A High-Speed Robust Tunnel using Forward Erasure Correction in Segment Routing

Louis Navarre, Francois Michel, Tom Barbette
UCLouvain (*ICTEAM*), Louvain-la-Neuve, Belgium
{louis.navarre, francois.michel, tom.barbette}@uclouvain.be

***Abstract*—Low-latency applications drive an increasing number of modern applications. Latency depends on factors such as link layer technologies and how higher-layer protocols cope with transmission errors and packet losses. Most transport protocols rely exclusively on retransmissions to cope with losses with minimal overhead but potentially large tail latency. This paper leverages network coding to propose the high-speed robust tunnel (HIRT), providing timely packet delivery to any application independently of the transport protocol. A network code recovers lost data without requiring time-consuming retransmissions by adding redundancy packets, thereby slightly increasing the bandwidth usage to reduce the tail latency. Our algorithm dynamically adapts the rate of redundancy packets by measuring the network loss patterns. We implement HIRT using IPv6 Segment Routing (SRv6). We suggest an efficient software implementation and demonstrate on CloudLab that our solution can protect traffic at high speeds (>50 Gbps) on standard servers even when facing severe packet losses in the network. We evaluate HIRT with HTTP over TCP/QUIC, and file system benchmarks over a real network with losses, Starlink. HIRT reduces the tail latency of short HTTP requests by 2× and the mean request completion time of longer requests by up to 20%. HIRT also decreases the tail latency of NFS requests by up to 20%.**

***Index Terms*—Forward Erasure Correction, Network Layer Coding, High-speed networking**

I. Introduction

An increasing number of applications require low latency, such as automotive use-cases, augmented reality, online gaming, and future developments like telemedicine. In these contexts, a slow response can result in poor user experiences at best, or lead to serious casualties at worst. An important consideration is that the latency of the slowest responses—referred to as tail latency—is also a key matter. Studies have shown that accepting a slower $1\,\%$ response for web services can seriously damage market shares [1], [2]. For instance, this requirement led 5G networks to be built with low latency as a key feature [3]–[5].

Standard Selective-Repeat ARQ (SR-ARQ) loss recovery mechanisms used by common transport protocols, such as TCP and QUIC [6], are not well-designed to provide low latency in lossy networks. First, time-consuming retransmissions can take *several* round-trip times (RTTs) [7]. In case of high end-to-end delays, the added retransmission time can severely impact the application and, consequently, the quality of experience.

Second, retransmission mechanisms require resources on the sender and the receiver. The sender must manage retransmission timers and both endpoints need to maintain buffers, either to store out-of-order data or for retransmissions when losses occur. This can be problematic for devices with limited memory and computing power such as embedded systems. Long flow completion times also consume the battery life of portable devices by keeping the network chips awake [8].

Finally, on networks exposing *non-congestion* induced losses such as Starlink [9] and 5G Fixed Wireless Access (FWA), loss-based congestion control algorithms may inadequately *decrease* the transmission rate at the source. Network service providers know that packet losses can negatively impact the performance perceived by their interests. To cope with packet losses in access networks, some network operators have deployed middleboxes that intercept TCP connections to improve their performance [10]. However, with the growth of QUIC [6] and the utilization of encrypted tunnels for VPNs, network operators cannot deploy middleboxes that intercept all these connections anymore.

Forward Erasure Correction (FEC) can be used to alleviate the cost of retransmissions [11]–[16]. It adds redundancy packets *a priori*, in contrast to ARQ mechanisms [17] which act *a posteriori*, retransmitting packets after some feedback is received from the receiver. With FEC, the encoder interleaves redundancy packets with the data it transmits. This extra data, alongside the correctly received packets, enables the decoder to recover lost packets. The added delay is significantly reduced as it corresponds to the arrival of the redundancy packets which is independent of the RTT. FEC is more memory and CPU consuming compared to classical SR-ARQ mechanisms on the computing nodes [18].

Forward Erasure Correction methods are usually applied at the datalink [19]–[21] and the transport layer [12], [18], [22]–[29], respectively to protect a single link or an entire path between two end-hosts. The end-to-end approach provided by transport-layer FEC (TL-FEC) has both benefits and drawbacks. The FEC algorithm can be application-aware, protecting only some application objects [16]. However, it cannot make many assumptions beforehand about the path that the packets follow, such as the delay or the reliability of the routers. Additionally, at the transport-layer, a FEC extension requires standardization and upgrading the implementations on all (potentially heterogeneous) end-hosts, which will require many years or more [30].

This paper leverages the fact that many network operators are replacing their core IP or MPLS network with Segment Routing [31]. In particular, the deployment of IPv6 Segment

979-8-3503-5171-2/24/$31.00 ©2024 IEEE

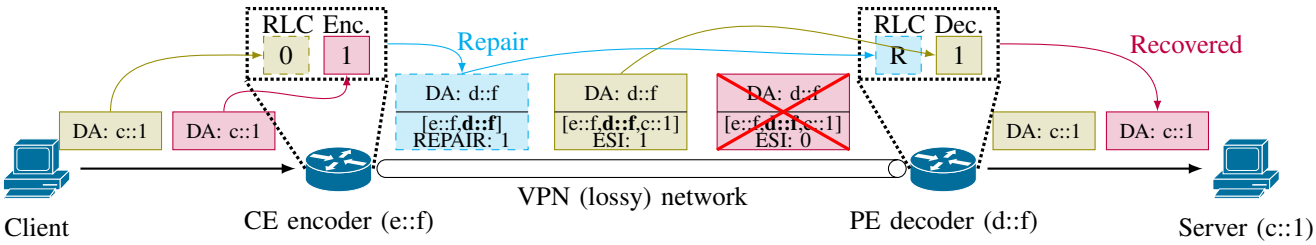


Figure 1. Architecture of HIRT to protect VPN traffic. The PE encapsulates the packets from its client with an SRv6 header and adds the *Encoder* and *Decoder* SIDs to protect the packets with HIRT. The *Encoder* also assigns an ESI to each source symbol and generates the repair symbols (dashed packet) using the RLC coding scheme. The CE recover the losses using the received symbols (source symbol with ESI=1 and the repair symbol) and decapsulates the SRv6 header of the source symbols before transmitting the packet to the server. The repair symbol does not go out of the tunnel.

Routing [32], [33] brings the support for network programming [34] that enables operators to deploy advanced services in their network. We propose the High-Speed Robust Tunnel (HIRT), which implements FEC at the network layer. As further detailed in Section II, it enables the protection of multiple segments simultaneously or even an entire backbone inside a tunnel. FEC can be suggested as a service to end-hosts wishing to optimize the latency of their applications without requiring to implement FEC themselves. Multiple end-hosts may benefit from the same tunnel simultaneously without compromising the security and integrity of their communication. Tunnels play an important role in the Internet today. They are used for Virtual Private Networks [35], networks using orthogonal technologies [36]–[38], MPLS [39], Segment Routing [31] and even in cellular networks [40].

Three main challenges arise when considering FEC at the network layer: (i) propose FEC *as a service* to protect specific flows requiring low latency without user intervention and using already deployed protocols; (ii) adapt the ratio of redundancy packets to the network conditions to avoid wasting too much bandwidth while offering good enough protection to recover losses in the tunnel; (iii) efficiently generate redundancy packets and provide fast lost data recovery to simultaneously protect distinct flows.

We provide an implementation of HIRT leveraging IPv6 Segment Routing (SRv6) [32] and its network programmability [34] to deploy FEC tunnels protecting specific segments in the network. Figure 1 shows the overall architecture of HIRT protecting a VPN traffic. HIRT solves the first challenge by allowing clients to benefit from FEC without knowing it and leveraging SRv6. In Figure 1, the *Customer Edge* (CE) encapsulates the traffic from the *Client* inside an SRv6 header to trigger FEC protection (Section III). The second challenge is addressed by leveraging feedback from the decoder, and using the Convolutional Random Linear Code [41] scheme to dynamically adapt the amount of redundancy (Section IV). To address the last challenge, we implement HIRT in FastClick [42] with several optimizations, including kernel bypass and a scalable multithreading architecture (Section V). We show that HIRT can protect >50 Gbps under $>3\,\%$ losses.

We further demonstrate the effectiveness of HIRT over the Low Earth Orbit (LEO) satellite network Starlink, which exhibits *non-congestion* induced losses [9] that can double the flow completion time of short requests. We use HTTP benchmarks to show that HIRT can effectively recover lost packets independently of the transport protocol and reduce this tail latency by up to a factor of 2. HIRT also reduces the request completion time of longer responses by several RTTs by recovering packet losses without impacting the end-host's transport congestion controller. Finally, we highlight the benefits of HIRT on a network file system benchmark using Starlink. We show that tail latency is reduced by up to $20\,\%$.

The use of FEC below the transport layer to protect aggregated traffic across data centers has already been explored by Maelstrom [43]. Maelstrom uses multiple interleaved XOR coding windows to recover from burst losses. In 2011, Maelstrom could protect up to $700\,\mathrm{Mbps}$ of traffic. The source code of Maelstrom is not publicly available for comparison. Instead, we build a simple packet simulator generating reproducible losses and implement HIRT and Maelstrom to further compare their performance in Section VIII.

Contributions. We make the following contributions:

- The design of HIRT inside IPv6 Segment Routing.
- An adaptive algorithm to schedule the generation of repair symbols based on loss estimation feedback and RLC.
- A high-speed implementation[1] of HIRT, enabling $>50\,\mathrm{Gbps}$ of traffic protection under heavy losses.
- Evaluations of HIRT in a real network exposing transmissions error (Starlink) under HTTP and file transfer benchmarks.
- A packet simulator where we compare HIRT and Maelstrom [43] under reproducible loss patterns.

II. Network Layer Forward Erasure Correction

Forward Erasure Correction is a recovery method consisting in sending redundancy information (repair symbols) alongside the data (source symbols). A recipient can recover lost source symbols by using this redundancy without the need for retransmissions. Erasure codes are generally used to generate the repair symbols, e.g., the Reed-Solomon codes [44]. Consider an emitter sending source symbols $S_1, S_2, \ldots, S_m$ and repair symbols $R_1, R_2 \ldots, R_n$. A receiver recovers any set of $k \leq m$ lost source symbols by receiving at least k repair symbols. The Convolutional Random Linear Codes (RLC) scheme [41]

[1]Source code available: https://github.com/louisna/HIRT.git.

is another method, used in this work, to generate the repair symbols. Each redundancy data is computed as a linear combination of the source symbols it protects. The coefficients of the linear combination are pseudo-randomly generated. The recipient recovers the lost symbols by solving a linear system using the correctly received symbols. The XOR scheme is a special case of RLC where all coefficients are set to 1.

Forward Erasure Correction techniques are typically applied at the datalink [19]–[21] and the transport layer [12], [18], [22]–[29]. Compared to the transport layer, implementing FEC at the network layer (NL-FEC) brings several benefits. First, the computation is done on intermediate routers. It enables to aggregate the traffic from multiple sources to increase the overall flow from the encoder to the decoder. Aggregating numerous flows helps to spare resources as the same redundant packets can protect different flows. When the losses happen in bursts, this prevents every transport sender from sending numerous redundant packets for each flow to ensure that long bursts can be recovered. This is not possible with transport-layer FEC.

Second, NL-FEC enables different use-cases compared to FEC at the transport layer. At the transport layer, FEC can leverage knowledge from the application to better schedule the repair symbols generation [16]. However, the FEC emitter cannot make any assumption on the paths its packets follow. At the network layer, one can dynamically deploy FEC to protect specific links or paths that exhibit *non-congestion* induced losses. It also becomes possible to deploy multiple independent tunnels in a given network.

The flexible deployment of the tunnels enables network operators to make several assumptions about the link or path to protect, such as the available resources of the routers (buffer sizes, network interface controllers, link reliability,...). When the network is over-provisioned in terms of bandwidth, which is the case for common internet or cloud providers [45], one may assume that packet losses are mainly due to transmission errors. When the error is due to congestion inside the tunnel, the decoder can mark the recovered packet with ECN [46] to notify the end user. The decoder can also mark repair packets with a lower quality-of-service so that the routers along the tunnel drop these packets first.

When operating over a link with non-congestion-induced packet losses, recovering these packets at the network layer and hiding those loss events to the transport layer avoids an unnecessary decrease of the transport layer congestion window. Hiding loss events to the congestion controller is explicitly discouraged at the transport layer where the origin of loss events is undetermined [47]. By deploying HIRT on links where losses are known to not be congestion-induced, erasures can be recovered and hidden to the transport layer to increase the throughput of transport connections with no harm for the network. We highlight this behavior in Section VII-A.

III. Integration in IPv6 Segment Routing

This section presents the design of HIRT within IPv6 Segment Routing (SRv6) using the Network Programming paradigm [34]. Similar approaches can be taken to integrate network layer FEC in different protocols, e.g., MPLS.

IPv6 Segment Routing is a source-routed mechanism where the source specifies the network segments that the packet must follow to reach its destination. In SRv6, each Segment Identifier (SID) is represented by an IPv6 address. An SID can embed network functions to trigger specific behavior on the packet directly in the data plane [32]. Operators often deploy SRv6 for intra-domain purposes but its deployment increases on the Internet [48]. As a result, HIRT is dependent of the deployment of SRv6.

A. HIRT in SRv6 Network Programming

We leverage the *Segment List* in the Segment Routing header to deploy FEC on routers. The two SIDs (of the encoder and decoder) are carried in the *Segment List* of the SRv6 header. This SRv6 header can be added either by the encoder, an upstream router or the source (in Figure 1, the CE adds this header). In the first case, the network operator may decide to protect all traffic to a given destination without requiring applications to be aware of the tunnel. In the second situation, any host may decide to use the tunnel for its specific traffic. Only applications that may benefit from HIRT (e.g., latency-sensitive ones) can activate it by adding the SRv6 header. This list is also used to recover some fields of the IPv6 packet header that may be altered inside the tunnel by intermediate routers, such as the destination address, that is different when a packet is processed by the *Encoder* and *Decoder* routers.

B. Source and repair symbols representation

HIRT protects data at the packet-level. Thus, the source symbols are IPv6 packets exchanged by end-hosts through the tunnel. Using packet-level protection enables the tunnel to make *no assumption* on the data it protects, nor does it require the encoder and the decoder to analyze the content of the packets. HIRT processes SRv6 packets as payload without compromising the security of communication. Alternatives to packet-level protection (e.g., a transport connection-level protection) would require additional state on both the encoder and decoder and are not explored in this paper.

Each source symbol is uniquely identified by an *Encoding Symbol ID* (ESI), a 32-bits value incremented by one for each source symbol protected by the encoder. To carry the ESI, the encoder adds an SRv6 *Type-Length-Value* (TLV) option [32]. Routers that do not recognize the TLV type skip it during the packet processing. This allows HIRT to be carried over transparently by standard routers.

Repair symbols carry the *repair FEC payload*, i.e., the redundancy data used to recover lost packets. They are generated by the encoder and sent towards the decoder (Figure 1). They are not forwarded outside the tunnel. The repair payload is encapsulated in an SRv6 header to be routed to the decoder. This forms the repair packet. A repair symbol is also uniquely identified, using the ESI of the last source symbol it protects. It also contains the number of source symbols protected by

the current repair symbol, as well FEC scheme-specific information, e.g., the RLC seed used to generate the coefficients of the repair symbol (not shown in the Figure). Finally, the SRv6 TLV of repair packets contains a field indicating which FEC algorithm was used to generate the repair symbols. HIRT only implements RLC but network operators could implement, e.g., XOR-based algorithms.

IV. Adaptive FEC

The use of Forward Erasure Correction involves a trade-off between transmission delay and bandwidth consumption [49]. Adding redundancy increases the probability of recovering lost data, reducing the transmission delay, but it also increases the bandwidth usage. Several works suggest schemes to control and dynamically adapt the coding rate of FEC based on feedback [13], [26], [50], [51]. However, these designs focus on providing (partial or full) reliability at the cost of added delay and do not prioritize efficient packet processing. We design an adaptive algorithm to schedule repair symbols with three considerations: (i) it must perform well in heterogeneous networks, such as those with varying delay or reliability (e.g., satellite networks); (ii) the resources required to generate the feedback must remain adequate to ensure a high-speed delivery; and (iii) no assumptions can be made about the protected traffic. For example, it may be a bulk download, a video conference call or a mix of both.

Our system leverages feedback from the decoder to the encoder to estimate the loss events in the tunnel. The amount of generated repair symbols is adjusted using this estimation. HIRT leverages the sliding-window RLC algorithm from RFC8681 [41] and adapts the step size according to the following adaptive algorithm.

A. Loss estimation and feedback

The decoder sends feedback messages to the encoder inside the data plane. Concretely, it regularly sends an SRv6 packet with feedback information inside an SRv6 TLV option [32]. These messages contain the estimated loss characteristics from the decoder's viewpoint since the last feedback. Each source symbol is identified with a monotonically increasing Encoding Symbol ID (ESI). The decoder marks a packet as lost if it sees a gap in the ESI sequence. The question of packet reordering is discussed in Section V. Feedbacks are computed and forwarded each n_f packets. The decoder returns the window size n_f and the number of missing ESI in that window.

With this feedback, the encoder adjusts its estimation of the mean loss inside the tunnel, $\hat{\mu}$. The algorithm uses a moving average update to give more importance on recent estimations to adjust to the heterogeneity of the network. The encoder also computes the variance in the mean number of losses, $\hat{\sigma}$, with these values. The updated loss estimation solves the first requirement.

This feedback is easy to compute both for the encoder and the decoder (a bitmap of size n_f is sufficient to store the state of received ESI). Moreover, this feedback does not contain sensitive data: a missing feedback (e.g., due to transmission error) does not block the encoder from producing FEC repair packets. This solves the second requirement. Finally, no assumptions are made on the protected traffic, except that we analyze the network condition based only on protected packets.

B. Adaptive algorithm

The encoder generates repair symbols to compensate the estimated loss distribution thanks to the feedback mechanism. As the number of repair packets to generate is based on the previous estimation (and thus computed *a priori*), the encoder may generate too much or too few symbols. The first case results in wasted bandwidth; the latter in the impossibility to recover all lost symbols.

To compensate for the $\hat{\mu}$ estimated symbols that will be dropped within the next n_f sent source symbols, the encoder generates $\lceil \hat{\mu} * n_f \rceil$ repair symbols every n_f source symbols. The variance $\hat{\sigma}$ is taken into account in this computation.

Two strategies exist to schedule the generation of repair packets. First, they can either be generated all at once, every n_f source symbols. In that case, if one of the first source symbols of the window is lost, the decoder must wait for the reception of all remaining source and repair symbols to be able to recover the loss. This can potentially add a great amount of delay for the receiving end-host. The other strategy is to interleave repair symbols with the data. Evaluations of Forward Erasure Correction methods showed that the second strategy provides lower delay to recover lost source symbols without impacting the quality of recovery [52]. HIRT relies on the second strategy using the RLC coding scheme as it can generate repair symbols interleaved with the source symbols.

The scheduler thus generates n_f repair packets every $\lceil \frac{n_f}{\hat{\mu}+\hat{\sigma}} \rceil$ source symbols. *HIRT interleaves these repair packets and generates a repair symbol with a step of* $\lceil \frac{1}{\hat{\mu}+\hat{\sigma}} \rceil$ *source symbols*. The step may be adjusted every received feedback.

V. High performance software implementation

A natural approach for network-layer Forward Erasure Correction with high-speed packet processing and recovery is to implement it in hardware directly, e.g., in P4 [53]. However, RLC requires computing linear combination of packets (at the encoder) and solving linear systems of equations (at the decoder), which is complex and tedious to be efficiently implemented in hardware. Simpler algorithms could be implemented, such as the XOR erasure code, but its efficiency has been demonstrated to be low [12] and we do not consider it in this work. Maelstrom [43] explores multi-layered XOR codes, which can protect burst losses at the cost of constant and significant overhead. Comparison between Maelstrom and HIRT is further explored in Section VIII.

HIRT leverages FastClick [42], an enhanced version of the Click modular router toolkit [54] to build software network functions supporting traffic above $100\,\mathrm{Gbps}$. FastClick integrates kernel bypass I/O such as DPDK [55] and packet batching. With FastClick, one defines packet processing as a graph of smaller components that run in user-space. We

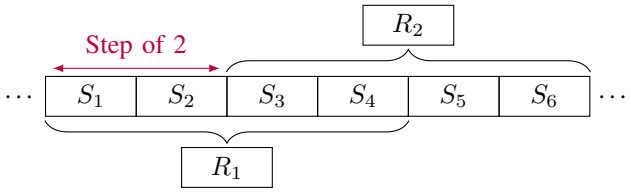


Figure 2. Sliding window coding with $w = 4$ (window size) and $\hat{\mu} = 0.5$ (mean loss estimation) resulting in a step of 2 between two repair symbols.

extended the toolkit to support basic SRv6 processing. This required ∼330 lines of code (LOC). FEC must be done carefully as it can be CPU-intensive to compute the repair symbols and to solve the RLC linear system. Packets that are not protected by the tunnel (i.e., packets without the SRv6 header with the encoder and decoder SIDs) are processed and forwarded without entering the HIRT component in FastClick. The whole implementation required ∼2300 LOC.

FEC sliding window. Conventional FEC approaches use a constant and limited FEC window size, i.e., all repair symbols protect the same number of source symbols. As detailed in Section IV, repair symbols are interleaved with the source symbols they protect. HIRT uses a constant sliding window of size w, i.e., $n_f = w$ all the time. At the regime, a new data entering the window removes the oldest symbol from it. Figure 2 shows an example of six source symbols protected by two repair symbols. Repair R_1 and R_2 respectively protect sources $S_{1\to4}$ and $S_{3\to6}$. Increasing w improves recovery capabilities at the cost of more processing each time a new repair packet is generated, and a linear system is built. Carefully choosing the right value of w thus brings a trade-off between recovery capability and processing performance.

DPDK. Our implementation uses DPDK [55]. The toolkit provides kernel bypass to speed up the packet processing in user-space [42]. DPDK reserves beforehand a pool of buffers for the packets, thus canceling the cost of memory allocation. Our solution also uses batching of 32 packets. These two features are already supported in FastClick.

AVX instruction. RLC is computationally intensive. It uses costly Galois Field operations to work with integer numbers in our equation systems. To alleviate the cost, we use the `moepgf` C library [56] which uses AVX SIMD instructions.

Zero copy. The source symbols are kept in memory to generate the repair symbols (at the encoder) and construct the linear system to recover lost packets (at the decoder). Instead of copying the packet in separate buffers, our implementation keeps a pointer to the packet buffer and updates the DPDK buffer's reference counter. Memory is only dropped once every reference to the same packet are no longer valid, i.e., the packet has been processed and is removed from the FEC buffers. The packet may be modified while being kept by reference counting. It happens when the SRv6 header is modified after the encoding or decoding operation is applied, e.g., to change the IPv6 Destination Header field and decrement the Hop Limit. FastClick prevents concurrent writing by automatically copying shared packets when an attempt to get a writable

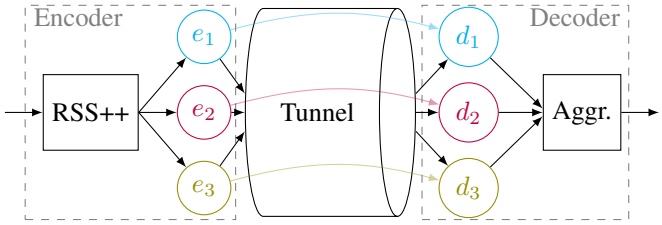


Figure 3. Multithreading architecture of our implementation with 3 parallel streams. The *Encoder* distributes the incoming packets among the encoder streams e_1, e_2 and e_3. Each encoder stream e_i communicates with its decoder stream d_i. All packets are independently sent with the same tunnel but each stream is independent from each other. At the end of the decoding streams (d), the traffic is aggregated and forwarded. The curved arrows between the e_is and d_is represent the streams.

reference to a shared packet is made. This adds unnecessary copies that slow down the performance. Instead, HIRT only keeps a copy of the fields that might be modified and manually fixes the modified buffer upon repair symbol generation and system solving. The size of these fields does not exceed 40 bytes. This trick allows disabling the copy-on-write security of FastClick and only keep one copy for each packet.

Multithread architecture. Our implementation supports multithreading. The bottleneck operations are the generation of the repair symbols, and the construction and the solving of the linear system. When a CPU core performs one of these operations, it cannot process other incoming packets in its queue. Our implementation scales by using several cores in parallel. For this, we split the traffic into parallel streams.

Figure 3 presents the multithread architecture with 3 concurrent processing threads. At the encoder, the network interface controller (NIC) distributes incoming packets among the available CPU cores. This distribution uses RSS++ [57] to spread the load equitably among the cores and dynamically scale according to the input load. RSS++ maps packets from each flow to the same CPU core to avoid possible reordering.

There is a one-to-one mapping between an encoding thread and an available CPU core. A *Stream ID* is assigned to each encoding thread. There is no communication between encoding threads with distinct Stream IDs. Each thread processes its packets and encodes them independently of the other threads. For example, each stream uses its own Encoding Symbol ID (ESI) space. This architecture creates several concurrent streams inside HIRT, as represented in Figure 3. The *Stream ID* of a source or repair symbol is carried inside SRv6 TLV of the source, repair and feedback packets. The decoder uses the same architecture. It uses the *Stream ID* from the TLV to map the symbol to the correct decoding CPU core. The system constructions and resolutions are also independent between streams. The feedback messages are also stream-dependent. Each stream has its own estimation of the loss pattern inside the tunnel. This architecture enables to arbitrarily increase the number of threads and Stream IDs *without* contention of a single performance bottleneck for the FEC computation.

Overloading detection. A vicious circle may occur in case of severe losses in the tunnel. In that scenario, the encoder decides to generate numerous repair symbols in a short amount of

time, increasing its CPU load. If the load reaches the maximal capacity, the router will have to drop incoming packets, thus creating losses at the tunnel ingress. This would deteriorate the performance of the applications using the tunnel. The decoder may also become overloaded in case of severe losses as it tries to solve numerous linear systems. Similarly, it would drop packets at its ingress, thus creating more losses inside the tunnel. These drops will increase the estimated network losses. In response to this additional loss, the encoder will decide to generate more repair symbols, keeping increasing the CPU load at the decoder.

Our implementation alleviates this situation by *randomly skipping* repair symbols generation and system solving if needed. When a stream CPU core sees its load reaching a defined threshold value, it starts to randomly skip these events with a probability *proportional* to the current CPU overload. Other methods could be implemented to, for example, genuinely decide which linear system the decoder should skip instead of randomly selecting one system to discard. However, this situation occurs when the CPU load is high, and it should not require additional resources to make a decision.

Loss detection and packet reordering. The strategy to consider a packet as lost has an impact on the performance of the implementation. A missing ESI in the source symbol buffer is interpreted as a lost packet. A repair symbol is always sent by the encoder *after* the source symbols it protects, as illustrated in Figure 1. Upon reception of the repair symbol, a missing ESI can only be due to losses or reordering. Intermediate routers in the protection tunnel can introduce reordering. A study highlighted that such events are rare but the average out-of-order length is of 8 packets [58]. If a source symbol is not lost and simply out-of-order, the decoder might trigger the recovery mechanism, increasing the CPU load. Two different strategies can be implemented to counter this side effect. First, upon reception of a repair symbol, the decoder can postpone the recovery mechanism for a few packets. The decoder could receive out-of-order packets and detect that a recovery is not necessary anymore. The second approach is to consider that out-of-order source symbols are already delayed for the application. In that case, recovering this source symbol means that it will be sent *faster* out of the tunnel compared to the first solution because the tunnel does not need to wait for the reception of the out-of-order packet. However, it adds CPU load because additional recoveries are triggered. Our implementation uses the second approach to decrease the impact of reordered packets, thus reducing the overall jitter for the applications using HIRT.

VI. Benchmarks

We assess the performance of HIRT at high speed to show that a carefully designed prototype, even when implemented in software, can support >50 Gbps of traffic under heavy losses using a few commodity CPU cores. Throughout the experiments, we set $w = 200$. We experimentally found that a higher value does not significantly improve the recovering capabilities while the processing overhead decreases the performances

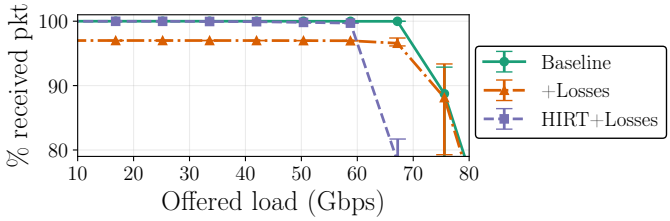


Figure 4. Throughput acc. offered load.

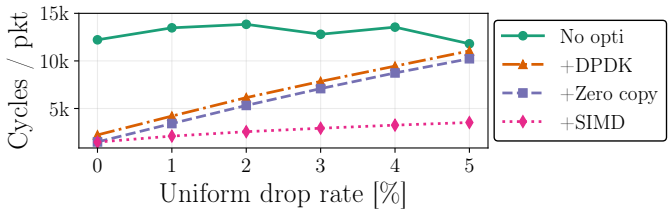


Figure 5. Impact of the optimizations on processing.

at high-speed. We rely on the CloudLab infrastructure [59] to benchmark our implementation. We use a linear topology composed of five nodes, respectively the server, the encoder, a node emulating losses, the decoder and the client. All nodes are equipped with x86 AMD EPYC 7452 processors. Each node has 32 CPU cores. The L1, L2 and L3 cache layers are respectively 32 kB, 512 kB and 16 384 kB long. All nodes are connected through 100 Gbps links inside the Utah cluster. We evaluate the limits of HIRT in terms of throughput.

Offered load. Figure 4 shows the percentage of packets received by the client given an increasing offered load (using 1024 bytes UDP packets). The baseline curve (i.e., without losses nor HIRT) shows that we start observing losses before an offered load of 80 Gbps. This result shows that we cannot fully utilize the available 100 Gbps bandwidth and that the machines are the bottleneck of the experiment. **When inducing 3 % of packets uniformly dropped, HIRT recovers all dropped packets below** 60 Gbps. When the CPU load of the routers increases, the amount of recovered packets is capped as explained in Section V to avoid dropping packets due to CPU contention.

Above 60 Gbps, the client receives fewer packets with HIRT compared to the `Baseline+Losses` situation, i.e., coding nodes drop packets in the tunnel because the FEC processing increases too much the CPU utilization. We argue that the CloudLab servers used in this experiment are the bottleneck. First, thanks to its multithreading architecture, providing more computing cores should allow HIRT to scale to even more offered payload. FastClick [42] and RSS++ [57] can sustain more than 100 Gbps traffic while the link capacity between the CloudLab nodes is not the bottleneck here. We believe that, with more advanced servers with more CPU cores available for processing, one could push the limits of HIRT to more than 100 Gbps. Additionally, HIRT supports only 10 Gbps less traffic compared to the *Baseline* curve to recover 3 % of losses. With fewer losses, we expect HIRT to sustain even more traffic and close the gap compared to the baseline. The drop rate of 3 % is chosen to understand the performance limits of HIRT.

Impact of the optimizations. Figure 5 shows the average

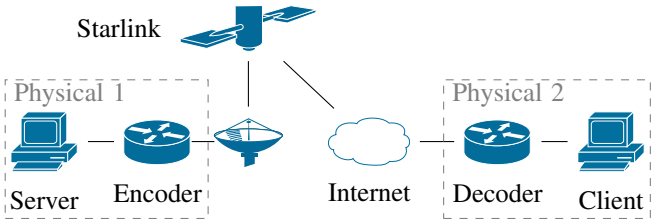


Figure 6. Experimental setup using the Starlink access. The *Server* and *Encoder* are run on the same physical machine using different network namespaces; similarly for *Client* and *Decoder*. The *Encoder* and *Decoder* are respectively the *CE* and *PE*.

number of CPU cycles per packets of HIRT for the decoding operation with the optimizations mentioned in Section V. We increase the uniform drop rate from 0 % to 5 %. Each subsequent curve adds an optimization compared to the previous one, e.g., the `+SIMD` curve results from enabling `DPDK`, the `Zero Copy` and the `SIMD` optimizations. Without optimization, we observe that the number of CPU cycles remains constant as the uniform drop rate increases, yet it should require more work. This indicates that, without optimization, the implementation struggles to sustain high traffic, even in the absence of losses. DPDK provides the biggest performance boost, as already established by previous research [42]. The number of cycles increases with the drop rate as more work must be done to recover lost packets. The technique to allow zero-copy enables an almost constant 600 cycles improvement. This is independent of the number of losses since the zero-copy only applies to received packets. Using SIMD instructions provides strong performance gain under heavy losses. With no packet drop, there is nearly no packet recovery and SIMD instructions do not accelerate processing. However, as the drop rate increases, the speedup rises towards 3× with 5 % of packets being dropped because more linear systems must be solved. **With all optimizations enabled, the number of CPU cycles decreases by a factor between 3 and 9 depending on the loss percentage**.

VII. Evaluation in a Real Network

We demonstrate the benefits of HIRT in a real-world environment using the Starlink [60] communication system with HTTP and NFS benchmarks. Starlink is a new Low Earth Orbit (LEO) satellite communication system that aims to provide high connectivity. Starlink is a rising topic of interest in the research community [9], [61]–[63]. More particularly, measurements have shown that this medium generates losses that are *not induced by congestion* [9]. The study measured a mean loss ratio of ∼0.45 %, with burst of losses of at most 9 packets in 75 % of measurements under light traffic. These losses *can* be recovered by an FEC algorithm, as the overhead induced by the repair packets should not contribute in increasing the number of data losses in the network.

Figure 6 illustrates the experimental setup. The server and the encoder share the same machine with different network namespaces; this is similar for the decoder and the client. The encoder is directly connected to the Starlink access point; the decoder is connected to our university campus network. The RTT is ∼50 ms, and we found similar loss conditions to [9].

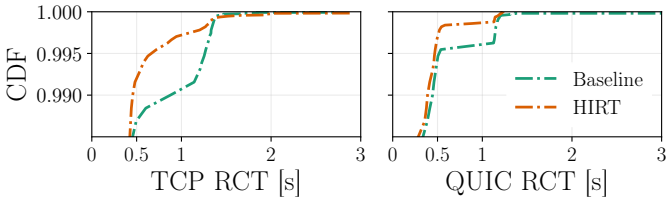


Figure 7. HTTP request completion time CDF for 10 kB responses.

A. HTTP requests over Starlink

We show how HIRT transparently protects a VPN traffic over Starlink. Such service can be provided to any VPN user to increase the reliability of the service, independently of their protocols and applications. Additionally, using a VPN inside HIRT shows that the tunnel can recover losses without accessing the protected data.

We use the Wireguard VPN [64] to leverage its Linux kernel implementation. We start an HTTP server on the server, and run a wrk [65] benchmark for 20 minutes. Wrk is an HTTP benchmark tool designed to evaluate the responsiveness of a server. We analyze the impact of the losses on the request completion time (RCT). Each benchmark consists in 5 parallel request flows to avoid congesting the access link. We run each benchmark individually.

HIRT reduces the last percentile of 10 kB HTTP request completion times by a factor of 2. Figure 7 shows the distribution of the request completion time (RCT) for responses of 10 kB, i.e., responses that can be sent within a single round-trip. We compare the RCT between HIRT and a no-FEC solution only relying on retransmissions (the *Baseline* curve). The left sub-figure uses regular TCP as the transport protocol. As Starlink exposes <1 % losses, the impact of packet erasure only appears around the last percentile of the distribution. We observe a similar RCT for lower percentiles and only show the last percentile latency. The *Baseline* curve suffers from the losses and the retransmissions increase the request completion time. It increases up to more than 1 s for almost 1 % of these requests, whereas the median over all data is ∼100 ms. As expected, HIRT decreases the RCT for the 99$^{\text{th}}$ percentile from 1 s to 500 ms. The measured overhead is 9.25 %. As we do not fully utilize the link, we do not expect results to vary between upload and download Starlink access.

HIRT protects traffic independently of the underlying protocols. We also used a version of wrk [66] which uses the `picoquic` [67] implementation of QUIC [6] instead of TCP for the HTTP requests. The right part of Figure 7 shows the request completion time for 10 kB HTTP responses using QUIC. Again, HIRT does not harm the mean RCT, and we only show the last percentile distribution. HIRT decreases the 99.5$^{\text{th}}$ RCT from 1 s to 500 ms.

Both the QUIC and TCP servers used CUBIC as their congestion control algorithm. The observed losses contribute to a decrease in the congestion window. However, for small HTTP responses such as 10 kB, these losses do not harmfully

impact this window. The initial congestion window of the Linux TCP implementation is 10 packets, which is sufficient to transmit all the response within a single flight. Packet losses in this case increase the request completion time by one up to a few RTTs, but no subsequent packet will be impacted by a congestion window decrease.

By hiding the losses from the transport protocol, HIRT decreases the median RCT of longer HTTP responses by $20\,\%$ **over Starlink**. Longer HTTP responses involving multiple packet flights are more impacted by packet losses when the transport protocol uses a loss-based congestion controller. Improvements in the loss recovery mechanism and its impact on the congestion control have been added to TCP [68]. However, loss-based congestion control algorithms use packet losses as an indicator of congestion. The erasures caused by Starlink can negatively impact the congestion window of the sender even if they are *not* due to congestion.

Figure 8 shows the impact of HIRT on the request completion time of TCP/HTTP responses of $1\,\mathrm{MB}$ (left sub-figure). Both endpoints use CUBIC. Contrary to the $10\,\mathrm{kB}$ requests, we see a positive impact of FEC considering all experiments, not just the last percentile, even if the losses exhibited by Starlink impact less than $1\,\%$ of the packets. With HIRT, the median is lower ($1.9\,\mathrm{s}$ compared to $2.36\,\mathrm{s}$, $19\,\%$ improvement). By recovering erased packets at the network layer, the TCP congestion controller is unaware of these losses. Thus, it avoids decreasing its congestion window. This increases the overall throughput compared to the baseline.

Figure 8 also reports the congestion window (`CWND`) of the server at the end of each HTTP response. As expected, it remains higher using FEC thanks to the recovered packets. Since CUBIC is a loss-based congestion control algorithm (CCA), hiding losses has a positive impact on the evolution of the source bit-rate. We expect less significant results with a delay-based CCA such as BBR. Finally, we expect the main outcomes from Figure 8 to remain identical with QUIC instead of TCP as well as with longer HTTP responses.

By knowing the characteristics of Starlink, we know that an important part of these observed losses are not due to congestion, and the transport congestion controller can make better decisions. However, as discussed in Section II, one cannot always guarantee that losses are transmission errors. To alleviate this situation, we run HIRT where we mark recovered packets with the Explicit Congestion Notification (ECN) flags [46]. With this notification, the tunnel still recovers packets but does not hide the losses anymore. This is represented by the `HIRT+ECN` curve from Figure 8. Both the RCT and the CWND are similar to the baseline. We even notice that the RCT is marginally higher when we activate ECN. We believe that the 5.4 Linux TCP implementation used by the server is more optimized to handle packet losses than ECN-flagged packets. An inspection of its source code confirmed that the TCP stack enters a congestion reduction phase for each received ECN packet, thus shrinking the congestion window. A more optimized control of ECN might improve the results. However, we note that for the last 10^{th} percentiles, `HIRT+ECN` improves the latency by $\sim 200\,\mathrm{ms}$ compared to the baseline.

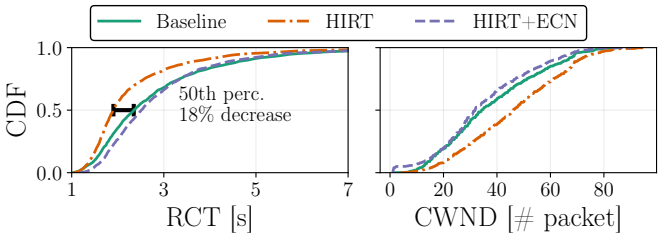


Figure 8. HTTP request completion time (left) and TCP congestion window (right) for 1 MB responses.

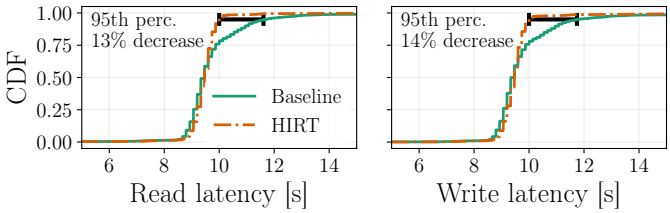


Figure 9. Read (left) and write (right) latency results of the `fio` benchmark.

B. File system benchmark

Network File System (NFS) is an important application for remote workers who need to access files on enterprise servers. As they require interactions with humans, these are latency sensitive. Increased latency hinders the user experience. A FEC protection can improve the tail latency of these applications. However, due to the low traffic because of the human interaction, implementing FEC in the application or relying on the transport layer would require a huge overhead to recover from lost packets. Moreover, NFS can use TCP or UDP. With HIRT, applications benefit from traffic aggregation and from the FEC protection independently of the transport protocol.

We show how an NFS application is impacted by the increased latency due to the losses of Starlink. We consider such use-case realistic as Starlink aims to provide internet connectivity to remote places in the world. Optimizing the delay of such applications is mandatory with the spread of homeworking today. We start an NFS server on the machine in our university campus and start a `fio` [69] benchmark on the *Client* from Figure 6. `fio` executes sequential read and write operations using NFS and measures the latency of each request. We run the benchmark for $20\,\mathrm{min}$ with randomly read/write sequences of $5\,\mathrm{MB}$ files and a block-size of $4\,\mathrm{kB}$. The objective is not to objectively evaluate the NFS disk capabilities, but see the relative impact of losses and HIRT on the latency. Figure 9 shows the CDF distribution of the latency of each read (left) and write (right) request from the `fio` benchmark. We see that the improvements of HIRT are significant, **as more than** $97\,\%$ **of the samples present a latency below** $10\,\mathrm{s}$**. Without HIRT, only** $\sim 75\,\%$ **of the samples are completed within this time**. For the 99^{th} percentile, HIRT improves the latency by $14\,\%$.

VIII. Comparison with Maelstrom

Maelstrom [43] is the closest related work to HIRT. It suggests deploying proxies between DCs and WANs and to

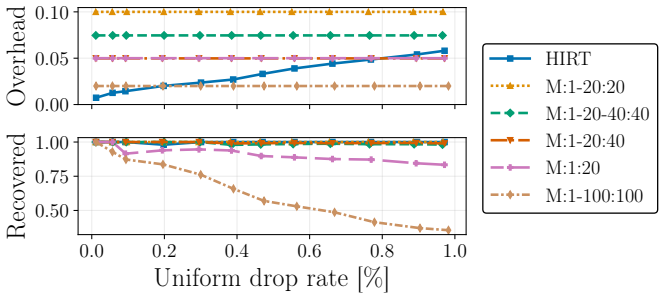
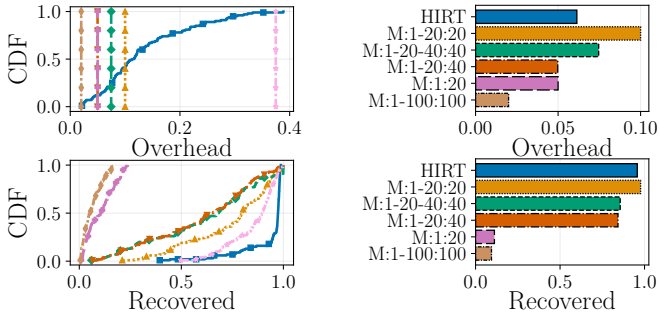


Figure 10. Uniform drop model simulation.



(a) Experimental design. (b) Starlink model.

Figure 11. Simulation with Gilbert-Elliot drop model.

inject FEC packets between unmodified packets. It proposes to encode packets in multiple layers. For example, the first layer encodes every packet, while subsequent layers with, e.g., an interleaving of 4 will encode 4 streams of 1 packet every 4 packets. The trivia is to allow recovery of dropped bursts. All layers use the XOR coding scheme to generate the repair symbols. We note the following differences between Maelstrom and HIRT: (i) Maelstrom suggests constant-rate redundancy generation inside the tunnel. It does not adapt to the network condition. As a result, Maelstrom may send too many or too few repair packets, respectively causing bandwidth waste or impossibility to recover the majority of lost source symbols; (ii) Maelstrom design does not focus on high-speed packet processing and could not sustain more than 700 Mbps [43] of protected traffic in 2011. The throughput should be higher with more recent machines, but we could not test it as its source code is not available.

We compare in this section the recovering capabilities of HIRT with Maelstrom. We build a simulator with the two methods to compare their behavior under the same patterns of losses. Finally, we implement the encoder and decoder components of HIRT and Maelstrom. Between these two nodes, we add a deterministic packet dropper simulating losses using a uniform and a Gilbert-Eliott [70] drop model. The simulator and implementation of HIRT and Maelstrom required ∼2500 LOC. We reuse the RLC library used inside FastClick for HIRT. We tune Maelstrom with different interleaves and window sizes to trade-off overhead with recovering capabilities. In Figure 10 and 11, *M:X-Y-Z:W* means Maelstrom with interleaves $I = (X, Y, Z)$ and a window W.

Uniform drop model. We first compare HIRT and Maelstrom by increasing the uniform drop rate in Figure 10. We measure the overhead, i.e., the ratio of repair packets injected to the network compared to the number of source symbols (upper sub-figure), and the ratio of recovered packets, i.e., the number of recovered symbols with respect to the number of losses (lower sub-figure). Maelstrom does not adapt to the uniform drop rate since it uses static interleaving. Figure 10 shows that HIRT correctly adapts to the observed losses, and provides lower overhead than Maelstrom while still recovering erasures. For example, with a drop rate of 0.5 %, HIRT recovers all erased symbols with an overhead of 3.3 % while *M:1-20:20* only recovers 89 % of them for an overhead of 5 %. With more layers (e.g., *M:1-20-40:20*), Maelstrom recovers almost all lost symbols at the cost of a higher and constant overhead compared to HIRT. Even if RLC is more computationally intensive, its recovering capabilities outperform Maelstrom's method.

Gilbert-Eliott model. This model enables to emulate burst losses by running a two-sates Markov model. Packets are forwarded if the model lies in the *Good* state, and dropped if it is in the *Bad* state. The probability to go from the *Good* to the *Bad* state is p; the probability to go from the *Bad* to the *Good* state is r. We first use an experimental design [71] approach with 100 points to explore the space of possible scenarios. We use the ranges $p \in [0.0001, 0.01]$ and $r \in [0.10, 0.50]$. Figure 11a presents the CDFs of the overhead (top) and ratio of recovered packets (bottom) in this setup.

The impact of burst losses on the performance of Maelstrom is significant, as with $I = (1, 20)$ and a window of 20, more than 50 % of the experiments recover less than 75 % of erased packets. On the other hand, HIRT correctly adapts to the loss rate thanks to the adaptive algorithm. More than 95 % of erased packets are recovered in 80 % of the experiments. Figure 11a also presents how Maelstrom performs with $I = (1, 20, 40)$ and a window of 8 packets, values similar to the original paper [43]. In that setup, the overhead is 34 %, which is above HIRT 99 % of the time. However, the ratio of recovered packets of Maelstrom remain almost always lower than HIRT.

Figure 11b shows results where we approximate Starlink's loss model [60]; we set $r = \frac{1}{3}$ and $p = 0.00151$. Again, HIRT outperforms Maelstrom. **Thanks to its adaptive scheduling and the Convolutional RLC erasure code, HIRT recovers many more erased packets than Maelstrom while ensuring lower overhead**. As the original implementation of Maelstrom is not publicly available, we could not compare its performance at high-speed. Even if we could expect a higher throughput than 700 Mbps [43] nowadays, we still believe that the architecture of HIRT and the optimizations done would sustain a higher throughput than Maelstrom.

IX. Related work

Network-layer coding. Sundararajan *et al.* [72] suggest a network coding extension of TCP for multicast and multipath traffic protection. The FEC mechanism is implemented between TCP and the network layer, and packet coding can be performed by intermediate routers to recover from packet erasures. Our approach is completely agnostic of the transport

layer of the protected packets. It leverages the SRv6 network programming [34] instead of extending the TCP stack to deploy the solution. Moreover, our work focuses on a design and implementation that can protect traffic at high speed.

Loss recovery. Another option to alleviate losses without FEC is to send *a priori* several times a same packet. Duplicating the traffic would increase too much the bandwidth consumption and CPU usage. Selective Loss Prevention [73] proposes a new algorithm to only duplicate selected packets in a TCP flow, such as packets with the SYN and PSH set. It decreases the overhead and improves the performance in lossy environment. HIRT suggests an underlying-layer-independent tunnel that can recover any packet in the tunnel. The FEC redundancy offers a protection covering a range of packets at the cost of a higher CPU usage to encode and decode the repair symbols.

AC-RLNC [13] proposes an adaptive and causal network coding with feedback algorithm for multipath and multi-hop communications. Using FEC, AC-RLNC adapts its coding rate leveraging feedback sent by the end-host every RTT. Repair symbols are sent *a priori* using this estimation, and *a posteriori* with feedback to reach reliability. No implementation of AC-RLNC could be found for performance comparison.

Congestion avoidance. Microburst is a known cause of burst loss in datacenters [74]. Deflection is a mechanism that temporally changes a packet's port to re-route the packet if it enters a full buffer to prevent a packet loss from such microbursts. Deflection adds reordering as some packets may be re-routed and not subsequent packets of the same flow. Vertigo [75] proposes a transport-independent extension in end-host stacks to recover the original packet order before handing over the packet flow to the transport protocol. Even if this is not directly related to HIRT, it shows that both packet reordering and packet losses occur in datacenters. We suggest an approach that can be combined with Vertigo.

Active networking. Active Congestion Control (ACC) leverages active networking to indicate to each router in the network how to respond to congestion [76]. By actively including the routers in the congestion control, ACC decreases the reaction time in wide area networks. Leveraging a similar mechanism could improve the resilience of HIRT in case of congestion. However, it would require per-flow state to work well with aggregated traffic.

An architecture for active networking suggested the use of pre-defined functions that could be directly called by the application [77]. The authors then proposed to implement congestion-related functions in the network and let the applications choose how the routers should react whether congestion arises. The applications cannot create new functions inside the network but only use the provided API. This approach is similar to SRv6 network programming [34]. HIRT leverages the network programming paradigm to let the applications dynamically ask protection inside the tunnel.

Data Center Networks (DCN). In ReWAN [78], authors suggest a similar idea using inter-DC scheme which implements recovery as a service. End hosts notify losses and get recovered packets. Most practical details, such as which source packets should be snooped to the repair service are left for future work. In a later proposal [79] authors measured inter-DC loss characteristics and pose that a combination of one-hop detour routing and FEC might allow recovering inter-DC losses. This scenario is enabled by our proposal.

CloudBurst [80] proposes to combine the use of FEC and multipath in datacenter networks. They implement a transport library, allowing users to transmit messages over multiple path inside a datacenter network. Our proposal achieves similar goal without host and application modification.

eBPF implementation. We also considered an eBPF implementation of network-layer FEC [81]. Due to the limitations of the eBPF verifier, it was impossible to implement the Random Linear Codes (RLC) in eBPF byte code. The generation of each repair symbol and the creation and solving of linear systems of equations during the recovery required regular communication with a user-space process, which slowed down the performance of the approach. This prototype could only protect traffic at a few megabits per second.

X. Discussion and conclusions

This paper presents HIRT, a transparent robust tunnel leveraging network coding to recover lost packets at high speed between two nodes in a network using an adaptive algorithm. We presented the design and implementation within IPv6 Segment Routing. The adaptive algorithm reacts to the estimated loss patterns to adjust the amount of redundancy packets inside the tunnel. The results showed that HIRT can withstand more than $50\,\mathrm{Gbps}$ of traffic and recover packets in case of severe loss. Thanks to its *stream* abstraction, our implementation scales up with several threads to support even more traffic. We showed through experiences in a real network exposing transmission errors, Starlink, that the tail latency of time-sensitive applications can be decreased.

In future work, we will study how an operator can change the traffic class of repair symbols to ensure it only uses spare bandwidth. Deploying HIRT on L4S-enabled networks also represent interesting perspectives of future work [82]. The ECN-marking behavior of HIRT naturally blends with L4S congestion controllers such as TCP Prague [83]. Losses occurring in non-congested networks can indeed be recovered and ECN-marked with only a negligible impact on the congestion window. Future work also includes evaluating HIRT with real-time applications, web browsing emulation, and comparison with transport-layer FEC such as FlEC [16]. We will also consider how HIRT behaves environments with congestion-induced losses and with fixed wireless access (FWA) mediums.

XI. Acknowledgments

We thank our shepherd, Prof. Eric Rozner, the ICNP reviewers, and Prof. Olivier Bonaventure for their valuable feedback. Louis Navarre is an F.R.S.–FNRS Research Fellow. Tom Barbette worked in part as a beneficiary of a FSR Incoming Post-doctoral Fellowship.

REFERENCES

[1] G. Jake Brutlag, "Speed matters." [Online]. Available: http://googleresearch.blogspot.com/2009/06/speed-matters.html

[2] Akamai, "Akamai online retail performance report: Milliseconds are critical," https://www.akamai.com/uk/en/about/news/press/2017-press/akamai-releases-spring-2017-state-of-online-retail-performance-report.jsp. [Online]. Available: https://www.akamai.com/uk/en/about/news/press/2017-press/akamai-releases-spring-2017-state-of-online-retail-performance-report.jsp

[3] M. A. Siddiqi, H. Yu, and J. Joung, "5g ultra-reliable low-latency communication implementation challenges and operational issues with iot devices," *Electronics*, vol. 8, no. 9, p. 981, 2019.

[4] J. Sachs, G. Wikstrom, T. Dudda, R. Baldemair, and K. Kittichokechai, "5g radio network design for ultra-reliable low-latency communication," *IEEE network*, vol. 32, no. 2, pp. 24–31, 2018.

[5] J. Sachs, L. A. Andersson, J. Araújo, C. Curescu, J. Lundsjö, G. Rune, E. Steinbach, and G. Wikström, "Adaptive 5g low-latency communication for tactile internet services," *Proceedings of the IEEE*, vol. 107, no. 2, pp. 325–349, 2018.

[6] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," RFC 9000, May 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9000

[7] D. V. Paxson and M. Allman, "Computing TCP's Retransmission Timer," RFC 2988, Nov. 2000. [Online]. Available: https://www.rfc-editor.org/info/rfc2988

[8] M. Welzl, "Not a trade-off: On the wi-fi energy efficiency of effective internet congestion control," in *2022 17th Wireless On-Demand Network Systems and Services Conference (WONS)*. IEEE, 2022, pp. 1–4.

[9] F. Michel, M. Trevisan, D. Giordano, and O. Bonaventure, "A first look at starlink performance," in *Proceedings of the 22nd ACM Internet Measurement Conference*, 2022, pp. 130–136.

[10] X. Xu, Y. Jiang, T. Flach, E. Katz-Bassett, D. Choffnes, and R. Govindan, "Investigating transparent web proxies in cellular networks," in *Passive and Active Measurement: 16th International Conference, PAM 2015, New York, NY, USA, March 19-20, 2015, Proceedings 16*. Springer, 2015, pp. 262–276.

[11] A. Badr, A. Khisti, W.-T. Tan, and J. Apostolopoulos, "Perfecting protection for interactive multimedia: A survey of forward error correction for low-delay interactive applications," *IEEE Signal Processing Magazine*, vol. 34, no. 2, pp. 95–113, 2017.

[12] F. Michel, Q. De Coninck, and O. Bonaventure, "Quic-fec: Bringing the benefits of forward erasure correction to quic," in *2019 IFIP Networking Conference (IFIP Networking)*. IEEE, 2019, pp. 1–9.

[13] A. Cohen, G. Thiran, V. B. Bracha, and M. Médard, "Adaptive causal network coding with feedback for multipath multi-hop communications," *IEEE Transactions on Communications*, vol. 69, no. 2, pp. 766–785, 2020.

[14] M. Kim, J. Cloud, A. ParandehGheibi, L. Urbina, K. Fouli, D. Leith, and M. Médard, "Network coded tcp (ctcp)," *arXiv preprint arXiv:1212.2291*, 2012.

[15] J. Detchart, E. Lochin, J. Lacan, and V. Roca, "Tetrys, an on-the-fly network coding protocol," 2015.

[16] F. Michel, A. Cohen, D. Malak, Q. De Coninck, M. Médard, and O. Bonaventure, "Flec: Enhancing quic with application-tailored reliability mechanisms," *IEEE/ACM Transactions on Networking*, 2022.

[17] E. Weldon, "An improved selective-repeat arq strategy," *IEEE Transactions on Communications*, vol. 30, no. 3, pp. 480–486, 1982.

[18] F. Chang, K. Onohara, and T. Mizuochi, "Forward error correction for 100 g transport networks," *IEEE Communications Magazine*, vol. 48, no. 3, pp. S48–S55, 2010.

[19] A. Chockalingam, M. Zorzi, and V. Tralli, "Wireless tcp performance with link layer fec/arq," in *1999 IEEE International Conference on Communications (Cat. No. 99CH36311)*, vol. 2. IEEE, 1999, pp. 1212–1216.

[20] L. Zhang and B. Zhang, "The novel frame boundary detection and fast frame synchronous structure for 10 gb/s ethernet phy fec sub-layer vlsi implementation," in *2011 7th International Conference on Wireless Communications, Networking and Mobile Computing*. IEEE, 2011, pp. 1–4.

[21] K. Fukuchi, D. Ogasahara, J. Hu, T. Takamichi, T. Koga, M. Sato, E. de Gabory, Y. Hashimoto, T. Yoshihara, W. Maeda *et al.*, "112gb/s optical transponder with pm-qpsk and coherent detection employing parallel fpga-based real-time digital signal processing, fec and 100gbe ethernet interface," in *36th European Conference and Exhibition on Optical Communication*. IEEE, 2010, pp. 1–3.

[22] I. Swett, M.-J. Montpetit, and V. Roca, "Coding for quic," 2018.

[23] V. Roca, I. Swett, and M.-J. Montpetit, "Sliding window random linear code (rlc) forward erasure correction (fec) schemes for quic," 2018.

[24] H. Lundqvist and G. Karlsson, "Tcp with end-to-end fec," in *International Zurich Seminar on Communications, 2004*. IEEE, 2004, pp. 152–155.

[25] Q. De Coninck, F. Michel, M. Piraux, F. Rochet, T. Given-Wilson, A. Legay, O. Pereira, and O. Bonaventure, "Pluginizing quic," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 59–74.

[26] A. Cohen, D. Malak, V. B. Bracha, and M. Médard, "Adaptive causal network coding with feedback," *IEEE Transactions on Communications*, vol. 68, no. 7, pp. 4325–4341, 2020.

[27] L. Rizzo and L. Vicisano, "Rmdp: An fec-based reliable multicast protocol for wireless environments," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 2, no. 2, pp. 23–31, 1998.

[28] W.-T. Tan and A. Zakhor, "Video multicast using layered fec and scalable compression," *IEEE Transactions on circuits and systems for video technology*, vol. 11, no. 3, pp. 373–386, 2001.

[29] M. Luby, L. Vicisano, J. Gemmell, L. Rizzo, M. Handley, and J. Crowcroft, "The use of forward error correction (fec) in reliable multicast," RFC 3453, December, Tech. Rep., 2002.

[30] T. Wirtgen, Q. De Coninck, R. Bush, L. Vanbever, and O. Bonaventure, "Xbgp: When you can't wait for the ietf and vendors," in *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, 2020, pp. 1–7.

[31] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois, "The segment routing architecture," in *2015 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2015, pp. 1–6.

[32] C. Filsfils, D. Dukes, S. Previdi, J. Leddy, S. Matsushima, and D. Voyer, "IPv6 Segment Routing Header (SRH)," RFC 8754, Mar. 2020. [Online]. Available: https://rfc-editor.org/rfc/rfc8754.txt

[33] H. Tian, C. Xie, E. Chingwena, S. Peng, and Q. Gao, "SRv6 Deployment Consideration," Internet Engineering Task Force, Internet-Draft draft-tian-spring-srv6-deployment-consideration-08, Feb. 2024, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-tian-spring-srv6-deployment-consideration/08/

[34] C. Filsfils, P. Camarillo, J. Leddy, D. Voyer, S. Matsushima, and Z. Li, "Segment Routing over IPv6 (SRv6) Network Programming," RFC 8986, Feb. 2021. [Online]. Available: https://rfc-editor.org/rfc/rfc8986.txt

[35] T. Berger, "Analysis of current vpn technologies," in *First International Conference on Availability, Reliability and Security (ARES'06)*. IEEE, 2006, pp. 8–pp.

[36] J. Bi, J. Wu, and X. Leng, "Ipv4/ipv6 transition technologies and univer6 architecture," *International Journal of Computer Science and Network Security*, vol. 7, no. 1, pp. 232–243, 2007.

[37] D. Schinazi, "The MASQUE Protocol," Internet Engineering Task Force, Internet-Draft draft-schinazi-masque-protocol-03, Mar. 2021, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-schinazi-masque-protocol-03

[38] M. Piraux and O. Bonaventure, "Tunneling internet protocols inside quic," Internet-Draft draft-piraux-quic-tunnel-01. IETF Secretariat. https://tools ..., Tech. Rep., 2020.

[39] G. Armitage, "Mpls: the magic behind the myths [multiprotocol label switching]," *IEEE Communications Magazine*, vol. 38, no. 1, pp. 124–131, 2000.

[40] J. Prados-Garzon, O. Adamuz-Hinojosa, P. Ameigeiras, J. J. Ramos-Munoz, P. Andres-Maldonado, and J. M. Lopez-Soler, "Handover implementation in a 5g sdn-based mobile network architecture," in *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. IEEE, 2016, pp. 1–6.

[41] V. Roca and B. Teibi, "Sliding Window Random Linear Code (RLC) Forward Erasure Correction (FEC) Schemes for FECFRAME," RFC 8681, Jan. 2020. [Online]. Available: https://rfc-editor.org/rfc/rfc8681.txt

[42] T. Barbette, C. Soldani, and L. Mathy, "Fast userspace packet processing," in *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, 2015, pp. 5–16.

[43] M. Balakrishnan, T. Marian, K. P. Birman, H. Weatherspoon, and L. Ganesh, "Maelstrom: Transparent Error Correction for Communication Between Data Centers," *IEEE/ACM Transactions on*

Networking, vol. 19, no. 3, pp. 617–629, Jun. 2011. [Online]. Available: http://ieeexplore.ieee.org/document/5765714/

[44] V. Roca, J. Peltotalo, J. Lacan, and S. Peltotalo, "Reed-Solomon Forward Error Correction (FEC) Schemes," RFC 5510, Apr. 2009. [Online]. Available: https://www.rfc-editor.org/info/rfc5510

[45] M. Piraux, L. Navarre, N. Rybowski, O. Bonaventure, and B. Donnet, "Revealing the evolution of a cloud provider through its network weather map," in *Proceedings of the 22nd ACM Internet Measurement Conference*, 2022, pp. 298–304.

[46] S. Floyd, D. K. K. Ramakrishnan, and D. L. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168, Sep. 2001. [Online]. Available: https://www.rfc-editor.org/info/rfc3168

[47] N. Kuhn, E. Lochin, F. Michel, and M. Welzl, "Forward Erasure Correction (FEC) Coding and Congestion Control in Transport," RFC 9265, Jul. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9265

[48] segment-routing.net. [Online]. Available: https://www.segment-routing.net

[49] C. Huitema, "The case for packet level fec," in *International Workshop on Protocols for High Speed Networks*. Springer, 1996, pp. 109–120.

[50] J.-C. Bolot, S. Fosse-Parisis, and D. Towsley, "Adaptive fec-based error control for internet telephony," in *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, vol. 3. IEEE, 1999, pp. 1453–1460.

[51] J.-S. Ahn, S.-W. Hong, and J. Heidemann, "An adaptive fec code control algorithm for mobile wireless sensor networks," *Journal of Communications and Networks*, vol. 7, no. 4, pp. 489–498, 2005.

[52] V. Roca, B. Teibi, C. Burdinat, T. Tran, and C. Thienot, "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, 2017, pp. 1–8.

[53] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[54] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 3, pp. 263–297, 2000.

[55] L. Foundation, "Data plane development kit (DPDK)," 2015. [Online]. Available: http://www.dpdk.org

[56] Moepgf. [Online]. Available: https://github.com/moepinet/libmoepgf.git

[57] T. Barbette, G. P. Katsikas, G. Q. Maguire Jr, and D. Kostić, "Rss++ load and state-aware receive side scaling," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, 2019, pp. 318–333.

[58] J. Bellardo and S. Savage, "Measuring packet reordering," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, 2002, pp. 97–105.

[59] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Ricart, L. Landweber, C. Elliott, M. Zink, E. Cecchet, S. Kar, and P. Mishra, "The design and operation of CloudLab," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, Jul. 2019, pp. 1–14. [Online]. Available: https://www.flux.utah.edu/paper/duplyakin-atc19

[60] "Starlink," 2023. [Online]. Available: https://www.starlink.com

[61] M. M. Kassem, A. Raman, D. Perino, and N. Sastry, "A browser-side view of starlink connectivity," in *Proceedings of the 22nd ACM Internet Measurement Conference*, 2022, pp. 151–158.

[62] S. Ma, Y. C. Chou, H. Zhao, L. Chen, X. Ma, and J. Liu, "Network characteristics of leo satellite constellations: A starlink-based measurement from end users," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE, 2023, pp. 1–10.

[63] D. Perdices, G. Perna, M. Trevisan, D. Giordano, and M. Mellia, "When satellite is all you have: watching the internet from 550 ms," in *Proceedings of the 22nd ACM Internet Measurement Conference*, 2022, pp. 137–150.

[64] J. A. Donenfeld, "Wireguard: next generation kernel network tunnel." in *NDSS*, 2017, pp. 1–12.

[65] wrk. [Online]. Available: https://github.com/wg/wrk

[66] T. Barbette, E. Wu, D. Kostić, G. Q. Maguire, P. Papadimitratos, and M. Chiesa, "Cheetah: A high-speed programmable load-balancer framework with guaranteed per-connection-consistency," *IEEE/ACM Transactions on Networking*, vol. 30, no. 1, pp. 354–367, 2021.

[67] "picoquic," 2023. [Online]. Available: https://github.com/private-octopus/picoquic

[68] E. Blanton, D. V. Paxson, and M. Allman, "TCP Congestion Control," RFC 5681, Sep. 2009. [Online]. Available: https://www.rfc-editor.org/info/rfc5681

[69] fio. [Online]. Available: https://github.com/axboe/fio

[70] G. Haßlinger and O. Hohlfeld, "The gilbert-elliott model for packet loss in real time services on the internet," in *14th GI/ITG Conference-Measurement, Modelling and Evalutation of Computer and Communication Systems*. VDE, 2008, pp. 1–15.

[71] R. E. Kirk, *Experimental design: Procedures for the behavioral sciences*. Sage Publications, 2012.

[72] J. K. Sundararajan, D. Shah, M. Médard, S. Jakubczak, M. Mitzenmacher, and J. Barros, "Network coding meets tcp: Theory and implementation," *Proceedings of the IEEE*, vol. 99, no. 3, pp. 490–512, 2011.

[73] Z. Zhou and X. Yang, "Speeding up tcp with selective loss prevention," in *2021 IEEE 29th International Conference on Network Protocols (ICNP)*. IEEE, 2021, pp. 1–6.

[74] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 2010, pp. 267–280.

[75] S. Abdous, E. Sharafzadeh, and S. Ghorbani, "Burst-tolerant datacenter networks with vertigo," in *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, 2021, pp. 1–15.

[76] T. Faber, "Acc: using active networking to enhance feedback congestion control mechanisms," *IEEE network*, vol. 12, no. 3, pp. 61–65, 1998.

[77] S. Bhattacharjee, K. L. Calvert, and E. W. Zegura, "An architecture for active networking," in *International Conference on High Performance Networking*. Springer, 1997, pp. 265–279.

[78] O. Haq and F. R. Dogar, "Leveraging the power of cloud for reliable wide area communication," in *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XIV. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2834050.2834109

[79] O. Haq, M. Raja, and F. R. Dogar, "Measuring and improving the reliability of wide-area cloud paths," in *Proceedings of the 26th International Conference on World Wide Web*, ser. WWW '17. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2017, p. 253–262. [Online]. Available: https://doi.org/10.1145/3038912.3052560

[80] G. Zeng, L. Chen, B. Yi, and K. Chen, "Optimizing tail latency in commodity datacenters using forward error correction," *CoRR*, vol. abs/2110.15157, 2021. [Online]. Available: https://arxiv.org/abs/2110.15157

[81] L. Navarre, F. Michel, and O. Bonaventure, "Srv6-fec: bringing forward erasure correction to ipv6 segment routing," in *Proceedings of the SIGCOMM'21 Poster and Demo Sessions*, 2021, pp. 45–47.

[82] B. Briscoe, K. D. Schepper, M. Bagnulo, and G. White, "Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service: Architecture," RFC 9330, Jan. 2023. [Online]. Available: https://www.rfc-editor.org/info/rfc9330

[83] K. D. Schepper, O. Tilmans, and B. Briscoe, "Prague Congestion Control," Internet Engineering Task Force, Internet-Draft draft-briscoe-iccrg-prague-congestion-control-01, Jul. 2022, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-briscoe-iccrg-prague-congestion-control/01/