SRv6-FEC: Bringing Forward Erasure Correction to IPv6 Segment Routing

Louis Navarre, François Michel, Olivier Bonaventure
navarre.louis@student.uclouvain.be
Université catholique de Louvain, Louvain-la-Neuve, Belgium

Motivation

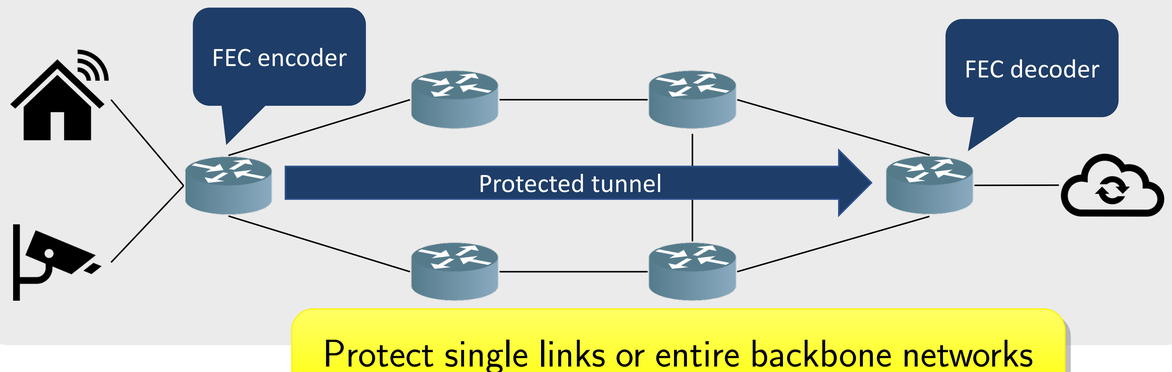
Forward Erasure Correction (FEC) provides **recovering capabilities** in **lossy networks**:

- Faster than pure packet retransmissions (reliable transfer)
- Heavily used for real-time applications

Retransmission mechanisms and FEC are **costly** for **resource-constrained** devices (e.g. Internet of Things (IoT) devices)

⇒ Implement a FEC mechanism as **a service in the network**, **transparently** for the devices

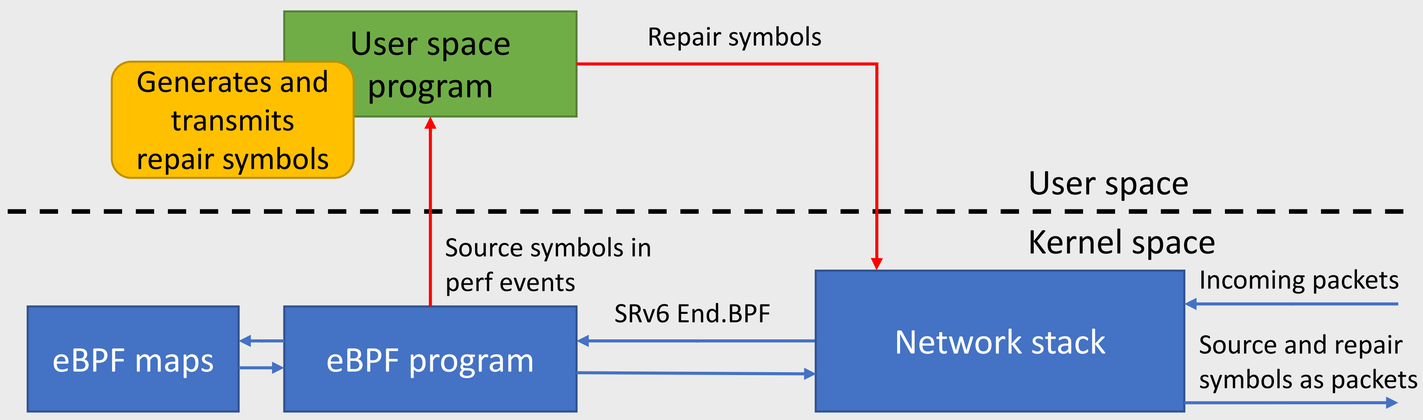
Deployment architecture



Protect single links or entire backbone networks

Implementation overview

Prototype implementation: `https://github.com/louisna/FEC-SRv6-libbpf`



Leverage **IPv6 Segment Routing (SRv6)** in the Linux kernel:

- Forward Erasure Correction plugin as an **eBPF program**
- The incoming packets trigger the **End.BPF** SRv6 action with the corresponding Segment ID in their Segment Routing Header
- IPv6 **Type-Length-Value (TLV)** fields carry FEC-related information such as the **Source FEC Payload ID**

⚠ FEC at the network layer ⇒ protect routing information about the packets (source/destination address, . . .)

Current limitations

The support of eBPF in Linux constraints us:

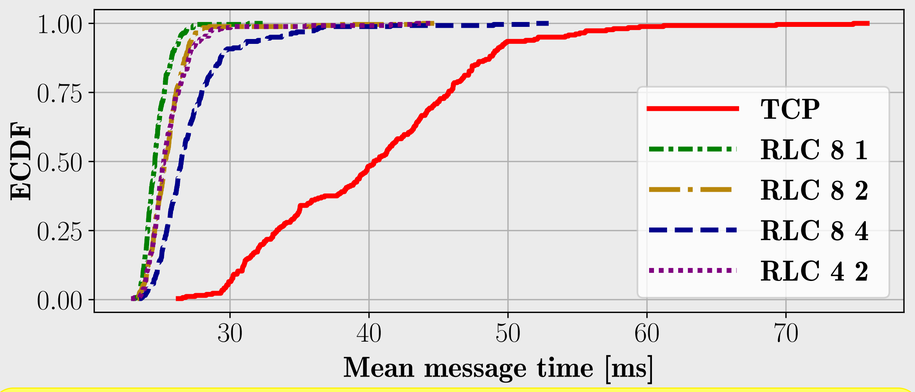
- Protection of IPv6 packets of at most 512 bytes
- Bottleneck user/kernel space communication:
  - Create repair symbols using RLC
  - Create and send new packets

Possible **improvement**: **extend** the eBPF support in the Linux kernel with new **helpers** and modified limits (e.g. higher instruction limit)

Evaluation over the MQTT protocol

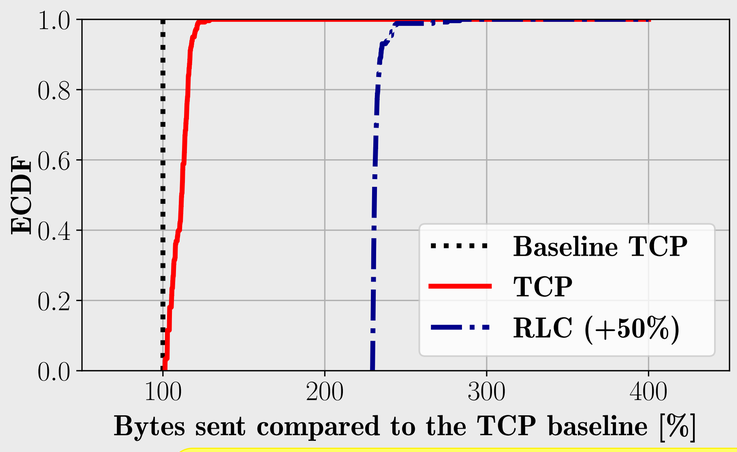
Experimental methodology:

- MQTT: IoT protocol over **TCP**
- Simulate **losses** with a parametrized and **reproducible** two-states Markov model using the **experimental design**
- Measure the **mean message time** to send an MQTT message (100 bytes) to the server and get an MQTT ack

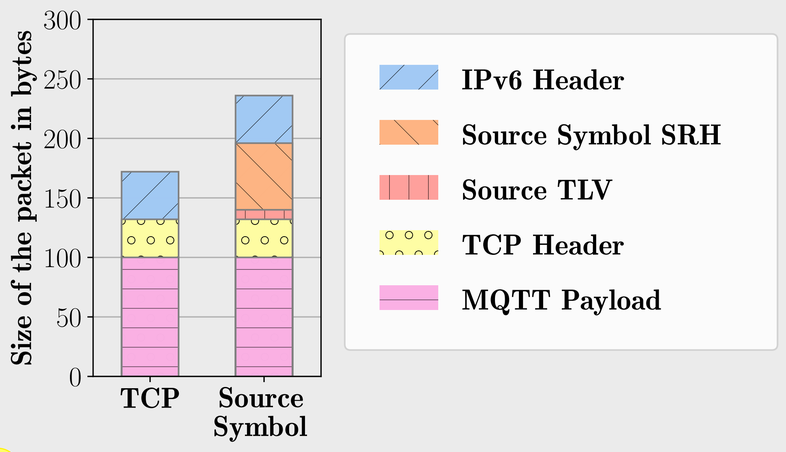


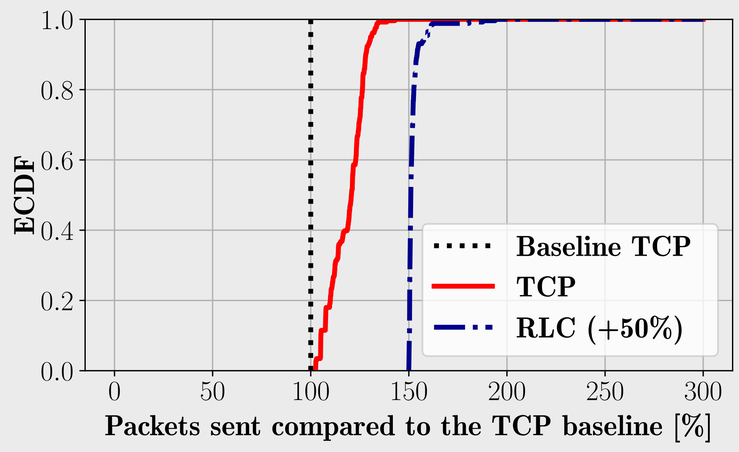
Losses recovery ⇒ decrease the number of retransmissions

Bandwidth usage overhead on the protected link



220 % bytes overhead but a $\frac{2}{3}$ code rate





Ignoring packet sizes, only 50 % overhead

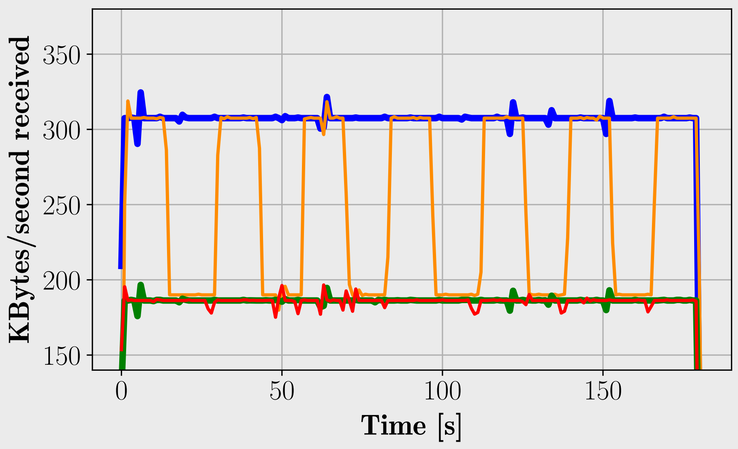
Decreasing the plugin overhead with a Controller

Dynamically (de)activate repair symbols generation:

- The FEC decoder regularly sends a feedback with the **measured percentage loss**
- The FEC encoder uses a **threshold function** and the feedback to (de)activate redundancy generation

⇒ Stop redundancy generation/transmission when the network is in good condition

Parameters of the controller: feedback sending rate and threshold value of the decision function



- Analyze a UDP client for 180 seconds
- Iteratively add/remove losses and analyze the impact of the Controller
- # received bytes on the protected link without and with the controller: the overhead decreases
- # received bytes on the server without and with the controller: only small losses occur

Losses triggering the redundancy generation again cannot be recovered without redundancy