Providing Scalable Data Services in Ubiquitous Networks

Raghavendra Prasad[1], Sanket Patil[2], Tanu Malik*[1], Amitabh Chaudhary[3], Venkat Venkatasubramanian[1]

[1] Purdue University, USA
{prasadr, tmalik, venkat}@purdue.edu
[2] International Institute of Information Technology, Bangalore, India
sanket.patil@iiitb.ac.in
[3] University of Notre Dame, USA
achaudha@cse.nd.edu

Abstract. A crucial element of performance in a networked system is its underlying structure or *topology*. Topology governs connectivity between nodes, the amount of data flow, and the efficiency of data flow. In traditional networks, due to physical limitations, topologies remain static through the course of the network operation. Ubiquitous data networks (UDN), on the other hand, are adaptive and can be configured for changes in their topology. This flexibility in controlling their topology makes them very appealing for supporting "anywhere, anyplace" communication. However, this raises the problem of designing a dynamic topology. In this paper, we describe a formal framework based on metrical task systems (MTS) that decides when and how the topology should be reconfigured in response to changes in communication requirements. We perform experiments on a UDN that needs to provide data services to a large number of clients and in which communication requirements vary. Our experiments validate the performance of our methods which always reconfigure the system on a need basis.

1 Introduction

In the vision of pervasive computing, users will exchange information and control their environments from anywhere using various wireline/wireless networks and computing devices [**?**]. Although such a definition of pervasive computing is very appealing to users, it has been reported that the technological path for building such an anytime, anywhere networking environment is less clear [**?**]. One of the most important technical issues is the auto-configuration of the topology between nodes and devices [**?**]. In traditional computing environments such as the Internet, the native routing infrastructure is fixed and the topology is predominantly static. However, in large-scale pervasive computing environments, the topology is mostly deployed and maintained by application service providers who contract with underlying ISPs and buy network bandwidth between nodes to provide value-added network services to end-systems. Thus, the topology can be dynamic and flexible.

We are interested in understanding how and when should a topology be configured in order to support distributed data management services. These services can be in the form of replica or a caching service provided by an ASP in which nodes (both wired

and wireless) acquire and disseminate context-based data. In a pervasive computing environment, configuring a topology to support such distributed data services can be challenging. Firstly, nodes and devices have limited resources. The resource limitation is often due to restricted buffer sizes and storage capacities at nodes, limited bandwidth availability between nodes or limited number of network connections that a node can support. An optimal usage of limited resources requires a topology design that routes the flow of data through most cost-efficient paths. Secondly, the data communication pattern of clients may change drastically over time. This may warrant a topology that quickly adapts to changes in communication requirements.

While the flexibility of reconfiguring a topology is essential, reconfiguring a topology every time a communication pattern changes may not be beneficial. Changing from one network topology to another is not cost-free: it incurs both management overhead as well as potential disruption of end-to-end flows. Additionally, data in transit may get lost, delayed, or erroneously routed. In the presence of these costs, it might be useful to monitor topology changes at the finest granularity. However, the topology should only be reconfigured if the long run benefits of the reconfiguration justify its cost.

In this paper, we are most interested in the dynamic topology design problem, i.e., the problem of determining how and when to reconfigure a topology in a resource-constrained pervasive computing environment in which data communication requirements change dynamically. In order to understand this problem, we first describe the problem of designing a static topology under resource constraints (Section 3.1). In particular, we provide a linear programming based formulation for solving the static topology design problem. The solution to the formulation is used later as a subroutine in the solution to the dynamic topology design problem. We then describe a method for calculating topology reconfiguration costs (Section 3.2). Finally, in Section 4 we focus on the reconfiguration framework that models the dynamic topology problem as a metrical task system (MTS) [**?**]. Task systems are general systems that capture the cost of reconfiguration between topologies in addition to the cost of satisfying a given demand in a given topology. By including the reconfiguration cost the system prevents oscillations between topologies that are sub-optimal in the long run.

The reconfiguration decisions taken by our framework guide the topology selection any time the communication requirements change. The distinguishing part of our framework is that reconfiguration decisions require no statistical modeling or aggregation of the communication requirements from the service provider. This lack of making any assumptions about how the communication requirements may change over time allows the framework to provide a minimum level of guarantee of adapting to changes in the communication requirement. To the best of our knowledge most topology reconfiguration frameworks are based on heuristic policies and provide no theoretical evidence or a systematic way of understanding when to change a topology. We evaluate our algorithms in Section 5 and conclude in Section 7.

2 Related Work

The topology design problem has received significant interest in large information systems such as optical networks, data-centric peer-to-peer networks, and more recently, complex networks. In these systems, the objective is to design an optimal topology under multiple optimality requirements of efficiency, cost, balance of load on the servers

and robustness to failures. The design of an optimal topology is obtained by deriving these measures from past usage patterns and then using network simulations to obtain an optimal topology. Such simulations, extensively described in [**?**, **?**], are often based on neural networks and genetic algorithms. Optimal topologies are obtained after executing the software for several hours. The premise is that once an optimal topology is chosen, it will remain static for the duration of the network operations.

In most adaptive networks such as ubiquitous networks [**?**] and overlay networks [**?**], communication patterns vary so significantly that it is often difficult to obtain a representative usage pattern to perform a simulation. IIn the past, research is proposed to perform simulations repeatedly to obtain optimal topologies such that the system reconfigures itself [**?**, **?**]. However, in these systems communication patterns are aggregated over large time scales and the reconfiguration is slow. Recently, [**?**] adaptive networks have focused on auto-configuration [**?**] in which systems self-monitor the communication requirements and reconfigure the topology as and when communication patterns change drastically. The dynamic topology problem has been recently studied in the context of overlay networks in which topologies can be designed either in favor of native networks, which improves performance, or the ultimate customers which reduces the operational cost of the whole system [**?**, **?**]. While our problem is similar to theirs, our context and cost metrics are different: We study the dynamic topology problem in the context of ubiquitous environment in which nodes and devices have limited resources and communication requirements change arbitrarily.

Given an optimal topology and a stable communication pattern, the problem of determining how to use the edges of the topology such that the cost of using the topology is minimized is itself an intractable problem. Several versions of the problem have been studied in computer networks under the class of multi-commodity flow problems [**?**]. In this paper, for simplicity, we have restricted ourselves to single commodity flows [**?**] which is a suitable model when considering communication requirements over a set of replica nodes. Our primary focus is to understand how to adaptively move between optimal topologies when communication patterns change dynamically.

3 Minimizing the Cost of Data Sharing in a UDN

We consider a ubiquitous computing environment established by an application service provider to provide data sharing services across the network. The provider strategically places replicas on the network to disseminate data. Clients (which can be wired or wireless) make connections to one of the replicas and send queries to it. The queries result in a variable amount of data being transfered from the replica to the client. Query results are routed according to the topology of the network. The application service provider pays for the usage of the network, i.e., the total amount of data that passes through the network per unit of time. The service provider would like to use those edges of the topology through which the cost of transferring the data is minimized subject to data flow constraints over the network. We now state the problem formally.

Let the topology, T, be represented by a graph $G = (V, E)$ in which V denotes the set of all nodes in the network and E denotes the set of all edges. Let there be P replicas and C clients on the network such that $P \cup C \subseteq V$ and $P \cap C = \Phi$. Each edge $e \in E$ in the topology T has a cost c_e which is the cost of sending unit data (1 byte) through each pair of nodes. Let b_e denote the maximum amount of bytes that can be sent on any

edge. Each client, C_i receives an online sequence of queries $\sigma_{C_i} = (q_1, \ldots, q_n)$ and let $r(\sigma_{C_i}) = (r_1, \ldots, r_n)$ be the result size of each incoming query at the client. $r(\cdot)$ denotes the data communication requirement of clients. We denote the data communication requirements of all clients at time t by $\sigma(t) = (\sigma_{C_i}(t), \ldots, \sigma_{C_M}(t))$.

A topology T is chosen from a feasible set of topologies. Given $|V|$ nodes, theoretically, there are a total of $2^{|V|(|V|-1)/2}$ possible topologies. However, not all of these topologies are desirable in practice. A topology is usually required to be connected so that every node remains in contact with the rest of the network.

In addition a topology may be either scale-free [] or symmetric (regular graph). In a scale-free topology, some of the nodes act as special peers or super nodes and have a relatively larger load than other nodes. In a symmetric topology nodes have nearly identical degree distributions and share uniform load.

We assign factor, $\rho \in [0, 1]$, for a topology which measures the skew in degree centrality. We define ρ as:

$$\rho = 1 - \frac{|V|\,(\hat{p} - \bar{p})}{(|V| - 1)(|V| - 2)} \tag{1}$$

Thus ρ is 0 for a scale-free topology such as a star and 1 for a symmetric topology such as a circle (See Figure **??**). We denote the set of all feasible topologies for a given ρ by 0-1 adjacency matrices $\mathcal{T}_\rho = T_1, T_2, \ldots, T_N$.

Finally, a topology reconfiguration policy is the sequence of topologies: $T = (T_1, \ldots, T_n), T_i \in \mathcal{T}$ used by the UDN over time in response to the communication requirement, $\sigma(t)$, changing over time. The total cost is defined as:

$$cost(\sigma, T) = \sum_{t=1}^{n} \sigma(t)(T_t) + \sum_{t=0}^{n-1} d(T_t, T_{t+1}), \tag{2}$$

in which the first term is the sum over time of costs of satisfying data requirement of all clients $\sigma(t)$ under the corresponding topology and the second term is the total cost to transition between topologies in T. Note, if $T_{i+1} = T_i$ there is no real change in the topology schedule and incurred reconfiguration cost is zero.

The total cost equation is minimized by an algorithm which generates the best topology schedule T. This requires an algorithm to identify when demand characteristics have changed significantly such that the current physical design is no longer optimal and choosing a new topology such that excessive costs are not incurred in moving from the current topology, relative to the benefit. An offline optimal algorithm OPT that knows the entire σ obtains a configuration schedule S with the minimum cost. An adaptive algorithm ALG determines $T = (T_0, ..., T_n)$ without seeing the complete workload $\sigma = (q_1, ..., q_n)$. We first describe the cost estimation functions which can be used by any algorithm and then describe algorithms which decide when and how to reconfigure.

3.1 The Static Topology Problem

Let the cost of flowing a unit amount of data through an edge e of a topology T be c_e. Let f_e be the amount of bytes that flow through this edge in order to satisfy the communication requirement at a client. The overall cost to support communication requirement

of all clients is the cost of flowing data through all the edges which is defined as:

$$\sum_{e \in E} |f_e| \, .h_e \tag{3}$$

Clearly the above should be minimized subject to the following constraints:

1. The flow in an edge should not exceed its capacity and there is no excess reverse flow in an edge.

$$\forall e = (u, v) \in E : f_{u,v} = -f_{v,u} \textit{ and } |f_e| \leq b_e \tag{4}$$

2. The replica nodes do not request data.

$$\forall p \in P, \forall \, u \ \in \textit{neighbors of } p : f_{u,p} \leq 0 \tag{5}$$

3. All demand for data at clients is satisfied.

$$\forall c \in C : \sum_{u \in \textit{ neighbors of } c} f_{u,c} = r(\sigma_{C_i}(t)) \tag{6}$$

4. The number of bytes passing through each node $b_u, u \in V$ remains balanced and equals the skew in the degree distribution.

$$\rho = \textit{max}(b_u) - \frac{\sum_u b_u}{|V|}, \tag{7}$$

where $\forall u \in V : b_u = \frac{\sum_{v \in neighbors of u} |f_{vu}| + d_u}{2}$

If the data were routed through using minimum-operation-cost paths, the static topology design problem is the problem of finding a topology T , under the constraints of connectivity and degree-bound, that can minimize the cost in Equation 4 for a communication requirement $\sigma(t)$ that remains constant over time, i.e., $\sigma(t) = \mathcal{C}$. We term such a topology, optimal-static topology for $\sigma(t)$, and denote it by $T^*(\sigma(t))$. Similar to most other topological design problems, the static topology design problem can be modeled as a linear programming problem and can be solved efficiently in the worst case.

3.2 The Reconfiguration Cost

Every time the system reconfigures its topology to adapt to changes in communication requirements, a reconfiguration cost is incurred. This cost is the overhead or the impairment to performance incurred by the transition from one topology to another. Various costs could be incurred during a topology reconfiguration, depending on the implementation details of the UDN. For example, establishing and changing links incurs control and management overhead which can be translated to energy costs in a wireless network or costs paid to ISPs in a wired network or a combination of both in wired/wireless setting. Any fraction of data in transit during topology reconfiguration is subject to routing disturbance leading to a rerouting overhead. Depending on the UDN implementation,

when topologies change, data in transit may wander through a path with a high operation cost. Finally, rerouting overhead can be magnified at the end-systems.

In this paper, we assume reconfiguration costs as the cost of auto-configuring the entire network. Configuring a network involves establishing basic IP-level parameters such as IP addresses and addresses of key servers. Auto-configuration involves automatic distribution of these IP configuration parameters in the entire network. In the wired networking environment, protocols such as Point-to-Point Protocol (PPP) [] , Dynamic Host Configuration Protocol (DHCP) [], and Mobile IP [] can configure individual hosts. Similarly there are protocols for such as PPP for serial links and DHCP for broadcast LANS. In the pervasive environments, DCDP is a popular protocol for auto-configuration. In DCDP auto-configuration is done by recursively splitting the address pool down a spanning tree formed out of the graph topology. Thus the total configuration cost of the network is essentially proportional to the height of the spanning tree []. Since we are considering the cost of reconfiguring a topology, we would need to auto-configure the network twice once for reclaiming the addresses from the old topology and then assigning the addresses on the new topology.

A general approximate measure for the reconfiguration cost is the total number of links that need to be changed during a transition

$$d(T_{old}, T_{new}) = \sum W.(g(T_{old}) + g(T_{new})) \tag{8}$$

in which $g(\cdot)$ is the auto-configuration cost and is proportional to the height of the spanning tree in each topology and W is weight parameter that converts this cost in terms of operation costs.

4 The Topology Reconfiguration Problem with Dynamic Data Requirements

In a real-world, client nodes receive a sequence, σ, of queries, in which the size of the query result differs. Thus the amount of data delivered from the replica to the client changes over time. In such a dynamic scenario, as shown in Section **??**, no one topology remains optimal and a reconfiguration of topology may be needed. In this section, we first describe a suite of policies which specify *when* the topology should be reconfigured. These policies are predictive in that they assume amount of data requested to remain stable for some period of time. However, in several environments, request for data is bursty in that arbitrarily large amounts of data are requested over short periods of time. For such environments we describe a more conservative approach based on metrical task systems []. Algorithms in task systems are based on competitive analysis and provide guarantees on the total cost of satisfying data demands and making transitions. We adapt algorithms proposed for task systems to our need.

4.1 Topology Reconfiguration Policies

Given the communication patterns, X(t), a reconfiguration policy, Y(t), is essentially a set of rules specifying when and how the overlay topology should be reconfigured. The following are three examples.

1. Policy 1, Periodic: Under this scheme the decision to perform a transtition is done once every C demand frequency by calculating the static optimal at that instant for that requirement. For instance, if C = 10 and if the demands arrive once per day, then the decision to make the transition is done once in 10 days and the transition is done to a statically optimal topology.
2. Policy 2, Moving Average: Find an average demand over the last 10 demands and keep switching to that configuration.
3. Policy 3, Greedy: This is a naive and greedy scheme wherein the topology which can satisfy the demand with the least cost is chosen for every demand.

The above policies are heuristic in that they make decisions based on the rate of change of data requirements. If the amount of data requested changes slowly, then the

Metrical Task System Online ski rental is a classical rent-or-buy problem. A skier, who doesn't own skis, needs to decide before every skiing trip that she makes whether she should rent skis for the trip or buy them. If she decides to buy skis, she will not have to rent for this or any future trips. Unfortunately, she doesn't know how many ski trips she will make in future, if any. This lack of knowledge about the future is a defining characteristic of on-line problems [**?**]. A well known on-line algorithm for this problem is rent skis as long as the total paid in rental costs does not match or exceed the purchase cost, then buy for the next trip. Irrespective of the number of future trips, the cost incurred by this online algorithm is at most twice of the cost incurred by the optimal offline algorithm.

If there were only two topologies and the cost function $d(\cdot)$ satisfies symmetry, the reconfiguration problem will be nearly identical to online ski rental. Staying in the current topology corresponds to renting skis and transitioning to another topology corresponds to buying skis. Since the algorithm can start a ski-rental in any of the states, it can be argued that this leads to an algorithm that cost no more than four times the optimal.

When there are more than two topologies the key issue is to decide which topology to compare with the current one. This will establish a correspondence with the online ski rental problem. A well-known algorithm is by Borodin et. al. []. Their algorithm assumes the state space of all topologies to form a metric which allows them to define a *traversal* over N topologies. We show that our reconfiguration function is indeed a metric function and then describe the algorithm.

To form a metric space, the reconfiguration function should satisfy the following properties:

1. $r(T_i, T_j) \geq 0, \forall i \neq j, T_i, T_j \in \mathcal{T}$ (positivity);
2. $d(T_i, T_i) = 0, \forall i \in \mathcal{T}$ (reflexivity); and
3. $d(T_i, T_j) + d(T_j, T_k) \geq d(T_i, T_k), \forall T_i, T_j, T_k \in \mathcal{T}$ (triangle inequality)
4. $d(T_i, T_j) = d(T_j, T_i) \forall T_i, T_j \in \mathcal{T}$

In our case the reconfiguration function $d(\cdot)$ depends upon the sum of auto-configuration costs in each topology. Since the cost is positive

When the costs are symmetrical, Borodin et. al [**?**] use *components* instead of configurations to perform an online ski rental. In particular their algorithm recursively traverses one component until the query execution cost incurred in that component is approximately that of moving to the other component, moving to the other component and traversing it (recursively), returning to the first component (and completing the cycle) and so on. To determine components, they consider a complete, undirected graph $G(V, E)$ on $\mathcal{S}$ in which V represents the set of all configurations, E represents the transitions, and the edge weights are the transition costs. By fixing a minimum spanning tree (MST) on G, components are recursively determined by pick the maximum weight edge, say (u, v), in the MST, removing it from the MST. This partitions all the configurations into two smaller components and the MST into two smaller trees. The traversal is defined in Algorithm 1.

This algorithm is shown to be $8(N-1)$-competitive [**?**]. Recall, ALG is said to be α-competitive if there exists a constant b such that for every finite query sequence σ,

$$cost(\mathsf{ALG}\ on\ \sigma) \leq \alpha * cost(\mathsf{OPT}\ on\ \sigma) + b. \tag{9}$$

OPT is the offline optimal that has complete knowledge of σ.

Input: Tree: $F(V, E)$
Output: Traversal for F: $\mathcal{T}$
if $E = \{\}$ **then**
  $\mathcal{T} \leftarrow \{\}$;
else if $E = \{(u, v)\}$ **then**
  Return $\mathcal{T}$: Start at u, traverse to v, traverse back to u;
else
  Let (u, v) be a maximum weight edge in E, with weight 2^M;
  On removing (u, v) let the resulting trees be $F_1(V_1, E_1)$ and $F_2(V_2, E_2)$, where $u \in V_1$, and $v \in V_2$;
  Let maximum weight edges in E_1 and E_2 have weights 2^{M_1} and 2^{M_2} respectively;
  $\mathcal{T}_1 \leftarrow \mathsf{traversal}(F_1)$;
  $\mathcal{T}_2 \leftarrow \mathsf{traversal}(F_2)$;
  Return $\mathcal{T}$: Start at u, follow $\mathcal{T}_1$ 2^{M-M_1} times, traverse (u, v), follow $\mathcal{T}_2$ 2^{M-M_2} times, traverse (v, u);
end

Algorithm 1: $\mathsf{traversal}(F)$

5 Experiments

We first describe our experimental setup and then the set of experiments performed. Our current objective is to get a validation of our policies through a simulated environment. Thus while our setup is a representation of a real-world pervasive environment, doing experiments with real data is part of future work. Our setup simulates a replica environment with 10 replicas and clients with varying communication requirements and demands. We consider various class of topologies between the replicas, and also model a simulated demand sequence lasting for a duration of 100 days for all the clients. In

order to capture the reconfiguration costs as described in 3.2 we introduce a parameterizable constant weight called K which when factored with the height of the old and new topologies provide the overall cost of transition.

The clients receive a demand sequence which represent the workloads resulting due to the client's action. We use workloads that simulate real-world data access in pervasive environments.

Communication requirements in a pervasive environment are bursty. To model these demands we consider we X(t) as a continuous Markov process. Dynamics are created by adding seasonal components (5) and a random walk process (1) to the mean.

We measure operational costs in terms of a dollar cost, where the cost of reconfiguration is captured by the K weight and the heights of the corresponding topologies involved. Though the actual dollar figure may not represent real world metrics, its relative scale and hence can be considered as a valid performance evaluation metric.
To perform all these simulations, we developed a fully customized Python based system that acts as both an event driven simulator and also a simulator for performing experiments on a UDN. We used GAMS; a popular high level modeling system for mathematical programming and optimizations.

5.1 Cost of Reconfiguration and Performance Analysis

We compute the cost of satisfying a query sequence under a topology schedule generated by various policies. The transitions between topologies is depicted with respect to the demands arranged in chronological order. Figures 1, 2 and 3 shows the topological transitions for various configurations. For instance Figure 3 shows the 20-node configuration and its topological transitions under various policies. Its easy to infer from the figure that Policy 3 which chooses the least expensive topology every time in a greedy fashion ends up jumping between topologies very frequently. We show later that this behavior is extremely detrimental.

We performed the simulation on these 3 configurations for a demand sequence lasting 100 time units. For each of these configurations we calculate the net cost incurred under the 4 policies. For each of these experiments we varied the K factor to show how the unit reconfiguration cost can have a significant impact on the performance of all the policies, Esp for policy 4.

Figure 4 which represents the costs incurred by each configuration under all the 4 policies with a K factor of 1000. We see that the policy 1 and 2 which represent the online algorithm and the static algorithm perform better than the proposed solution. This attributed to the fact that for networks which do not involve a significant reconfiguration cost it might alright to choose policy 1 or 2 as now the focus is now not on minimizing the number of reconfigurations. Evenin such cases policy 4 performs almost as good as policy 1 and 2.

Figure 6 which represents the costs incured by each configuration under all the 4 policies with a K factor of 4000. Policy 4 clearly out performs policy 1 and 2 as now the onus is on the reconfiguration costs as it has a significant weight. It is obvious from all the results that a naive greedy approach is unacceptably bad.

Figure 5 which represents the costs incured by each configuration under all the 4 policies with a K factor of 2000. We see that Policy 4 is leveled out with policy 1 and 2.

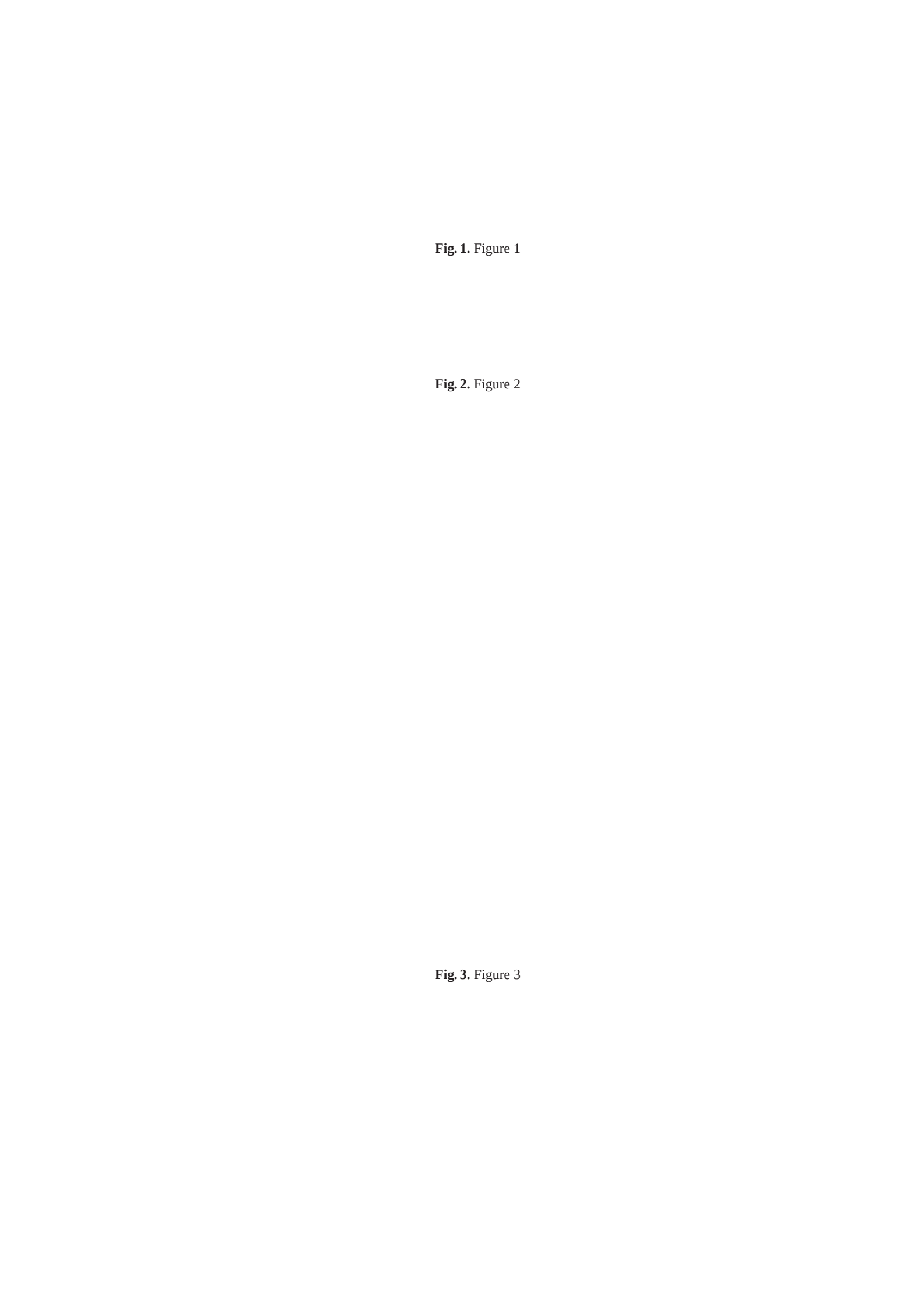

Fig. 1. Figure 1

Fig. 2. Figure 2

Fig. 3. Figure 3

Fig. 4. Figure 4

Fig. 5. Figure 5

Fig. 6. Figure 6

Fig. 7. Table 1

This is due the fact that though the K factor is not as high as 4000 it still is an important contributor to the overall costs. Table 1 gives the exact costs for ease of comparison.

6 Future work

For the scope of this project, we have restricted ourselves to configurations that have a small number of nodes relative to the real world. In the future, we plan to scale the number of nodes. Another interesting dimension to this problem is that of state space exploration. Clearly, with huge node configurations, the state space becomes extremely large and efficient heuristics are needed to walk through this space. This apart, the applications to this kind of dynamic topological reconfigurations can be extended to the domain of cloud computing. With concepts like premium service and fault tolerant clouds, it becomes important to determine a schedule to handle traffic that cannot be modeled statistically.

7 Conclusion

In what started as an Industrial Engineering optimization problem, we spotted its novel use in solving issues related to reconfiguring topologies in UDN. Though metrical task systems have been in use for a long time in the areas of database systems and caching, to the best of our knowledge, it is being used for the first time to tackle issues related to

UDN. Though the underlying concept is inspired from metrical task systems, its design, structure and application are novel. In this paper, we have achieved to show as to how problems belonging to this category can be efficiently solved using an on-line approach. We also show that for any application that cannot make any valid assumptions about the workload or its future events, our algorithm provides an efficient way to schedule reconfigurations to minimize the cost incurred.

References