GRiDA: GRen Distributed Algorithm for energy-efficient IP backbone networks

Aruna Prem Bianzino\textsuperscript{a,b,*}, Luca Chiaraviglio\textsuperscript{a,c}, Marco Mellia\textsuperscript{a,c}, Jean-Louis Rougier\textsuperscript{b}

\textsuperscript{a}Consorzio Nazionale Interuniversitario per le Tele comunicazioni, Torino, Italy
\textsuperscript{b}Institut TELECOM, TELECOM ParisTech, Paris, France
\textsuperscript{c}Dipartimento di Elettronica, Politecnico di Torino, Torino, Italy

\section*{A R T I C L E  I N  F O}

\textbf{Article history:}
Received 2 March 2012
Received in revised form 21 June 2012
Accepted 24 June 2012
Available online xxxx

\textbf{Keywords:}
Green networks
Distributed algorithms
Sleep mode

\section*{A B S T R A C T}

In this work, we face the problem of reducing the power consumption of Internet backbone networks. We propose a novel algorithm, called GRiDA, to selectively switch off links in an Internet Service Provider IP-based network to reduce the system energy consumption. Differently from approaches that have been proposed in the literature, our solution is completely distributed and thus not requiring any centralized oracle. It leverages link-state protocol like OSPF to share a limited amount of information, and to reduce the problem complexity. Another key feature of GRiDA is that it does not require the knowledge of the actual and past/future traffic matrices, being able to run in real-time, where this information would not be available. Results, obtained on realistic case studies, show that GRiDA achieves performance comparable to several existing centralized algorithms, guaranteeing energy savings up to 50%.

\section*{1. Introduction}

According to different studies\cite{2,3}, the carbon footprint of Information and Communication Technologies (ICTs) is constantly increasing, representing today up to 10\% of the global CO2 emissions. Among the main ICT sectors, 37\% of the total emissions are due to telecommunication infrastructures and their devices, while data centers and user devices are responsible for the remaining part\cite{2}. It is therefore not surprising that researchers, manufacturers and network providers are spending significant efforts to reduce the power consumption of ICT systems from different angles.

To this extent, networking devices waste a considerable amount of power. In particular, energy consumption has always been increased in the last years, coupled with the increase of the offered performance\cite{4}. Actually, power consumption of networking devices scales with the installed capacity, rather than the current load\cite{5}. Thus, for an Internet Service Provider (ISP) the network power consumption is practically constant, unrespectively to traffic fluctuations, since all devices consume always the same amount of power. In turn, devices are underutilized, especially during off-peak hours when traffic is low. This represents a clear opportunity for saving energy, since many resources (i.e., routers and links) are powered on without being fully utilized, while a carefully selected subset of them can be switched off or put into sleep mode without affecting the offered Quality of Service (QoS).

In the literature, different approaches have been proposed to reduce the gap between the capacity offered by the network and the resources consumed by users (see\cite{4,6} for an overview). The proposed approaches can be divided into two main categories: power proportional techniques that adapt the capacity (and thus consumption)
of the devices to the actual load, and sleep mode approaches, that leverage on the idea of introducing idle mode capabilities. While the first approach involves deep modifications in the design of hardware components, the second approach requires coordination among networking devices to carefully re-route the traffic that results from putting into sleep mode some devices.

In this paper, we face the problem of reducing power consumption in backbone networks adopting the sleep mode approach. The intuition has been already proposed in the literature, starting from the seminal work of Gupta et al. [7]. In particular, approaches ranging from traffic engineering [8], to routing protocols [9], and new architectures [10] have been proposed. These works tackle the minimization of network power consumption by powering off elements, such as routers and links, and large savings are possible when sleep mode states are exploited. However, to the best of our knowledge all of the previous works either ignore the traffic flowing in the network [9], or assume the complete knowledge of the traffic matrix at each given time [8,11–13]. Similarly, all the previous solutions are completely centralized [11] or require at least the presence of a control node [10] which has the full knowledge of each device status. Thus, the applicability of the aforementioned approaches is limited to specific cases.

In our work, we follow a different approach: we propose a novel distributed on-line algorithm, called GRiDA (GReen Distributed Algorithm), to put into sleep mode links in an IP-based network. Our solution is fully distributed among the nodes to: (i) limit the amount of shared information, (ii) limit coordination among nodes to only Link-State Advertisement (LSA) exchanges as provided by, e.g., OSPF, and (iii) reduce the problem complexity. Contrary to previous works, we assume that nodes do not know the traffic matrix, whose real-time knowledge is indeed unrealistic in the current pure-IP networks. Thus, the switching off is fully decentralized to the single nodes, which take local decisions at random intervals without any coordination or synchronization among them. The solution results thus to be more robust and simpler to implement with respect to the centralized approaches proposed so far.

We assume local decisions to be based only on the knowledge of the current load and power consumption of node incident links, and on the knowledge of the current network topology, assured by a link-state routing algorithm, e.g., OSPF or IS-IS. Contrary to what usually hypothesized by other works in the literature, no knowledge of the current traffic demand is required to guarantee a proper reaction to faults and changes in the traffic demand. The process of link switching off/on is fully decentralized to the single nodes, which take local decisions at random intervals without any coordination or synchronization among them. The solution results thus to be more robust and simpler to implement with respect to the centralized approaches proposed so far.

We assess the effectiveness of our solution on realistic case studies and real topologies. Results show that GRiDA achieves performance comparable with the centralized solutions that act off-line, assuming the perfect knowledge of the traffic matrix. In this work we propose an improved version of the solution, and a more complete set of results, with respect to the one presented in our previous work [1]. The main differences of this work with respect to [1] are the following ones: (i) we improve the exploration capabilities of the algorithm by updating the penalty associated to choices at each choice step (rather than at each LSA arrival), (ii) we change the objective function by taking into account normalized power consumption rather than absolute values, (iii) we define a new set of metrics to assess the algorithm performance, and (iv) we extend our analysis with a complete sensitivity analysis of the algorithm parameters.

The paper is organized as follows: the description of the algorithm is reported in Section 2. The realistic case studies considered for algorithm evaluation are described in Section 3. Section 4 presents the simulation results, and discusses the implementation issues. Finally, conclusions are drawn in Section 5.

### 2. Algorithm description

The GRiDA algorithm aims at reducing the network power consumption by adapting the network capacity to the current traffic demand. In particular, it (i) switches off links whenever they are underutilized, and whenever their absence in the network does not affect the network functionalities, and (ii) switches on idle links when capacity is required to guarantee a proper reaction to faults and changes in the traffic demand. The process of link switching off/on is fully decentralized to the single nodes, which take local decisions at random intervals without any coordination or synchronization among them. The solution results thus to be more robust and simpler to implement with respect to the centralized approaches proposed so far.

We assume local decisions to be based only on the knowledge of the current load and power consumption of node incident links, and on the knowledge of the current network topology, assured by a link-state routing algorithm, e.g., OSPF or IS-IS. Contrary to what usually hypothesized by other works in the literature, no knowledge of the network wide traffic matrix is assumed.

In GRiDA, LSA messages distribute information about the current network topology, augmented by information about eventual congestion in the network, i.e., link load overcoming a threshold ((\(\phi\))), or presence of disconnected source/destination pairs. LSAs are delivered to nodes at fixed time intervals ((\(\Delta_{LSA}\))), selected by the network administrator, according to OSPF or IS-IS configuration.

We represent the network infrastructure as a di-graph \(G = (V, E)\), where \(V\) is the set of vertices and \(E\) is the set of edges. Vertices represent network nodes, while edges represent network links, being \(N = |V|\) and \(L = |E|\) the number of nodes and links respectively.

#### 2.1. Node choice

A decision of a node \(n\) corresponds to entering a specific node configuration \(K^{(n)} \in K^{(n)}\), where \(K^{(n)}\) is the set of all possible configurations for node \(n\); a configuration \(K^{(n)}\) is a combination of on/off states for incident links. More formally, given a node \(n\), of degree \(d^{(n)}\); and an ordered list of

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>ISP1</th>
<th>Geant</th>
<th>ISP2</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRiDA</td>
<td>49.84</td>
<td>46.47</td>
<td>39.43</td>
</tr>
</tbody>
</table>

Table 1

Algorithm comparison: average energy saving in percentage.
the incident links (in lexicographical order), a configuration is the vector \((k_{i}^{(n)}, \ldots, k_{d}^{(n)})\) of the configurations of the \(d^{(n)}\) incident links. The configuration \(k_{i}^{(n)}\) of a link \(l\) is a binary variable indicating the state of the link: \(k_{i}^{(n)} = 0\) if the link is powered off, and \(k_{i}^{(n)} = 1\) if the link is powered on. Therefore \(|K_{n}| = 2^{d^{(n)}}\).

The status \(S\) of a node \(n\) is the vector \((s_{1}^{(n)}, \ldots, s_{d}^{(n)})\) of the status associated to all the \(d^{(n)}\) links incident to \(n\). For each link \(l\) the status \(s_{l}^{(n)}\) may assume two possible values, defined on the bases of the link load \((\rho^{(l)})\): off, i.e., the link is powered off or powered on but not used \((\rho^{(l)} = 0)\), or normal, i.e., the link is used \((\rho^{(l)} > 0)\).

A utility function is defined as: \(U(K^{(n)}, S^{(n)}) = c(K^{(n)}) + p(K^{(n)}, S^{(n)})\), where \(c(K^{(n)})\) is the power consumption of node \(n\) computed as the sum of the power consumed by the link in on-status in configuration \(K^{(n)}\), normalized to the sum of the power of all incident links \((i.e., c(K^{(n)}) = \sum_{l=1}^{d^{(n)}}, \frac{k_{i}^{(n)} c_{i}}{\sum_{l=1}^{d^{(n)}} c_{l}}\), where \(c_{i}\) is the power consumption of link \(l\), see Section 3.1 for details), and \(p(K^{(n)}, S^{(n)})\) is a penalty associated to the configuration on the basis of the current status and the learning. Since the same procedure is applied to all nodes, from now on we get rid of the index \(n\) for ease of notation.

For a single node, the problem turns into selecting the best configuration that minimizes the utility function, while guaranteeing the global system to work properly. This problem can be solved by the support of the Q-learning technique [14], as the node choice is a function of the current status of the same node, and each possible choice is associated to an estimated utility function, updated by learning. Hence, node decisions in normal network working state correspond to the \(K\) minimizing \(U(K,S)\). To ensure fast reaction to faults and sudden traffic changes, we introduced three safety mechanisms:

- a connectivity check is performed on the network topology resulting from the chosen configuration, through a breadth-first search. This means that GRIDa always ensures full connectivity among nodes, i.e., at least one link per node is always powered on. If a choice would lead to a network disconnection, it is not applied and its penalty is updated with the additive factor \(\beta\) as if a violation occurs (detailed in Section 2.2),
- if a choice taken in a non-congestion network state is followed by a congestion advertised by an LSA, the choice is regretted, i.e., the node returns to the previous configuration, and the penalty corresponding to that choice is updated with an additive factor \(\beta\),
- in a congestion network state, a node which is taking a decision will automatically select the all-on configuration. This choice is not subject to the regret mechanism dependent on the following received LSA.

The algorithm exploits the Q-learning method in the case of multiagents, since the utility (cost) of each agent (node) is affected by the actions of the other agents. According to [15] theoretical convergence is not guaranteed, since the environment is non-stationary due to adaptation of other agents (nodes). However, the introduced safety mechanisms limit the impact of unstable behavior, while guaranteeing a good level of exploration of the possible choices for each node. Furthermore, in the case of the real network scenarios we evaluated, the algorithm results to converge to a stable solution, when learning parameters are properly set (see Section 4 for details).

### Algorithm 1. The pseudo-code of the node choice event.

#### Node Choice

**Input:** \(K^{old}, S\)

**Output:** \(K, K^{old}, S^{old}\)

\(S^{old} = S\)

if lastLSA == OK:

\(K' = \text{min}_{K} U(K,S)\)

for \(J\) in \(K\):

\(p(J,S) = U(K,S) \cdot \delta\)

if (connectivity_check \((K') == \text{OK})\):

\(K = K'\)

if \(K \neq K^{old}\):

\(\text{to be checked} = \text{TRUE}\)

else:

\(p(K',S) = p(K',S) + \beta\)

else:

\(K = \text{all-on configuration}\)

#### 2.2. Penalty evolution

The values of \(p(K,S)\) are updated step-by-step, on the basis of the learning: if the decision of entering configuration \(K\) when in status \(S\) is followed by an LSA reporting a network critical state, the cost associated to that choice (i.e., \(p(K,S)\)) is incremented by an additive factor \(\beta\ (\geq 0)\):

\[ p(K,S) = p(K,S) + \beta \tag{1} \]

Note that the power consumption is normalized when considered in the utility function, so that \(\beta\) has the same impact on each node, independently from its power consumption. Note also that the penalty is increased by \(\beta\) when the connectivity check fails.

If a decision is taken in a status \(S\) and no violations have been reported by the previous LSA, the penalty associated to decisions in status \(S\) (i.e., \(p(*,S)\)) are decremented by a multiplicative factor \(\delta \in [0, 1]\):

\[ p(J,S) = p(J,S) \cdot \delta \forall J \in K_{n} \tag{2} \]

The pseudo code for the decision process is reported in **Algorithm 1**, where \(S\) is the current status of the node.

Intuitively, (1) penalizes choices which likely caused violations of connectivity or capacity constraints; (2) pushes nodes toward the exploration of all the possible choices by reducing the effect of the accumulated memory, since the factor \(\delta\) is applied to all \(p(*,S)\).

The pseudo code describing the procedure executed by nodes at LSA arrivals, and the corresponding penalty updates for choices which brought to violations, is reported in **Algorithm 2**, where \(K\) is the current node configuration, \(K^{old}\) is the node configuration before the last choice, \(S^{old}\) is
the node status at the time the last choice has been taken, and \( p \) is the penalty state of the node.

**Algorithm 2.** The pseudo-code of the LSA arrival event.

```
<table>
<thead>
<tr>
<th>LSA Arrival</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> ( K, S_{\text{old}}, p )</td>
</tr>
<tr>
<td><strong>Output:</strong> ( K, p )</td>
</tr>
<tr>
<td>if to_be_checked == TRUE:</td>
</tr>
<tr>
<td>if LSA! == OK:</td>
</tr>
<tr>
<td>( p(K, S_{\text{old}}) = p(K, S_{\text{old}}) + \beta )</td>
</tr>
<tr>
<td>( K = K_{\text{old}} )</td>
</tr>
<tr>
<td>to_be_checked = FALSE</td>
</tr>
</tbody>
</table>
```

2.3. Algorithm initialization

In order to speed up convergence, the cost function \( p(K, S) \) is properly initialized. The intuition is to discriminate between (i) switching off an unloaded link, or (ii) switching off a link which is carrying traffic (which can be less safe for the network functionality). In addition, we need to avoid multiple attempts of radical switching off choices during convergence by further penalizing configurations with an higher number of off links and link loads larger than zero.

More formally, an initial penalty function \( \theta_i(k_i, s_i) \) is associated to each link \( l \) in each possible status \( s_i \in S \) entering each possible configuration \( k_i \in K \):

\[
\theta_i(k_i, s_i) = \begin{cases} 
0 & s_i = \text{off} \lor k_i = 1 \\
1/d & \text{else}
\end{cases}
\] (3)

The \( 1/d \) factor is a normalization over the node degree. Then, the penalty \( p(K, S) \) is initialized to \( \sum_{i \in n} \theta_i(k_i, s_i) \). The procedure is repeated for all nodes \( n \in V \), and for all configurations \( K \in K \) and all status \( S \in S \).

2.4. Solution complexity

Making a choice at a given node implies the following three steps: (i) access to the penalties corresponding to the current status, (ii) computation of the utility corresponding to all the possible configurations from the current status, and (iii) executing a connectivity check.

First, we analyze the *time complexity* of our solution. The first step implies, in the worst case, to find the correct memory entry among all the \( 2^d \) possible statuses, which can be done through a binary search tree in a time \( O(\log 2^d) = O(d) \). The computation of the utility function is simply the sum of the penalty and of the energy cost of the incident links, which should be computed for all the \( 2^d \) possible configurations, resulting in a time \( O(d2^d) \). Finally, the connectivity check on the chosen configuration results in a time \( O(N + L) = O(N + dN) = O(dN) \), considering a breadth-first search. Summing the contribution of the three steps, the time complexity of the solution results in \( O(d2^d + dN) \), scaling linearly with network size \( N \), and exponentially with node degree \( d \). As it has been shown in [16], the node degree is actually limited in real network scenarios, thus it does not represent a critical issue.

For what concerns the solution *space complexity*, instead, a node needs to store in the worst case, for each possible status, a penalty for each possible configuration, resulting in a matrix of \( 2^d \times 2^d = 4^d \) memory entries. Actually, simulation results show that less than 10% of the entries are visited on average (see Section 4.2), and hence just a minimal amount of memory is required. Thus, rather than storing the entire matrix, compact structures may be adopted to reduce the size of the matrix.\(^1\)

3. Scenario description

To provide a relevant evaluation of the described algorithm, we tested it over 3 different scenarios, ranging from a metropolitan segment network to a European-wide network.

3.1. The power and traffic model

We are interested in the power consumption related to links which includes the power consumption of the router ports and of the electronic regenerators along the link. To have comparable results, we adopted here the same power model used in [11,17]. We consider ports consuming \( c_{\text{nic}} = 50 \text{ W} \) for each \( B_{\text{CAP}} = 1 \text{ Gbps} \) of link capacity, and regenerators consuming \( c_{\text{reg}} = 1 \text{ kW} \) for each \( B_{\text{CAP}} = 10 \text{ Gbps} \) of link capacity, with a regenerator every \( m^2 = 70 \text{ km} \). Therefore, the power consumption \( c_i(l) \) of a link \( l \) with capacity \( B_{\text{CAP}} \) and length \( m(l) \), results in: \( c_i(l) = \frac{g(l)}{m(l)} c_{\text{nic}} + 2 c_{\text{reg}} \).

In our simulations, traffic requests are constant over fixed time intervals \( A_{\text{sim}} \), after which a new traffic matrix is considered. Traffic is expected to change on moderate

\(^1\) Notice that also the initial penalty matrix \( \theta \) can be efficiently stored in a compact format since it is only based on the set of available configurations and states.
time scale, so that \( T_{IM} = 30' \) or higher. The traffic matrices have been obtained from direct traffic measurements where available; otherwise, they are computed starting from a single measured traffic matrix and imposing an artificial traffic profile.

### 3.2. The network scenarios

**ISP 1:** The first testing scenario is an access/metropolitan segment of a traditional telecom operator network [18]. The topology is reported in Fig. 1, where nodes are represented by circles. Labels represent node IDs. This topology includes access nodes (IDs 1–8), which are sources and destinations of traffic requests, transit nodes (IDs 9–21), performing only traffic switching, and a peering node (ID T), providing access to the ISP transport network and the Internet. The small black squares in Fig. 1 indicate the presence of regenerators on links.

For this scenario, an actual traffic matrix has been provided. The maximum link utilization is guaranteed to be smaller than 70% (\( \phi = 0.7 \)), and 47 traffic matrices have been generated applying the sinusoidal traffic profile described in [11], and represented in Fig. 2 (left) by the green dashed line labeled “ISP 1”.

**Geant:** We consider the actual Geant network [19], whose topology is reported in Fig. 2 (right). Nodes are represented by circles, while black squares indicate the presence of regenerators, whose number is reported as label. All nodes are sources and destinations of traffic. For this network topology, actual traffic matrices are publicly available, among which we selected the 48 traffic matrices of 05/05/2005 (a typical working day). The corresponding variation in terms of total traffic load is reported in Fig. 2 (left) by the red continuous line.

**ISP 2:** Finally, we considered a topology inspired by the national network of an ISP (see [11] for details). It is a hierarchical network composed of 373 nodes, organized in five levels: core, backbone, metro, access and Internet nodes. The core level is composed by few nodes densely interconnected by high-capacity links, and offering connectivity to the Internet by means of a peering node. Going down in the hierarchical levels, the number of nodes increases, and the link capacity decreases.

The access nodes and the Internet peering node are sources and destinations of traffic. The traffic requests for this topology have been generated following a measured traffic profile (reported in Fig. 2 (left) by the blue dotted line), as described in [11].

### 3.3. Parameter setting

Fig. 3A new traffic matrix is considered every time interval \( T_{IM} \). A randomly selected node is waken up to take a decision every random interval \( T_c \), uniformly distributed between the LSA interval \( T_{LSA} \) and \( T_{Max} \). Time intervals must be chosen in order to have, on the one hand, at least one LSA occurrence between two consecutive decisions, and on the other hand, a significant number of decisions per node to allow algorithm convergence. Note that LSA timings are compliant with current OSPF specifications [20]. On average, a single node takes a decision every \( T_c \times N \), where \( N \) is the number of nodes in the network.

Values for the parameters in the different simulation scenarios are summarized in Table 2. The number of nodes for the ISP 2 network reflects the fact that core, backbone, metro nodes are running the GRiDA algorithm, while access and Internet nodes are not running the GRiDA algorithm. Indeed, access nodes in the ISP 2 network are not connected among them, hence, it is not necessary to run GRiDA on those nodes.

### 4. Performance evaluation

We implement GRiDA on a custom event-based simulator written in python and C languages. Node choices, LSA arrivals and traffic matrix changes are the possible events. Moreover, network statistics including link load, node configurations and power consumption are stored in a log. Unless otherwise specified, we simulate a time period between the LSA interval \( D \) and \( T_{Max} \). Time intervals between the LSA occurrence between two consecutive decisions, and on the other hand, a significant number of decisions per node to allow algorithm convergence. Note that LSA timings are compliant with current OSPF specifications [20]. On average, a single node takes a decision every \( T_c \times N \), where \( N \) is the number of nodes in the network.

Values for the parameters in the different simulation scenarios are summarized in Table 2. The number of nodes for the ISP 2 network reflects the fact that core, backbone, metro nodes are running the GRiDA algorithm, while access and Internet nodes are not running the GRiDA algorithm. Indeed, access nodes in the ISP 2 network are not connected among them, hence, it is not necessary to run GRiDA on those nodes.

### 4. Performance evaluation

We implement GRiDA on a custom event-based simulator written in python and C languages. Node choices, LSA arrivals and traffic matrix changes are the possible events. Moreover, network statistics including link load, node configurations and power consumption are stored in a log. Unless otherwise specified, we simulate a time period...
of 1 week, by repeating the set of traffic matrices. In the following, we first analyze the transient behavior of the algorithm on the different scenarios. As a second step we consider the sensitivity of average performance metrics to parameter variations.

4.1. Transient analysis

We start by evaluating the performance of GRiDA on the ISP 1 scenario. We set $\delta = 1$ for testing the convergence of the algorithm. We then compare the power saving of GRiDA against the upper bound obtained solving the optimal problem of [21] for the off-peak traffic, and the centralized Least-Flow (LF) and Most-Power (MP) heuristics of [12,11], which are centralized heuristics that find the subset of links that must be powered off to carry the current traffic. In the heuristics, links are sorted by incremental carried traffic or by decreasing power consumption, respectively. The algorithms then consider one link at a time trying to see if it is possible to turn it off. In particular, the central control node: (i) disables the link, (ii) recomputes the traffic routing over the residual capacity, and finally, (iii) if network connectivity and maximum link utilization constraints are met, it turns off the link, otherwise it leaves the link in on status. A perfect knowledge of the traffic matrix is assumed to route traffic on the residual network and check connectivity constraints.

![Fig. 3. A network topology from a telecom operator: ISP 2.](image_url)

**Table 2**

Simulation parameters in the three simulation scenarios.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>ISP 1</th>
<th>Geant</th>
<th>ISP 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta_{LFA}$ (s)</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>$\Delta_{TM}$ (min)</td>
<td>30</td>
<td>30</td>
<td>48</td>
</tr>
<tr>
<td>$\Delta_{Max}$ (s)</td>
<td>25</td>
<td>25</td>
<td>50</td>
</tr>
<tr>
<td>$N$</td>
<td>22</td>
<td>23</td>
<td>112 + 261</td>
</tr>
<tr>
<td>$\delta$</td>
<td>1.0</td>
<td>0.999</td>
<td>0.9</td>
</tr>
<tr>
<td>$\beta$</td>
<td>50</td>
<td>50</td>
<td>1</td>
</tr>
<tr>
<td>$\phi$</td>
<td>0.7</td>
<td>0.7</td>
<td>0.5</td>
</tr>
<tr>
<td>Choices/node/traffic matrix</td>
<td>5.5</td>
<td>5.2</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Fig. 4a reports the power-saving versus time of GRiDA, LF, MP and the upper bound. It reports the power saving computed as the percentage of saved power with respect to a configuration in which all links are powered on. Since the LF and MP heuristics are centralized and require the knowledge of the traffic matrix, we run them at every traffic matrix change. After an initial transient, the power saving of GRiDA is constant: this is due to the fact that $\delta = 1$ and the network is largely over-provisioned; thus the algorithm converges to a solution that does not involve any increment in the penalty function. Interestingly, GRiDA outperforms both the LF and MP heuristics, saving 52% of power after convergence.

Please cite this article in press as: A.P. Bianzino et al., GRiDA: GReen Distributed Algorithm for energy-efficient IP backbone networks, Comput. Netw. (2012), http://dx.doi.org/10.1016/j.comnet.2012.06.011
To give more insight, Fig. 4b reports the cumulative number of link reconstructions due to network violations, for different values of $\beta$. For $\beta > 1$ reconstructions occur only during the initial transient. To this extent, low values of $\beta$ result in a large number of reconstructions, since the learning rate of the algorithm is lower. The intuition suggests that in this case the predominant term in the utility function is the power consumption, thus each node always selects the most aggressive configuration in term of power savings, resulting in a large number of violations. On the contrary, when $\beta > 1$ the number of reconstructions steadily decreases. Thus, a trade off emerges among responsiveness of the algorithm and number of reconstructions.

We now evaluate the performance of GRiDA under anomalous network conditions. In particular, a node failure is simulated after the convergence of GRiDA. Fig. 5 reports the power saving before and after the failure event. GRiDA is able to wisely adapt to a new configuration with only nine reconstructions due to network violations. In fact, as soon as the failure is detected, GRiDA starts turning on links given LSAs reporting network anomalies. Then, the algorithm starts again to switch off links until a stable configuration is reached. While GRiDA has not been designed to explicitly handle failures, it helps the failure management algorithm to recover from critical conditions.

We consider now the Geant topology with parameter set reported in Table 2. Fig. 6a reports the power saving versus time. Also in this case GRiDA outperforms both the LF and MP heuristics. Notice that here we set $\delta = 0.999$, thus GRiDA does not converge to a stable solution, since the penalty costs are decreasing with time.\(^4\) This allows GRiDA to adapt the power saving to the actual traffic. Fig. 6b reports instead the cumulative number of reconstructions. Also in this case, the number of unaccepted changes decreases as $\beta$ increases.

We consider now the ISP 2 topology. In this case, we have taken as reference the optimal solution of [17] solved for each TM,\(^5\) and the MP-MP and LF-LF heuristics, which has been proven in [11,12] to be the most effective ones for this topology. In particular, both MP-MP and LF-LF try to switch off first all the links incident to a node (which are sorted according to a Most Power or Least Flow criterion, respectively). Then, as a second step, the remaining links are eventually powered off individually (according to a Most Power or Least Flow ordering). We refer the reader to [11,12] for a detailed description of these algorithms.

Fig. 7a reports the algorithm comparison in terms of power saving. Interestingly, saving follows a strong day-night trend for all algorithms. In particular, more power saving is possible when the network is lightly loaded, i.e., during night. In this case, GRiDA is able to save an amount of power comparable to centralized heuristics, but without requiring the knowledge of the current traffic matrix. Moreover, the variability of the traffic impose GRiDA to quickly adapt the configurations. To give more insight, Fig. 7b reports the average link load and maximum link load in the network running GRiDA. Average link loads are computed for each traffic matrix. Interestingly, during night-time the maximum link load is below 30%, i.e., far from the load threshold $\phi = 0.5$. This suggests that the connectivity constraint is stricter than the maximum load constraint. During high traffic periods, some link loads actually gets close to 0.5. Indeed, some violations are present, even if of short duration and of small intensity. We will quantify violations better in the next section. Moreover, the average link load is always lower than 10%, suggesting that most links are lightly loaded even when GRiDA is run over the network.

\(^4\) Results presented in [1] show that a stable solution is reached also in this scenario setting $\delta = 1$.

\(^5\) The solution has been obtained running CPLEX on a high performance cluster hosted in our Campus [22].
Finally, Fig. 7c reports the average number of OFF–ON and ON–OFF link choices per node, per $\Delta t_D$ interval. Note that here we are accounting also the link reconfigurations triggered by a negative LSA. The figure reports also the average node degree $L_N$. Interestingly, GRiDA tries to turn off on average less than one link per node every $\Delta t_D$. On the contrary, during the morning GRiDA quickly reacts to traffic increase, and about two links per node are powered onto to increase the network capacity.

4.2. Average performance

We now investigate how the parameter settings impact the performance of the algorithm. In this case, we consider only the ISP 2 scenario since it is the largest one in terms of nodes and links. We evaluate the performance considering the following metrics: energy saving, number of unaccepted choices, and network overload. The intuition is to have a set of metrics to quantify the gains from saving energy.
while monitoring QoS for users. In particular, energy saving is evaluated as the integral of link power saving over a 1 week long time interval. The unaccepted choices metric is the ratio between the number of switch-off choices which are undone due to the immediate critical state indication by LSA, and the total switch off choices.\(^6\) Percentage is computed with respect to the total number of switch off attempts. The network overload is defined as the fraction of traffic exceeding the load threshold \(\phi\) with respect to the total carried traffic, i.e.:

\[
\zeta = \frac{\int \sum_{s \in F} \max(\rho^l(t) - \phi, 0) dt}{\int \sum_{s \in F} \rho^l(t) dt}
\]

where \(\rho^l(t)\) is the traffic request from node \(s\) to node \(d\) at time instant \(t\).

This is a relative indicator for the network congestion level, averaged over the simulation period, accounting for the number of load violations, their entity, and their duration.

### 4.2.1. Learning update

We first evaluate the impact of \(\delta\). Intuitively, this multiplicative parameter affects how much the past choices impact the current decision, i.e., if \(\delta = 0\), the penalty function is reset to 0 for the current state, every time that a positive LSA is received, while if \(\delta = 1\), penalties obtained by learning are kept forever. Fig. 8a reports the average link power saving for \(\delta \in [0, 1]\) and different values of \(\beta\). Interestingly, with \(\beta = 0.1\) the saving rapidly increases for increasing \(\delta\).

In particular, saving is 0 if \(\delta \leq 0.5\). This is due to the fact that the penalty \(\beta\) is not strong enough to choose a candidate configuration different from the all-off one, which is then undone since the connectivity check fails. Let us explain better this behavior with an example, supposing that a generic node \(n\) is running GRiDA. During the first choice of \(n\), the “all-off” configuration is selected, since it is the most convenient in terms of energy, i.e. \(c(K) = 0\). The penalty function is zero (we neglect the impact of \(\beta\)). This causes the connectivity check to fail and consequently the penalty function is updated to \(\beta\). During the following choice of \(n\), the minimum utility function is recomputed. If the all-off configuration is still the most convenient one, the connectivity check fails again and its associated penalty becomes \(\beta = \beta \delta\). After \(Z\) iterations with failed connectivity check the utility function for the all-off configuration becomes:

\[
U(K, S) = p(K, S) - \sum_{z=0}^{Z-1} \beta \delta^z = \beta \frac{1 - \delta^Z}{1 - \delta}
\]

This happens until the best current utility is lower than the utility function with at least one link on, i.e., \(U(K, S) < U(K_{\min}, S)\), with \(U(K_{\min}, S) = p(K_{\min}, S) + c_{\min} = c_{\min} = \min_{c \in \mathcal{C}} \{ c(j) \sum_{j=1}^{n} j \geq 1 \}\). Thus, only if \(p(K, S) > c_{\min}\) a different configuration is tested. For \(Z \rightarrow \infty\), the algorithm selects another configuration different from the all-off one iff \(\frac{1}{1 - \delta} > c_{\min}\). In our case, for \(\beta = 0.1, \delta > 0.5\) is necessary.

Fig. 8a reports the curve for \(\beta = 1\). Saving is averaged over a 1 week interval. The maximum error for saving is 3% with 95% of confidence. In this case, the initial penalty...
\(\beta\) is strong enough to let succeed the connectivity check, and savings between 25\% and 40\% are achieved. However, savings depend on \(\delta\) also in this case.

Fig. 8a and Fig. 8c report the network overload and the percentage of unaccepted choices, respectively. Interestingly, both metrics are minimized for intermediate values of \(\delta\), i.e., when the algorithm trades between full knowledge of past learning (\(\delta = 1.0\)) and power consumption (\(\delta = 0.0\)). Moreover, network overload is always extremely small, i.e., typically smaller than 10\%/\(C_0\) with \(\beta = 1\), suggesting that GRiDA is very effective in limiting the amount of traffic rerouted over congested links, with more than 60\% of choices that are accepted over 1 week.

To give more insight, Fig. 9a and b reports the percentage of off links and power saving over time, respectively. \(\beta\) is set to 1 in this case. Interestingly, with \(\delta = 0.0\) less than 10\% of links are switched off on average, but power saving is typically higher than 30\%, meaning that the links switched off are very expensive in terms of energy. Moreover, the savings tend to be constant, meaning that the algorithm always chooses the same configurations that minimize \(U(K, S)\). On the contrary, for \(\delta = 1.0\) the algorithm follows a typical day-night trend, with almost 50\% of links powered off during night and less than 10\% during peak hour. Note that power saving during the night is 40\%, suggesting that the additional links that are powered off...
with respect to $\delta = 0.0$ are short and hence consume a small amount of power. The intuition suggests that, as $\delta$ is increasing, the link status change more frequently, resulting in a larger states exploration operated by GRiDA.

Fig. 9c reports the percentage of the explored states over the possible ones for each node running GRiDA. As expected, for $\delta = 1.0$ the percentage of exploration tops 90%. This value is reached by backbone nodes that are connected by few links whose states change quite frequently. On the contrary, when $\delta = 0.0$, the percentage of exploration is below 30%, confirming that the network reaches a stable configuration which does not involve frequent changes of the node states.

### 4.2.2. Penalty update

We now evaluate the impact of $\beta$. In particular, we keep $\delta = 0.9$. Table 3 reports the average performance metrics.

Interestingly, the best results are obtained with lower values of $\beta$, suggesting that larger $\beta$ tend to penalize both power savings and overload, since frequent reconfigurations occur.

Finally, the table reports also the optimal power saving and the MP-MP/LF-LF heuristics, showing that GRiDA saves a comparable amount of power without requiring the knowledge of the actual traffic matrix.

### 4.2.3. Choice interval

We look at the sensitivity of GRiDA to the time intervals at which choices about links are made $A_c$. Fig. 10 reports the performance metrics for increasing values of $A_{c, \text{Max}}$. $\beta = (1, 100)$ and $\delta = (0.4, 0.6, 0.9)$. Interestingly, large $A_{c, \text{Max}}$ will slow down the algorithm convergence, while small values of $A_{c, \text{Max}}$ may cause unnecessary changes to the network topology that have to be quickly undone. This intuition is confirmed by Fig. 10c, in which the percentage of unaccepted changes rapidly decreases as $A_{c, \text{Max}}$ increases, for all the cases. However, this is not beneficial for the network since the network overload steadily increases while the saving decreases, since the system becomes slower in reacting to the changes of traffic. For example, with $A_{c, \text{Max}} = 1000$, $\beta = 1$ and $\delta = 0.4$ the average percentage of unaccepted choices is below 15%, but the network overload is two orders of magnitude higher than the $A_{c, \text{Max}} = 50$ s case. To give more insight, Fig. 11a and b report the evolution over time for $\beta = 1$ and $\delta = 0.9$ in terms of percentage of off links and maximum link load, respectively. As expected, with lar-

![Image](image_url)
ger values of $\Delta_c, \text{Max}$, the algorithm hardly follows the day-night trend, being able to switch off less than 10% of links. In addition, maximum link utilization steadily increases during the day, as the algorithm is less prompt to react to traffic surges. On the contrary, with a choice setting $\Delta_c, \text{Max} = 50 \text{ s}$, the performance of the algorithm is better, being it able to track the day-night trend. Energy saving is 35% and 25% for $\Delta_c, \text{Max} = 20 \text{ s}$ and $\Delta_c, \text{Max} = 1000 \text{ s}$, respectively, confirming that the algorithm performance degrades for large values of $\Delta_c, \text{Max}$.

4.2.4. LSA interval

Finally, we vary the LSA interval $\Delta_{\text{LSA}}$, considering also the case when $\Delta_{\text{LSA}}$ is greater than $\Delta_c, \text{Max}$, i.e., more than one node wakes up in the same $\Delta_{\text{LSA}}$ interval. Intuitively, a low LSA rate may deteriorate the algorithm performance since in this scenario node choices are based on outdate network states and traffic changes can cause overload situations to which the system does not promptly react. Table 4 reports the variations of the performance indicators with $\Delta_{\text{LSA}} \in [5 \text{ s}, 30 \text{ s}]$, as commonly adopted by OSPF. Results are obtained setting $\beta = 1$, $\delta = 0.9$, and $\Delta_c, \text{Max} = 50 \text{ s}$. Interestingly, all metrics present just minor oscillations with respect to $\Delta_{\text{LSA}}$, suggesting that the algorithm is robust even for large values of the parameter.

4.3. Implementation issues

In this work, we suppose network devices to support a power saving state for links, which can be selected by any of its two adjacent nodes, by means of a simple signaling protocol. We suppose undirected links, i.e., the power state of the link has to be the same for both directions. The extension to support unidirectional power states is straightforward.

A link switch off procedure can occur without traffic losses. In practice, every time a node decides to switch off a link, a proper signaling mechanism is in place to allow it to signal the choice to all nodes in the network (e.g., by increasing the link routing cost), so that routing tables of other nodes can be properly reconfigured before the link is actually switched off. In practice, this additional delay allows routing tables to be recomputed and to converge. This delay must be smaller that $\Delta_{\text{LSA}}$, i.e., smaller than 5 s.

Our solution requires nodes to run a link-state routing algorithm, through which eventual link overload occurrences are signaled to all the nodes in the network. This allows nodes to timely signal and quickly react to eventual network congestions. Opaque LSAs [23] may allow to easily

---

Table 4

<table>
<thead>
<tr>
<th>$\Delta_{\text{LSA}}$</th>
<th>Saving (%)</th>
<th>Unacc. choices (%)</th>
<th>$\zeta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>29.18</td>
<td>33</td>
<td>9.18e−04</td>
</tr>
<tr>
<td>10</td>
<td>29.40</td>
<td>33</td>
<td>9.00e−04</td>
</tr>
<tr>
<td>20</td>
<td>28.03</td>
<td>28</td>
<td>9.49e−04</td>
</tr>
<tr>
<td>30</td>
<td>28.43</td>
<td>25</td>
<td>1.07e−03</td>
</tr>
</tbody>
</table>

Fig. 11. ISP 2: impact of $\Delta_c, \text{Max}$: (a) links off and (b) maximum link load.

Please cite this article in press as: A.P. Bianzino et al., GRiDA: GReen Distributed Algorithm for energy-efficient IP backbone networks, Comput. Netw. (2012), http://dx.doi.org/10.1016/j.comnet.2012.06.011
carry the additional information in practical implementations, without any change to the link-state protocol. It should be noted that node disconnections from the rest of the network are prevented by GRiDA, through the connectivity check mechanism, which is run before trying to power off a link.

Finally, the number of additional LSAs generated by the algorithm is small. For example, consider the ISP 2 scenario, which is the largest one. We can upper-bound the algorithm overhead as follows: when standard OSPF is run, and GRiDA is not enabled, a LSA is generated by each node every \( A_{LSA} \) interval. Considering \( N = 112 \) transit nodes and \( A_{LSA} = 5 \) s, each node has to process a number of LSAs equal to \( 60 \times 112/5 = 1344 \) per minute. Consider now the same scenario, with GRiDA enabled. Additional LSAs are generated at every non-null node choice, and at every network overload occurrence. Fig. 7c reports the number of reconfigurations, corresponding to both cases. The sum of events never overcomes three reconfigurations per node per \( A_{SM} \). As such, the number of additional LSAs is equal to: \( 3 \times 112/48 = 7 \) for each minute. The number of LSAs is hence increased by 0.52% only. Thus, the additional overhead introduced by GRiDA is negligible.

5. Conclusions

We have presented GRiDA, a distributed online algorithm to reduce power consumption in backbone networks. Our solution is based on a reinforcement learning technique that requires only the exchange of periodic Link-State Advertisements in the network. Results, obtained on realistic case studies, show that GRiDA achieves performance comparable to different existing centralized algorithms, saving on average 40–50% of links energy, without significantly affecting the QoS. Moreover, GRiDA is able to react to faults, as well as to sudden changes in the traffic requests.

Differently from other approaches, GRiDA does neither require a centralized controller node, nor the knowledge of the current traffic matrix.

Interesting directions that can be studied are how to extend the algorithm to consider the possibility of turning off nodes rather than single links only, and a more detailed analysis of the impact on QoS metrics, such as flow throughput or packet loss.

Acknowledgment

This work was supported by the European FP7 ECONET project under Grant Agreement No. 258454.

References

Luca Chiaraviglio (M’09) received the PhD in Electronics and Communications Engineering from the Politecnico di Torino, Torino, Italy in February 2011. Between August 2009 and April 2010, he was a visiting scholar at the WING group of Boston University, under the supervision of Prof. Ibrahim Matta. Currently, he is a postdoc researcher at Politecnico di Torino. His main research interests are in the field of energy-aware design for backbone and access networks.

Marco Mellia (SM’08) received his Ph.D. degree in Telecommunications Engineering in 2001 from Politecnico di Torino. In 1999, he was with the CS Department at Carnegie Mellon University, Pittsburgh (PA) and since April 2001 he is with EE Department of Politecnico di Torino. He has co-authored over 150 papers published in international journals and conferences, and he participated in the program committees of several conferences including IEEE Infocom and ACM Sigcomm. His research interests are in the fields of traffic measurement, P2P applications and energy-aware network design.

Jean-Louis Rougier received the Engineering diploma and the Ph.D. degree in Networks and Computer Science from Télécom ParisTech (formerly called École Nationale Supérieure des Télécommunications), Paris, France, in 1996 and 1999, respectively. He joined the Computer Science and Networks Department of Télécom ParisTech in 2000 as an Associate Professor. He has taught and worked in research projects on routing and traffic engineering in networks, in various context (from Optical, Internet to Wireless Networks). His research interests are Future Internet Architecture, Traffic Engineering, Routing protocols, Green Networking, Wireless Mesh Networks and Performance Evaluation and Algorithms.