OSPF optimization via dynamic network management for green IP networks

Antonio Capone, Carmelo Cascone
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano
Milano, Italy
capone@elet.polimi.it,
carmelo.cascone@mail.polimi.it

Luca G. Gianoli, Brunilde Sansò
Département de génie électrique
École Polytechnique de Montréal
Montréal, Canada
luca-giovanni.gianoli@polymtl.ca
brunilde.sanso@polymtl.ca

***Abstract*—In this paper we present a novel experimental platform for network management able to dynamically optimize the energy consumption of backbone IP networks operating with OSPF. The idea is to efficiently adjust link weights to put to sleep idle devices. The framework relies on multiple pre-computed sets of link weights that are applied to the considered network domain according to real time measurements on the link utilization. The management module exploits the Simple Network Management Protocol (SNMP) to collect the load measurements and modify the link weights. The pre-computed link weights are calculated by running a state-of-the-art algorithm for off-line energy-aware traffic engineering based on predicted traffic matrices. The modules of the platforms have been implemented for Linux based environments and tested using emulated networks of virtual machines. Experimental results showed a significant reduction in terms of network resources required to route traffic demands, demonstrating how, in average, 20% of nodes and 40% of links, can be put to sleep without compromising network performance and stability.**

I. Introduction

In 2007, it was estimated that the comprehensive power consumption of the ICT sector was close to $156\,GW$, with the solely network equipments, excluding servers in data centers, responsible for $22\,GW$ [1]. As for the the single Internet Service Providers (ISP), data reported in [2] shows that in 2009 the power requirements of the largest ISPs overcame, in some cases, $10\,TWh$ per year. Furthermore, in case of medium size ISPs such as Telecom Italia and GRNET, the yearly energy consumption of access and core networks is estimated to exceed $400\,GWh$ in 2015.

Thus, the study of new methods and strategies to optimize IP network energy consumptions have been receiving an increasing attention by the scientific community, device manufacturers and ISPs [3], [4].

A natural strategy to reduce the network energy consumption consists of adjusting network routing so that, depending on the traffic level, redundant routers and links can be put to sleep [5]. In fact, network devices in sleeping state consume around 90% less than idle active ones [6]. Now, since network utilization is very often below 50% [7] due to ISP's over-provisioning, a large subset of network devices may be potentially put in stand-by even during peak traffic periods. The advantages of the sleeping approach with respect to those based exclusively on load balancing are discussed in [8].

Sleep-based energy-aware network management can be centralized, with a network management platform that adjusts the network configuration according to global network measurements (see for instance [9], [10]), or distributed, with multiple agents that take independent decisions based on local or global data (see for instance [11], [12]).

We focus on IP networks operated with the Open Shortest Path First (OSPF) protocol with Equal Cost Multi-Path (ECMP). According to OSPF, traffic demands are routed through the shortest paths defined by the link weights. The network routing can thus be optimized (traffic engineering) by adjusting the link weights themselves [9]. The shortest path scheme prevents energy management strategies to optimize routing on a per-flow basis (like with MPLS), operation that can be practically very complex when an large number of traffic demands is considered.

In this paper we propose a novel approach to energy efficient operation of IP networks based on a new Network Management platform able to adjust dynamically OSPF weights according to traffic measurements collected in real-time from controlled routers. Depending on traffic patterns and link weights, a subset of routers or their interface cards can be completely unloaded so as to be put to sleep by a distributed mechanism that can also reactivate them when traffic and weights change. The platform selects the OSPF weights among a set of precomputed scenarios that are optimized off-line using the approach presented by some of the authors in [9]. The selection of the scenarios is performed dynamically according to the traffic statistics collected by the management platform, by using some thresholds on load levels on different sets of links. The SNMP-based on-line management engine of the platform is used to both gather traffic statistics and apply weights.

The proposed platform together with the optimization modules, traffic analyser, and weighs handler have been implemented into an experimental framework for Linux-based devices. The proposed approach for energy efficient IP network has been tested on a set of network topologies which have been implemented into an emulated

978-3-901882-56-2

environment using virtual machines. Since the practical implementation of the sleep mode into router devices is out of the scope of this paper, we assume that network devices and their line cards can automatically go to sleep through centralized or autonomous mechanisms able to detect the absence of traffic (see e.g. [13]).

The reminder of the paper is organized as follows. In Section II the literature is briefly reviewed. The idea of the on-line OSPF energy-aware optimization is thoroughly presented in Section III. The detailed architecture of the network management platform is described in Section IV, whereas computational results are discussed in Section V. Conclusions follow in Section VI.

II. Related work

Due to the growing attention towards green networking, and more specifically, energy efficient IP networks, several work on energy-aware network management have appeared in recent years [4].

The approaches can be categorized according to the routing scheme considered, the number of agents involved in the optimization (i.e. centralized or distributed) and the frequency and moment in which the optimization is performed (i.e. one time off-line, several times on-line).

Methods for on-line network management are presented in [10]–[12], [14]–[17]. The distributed algorithms proposed in [11], [12] to put to sleep network links can work with IP networks operated with OSPF. The OSPF link state packets are then exploited to provide all routers with the data concerning the load on each link. According to a given policy, the complete set of routers [11], or each single router independently [12], periodically selects a potential link to be put to sleep; if the switching-off leads toward the violation of the max-utilization allowed, the link itself is immediately reactivated. Differently from our approach, no traffic matrices are considered and network configuration is changed very frequently, which may lead to network instability; moreover, network nodes cannot be put to sleep, network routing is not optimized (link weights are kept unchanged), and no congestion guarantees are given when a link is put to sleep.

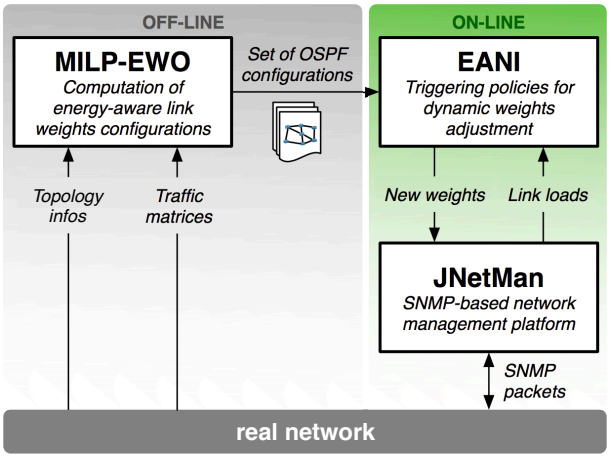


Figure 1. Flow chart of the network management platform.

Another method that exploits link state packets to disseminate link state information is proposed in [15]. A centralized network management platform is used to efficiently configure the OSPF link weights and the equal cost path splitting ratios. Differently from our work, the method has not been implemented in a real network environment but has been only tested by running ns2 simulations. Furthermore, the procedure requires to change the OSPF splitting ratio, that is typically kept fixed by ISPs, and no stability guarantees are provided. Some distributed algorithms for energy-aware traffic engineering in networks operated with flow-based routing protocols are proposed in [14], [17]. In [14] each edge router of the considered domain periodically optimizes the amount of traffic transmitted along each pre-configured path, while in [17], in addition to the routing optimization, energy-aware admission control is performed. W.r.t. [14], [17], we consider IP networks operated with OSPF, keep all the network protocols unchanged, and exploit only network technologies already implemented in a typical IP network.

The centralized approach proposed in [10] considers networks operated with a hybrid OSPF+MPLS routing protocol and exploits a restricted path MILP formulation to put to sleep network links. Neither in this case network nodes can be put to sleep and no mechanisms are used to prevent frequent changes. Finally, in [16], the authors propose a MILP-based on-line heuristic to put to sleep line cards and chassis while respecting a limitation on the number of switching-on allowed to each line card along an entire day. The method considers networks operated with MPLS and requires the knowledge of the instant traffic matrices.

As for the off-line approaches found in the literature, [9], [18] present a mixed integer linear programming algorithm for energy-aware weight optimization (MILP-EWO). This approach is incorporated as an input in our on-line method. Other off-line approaches include a MILP formulation and some simple greedy heuristics to put to sleep network devices [19], exact and heuristic algorithms to minimize the daily energy consumption in MPLS networks [20], an upgraded version of OSPF that minimizes the number of active links by using a restricted set of shortest path trees [21], methods for energy-aware traffic engineering in Carrier Grade Ethernet networks [22] and heuristics based on local search and Lagrangian relaxation to maximize energy savings in OSPF networks [23].

To the best of our knowledge no previous work combines off-line and on-line techniques to optimize OSPF weights for an energy-aware environment.

III. Dynamic energy-aware OSPF optimization

The energy-aware optimization of the link weights represents a promising strategy to jointly minimize both energy consumption and network congestion in IP networks operated with OSPF (see [9], [18]). A building block of our management platform is the MILP-EWO algorithm based on the idea presented in the aforementioned papers. That idea consists on selecting very high OSPF weights for the links that the planner would like to exclude from

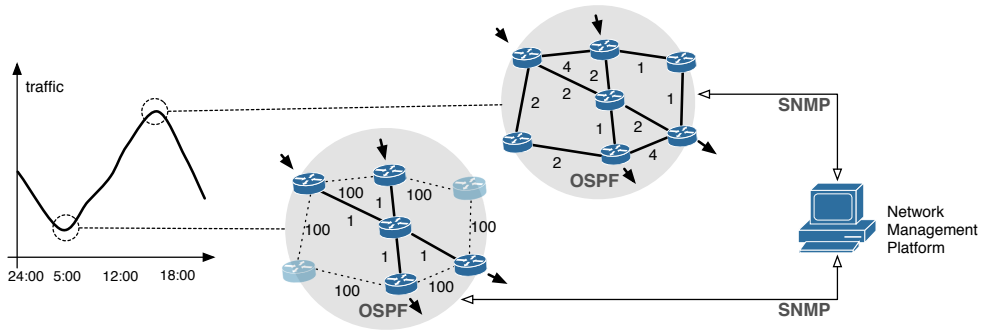


Figure 2. The Network Management platform dynamically adapts the network topology to the current traffic conditions, to switch off under-utilized elements by exploiting OSPF link weights. SNMP is used to monitor traffic levels and to apply link weights.

operation and, eventually, put to sleep, while optimizing the remaining weights to achieve optimal load balancing.

The network management platform we propose, however, uses this building block in a completely new way and in a context much more dynamic. In the platform, MILP-EWO (see Figure I) is integrated into a real-time green-framework, where both off-line and on-line optimization methods are efficiently combined to guarantee both stability and responsiveness to traffic and network changes.

We select a set of traffic patterns that represent typical working conditions of the network and then compute for each of them an OSPF link weights configuration (for convenience *OSPF configuration* or simply OC) using off-line the optimization algorithm. The number of traffic patterns to be taken into account greatly depends on the specific characteristics of the network considered. In [24], it is shown that a few network configurations are sufficient to obtain quasi-optimal energy savings, but the approach we propose here is general and can also be applied when the number of traffic scenarios is large.

The platform allows to apply the OCs to the network dynamically according to real-time measurements on link loads. In order to do that the platform has been developed using some modules that allow to collect traffic measurements, select the most appropriate OC and apply it using SNMP instruments.

In Figure I, the interconnections of the three main modules of our platform are indicated: (i) the *Mixed Integer Linear Programming based algorithm for Energy-aware Weights Optimization* (MILP-EWO), (ii) the *Energy-Aware Network Intelligence* (EANI), and the *Java-based Network Management platform* (JNetMan) [25].

MILP-EWO is used in an off-line phase to compute the restricted set of *OSPF configurations* that, afterwards, will be efficiently applied on-line by EANI and JNetMan. Note that each *OSPF configuration* is typically obtained by considering an estimated traffic matrix corresponding to a particular traffic level. The choice of the traffic matrices given as input to MILP-EWO is crucial for the effectiveness of the entire approach and should be made according to the traffic profile and the network domain considered. It is worth pointing out that traffic matrices can be estimated with good accuracy by network providers [26] by exploiting both direct [27] and indirect measurements [28].

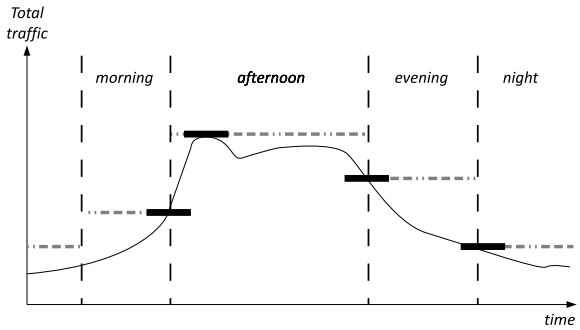


Figure 3. Example of daily profile of the totale traffic entering an ISP network. Each bold black line represents an instant chosen to sample a traffic matrix.

As shown in Figure 3, due to the daily regular behavior of Internet traffic, which is typically characterized by very slow dynamics [2], [26], traffic matrices can be chosen by ideally splitting a single day into different macro intervals, and subsequently sampling a set of matrices that allow to approximate with reasonable accuracy the traffic load of each period.

In the on-line phase, the *OSPF configurations* are efficiently managed in real-time by EANI, which is responsible for adapting the network configuration to the traffic conditions. Real time traffic measurements are provided to EANI by the SNMP-based management platform JNetMan, which is exploited to practically modify the OSPF weights too. Figure 2 illustrates the general idea.

The detailed architecture of the EANI is shown in Figure 4. Four main operational blocks can be identified: a *network monitor*, an *OC monitor*, an *OCs database*, and an *OC enabler*. The *network monitor* periodically requests JNetMan to retrieve the current utilization level of each link. A *bandwidth utilization report* is then produced and sent by the *network monitor* to the *OC monitor*. The latter, by consulting these reports, is responsible for selecting the proper OSPF configuration from the *OCs database* according to the configured "switching" policy; the term "switching" here refers to moving from one OSPF configuration to another. The switching policy determines (i) under which conditions to update the current OSPF configuration, and, in case of configuration switching, ii)

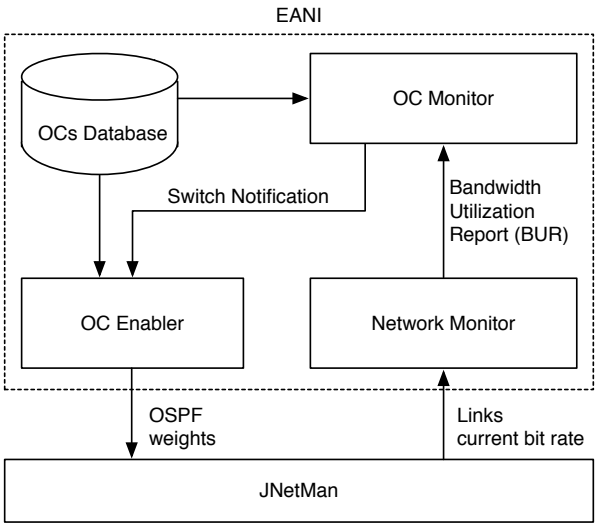


Figure 4. Architecture of EANI.

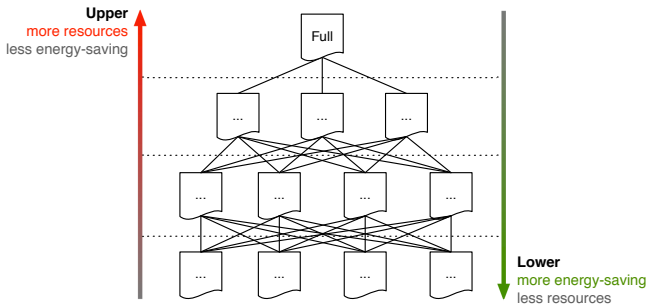


Figure 5. Example of an OCs graph: OCs are sorted according to the amount of network elements active. "Full" represent the complete topology, with all elements active. Movements to different upper/lower OCs are made according to traffic variations.

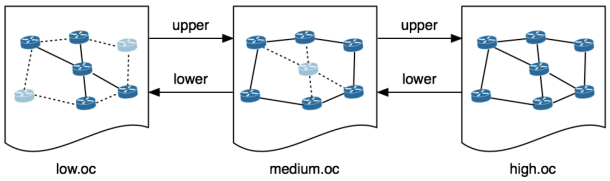


Figure 6. Example of *OCs chain*, a particular case of an OCs graph. In this case the most powerful `high.oc` represents the full topology, while `low.oc` embodies the most energy-saving topology.

which alternative configuration to select. The main idea is to switch configurations when the network is observed to be either underutilized or too congested.

In our implementation we defined a switching policy based on four specific thresholds ψ^+_{av}, ψ^-_{av}, ψ^+_{max}, ψ^-_{max}, representing, respectively, the network maximum average utilization for consumption increase or decrease, and the maximum average utilization for consumption increase or decrease. The thresholds should be properly selected to prevent that a network that is in normal conditions continuously oscillates between different *OSPF configurations*. See Section III-A for further details on the definition of the switching policy. In the *OCs database*, each *OSPF configuration* is defined as an ASCII file, uniquely described by (i) the set of OSPF *link weights* assigned to the links, (ii) the bandwidth utilization *thresholds* (ψ^+_{av}, ψ^-_{av}, ψ^+_{max}, ψ^-_{max}) to be respected, and (iii) the *pointers* to other OCs to be considered when utilization thresholds are exceeded (i.e. the switching policy).

Once a configuration switch is requested by the *OC monitor*, a *switch notification* containing the pointer to the new configuration to be applied is sent to the *OC enabler*. This latter retrieves the configuration details from the database and starts an OSPF link weight update procedure by using the JNetMan API . The notion of *pointer* will be further discussed in Section III-A.

In big networks with hundreds or more subnets this operation requires considerable computational resources, and convergence to a new routing topology may take tens of seconds. To reduce convergence time, the *OC enabler* ignores the links whose weight remains unchanged in the new configuration. Weight changes might also create temporary routing loops that could lead to packet loss. To prevent this phenomenon, the *OC enabler* updates link weights one by one at fixed intervals necessary to allow all the routing tables to be updated before the adjustment of a further weight.

A. OSPF switching policy

The definition of a proper switching policy is crucial for guaranteeing the correct application of the *OSPF configurations*. A reasonable assumption, considered here, is that traffic matrices with higher loads lead towards network topologies with a higher power consumption.

The general proposed framework of the OSPF switching policy is presented in Figure 5 where the configurations are portrayed in a hierarchical fashion with the top configuration being the one with all the elements on and the ones in the lower part of the graph having the lowest consumption. The arcs between configurations represent the *pointers* that link one configuration with another. Essentially, the pointers define how to "switch" from one to another configuration.

Let μ_{max} and μ_{av} be, respectively, the maximum and average network utilization. Since in typical IP backbone networks traffic matrices tend to vary while preserving a constant ratio between different traffic demands (they are related to the number of clients connected to a given node), we have defined a basic switching policy where the upper and lower OSPF configuration pointers are applied, respectively, when $\mu_{max} \geq \psi^+_{max} \vee \mu_{av} \geq \psi^+_{av}$ and $\mu_{max} \leq \psi^-_{max} \wedge \mu_{av} \leq \psi^-_{av}$.

However, to efficiently manage more general scenarios where traffic loads on different links can vary independently, a second type of switching policy has also been developed, according to which the so-called *link-specific pointers* are thus consulted when the thresholds are exceeded only for a specific subset of links.

This kind of relationships expressed in *pointers* OC field allows EANI to offer maximum flexibility and adapt-

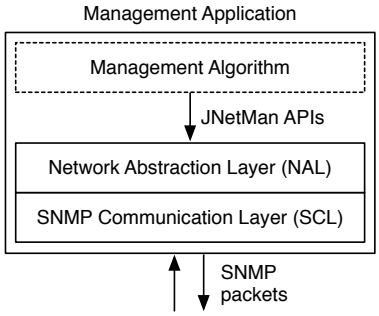


Figure 7. Architecture of a JNetMan based management application: NAL and SCL provide a set of high-level management API to the *management algorithm*.

ability to traffic variations scenarios. In the simplest case where only the *general upper/lower pointers* are defined for each OC, the general graph of Figure 5 is reduced to a simple *OCs chain* such as the one portrayed in Figure 6.

IV. Java Framework for SNMP-based Network Management Applications

Our solution uses the management features offered by the SNMP protocol. In out platform we developed JNetMan, a java framework to aid researchers in the development of complex network management applications. We recently made JNetMan an open-source project freely distributed under the Apache License 2.0 model, source code and documentation is available for download at [25].

The aim of JNetMan is to offer a clear separation between the specific management algorithm and the low-level SNMP protocol operations needed to execute the algorithm in the network. Most of the green networking work published to date usually refers to a set of high-level management operations needed to practically enable a specific solution in a given network. Developing a specific management solution with SNMP requires to handle a set of more low-level operations such as authentication, retrieving OIDs from MIB files, casting from application specific variable types to SMI defined datatypes, packets generations, requests timeout, etc. With JNetMan we introduce a software framework able to offer a set of powerful primitives to directly interact with network devices trough a high-level abstraction of the network and its related management operations. The result is a reusable software platform that can be used by researchers to easily develop specific solutions for the management of telecommunications networks, thus allowing fast experimental validation of new algorithms.

Fig. 7 illustrates the architecture of the JNetMan framework, which is composed of two main operational layers, namely *Network Abstraction Layer* (NAL), and *SNMP Communication Layer* (SCL). In this figure the *management algorithm* is intended as a place holder for the implementation of a specific management solution that makes use of JNetMan's API, as it could be EANI.

The NAL represents the high-level interface used to manage the network, providing a first logical abstraction on 2 levels: *network entity*, and *management agent*. Inside JNetMan a network is intended as a set of *network entities*, namely *node*, *interface*, and *link*. A *node* is a network node, it can be intended as an interconnection device or as an end device. Every node has one or more network *interfaces*, and every *link* is intended as a connection between two *interfaces*. In the classic SNMP architecture every operation is referred to the agent executed in the network device. We instead provide a more natural entity-based division of the management operations. JNetMan defines a *management agent* for every *network entity*, in this way we'll have a *node agent*, an *interface agent* and a *link agent*. Each of these agents define a set of specific management operations that concern the corresponding network entity. For example, obtaining the current average bit rate (bit/s) of an Ethernet link is a JNetMan management operation defined for the *link* entity, and it's operated by calling the corresponding primitive defined inside the *link agent*. By using a line of Java code of the JNetMan's API this will be:

```
/*
 * This method returns the average bitrate (bit/s) of the
 * link named L1, evaluated over an interval of 5 seconds.
 */
network.getLink("L1").getAgent().getCurrentBitrate(5);
```

While the NAL provides a set of high-level network management API to the management algorithm, the SCL represents the implementation of the low-level SNMP operations. This layer helps the NAL to complete the management tasks by providing: a set of helper methods to process values gathered from specific MIB-modules, and primitives to safely convert datatypes from SMI to Java and vice versa. Beside this, the basic task of the SCL is to establish a reliable SNMP communication channel between the management host and network devices. For the implementation of all these SNMP low-level operations we used an existing open-source solution called SNMP4J [29], an enterprise class free open source and state-of-the-art SNMP implementation for Java.

To correctly operate, JNetMan needs to know informations such as: the name and the IP address of nodes to manage; the interfaces installed for each of these nodes; links names and the corresponding connected interfaces, plus a number of other configuration parameters. All this informations as taken as input by JNetMan in the form of plain ASCII files, namely *properties files*, (usually no more than a few hundreds of kilobytes, highly compressible, that can be easily published over the Web or transferred by email). JNetMan reads properties files to internally build a reference model of the network topology and to set all the configuration parameters, thus making it very simple and straightforward to manage an existing network consisting of several nodes, interfaces and links.

V. Computational results

A. Test-bed

The proposed network management platform has been tested in an emulated network environment developed by using Netkit [30] [31], a freely available network emulation

Table I. NETWORK TOPOLOGIES FROM SNDLIB [35] USED FOR EXPERIMENTATIONS.

Network	Nodes	Links	Edge_{nodes}	Core_{nodes}
abilene	11	14	6	5
polska	12	18	6	6

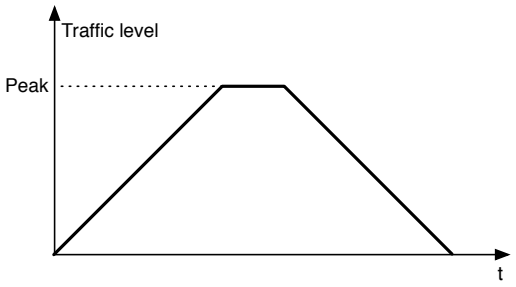


Figure 8. "Fade-in fade-out" profile used to generate traffic during tests.

environment based on User-Mode Linux, and a set of other networking tools, i.e. the OSPF routing daemon Quagga [32], the SNMP agent Net-Snmp [33], and the Distributed Internet Traffic Generator (D-ITG) [34].

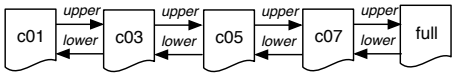
1) Network topologies and traffic matrices: We considered two real network topologies provided by the widely known SNDLib [35], namely `polska` and `abilene` networks (see Table I). Furthermore, in each topology, network nodes were randomly split between core nodes and edge nodes. Since the first ones, by definition, cannot represent neither traffic sources or traffic destinations, they are the only nodes that can be put to sleep. Note that we could not use larger topology due to Netkit resource constraints.

Traffic was generated by computing, for each network, a peak traffic matrix. We derived it by scaling the matrix provided by the SNDLib with a fixed parameter ϖ^{peak}. This latter was chosen as the largest one whose resulting matrix were routed, with fully splittable routing, without exceeding a 70% maximum utilization limit. To better evaluate the effectiveness of the proposed platform, we configured the traffic generators to simulate, for each demand, the simple, but also quite realistic "fade-in fade-out" profile shown in Figure 8, where the peak level is represented by the peak matrix values. Packets were generated according to a normal distribution for both IDT (Inter Departure Time) and PS (Packet Size). The IDT and PS average values of each single demand were derived from the corresponding traffic matrix to reproduce the desired average transmission rate. As for the variance values, we experimented with both 10%, 20% and 30% variance for both IDT and PS processes.

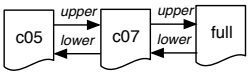
2) Parameter setting: A summary of the OSPF configurations obtained by executing MILP-EWO for both `polska` and `abilene` networks is shown in Table II, while Figure 9 illustrates the resulting *OCs chains*. More precisely, to compute the *OSPF configuration* c0x (e.g c01, or c03), we scaled the peak matrix with a second fixed parameter θ^{0x} so as to respect a $x0\%$ maximum utilization limit (c03 $\rightarrow$ 30%). We then ran MILP-EWO with the resulting traffic matrix while imposing a maximum utilization limit of 70%. Note that the *full* configuration was obtained by

Table II. DESCRIPTION OF THE *OSPF configurations* COMPUTED BY MILP-EWO.

Network	ψ_{max}	OC	Links_{active}	Nodes_{active}
polska	10%	c01	7 (39%)	8 (67%)
	30%	c03	9 (50%)	9 (75%)
	50%	c05	13 (72%)	11 (92%)
	70%	c07	14 (78%)	12 (100%)
	100%	full	18 (100%)	12 (100%)
abilene	50%	c05	9 (82%)	10 (71%)
	70%	c07	10 (91%)	12 (86%)
	100%	full	11 (100%)	14 (100%)



(a) **polska**



(b) **abilene**

Figure 9. *OCs chains* used in tests.

using θ^{07} and forcing MILP-EWO to keep activated all the network elements. With `abilene` we didn't consider OCs c01 and c03 because equivalent to c05.

During the tests we adjusted the utilization thresholds ψ in order to experiment with both a *restrictive* and a *permissive* switching policy. The considered ψ values are reported in Table III. Note that in the *permissive* policy ψ^{-}_{max} was increased up to 60% (`polska`) and 65% (`abilene`).

Finally, note that the duration of each simulation was of 30 minutes with `polska` and 20 minutes with `abilene`. Furthermore, JNetMan was configured to collect load measurements every 20 (`polska`) or 15 (`abilene`) seconds.

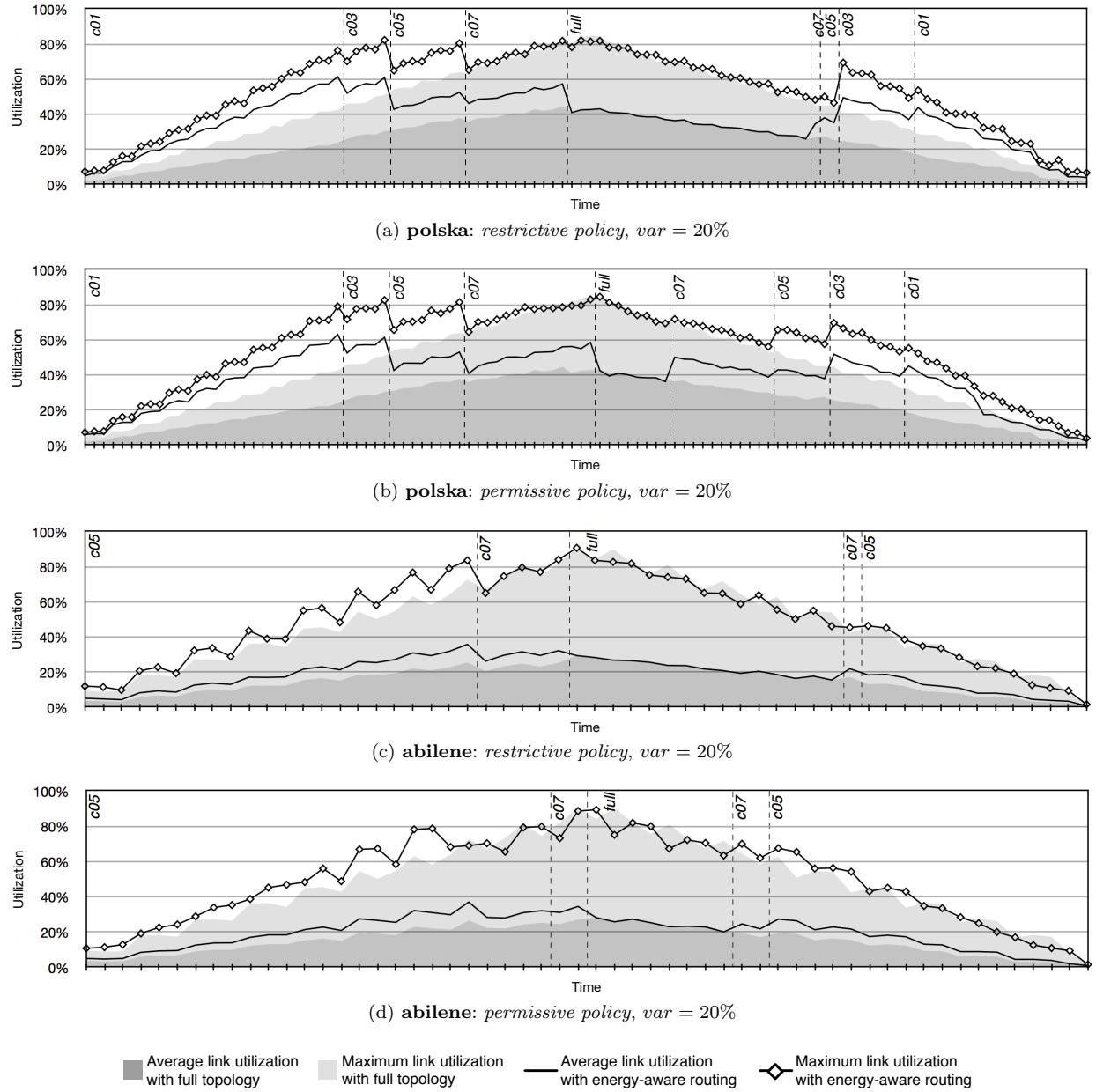
B. Experimentation

We ran several tests for each network, varying both switching policy and variance parameters of the packet generation processes. In Figure 10 we report the time distribution for each single OC observed with both *restrictive* and *permissive* switching policies. The values were computed by averaging over three instances characterized by different variance values (i.e. 10%, 20% and 30%). The *permissive* policy allowed to reduce by one third the use of the *full* configuration and to redistribute it between the remaining less consuming OCs.

In Figure 11 we show the average and maximum utilization levels observed by varying switching policy and fixing IDT and PS variance to 20%. Note that, for comparison purposes, we also report the values obtained by applying the *full* configuration along the entire simulations. The

Table III. SWITCHING POLICIES.

Network	Switching policy	ψ^{+}_{av}	ψ^{-}_{av}	ψ^{+}_{max}	ψ^{-}_{max}
polska	*restrictive*	60%	40%	80%	50%
	permissive	60%	40%	80%	60%
abilene	*restrictive*	60%	40%	80%	50%
	permissive	60%	40%	80%	65%



(a) **polska**: *restrictive policy*, $var = 20\%$

(b) **polska**: *permissive policy*, $var = 20\%$

(c) **abilene**: *restrictive policy*, $var = 20\%$

(d) **abilene**: *permissive policy*, $var = 20\%$

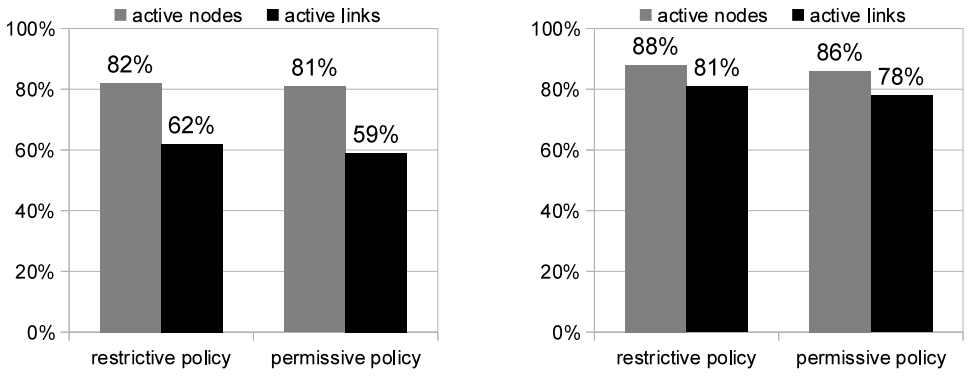
Figure 11. Link utilization charts obtained during 4 test instances: varying the switching policy has led to a general optimization of the *OCs chain*, switches to lower OCs are permitted more rapidly using a *permissive policy*.

first result to point out is that the configuration sequence dynamically applied by EANI correctly reflects the "fade-in fade-out" traffic profile: the defined OC chain is in fact progressively applied, from the lowest configuration to the full topology and vice versa. Furthermore, note that, despite the traffic variability induced by the IDT and PS variance, the threshold-based switching policies successfully prevent undesired oscillations between upper and lower configurations. The same behaviour was observed with variance incremented up to the 30% value.

As expected, since energy-aware OSPF optimization caused traffic to be routed through a subset of network elements, both higher maximum and average link utilization levels w.r.t. to the full topology case were observed. With `abilene` the utilization increase, in terms of absolute utilization, was up around 10%, while in `polska`, was up to 30% during low traffic periods.

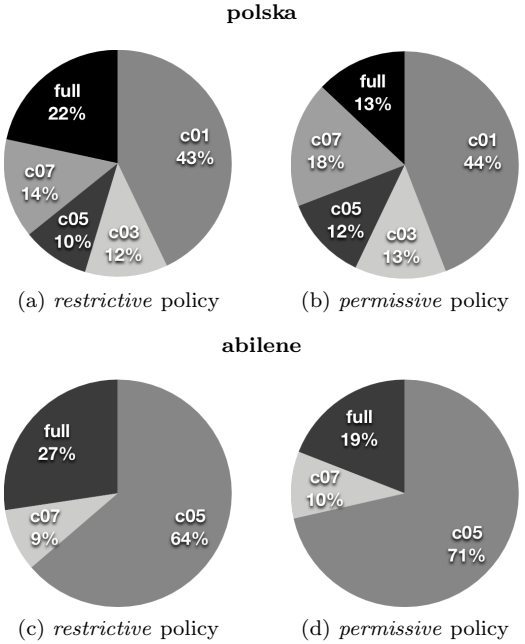
It is very interesting to notice how the adjustment of the utilization thresholds may positively influence the performance of the approach. Note, for instance, that the adjustments made to obtain the *permissive* policy allowed, in both networks, to avoid the undesired very quick switch from the full configuration towards c03 (`polska`) or c05 (`abilene`), where certain configurations were maintained for only a few seconds.

Finally, in Figure 12 we analyse the network resources/energy consumption. Since we experimented with an emulated environment with no real routers or links, instead of using power consumption values referred to



(a) polska (b) abilene

Figure 12. Relative network resources usage against an "always full topology" situation.



(a) *restrictive* policy (b) *permissive* policy

(c) *restrictive* policy (d) *permissive* policy

Figure 10. Distribution of OCs as percentage of time w.r.t the total duration of the simulations.

some possible real devices, we report results in terms of percentage of active routers and links. The percentages are weighted w.r.t. to the time used by each configuration. Results show that in `polska` we were able, in average, to put to sleep 20% of network nodes and 40% of network links. The use of the *permissive* policy allowed to gain only a further 3% for what concerns network links, and 1% as for network nodes. That means that, in case of greater attention for network performance w.r.t. network consumptions, it would be possible to adopt more *restrictive* policies while preserving the platform capability to significantly reduce the consumptions. As for `abilene`, due to the simpler topology structure, we observed a smaller consumption reduction, with, in average, 15% of nodes and 20% of links put to sleep during the simulations.

VI. Conclusion

We have proposed a novel experimental platform for network management able to dynamically optimize the energy consumption of backbone IP networks operating with OSPF. The new framework combines both off-line and on-line optimization to guarantee both network stability and responsiveness to real-time traffic conditions. We have shown how to dynamically apply a restricted subset of *OSPF configurations* (sets of link weights) in IP networks to minimize energy consumption while preserving the network capability to provide the requested QoS. We have also pointed out how an optimized policy based on several utilization thresholds allows to prevent the network to frequently oscillate between different configurations. In terms of network resources requirements, the new green platform demonstrates how, using real network topologies and quite realistic traffic scenarios, in average, 20% of nodes and 40% of links can be put to sleep. Since the tests were conducted in a linux-based emulated network environment, we leave as future work the execution of experiments in real networks.

References

[1] W. Vereecken, L. Deboosere, D. Colle, B. Vermeulen, M. Pickavet, B. Dhoedt, and P. Demeester. Energy efficiency in telecommunication networks. pages 44–51, 2008.

[2] R. Bolla, R. Bruschi, A. Carrega, F. Davoli, D. Suino, C. Vassilakis, and A. Zafeiropoulos. Cutting the energy bills of internet service providers and telecoms through power management: An impact analysis. *Computer Networks*, 56(10):2320–2342, 2012.

[3] GreenTouch consortium. http://www.greentouch.org.

[4] R. Bolla, R. Bruschi, F. Davoli, and F. Cucchietti. Energy efficiency in the future internet: a survey of existing approaches and trends in energy-aware fixed network infrastructures. *Communications Surveys & Tutorials, IEEE*, 13(2):223–244, 2011.

[5] W. Vereecken, W. Van Heddeghem, M. Deruyck, B. Puype, B. Lannoo, W. Joseph, D. Colle, L. Martens, and P. Demeester. Power consumption in telecommunication networks: overview and reduction strategies. *Communications Magazine*, 49(6):62–69, 2011.

[6] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang, and S. Wright. Power awareness in network design and routing. In *Proc. of INFOCOM-2008*, pages 457–465, Phoenix, Arizona.

[7] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and S.C. Diot. Packet-level traffic measurements from the Sprint IP backbone. *Network*, 17(6):6–16, 2003.

[8] L. Chiaraviglio, D. Ciullo, M. Mellia, and M. Meo. Modeling sleep modes gains with random graphs. In *Proc. of IEEE INFOCOM-2011, Computer Communications Workshops*, pages 355–360. IEEE.

[9] E. Amaldi, A. Capone, and L.G. Gianoli. Energy-aware ip traffic engineering with shortest path routing. *Computer Networks*, 57(6):1503 – 1517, 2013.

[10] M. Zhang, C. Yi, B. Liu, and B. Zhang. Greente: Power-aware traffic engineering. In *Network Protocols (ICNP), 18th IEEE International Conference on*, pages 21–30. IEEE, 2010.

[11] A.P. Bianzino, L. Chiaraviglio, and M. Mellia. Distributed algorithms for green ip networks. In *Proc. of IEEE INFOCOM-2012, Computer Communications Workshops*, pages 121–126.

[12] Aruna Prem Bianzino, Luca Chiaraviglio, Marco Mellia, and Jean-Louis Rougier. Grida: Green distributed algorithm for energy-efficient ip backbone networks. *Computer Networks*, 56(14):3219 – 3232, 2012.

[13] R. Bolla, R. Bruschi, and M. Listanti. Enabling backbone networks to sleep. *Network*, 25(2):26–31, 2011.

[14] N. Vasić and D. Kostić. Energy-aware traffic engineering. In *Proc. of the 1st International Conference on Energy-Efficient Computing and Networking*, pages 169–178. ACM, 2010.

[15] M. Shenand H. Liu, K. Xu, N. Wang, and Y. Zhong. Routing on demand: toward the energy-aware traffic engineering with ospf. pages 232–246, 2012.

[16] B. Addis, A. Capone, G. Carello, L.G. Gianoli, and B. Sansò. Energy management through optimized routing and device powering for greener communication networks. *Networking, IEEE/ACM Transactions on*, PP(99):1–1, 2013.

[17] S. Avallone and G. Ventre. Energy efficient online routing of flows with additive constraints. *Computer Networks*, 56(10):2368 – 2382, 2012.

[18] E. Amaldi, A. Capone, L.G. Gianoli, and L. Mascetti. A milp-based heuristic for energy-aware traffic engineering with shortest path routing. *Network Optimization*, pages 464–477, 2011.

[19] L. Chiaraviglio, M. Mellia, and F. Neri. Minimizing isp network energy cost: Formulation and solutions. *IEEE/ACM Transactions on Networking (TON)*, 20(2):463–476, 2012.

[20] B Addis, A. Capone, G. Carello, L.G. Gianoli, and B. Sansò. A robust optimization approach for energy-aware routing in MPLS networks. In *Proc. of ICNC-13, 28-31 January, San Diego, USA*. IEEE, 2013.

[21] A. Cianfrani, V. Eramo, M. Listanti, M. Polverini, and A. Vasilakos. An ospf-integrated routing strategy for qos-aware energy saving in ip backbone networks. *Network and Service Management, IEEE Transactions on*, PP(99):1–14, 2012.

[22] A. Capone, D. Corti, L.G. Gianoli, and B. Sansò. An optimization framework for the energy management of carrier ethernet networks with multiple spanning trees. *Computer Networks*, 56(17):3666 – 3681, 2012.

[23] S.S.W. Lee, P.K. Tseng, and A. Chen. Link weight assignment and loop-free routing table update for link state routing protocols in energy-aware internet. *Future Generation Computer Systems*, 28(2):437–445, 2012.

[24] L. Chiaraviglio and A. Cianfrani. On the effectiveness of sleep modes in backbone networks with limited configurations. In *Proc. of IEEE SoftCOM-2012*, pages 1 –6, sept. 2012.

[25] Java framework for SNMP-based Network Management (JNetMan). http://www.jnetman.org.

[26] A. Mackarel and et al. Study of environmental impact, dn3.5.2, geant project, May 2011.

[27] Introduction to Cisco IOS NetFlow - A Technical Overview. Tech. rep., Cisco Systems, 2012.

[28] P. Casas, S. Vaton, L. Fillatre, and L. Chonavel. Efficient methods for traffic matrix modeling and on-line estimation in large-scale ip networks. In *Proc. of 21st International Teletraffic Congress, ITC 2009*, pages 1–8. IEEE, 2009.

[29] The SNMP API for Java (SNMP4J) [Online]. http://www.snmp4j.org.

[30] M. Rimondini. Emulation of computer networks with netkit. *Dipartimento di Informatica e Automazione, Roma Tre University, http://www. netkit. org/, RT-DIA-113-2007*, 2007.

[31] M. Pizzonia and M. Rimondini. Netkit: easy emulation of complex networks on inexpensive hardware. In *Proc. of TRIDENTCOM 2008*, pages 1–10. ICST, 2008.

[32] Quagga Routing Software Suite [Online]. http://www.nongnu.org/quagga/.

[33] Net-SNMP suite [Online]. http://www.net-snmp.org.

[34] A. Botta, A. Dainotti, and A. Pescapé. A tool for the generation of realistic network workload for emerging networking scenarios. *Computer Networks*, 2012.

[35] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski. Sndlib 1.0 survivable network design library. *Networks*, 55(3):276–286, 2010.