ONOS Intent Monitor and Reroute service: enabling plug&play routing logic

Davide Sanvito, Daniele Moro, Mattia Gullì*, Ilario Filippini, Antonio Capone
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano, Italy
{name.surname}@polimi.it - *mattia.gulli@mail.polimi.it

Andrea Campanella
Open Networking Foundation
Menlo Park, CA, USA
andrea@opennetworking.org

***Abstract*—Software-Defined Networking (SDN) brought an unprecedented flexibility and programmability into computer networks. In order to simplify the management of an SDN network, several high-level languages have been proposed. The Intent Framework provided by Open Network Operating System (ONOS), for example, allows programmers to specify high-level policies without worrying about low-level device details, which are inferred by the controller. In addition, the Intent Framework ensures that the objective is met by transparently re-compiling the intents as a consequence of environment changes (e.g. link failures). In this work we extend the Intent Framework to make it able to both compile multiple intents together and to re-optimize their paths according to the network state based on flow statistics. We present Intent Monitor and Reroute service, a new ONOS module to optimize traffic forwarding of any ONOS applications based on intents, via an external plug&play routing logic with no modifications to ONOS applications at all. Finally we evaluate the benefits by enhancing the ONOS SDN-IP application with an adaptive Robust Traffic Engineering algorithm.**

I. Introduction

Software-Defined Networking brought a revolution in computer networks: thanks to the separation of the Control Plane (CP) and Data Plane (DP) and a common open programming interface between them, networks are more programmable and flexible to manage. Despite the flexibility of the match-action abstraction, the configuration of the set of flow rules in each single device to obtain a desired global network policy represents a really error prone task. To make networks not only more programmable, but also easier to manage, several high-level languages have been proposed [1].

At the same time, a complementary paradigm arose: intent-based networking. Through it, application developers can specify a high-level policy without worrying about how the required functionality will be implemented in the network. The SDN controller offers a North-Bound Interface (NBI) to submit intents and translate them, via a compilation process, to low-level flow rules to achieve the requested objectives. Thanks to the intent-based programming, not only the network complexity is abstracted (e.g., "give connectivity between these 2 points" without explicitly mentioning nodes along the path), but the controller will also transparently handle changes in the underlying network to meet the high-level policy. For example, in case of network failures, the intent will be automatically re-compiled and the network will be reconfigured to restore the requested connectivity without any intervention from the application or the user. Popular SDN controllers already provide intent-based NBI: for example OpenDaylight [2] offers Network Intent Composition interface, while ONOS [3] has the Intent Framework. In the rest of the paper we will focus on the ONOS controller.

On the one hand intent-based programming greatly simplifies a network programmer's work, but on the other hand it hides all the details of the network. One of the main limitations of the Intent Framework is that each connectivity intent gets individually compiled to one of the shortest paths, including constraints about the required bandwidth or a specific set of nodes to be traversed. In this work, we aim at extending the ONOS Intent Framework to jointly consider multiple intents during the compilation and to make it able to re-actively adapt the routing not only according to topology changes, but also based on flow-level statistics events in order to optimize a global network objective, e.g. minimizing Maximum Link Utilization (MLU). We want to integrate a Traffic Engineering (T.E.) logic which can be transparently re-used by any intent-based application. We extended ONOS by introducing a new service, Intent Monitor and Reroute (IMR), which allows ONOS applications and users to specify a set of intents whose statistics are monitored and exposed to an external routing logic. This enables the possibility to dynamically complement ONOS with an external plug&play T.E. module. In order to show the benefits of this proposal, we selected SDN-IP [4], an ONOS application developed by Open Networking Foundation (ONF), which enables the intercommunication of legacy networks speaking BGP through a SDN-based network, completely based on intents. We show how we can exploit the new IMR service to improve network performance, with no modifications to the application code, by interconnecting our new service to an external module. The external module runs Clustered Robust Routing (CRR) [5], an adaptive Robust Traffic Engineering algorithm proposed by some of the authors.

The rest of this paper is organized as follows: Sec. II presents ONOS and its Intent Framework. Sec. III describes our new ONOS service, while Sec. IV describes the SDN-IP application. Finally, Sec. V presents a possible off-platform application to be interconnected to IMR service and Sec. VI reports our conclusions.

978-1-5386-4633-5/18/$31.00

II. ONOS

Open Network Operating System (ONOS) is an open source SDN network operating system built for Service Provider networks. ONOS provides high performance, scalability and availability thanks to its distributed core and proper abstractions. ONOS core is in charge of maintaining the network state, interacting with network devices via the south-bound APIs and offering service to the applications through the north-bound APIs. ONOS provides three network abstractions at different levels of abstraction: Flow Rules abstract the protocol to configure forwarding logic in devices, Flow Objectives abstract the device pipeline and finally Intents abstract the topology.

A. ONOS Intent Framework

Intents represents the highest level of abstraction: programmers can focus on *what* should be done, rather than *how* it should be done, by expressing their "intentions" via high-level policies. For example, a Point To Point intent requests an unidirectional connectivity between two elements in the topology (e.g. two hosts). At the same time, intents can be tied to a specific traffic subset (expressed via a selector, a set of specific values for packet header fields) and associated with a treatment (a set of actions to be applied to all the packets processed by the intent). There are several types of intent and each one is supplied with a compiler which enables ONOS core to translate the high-level policy to low-level rules to be installed in network devices. In addition to endpoints, traffic selector and treatments, some intents includes the possibility to specify a set of constraints to limit the outcomes of the compilation: for example we can require that the resulting paths traverse a given set of nodes or we can ask to reserve a certain amount of bandwidth.

Even if the Intent Framework is designed to be extensible with additional intents and compilers, at the current state, each intent gets individually compiled solely based on its information. One of the most interesting features of the Intent Framework is its ability to transparently recompile an intent in case of topology events (such as network failures) in order to meet the requested objective. In this work, we want to make the Intent Framework able to re-optimize the paths according to topology state acquired through flow level statistics. In addition, the compilation should jointly consider multiple intents together to optimize a global network objective, such as minimizing Maximum Link Utilization (MLU).

Since one of the requirements of ONOS is high performance, we cannot add a computationally heavy component such as an optimization tool in the same machine which runs one of the controller instances. Thus, rather than modifying the Intent Framework itself, we developed an additional ONOS service that can orchestrate the monitoring and rerouting of the intents and we defined a set of APIs to make it communicate with an off-platform application (OPA). OPAs are applications which run in a separate process space and leverage ONOS via REST APIs or gRPC. ONOS community is currently working on adding support to gRPC NBI to most of its core services and, when available, this would provide a faster inter-communication with OPAs, with respect to REST NBI, and a better isolation of applications from ONOS core. The OPA will be in charge of implementing the re-routing logic which can range from classical optimization tools to novel Machine Learning or Artificial Intelligence approaches based on collected statistics. Another advantage of resorting to an external module is the possibility of re-using existing Traffic Engineering tools.

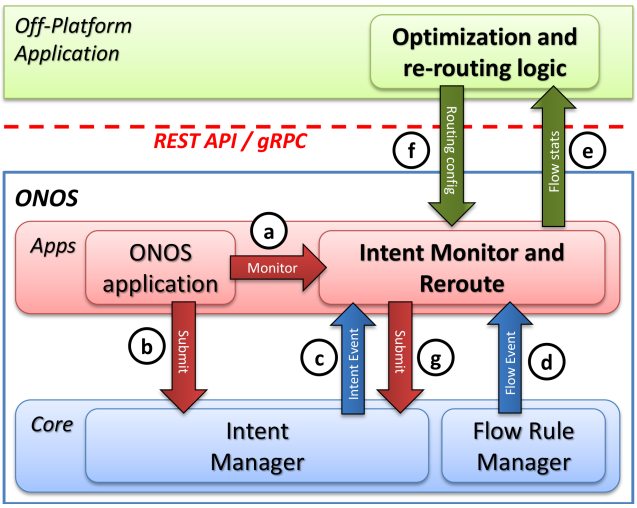

Fig. 1: IMR with ONOS and with the OPA

III. Intent Monitor and Reroute service

The main feature of the Intent Monitor and Reroute service, hereinafter referred to as IMR, is to offer ONOS applications and users the possibility to require the monitoring and re-routing of a specific intent. By monitoring we mean retrieving statistic of each low-level flow generated by ONOS to apply the intent in the network. IMR currently supports two intent types: Point To Point and Link Collection. IMR interacts with ONOS Intent Manager and Flow Rule Manager to keep track of the mapping between the intent and the corresponding flow rules and their information.

Fig.1 depicts the interactions between IMR and other components of ONOS. IMR is not in charge of submitting new intents to the Intent Framework, rather applications or users just require (interaction `a` in Fig.1) the monitoring of an intent via its key. IMR defines three possible states for an intent: `Not Monitored`, `To Be Monitored` and `Monitored`. An intent is by default in `Not Monitored` state[1]. When an application or an user requires to monitor an intent (`a`), IMR updates its state either to `To Be Monitored` or `Monitored`. If the intent has already been submitted by the application (`b`), IMR changes its state to `Monitored`, effectively starting the tracking phase of its statistics. In the other case, IMR moves its state to `To Be Monitored` by putting it into a waiting list. Transitions from `To Be Monitored` state to `Monitored` will be triggered when

[1]This is a pseudo-state: no information is really stored in the IMR for intents in this state.

the related IntentEvent of type *INSTALLED* is received (c) from the Intent Manager. The opposite transition will be instead caused by a *WITHDRAWN* IntentEvent.

The second key feature of IMR is to make an OPA able to collect (e) the latest statistics received (d) for a given monitored intent, regardless of any recompilation performed by the Intent Framework, due to a topology change, for example.

In addition, IMR offers OPAs the possibility of re-routing (g) monitored intents. In this way the OPA, based on the statistics retrieved from IMR, can specify (f) for each intent a particular path in order to optimize a global network objective.

IMR implements thus a service to enable the routing optimization of any ONOS application based on intents with minimal or no modifications to their code. An ONOS application that wants to exploit features of IMR needs just to submit the intent keys to it and then all the work is left to the OPA. The OPA will collect the statistics, apply its optimization algorithm and re-route the intents. IMR can be also used to jointly optimize intents coming from different applications. Finally, IMR might act as a pre-filtering stage for statistics collection by an OPA for purposes other than pure routing, such as monitoring and analytics.

A. CLI APIs

In addition to applications, also users can request the monitoring of intents via IMR CLI commands.

- *imr:startmon appId appName [intentKey]*
- *imr:stopmon appId appName [intentKey]*

These commands require IMR to start or to stop the monitoring of all the intents currently submitted by an application or of a specific intent.

B. REST APIs

IMR exposes a REST API to retrieve the statistics with the following endpoints:

- *GET /intentStats*
- *GET /intentStats/appId/appName*
- *GET /intentStats/appId/appName/intentKey*

OPAs can thus retrieve the statistics of the monitored flows of all the applications, a specific application or a specific intent. Latest available statistics are memorized in ONOS Flow Statistic Store which gets updated according to Flow Rule Event (interaction d in Fig.1). The statistics exposed by IMR are ONOS Flow Entry data (a generalized match-action rule applied to a network device) containing the number of bytes and packets that matched the rule and the life of the rule itself. From this information the OPA is also able to reconstruct the path currently used by an intent.

Once the OPA has retrieved the set of statistics, it might be interested in rerouting some flows. This can be achieved with the second REST endpoint:

- *POST /reRouteIntents*

The endpoint allows to enforce for a specific set of intents a corresponding weighted set of paths via a JSON message. The OPA specifies a path in terms of traversed network devices, while the IMR application itself will translate it into a list of links exploiting the ONOS Link Service and will submit (interaction g in Fig. 1) a new intent with the suggested path.

In order to compute a path for a monitored intent, the OPA requires to know its endpoints and this information can be retrieved via the REST API:

- *GET /monitoredIntents*
- *GET /monitoredIntents/appId/appName*
- *GET /monitoredIntents/appId/appName/intentKey*

Through these endpoints OPA gets the endpoints ID of the intents. The current network topology can be instead retrieved using standard ONOS REST API. The detailed format of JSON messages can be found at [6].

C. Open-source contribution

IMR application has been submitted as open-source contribution to the ONOS codebase [7]. A tutorial and the Wiki of this application can be found at [6]. It is in our plan to extend the support for all the Connectivity intents. Moreover, IMR will also be extended to support gRPC interaction with OPAs to replace current REST APIs. As an additional contribution we also modified the Point To Point intent to support a suggested path through a new attribute. The ONOS compiler treats this information as a *soft* constraint: if the suggested path is not available or a failure occurs, the compiler will transparently fall back to the standard shortest path computation.

IV. SDN-IP ONOS APPLICATION

SDN-IP [4] is an ONOS application which enables a SDN network to connect to legacy IP networks using the standard Border Gateway Protocol (BGP). Externally, the SDN network appears as a traditional Autonomous Systems (AS) exchanging routing information via eBGP and providing connectivity between external networks. Internally, the SDN network is made of a set of a BGP speakers and a network of OpenFlow (OF) [8] devices controlled by ONOS. Internal BGP speakers communicate via iBGP among themselves and with the SDN-IP application running on the controller, which behaves as a passive BGP speaker.

SDN-IP makes a massive use of intents: both the connections between the external BGP speakers and the internal BGP nodes and the connectivity between different AS are managed through intents. In particular, a set of Point To Point intents between external and internal BGP speakers guarantees the connectivity for eBGP peering sessions. In addition, whenever a BGP announcement for a new IP prefix is received from a peering interface, the application creates a Multi Point To Single Point intent enabling connectivity towards that interface for packets coming from any other peering interface. The intent matches packets intended for that IP prefix and modifies the MAC destination address to the physical address of the next-hop router. In case of routing updates (i.e. a better route for a known IP prefix is discovered), the application takes care of replacing the old intent.

In case of SDN-IP, one of the main advantage of relying on the Intent Framework is that ONOS automatically restores connectivity of both BGP sessions and transit traffic between AS, without any effort on the application side.

A. Extended SDN-IP application

In order to show the benefits of IMR service, we modified the SDN-IP application to require IMR to monitor all the intents related to transit traffic. Multi Point To Single Point intents are compiled to shortest paths rules with IP destination based routing. Since IMR does not currently support Multi Point To Single Point intents and since the OPA presented in the section V implements an IP source-destination routing, we adapted the application to submit Point To Point intents. The extended application is available at [9].

V. Off-Platform application routing logic

Once the OPA has retrieved a set of traffic measurements, we can execute any T.E. algorithm to produce a routing configuration to be enforced into the network via the REST API exposed by IMR service. The OPA logic is independent of the application, thus it is not specifically tied to SDN-IP. The most trivial approach would be to re-optimize the routing every time we receive an updated statistics. This would provide the best possible performance for the specific traffic pattern just observed, but there are no guarantees it will be a suitable choice also for the subsequent traffic scenarios. In addition, keeping changing the routing too often can create instabilities into the network. A completely opposite approach is to compute a single stable routing configuration reasonably good for a large number of scenarios (e.g. oblivious routing [10]) and proactively install it for a given amount of time (e.g. one day).

A. Clustered Robust Routing (CRR)

Different routing logics with different network objectives can be implemented by the OPA. As an example, we selected Clustered Robust Routing (CRR) [5], an adaptive Robust Traffic Engineering algorithm proposed by some of the authors. CRR allows to tune the trade off between completely dynamic and completely stable routing. The basic idea is to feed an optimization model with the historical TM data collected over a training period. At the end of the period, by exploiting the quasi-periodicity of the traffic (under some time scales, for example 1 day), the algorithm computes a set of routing configuration to be proactively applied to the network during the following period. To cope with traffic deviations, with respect to the corresponding traffic profile of the training period, routings are computed to be robust over subsets of traffic matrix space.

CRR defines two optimization models: the first one is a Robust Routing (RR) model used to compute a single routing configuration suitable for a discrete set of traffic matrices and which minimizes the average MLU, while a second is used to cluster a set of TMs in the space, time and routing domain. The clusters cover the entire set of TMs collected during the training period and are ordered (i.e. non-contiguous TMs cannot be clustered together). Finally, a routing configuration for each cluster is computed using the RR model. By construction, it is guaranteed that each cluster has a minimum length (enforced as a model parameter), which translates in having a routing configuration which is kept for a controlled minimum amount of time.

CRR OPA has been implemented as a Python application which:

- collects TMs data from ONOS via IMR's REST API
- solves the two optimization models using Gurobi [11] solver
- schedules the activation of the robust routing configurations and applies them via IMR's REST API

Of course the approach can be iterated: after the end of the first training period, the routing configuration is applied and further training data is collected to compute the set of routing configurations to be applied during the third period.

Depending on the training data acquired and on the model parameters, a CRR solution for a given training period might not be found. In this case several possibilities are available ranging from solving different instances of CRR with different parameters to keeping the current network configuration (the last routing configuration of the previous period), from repeating the entire schedule of the previous period to resort to the shortest path per each flow. We leave the evaluation and comparison of the different approaches for a future work.

Despite the ability to cope with small deviations from the expected traffic profile, thanks to the robust nature of each routing configuration, we still need to face link failures. However, the *soft* nature of the constraints we used to enforce a path via IMR service does not limit the effectiveness of the Intent Framework in autonomously recovering from failures [12].

Even if we adopted a combination of proactive approach to follow the evolution of expected traffic scenario and a reactive approach to handle failures, the OPA CRR still plays a key role in coping with unexpected traffic conditions: while applying the scheduled routing configuration, the OPA keeps monitoring traffic not only to feed the model for the subsequent period, but also to verify that the scheduled routing configuration is applied and appropriate for that particular traffic condition. If, for example, the measured TM is not feasible for the current configuration or if the instantaneous MLU significantly differs from the expected, proper countermeasures should be taken (e.g select a more robust configuration or recompute a dedicated routing). The same reasoning applies to topology changes: albeit the essential fast reaction of the Intent Framework to preserve connectivity, link failures might invalidate the current set of routing configurations and trigger a new re-optimization.

B. Numerical results

In this subsection we are going to evaluate the benefits of integrating our extended ONOS SDN-IP application with the CRR OPA via the IMR ONOS service with the aim of optimizing the forwarding of transit traffic without efforts on the application side.

We created an emulated Mininet [13] network replicating the Abilene [14] backbone topology and, by attaching an external BGP speaker with a single host to each node, replayed a subset of 3-days TM data using *iperf* traffic generation tool (we played 5 minutes of Abilene every 15 seconds).

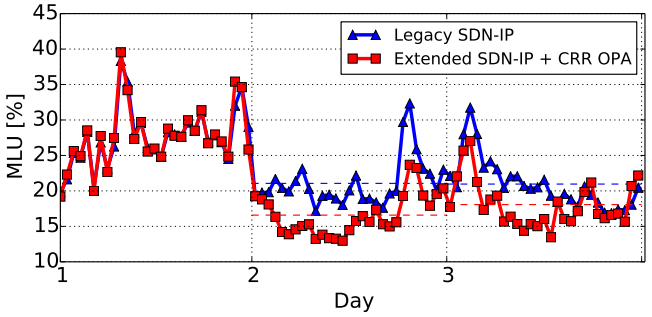

Fig. 2: Average MLU comparison

In Fig. 2 we plotted the Maximum Link Utilization (MLU) for each measurement round (switches' port stats are retrieved by polling ONOS via REST API every 15 seconds) and the average MLU over each day and we compared the legacy SDN-IP application performance with the extended one. In the former case (blue line) flows are forwarded on their shortest path during all the days (default behaviour of Intent Framework). In the latter (red line) the OPA collects for 1 day the statistics of the traffic (which is initially forwarded on the shortest path) and jointly optimizes the paths used for the subsequent day using the CRR algorithm to create three clusters. During the following day, the OPA schedules the routing configurations, according to the output of the CRR, and at the same time collects TM data to optimize the routing for the third day. The approach can be re-iterated for many subsequent days. In Fig. 2 we can appreciate a 5% decrease in the average MLU after the first day of training.

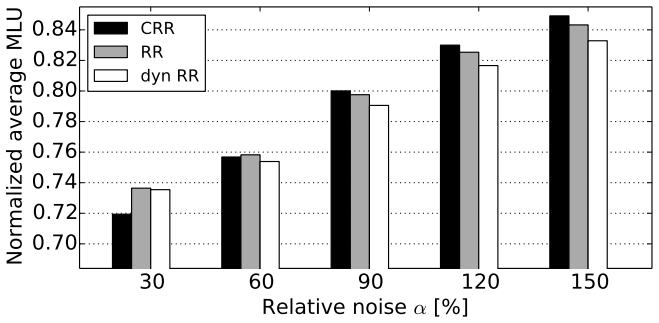

Fig. 3: Handling of unexpected traffic scenarios

As previously pointed out, the monitoring executed by the OPA is a crucial operation to dynamically handle unexpected traffic scenarios such as traffic burst. The best prediction available to the optimization module for the current day is the data of the day just ended. In this experiment, rather than considering data from two subsequent days, we assume the second day to be a noisy version of the previous one (i.e. imperfect prediction). At each measurement round, the OPA verifies if the instantaneous MLU significantly differs from the expected (i.e. the one obtained in the corresponding measure of the previous day) to detect an anomaly condition. The OPA can then interrupt the scheduled routing updates and activate a different solution up to the end of the day. In Fig. 3 we simulated the detection mechanism and compared the average MLU of the remaining portion of the day for different value of percentage noise α[2] by adopting three different approaches: keep the current *set* of routing (CRR), apply a *single* robust routing configuration based on the data of the previous day (RR) or compute a single robust routing configuration based on the latest 24h data (RR dyn). Results have been normalized to the average MLU obtained when resorting to the shortest path case. We can notice that in any case it is better not to fall back to the shortest path (default behaviour of intents). As the noise increases (i.e. the quality of prediction decreases) it is more convenient to adopt a solution which is more robust (i.e. RR instead of CRR) or to re-compute a brand new robust solution (RR dyn) which includes partial (up to the anomaly) updated information on the current day. We leave a more extensive evaluation of possible different reactive approaches to future works.

VI. Conclusion

We presented Intent Monitor and Reroute service, a new ONOS module to optimize the forwarding of any ONOS applications based on intents, via an external plug&play routing logic, with very few modifications to the application code. Numerical results proved that we can easily improve the performance of the network by integrating a routing logic completely decoupled from the application which requested the intents. This proposal has been accepted as an official contribution and is currently under code review from the ONOS developers. Future works include the extension of IMR service to support gRPC (as a more efficient push-based alternative to the pull-based REST API) and multi-path routing (by exploiting OF group tables), the ability of monitoring additional types of ONOS intents with different constraints and alternative routing logics taking into account other metrics such as delay.

References

[1] C. Trois *et al.*, "A survey on sdn programming languages: Toward a taxonomy," *Commun. Surveys Tuts.*, vol. 18, no. 4, Oct. 2016.
[2] J. Medved *et al.*, "Opendaylight: Towards a model-driven sdn controller architecture," in *IEEE WoWMoM*, 2014.
[3] P. Berde *et al.*, "ONOS: towards an open, distributed SDN OS," in *ACM HotSDN*, 2014.
[4] P. Lin *et al.*, "Seamless interworking of SDN and IP," in *ACM SIGCOMM CCR*, 2013.
[5] D. Sanvito *et al.*, "Adaptive Robust Traffic Engineering in Software Defined Networks," in *IFIP Networking*, 2018.
[6] "ONOS Wiki: IMR - Intent Monitor and Reroute service," https://wiki.onosproject.org/x/hoQgAQ.
[7] "ONOS Gerrit: IMR," https://gerrit.onosproject.org/#/c/16234/.
[8] N. McKeown *et al.*, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM CCR*, 2008.
[9] "GitHub repository," https://github.com/ANTLab-polimi/onos/tree/imr-polimi-sdnip-netsoft2k18.
[10] Y. Azar *et al.*, "Optimal oblivious routing in polynomial time," in *ACM STOC*, 2003.
[11] I. Gurobi Optimization, "Gurobi optimizer reference manual," 2016. [Online]. Available: http://www.gurobi.com
[12] "ONOS Wiki: Intent Framework," https://wiki.onosproject.org/x/XgAZ.
[13] B. Lantz *et al.*, "A network in a laptop: Rapid prototyping for software-defined networks," in *ACM Hotnets-IX*, 2010.
[14] A. Lakhina *et al.*, "Structural analysis of network traffic flows," in *ACM SIGMETRICS*, 2004.

[2]each demand is added a uniform relative error $[-\alpha, \alpha]\%$