Optimal adaptive routing in packet-switched networks

Bruno Gaujal and Baptiste Jonglez

Univ. Grenoble Alpes
Inria
ENS Lyon
baptiste.jonglez@ens-lyon.org
bruno.gaujal@inria.fr

1. Introduction

Communication networks are becoming increasingly multipath and a common challenge is to exploit this path diversity. More precisely, the problem can be modelled as a *multi-commodity flow problem* : Given a number of concurrent source-destination flows, the problem is to assign these flows to network paths, while respecting capacity constraints.
Here, we present a novel algorithm for adaptive routing in arbitrary network topologies, mapping source-destination flows to paths. We claim that it provides a viable and stable solution to adapt to traffic conditions, and effectively avoids congestion. Our algorithm is based on theoretical grounds from game theory, while our implementation leverages SDN protocols to ease deployment.
On the theoretical side, our distributed routing algorithm is endowed with the following desirable properties for efficient implementation :

- It is fully distributed without any information sharing ;
- It is oblivious to the network topology ;
- It only uses on-line and local information ;
- There are no endless oscillations and it is numerically stable ;
- It is robust to out-dated information and measurement errors ;
- It does not require time synchronization between routers ;
- and it converges fast even if the number of flows is very large.

2. Routing Algorithm

Let us consider a routing problem in a communication network. Several *flows* of packets must be routed over a communication network. The topology of the network is fixed but arbitrary.
Each flow $k \in \mathcal{K}$ is characterized by a source-node, a destination-node and a nominal arrival rate of packets, λ_k. Also, each flow is affected a set $\mathcal{P}_k$ of paths in the network from its source to its destination, made of P_k paths. A *configuration* is a choice of one path per flow. The delay over each link and each node in the network depends on the the load on the link (node), in an unspecified manner. For one flow, say k, we denote by $d_k(p_1, \cdots, p_k, \cdots, p_K)$ the end to end *average delay* experienced by packets of flow k under the configutration where flow 1 uses path p_1, flow 2 uses path p_2, and so forth.
The following algorithm is run by each flow k, independently. It is probabilistic and maintains two vectors of size P_k. The *probabilistic choice* vector, $\mathbf{q}_k = (q_1 \dots q_{P_k})$ gives at each step the probabilities to choose the paths and the *score vector* $\mathbf{Y}_k = (Y_1 \dots Y_{P_k})$ that attributes a score to the paths.
The main loop of the algorithm is as follows (index k is skipped).
At each local clock tick, a path p is chosen according to $\mathbf{q}$, and packets are sent along p. The average delay of packets over this path is measured. The score Y_p is updated according to a discrete dynamics inspired from game theory and in turn, the probability vector is modifed for the next path selection. This repeats forever, or until a stable path has been reached for all flows, *i.e.* $\mathbf{q}$ becomes a degenerate probability vector (all coordinates are zero but one) for all flows. The algorithm uses 3 parameters : τ is a discounting factor over past scores, $(\gamma_n)_{n\in\mathbb{N}}$ is a vanishing sequence of step sizes and $(\beta_n)_{n\in\mathbb{N}}$ is a sequence of bounding terms controlling the growth rate of the scores. In the algorithm, $\wedge$ denotes the minimum operator.

Algorithm 1: OPS : Online Path Selection for k

Initialize :
$n \leftarrow 0$; $\mathbf{q} \leftarrow (\frac{1}{P}, \dots, \frac{1}{P})$; $\mathbf{Y} \leftarrow (0, 0, \dots, 0)$;
repeat
  When local clock ticks for the nth time ;
  $n \leftarrow n + 1$;
  select new path p w.r.t. probability vector $\mathbf{q}$;
  Use path p and measure its delay D ;
  Update score of p :
  $$Y_p \leftarrow \left(Y_p - \gamma_n (D + \tau Y_p)/q_p \right) \wedge \beta_n \, ;$$
  update proba. : $\forall s \in \mathcal{P}_k, q_s \leftarrow \frac{\exp(Y_s)}{\sum_\ell \exp(Y_\ell)}$;
until *end of time*;

Theorem 1 (Convergence to equilibirum)
Under mild technical assumptions, for all $\epsilon > 0$, there exist $\tau > 0$ such that the algorithm converges to an ϵ-optimal configuration, in the following sense :

For each flow ℓ, the probability vector $\mathbf{q}$ converges to an almost degenerate probability : q_p becomes smaller than ϵ for all $p \in \mathcal{P}_\ell$ except for one path, say p^*_ℓ, for which it grows larger than $1-\epsilon$.
Furthermore, after convergence, no flow can reduce its delay : $\forall p' \in \mathcal{P}_\ell$,
$d_\ell(p^*_1, \ldots, p^*_\ell, \ldots, p^*_K) \leqslant d_\ell(p^*_1, \ldots, p', \ldots, p^*_K)$.

The proof is based on a general convergence theorem from game theory, proved in [1]. It is essentially based on two facts. The empirical delay D measured on packets using path p for flow f has no bias, conditionally on the past : At step n, $\mathbb{E}(D|\mathcal{F}_n) = d_k(p_1, \ldots, p_K)$. This implies that the scores $\mathbf{Y}$ form a stochastic approximation of a continuous deterministic dynamics that converges to Nash Equilibria in all potential games.

3. Implementation and experimental Results

Looking at our OPS algorihtm, we can note that it is completely distributed : it requires only local measures, local choices, and no coordination is needed between routers. This eases implementation. The only difficulty lies in the ability to select paths from source to destination : this is not practical in current next-hop forwarding networks. Our implementation is based on an equivalent version of OPS, where one gateway router makes a choice among all possible next-hop routers for each of its flow. Thus, the local actions of several routers between source and destination implicitely determine the path from source to destination.

Our implementation takes the form of an Openflow controller, using the Ryu library. The routing table of each gateway router is programmed and constantly updated by a dedicated controller. Furthermore, each gateway router sends packet headers to its controller, for delay computation.

To run realistic experiments, we use Mininet [2], a widely adopted network emulator. Mininet is used to build a virtual network topology, thanks to the *network namespaces* feature of the Linux kernel. On this virtual topology, we can run our implementation exactly as if we had a real network at hand.

To validate our approach, we use a simple topology, with two gateway routers and three hosts. Two hosts are simply connected to their gateway, while the third host can be reached via two different paths. The topology is shown in Figure 1.

We consider two TCP flows from host 1 and host 2, both destinated to host 3. Each flow is restricted by the sender to use no more than 8 Mbit/s, to avoid saturating all links whatever the choice of flow assignment. If both flows are forwarded over the same rightmost link, congestion will occur, because of the limited capacity.

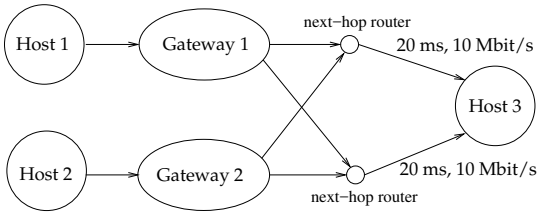

FIGURE 1 – Simple topology, where the two rightmost links have limited capacity (10 Mbit/s) and a latency of 20 ms. Both host 1 and host 2 send a flow to host 3.

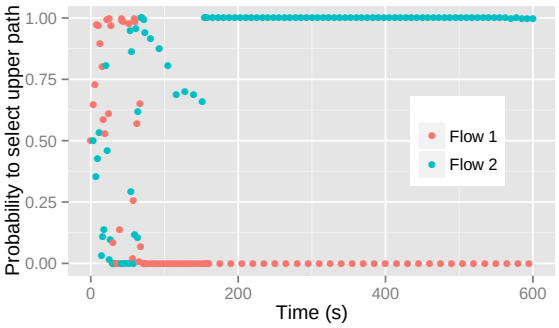

FIGURE 2 – Probability over time to select the upper path for each flow.

Figure 2 shows our experimental results. For each flow, the probability to use the upper path is plotted over time. After an initial period of exploration, the gateway routers converge to a stable state, in which each path to host 3 supports a single flow.

Bibliographie

1. Pierre Coucheney, Bruno Gaujal, and Panayotis Mertikopoulos. Penalty-Regulated Dynamics and Robust Learning Procedures in Games. *Mathematics of Operations Research*, 40(3) :611–633, 2015.
2. Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop : rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 19. ACM, 2010.