Responsive Algorithms for Handling Load Surges and Switching Links On in Green Networks

Radu CÂRPA, Marcos Dias de ASSUNÇÃO, Olivier GLÜCK, Laurent LEFÈVRE , Jean-Christophe MIGNOT
Inria Avalon - LIP Laboratory - École normale supérieure of Lyon, University of Lyon, France
Email: radu.carpa@ens-lyon.fr, marcos.dias.de.assuncao@ens-lyon.fr, olivier.gluck@ens-lyon.fr,
laurent.lefevre@inria.fr, jean-christophe.mignot@ens-lyon.fr

***Abstract*—Reducing the energy consumed by wired computer networks is a challenge that has been actively investigated over the past few years. A popular mechanism proposed to reduce the consumption aims to put links and line cards to sleep mode during off-peak hours. Such a mechanism, however, decreases the available network capacity and increases the risk of congestion if traffic rises unexpectedly. This paper proposes a solution to rapidly react to network bursts and turn-*on* sleeping links, which we term as SegmenT Routing based Energy Efficient Traffic Engineering for switching ON (STREETE-ON). The proposed algorithm was implemented in the OMNeT++ network simulator using state-of-art dynamic graph algorithms. In such a way, we achieved execution times of tens of milliseconds for a 50-node network. Experimental results show that STREETE-ON can effectively prevent network congestion, avoid turning-*on* unneeded links, and preserve good energy-efficiency of the network.**

I. INTRODUCTION

The rising traffic demands of today's services – *e.g.* those relying on cloud computing, video-on-demand, and big-data processing – continue to demand increasing data transfer rates from underlying networks. Network operators, who feel continuous pressure to differentiate and deliver quality of service, often tackle this challenge by expanding network capacity, hence constantly adding new equipments and links, or increasing the rates of existing links. Certain studies, however, argue that if traffic continues to grow at the current rate, in less than 20 years network operators may reach an energy capacity crunch where the power grid may be unable to provide the energy the overall network requires [1].

Major organisations have been trying to minimise the energy consumption of their infrastructure by reducing the number of required equipments and maximising utilisation. Google, for instance, created custom Software Defined Network (SDN) devices and prioritised traffic in order to achieve near-100% utilisation of intra-domain links [2]. With an intelligent traffic orchestration, Google is able to reduce the energy consumption while providing high quality of service. However, such a traffic regulation requires full knowledge of communicating applications and their demands; factors over which ordinary network operators usually do not have control. As a result, they provision network capacity to handle peak load, while devices consume energy even under low utilisation. Evidence shows that the utilisation of production 100Gbps links can be as low as 10% during normal operation [3].

As network equipments usually draw the same amount of power regardless of their utilisation [4], a commonly exploited means to reduce the energy consumed by a network consists in setting its equipments to low power modes when partially used, or switching them off completely. While turning links *off* is a widely exploited technique at a conceptual level, most previous work does not consider dynamic networks. The literature describes several techniques, including Mixed Integer Linear Programming (MILP) formulations that are solved offline and rely on estimated traffic matrices [5]. Complex linear programming formulations, heuristics and meta-heuristics have shown that substantial energy savings can be achieved. However, these approaches are often slow, taking tens of minutes to compute a solution. Arguably, their viability is justified by assuming that network traffic varies slowly and presents diurnal patterns; periods of low traffic at night and peak demand during the day.

This assumption, however, does not hold true at short time scales [6]. The shorter the interval over which the average traffic is measured, the more variations become noticeable. A slow variation when measuring traffic per day may contain multiple bursts when zoomed in and analysed at smaller scales. Moreover, previous work highlights that traffic bursts are more visible when flow aggregation is low, which usually corresponds to a moment of low network utilisation. Considering traffic bursts at small scales is important in order to determine when network equipments should be brought back to fully operational state after being shut down to save energy. It is important under such scenarios to react quickly to bursts when links that have been previously turned *off* become necessary.

The literature has thus far mostly overlooked the problem of turning links *on*. One of the exceptions [7] aims to partition the network over cycles. At each cycle, a single link can be turned *off* and the traffic is rerouted over the remaining links. Reacting to congestion is achieved by turning *on* the respective link. A distributed approach is also proposed [8] where a node evaluates at a random time interval whether its adjacent links should be turned *on*; a solution that reacts slowly to network changes. Another approach consists in turning-*on* all links at a pre-defined time in the morning, when traffic increases [9]. If an unexpected burst happens during the night, however, certain network links will remain in low power consumption mode. A previous work uses traffic matrices to estimate the best links to be turned *on* [10], considering only the actual network state.

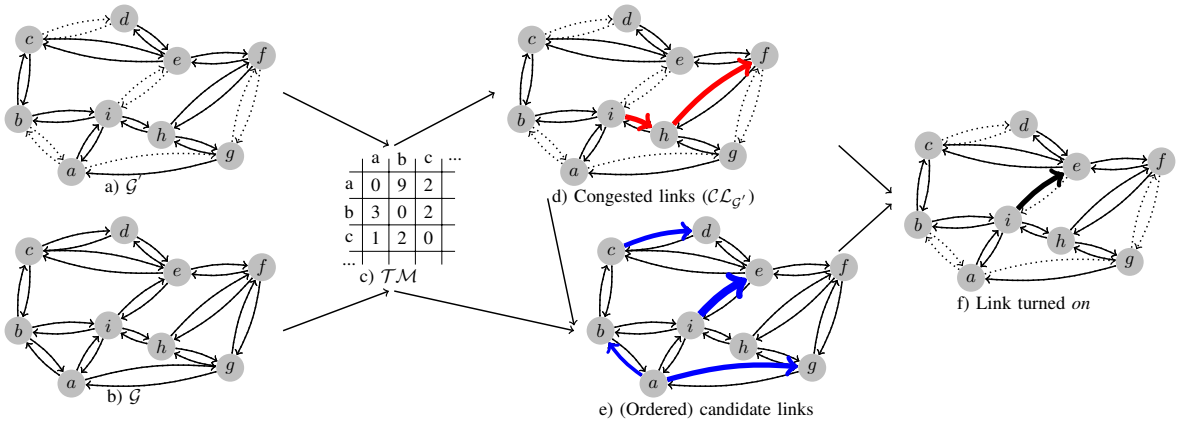


Figure 1: STREETE-ON: simulating the all-on network view to select the best link(s) to turn on

In this work, we propose schemes to react fast to traffic bursts in operators' core networks and to switch *on* previously turned-*off* links as quick as possible. Our solution aims to complement previous work by enabling fast reaction to unexpected network events while further optimisation can be performed in background using the previously proposed solutions. We analyse the impact of traffic variations in an energy-optimised topology and evaluate the trade-offs between energy savings and packet loss while attempting to minimise the number of times links are turned-*off*/*on*. In particular, we evaluate the impact of the time needed for the solution to react to network changes.

The rest of this paper is structured as follows. Section II introduces the notations and formalises the problem. Our solution is presented in Section III and the results are presented and discussed in Section IV. Section V describes ongoing work, and Section VI concludes the paper.

II. Formal Description of the Links-On Process

The challenge of turning links *off* for reducing the energy consumption of networks was frequently considered by the research community. In this work, we consider unexpected traffic bursts in a network with links that have been turned *off*. Our goal is to react as fast as possible by turning links back *on* to avoid congestion.

This section presents our assumptions, introduces the notations used in the paper, and provides an abstract graph formulation of the problem.

A. Network Context

The work takes into account an operator network with the following characteristics:

a) Virtual Tunnels: We assume that a full mesh of virtual tunnels among the nodes of the network allows to change the path of the data passing through the network dynamically. All the data entering the network domain at a source node s and travelling towards the destination node d will be transmitted through the tunnel $s \rightarrow d$. The paths of tunnels are modified to route the traffic through the links kept *on* and avoid switched *off* links.

A classical way to create virtual tunnels consists in using the MPLS protocol as forwarding dataplane and the RSVP-TE protocol to signal the tunnels and change their paths.

b) Monitoring the Bandwidth: Network nodes are capable to monitor the bandwidth used by each of the virtual tunnels starting at a given node.

c) Centralised Management: A centralised path computation engine is available on the network and has the ability to collect statistics from nodes and change the paths of data flows in the network.

d) Unidirectional Links: We assume that links can be turned *on* and *off* independently in each direction without affecting the opposite direction. For example, in long-distance optical links, the transponders may be unidirectional. Turning a link *off* will correspond to turning *off* the transponder and part of the line card.

e) Homogeneous Links: We assume that the network link speeds and their energy consumption are homogeneous.

B. Graph Modelling

Consider a representation of the network by a connected directed graph $\mathcal{G}(V, E)$. Each network device corresponds to a vertex $v \in V$ and each link to an edge $e \in E$. A function $c : E \rightarrow \mathbb{R}^+$ represents the capacities of the edges, *i.e.* the link speed. A function $l : E \rightarrow \mathbb{R}^+$ represents the length (cost) of the edge.

The traffic entering the network at the source node $s \in V$ and flowing towards the egress destination node $d \in V$ is defined as the flow $f_{s,d} \geqslant 0$. In this work we assume that there is no traffic from a node to itself, *i.e.* $\forall s \in V, f_{s,s} = 0$. Intuitively, $\forall d \in V$, the value of $f_{s,d}$ will be computed by the node s by monitoring the bandwidth passing through the virtual tunnel $s \rightarrow d$.

A network-wide traffic matrix $\mathcal{TM} = [f_{i,j}]$ is a square matrix where each case i, j corresponds to the value of the flow $f_{i,j}$ (Figure 1c). We consider that this traffic matrix is assembled on the network controller by collecting the data from all the nodes.

A path $\mathcal{P} = (e_1, e_2, ...)$ is a sequence of edges such that $((e_i = (a, b)$ and $e_{i+1} = (c, d)) \Rightarrow b = c)$. The length of a

path $\mathcal{P} = (e_1, e_2, ...)$ is the sum of the lengths of the edges in the path: $l(\mathcal{P}) = \sum_{e \in \mathcal{P}} l(e)$. A shortest path from the node s to d is a path such that there is no other path $\mathcal{P}'$ with $l(\mathcal{P}') < l(\mathcal{P})$.

For each $(s, d) \in V^2, s \neq d$, there exists at least one path from s to d and the tunnel $s \to d$ transporting the flow $f_{s,d}$ always traverses the network via the shortest path from s to d in $\mathcal{G}$. The selection among multiple shortest paths of the same length is randomly-deterministic, *i.e.* the first path found during the computations is chosen. It depends on the initial construction of the in-memory data structure, but it is always the same for a given in-memory structure.

The utilisation of an edge in the graph $\mathcal{G}$ with a certain traffic matrix $\mathcal{TM}$ is represented by the function $u_{\mathcal{G},\mathcal{TM}} : E \to [0, 1]$ defined as the sum of the flows passing through the edge divided by the capacity of the edge.

For a given network $\mathcal{G}$ and a traffic matrix $\mathcal{TM}$, we define the set of links at critical load $\mathcal{CL}_{\mathcal{G},\mathcal{TM}} = \{e \in \mathcal{G} \mid u_{\mathcal{G},\mathcal{TM}}(e) > \alpha\}$. It contains the links with an utilisation higher than α in $\mathcal{G}$. For example, if $\alpha = 0.8$, $\mathcal{CL}_{\mathcal{G}'}$ will contain the links with an utilisation higher then 80%. Throughout this paper we'll refer to links in $\mathcal{CL}$ as "links close to congestion".

III. STREETE-ON

A. Problem Statement

Figure 1 illustrates our algorithm. Let $\mathcal{G}$ be the initial network topology, we introduce $\mathcal{G}'$ such that $V' = V$ and $E' \subset E$, *i.e.* a subset of links is switched *off*. An example of $\mathcal{G}$ and $\mathcal{G}'$ can be seen in Figure 1*a* and 1*b*.

A novelty is that the algorithm uses both the energy-efficient network topology $\mathcal{G}'$ and the initial full network $\mathcal{G}$ in order to take a decision. We consider that the best quality of service that a network can provide is achieved by keeping all the links *on*. Hence, we leverage knowledge about the all-*on* network to decide which links are the best candidates to be turned *on* and avoid congestion.

Starting from the energy efficient network topology $\mathcal{G}'$, it uses the traffic matrix to find the links at critical load $\mathcal{CL}_{\mathcal{G}'}$. The full network topology $\mathcal{G}$ is then used to select good candidate links. The algorithm finishes by doing a rapid in-memory simulation of turning those links *on* and estimating the impact of a decision without actually turning links *on* in the physical network. Whenever the computation decides that it is worth turning links *on* to reduce the utilisation of links from $\mathcal{CL}$, the decision is sent to network nodes and the physical links are turned on.

The controller hence aims to solve the following problem: if $\mathcal{CL}_{\mathcal{G}',\mathcal{TM}} \neq \emptyset$, find a subset of edges $\supseteq (E \setminus E')$, such that, by inserting them into $\mathcal{G}'$, $\mathcal{CL}_{\mathcal{G}',\mathcal{TM}}$ would become an empty set.

B. Algorithm

Algorithm 1 details our solution. Our greedy heuristic consists in first trying to turn-*on* the links which are *off* in the energy-optimised network $\mathcal{G}'$, but would have a high utilisation in $\mathcal{G}$ (lines 3 and 8). By using this heuristic we prioritise turning *on* the links which would create shortcuts for large flows. For example, in Figure 1 the link ab is selected as a candidate by analysing the traffic matrix and the two topologies. It is a link which is *off* in $\mathcal{G}'$, but would transfer much data in $\mathcal{G}$ (the flows $f_{a,b}$ and $f_{a,c}$).

Algorithm 1 STREETE-*ON*

```
1:  procedure STREETE-ON(G, G', TM)
2:      result ← ∅
3:      candidateLinks ← E \ E'
4:      congested ← CL_{G',TM}
5:      changed ← true
6:      while congested ≠ ∅ and changed = true do
7:          changed ← false
8:          for all e ∈ sorted(candidateLinks, u_{G,TM}) do
9:                                  ▷ sorted in descending order of utilisation in G
10:             if congested ≠ ∅ then
11:                 E' ← E' ∪ {e}                    ▷ Turn on the link e
12:                 congested_after ← CL_{G',TM}
13:                 if congestion decreased then
14:                     changed ← true
15:                     candidateLinks ← candidateLinks \ {e}
16:                     congested ← congested_after
17:                     result ← result ∪ {e}
18:                 else
19:                     E' ← E' \ e                  ▷ Do not keep e on
20:                 end if
21:             end if
22:         end for
23:     end while
24:     if congested ≠ ∅ and CL_{G,TM} ≠ ∅ then
25:         result ← result ∪ (E \ E')
26:         E' ← E                                   ▷ Turn all the links on
27:     end if
28:     return result
29: end procedure
```

At lines 11-12 the algorithm simulates turning the link *on* and estimates the congestion after this operation. At lines 13-20 it tests if congestion decreased. If yes, the link will be scheduled to be turned *on* in the physical network, else the link is not scheduled for ignition.

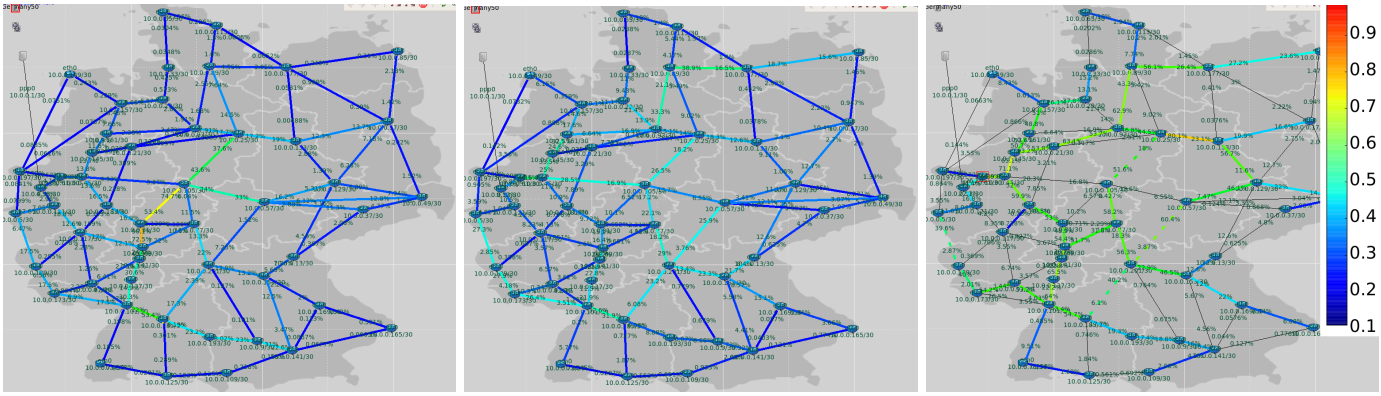
"congestion decreased" at line 13 is defined as:

- solving all the congestion: $congested_{after} = \emptyset$
- or not creating new congested links, while decongesting at least one of them:
  ($\nexists e$ s.t. $e \notin congested$ and $e \in congested_{after}$) and
  ($\exists f$ s.t. $f \in congested$ and $f \notin congested_{after}$)
- or not creating new congested links, while decreasing the utilisation of at least one of congested:
  ($\nexists e$ s.t. $e \notin congested$ and $e \in congested_{after}$) and
  ($\exists f \in congested$ s.t. $u_{\mathcal{G}',\mathcal{TM}}(f)$ decreased)

Sometimes a combination of links must be turned *on* to be able to avoid congestion. The while loop at line 6 ensures the repetition of the algorithm until the congestion is solved.

The presented algorithm hides a complex operation in accesses to $\mathcal{CL}$ at lines 4, 12 and 24: recomputing the shortest paths to estimate the load of links and find the congested ones.

A naive approach for recomputing the shortest paths uses n static Dijkstra computations. In this work we seek to react



(a) Initial load in $\mathcal{G}$ with a $\mathcal{TM}$ from SNDLib (b) Initial load in $\mathcal{G}$ with a generated $\mathcal{TM}$ (c) Example of initial $\mathcal{G}^{'}$ in Germany 50

Figure 2: Examples of $\mathcal{G}$ and $\mathcal{G}^{'}$ in the Germany 50 network

fast to a network congestion, so we used a optimised dynamic All Pairs Shortest Path algorithm from the literature [11]. It is designed to execute only partial re-computation of the shortest paths in case of edge insertion/deletion. Among the algorithms analysed by the authors, we decided to use *D-RRL*: a slightly modified version of the algorithm initially proposed in [12]. The maximum theoretical complexity of this algorithm is the same as static Dijkstra computations. However, in practice it proves itself much faster.

C. Implementation

As stated beforehand, we use network tunnels to transparently reroute traffic. However, the protocol usually used to signal virtual tunnels, RSVP-TE, scales badly with network size. The worst case number of states that a network device must keep active for a full mesh of virtual tunnels is $O(|V|^2)$ and in industry it is considered a best practice to not use RSVP-TE to signal a full mesh of tunnels.

To reduce the complexity, we use the SPRING protocol, which is a mix of source routing and shortest path routing protocols. Due to its source routing nature, changes in the paths of a tunnel must be applied only on the node situated at the inbound end of the tunnel. No time and signalling are lost re-configuring the midpoint devices. This enables fast flow setup and easy reconfiguration of virtual circuits with minimum overhead. Moreover, for a full mesh of virtual tunnels, each node must keep exactly $|V| - 1$ states: one per each tunnel which starts at the concerned node. An interested reader can relate to our previous work [13] or to the draft IETF RFC [14] for a more detailed overview of the SPRING protocol, which is a very efficient way to simplify the implementation compared to MPLS + RSVP-TE.

IV. Performance Analysis

The proposed STREETE-ON algorithm was implemented in the OMNeT++ discrete event simulator [15]. The STREETE-ON algorithm is executed by a centralised controller when a link at critical load is detected. Figure 2c shows the load distribution in a network before executing any turn-on algorithm. The small black lines represent the links in a sleep state, while the dotted ones are active only in one direction. The colour illustrates the utilisation of the links.

A. Analysed Algorithms

We evaluate the performance of our solution against the following intuitive algorithms which allow to take fast turn-*on* decisions:

- Turn *on* the links in the inverse order that they were switched off. Hereafter we refer to this algorithm as *lastoff*.
- Turn *on* all the links, referred to as *allon*.
- Turn *on* the links concentrically around the most congested one; hereafter referred to as *locality*.

Similarly to STREETE-ON, the examined algorithms simulate the turn-*on* process in memory. Only after choosing all the links that must be turned-*on*, the decision is applied to the physical network.

f) lastoff: the algorithm 2 consists in turning-*on* one by one the links in the inverse order they were switched off until congestion is avoided. At line 3, the last turned-*off* link is selected among those being *off*, after that the link is scheduled to be turned-on at lines 4-5. The operation is repeated until the congestion is solved or all the links are turned-*on*.

Algorithm 2 *lastoff*

1: $result \leftarrow \emptyset$
2: **while** $\mathcal{CL}_{\mathcal{G}^{'},\mathcal{TM}} \neq \emptyset$ **and** $E \setminus E^{'} \neq \emptyset$ **do**
3: $\quad e \leftarrow lastTurnedOff(E \setminus E^{'})$
4: $\quad E^{'} \leftarrow E^{'} \cup \{e\}$
5: $\quad result \leftarrow result \cup \{e\}$
6: **end while**

g) allon: the algorithm 3 turns-*on* all the links (lines 2-3) as soon as a link at critical load is detected in the network (line 1).

Algorithm 3 *allon*

1: **if** $\mathcal{CL}_{\mathcal{G}',\mathcal{TM}} \neq \emptyset$ **then**
2: $result \leftarrow E \setminus E^{'}$
3: $E^{'} \leftarrow E$
4: **else**
5: $result \leftarrow \emptyset$
6: **end if**

h) locality: at line 3, the algorithm 4 searches for the link with the highest utilisation in $\mathcal{G}^{'}$. After that, at line 5 it executes an Breadth-first search algorithm to traverse the graph up to a maximum depth given in parameter and to extract all the sleeping links. All the found links are turned *on*. If the congestion is not avoided, the algorithm is repeated with larger depth until turning *on* all the links or avoiding the congestion.

The algorithm corresponds hence to turning-*on* links in cycles around the one with the highest utilisation.

Algorithm 4 *locality*

1: $result \leftarrow \emptyset$
2: $depth \leftarrow 1$
3: $maxe = maxUtilizationLink(\mathcal{CL}_{\mathcal{G}',\mathcal{TM}})$
4: **while** $E \setminus E^{'} \neq \emptyset$ **and** $\mathcal{CL}_{\mathcal{G}',\mathcal{TM}} \neq \emptyset$ **do**
5: $toTurnOff \leftarrow (E \setminus E^{'}) \cap linksInBFS(\mathcal{G}, maxe, depth)$
6: $E^{'} \leftarrow E^{'} \cup toTurnOff$
7: $result \leftarrow result \cup toTurnOf$
8: $depth \leftarrow depth + 1$
9: **end while**

B. Experimental Setup

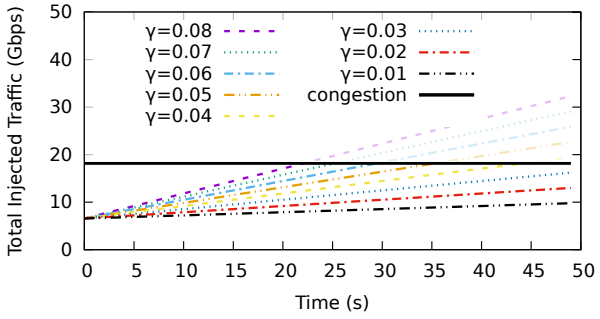


Figure 3: Influence of γ on total injected traffic.

Each link in OMNeT++ was simulated with a speed of 3Gbps. The propagation delay was given as the bird-fly distance by Google Maps API.

To validate our solution on a realistic use case, we used the Germany 50 (Figure 2) topology and associated traffic matrices from SNDLib [16]. The network has 50 routers and 176($88 \cdot 2$) links . However, using the real traffic matrices from SNDLib also has its shortcomings, since the distribution of the loads throughout the network is stable. For example, in Figure 2a we observe that the most loaded links are concentrated around the node situated in Frankfurt, which corresponds to reality as Frankfurt is a major international interconnect node. Unfortunately, most traffic matrices from SNDLib have a similar distribution of the load.

To be able to analyse the efficiency of the algorithms, we would like to test them with different load distributions. When comparing two algorithms on the same network topology, one may behave better when the network load is concentrated in the centre, while another when the load is at the extremities or is uniformly distributed.

We generated multiple traffic matrices based on those from SNDLib. Each generated traffic matrix corresponds to a random permutation of the values of the 2450 flows. As a result, sometimes the most intensive flows have to traverse the entire network, other times are isolated at an extremity. Figure 2b shows different distributions of the initial network load.

To test the turn-*on* algorithms, we start by executing a turn-*off* algorithm which puts the least utilised links to sleep until no more links can be turned-*off* without provoking a congestion. After that, we increase the initial network load by an additive factor γ every second. The value of a flow at the time t of a simulation is equal to $f_{a,b}(t) = f_{a,b}(0) * (1 + \gamma * t)$. To easily visualise the influence of this parameter, Figure 3 shows the variation of the total network load with different values of γ for a fixed traffic matrix. Hence, γ defines the speed of the increase of the traffic flows and allows to test the reactivity of the algorithm. In the most aggressive case, with $\gamma = 0.08$, the network load almost doubles in 5 seconds.

A total of 640 simulation runs were executed. Corresponding to 80 runs per γ with 20 different random load distribution per γ. Each simulation correspond to 50 seconds of network lifetime, as shown in Figure 3.

C. Avoiding Congestion

Intuitively, one can think that algorithms which turn-*on* multiple links at a time will result in a faster avoidance of the congestion, meaning that the network will not reach a state when packets will start being dropped in router queues.

Simulation results summarised in Figure 4 partially confirm that statement. *Allon* has the lowest number of lost packets, while STREETE-ON tends to have slightly worse results compared to other algorithms. However, it must be noted that the difference in number of lost packets is counted in tens of packets. With the simulated link speeds of 3Gbps (250k 1500byte packets per second) it corresponds to less than 0.05% packet lost on the link.

The results in terms of lost packets are difficult to evaluate, because they depend on many parameters. We can however affirm that under the tested network conditions with $\alpha = 0.8$, with the propagation delay of Germany 50, with the tested traffic and selected γ values, all solutions are good in avoiding the congestion and only few packets are lost.

The results from Figure 4 include only the lost packets until the all-*on* network $\mathcal{G}$ become congested (part under the black 'congestion' line in Figure 3).

As the values do not differ much for different γ-values, we describe results for every second value of γ in Figure 4.

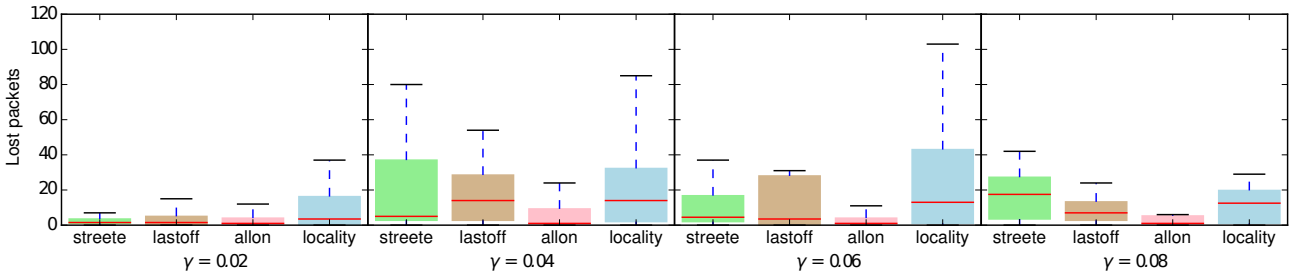


Figure 4: Number of lost packets with different values of γ.

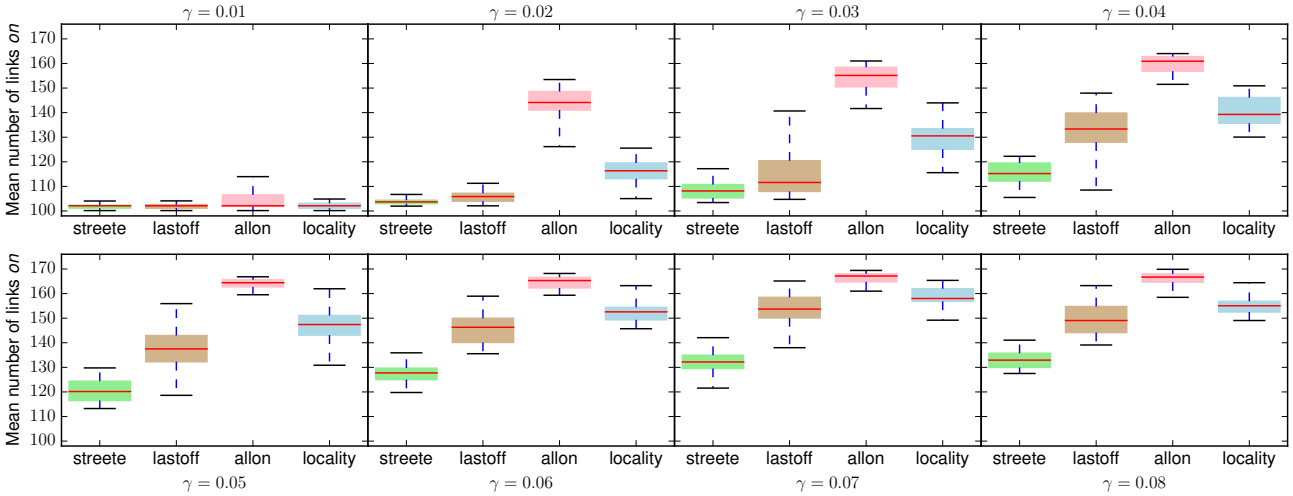


Figure 5: Number of links *on* with different values of γ.

D. Impact on Energy Efficiency

Figure 5 shows the mean number of links *on* during the 50 seconds of simulation run. Each boxplot represents the distribution of the means over 100 runs of simulations for a corresponding γ. Each mean is calculated from samples taken every 1 second, *i.e.* it is a mean of 50 values: number of links *on* at t=1, t=2, ..., t=50.

When links are only turned *on*, the lower the boxplot, the more conservative the algorithm is, avoiding to turn-*on* unnecessary links. If all the links have the same transmission rate and energy consumption, turning-*on* less links results in less energy being consumed. Hence, the energy efficiency is better. Moreover, when used together with a turn-*off* algorithm, avoiding to turn-on unneeded links allows to avoid waking-up network interfaces which will be turned-*off* shortly after, effectively reducing the number of interface's power cycles.

It may be seen that with low values of γ, all the algorithms behave similarly. This is due to the network rarely reaching a congested state. In the small number of cases where turning *on* was necessary, the congestion was avoided by activating links at a time near the end of the simulation.

The *allon* algorithm is worse than others as it unnecessarily turns *on* all the network at the smallest sign of congestion. This tendency is also preserved at highest values of γ.

It is interesting to compare how our algorithm behaves when compared to other conservative algorithms like *lastoff*. Even though all the flows in the network increase by the same factor each second, turning *on* the last turned-*off* link is not the best solution. Actually, it will turn-*on* too many links. STREETE-ON is able to solve the congestion with less links.

As expected, in terms of energy efficiency, the *locality* algorithm does not perform well. It turns-*on* many links around the place of congestion, which is sub-optimal. Hence, we can affirm that STREETE-ON can achieve better energy-efficiency for a network while avoiding congestion as good as the other analysed algorithms.

E. Computational Time

The simulations were performed on an Intel Xeon E5-2620 v2 processor running at 2.10GHz. The average execution time on all the simulation runs of STREETE-ON was of 33ms on the Germany 50 network. The maximum recorded execution time of the algorithm reached 150ms.

The implementation using the state of art dynamic graph all-pair shortest path algorithm showed a very promising result, reducing the computational time from a couple of seconds observed in the case of a naive implementation with statistic Dijkstra computations.

The values of the executed time were computed via standard C++ tools: std::clock() and the CLOCKS_PER_SEC constant.

V. Limitations of the Analysis

Multiple factors can change the outcome of the presented analysis. For example, the speed at which traffic increases may influence packet loss. We analysed this factor using the parameter γ, but under very fast traffic increase, such as when a link utilisation abruptly goes from 0 to 1, the following factors should be considered:

- Time to detect the traffic increase on network nodes.
- Network propagation time to inform the controller.
- Execution time of the algorithm.

For any chosen algorithm, the first two factors will be equal and can hardly be optimised. However, the last factor depends on the computational complexity of the algorithm. The algorithm D-RRL used in our solution has an expensive worst-case complexity, but the worst-case does not happen in practice. Although preliminary tests on a 100-node network showed that the computational time of our solution starts to exceed one second – which is still a good result – we think that it may become a limiting factor in very large networks.

An interesting observation about the use of shortest path routing is that in some cases the congestion is worsened by turning *on* a link, which may not be intuitive. Figure 6 illustrates such a case. Two flows are routed in the network and, when the link *hd* is down, the flows take a detour over longer paths. As a result, none of the links is at critical load. If we turn-*on* the link, the flows are rerouted over *hd* and its utilisation rises almost leading to a congestion. The lines 24 - 27 of STREETE-ON algorithm (Alg. 1) are used as a workaround to avoid never turning the links *on*. However, this solution is disputable.

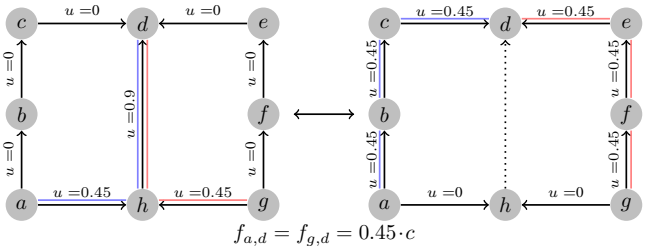


Figure 6: Congestion avoidance by turning *off*

The analysis is performed here considered homogeneous links, whereas STREETE-ON can be extended to consider heterogeneous link speeds. We have chosen the value 0.8 for λ empirically, where a link is considered at high load and therefore close to congestion when it is 80% utilised. With smaller values of λ, less links will be turned off. With higher values, there is more probability to start dropping packets in router queues.

VI. Conclusion

Solutions for improving the energy efficiency of wired computer networks propose to turn the links *off* to reduce energy consumption. This paper proposed a reactive algorithm which dynamically monitors the network utilisation and turns the links back *on* when a state close to congestion is detected. The algorithm analyses simultaneously the actual state of the network with links turned off ($\mathcal{G}'$), and the all-on state ($\mathcal{G}$) in order to choose the best candidate links to be switched *on*. It provides a fast solution which rapidly reacts to a risk of congestion while avoiding to turn-*on* too many links, thus keeping a low energy consumption by the network.

Acknowledgments

This work is financially supported by the CHIST-ERA STAR [17] project.

References

[1] D. Kilper, "Energy challenges in access and aggregation networks." Symposium: Communication networks beyond the capacity crunch, May 2015. Accessed: sep/2015.

[2] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined wan," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, (New York, NY, USA), pp. 3–14, ACM, 2013.

[3] "Geant network looking glass and usage map." https://tools.geant.net/portal/. Accessed: sep/2015.

[4] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang, and S. Wright, "Power awareness in network design and routing," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, April 2008.

[5] W. Van Heddeghem, B. Lannoo, D. Colle, M. Pickavet, and P. Demeester, "A quantitative survey of the power saving potential in ip-over-wdm backbone networks," *Communications Surveys Tutorials, IEEE*, vol. PP, no. 99, pp. 1–1, 2014.

[6] R. De Schmidt, R. Sadre, and A. Pras, "Gaussian traffic revisited," in *IFIP Networking Conference, 2013*, pp. 1–9, May 2013.

[7] Y. Yang, M. Xu, and Q. Li, "Towards fast rerouting-based energy efficient routing," *Computer Networks*, vol. 70, pp. 1 – 15, 2014.

[8] A. Bianzino, L. Chiaraviglio, and M. Mellia, "Grida: A green distributed algorithm for backbone networks," in *Online Conference on Green Communications (GreenCom), 2011 IEEE*, pp. 113–119, Sept 2011.

[9] F. Francois, N. Wang, K. Moessner, and S. Georgoulas, "Optimizing link sleeping reconfigurations in isp networks with off-peak time failure protection," *Network and Service Management, IEEE Transactions on*, vol. 10, pp. 176–188, June 2013.

[10] M. Kamola and P. Arabas, "Shortest path green routing and the importance of traffic matrix knowledge," in *Digital Communications - Green ICT (TIWDC), 2013 24th Tyrrhenian International Workshop on*, pp. 1–6, Sept 2013.

[11] C. Demetrescu and G. F. Italiano, "Experimental analysis of dynamic all pairs shortest path algorithms," *ACM Trans. Algorithms*, vol. 2, pp. 578–601, Oct. 2006.

[12] G. Ramalingam and T. Reps, "An incremental algorithm for a generalization of the shortest-path problem," *J. Algorithms*, vol. 21, pp. 267–305, Sept. 1996.

[13] R. Carpa, O. Gluck, L. Lefevre, and J.-C. Mignot, "Improving the energy efficiency of software-defined backbone networks," *Photonic Network Communications*, pp. 1–11, 2015.

[14] C. Filsfils, S. Previdi, A. Bashandy, B. Decraene, S. Litkowski, and R. Shakir, "Segment routing architecture", draft-ietf-spring-segment-routing-06 (work in progress)," October 2015.

[15] "OMNeT++ Discrete Event Simulator." https://omnetpp.org/.

[16] "SNDlib: a library of test instances for Survivable fixed telecommunication Network Design." http://sndlib.zib.de/.

[17] "CHIST-ERA STAR (SwiTching And tRansmission) project." http://www.chistera.eu/projects/star.