An Architecture for Task and Traffic Offloading in Edge Computing via Deep Learning

Alessandro Gaballo

April 9, 2018

Supervisor: Flavio Esposito - Saint Louis University
Cosupervisor: Guido Marchetto - Politecnico di Torino

SAINT LOUIS UNIVERSITY

How many times have you delegated a task?

WHO CAN HELP ME?

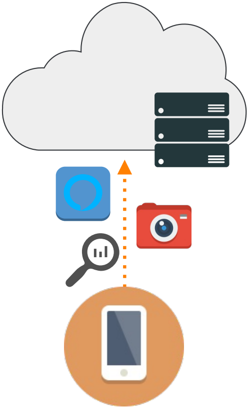
WHO IS THE BEST CHOICE?

What is task offloading and how should it be done?

Task offloading is the process of transferring tasks to another platform.

There is no architecture describing the offloading mechanisms.

Currently used routing strategies, such as OSPF, are performance unaware.



What tools do we have and what can we do?

Recently the ideas of Software-Defined Networking (SDN) and Knowledge-Defined Networking (KDN) [1] have spreaded.

The combination of SDN & KDN represents a powerful tool for network management.

In this work we design a task offloading architecture to address the complexity problem and leverage a network knowledge plane to support performance aware traffic steering.

[1] D. Clark, C. Partridge, J. Ramming, and J. Wroclawski.
A knowledge plane for the internet.
In Proc. of SIGCOMM '03. ACM, New York, NY, USA, 3-10.

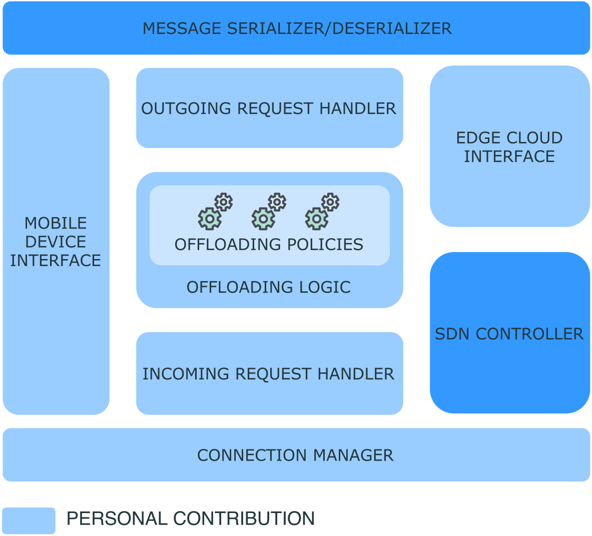
Talk overview

- Offloading Architecture
- Results
- Future work

Talk overview

- Offloading Architecture
  - Architecture
  - Task Offloading Protocol
  - Path Prediction via Deep Learning
- Results
- Future work

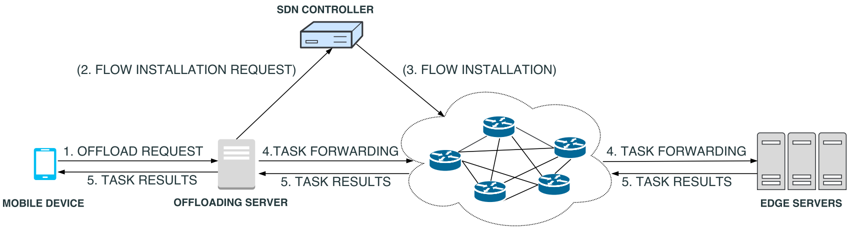
Our contribution: offloading architecture



Talk overview

- Offloading Architecture
    - Architecture
    - Task Offloading Protocol
    - Path Prediction via Deep Learning
- Results
- Future work

Our contribution: task offloading protocol



The protocol allows the client to specify:

- task requirements such as CPU, memory and latency
- offloading logic (e.g. nearest server)

Talk overview

- Offloading Architecture
  - Architecture
  - Task Offloading Protocol
  - Path Prediction via Deep Learning
- Results
- Future work

Our contribution: path prediction via deep learning

Deep learning is a powerful tool for inference tasks

$$\Downarrow$$

IDEA: Routing problem as inference problem

Long Short-Term Memory (LSTM)

LSTM is an evolution of recurrent neural networks (RNNs) capable of memorizing data temporal patterns

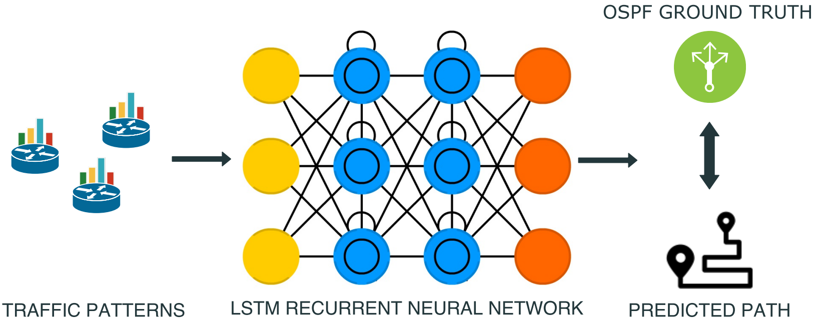
Objective:

Learn how traffic patterns evolve and route accordingly

Learning from who?

LSTM RNNs are a supervised learning method, they require data to learn from.

We use OSPF routing decisions as a ground truth.

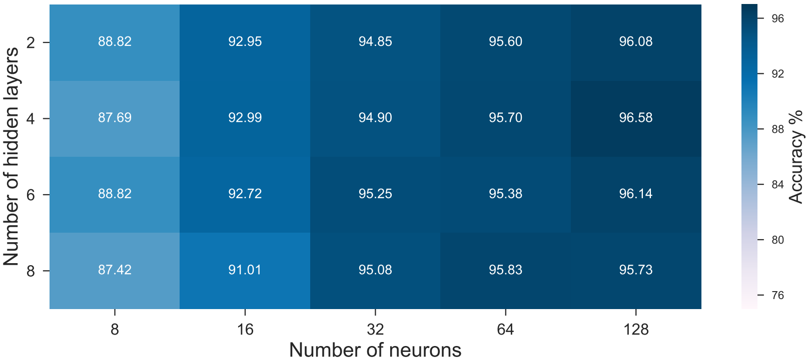


Talk overview

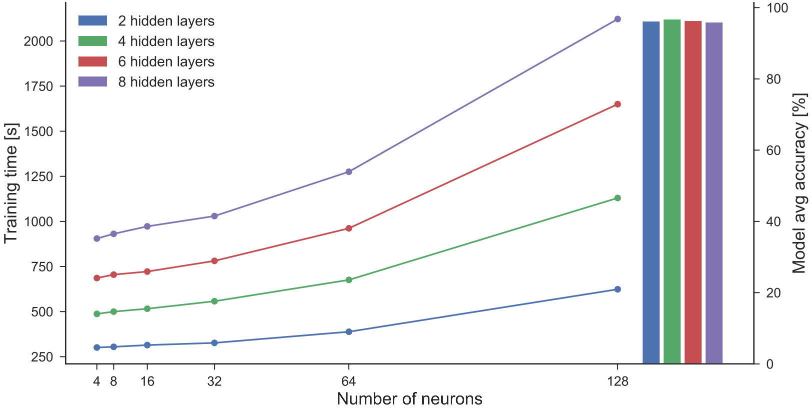
- Offloading Architecture
- Results
  - LSTM architecture
  - Emulating OSPF
  - Performance aware routing
- Future work

Accuracy: more neurons or more layers?

Neurons affect performance more than hidden layers



Finding the trade-off: training time



** Accuracy shown only for models with 128 neurons*

Talk overview

- Offloading Architecture
- Results
  - LSTM architecture
  - Emulating OSPF
  - Performance aware routing
- Future work

The system is able to emulate OSPF

We test the system's ability to behave like OSPF by averaging the performance of all the models on the test set.

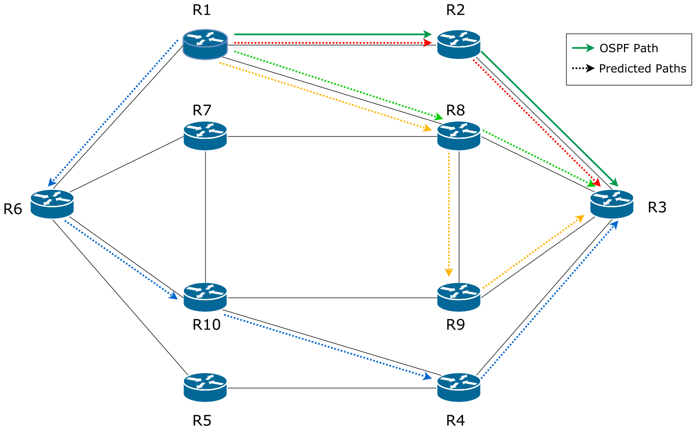
The LSTM achieves an average accuracy of **98.71%** with respect to OSPF.

Talk overview

- Offloading Architecture
- Results
  - LSTM architecture
  - Emulating OSPF
  - Performance aware routing
- Future work

The LSTM system exhibits a dynamic behavior

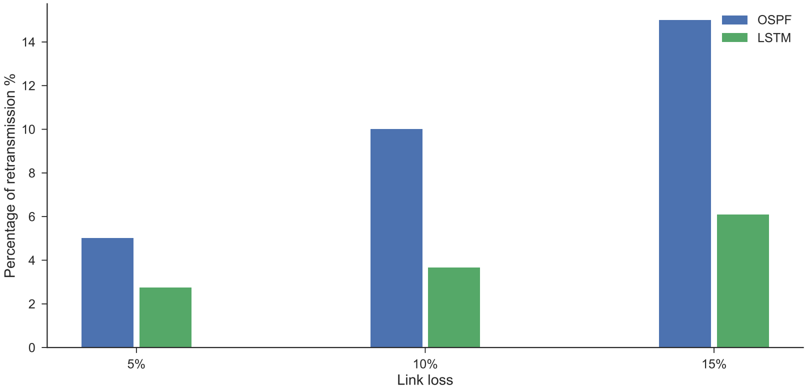
We select a target and analyze how our system behaves differently from OSPF in case of link loss



The LSTM path predictor suggests multiple paths

LSTMs outrun current approaches in terms of retransmission

In case of malfunctioning links, our system has a lower retransmission percentage then traditional routing



OSPF = Open Shortest-Path First, LSTM = Long Short-Term Memory

Talk overview

- Offloading Architecture
- Results
- Future work

Future work

Current results are encouraging, therefore we want to further investigate the problem by:

- getting rid of Mininet emulation environment constraints
- testing the method on larger networks (e.g GENI)
- running the testbed on a more scalable platform (e.g GPU)
- exploring more machine learning techniques
  (e.g reinforcement learning)

Take home messages

- Mobile edge computing is a complex problem

Take home messages

- Mobile edge computing is a complex problem
- We prototyped an architecture for MEC offloading orchestration

Take home messages

- Mobile edge computing is a complex problem
- We prototyped an architecture for MEC offloading orchestration
- We developed a machine learning-based, performance aware routing strategy that improves on classic iBGP mechanisms

An Architecture for Task and Traffic Offloading in Edge Computing via Deep Learning

Thank you for the attention

Alessandro Gaballo

Protocol 1

```
message OffloadRequest {
    message Requirements {
        enum Latency {
            URGENT = 0;
            STANDARD = 1;
            LOOSE = 2;
        }

        float cpu = 1;
        int32 memory = 2;
        Latency latency = 3;
    }
```

Protocol 2

```
enum Type {
    LAMBDA = 0;
    STANDARD = 1;
}

message Task {
    message TaskWrapper {
        enum WrapperType {
            JAR = 0;
            EGG = 1;
        }
```

Protocol 3

```
        string name = 1;
        WrapperType type = 2;
        bytes task = 3;
    }

    oneof task_location {
        string task_id = 1;
        TaskWrapper wrapper = 2;
    }
}
```

Protocol 4

```
    Requirements requirements = 1;
    Type type = 2;
    Task task = 3;

}

message Response{
    enum Result {
        OK = 0;
        INVALID_MSG_SIZE = 1;
        INVALID_REQUEST = 2;
    }
```

Protocol 5

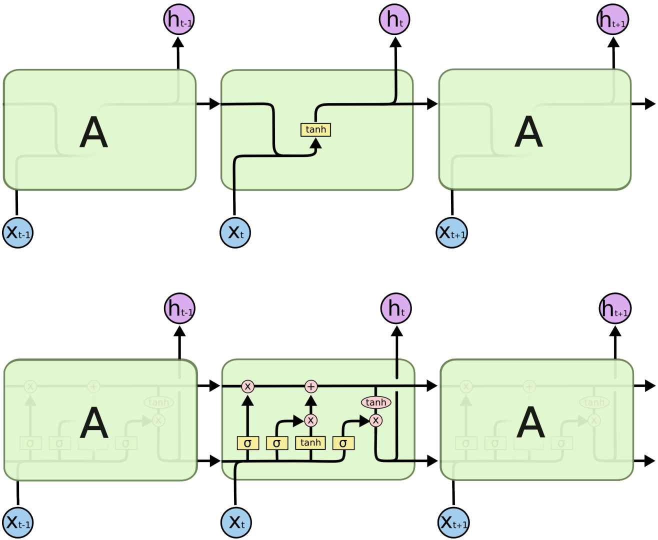
```
    Result result = 1;
    string msg = 2;
}

message Message{
    enum Type {
        OFFLOAD_REQUEST = 0;
        RESPONSE = 1;
        TASK = 2;
    }
```

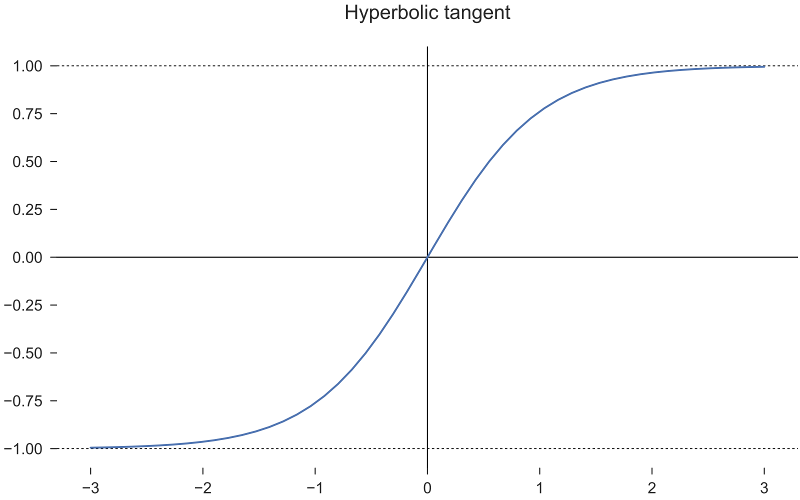
Protocol 6

```
    Type type = 1;
    oneof msg_type {
        OffloadRequest off_req = 2;
        OffloadRequest.Task task = 3;
        Response response = 4;
    }
}
```

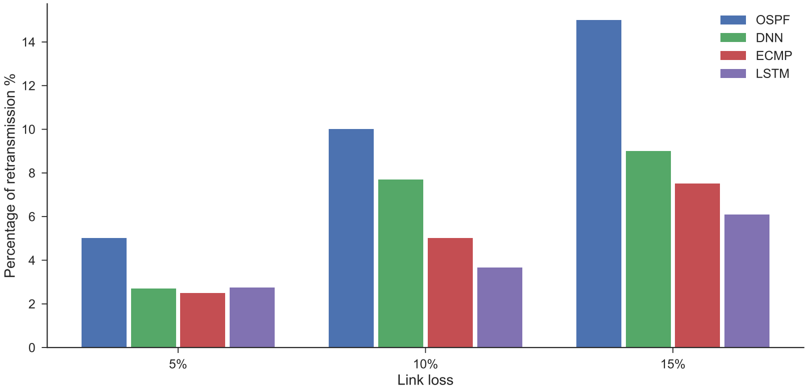
Recurrent Neural Network vs Long Short Term Memory



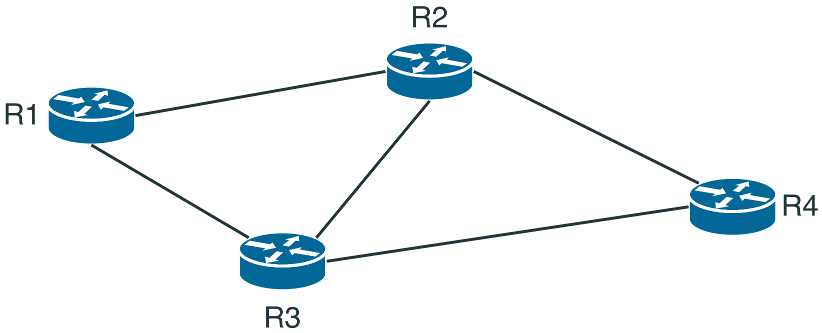
LSTM cells - activation function



Comparing with other techniques



Example: computing the path from R1 to R4



Step 1 - model = R1-R4 –> next hop = R2

Step 2 - model = R2-R4 –> next hop = R3

Step 3 - model = R3-R4 –> next hop = R4

Computed path: R1 - R2 - R3 - R4

Dataset generation

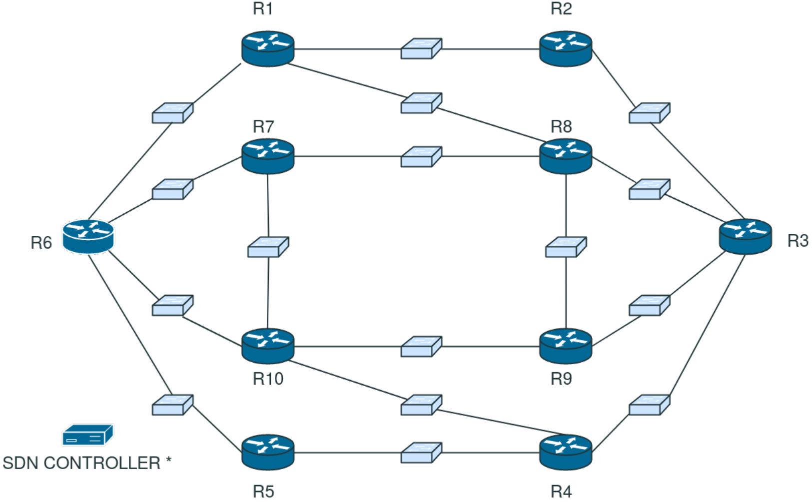
To train our model we need:

- network topology
- routing algorithm
- packet counter

We could not find any public dataset suited to our needs so we create our own.

Network topology

We create this topology using MiniNeXt



** All switches are connected to the SDN controller*

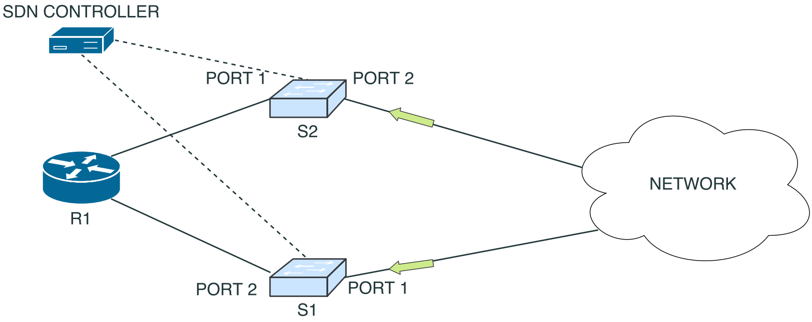
Routing algorithm

To run routing algorithms on MiniNeXt nodes we use Quagga. Quagga is a routing suite providing different routing algorithms (e.g OSPF, IS-IS, RIP).

We choose Open Shortest Path First (OSPF) because of its wide adoption as iBGP.

Packet counter

The SDN controller –Ryu– is responsible of retrieving the packet count.

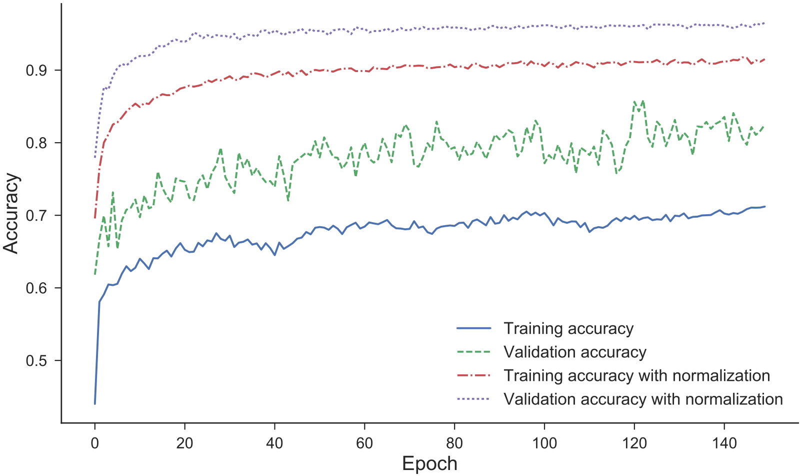


Dataset generation steps

For an arbitrary number of times:

1. Initialize the topology with different link speed
2. Simulate traffic between routers
3. Save packet count and routing tables
4. Stop the traffic simulation and tear down the network
5. Back to step 1

Tuning the network: input normalization



** Accuracy shown only for models with 128 neurons*

Our prediction model

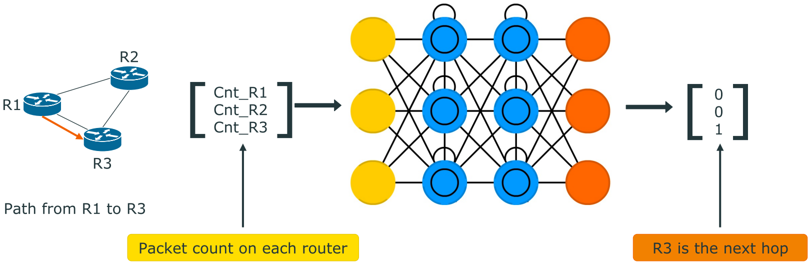
To use a LSTM we must define the model input and output.

Input:

Incoming packets count on each router

Output:

One-hot encoded vector with the next hop in the path



Limitations

Testbed:

- The functionalities of Mininet are limited
- Higher link loss decreases the prediction efficiency

Computation:

- The number of models to train is big
- The trained models occupy GBs of memory