Adapting to a Reliable Network Path

Baruch Awerbuch * Yishay Mansour †

ABSTRACT

We consider the model of unreliable network links, where at each time unit a link might be either up or down. We consider two related problems. The first, establishing end to end communication between two given nodes, where the performance measure is the average number of times the chosen path was disconnected. The second, is to build a spanning tree rooted at a given source node, where the performance measure is the average number of nodes that are the disconnected from the source. For both problems we design competitive algorithms.

1. INTRODUCTION

To motivate our setting consider a Quality of Service (QoS) network, such as ATM. When a source would like to build a communication to a given receiver it establishes a Virtual Path (VP) to that receiver. In order to support the desired QoS all the switches on the path have committed to allocated the required resources to support the establishment of the given VP and its QoS. Assume that after the establishment of the VP, there is a rather long task that uses the path (a long FTP, a conference call, a phone conversation, video transmission, Kazaa session etc.). During this time the situation in the network might change. For example some links might become inactive (due to malfunction). Alternatively, some of the switches may become highly loaded and are unable to support the VP. In such cases the VP might be torn down before completing the task. Clearly we would like to minimize such events as much as possible.

We consider two natural problems, *end to end path* and *source broadcast tree*. For the end to end path, we are given a source and destination and we like to establish a path between the two. (This abstracts the VP scenario.) For the source broadcast tree, we are given a source and like to establish a spanning tree rooted at the source. (This abstracts a broadcast from the source.)

Our model abstracts the link failures by assuming that at every time step a link may be either *up* or *down*. A path is disconnected if one of its links is down. A disconnected path incurs a cost of one, while a connected path has zero cost. When considering a tree, recall that each node in the tree has a unique path to the source. A node in a tree is disconnected if its path to the root is disconnected. The cost of a tree is the number of disconnected nodes from the source.

Our analysis is based on competitive analysis [11]. We compare the performance of our online algorithms to the best static offline. Namely the offline algorithm select either a fixed path (or a fixed tree) and uses it during all times steps. Our results bound the difference between the cost of the best static offline and our online algorithms. This difference is also called in the learning literature *regret*.

For both setting we derive competitive online algorithms, where the regret is proportional to the diameter of the graph and sub-linear in the number of steps. This implies that the average regret, the regret per time step, vanishes for long enough sequences. In such a case our dynamic polynomial-time online algorithms almost match the performance of the best static offline.

It is worth while to first examine the simple greedy algorithm (which picks the best static path so far). First, computing the optimal static offline is computationally hard, and therefore this intuitive algorithm does not run in polynomial time. Second, we show that the *ratio* between the performance of Greedy and the best static offline path can be as large as the maximal cut between the source and destination.

Related work. Near optimal algorithms for finding a shortest path or a minimum spanning tree in a graph, where at each time a different cost is assigned to each edge, were studied in [8, 12]. Those works (as does ours) are based on results from computational learning theory for the "best experts" and the "adversarial multi-arm bandit" problems. The above results have an additive cost function, and finding the optimal static offline can be done using any shortest path or minimum spanning tree algorithm. In contrast, in

*Johns Hopkins University, Baltimore, MD 21218. E-mail: baruch@cs.jhu.edu.

†School of Computer Science, Tel-Aviv University, Tel Aviv University, Tel-Aviv, 69978, Israel. E-Mail: mansour@tau.ac.il . Research supported in part by the Israel Science Foundation

our setting the cost function is non-linear (it is a threshold function). This non-linear behavior makes even computing the best static offline solution NP-hard.

There has been many works on dynamic networks. Some of the work applies to dynamically changing networks where reliable path between sender and received cannot ever be constructed [2], and thus packet switching model is used instead of circuit switched model in the current paper. The relevant measure is the number of packets lost. In contrast we use an underlying model of circuit switching network and as a complexity measure the number of disconnections. There is also another body of work [1, 4, 7, 6] which assumes that after some time the network stabilizes (no more faults occur) and try to achieve both correctness and performance (message and time complexity) starting at the end of the last fault. In contrast we assume that the faults are continuous.

Structure of this paper. In the rest of this paper we proceeds as follows. Model is presented in section 2. Greedy solution are described in Sec. 3. Existing work on learning framework and its derivatives are described in Sec. 4. Our algorithms are presented in Section 5. Section 6 presents the proofs.

2. MODEL

2.1 The network description

Network Structure. Consider a bi-directional communication network $G(V, E)$. We allow parallel edges to exists between nodes (which represent different possible direct connections between the nodes). For simplicity we assume existence of a global clock. We assume that delay of each edge is negligible compared to 1 time unit of that clock.

Fault model. For every time t and for every edge $e \in E$, we have a Boolean variable $f(e,t)$ indicating whether the edge e failed in time t (in which case $f(e,t) = 1$) or did not fail (in which case $f(e,t) = 0$). For a path P, the value $\texttt{Bcost}(P,t)$ is zero if all the edges on the path P are active at time t, and otherwise one. Formally, for a path $P =< e_1, \ldots, e_k >$ we define $\texttt{Bcost}(P,t) = 1 - \prod_{i=1}^{k}(1 - f(e_i,t))$.

Path acks. Messages are communicated via source routing: a sender s chooses at each time t a path P to use for each receiver r. If all edges of P are up at this time, i.e. $\texttt{Bcost}(P,t) = 0$, then message successfully arrives at the receiver, and receiver sends back ack to the source which arrives at the source prior to time $t+1$. Otherwise, there exists some edge $e_i \in P$ that is down at time t. Let e^* be the edge with the minimal index in P that has failed. In this case a *negative* ack arrives at the source prior to time $t+1$, from u_0 where $(u_0, v_0) = e^*$. Note that the negative ack pinpoints the closest failed edge to the source.

Notations. The following definition, of an averaging operator, would be convenient.

DEFINITION 2.1. *Let $\mathcal{T}$ be a set of times, and h any function defined for $t \in \mathcal{T}$. Let,*

$$E_{t\in\mathcal{T}}[h(t)] = \frac{1}{|\mathcal{T}|}\sum_{t\in\mathcal{T}} h(t).$$

2.2 Statement of Problem

The main problem that we would address in this paper is the end to end problem.

End-to-end path selection: Consider a source s and a destination r. At each time t the online algorithm at the source node s selects a path $P_t =< e_1, \ldots, e_k >$ from s to r. The goal is to minimize $\sum_t \texttt{Bcost}(P_t, t)$ based solely on past experience, namely values $\texttt{Bcost}(P_\tau, \tau)$ for $\tau < t$.

In order to address the difficulty of the end to end problem we define the following auxiliary problem, which is interesting on its own right.

Broadcast Source tree: Consider a source s. At each time t the online algorithm selects a spanning tree $W_t =< e_1, \ldots, e_k >$ rooted at s, which defines unique path $P_{W_t}(v)$ to each node v from s. At each time t a message is broadcast along W_t. In this case the ack mechanism return the number of nodes the message reached, or alternatively, the number of nodes that where disconnected at time t in W_t.

Equivalently, the feedback that algorithm receives is a set of edges $F_t \subset W_t$ that failed at time t, i.e. $f(e,t) = 1$. The subset F_t is the maximal subset such that no two edges $e_1, e_2 \in F_t$ are such that e_1 is part of the path from e_2 to s in W_t. Namely, F_t is the cut that separates the source s from all the nodes which are not reachable from s at time t using W_t. Given F_t we can decide for each node in W_t whether at time t is was connected to the source s through W_t. Again, the only information we will use is whether a node v was connected to the source s at time t.

For both algorithms we assume that the total duration T is known in advance to the algorithm. (In case this does not hold, we will have a small constant multiplicative factor in the performance of our algorithms.)

2.3 Our Results

Optimal static path: For every path P we define the aggregate binary cost

$$\texttt{Bcost}(P, \{0,T\}) = E_{t\in\{0,T\}}[\texttt{Bcost}(P,t)]$$

where $\{0,T\}$ is the set of integers $\{0,\ldots,T\}$. We would like to match performance of our online algorithm to the cost of the optimal static path, namely one minimizing aggregate (binary) cost $\texttt{Bcost}(P,\{0,T\})$ over all paths P of length at most $H < n$.

Finding an offline approximation such path is an NP-hard problem [10]. In a nutshell, one can reduce set packing to finding the best path. The set packing problem is as follows: we have a collection of sets $C = S_1, S_2, ..., S_m$ on n nodes, and we would like to find the maximum number of non-intersecting sets. We also note that independent set is a special case of set packing, and therefore the hardness of approximation of independent set carries over to set packing.

In order to derive a polynomial approximations of our inapproximable "binary fault cost", for our proofs and performance bounds we also use another measure for the path, which is the sum of the faults, i.e., $\texttt{Acost}(P_t, t)$ which is the number edges on the path P which failed at time t. Formally,

$\texttt{Acost}(P,t) = \sum_{i=1}^{k} f(e_i,t)$. Note that for any path P of length H and time t, we have $H \cdot \texttt{Bcost}(P,t) \geq \texttt{Acost}(P,t) \geq \texttt{Bcost}(P,t)$ and if $\texttt{Bcost}(P,t) = 0$ then $\texttt{Acost}(P,t) = 0$. For this metric, we can also define aggregate "additive fault cost", defined as $\texttt{Acost}(P,\{0,T\}) = E_{t\in\{0,T\}}[\texttt{Acost}(P,t)]$. The optimal *additive-cost* static path is the path P of length at most $H < n$ that minimizes $\texttt{Acost}(P,\{0,T\})$. From the above discussion it follows that the optimal additive-cost static path is an H-approximation for real "binary-cost" metric. Clearly, this would be interesting for short paths (e.g., constant or logarithmic length) and would give a reasonable approximation in such cases. In the following discussion, we will be evaluating our (polynomial-time) algorithms in terms of their proximity to both optimal binary and additive costs.

For the end to end path problem we derive an algorithm $E2EA$, whose performance is stated in the following theorem. Note that for large values of T the dominating part is $\texttt{Acost}(P,\{0,T\})$, since $H \leq n$. We derive the following performance bound for our algorithm.

THEOREM 2.2. *For any path P between s and r of length H in graph G,*

$$\begin{aligned}\texttt{Bcost}(E2EA) &\leq \texttt{Acost}(P,\{0,T\}) + \\ &\quad O\left(H\sqrt{n\log(Tn)/T^{1/3}}\right) \\ &\leq H\cdot\texttt{Bcost}(P,\{0,T\}) + \\ &\quad O\left(H\sqrt{n\log(Tn)/T^{1/3}}\right)\end{aligned}$$

Note that the above bound on the regret is non-trivial for $T = \Omega(H^6 n^3 \log^2 n)$, and as T increases the average regret vanishes.

Optimal static source broadcast tree For every tree W and node v we define the binary cost as

$$\texttt{Bcost}(W,v,\{0,T\}) = E_{t\in\{0,T\}}[\texttt{Bcost}(P(W,v),t)]$$

where $P(W,v)$ is the path in W that connects s to v. The optimal static source broadcast tree W is the tree that minimizes $\texttt{Bcost}(W,\{0,T\}) = \sum_{v\in V}\texttt{Bcost}(W,v,\{0,T\})$. Similarly, for every source broadcast tree W we define the additive cost as

$$\texttt{Acost}(W,\{0,T\}) = \sum_{v\in V} E_{t\in\{0,T\}}[\texttt{Acost}(P(W,v),t)].$$

THEOREM 2.3. *Consider a directed leveled graph G with H levels. Let W be the optimal static source broadcast tree for s. Then, there exists a randomized online algorithm SBTA, whose performance is*

$$\begin{aligned}\texttt{Bcost}(SBTA,\{0,T\}) &\leq \texttt{Acost}(W,\{0,T\}) + \\ &\quad O\left(nH\sqrt{deg_{in}(G)/T}\right) \\ &\leq H\cdot\texttt{Bcost}(W,\{0,T\}) + \\ &\quad O\left(nH\sqrt{deg_{in}(G)/T}\right),\end{aligned}$$

3. GREEDY SOLUTIONS FOR SOURCE ROUTING PATH SELECTION

Before presenting our algorithms it would be worth while to present a few alternative models and algorithms and discuss their benefits and weaknesses.

The first alternative model would be the *global information* model, where the algorithm at the end of each time step t receives for each edge e the value of $f(e,t)$. In this model we can discuss both the online scenario (where the online algorithm receives only information regarding previous time steps) and the offline scenario (where the algorithm receives all the information and need to compute an optimal static solution).

The greedy solution *GreedyBool* is natural: *select best performing path so far*. We remark that finding the path that is best so far is computationally hard; but suppose we are even allowed to use exponential resources. This exponential-time greedy algorithm *GreedyBool* selects $P_t = \mathsf{argmin} E_{\tau\in\{0,t-1\}}[\texttt{Bcost}(P,\tau)]$. We can also consider a polynomial greedy approximation *GreedyAdd*, that selects best path so far in *additive cost*, namely selects $P_t = \mathsf{argmin} E_{\tau\in\{0,t-1\}}[\texttt{Acost}(P,\tau)]$.

CLAIM 3.1. *For every graph $G = (V,E)$ any input sequence and any path P we have that*

$$\begin{aligned}\texttt{Acost}(\textit{GreedyBool},\{0,T\}) &\leq |E|\cdot\texttt{Bcost}(P,\{0,T\}) + |E| \\ \texttt{Acost}(\textit{GreedyAdd},\{0,T\}) &\leq |E|\cdot\texttt{Acost}(P,\{0,T\}) + |E|\end{aligned}$$

PROOF. Since after $|E|$ time steps where (either) Greedy fails to reach the destination the cost of the static optimal path increases by at least one, the claim follows. □

CLAIM 3.2. *For every graph $G = (V,E)$, and any nodes $s,r \in V$, let $MaxCut(s,r)$ be the maximum edge cut separating s from r. For any deterministic online algorithm DET there exists an input sequence and a path P for which,*

$$\texttt{Bcost}(DET,\{0,T\}) \geq |MaxCut(s,r)|\cdot\texttt{Bcost}(P,\{0,T\})$$

PROOF. Let $C = MaxCut(s,r)$ be a maximum edge cut between s and r. Since DET is a deterministic online algorithm, the adversary can compute, for every time t, the paths P_t that DET will use. Given P_t, let $e \in P_t \bigcap C$ be the edge in the cut that is being used by DET. At time t adversary set $f(e,t) = 1$ while for any other edge e' has $f(e',t) = 0$. Clearly, $\texttt{Bcost}(\text{DET},\{0,T\}) = T+1$. In addition there exists a path P that fails only $(T+1)/|C|$ times, which completes the proof. □

The above claim shows that no deterministic algorithm can not even guarantee a constant factor from the best static path, let alone a vanishing average regret. (A similar lower bound is known for deterministic multi-arm bandit algorithms.)

4. RESULTS USING TRADITIONAL LEARNING FRAMEWORK

In this section we will review a few basic concepts from computational learning theory, and their known solutions. This will not be by any means a comprehensive tutorial, and we will mainly focused on the essentials required to understand our work. We will also try and relate those results to the setting of the paper.

The first problem is the "best expert" problem. In that setting we have N experts. In each time step each expert suggests an action. The online algorithm needs to decide which action it will use. In parallel to the online algorithm selection of an action the adversary selects a loss of each expert. At the end of the step the online algorithm reveals its action and the adversary reveals the loss of each expert. The loss of the online algorithm at that time step is the loss of the expert it selected. The aim of the online algorithm is to minimize its total loss, i.e. the sum of the losses at each time step. (For simplicity we assume that the loss is a real number in $[0, 1]$.)

From the definition of the best expert problem it should be clear that in general we cannot hope to perform better than the best expert. (For example, it might be the case that all the experts are identical.) A surprising result is that there is an online algorithm that achieves a near optimal loss, as describe in the following theorem.

THEOREM 4.1 ([9, 5]). *There exists a randomized algorithm, BE, such that for any sequence of T time steps,*

$$E_{t\in\{0,T\}}[E[cost(BE(t))]] \leq \min_i\{E_{t\in\{0,T\}}[cost(A_i,t)]\} + O(\sqrt{N/T}),$$

where A_i is the ith expert.

The basic idea for the BE algorithm is to use for each expert a weight that depends exponentially on the loss of each expert. Let $L^t(A_i)$ be the loss of the ith expert up to time t. The weight of A_i would be $b^{L^t(A_i)}$ for some constant $b \in (0, 1)$ (which depends on the total number of time steps, T, but not on the input sequence). The algorithm BE simply selects expert A_i with probability proportional to its weight.

Another important problem is the adversarial "multi-arm bandit" problem. The setting is essentially the same as in the best expert setting. The only difference is that the online algorithm does not receive from the adversary the loss of all the experts, but rather it receives only the loss of the expert it selected. The following theorem shows that the `MAB` algorithm, presented in [3], can get an almost optimal performance.

THEOREM 4.2 ([3]). *There exists a randomized algorithm,* `MAB`*, such that for any sequence of T time steps,*

$$E_{t\in\{0,T\}}[E[cost(MAB(t))]] \leq \min_i\{E_{t\in\{0,T\}}[cost(A_i,t)]\} + \alpha\sqrt{(N\log N)/T},$$

for some constant $\alpha > 0$.

When we later use the `MAB` algorithm as a black-box, specifically we intend the algorithm of [3], for which Theorem 4.2 holds.

The basic idea behind `MAB` is to use the BE algorithm. However, we need to overcome the difficulty of receiving only partial feedback (namely only the loss of the action we selected). One way to get around this problem is to keep for each expert A_i a loss estimate ℓ_i^t. When we select action A_i and receive a cost of c_i we update ℓ_i^{t+1} to $\ell_i^t + c_i/p_i$, where p_i is the a priori probability that we select A_i at that time step. While it is easy to see that the expectation are now maintained, i.e., $E[\ell_i^t] = L^t(A_i)$, still one needs to take special care of the magnitude of the updates (and other resulting complications) as was done in [3].

For one of our algorithms we will need the following variant. Let $\texttt{MAB}'$ be an algorithm that uses the `MAB` algorithm as a black box and works in phases. Each phase is of length λ. At the start of each phase $\texttt{MAB}'$ uses the `MAB` to select an action, and remains with that action during the entire phase. At the end of the phase, $\texttt{MAB}'$ selects a random time t distributed uniformly overall the time steps in the phase. The $\texttt{MAB}'$ return to the `MAB`, at the end of the phase, as a feedback the cost incurred at time t, i.e., $cost(\texttt{MAB}'(t))$. The proof is that the number of effective time steps is T/λ, and in each one we decide on one action (used in λ steps). The offline is not changed, since it selects the same action in all time steps.

THEOREM 4.3. *For any sequence of T time steps, the $\texttt{MAB}'$ expected cost is,*

$$E_{t\in\{0,T\}}[E[cost(\texttt{MAB}'(t))]] \leq \min_i\{E_{t\in\{0,T\}}[cost(A_i,t)]\} + \alpha(\sqrt{\lambda(N\log N)/T})$$

for $T \geq k$, and some constant $\alpha > 0$.

The multi-arm bandit setting can be easily related to the setting of this paper; there are a number of ways to accomplish it.

- *source routing as multi-armed bandit problem w.r.t. all paths:* We can view each path (or tree, respectively) as an expert. Theorems 4.2 and 4.3 would imply the resulting performance bound. Unfortunately, the *number* of such experts, N, is exponential in the number of nodes. This causes two problems: one is computational and the other is performance. The computational problem is that we need to compute an exponential number of weights. a trivial lower bound for the multi-arm bandit problem is that we need to use each expert at least once (otherwise there will be some expert on which the algorithm does not receive any information). This causes a performance problem, namely we will need exponential number of trials before getting a meaningful bound.
- *packet routing as multi-armed bandit problem w.r.t. local outgoing ports:* at each node (router), whenever packet arrives at that node destined to a given receiver, router simply checks its outgoing ports, and

treats these ports as options for local multi-armed bandit problem. In this case, number of options K is outdegree $deg_{out} \leq n$ of a router, which is small, so we do not have a problem with exponential number of options. However, the feedback we receive, namely success of routing over this port, is completely garbled by potentially wrong decisions of downstream routers. It is easy to show that the number of errors in the algorithms is still $\Omega(2^n)$.

- *relaxing the problem by assuming full information:* this would make it easier. Specifically, suppose source knows about availability of every single edge at every single time in the past (which is totally unrealistic in network with many unreliable components). This becomes an *best experts* problem with $N = 2^n$ options, and additive mistake of only $O(\sqrt{\log N/T}) = O(\sqrt{n/T})$. Since even the offline problem is computationally intractable, it seems unreasonable to expect to be able to simulate this in polynomial time.
- *relaxing changing the problem further by assuming full information and considering additive cost:* Now the problem became much easier. Existing work by Takimoto and Warmuth [12] and subsequent work by Kalai and Vempala [8] do provide solutions with polynomial regret term of only $O(\sqrt{\log N/T}) = O(\sqrt{n/T})$ and polynomial computation.

5. OUR ALGORITHMS

In this work we intend to present polynomial performance bounds as well polynomial computation for the original problem, using the basic bandit framework of [3]. The main idea would is to use the special structure of our problem to reduce both the required computation and improve the performance. Exploitation of specific structure was also used in different context and for different purpose (just to achiever poly-time computation bound) in [12, 8]. In our network setting we will try to assign an expert to each node and measure its performance.

5.1 Notations

Suppose that packet traversing the network carries a hop count. When packet arrives to node v after traversing i hops, we consider the packet as if it arrives at "virtual" node (v, i).

Consider a directed leveled acyclic "virtual" graph $G'(V', E')$ where

$$V' = \{(v, i) | v \in V, i \leq H\}$$

and

$$E' = \{(v', i) \rightarrow (u', i+1) | (u, v) \in E\}$$

Furthermore, we can delete from the graph G' all the nodes that are not reachable from the source, and the edges outgoing from them.

We observe that directed leveled graph G' of depth H simulated any communication where packet traverses bounded number of hops $H \leq |V|$. Thus, for the rest of the paper, we consider, without loss of generality, that our network graph to be leveled, directed, acyclic, and every node is reachable from the source.

For a node v we denote by $deg_{in}(v)$ the in degree of node v, i.e. $deg_{in}(v) = |\{(u, v) \in E\}|$. For a graph G we denote by $deg_{in}(G)$ the maximum in degree of a node in G, i.e. $deg_{in}(G) = \max\{deg_{in}(v)\}$. A path P from s to r is a sequence of edges $e_1 = (s, v_1), e_2 = (v_1, v_2), \ldots, e_k = (v_{k-1}, r)$

5.2 Source Broadcast Tree Algorithm (SBTA)

Recall that in the source broadcast tree problem we like to create a tree that would maximize the number of nodes that are connected to the source. One can motivate such a construction as a subproblem in routing, where one like to create the best tree to which packets be routed to a given destination. Although the statement of the problem would allow to disconnect some node (for ever) our algorithm would try to avoid such pit falls. For every node the number of times its disconnected from the source is proportional to the number of times its disconnected from the source in the optimal static path (up to a factor of H).

For the source broadcast tree problem we like to minimize the number of average nodes that are disconnected from s. Formally we like to minimize

$$\begin{aligned} \texttt{Bcost}(W, \{0, T\}) &= \sum_{v \in V} \texttt{Bcost}(W, v, t) \\ &= E_{t \in \{0,T\}}[\texttt{Bcost}(P(W, v), t)]. \end{aligned}$$

Note that this is equivalent to minimizing the average number of disconnected nodes. Another piece of notation. For each node v, we denote by $\texttt{Bcost}(W, v)$ the average number of times it is disconnected from the source s using the tree W.

In order to gain a better intuition of our algorithm, assume we select some weights for the edges and like to build a minimum weight spanning tree. Since we have a directed acyclic graph G, and we know that all nodes are reachable from the source, we can make the following observation. The minimum spanning tree of G will include for each node the incoming edge of minimum weight. Formally, the minimum spanning tree includes for each node $v \in V$ the edge $e = (u, v) \in E$ such that $w(e) = \min_{(u',v) \in E}\{w((u', v))\}$. (If some other edge (u', v) participates in the minimum weight spanning tree, then we can replace it by e without increasing the weight of the tree. We are using here the fact that G is acyclic.)

The intuition for the source broadcast tree algorithm is to "reduce" it to a multi arm bandit problem. The main problem is that the decisions of the nodes distant from the source s are greatly influenced by the decisions of the nodes between them and the source. This is a classical case of "credit assignment" where we need to decide at a node whether the disconnection from the source is due to its decision (which would implicitly imply that we like to change the decision) or from the decision of the nodes between it and the source (in which case we do not need to penalize the action). In the source broadcast tree algorithm we take the approach to penalize the action of the node, each time the node is disconnected from the source. The correctness would follow from the fact that we can show, by induction on the levels, that nodes selects good edges, and thus define a good tree.

Formally, in the SBTA algorithm each node v runs an $\texttt{MAB}_v$

algorithm on its *incoming* links. Namely, the available actions are the incoming edges to v, i.e., $(u,v) \in E$. At each time t, every node v, using its $\texttt{MAB}_v$, selects an incoming link $e = (u,v)$. The collection of all the edges compose the tree W_t. (To see that W_t is a tree, note that each node, except s, has in-degree one, and the graph is acyclic. Therefore W_t is a tree that spans the entire network and is rooted in s.) When receiving the feedback at time t, we return to the $\texttt{MAB}_v$ at v the value of $\texttt{Bcost}(SBTA, v, t) = \texttt{Bcost}(P(W_t, v), t)$, i.e., we return 1 if node v was disconnected and 0 if it was connected.

Let the cost of the Source Broadcast Tree Algorithm (SBTA) at time t be $\texttt{Bcost}(SBTA, t) = \sum_{v \in V} \texttt{Bcost}(P(W_t, v), t)$, let $\texttt{Bcost}(SBTA, v, \mathcal{T}) = \sum_{t \in \mathcal{T}} \texttt{Bcost}(SBTA, v, t)$ and let $\texttt{Bcost}(SBTA) = \sum_{t \in \{0,T\}} \texttt{Bcost}(SBTA, t)$.

5.3 End to End Algorithm

In the end to end problem we need to establish a reliable connection from the source s to the receiver r. The reliability of the connection is measured in how many times the receiver was disconnected from the source. While in the source broadcast tree problem all the nodes are involved in every time step, in the end to end problem only a fraction of the nodes is involved (namely the nodes on the selected path from the source to the receiver).

The basic idea of the algorithm is to work in a similar way to the source broadcast tree algorithm, in fact we will show how to use the ideas of the $SBTA$ to solve the end to end problem. The main difference between the two settings is that in the end to end problem we select a path from s to r rather than a spanning tree. This implies that only some of the nodes are involved (in contrast, in the source broadcast tree all the nodes are involved). We receive feedback only in some of the nodes, the nodes on the selected path, while some nodes, those not on the selected path, do not receive any feedback in that time step.

To overcome this problem we use two ideas. The first idea is to use phases, and update the MABs at the nodes only at the end of the phase This implies that during a phase we have a single fixed tree. The second idea is to use exploration steps. The aim of the exploration steps is to sample the nodes that are not on the path in order to give them adequate feedback. We will show that with high probability all nodes get sampled during the phase, and thus we can run this modified source broadcast tree algorithm.

The main idea behind our $E2EA$ algorithm is to use the $SBTA$ algorithm. The idea is that the SBTA has a guarantee for each node, not only the resulting spanning tree. This implies that if we can use $SBTA$ we can have a guaranteed performance for each node. The obvious problem is that we do not have a feedback at each node at each time unit. In order to achieve a feedback for each node we will use exploration steps. In the exploration steps we will sample all the nodes in the graph, over time. This leads to the second idea, rather than update after each time step the resulting tree, as done in $SBTA$, we will update every phase. In addition we would like the cost of the exploration steps to be negligible in the total cost, since we will not be able to guarantee any real performance measure in those step. The only aim of the exploration steps is to gather information regarding the network behavior, and be able to return the appropriate feedback at all the nodes.

E2EA Algorithm: The E2EA algorithm has a parameter $q \in (0,1)$. We divide the time into phases, where the length of each phase is $\lambda = \Theta((n/q)\log n)$. Let $\tau_1, \tau_2, \ldots \tau_{T/\lambda}$ be the phases.

Beginning of a phase: Each node v has a $\texttt{MAB}'_v$ algorithm that selects an incoming edge. Namely, the actions available at node v are $(u,v) \in E$, i.e., the edges incoming to node v. Let W_{τ_i} be the set of edges selected in phase τ_i. As before, W_{τ_i} is a spanning tree rooted at s. Since W_{τ_i} is a spanning tree, each node v has in it a path from the root s, namely $P(W_{\tau_i}, v)$.

During a phase: In each time t we flip a coin, and with probability q we perform an `exploration step` and with probability $1-q$ we perform an `exploitation step`. This implies that in a phase we have, with high probability, we have $\Theta(n \log n)$ exploration steps and the rest are exploitation steps.

Exploration step: In an exploration step we select a random node v. (The size of a phase, λ, is selected such that with high probability each node is explored at least once in every phase.) Given v we build the path from s to r as follows: We start from s and reach v using $P(W_{\tau_i}, v)$. From v we select any path to reach the destination r. Let $\texttt{Bcost}(v,t) = 0$ denote that $P(W_{\tau_i}, v)$ was operating, and $\texttt{Bcost}(v,t) = 1$ denote that it was disconnected. (Note we are looking only whether we reach the selected node v and not the destination r.) Let $\mathcal{T}(\tau_i, v)$ be the time steps when we sampled v in exploration steps.

Exploitation step: In an exploitation step we simply select the path from s to r in W_{τ_i}, i.e. $P_t = P(W_{\tau_i}, r)$.

End of a phase: At the end of each phase, for each node v we select a random $t \in \mathcal{T}(\tau_i, v)$ and returns $\texttt{MAB}_v$ the value of $\texttt{Bcost}(v,t)$, denote this value by $r(v,t)$.

6. ANALYSIS OF OUR ALGORITHMS

6.1 Analysis of Source-Tree Algorithm (SBTA)

For the analysis we will relate the number of times a node v was disconnected from the source s to its distance from the source. The proof would be by induction on the distance from the source.

LEMMA 6.1. *Let v be a node at distance H from s and W an arbitrary spanning tree. Then,*

$$\texttt{Bcost}(SBTA, v, \{0,T\}) \leq \texttt{Acost}(W, v, \{0,T\}) + H[\alpha\sqrt{deg_{in}(G)/T}],$$

for some constant $\alpha > 0$.

PROOF. The proof is by induction on H. For $H = 1$ we are simply running the MAB algorithm and the bound is directly from Theorem 4.2. Assume the claim hold for $H-1$ and show it for H. Let $u_1, \ldots u_k$ be the neighbors of v in

level $H-1$. By the induction hypothesis, we have that u_i is disconnected from s at most

$$c_i = \texttt{Acost}(W, u_i, \{0,T\}) + (H-1)[\alpha\sqrt{deg_{in}(G)/T}]$$

Let u^* be the neighbor u_i that minimizes,

$$E_{t\in\{0,T\}}\left[\max\{f((u_i,v),t), \texttt{Bcost}(SBTA, u_i, t)\}\right] = c^*$$

This implies that at node v, the MAB has an action, namely (u^*, v), whose cost is *exactly* c^*. Note that since

$$\max\{f((u^*,v),t), \texttt{Bcost}(SBTA, u^*, t)\} \leq f((u_i,v),t) + \texttt{Bcost}(SBTA, u_i, t)$$

we have

$$E_{t\in\{0,T\}}[\max\{f((u^*,v),t), \texttt{Bcost}(SBTA, u^*, t)\}] \leq E_{t\in\{0,T\}}[f((u^*,v),t)] + E_{t\in\{0,T\}}[\texttt{Bcost}(SBTA, u^*, t)]$$

It follows that for any u_i,

$$c^* \leq c_i + E_{t\in\{0,T\}}[f((u^*,v),t)]$$

By Theorem 4.2 we have that the cost of the MAB in v is bounded by

$$\begin{aligned} c^* + [\alpha\sqrt{deg_{in}(v)/T}] &\leq \texttt{Acost}(W, v, \{0,T\}) + \\ &\quad H[\alpha\sqrt{deg_{in}(G)/T}], \end{aligned}$$

since

$$\texttt{Acost}(W, v, \{0,T\}) \geq \min_i\{\texttt{Acost}(W, u_i, \{0,T\}) + E_{t\in\{0,T\}}[f((u_i,v),t)]\},$$

which completes the proof of the lemma. □

We relate the cost of the online algorithm to the cost of the best additive cost tree (which is simply a minimum weight spanning tree, where the weight of an edge is the frequency that with which its fails). Also, the average regret is becoming negligible as the number of time steps T increases. The following theorem performance bound for SBTA.

THEOREM 6.2. *Consider a directed leveled graph G with H levels. Let W be the optimal static source broadcast tree for s. Then,*

$$\begin{aligned} \texttt{Bcost}(SBTA, \{0,T\}) &\leq \texttt{Acost}(W, \{0,T\}) + \\ &\quad nH[\alpha\sqrt{deg_{in}(G)/T}] \\ &\leq H\texttt{Bcost}(W, \{0,T\}) + \\ &\quad nH[\alpha\sqrt{deg_{in}(G)/T}] \end{aligned}$$

for some constant $\alpha > 0$.

For the End to End problem we will need the following variant of $SBTA$. Let $SBTA'$ use the $SBTA$, and have phases of length λ. At the start of each phase, $SBTA'$ uses $SBTA$ to select a spanning tree, which will remain unchanged during the phase. At the end of the phase, $SBTA'$ gives $SBTA$ a feedback at each node as follows. At each node v, $SBTA'$ it selects a time t uniformly distributed over the time steps in the phase. $SBTA'$ returns $\texttt{Bcost}(v,t)$ as the feedback to $\texttt{MAB}_v$ at v. Namely, $SBTA'$ uses MAB′ rather than MAB. (The times of the selected feedbacks at different nodes may be correlated, but when we consider each node separately, the time of the feedback is distributed uniformly in the phase.) Using Theorem 4.3 we derive,

COROLLARY 6.3. *Let v be a node at distance H from s and W an arbitrary spanning tree. Then,*

$$\begin{aligned} \texttt{Bcost}(SBTA', v, \{0,T\}) &\leq \texttt{Acost}(W, v, \{0,T\}) + \\ &\quad H[\alpha\sqrt{\lambda deg_{in}(G)/T}], \end{aligned}$$

for some constant $\alpha > 0$.

6.2 Analysis of end-to-end

First we claim that with high probability each node is sampled at least once in every phase.

CLAIM 6.4. *For $\lambda = \Omega((n/q)\log Tn/\lambda\delta)$, with probability $1-\delta$, during each phase, each node v is sampled at least once.*

For a given node v, a-priori, all time steps in a phase are equally likely to be the one sampled. This implies that $r(v,t)$ is an unbiased sample of the feedbacks that v would receive in the phase.

CLAIM 6.5. *For each phase τ_i and each node v, assuming node v was sampled in τ_i, we have*

$$r(v,t) = E_{t\in\tau_i}[\texttt{Bcost}(v,t)]$$

We have established that each node is sampled once in every phase and the samples are unbiased. This implies that the MAB′ that is in the node, would be updated in the same way it would be updated in $SBTA'$. Therefore can use Corollary 6.3 and derive the following.

LEMMA 6.6. *Let v be a node at distance H from s and P an arbitrary path from s to v. Then,*

$$\begin{aligned} \texttt{Bcost}(E2EA, v, \{0,T\}) &\leq \texttt{Acost}(P, \{0,T\}) + \\ &\quad H[\alpha\sqrt{\lambda deg_{in}(G)/T}], \end{aligned}$$

for some constant $\alpha > 0$.

The above lemma can be used to bound the cost when in ever time step we build a path to node r. However, this is not the case. We build the path to r in the exploitation steps, while in the exploration steps we build very different paths. Therefore, to establish the performance of the $E2EA$ we need to add the cost of the exploration steps (and in the worse case they all might have cost 1). Adding the cost of the exploration steps we can derive the following.

THEOREM 6.7. *Let G be a graph with H levels and P any path connecting s to r. Then,*

$$\begin{aligned} \texttt{Bcost}(E2EA) &\leq \texttt{Acost}(P, \{0,T\}) + \\ &\quad H[\alpha\sqrt{\lambda deg_{in}(G)/T}] + qT, \end{aligned}$$

for some constant $\alpha > 0$.

The final step in the algorithm is to set the parameter q. By setting $q = n/T^{2/3}$, and noting that $deg_{in}(G) \leq n$, we derive the following performance bound for the $E2EA$.

THEOREM 6.8. *Let G be a graph with H levels and P any path connecting s to r. Then,*

$$\texttt{Bcost}(E2EA) \leq \texttt{Acost}(P, \{0, T\}) + O\left(H\sqrt{n\log(Tn)/T^{1/3}}\right)$$

For the bound in the above theorem to be interesting we will need that $T = \Omega(H^6 n^3 \log^2 n)$. Note that Lemma 6.6 derives that not only we compete with the best path to the receiver r, but in fact we compete simultaneously with the best path to *any* node v.

7. REFERENCES

[1] Yehuda Afek, Baruch Awerbuch, and Eli Gafni. Applying static network protocols to dynamic networks. In *Proc. 28th IEEE Symp. on Found. of Comp. Science*, pages 358–370, October 1987.

[2] Yehuda Afek, Baruch Awerbuch, Eli Gagni, Yishay Mansour, Adi Rosen, and Nir Shavit. Slide-the key to polynomial end-to-end communication. *J. Algorithms*, 22(1):158–186, 1997.

[3] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. Gambling in a rigged casino: the adversarial multi-armed bandit problem. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 322–331. IEEE Computer Society Press, Los Alamitos, CA, 1995.

[4] Baruch Awerbuch. On the effects of feedback in dynamic network protocols. In *Proc. 29th IEEE Symp. on Found. of Comp. Science*, pages 231–245, October 1988.

[5] Nicolò Cesa-Bianchi, Yoav Freund, David P. Helmbold, David Haussler, Robert E. Schapire, and Manfred K. Warmuth. How to use expert advice. pages 382–391, 1993. To appear, *Journal of the Association for Computing Machinery.*

[6] Steven G. Finn. Resynch procedures and a fail-safe network protocol. *IEEE Trans. on Commun.*, COM-27(6):840–845, June 1979.

[7] Pierre A. Humblet and Stuart R. Soloway. A fail-safe layer for distributed network algorithms and changing topologies. Submitted to IEEE Trans. Comm., May 1987.

[8] Adam Kalai and Santosh Vempala. Geometric algorithms for online optimization, 2003. unpublished manuscript.

[9] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:212–261, 1994. A preliminary version appeared in FOCS 1989.

[10] Adam Meyerson. Private communication.

[11] Sleator and Tarjan. Amortized efficiency of list update and paging rules. *Communication of the ACM*, 28(2):202–208, 1985.

[12] Eiji Takimoto and Manfred K. Warmuth. Path kernels and multiplicative updates. In *COLT Proceedings*, 2002.