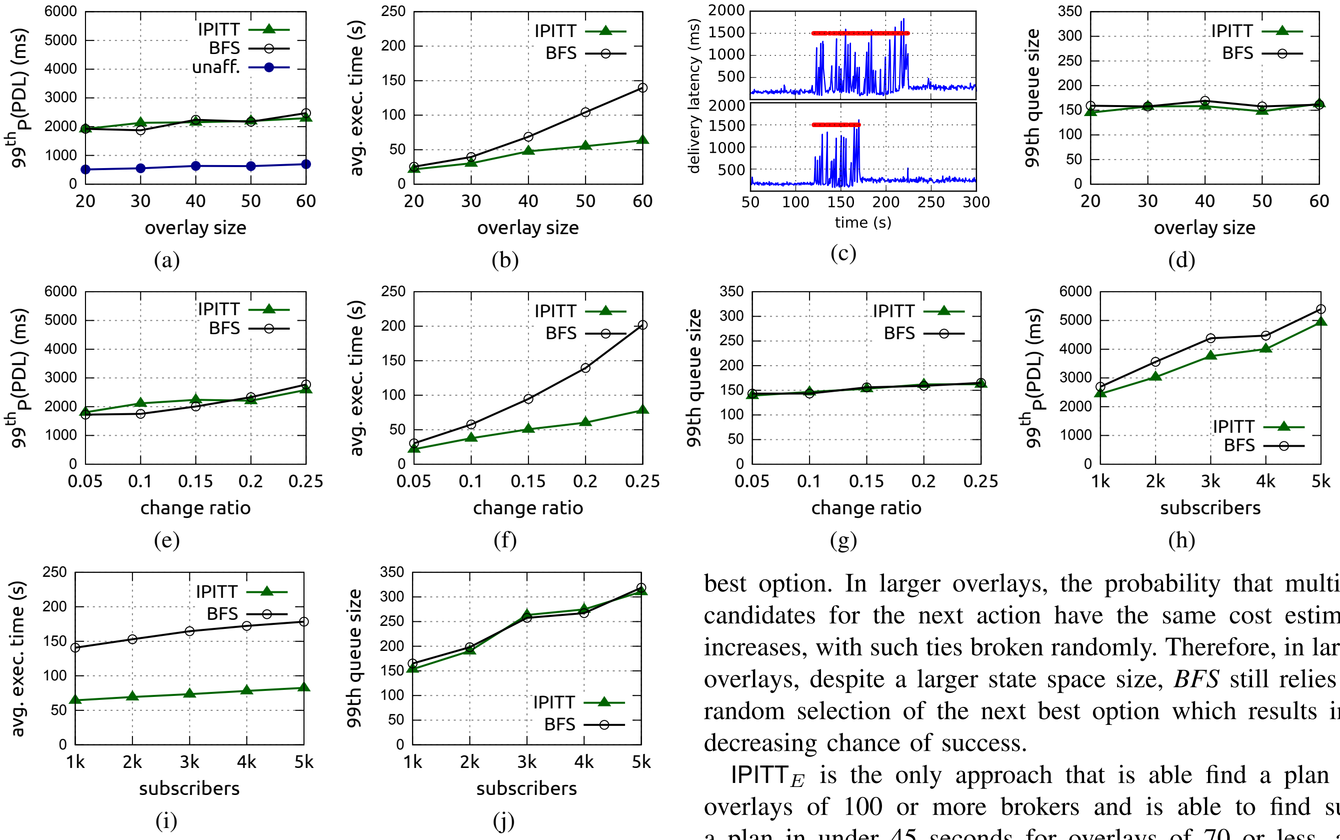
Publication delivery latency is the time it takes to deliver a publication to a matching subscriber. We collect all delivery latencies from subscribers and calculate the 99^{th} percentile of publication delivery latencies (99^{th}_{P}(PDL)) of all publications which are in transit during the topology transformation and are therefore affected by it. This metric represents the effect of the transformation on the clients of the pub/sub system.

Broker queue size is the growth of the message queue size of brokers during transformation. Performing each *shift* action requires the buffering of messages until the *shift* is finished. This metric represents the buffering load and therefore the impact of the transformation on the overlay brokers. We measure queue size of each broker at the beginning and end of each *shift*, calculate the difference and take the 99^{th} percentile of the collected values.

Plan execution time is the time it takes to completely execute a transformation plan computed by the planner on the overlay. This metric represents the duration of service disruption and its effect on the clients.

C. Experiments

Impact of overlay size on planning: In this experiment, we study the impact of overlay size on the planning time and plan quality. We change the *overlay size* from 20 to 150 brokers, which is in line with the overlay size of systems such as GooPS [4], with a change ratio of 20%. This means each goal topology is generated by randomly changing 20% of



(a)

(b)

(c)

(d)

(e)

(f)

(g)

(h)

(i)

(j)

Fig. 6: Plan execution evaluation

the links in the initial topology. We evaluate IPITT using the first heuristic which generates actions for all paths (IPITT$_A$), the second heuristic which generates actions for each path separately (IPITT$_E$), naive IPITT (IPITT$_N$) which generates all possible actions, and *BFS* with a time limit of 10 minutes.

Figure 5a shows that increasing the number of brokers in the overlay results in a longer planning time. This is inevitable as a larger number of brokers results in a larger IP model for IPITT and a larger state space graph for *BFS*. Nonetheless, IPITT$_E$ results in the lowest planning time for all overlay sizes. IPITT$_N$ is able to find a plan only for overlays of size 50 or less and it cannot find a plan for overlays with more than 60 brokers. The reason is that the model size and consequently the planning time exhibits a polynomial growth (n^3). IPITT$_A$ shows a similar result but with a slower growth rate. However, since increasing the overlay size results in a larger number of brokers located on a transformation path and IPITT$_A$ generates all actions for all such brokers, the planning time increases polynomially. IPITT$_A$ can find a valid plan for overlays of size 60 or smaller in less than 10 minutes and is not able to find a plan for overlays of size larger than 80.

BFS performs similarly to IPITT$_A$ and outperforms IPITT$_A$ for overlay sizes of 60 or more. Furthermore, *BFS* can find a plan for overlays of up to 90 brokers but not beyond that. The reason can be attributed to how it searches the state space graph. *BFS* identifies the next action based on the estimated cost and follows this branch until it reaches the specified limit. Upon reaching the end, it starts over from the second best option. In larger overlays, the probability that multiple candidates for the next action have the same cost estimate increases, with such ties broken randomly. Therefore, in larger overlays, despite a larger state space size, *BFS* still relies on random selection of the next best option which results in a decreasing chance of success.

IPITT$_E$ is the only approach that is able find a plan for overlays of 100 or more brokers and is able to find such a plan in under 45 seconds for overlays of 70 or less, and under 400 seconds for overlays of size 150. The reason for the scalability of IPITT$_E$ is that the action set generation does not grow polynomially with the overlay size since IPITT$_E$ generates actions for each path separately. Therefore, compared to *BFS*, IPITT$_E$ reduces the planning time for overlays of size 50 or less by a factor of 61 and for overlays of size 60 or more by a factor of 10. Compared to IPITT$_N$, IPITT$_E$ reduces the planning time by a factor of up to 82 times. However, our current heuristic is not able to find a plan for overlays of size 200 or more in under 10 minutes. While we have not performed any optimization for our evaluation, there exists techniques such as warm starting of the IP solver that can improve the overall performance of the IP solver [40]. Furthermore, IPITT$_E$ uses a simple heuristic to limit the search space which can be further extended to provide a smaller action set.

Table I shows the memory footprint and IP model size of the different approaches for an overlay size of 60 and change ratio of 20%. Since the IP formulation creates several variables and constraints for each action, reducing the number of shift actions greatly reduces the IP model size. Consequently, the solver can find a solution faster and requires less memory.

Figures 5b and 5c show the quality of the plan calculated by each approach in terms of number of actions and number of steps. For overlays of size 40 or more, IPITT$_E$ can find a plan with up to 45% fewer actions compared to *BFS*. Furthermore, compared to IPITT$_N$ and IPITT$_A$ which can find an optimal or near optimal plan due to their larger search space, IPITT$_E$ calculates a plan with only up to 5% and 12%

more actions, respectively. *BFS* does not parallelize the plan actions: Therefore, the number of steps is equal to the number of actions in the plan. In contrast, the IP approach produces a plan with the lowest possible number of steps. However, due to the use of heuristics to reduce the search space size, only IPITT_N guarantees that no plan with a lower number of steps is possible. In comparison to the optimal plan, IPITT_E produces plans with up to 32% higher number of steps. The increase in IPITT_A is at most 11%. This means that generating an action set which includes *shift*s across different transformation paths can result in more efficient plans.

Impact of change ratio on planning: In this experiment, we study the impact of the *change ratio* on planning time and plan quality. We increase the change ratio from 5% to 30% in an overlay of 60 brokers. With frequent evaluation of the overlay, high change ratios are in practice unlikely. Figure 5d shows the planning time of each approach. In all cases, the planning time increases because a higher change ratio results in a larger IP model or space state graph. A higher change ratio requires a higher number of actions and a larger plan, which increases T in the IP model and graph depth in *BFS*. Due to the larger problem size, IPITT_N and IPITT_A are not able to find a plan for change ratios of 25% or higher. *BFS* can find a plan for 30% change ratio in 546 seconds and IPITT_E in 117 seconds. Although *BFS* does not return a plan until the time limit is reached, in this experiment, we have recorded the time that *BFS* finds the best plan. Nonetheless, IPITT_E can reduce planning time by up to 95% compared to that of *BFS*. Regarding plan quality, similar to previous experiment, IP-based approaches can outperform *BFS*. While for change ratios of 5% and 10%, IPITT_E and *BFS* produce plans with the same number of actions, in larger change ratios, IPITT_E results in plans with up to 42% less actions. In comparison to the optimal plans calculated by IPITT_N, IPITT_E produces plans with up to 5% more actions and up to 32% more steps.

These two experiments confirm the effectiveness of our two proposed heuristics in reducing the search space size of the ITT problem and scaling the ITT planner to overlays of up to 150 brokers. Furthermore, we have shown the different levels of trade-off that the three IP-based approaches can provide.

In the next two experiments, we use the calculated plans by *BFS* and IPITT_E which we simply refer to as IPITT.

Impact of ITT on clients: In this experiment, we study the impact of ITT on the subscribers of the pub/sub system. Figure 6a shows the impact of increasing the overlay size on 99^{th}_P(PDL) with 1000 subscribers and a change ratio of 20%. Since each action forces temporary publication buffering by the involved brokers, subscribers receiving publications which traverse brokers involved in a *shift* operation experience a higher delivery latency. IPITT and *BFS* show similar affected latencies which are up to 2 seconds more than the 99^{th}_P(PDL) for publications not affected by the transformation. The impact of transformation on latency is similar in both approaches since it primarily depends on the type of operation used, which is the same (*shift*). In all cases, increasing the overlay size results in a higher latency since larger overlays have a longer path between publishers and subscribers.

Figure 6b shows the average plan execution time (M_{PET}) of each approach. Since IPITT produces plans with fewer steps, and actions in each step are executed in parallel, the M_{PET} of IPITT is up to 55% shorter than that of *BFS*. While the execution time does not affect the 99^{th}_P(PDL) (this is affected by how long the operation buffers publications), it does influence the number of messages that have a higher delivery latency. Figure 6c shows the delivery latency of delivered publications to the same client in two different scenarios. In the lower section, the transformation uses the plan generated by IPITT and the upper section is the delivery latencies during the execution of a *BFS* plan. The red dots on top of each plot show the time taken by *shift* actions executed on the overlay. While in both approaches the plan execution period results in an increase in delivery latency, higher number of publications face this increase in *BFS*. The difference in the number of affected messages is proportional to the plan execution time (Figure 6b). This experiment confirms that IPITT is able to minimize disruption to publication delivery by finding a plan with a minimal number of steps.

Figure 6e shows the impact of increasing the change ratio on the 99^{th}_P(PDL) in an overlay of 60 brokers with 1000 subscribers. A higher change ratio requires a plan with a larger number of actions and steps. Consequently, publications affected by the transformation are more likely to encounter more than one action execution and be buffered more than once while being routed. Therefore, both approaches show an increase in 99^{th}_P(PDL) of up to 2.2 seconds. However, similar to the previous experiment, since the M_{PET} of IPITT is up to 60% shorter than that of *BFS* (Figure 6f), a smaller number of publications are affected by this disruption.

Figure 6h shows that increasing the number of subscribers in an overlay of 60 brokers with 20% change ratio results in a higher 99^{th}_P(PDL). The reason is that a larger volume of subscriptions in the system leads to a larger routing table size at the brokers. Since *shift* actions perform changes on the routing table of each involved broker, greater table sizes result in longer latencies. For the same reason, the M_{PET} of each approach grows with an increase in subscribers. However, both 99^{th}_P(PDL) and M_{PET} grow sublinearly and therefore the incremental transformation is scalable with respect to the number of subscribers. This increase depends on the type and implementation of the operation. In this case, it affects both approaches similarly.

These experiments confirm that IPITT provides a scalable approach to incremental topology transformation while minimizing disruption to clients.

Impact of ITT on brokers: In this experiment, we study the impact of ITT and plan quality on the overlay brokers. Since transformations without message loss require temporary publication buffering, it is important that the generated plans do not result in buffering requirements that are beyond the capacity of individual brokers. An incremental transformation approach requires less buffering since each action involves only a small group of brokers and finishes quickly. Figures

6d and 6g show the growth of the queue size of brokers during topology transformation. Increasing the overlay size and change ratio does not impact any of the approaches. This is due to the inherently low buffering requirement of incremental transformation. However, IPITT is able to execute plans faster without any increase in queue size. The reason is that the plan calculation and execution of IPITT parallelize actions which are independent from each other and do not involve the same broker. Therefore, a higher number of actions per step does not influence the queue size of individual brokers. Figure 6j shows that increasing subscribers results in a sublinear growth of the queue size of brokers by up to 110% when the number of subscribers increases by a factor of 5. The reason is that for the same workload, a higher number of subscribers downstream (at leaf brokers) results in a higher number of publications routed downstream. Therefore, the number of publications in transit is higher at any given time when compared to measurements using the same overlay and workload but with a lower number of subscriptions. In fact, any parameter that increases the number of publications in the overlay, such as publication rate and match ratio, will have the same effect on the brokers during topology transformation. This experiment shows that plans generated by IPITT can perform incremental transformations on the overlay with minimal load increase on brokers.

VII. Conclusions

In this paper, we presented IPITT, an IP-based approach to incremental topology transformation of pub/sub systems. Using an IP-based formulation, IPITT is able to find transformation plans with a minimal number of steps and minimize service disruption. Furthermore, IPITT facilitates integration of new constraints and cost models to customize the generated plans. Compared to existing solutions, IPITT, with our proposed optimization heuristics, reduces planning time by a factor of 10 while producing plans that have up to 45% fewer actions and can be executed up to 55% faster. Furthermore, we showed that IPITT minimizes disruption to clients by producing plans with higher quality.

Acknowledgments

This research was supported by the Alexander von Humboldt Foundation. We thank our shepherd, Yu Hua, who helped us improve this work.

References

[1] E. Fidler, H.-A. Jacobsen, G. Li *et al.*, "The PADRES distributed publish/subscribe system." in *FIW*, 2005, pp. 12–30.

[2] G. Cugola, E. Di Nitto, and A. Fuggetta, "The JEDI event-based infrastructure and its application to the development of the opss wfms," *IEEE transactions on Software Engineering*, 2001.

[3] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and Evaluation of a Wide-area Event Notification Service," *ACM Trans. Comput. Syst.*, 2001.

[4] J. Reumann, "GooPS: Pub/sub at Google," *Lecture & Personal Communications at EuroSys & CANOE Summer School*, 2009.

[5] "Yahoo Pulsar pub/sub messaging platform," https://github.com/yahoo/pulsar, accessed: 2016-12-05.

[6] J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: A distributed messaging system for log processing," in *NetDB*, 2011.

[7] Y. Sharma *et al.*, "Wormhole: reliable pub-sub to support geo-replicated internet services," in *NSDI 15*, 2015.

[8] W. W. Terpstra, S. Behnel, L. Fiege *et al.*, "A Peer-to-Peer Approach to Content-Based Publish/Subscribe," in *DEBS 2003*.

[9] Y. Yoon, N. Robinson, V. Muthusamy *et al.*, "Planning the transformation of overlays," in *SAC 2016*.

[10] C. Chen, R. Vitenberg, and H.-A. Jacobsen, "A generalized algorithm for publish/subscribe overlay design and its fast implementation," in *DISC 2012*.

[11] G. V. Chockler, R. Melamed, Y. Tock, and R. Vitenberg, "Constructing scalable overlays for pub-sub with many topics," in *PODC 2007*.

[12] Y. Yoon, V. Muthusamy, and H. A. Jacobsen, "Foundations for highly available content-based publish/subscribe overlays," in *ICDCS 2011*.

[13] E. D. Nitto, D. J. Dubois, and A. Margara, "Reconfiguration primitives for self-adapting overlays in distributed publish-subscribe systems," in *SASO 2012*.

[14] H. Parzyjegla, G. G. Muhl, and M. A. Jaeger, "Reconfiguring publish/-subscribe overlay topologies," in *ICDCSW'06*.

[15] B. Elshqeirat, S. Soh, S. Rai, and M. Lazarescu, "Topology design with minimal cost subject to network reliability constraint," *IEEE Transactions on Reliability*, 2015.

[16] Y. Zhao and J. Wu, "On the construction of the minimum cost content-based publish/subscribe overlays," in *SECON 2011*.

[17] C. Chen, Y. Tock, H. A. Jacobsen, and R. Vitenberg, "Weighted overlay design for topic-based publish/subscribe systems on geo-distributed data centers," in *ICDCS 2015*.

[18] R. Baldoni, C. Marchetti, A. Virgillito *et al.*, "Content-based publish-subscribe over structured overlay networks," in *ICDCS'05*.

[19] M. A. Jaeger, H. Parzyjegla, G. Mühl *et al.*, "Self-organizing broker topologies for publish/subscribe systems," in *SAC'07*.

[20] A. K. Y. Cheung and H. Jacobsen, "Green resource allocation algorithms for publish/subscribe systems," in *ICDCS 2011*.

[21] H. H. Liu, X. Wu, M. Zhang *et al.*, "zupdate: Updating data center networks with zero loss," *SIGCOMM 2013*.

[22] D. Nau, M. Ghallab, and P. Traverso, *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., 2004.

[23] S. Richter and M. Westphal, "The LAMA planner: Guiding cost-based anytime planning with landmarks," *Journal of Artificial Intelligence Research*, 2010.

[24] N. Lipovetzky and H. Geffner, "Searching for plans with carefully designed probes." in *ICAPS*, 2011.

[25] "Gurobi optimizer," http://www.gurobi.com/products/gurobi-optimizer, accessed: 2016-11-08.

[26] "CPLEX optimizer," http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/, accessed: 2016-11-08.

[27] "Gurobi compute server," http://www.gurobi.com/products/gurobi-compute-server/gurobi-compute-server, accessed: 2016-11-08.

[28] Y. Yoon, "Adaptation techniques for publish/subscribe overlays," Ph.D. dissertation, University of Toronto, 2013.

[29] D. McDermott, M. Ghallab, A. Howe *et al.*, "PDDL-the planning domain definition language," 1998.

[30] J. N. Hooker, "A quantitative approach to logical inference," *Decision Support Systems*, 1988.

[31] M. H. L. Van den Briel, *Integer programming approaches for automated planning*. ProQuest, 2008.

[32] "IBM ILOG CPLEX enterprise server," https://www-01.ibm.com/software/commerce/optimization/cplex-enterprise-server/, accessed: 2016-11-21.

[33] M. van den Briel and S. Kambhampati, "Optiplan: Unifying IP-based and graph-based planning." *J. Artif. Intell. Res.(JAIR)*, 2005.

[34] J. Fan and M. H. Ammar, "Dynamic topology configuration in service overlay networks: A study of reconfiguration policies." in *INFOCOM*, 2006.

[35] V. Muthusamy, M. Petrovic, D. Gao, and H.-A. Jacobsen, "Publisher mobility in distributed publish/subscribe systems," in *ICDCS Workshops*, 2005.

[36] L. Fiege, F. C. Gärtner, O. Kasten, and A. Zeidler, "Supporting mobility in content-based publish/subscribe middleware," in *Middleware*, 2003.

[37] P. Hunt, M. Konar, F. P. Junqueira *et al.*, "ZooKeeper: Wait-free coordination for internet-scale systems." in *USENIX Annual Technical Conference*, 2010.

[38] P. T. Eugster, P. A. Felber, R. Guerraoui *et al.*, "The many faces of publish/subscribe," *ACM Computing Surveys (CSUR)*, 2003.

[39] H. Liu, V. Ramasubramanian, and E. G. Sirer, “Client behavior and feed characteristics of rss, a publish-subscribe system for web micronews,” in *SIGCOMM*, 2005.

[40] J. Gondzio, “Warm start of the primal-dual method applied in the cutting-plane scheme,” *Mathematical Programming*, 1998.

Appendix A
IP formulation of automated planning

The IP formulation of automated planning defines the following sets for each proposition $f \in F$:

- pre_f: set of actions that have f as precondition
- add_f: set of actions that have f as add effect
- del_f: set of actions that have f as delete effect

Furthermore, five more binary variables besides $x_{a,t}$ are defined. These state change variables are used to model transitions between different states.

- $y^{maintain}_{f,t} = 1$ if the value of proposition f (true/false) is propagated to step t.
- $y^{preadd}_{f,t} = 1$ if an action is executed in step t that requires proposition f and does not delete it.
- $y^{predel}_{f,t} = 1$ if an action is executed in step t that requires proposition f and deletes it.
- $y^{add}_{f,t} = 1$ if an action is executed in step t that does not require proposition f but adds it to the state.
- $y^{del}_{f,t} = 1$ if an action is executed in step t that does not require proposition f but removes it from the state.

The complete formulation is as follows:

$$\min \sum_{a \in A} \sum_{t \in \{1,\ldots,T\}} x_{a,t}$$

$$y^{add}_{f,0} = 1 \qquad \forall\ f \in I \tag{1}$$

$$y^{add}_{f,0} + y^{maintain}_{f,0} + y^{preadd}_{f,0} = 0 \qquad \forall\ f \notin I \tag{2}$$

$$y^{add}_{f,T} + y^{maintain}_{f,T} + y^{preadd}_{f,T} \geq 1 \qquad \forall\ f \in G \tag{3}$$

$$\sum_{a \in add_f \backslash pre_f} x_{a,t} \geq y^{add}_{f,t} \tag{4}$$

$$x_{a,t} \leq y^{add}_{f,t} \qquad \forall\ a \in add_f \backslash pre_f \tag{5}$$

$$\sum_{a \in del_f \backslash pre_f} x_{a,t} \geq y^{del}_{f,t} \tag{6}$$

$$x_{a,t} \leq y^{del}_{f,t} \qquad \forall\ a \in del_f \backslash pre_f \tag{7}$$

$$\sum_{a \in pre_f \backslash del_f} x_{a,t} \geq y^{preadd}_{f,t} \tag{8}$$

$$x_{a,t} \leq y^{preadd}_{f,t} \qquad \forall\ a \in pre_f \backslash del_f \tag{9}$$

$$\sum_{a \in pre_f \wedge del_f} x_{a,t} \ = \ y^{predel}_{f,t} \tag{10}$$

$$y^{add}_{f,t} + y^{maintain}_{f,t} + y^{del}_{f,t} + y^{predel}_{f,t} \leq 1 \tag{11}$$

$$y^{preadd}_{f,t} + y^{maintain}_{f,t} + y^{del}_{f,t} + y^{predel}_{f,t} \leq 1 \tag{12}$$

$$y^{preadd}_{f,t} + y^{maintain}_{f,t} + y^{predel}_{f,t} \leq y^{preadd}_{f,t-1} + y^{add}_{f,t-1} + y^{maintain}_{f,t-1} \\ \forall\ f \in F, t \in \{1,\ldots,T\} \tag{13}$$

$$x_{a,t}, y^{preadd}_{f,t}, y^{predel}_{f,t}, y^{add}_{f,t}, y^{del}_{f,t}, y^{maintain}_{f,t} \in \{0,1\} \tag{14}$$

Constraints 1 and 2 add the initial state propositions. Constraint 3 ensures that any proposition comprising the goal state is either propagated to the last step or added in the last step. Constraints 4 to 10 link state change variables to actions ensuring propositions are added to the state or removed from it only as a result of action execution. Constraints 11 and 12 act as mutexes between state change variables, preventing parallel changes that can result in an inconsistent state. This means that during each step of the transformation, a proposition f can be either added, propagated, or removed from the state. Constraint 13 ensures backward chaining requirements, which means that if in step $t-1$ a proposition f is added or maintained, then f can be added or removed in step t, or it can be propagated to step t. Finally, the last constraint ensures that all defined decision variables have a valid value in each step.