Providing Scalable Data Services in Ubiquitous Networks

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

[1] Purdue University, USA
tmalik, prasad, venkat@purdue.edu
[2] IIIT, Hyderabad, India
[3] University of Notre Dame, USA
achaudha@cse.nd.edu

Abstract. The topology is a fundamental part of a network that governs connectivity between nodes, the amount of data flow and the efficiency of data flow between nodes. In traditional networks, due to physical limitations, topology remains static for the course of the network operation. Ubiquitous data networks, alternatively, are more adaptive and can be configured for changes in their topology. This flexibility in controlling their topology makes them very appealing and an attractive medium for supporting "'anywhere, anyplace"' communication. However, it raises the problem of designing a dynamic topology i.e., reconfiguring the topology when communication requirements change over time. We describing methods that determine an optimal topology configuration of a ubiquitous data network (UDN) in which communication requirements remain static over time. We then describe a a formal framework based on metrical task systems that decides when and how topology should be reconfigured as communication requirements change over time. We perform experiments on a UDN that needs to provide data services to large number of clients an in which communication requirements vary. Our experiments validate the performance of our methods always reconfiguring 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 issue is the auto-configuration of the topology between the devices. Traditional computing environemnts such as the Internet the native routing infrastructure is fixed and the topology is predomonantly static. However in pervasive computing the topology needs to be flexible.

We are particularly interested in studying how the underlying topology influences data management services that run on top of it. Data mgmt in Pervasive computing environments present entirely new set of challenges because of the fact that data may be acquired and disseminated at various points within the system. In addition demands for

data may change significantly over time. The routing of the data will therefore govern the overall efficiency and cost of delivering this data. To support various kinds of dm services, pervasive system would not only need to find the optimal topology but also how to rapidly configure themselves that minimize the operation cost for disseminating data given a set of communication requirements.

If the communication requirement is constant over time, the optimal choice of overlay topology is static. If communication requirements change over time, for example, from the ones shown in Fig **??** to the ones shown in Fig **??**, it may be better for the UDN to be reconfigured. If the network is small or has a moderate number of nodes, it is feasible for a provider to monitor and statistically model the communication requirements in the system. However, a UDN often consists of a very large number of nodes making the topology reconfigurability problem even more intractable. The fundamental question then is when and how the overlay topology should be reconfigured under dynamic communication requirements.

Reconfiguring a topology cannot be done at every time step even if new there exists a topology that improves efficiency for the new communication requirements. Even though a UDN can be auto-configured in a small time scale, changing network topology is not cost-free; it incurs both management overhead as well as potential disruption of end-to-end flows; 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 a finest granularity but the topology should only be changed if the long run benefits of making a change justifies the cost of the change.

In this paper, we are most interested in the dynamic topology design problem, i.e., the problem of determining when to reconfigure a topology as communication requirements change dynamically. In order to understand this problem, we first describe the static topology design problem. In particular, we provide an LP-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 problem. In Section **??** we concentrate on the reconfiguration framework that models the dynamic topology problem as a metrical task system. Task systems are general systems that capture the cost of reconfiguration between two topologies in addition to the cost of satisfying a given demand in a given topology. By including the reconfiguration cost the system prevents oscillations into states 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 **??** and conclude in Section **??**.

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 arbitrary 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 measures of efficiency, robustness, load and cost from past usage patterns and then using network simulation to obtain an optimal topology. Such simulations are extensively described in [] in which optimal topologies are obtained when there are conflicting performance requirements that need to be balanced. However, in most of these systems a topology once chosen remains static for the duration of the system operation.

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.

In most adaptive networks [] such as UDNs and overlay networks [], communication pattens vary so significantly that it is often difficult to obtain an representative usage pattern to perform a simulation. In the past [], research proposed to perform simulation repeatedly to obtain an optimal topology and 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 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 the favor of native networks, which improves performance or the ultimate customers which reduces the operation cost of the whole system []. In this paper, we have studied the similar problem in the context of UDNs.

[] consider the the problem of determining dynamic topology reconfiguration for service overlay networks with dynamic communication requirement, and the goal is to find the optimal reconfiguration policies that can minimize the potential overall cost of using an overlay. While their work is very similar in spirit to ours, their methods are limited to very small networks in which a statistical model for the dynamic communication requirements can be built. Consequently they propose heuristic methods for constructing different flavors of reconfiguration policies. On the contrary, in this paper we provide a systematic study with theoretical evidence for the advantage of reconfigurability.

3 Minimizing the Cost of Data Sharing in a UDN

We consider a UDN established by an application service provider to provide critical services across the network. In such a UDN the provider places replicas at strategic locations to receive query requests and disseminate data. Clients (which can be wired or wireless) make connections to data sources and send queries which result in a variable

amount of data being transfered from the replica to the client. The topology of the network determines the cost the application service provider will have to incur in routing the result of each client query on the network. Ideally, the service provider would like to use those edges of the topology through which the cost of transferring the data is minimized.

Let $P = (P_1, \ldots, P_N)$ be the set of data nodes in the network that correspond to the replica nodes, $C = (C_1, \ldots, C_M)$ be the set of client nodes and $V \supset P \cup C$ be the set of all nodes in the network. Let c be the link matrix to denote the cost of sending unit data (1 byte) through each pair of nodes. Let b be the link matrix to denote the maximum amount of bytes that can be sent on any edge. Let each client receive an online sequence of queries q be the queries and let r be their corresponding expected result size. r demotes the comunication req of clients. We denote the communication requirements at all clients at time t by $\bar{Q(t)} = (Q_{C_1}(t), \ldots, Q_{C_M}(t))$.

We choose a feasible set of topologies for the UDN. Theoretically, there are a total of $2^{n(n-1)/2}$ possible topologies over n nodes. However, not all of these topologies are desirable in practice. A topology is usually required to be connected so that every node remains contact with the rest of the network. In addition it should have some capabilities for distributing the amount of data on the network. This ability of the topology to distribute load among nodes is measured as the skew in degree distribution. We define this as the difference in the maximum degree in the graph $\hat{p}$ and the mean degree of the nodes $\bar{p}$. For a connected graph of n nodes, the worst skew occurs for the star topology. The central node has a degree of $n-1$ and all the nodes surrounding it have a degree of 1. Therefore, the worst skew is $\frac{(n-1)(n-2)}{n}$ The best skew is 0, when all the nodes have the same degree. This occurs when the topologies are regular graph topologies as in a circular topology or a clique. Thus, ability to distribute load in a topology is a mapping from a value in the interval $\left[0, \frac{(n-1)(n-2)}{n}\right]$ to a value in the interval $[0, 1]$. We denote the set of feasible topologies by 0-1 adjacency matrices $\mathcal{T} = T_1, T_2, \ldots, T_l$.

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, $\bar{Q(t)}$, changing over time. The total cost is defined as

$$cost(\sigma, S) = \sum_{i=1}^{n} q_i(S_i) + \sum_{i=0}^{n-1} d(S_i, S_{i+1}), \tag{1}$$

in which the first term is the sum of costs of each query in σ under the corresponding configuration and the second term is the total cost to transition between configurations in S. Note, if $S_{i+1} = S_i$ there is no real change in the configuration schedule and incurred transition cost is zero. An offline optimal algorithm OPT knows the entire σ and obtains a configuration schedule S with the minimum cost. An online algorithm ALG for $\mathrm{AdaptPD}$ determines $S = (S_0, ..., S_n)$ without seeing the complete workload $\sigma = (q_1, ..., q_n)$. Thus ALG determines each physical configuration S_i, based on the so far known workload $(q_1, ..., q_i)$ and makes no assumptions about the future workload in advance.

3.1 The Static Topology Problem

given any edge in a topology t, we know that the cost of flowing a unit amount of data through that edge is he. Let fe be the amt of bytes that flow through this edge in order to satisfy the communication requirement at a client. Then the overall cost to support commun req of all clients is the cost of flowing data thr all the edges. formally

$$\sum_{(u,v)\in E} |f_{u,v}| \,.h(u,v) \tag{2}$$

clearly the above should be minimized subject to the constraints: The flow in an edge should not exceed its capacity and there is no excess reverse flow in an edge.

$$\forall (u,v) \in E : f_{u,v} = -f_{v,u} \textit{ and } |f_{u,v}| \leq b(u,v) \tag{3}$$

We assume that the replica nodes do not have any communication requriements.

$$\forall p \in P, \forall\, u \ \in \textit{neighbors of } p : f_{u,p} \leq 0 \tag{4}$$

Finally, the flow constraint for satisfying all the demand.

$$\forall c \in C : \sum_{u\in \textit{ neighbors of } c} f_{u,c} = Q_c \tag{5}$$

The above is minimized for a given coom req at atimt instance t and a given topology if the data are 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 **??** for a communication req Q at time t. We term such a topology optimal-static topology for Q(t) and denote it by T (Q(t)). Similar to most other topological design problems, the static topology design problem can be modeled as an linear programming problem and is an NP-hard problem.

If the communication requirement Q(t) is constant over time, then the optimal overlay topology reconfiguration policy is one that always uses the optimal-static topology for Q, i.e., Y (t) = T (C) for all t.

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

$$d(T_{old}, T_{new}) = \sum g(T_{old}) + g(T_{new}) \tag{6}$$

in which $g(\cdot)$ is the auto-configuration cost and is proportional to the height of the spanning tree in each topology.

4 The Topology Reconfiguration Problem with dynamic demand requirements

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.

Policy1: Find an average demand over the last 10 demands and keep switching to that configuration. Policy2: Find the static optimal after every 10/20 demands and jump to that configuration. Policy 3: Run for each demand and keep jumping to that configuration.

Policy 4:

Related Problems: 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 configurations and the cost function $d(\cdot)$ satisfies symmetry, the $\mathrm{OnlinePD}$ problem will be nearly identical to online ski rental. Staying in the current configuration corresponds to renting skis and transitioning to another configuration 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.

In larger number of configurations the key issue in establishing a correspondence with the online ski rental problem is in deciding which configuration to compare with the current one. 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,

$$cost(\texttt{ALG}\ on\ \sigma) \leq \alpha * cost(\texttt{OPT}\ on\ \sigma) + b. \tag{7}$$

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

d is any function that satisfies the following properties:

1. $d(S_i, S_j) \geq 0, \forall i \neq j, S_i, S_j \in \mathcal{S}$ (positivity);
2. $d(S_i, S_i) = 0, \forall i \in \mathcal{S}$ (reflexivity); and
3. $d(S_i, S_j) + d(S_j, S_k) \geq d(S_i, S_k), \forall S_i, S_j, S_k \in \mathcal{S}$ (triangle inequality)

In particular, d does not satisfy the symmetry property, *i.e.,* $\exists S_i, S_j \in \mathcal{S}\ d(S_i, S_j) \neq d(S_j, S_i)$. This asymmetry in transition costs exists because the sequence of operations (i.e. insertion or deletion) required for making physical design changes in a database exhibit different costs.

5 Experiments

We perform experimental set up

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. The size of the result

Policy5 has polynomial-time complexity and finds the minimal spanning tree using the Prims algorithm. It is a general algorithm in that it makes no assumptions about the workload. However, this generality comes at a cost. Namely, given some knowledge about the characteristics of a specific workload, we can design highly tuned workload

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: $\mathsf{traversal}(F)$

adaptive algorithms. To measure the cost of generality, we compare it with other policies described in Section **??**.

We measure operational costs in terms of a dollar cost for We associate a dollar cost ρ with the We measure the cost of the algorithms in terms of average response time of queries executed in SDSS. This is the measure from the time a query is submitted until the results are returned. If a transition to a new configuration is necessary, the algorithm undergoes a transition before executing the query. This increases the response time of the current query but amortizes the benefit over future queries. Our results reflect average response time over the entire workload.

5.1 Results

5.2 Cost of Reconfiguration

We compute the cost of satisfying a query sequence under a topology schedule generated by various policies. figure **??** shows the division of operational costs and the cost of transitioning between topologies. Policy5 improves on the cost of Policy1 by a factor of Y. This is a very encouraging result for pervasive environments where devices are resource-constrained and policies that improve operation costs are needed. However, Policy 3 further improves cost by Y. This is because Policy3 relies on the predictive modeling of the demand. However, the improvement is low considering that Policy5 is general and makes no assumptions regarding workload access patterns. Policy1 suffers due to being over-reactive making changes even when they are not required and incurs a very high transition cost.

Another interesting feature of the results is that Policy5 incurs much lower transition costs than Policy4. This artifact is due to the conservative nature of Policy5. It evaluates only two alternatives at a time and transitions only if it expects significant performance advantages. On the other hand, Policy4 responds quicker to workload changes by evaluating all candidate topologies simultaneously and choosing a topology that benefits

the most recent sequence of queries. This optimism of Policy4 is tolerable in this workload but can account for significant transition costs in workloads that change even more rapidly.

5.3 Quality of a Schedule

In this experiment we compare the quality of schedule generated over the length of the sequence. The better the policy, its schedule will mimic that of the static optimal.

5.4 Effect of Degree Bound

As discussed in Section II, the degree bound of an overlay characterize the types of link maintenance costs that are not directly related to the bandwidth consumption for delivering user data. It reflects an overlay network providers aversion to these types of cost. In this section, we are interested in understanding how dynamic topology reconfiguration policies are affected by the degree bound, and the other way, what dynamic topology reconfiguration implies to the choice of degree bound. Fig. 14 shows how the degree of an overlay network affects the cost of the reconfiguration policies. Most parameters for the experiment are the same as those given in Table I. The transitions between communication patterns are Markovian. From each communication pattern, the system can make up to 4 transitions and the transition rates are random values in range [1,5].