

eFRAC: eBPF Framework for Resource-Aware Compression

mattia fiore

October 2025

Small abstract

Problem Statement. Wireless links exhibit strong bandwidth fluctuations and sudden latency spikes due to interference, mobility, and hidden terminal effects. Existing mechanisms either compress payloads or adapt traffic rate, but none address the need for fast, fine-grained reaction at the network dataplane. Header compression can reduce protocol overhead, but current approaches require static configuration and cannot respond dynamically to short congestion events.

Research Gap. Header compression schemes (e.g., ROHC) are protocol-specific and must be configured a priori. Payload compression techniques (e.g., delta encoding, semantic/text compression) operate at the application layer and react too slowly to transient congestion. No existing system is able to orchestrate multiple compression techniques across layers and protocols, nor to dynamically decide when header or payload compression should be applied based on real-time wireless link conditions.

Contribution. We propose a framework that monitors packet sojourn time inside the wireless dataplane and dynamically selects the optimal compression strategy (header or payload). The decision logic executes in eBPF for sub-millisecond reaction, enabling cross-layer, multi-protocol adaptive compression in response to instantaneous congestion signals.

1 Motivation and Related Work

Leggi questo: [1] They do they same stuff as ours but they dont do header compression.

Leggi questo: [2] They do a static compression for both header and payload. Important for Wireless background and testing methodologies.

Figure 2 successfully proves our hypothesis. The X-axis represents simulation time, and the Y-axis shows the queue level in MBytes. The red dashed line at 0.05 MB is the total queue capacity, where packets begin to drop. The solid

lines show the mean performance over 50 runs, while the shaded areas represent the 5%-95% confidence intervals.

We observe three distinct phases. In the first phase (0–8 seconds), all strategies perform identically, as no optimizations are active until the queue reaches a 20% threshold (0.01 MB). In the second phase (8–40 seconds), the optimizations activate, and the lines diverge. The Baseline (Blue) fills rapidly, hitting capacity around 38 seconds. Enabling HC (Orange) slows this process, buying an additional 7 seconds. The HC + DPI strategy (Green) is the most effective, as it both reduces packet size (HC) and drops redundant packets (DPI), delaying saturation until approximately 50 seconds. Finally, in the saturation phase (after 50 seconds), all scenarios eventually fill the queue, confirming the goal is to *delay*, not prevent, saturation. The HC + DPI strategy is the clear winner in achieving this delay.

2 Plan for Thesis

What must be implemented:

- eBPF bandwidth estimation
- eBPF signaling
- eBPF policy for applying either header compression or payload compression

What should be taken from other repositories:

- header compression algorithms like RoHCv2 and some other.
- payload compression algorithms one audio to text and the other delta encoding on coat. This can be simulated and can be taken from pre-prepared traces.

What could be added if there is more time:

- moving header compression logic in the kernel.
- create deep packet inspections for dropping repeated packets. With the delta very small from the previous (for CoT or other types of services similar)
- implementing the LLM transcription

2.1 Roadmap

First month: Bandwidth estimation (up to 15 Nov) Search for a bandwidth estimation approach that might work in the wireless environment and can work on eBPF. Simulate the network with Mininet-Wifi, change the network bandwidth, and see how the bandwidth estimation react.

Second month: Sender Logic implementation (up to 15 Dec) Generate a dataset for simulating payload compression. For example, an audio message and its text version, in order to simulate a model compressing audio into text. Or CoT simulates positions and creates delta encoding for the data. Write the code to make a host send data at a specific frequency, and the eBPF hook to make it change strategy.

Third month: Add header compression (up to 15 Jan) Add some header compression schemes in the system so and make them work in the userspace when signaled.

Fourth month: Testing policy (up to 15 Feb) Study what policy to apply since the dynamic of the compression is different. Header compression is faster to react but can drop more slowly. Payload compression reacts more slowly, but the drop can be significant.

Fifth month: Testing in software, possibly also hardware. (up to 15 March) Inject the code on hardware devices (Raspberry + antennas) and obtain real measurements. Get plots and significant measurements. Plot ideas:

- How the system is influenced by different policies: graph of bandwidth of the link/ time and the data rate from the sender.
- How good the bandwidth estimation is: Telemetry measurement/Bandwidth
- How many packets are we saving?: Against no compression, payload compression, and payload compression + header compression.

Sixth month: write thesis (up to 15 Apr) Write the thesis.

References

- [1] C. Liu, J. Paparrizos, and A. J. Elmore, “Adaedge: A dynamic compression selection framework for resource constrained devices,” in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pp. 1506–1519, 2024.
- [2] I. Kallala, T. Watteyne, Q. Lampin, M. Dumay, S. Coutant, C. Adjih, and P. Muhlethaler, “(demo) joint automated header and payload compression in constrained networks,” in *2024 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1–3, 2024.

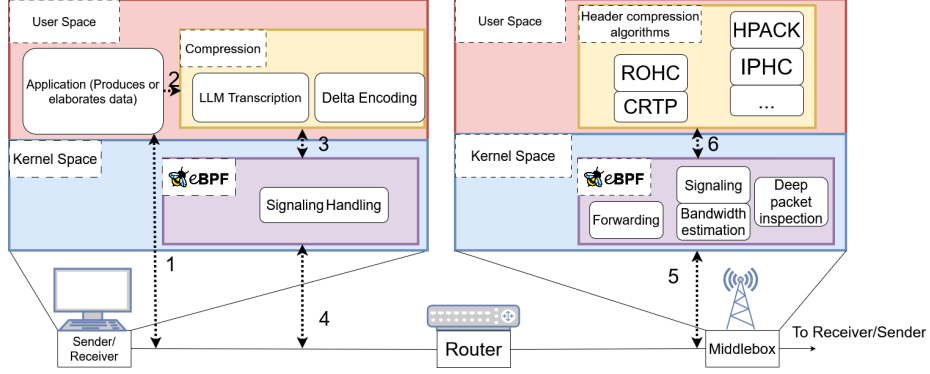


Figure 1: Architecture of the eFRAC system, illustrating the interaction between user-space components and the eBPF-enabled kernel, the dashed lines represent the typical flow of the system with the functionalities implemented by eBPF. 1. (Normal Operation) The applications running in the user space of a terminal produce data and need to send it to at least another terminal over the network, they can directly send it over the network while it's not under attack. From the receiver's perspective it receives it. 2 and 3. (Degraded Flow) When the network is under attack or operating in conditions that reduce its available bandwidth it can compress the quantity of data it sends over the network through Delta Encoding or LLM Transcription (for images or videos), when reaching the receiver that data is transcribed back to the original message without losing key informations by achieving a high semantic similarity. 4. Data is sent or received over the network, when received it must transit through decompression (if we're operating with a low bandwidth) before being sent to the application. Furthermore we also receive out of band dedicated signaling packets, that get handled by an eBPF program which uses the network data received from the signaling to adapt its encoding behavior. 5. In the middlebox's kernel we host eBPF programs that operate Signaling, Forwarding, Bandwidth estimation and Deep packet inspection services over the network. Produced control plane packets are sent over the nodes of the network. 6. When the network is under attack the packets are forwarded to the user space to have their headers compressed depending on what kind of packet they are.

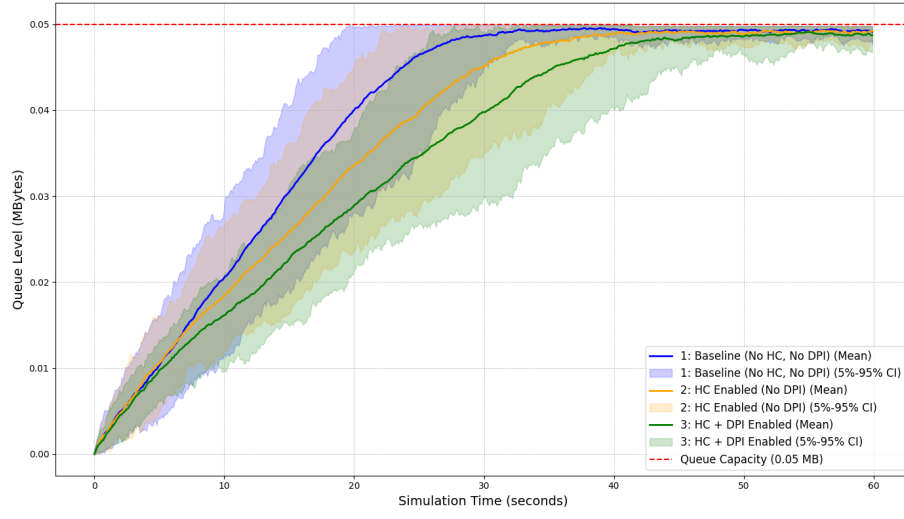


Figure 2: Simulation of queue level (MBytes) over time (seconds) comparing three strategies: Baseline (Blue), HC Enabled (Orange), and HC + DPI Enabled (Green). The combined HC + DPI strategy is the most effective, significantly delaying queue saturation (0.05 MB capacity).