**Blank Space Code:**

This repository contains the implementation of the Blank Space AC-RLNC (BS) algorithm, a novel Adaptive and Causal Network Coding solution designed to mitigate the triplet trade-off between throughput-delay-efficiency in multi-hop networks. The full algorithm is described in our paper: [Blank Space: Adaptive Causal Coding for Streaming Communications Over Multi-Hop Networks]

<https://arxiv.org/abs/2502.11984>

**Document overview:**

… - Parameters set at configuration. If not specified otherwise, refer to utils\config.py. **If a parameter is not mentioned here as an input parameter, do not touch its default value.**

… - Important note.

**To run the code:**

*Direct instructions. More details for each file are explained in the Code section.*

Initial Setup:

1. Generate Data for channels using erasure\_generator\erasure\_generate.py
2. Generate Bernoulli inputs in erasure\_generator\ber\_input\_series.py
3. Set parameters in config\config.py
   1. Results Folder
   2. Project Folder

For Running:

1. Set running type in run\run\_all.py\run\_main() function – this is the “main”.
2. Set parameters in config\config.py

Note:

1. For each run you must choose another run name.   
"results\_filename\_base": "run1", # results filename - must change each run

2. Plot\_results for paper – choose the "results\_filename\_base".

In this version: Must be all three: MIXALL, BS-EMPTY, AC-FEC

In this version: Must have SR-ARQ results. – not in github.

**Project Overview:**

---- Project

----- Data

----- PythonProject

----- Results

**Code:**

1. utils - configuration setup
   1. config.py – **in “param“ - input parameters.**
   2. **A screenshot of a computer

      AI-generated content may be incorrect.**config\_setup.py – conversion from json. Acknowledge this if adding a new field (like “param” or “run\_index”) in the CFG dictionary.
2. run – main running functions
   1. **run\_1.py** – runs one simulation and saves results in Results folder.
   2. **run\_all.py** – runs rep (repeats) times simulation of the same configuration but different noise realizations (Monte-Carlo).
   3. **data\_storage.py** – class to store data
   4. **plot\_results.py**
   5. *Set\_sim\_params – isn’t used in this version.*
3. ns – network system - The actual system – nodes, buffers, packets generating and transmission. – explained separately in docs and examples.
4. erasures\_generator – generates binary series according to the given parameter.
   1. This is a standalone script. Define parameters inside the script and run it by itself.
   2. Options:
      1. BEC Channels
      2. Gilbert Eliot
   3. Format:
      1. Choose the number of channels.
      2. Choose erasure rates.
      3. Choose the main data path.
      4. Automatically save the results inside the folder *“main\_path\Type\ch\_n\”.* Later these series can be the erasure series for each channel and are read accordingly! Using "er\_load": "from\_csv".More on these options in Data section.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

1. acrlnc\_node – AC-RLNC encoder and decoder operations – Explained later.

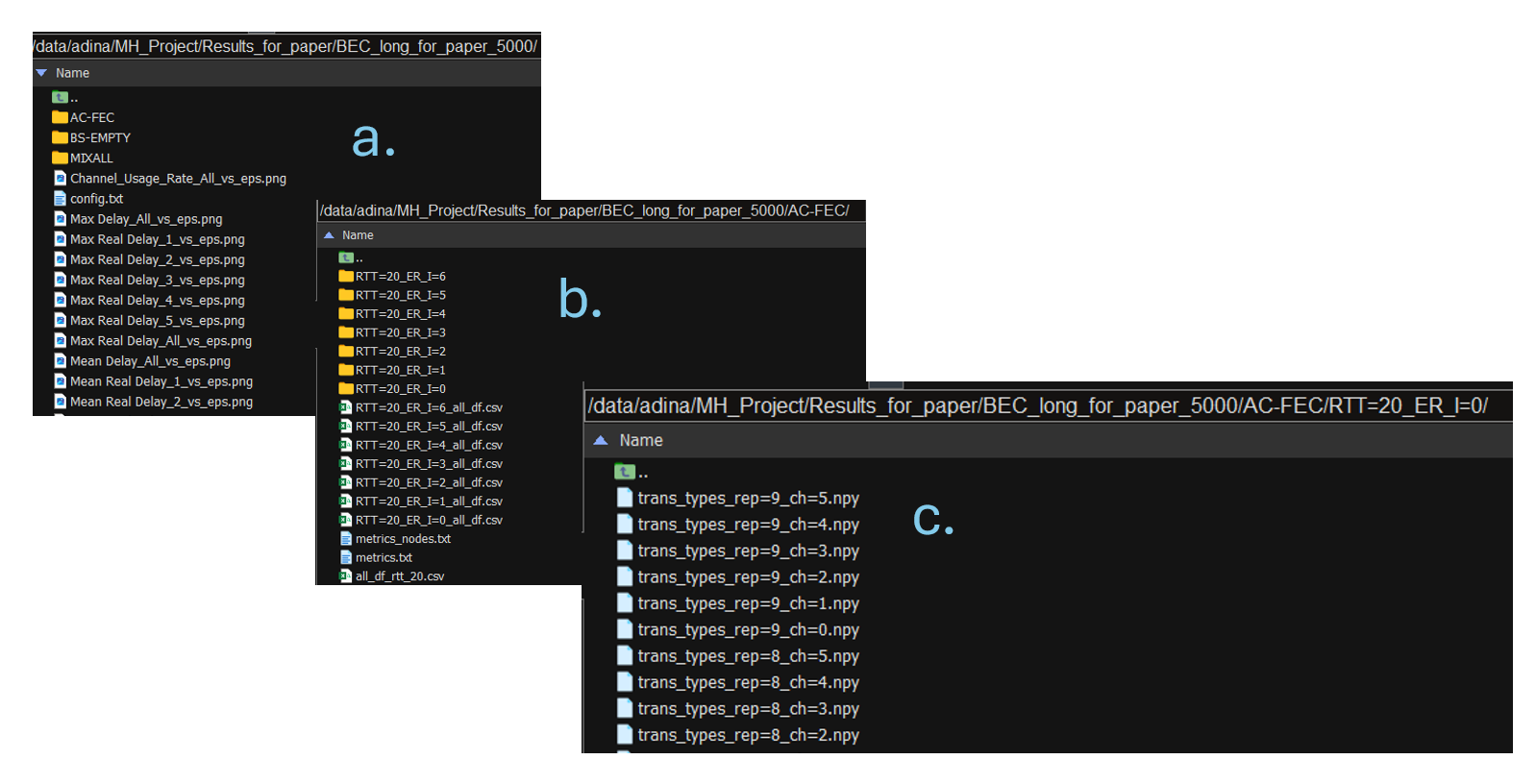
**Data**

1. Information data - No actual data is transmitted. The packets are represented by indices. More on this representation later.
2. Erasure Channels – represented by binary series where 0=erasure, 1=reception.
3. Pre-determined erasure series should be in a separate folder “Data”: A black screen with white text

   AI-generated content may be incorrect.
4. Optional series:
   1. BEC distribution.
   2. Gilbert Eliot distribution.
   3. Pre-determined erasure series (From Data files).
   4. Generate erasure realization on the fly.

**Results:**

1. All run results are saved automatically in the “Results” folder.
   1. For each configuration
   2. For each *changing erasure rate* – explained later.
   3. For each repetition for each channel.
   4. Not working but exists: SNR data with pre-defined threshold.



**System Overview:**

A diagram of a diagram

AI-generated content may be incorrect.

1. N node network.
   1. Data flows between every two nodes.
   2. Data from the source is intended for the destination.
   3. A multicast scenario where all intermediate nodes read information is available, as explained below.
2. Slotted communication
   1. Time horizon – T
   2. Operations are handled as one per time slot.
3. Local
   1. In this version: all channels use the same one.
   2. The AC-RLNC codes do not support otherwise*. If changing this - be cautious when considering feedback and window management.*
   3. 1 time slot delay – generating a packet operation takes 1 time slot. Hence, for N nodes, with local RTT, the actual RTT is , 1 for the forward packet and 1 for its feedback packet.
4. Forward Erasure channels.
   1. Erasure rate for each channel.
   2. Set the erasure rates, this also set the number of nodes.
      1. "er\_rates": [0.1, 0.2, 0.5, 0.4, 0.3]
   3. Set a channel with varying epsilon to see different behaviors and bottlenecks.
      1. "er\_var\_ind": [1]
      2. "er\_var\_values": [0.2, 0.5, 4], Varying range: - start, end, steps number
      3. This OVERWRITES the epsilon set at er\_rates.
      4. Cannot be empty but can be one value.
   4. Gilbert Eliot Channel:
      1. An option for one GE channel with fixed parameters, load from Data Folder.
      2. Set in the erasure type with the following format:  "er\_type": "BEC", # erasure type: BEC, GE\_p\_0.01\_g\_0.1\_b\_1,
      3. Choose the GE channel index: "ge\_channel": [1],  # GE channel index
      4. (If type is BEC, ge\_channel is ignored)
   5. Source Packets:
      1. Option 1: source packets are generated all the time (Once per time slot).
      2. Option 2: They arrive according to a Bernoulli distribution – must be from a saved data folder. The folder name is determined in run\_all.py\run\_all(): (line 184)

ber\_events = os.path.join(cfg.param.project\_folder, "Data", *f*"ber\_series\_less", "rate\_{np.round(1-max(cfg.param.er\_rates), 2)}", *f*"series\_{r+1}.txt")

* + 1. "in\_type": "ber", # input type: ber, all

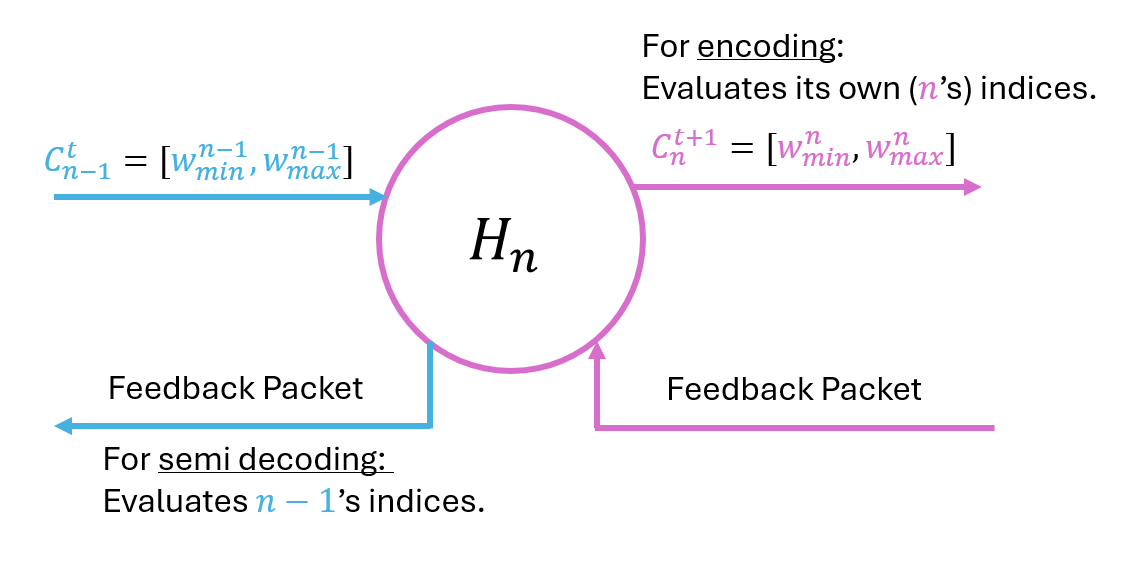
1. Reliable feedback channels.

**Data and Packets Representation:**

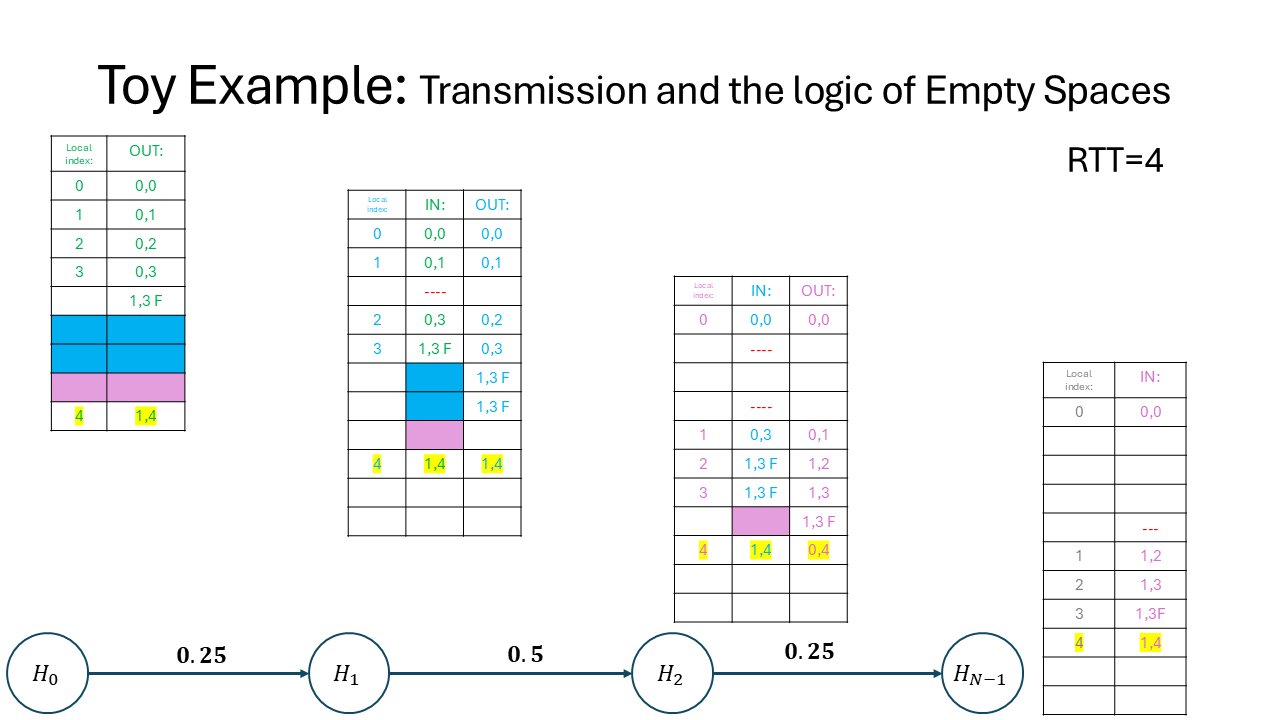
1. Packets are represented by a local index at each node.
2. The local indices of the source node are the **information packets** intended to be read at the destination.
3. A linear combination of packets is represented by parentheses [w\_min, w\_max] where:
   1. w\_min = first information packet (its index) in the coded combination.
   2. w\_max = last information packet (its index) in the coded combination.
   3. w\_min is incremented according to feedback of decoded packets.
   4. w\_max is determined by the encoder.
   5. Note that the buffer may contain information packets with indexes greater than w\_max.

**Nodes Operation - General:**

1. Note differences in handling indices:
   1. For semi-decoding: Prev node indices.
   2. For encoding: Current node indices.
2. To have all encoders implemented in the same manner, the source and destination are implemented by an extra naïve node (indicated by rectangles in the system image).

****

**Transmission Toy Example:**



**Nodes Operation – In Code:**

**Files:**

1. ns/nc\_enc.py –
   1. Connection between the general code and the encoder.
   2. Saves the results. 🡪 May be not optimal (was improved) for run time.
2. acrlnc\_node options:
   1. decoder
   2. eps\_est
      1. Holds all past feedback and estimate the erasure rate.🡪 May be not optimal (was improved) for run time due to buffer operations.
      2. Set in config: "er\_estimate\_type": "stat",  # genie, stat, stat\_max, oracle
      3. genie option – knows the epsilon\_n but not the actual realization
      4. oracle option – knows the actual realization.
      5. Stat option – mean over the feedback.
      6. Stat\_max – mean+std, Need to check if this works. Also uses: "sigma": 1,  # noise variance
   3. encoder.py +ac\_node.py
      1. Implements NET AC-RLNC at each node.
   4. encoder\_mix\_all.py + ac\_node\_mix\_all.py
      1. Implement AC-R LNC at the source only ()
      2. Any intermediate node “mix all” the packets in the buffer and forwards them. Hence, encoder\_mix\_all is pretty degenerated.
   5. Configuration Options – Set at run\_all/main\_run()
      1. MIXALL – calls the mix\_all versions.
      2. All other options call the other option with different operations:
         1. AC-FEC – No empty (=pause) transmissions at all.
         2. AC-EMPTY – Empty-Buffer mechanism (No-New No-FEC in the paper) is enabled, BS operation is stopped.
         3. BS-FEC – BS enabled, No-New No-FEC is stopped.
         4. BS-EMPTY – Both pause mechanisms are enabled.

**Operation: ac\_node\ ac\_node\_mix\_all**

1. Each packet contains fields with information:
   1. nc\_header = - indices of the previous node.
   2. nc\_id = local id
      1. assigned when inserted to the buffer.
      2. update\_pt\_buffer\_add(self, in\_packet\_info, in\_packet\_recep\_flag)
   3. src: who is the previous node, used to set the current node number .
   4. FEC Type: the transmission type
      1. NEW
      2. FEC
      3. FB-FEC
      4. EOW
      5. EMPTY-BUFFER
      6. EPMTY-BLS
      7. output\_packet\_processing(self, in\_packet\_recep\_flag, fb\_packet, in\_packet\_info)
2. Each node holds a buffer of incoming packets.
3. Discard decoded packets.
   1. update\_pt\_buffer\_discard(self, fb\_packet)
4. Add packets that hold new information. That is – NEW or any fec option that is needed (By checking the DoF amount).
   1. update\_pt\_buffer\_add(self, in\_packet\_info, in\_packet\_recep\_flag)
      1. When information packets arrive at the source all the time update\_pt\_buffer\_add\_all(self, in\_packet\_info, in\_packet\_recep\_flag)
      2. When information packets arrive at the source according to Bernoulli distribution. Implementation: packets are generated all the time, but this indicates to avoid this one according to the ber\_series input.

update\_pt\_buffer\_add\_ber(self, in\_packet\_info, in\_packet\_recep\_flag)

* 1. accept\_fec(self)

1. Send Feedback
   1. If semi-decoding happened: ack=1, and dec\_id=last decoded packet (in indices).
   2. ack\_id isn’t actually used.
   3. *dec\_id* is set with *-1* and later is incremented by 1 because I put it in the “ *self*.out\_fb.nc\_serial” field (also called nc\_header). *Can be modified with care.*
   4. decode\_and\_create\_fb(self, in\_packet\_recep\_flag, fb\_packet)

**Operation: encoder\ encoder\_mix\_all**

**Parameters Summary:**

**Initialize Parameters**

**User input**

**Place holders for run, do not touch.**

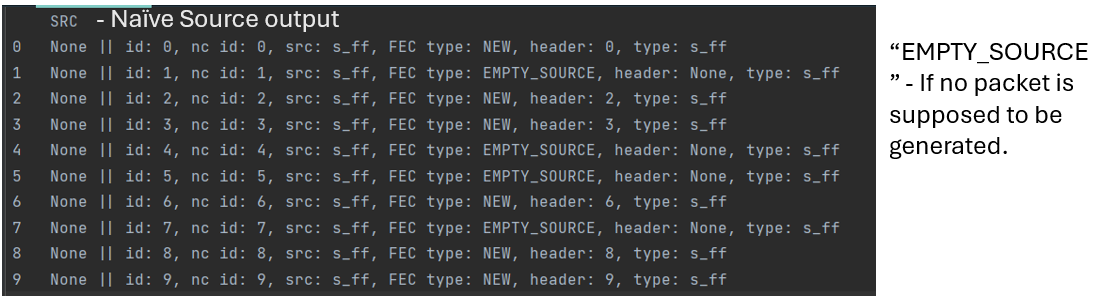
"project\_folder": r"C:\Users\adina\Technion\Research\MH\_Project\Code", # Code folder (And data)  
"results\_folder": r"C:\Users\adina\Technion\Research\MH\_Project\Code\Results", # results folder

"rtt": [20], # rtt – Only 1 works (For example [10,20] will not work).  
"results\_filename\_base": "run1", # results filename - must change each run  
"T": 1000, # number of timesteps in each rep  
"debug": False, # debug flag – Not sure if it is working. Only on some parts.  
"rep": 5, # number of repetitions (Monte Carlo)  
"er\_rates": [0.1, 0.2, 0.5, 0.4, 0.3], # erasure rate of each channel: 0=perfect, 1=all erasure  
"er\_var\_ind": [1], # Indices to vary - can't be empty. To have all channels at fixed rate - choose one index and set er\_var\_values to be one value.  
"er\_var\_values": [0.2, 0.3, 2], # Values to vary - same to all indices - start, end, steps number - overrites er\_rates  
"er\_load": "from\_csv", # erasure type: erasure, from\_csv – erasure: random in running, from\_csv – from Data files. (If want to debug on “erasure” : uncomment # random.seed(0) # Seed for debug – line 64 in ns\port\airinterface. Careful: Will use the seed for all repetitions.)  
"er\_type": "BEC", # erasure type: BEC, GE\_p\_0.01\_g\_0.1\_b\_1, GE\_p\_0.1\_q\_0.3\_g\_0.01\_b\_0.8, For the GE only one channel is going to be GE. GE\_p\_0.01\_q\_0.02\_g\_0.1\_b\_1  
"ge\_channel": [1], # GE channel index  
"er\_estimate\_type": "stat", # genie, stat, stat\_max, oracle  
"sigma": 1, # noise variance  
"print\_flag": False, # print flag, False, True – Plot running log.  
"in\_type": "ber", # input type: ber, all

"results\_filename": "",  
"er\_series\_path": "", # erasure series path  
"prot\_type": "", # "MIXALL", "BS", "AC"  
"data\_storage": None, # data storage

**Transmission Logs:**

**ff\_log.txt inside RTT=20\_ER\_I=0 folder:**



A screen shot of a computer

AI-generated content may be incorrect.

**Metrics.txt:**

results in the destination node for each repetition.

The mean+std for all results are in the end.

Delay \ NC Delay = delay with respect to time **leaving** the source node

Real Delay = delay with respect to time **arriving** the source node

Node-1 = the destination node.

**Metrics\_nodes.txt:**

The metrics are calculated at each node.

The mean values are in the metrics.txt file (With their index).

**SR-ARQ Results – Calculated separately in MATLAB, are not on GitHub right now.**