Dynamic Computations in Ever-Changing Networks Invited Talk Abstract[*]

Idit Keidar
Dept. of Electrical Engineering, Technion
Haifa, 32000, Israel
idish@ee.technion.ac.il

ABSTRACT

This talk focuses on *dynamic computations* (sometimes called live, on-going, continuous, or stabilizing), which *continuously* adapt their output to reflect input and network topology changes. Three specific examples are discussed: *continuous weighted matching*, *live monitoring*, and *peer sampling* (also called gossip-based membership). Such computations are of interest in *ever-changing* networks, where the network topology itself (nodes and links) constantly changes, as do the inputs to the computation, e.g., sensor reads. Ever-changing networks occur in many settings nowadays, including ad-hoc, vehicular, and sensor networks, social networks, and clouds spanning multiple data-centers.

Categories and Subject Descriptors

C.2.4 [**Computer Systems Organization**]: Computer Communication Networks—*Distributed Systems*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed Systems*; H.4.3 [**Information Systems Applications**]: Communications Applications

General Terms

Algorithms, Reliability, Theory

Keywords

Dynamic systems, network algorithms, weighted matching, group membership, peer sampling, average aggregation

1. INTRODUCTION

In recent years, we increasingly see *ever-changing networks* deployed in the real world. They occur not only in mobile settings, such as ad hoc and vehicular networks, but also in stationary settings. For example, sensor networks are subject to churn (dynamic changes in the set of nodes) as sensor motes frequently fail and are replaced by new ones; their network topology dynamically changes as communication links disappear and reappear due to weather and battery conditions. Social networks are also ever-changing, as their topology is defined by users' dynamic interests and "friends". Even networks that were traditionally static, like those found in large data centers and clouds, today exhibit high churn rates because data center operators constantly bring online new servers and virtual machines while old ones die out. And in the near future, one can envision dynamic switching in data centers as a part of network virtualization. All these examples illustrate the importance of treating ever-changing networks as a first class citizen.

Traditional distributed network computing is based on a static foundation. It treats dynamism as a "problem" or failure that needs to be overcome: Fault-tolerant algorithms are expected to achieve a correct result despite such failures; while self-stabilizing algorithms are expected to converge to a correct state after the "problem" is gone. In both cases, the network is expected to stop changing at some point, as do the algorithm's inputs and output. In the past decade or so, this "eventually static" paradigm has begun to crack with the advent of churn-resistant peer-to-peer networks. Indeed, quite a bit of recent work deals with building overlay networks and on supporting peer-to-peer lookup under constant churn, though typically over a static (fully connected) underlying network topology. Still, such systems do not really compute anything distributively. More elaborate network problems have yet to be revisited in an ever-changing network model.

In this talk, we focus on *dynamic computations* (sometimes called live, on-going, continuous, or stabilizing), which *continuously* adapt their output to reflect topology changes in an ever-changing network, as well as changes in their input. Such problems are more challenging than their static, one-shot counterparts. Differently from a multi-shot computation, which runs multiple instances of a static one, a dynamic computation does not begin at pre-defined points in time, but rather responds to changes in input and topology. We discuss three examples of dynamic computations: *continuous weighted matching*, *live monitoring* [1], and *peer sampling* (also called gossip-based membership) [2].

2. DYNAMIC MATCHING

A popular static (one-shot) network problem is *(weighted) matching*. A *matching* is a set of graph edges that are vertex-

[*]Supported by the Israeli Science Foundation (ISF), Technion Funds for Security Research, and the Israeli Ministry of Trade and Labor Rescue consortium.

disjoint, i.e., no node participates in more than one edge. The goal of a matching algorithm is to *maximize* the number of edges, or rather, to approximate the maximum within a constant factor (e.g., 2) efficiently. The *weighted* variant of the problem is appropriate where different links in the network have different characteristics (e.g., bandwidth, throughput, loss rates, and load), which can be captured using *weights*. Here, the goal is to approximate the maximum matching weight, which is the sum of the weights of all selected edges.

Matching is useful in wireless networks, where traditional overlay networks are insufficient, because simultaneous communication by adjacent nodes leads to interference. In such networks, the communication pattern can be designed using a matching algorithm to ensure that no interference occurs. Though the weighted matching problem garnered a lot of attention of late, virtually all recent works consider the one-shot problem in a synchronous network with a static topology. This talk, instead, introduces the *continuous weighted matching problem*, which constantly outputs a valid matching while adapting to the network's ever-changing topology. The talk further presents an asynchronous algorithm for its solution.

3. LIVE MONITORING

There is a plethora of work on gathering data in a large network or computing some function thereof. For example, a frequently studied problem is *average aggregation* (or *average consensus*), which computes the average sensor read in a sensor network. By and large, this work focuses on solving the problem *once*. Some solutions assume that data (e.g., sensed values) is static (i.e., never changes), while others gather the latest sensor information (once) in response to an explicit query. However, in a real network, the values constantly change, and a monitoring service ought to track the latest data values in the system. While this can be addressed by periodic polling, this approach may lead to inaccurate results because it can incur a large delay in detecting abrupt changes, and may also be wasteful, due to inevitably sometimes polling when there are no changes.

This talk discusses LiMoSense [1], a fault-tolerant live monitoring algorithm for dynamic sensor networks. LiMoSense is an average aggregation algorithm that performs *live monitoring*, i.e., it constantly obtains a timely and accurate picture of dynamically changing data. LiMoSense uses gossip to dynamically track and aggregate a large collection of ever-changing sensor reads. It overcomes message loss, node failures and recoveries, and dynamic network topology changes.

4. PEER SAMPLING (GOSSIP-BASED MEMBERSHIP)

In a large-scale ever-changing network, it is impossible for nodes to constantly keep track of all the nodes in the system. However, in order to allow nodes to communicate with each other, each node must know the ids, (for example, IP addresses and ports), of some other nodes. The set of ids a node knows is called its *view*. For fast and robust communication, it is important for views to include some *random* nodes. Obtaining such random ids is called *peer sampling*. Beyond maintaining an overlay graph for communication, independent random node id samples are useful for a variety of additional applications, such as gathering statistics, gossip-based aggregation, and choosing locations for data caching.

The most common approach to peer sampling is using gossip-based membership protocols. In such protocols, nodes exchange ("gossip about") ids from their views with their neighbors, and use this information to update their views.

In an ever-changing network, peer sampling is a dynamic problem: Each node constantly outputs a sample or view, and peer samples must evolve to reflect joining nodes and exclude ones that left or failed. Moreover, the system should converge to provide independent uniform samples from any sufficiently connected initial topology resulting from joins, leaves, and failures.

In this talk we overview *Send & Forget (S&F)* [2], a peer-sampling protocol for dynamic networks subject to churn and message loss. S&F has a number of desirable mathematically proven guarantees, including convergence to independent uniform samples from any sufficiently connected initial topology resulting from joins, leaves, and failures. The protocol works well in practice because it does not require atomic actions and it overcomes message loss.

5. CONCLUSIONS

Ever-changing networks already exist today, and will become ever more prevalent in the future. Over such networks, solving dynamic versions of network problems is important. We discussed three examples of such problems, but many more have yet to be studied.

6. ACKNOWLEDGMENTS

I am grateful to my collaborators on the projects described in this talk— Liat Atsmon Guz, Ittay Eyal, Maxim Gurevich, Shay Kutten, Raphael Rom, and Gil Zussman.

7. REFERENCES

[1] I. Eyal, I. Keidar, and R. Rom. Limosense - live monitoring in dynamic sensor networks. In *7th Int'l Symp. on Algorithms for Sensor Systems, Wireless Ad Hoc Networks and Autonomous Mobile Entities (ALGOSENSORS'11)*, Sept. 2011. To appear.

[2] M. Gurevich and I. Keidar. Correctness of gossip-based membership under message loss. *SIAM J. on Comp.*, 39(8):3830–3859, Dec. 2010. Previous version in PODC'09.