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

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

François Michel
Université catholique de Louvain
Louvain-la-Neuve, Belgium
francois.michel@uclouvain.be

Olivier Bonaventure
Université catholique de Louvain
Louvain-la-Neuve, Belgium
olivier.bonaventure@uclouvain.be

ABSTRACT

IPv6 Segment Routing (SRv6) is a recent implementation of the source routing paradigm in IPv6 network. A programmability framework has recently been added to SRv6 and enables it to support diverse use-cases. We leverage this support to design, implement and assess a Forward Erasure Correction (FEC) technique that transparently protects IPv6 packets. We implement our encoders and decoders using eBPF on the Linux kernel and evaluate the benefits that they bring with IoT devices.

CCS CONCEPTS

• **Networks** → **Programmable networks**; **Error detection and error correction**.

KEYWORDS

IPv6 Segment Routing, Forward Erasure Correction, extended Berkeley Packet Filter, Internet of Things

ACM Reference Format:
Louis Navarre, François Michel, and Olivier Bonaventure. 2021. SRv6-FEC: Bringing Forward Erasure Correction to IPv6 Segment Routing. In *SIGCOMM '21 Poster and Demo Sessions (SIGCOMM '21 Demos and Posters), August 23–27, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3472716.3472863

1 INTRODUCTION

Most protocols rely on retransmissions to recover from packet losses which can be caused by random transmission errors or congestion. Retransmissions work well, but they have two important drawbacks. First, the source must store unacknowledged data to be able to retransmit it. This can be a problem for applications running on embedded systems such as some Internet of Things (IoT) devices. Second, retransmissions inevitably increase the tail latency for delay-sensitive applications.

Different types of network coding and Forward Erasure Correction (FEC) techniques have been proposed to cope with this problem [1, 15, 17]. However, implementing these techniques on IoT devices, for example at the transport layer (e.g., TCP), can be challenging given their CPU cost, notably when recovering from errors.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
SIGCOMM '21 Demos and Posters, August 23–27, 2021, Virtual Event, USA
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8629-6/21/08…$15.00
https://doi.org/10.1145/3472716.3472863

In this paper, we explore a different approach, performing the computation inside the network. With IPv6 Segment Routing [10] (SRv6) and its network programmability [9] in particular, it becomes possible for SRv6 routers to expose functions that they can apply when forwarding packets. We implement encode and decode functions on SRv6 routers such that IoT devices send their packets through an encode function on one router and a decode function on a distant router. Each packet sent by the IoT device includes in its SRv6 Header the addresses of the intermediate routers with the functions that they need to execute.

Thanks to these SRv6 functions, an automatic and protected tunnel is created between the upstream and the downstream routers to protect packets from losses. We can deploy this tunnel anywhere in the network. For example, we could attach the encode function at the ingress router of a backbone network and the decode function at its egress, right before the IoT server, transparently for the IoT device and the backbone network. The routers use XOR-based FEC or Random Linear Codes (RLC) to enable the downstream router to recover from losses with minimal impact on latency.

We briefly describe our FEC Framework and its implementation using eBPF on Linux in section 2 and evaluate its performance in section 3. Our prototype code is available [16].

2 FEC WITH SRV6 NETWORK PROGRAMMING

This section presents our Forward Erasure Correction plugin at the network layer, leveraging IPv6 Segment Routing. Our plugin is specific to the Linux implementation of SRv6 since we rely on its BPF endpoint [22]. However, the approach will work under any implementation of SRv6 following RFC8754 [10]. This section presents our solution and the current technical limitations of the eBPF support in Linux 5.4 temporarily limiting the domain of application of our solution. We base our FEC Frameworks respectively on RFC6363 [20] and RFC8680 [18] for Block and Convolutional Codes. The FEC Framework provides a common abstraction for the FEC Schemes, i.e., the specific coding algorithms.

2.1 Source and repair symbols

The source symbols correspond to the data that are protected by FEC. The repair symbols carry the redundancy payload used to recover the lost source symbols. To inform the decoder of the presence of a source symbol, we leverage the Type-Length-Value (TLV) options of the extended headers, specified by RFC8200 [6]. This source TLV carries a 32-bits *Source FEC Payload ID (SFPID)* identifying the source symbol. Any packet containing a Segment Routing Header (SRH) with an SFPID will be protected with FEC.

We transmit the repair symbols in dedicated packets. Information needed for the FEC framework to recover packets such as the *Repair FEC Information* or the window/block size of the Framework are also sent in a Segment Routing TLV.

2.2 Challenges of FEC in IPv6

Recent works about FEC focus on end-to-end protection [4, 5, 15]. Yet, at the internet layer, we must also protect routing information, as we potentially aggregate multiple flows with different sources and destinations. FEC requires the source symbols to be the same between the encoder and the decoder, otherwise the data will be incorrectly recovered. RFC8280 [6] and RFC8754 [10] state that multiple fields of the IPv6 and SRv6 headers can be changed by intermediate routers of the network, such as the *Traffic Class* or the *Destination Address* using IPv6 Segment Routing. Our design takes into account these varying fields and provides an algorithm to recover for example the true *Destination Address* of a recovered packet based on the *Segment List* of the SRH. However, we do not attempt to recover varying fields of other extended headers.

2.3 Technical limitations of eBPF

Due to technical limitations of the eBPF support inside Linux 5.4, we restrict the plugin to the protection of small packets (at most 512 bytes). However, the IoT traffic is generally composed of small packets [21] and can still benefit from our prototype. We also perform specific tasks in user space instead of within the eBPF program, such as the RLC encoding/decoding or the repair symbol transmission. We leverage the eBPF architecture to communicate between the kernel and user space. Modifying the kernel or implementing kernel modules could overcome these limitations.

3 EVALUATION

We assess our plugin in a lossy environment to evaluate the benefits of FEC. We use MQTT-Benchmark [13], a `Go` implementation of the MQTT protocol [11] running over TCP, to simulate IoT clients pushing messages to a Mosquitto [14] broker. We use a topology where the clients and the broker are separated by 3 routers, respectively to add an SRH on the packets, and run the encoder and the decoder plugins. Our evaluation is performed using the IPMininet [3] environment, an extension of Mininet [8] supporting SRv6.

IPMininet provides tools to emulate the losses. However, we implement our own two-states Markov model to emulate losses into the network [7] using eBPF. The losses are generated using a seed, making the results reproducible for deeper analysis. In the `PASS` state, the packet is accepted and is forwarded; in the `DROP` state, the model erases the data, simulating a loss. We use a probability of k to remain in the `PASS` state, and of d to stay in the `DROP` state. This model allows emulating burst losses of length $\frac{1}{1-d}$ on average.

We use the experimental design approach [12] to assess our solution. This method samples the experiment's parameters values from a defined range and reports the Experimental Cumulative Distribution Function (ECDF) associated with the results. It allows exploring numerous scenarios that could happen in real-life communication. We uniformly sample the space defined by $k \in [0.9, 0.99]$ and $d \in [0, 0.5]$ with 260 points. A test consists of 10 MQTT clients pushing 500 messages of 100 bytes each to the broker. We also add a 10 ms delay on the link between the encoder and the decoder. As IoT devices do not send much traffic [2], we analyze the mean message time (MMT) for a client to send a message to the broker to see the impact of the plugin on retransmissions, instead of the throughput. Due to space constraints, we only evaluate here the Convolutional RLC FEC Scheme, as it provides a lower delay to recover lost symbols than Block codes [19]. The *window step* gives the shifting value of the source symbols window after each iteration.

Figure 1 shows the results obtained by experimental design with our setup. We see that we reduce the MMT using FEC. Without the plugin, more than 50% of the experiments have an MMT longer than 40 ms, where the baseline - without the plugin nor losses - is 23 ms. On the contrary, depending on the RLC parameters, the maximum MMT lies in 99% of the experiments between 34 ms and 40 ms. Moreover, this quantity is lower than 30 ms for 90-99% of the experiments depending on the RLC parameters. We see that FEC recovers numerous lost packets, hence decreasing the need for time-consuming TCP retransmissions. Figure 1 also shows the impact of the window size and step on the results. As expected, we provide better recovering capabilities when the window step decreases: we generate more repair symbols for the same set of source symbols, thus increasing the probability of recovering longer burst losses. The dotted line illustrates that reducing the window size from 8 to 4 slightly decreases the performance, but its impact is lower than the window step. Indeed, increasing the window size adds more overhead, and the generated burst losses are not long enough to see the positive impact on the recovering capabilities.

Using FEC, we increase the link utilization. With `RLC_4_2`, we send 125% more bytes compared to the baseline - without the plugin nor losses - but we only see an increase of 50% in terms of sent packets - corresponding to the $\frac{2}{3}$ code rate. This difference is due to the Segment Routing Header overhead, which would be amortized by protecting packets larger than 512 bytes. However, in 90-95% of the experiments, we observed a stable link utilization, meaning that fewer packets are retransmitted by the TCP client. It shows the benefits of FEC where the losses are hidden to the sender, especially IoT devices with constrained resources.

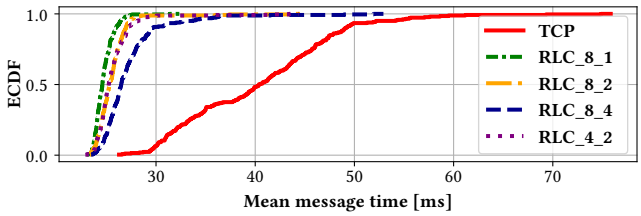

Figure 1: MMT between an MQTT client and the Mosquitto broker, with SRv6-FEC compared to TCP only. `RLC_X_Y` is the RLC FEC Scheme with a window size of `X` and of step of `Y`.

4 CONCLUSION

In this paper, we have shown that using IPv6 Segment Routing it becomes possible to provide in-network FEC services to protect delay-sensitive applications that are unable to implement them. Our evaluation shows that this reduces the latency of IoT applications such as MQTT when packet losses occur.

REFERENCES

[1] Meinolf Blawat, Klaus Gaedke, Ingo Huetter, Xiao-Ming Chen, Brian Turczyk, Samuel Inverso, Benjamin W Pruitt, and George M Church. 2016. Forward error correction for DNA data storage. *Procedia Computer Science* 80 (2016), 1011–1022.

[2] Yuang Chen and Thomas Kunz. 2016. Performance evaluation of IoT protocols under a constrained wireless access network. In *2016 International Conference on Selected Topics in Mobile & Wireless Networking (MoWNeT)*. IEEE, 1–7.

[3] CNP3. 2017. IPMininet. https://github.com/cnp3/ipmininet. Accessed: 2021-05-16.

[4] Alejandro Cohen, Derya Malak, Vered Bar Bracha, and Muriel Médard. 2020. Adaptive causal network coding with feedback. *IEEE Transactions on Communications* 68, 7 (2020), 4325–4341.

[5] Quentin De Coninck, François Michel, Maxime Piraux, Florentin Rochet, Thomas Given-Wilson, Axel Legay, Olivier Pereira, and Olivier Bonaventure. 2019. Pluginizing quic. In *Proceedings of the ACM Special Interest Group on Data Communication*. 59–74.

[6] Dr. Steve E. Deering and Bob Hinden. 2017. Internet Protocol, Version 6 (IPv6) Specification. RFC 8200. https://doi.org/10.17487/RFC8200

[7] Edwin O Elliott. 1963. Estimates of error rates for codes on burst-noise channels. *The Bell System Technical Journal* 42, 5 (1963), 1977–1997.

[8] Bob Lantz *et al.* 2009. Mininet. https://http://mininet.org. Accessed: 2021-05-16.

[9] Clarence Filsfils, Pablo Camarillo, John Leddy, Daniel Voyer, Satoru Matsushima, and Zhenbin Li. 2021. Segment Routing over IPv6 (SRv6) Network Programming. RFC 8986. https://doi.org/10.17487/RFC8986

[10] Clarence Filsfils, Darren Dukes, Stefano Previdi, John Leddy, Satoru Matsushima, and Daniel Voyer. 2020. IPv6 Segment Routing Header (SRH). RFC 8754. https://doi.org/10.17487/RFC8754

[11] Urs Hunkeler, Hong Linh Truong, and Andy Stanford-Clark. 2008. MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks. In *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08)*. IEEE, 791–798.

[12] Roger E Kirk. 2012. Experimental design. *Handbook of Psychology, Second Edition* 2 (2012).

[13] Alexandr Krylovskiy. 2015. MQTT benchmarking tool. https://github.com/krylovsk/mqtt-benchmark. Accessed: 2021-04-30.

[14] Roger A Light. 2017. Mosquitto: server and client implementation of the MQTT protocol. *Journal of Open Source Software* 2, 13 (2017), 265.

[15] François Michel, Quentin De Coninck, and Olivier Bonaventure. 2019. QUIC-FEC: Bringing the benefits of Forward Erasure Correction to QUIC. In *2019 IFIP Networking Conference (IFIP Networking)*. IEEE, 1–9.

[16] Louis Navarre. 2021. SRv6-FEC: prototype code (GitHub). https://github.com/louisna/FEC-SRv6-libbpf.git. Accessed: 2021-06-08.

[17] Pouya Ostovari and Jie Wu. 2015. Reliable broadcast with joint forward error correction and erasure codes in wireless communication networks. In *2015 IEEE 12th International Conference on Mobile Ad Hoc and Sensor Systems*. IEEE, 324–332.

[18] Vincent Roca and Ali C. Begen. 2020. Forward Error Correction (FEC) Framework Extension to Sliding Window Codes. RFC 8680. https://doi.org/10.17487/RFC8680

[19] Vincent Roca, Belkacem Teibi, Christophe Burdinat, Tuan Tran, and Cédric Thienot. 2017. Less latency and better protection with al-fec sliding window codes: A robust multimedia cbr broadcast case study. In *2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, 1–8.

[20] Vincent Roca, Mark Watson, and Ali C. Begen. 2011. Forward Error Correction (FEC) Framework. RFC 6363. https://doi.org/10.17487/RFC6363

[21] Arunan Sivanathan, Daniel Sherratt, Hassan Habibi Gharakheili, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. 2017. Characterizing and classifying IoT traffic in smart cities and campuses. In *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 559–564.

[22] Mathieu Xhonneux, Fabien Duchene, and Olivier Bonaventure. 2018. Leveraging ebpf for programmable network functions with ipv6 segment routing. In *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*. 67–72.