An SD-WAN Controller for Delay Jitter Minimization in Coded Multi-access Systems

Ahtisham Ali Ansari, Sri Pramodh Rachuri, Arzad A. Kherani
Department of Electrical Engineering and Computer Science,
Indian Institute of Technology Bhilai, India
ahtishama, rachuris, arzad.alam @iitbhilai.ac.in

Deepaknath Tandur
ABB Corporate Research,
Bangalore, India
deepaknath.tandur@in.abb.com

***Abstract*—For an industrial network that is expected to perform with high reliability and availability, we consider the problem of designing a low-cost approach to continuous monitoring and maintenance of the communication links. A software defined wide area network (SD-WAN) based solution is proposed to adaptively monitor and manage the connectivity links from an end device as per the application requirement. Each such device may have multiple cellular interfaces, and we control the end-to-end delay jitter variation across the different interfaces via an intelligent traffic splitting scheme. We propose use of inter-stream coding to achieve target delay jitter and also a high reliability/availability. We use a decoupling approach to adapt the probabilistic traffic split into various interfaces and the extent of inter-stream coding. We provide an end-to-end measurement-based traffic splitting scheme that relies on a Machine Learning algorithm. We use a stochastic approximation-like algorithm (operating at a slower timescale) to obtain the right coding level. The various modules developed are pluggable in a pipeline-manner and work with real interfaces as well as simulators like openairinterface. We validate our analysis, and provide traffic split performance results from our working multi-access system (using cellular dongles, and also over openairinterface).**

***Index Terms*—SD-WAN, Multi-access systems, Delay jitter, LTE**

I. Introduction

A multi-access system opens up the possibility of using the various available interfaces to increase the aggregate data transfer rate by pumping more and more data into the available interfaces. This has led to several commercial solutions available in the market where the control of traffic into the various available interfaces, and other such parameters, is achieved using a software defined wide area network (SD-WAN) approach.

Emergence of SD-WAN on fixed (enterprise) operators has had significant effects on the market for MPLS VPNs, enabling businesses to bond together normal Internet connections and small-capacity MPLS links. The success of such solutions leads one to wonder about using such solutions with only cellular links.

Most of the multi-acess SD-WAN solutions available in the market [1] are proprietary in nature and vary in their implementation. The encapsulation of the traffic can be implemented at different layers of the communication stack, from MAC to application layer [2]. However, all the solutions utilize some sort of policy management by a central controller. Software-based combining of the multiple connections has been at the heart of the success story. This SD-WAN approach is used to, for example, bond the various interfaces, among various other capabilities. The main advantage of using the SD-WAN approach to aggregate multiple legacy ISP broadband lines seems to be coming from the relatively low cost when compared to an MPLS-like setup.

The available enterprise solutions have mostly used the multi-access capability to increase the throughput performance of the system. However, one can also look at the problem of improving the end-to-end delay and delay jitter performance for static applications that require such guarantees. Such applications may generate very small traffic volume, but require stricter delay or delay jitter performance. Exploring the viability of cellular links-based multi-access systems to achieve such performance is required. Traffic sources generating higher data rates may also benefit from such solutions in a mobile environment.

In [3], we worked on delay minimization in Multi-Access Sytems by adding redundancy. In this paper we propose a inter-stream coding based multi-access SD-WAN system in which multiple cellular communication interfaces are utilized to send data with added redundancy in order to improve the overall end-to-end delay jitter performance. Voice and video transmission quality suffer when the information sent over the network is spaced inconsistently leading to a variable tempo for the stream. This is more so in the wireless environment, as the channel conditions may vary very quickly. The result is audio or video that can have gaps in timing and become impaired. The proposed method measures these gaps between the packets and can evenly space these packets on the other side providing what is called a ”jitter buffer” to realign the timing of these packets to keep the video or audio stream cadence intact. Jitter buffering has been performed before but traditionally at the application servers and endpoints (i.e. IP phones or IP video appliances). The unique differentiation here is performing this inline on the network with multiple communication interfaces at the end points. The paper proposes the approach which is easily scalable based on the requirements.

II. Inter-Stream Coding in Multi-Access Systems

Consider the simple case where the individual per-packet delay on the various uplink cellular interfaces are independent random variables, say $X_i(n)$ for the n^{th} packet on interface $i \in \{1, \ldots, N\}$. Considering the simplest case where the same packet is replicated across all the interfaces (simultaneously) and the receiver considers the first received packet and discards the duplicate packets that come later. The end-to-end delay seen by the n^{th} packet from the source stream is then $\min_{1 \leq i \leq N} X_i(n)$, which is known to converge to a deterministic value as N increases.

In the above discussion we did not take into account the traffic generation rate of the source. Cellular interfaces pose several issues where the per-packet delay seen by the traffic stream is a function of the input rate into the interface. This happens due to the complex interaction between the MAC layer and Physical layer in data transmission in LTE, i.e., the interaction between the transport block size allocation (which could cover multiple IP packets) and the HARQ mechanism to recover from wireless losses. This is illustrated in Figure 1 which provides a detailed description of the various layers (LTE) involved in cellular multi-access system that we are considering.

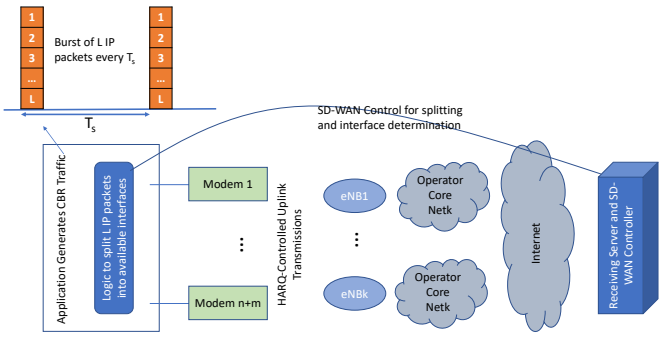

Fig. 1. SD-WAN Control Options for Multi-Access.

Further, when the traffic generation rate is large enough to rule out the use of pure replication as mentioned above, Inter-stream coding can be used for reliability and delay performance improvement in the following manner. Out of the $n+m$ interfaces, registered on possibly different operator networks, we use m packets from the data stream to generate additional n coded packets and then we send the original data packets individually on m interfaces, while the remaining n interfaces are used to send coded packets. We use Fountain codes [4] so that the receiver can recover m packets from any m of the $n+m$ streams. Fountain codes, being rateless codes, can generate unlimited number of encoded symbols with given set of source symbols. And the original source symbols can be recovered from any subset of the encoded symbols, given the size of the subset should be equal to or only slightly larger than the number of source symbols. A fountain code is considered as optimal if the original k source symbols can be recovered from any subset of encoding symbols of size k. We are using systematic fountain codes, in which original symbols are also transmitted along with the encoded symbols. Systematic fountain coding allows us to read original messages without any decoding.

The framework presented in this paper can be used in multiple ways in practice, for example,

- When the source traffic generation rate R is small, it is intuitive that the network coding approach boils down to a pure replication code. This is the case for example when a single data packet is sent infrequently, so that $m = 1$ is a virtually forced constraint. This scenario is of practical importance as it helps a utility provider to reduce the MPLS-like subscription cost. For such scenario, we can have very large n, which essentially implies a very small deterministic delay, thus providing a delay and jitter guarantee that is comparable to those obtained using expensive approaches.
- Using an LTE system, while the commercial roll-out of uRLLC solutions are awaited, would lead to providing an LTE-specific guarantee on delays. Redundancy clearly improves the reliability performance, while achieving a smaller-than-single-interface-LTE delay. This is because in standard LTE system, the time required for a schedulign request to uplink allocation to actual transmission could be lower bounded by $10ms$. While having multiple interfaces helps us reduce the time that one needs to wait for the scheduling request opportunity, thus providing a virtually deterministic uplink transmission time of $10ms$.

When using multiple parallel LTE uplink connections (via multiple LTE dongles), the following problems fall in the purview of SD-WAN:

P.1 Controlling the number of interfaces to be used for a given stream

P.2 Controlling the redundancy level to achieve a target delay jitter performance

P.3 Controlling the split of traffic into the available interfaces

A. A Decoupling

We do not consider the problem of adapting the number of interfaces (i.e., problem **P.1**). However, we follow an approach of decoupling the problems **P.2** and **P.3**.

Since the network operating point (including the load on the eNB, etc.) would vary with time, we provide a learning algorithm that continuously adapts the parameters used to achieve **P.2** and **P.3**.

When the number of interfaces available is given, say N, the problem of **P.2** is essentially that of finding the right split of N into two quantities m and n such that m is the number of original packets used and n is the number of coded packets generated per m original packets. This means that if the overall traffic generation rate is R bits per second, then the actual total load on the N interfaces is $N \times R/m$.

The approach of decoupling the problems of **P.2** and **P.3** would mean that the traffic split into the N interfaces is

oblivious to the nature of the packet, i.e., which packet is original or which packet is coded. This means that we find an $N-$dimensional probability (row) vector $\mathbf{p} = (p_1, \ldots, p_N)$ such that $\mathbf{p} \cdot \mathbb{1} = 1$, where $\mathbb{1}$ is a column vector of all ones. The interpretation of p_i is that of the probability that a packet coming into the splitter function is sent over interface number i. It is to be observed that we are implicitly forgetting the nature of the packet, i.e., original or encoded, when using this probabilistic split. One could envision multiple variants of this logic where awareness of the nature of the packet is included in the splitting logic, however, we do not address those aspects in the current work.

III. The Learning Algorithms

The learning algorithm used in this paper can be thought of as a two-timescale stochastic approximation algorithm where the redundancy (n) is decided on a slower time scale compared to that of the splitting probability. We provide a brief description of these algorithms in the below and detailed implementation description is provided later in the paper.

A. Splitting

At the k^{th} update occasion, the value of $\mathbf{p}(k)$ is changed as follows:

$$\mathbf{p}(k+1) = \mathbf{p}(k) + \epsilon \frac{(\mathbf{p}(k) - softMax(k))}{||(\mathbf{p}(k) - softMax(k))||}.$$

This approach is used to move the operating point over the $N-$dimensional probability simplex. $softMax(k)$ is the output of the softMax algorithm at the k^{th} update instant, and takes into account the estimate of the delay jitter for each of the interfaces. Here $||\cdot||$ is the standard L_2 norm. The approach here is to move very small amount (ϵ) in the direction towards $softMax(k)$ vector.

B. Redundancy

In our recent paper (citation suppressed due to double blind review), we showed that for a given n, the end-to-end per-packet delay would be a decreasing function of m. Unfortunately, no such structural result is available for the case of delay jitter when the constraint of $n + m = N$ is imposed. Thus, it is not straightforward to know the favorable direction in which the value of n should be moved based on the delay jitter observations. Under such cases, we use the following intuition to guide us:

1) for a given operating scenario (i.e., a given network load and radio conditions for an eNB), there is an optimal value m^* such that the end-to-end delay jitter is minimized for $m = m^*$. This implicitly assumes that the splitting probabilities are optimally selected for each operating value of m, i.e., the faster time-scale of splitting probability adaptation is used. We also assume that the delay jitter as a function of m is a unimodal min function. Even though it looks intuitive that using a value of $m = 1$ would provide the best delay jitter, one has to note that the aggregate rate into the various interfaces will increase with a decrease in m, thus countering the beneficial effect of decreasing m.

Thus, when we get the estimates of delay jitter ($J(t)$) at two successive update instants, we use the following equation to change the value of m:

$$m(t+1) = m(t) + \delta(t)$$

where $\delta(t)$ is obtained using Table I. Here θ is a tolerance level to avoid ping-pong effect.

$m(t) - m(t-1)$	$J(t) - J(t-1)$	$\delta(t)$
> 0	$< -\theta$	1
> 0	$> \theta$	-1
< 0	$< -\theta$	-1
< 0	$> \theta$	1

TABLE I
TABLE SHOWING THE DIRECTIONS OF CORRECTION FOR ADAPTING THE VALUE OF m.

IV. Implementation Details of the Adaptation Logic

We have developed the following independently pluggable blocks:

Inter-Stream Coder (ISC) For a given configuration input of (n, m), the block takes m original incoming packets into the module and outputs n addition packets that are the coded packets using fountain coding operation on the m original packets. Thus, for each m input packets, this module provides $m + n$ output packets. The collection of such $m + n$ packets is called a *segment*. This block also adds the required header information including the values of m and n, and the sequence number of individual packets within such a segment. Thus, for an incoming rate of R packets per second, the ISC module provides an output stream of $R\frac{n+m}{m}$ packets per second.

Inter-Stream Decoder (ISD) For a given stream of input packets, each packet is parsed to get the information about the segment that this packet belongs to, and when m packets of the same segment[1] are received, it performs decoding to retrieve the m original packets.

Splitter The splitter takes a packet stream and uses the algorithm provided in Section III-A to split these packets into the N available interfaces using the probabilistic splitting. This module does not look into the nature of the packet that it is deciding to send on any given interface. This module is expected to be expanded in the future to take into account a cross-layer approach where it can be linked with the ISC block, with an awareness of coded/redundant packet.

Redundancy Recommendation Engine (RRE) This module is on the sender side and provides the recommendation of m and n to the ISC modules. This module works based on the feedback it receives from its

[1]Recall that a segment consists of m original and n coded packets. This module does not distinguish between original and encoded packets.

peer. This module implements the algorithm provided in Section III-B.

The ISC and ISD functionality can be considered as a new layer between the application layer and network layer. Figure 2 gives the functioning of the ISC when $m = 2$ and $n = 1$. To help ISD in decoding the packets at the receiver side, the ISC module adds the following attributes in every generated packet:

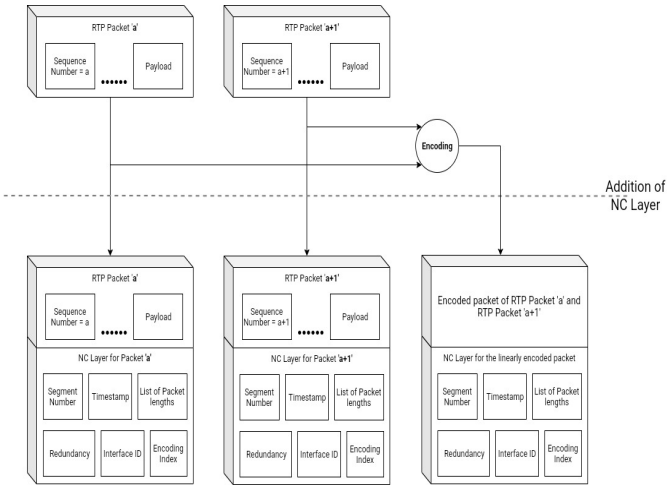

Fig. 2. NC Layer when $m = 2$ and $n = 1$.

1) Packet number: packets of a segment are numbered in the range 1 to $n + m$.
2) Length of packets
3) Parent packet's sequence number

The coding technique we are using in our ISC implementation is linear coding. Linear coding is based on linear combination of messages to encode and decode. For encoding and decoding we are using an encoder matrix. It generates encoded packet in the following manner:

En.1 Take m number of original packets and take dot product of it with a row of encoder matrix (the size of row of encoder matrix is equal to m).

En.2 Add the row index of encoder matrix in the ISC layer under 'encoding_index'.

En.3 Repeat steps 1 and 2 above n number of time to generate n encoded packets.

The ISD module on the receiver maintains a dictionary to store the segments. It takes any m packets from the segment to decode the original packets and put them in a encoded packet list. The decoding is done in the following manner:

De.1 First it creates the encoder matrix used for encoding the packets at the sender by checking the 'encoding_index' field of the ISC layer.

De.2 Take the inverse of the $m \times m$ encoder matrix to produce a decoder matrix.

De.3 Multiply the decoder matrix with the encoded packet list to produce decoded packet list.

De.4 Send the decoded packet to the gstreamer for playout. We call this step *Flush*.

De.5 Delete the segment from the dictionary.

In our implementation, we found that, sometimes, for a segment ISD is not receiving enough packets (received packets $< m$) to run the decoding algorithm. In this case for decoding, we follow the following steps:

MF.1 Wait for $W(= 40)$ segments.

MF.2 If ISD does not receive enough packets for the segment in the waiting time, send the original packets to the gstreamer for palyout and discard the encoded packets. We refer this step as "Mini Flush"

MF.3 Delete the segment from dictionary.

The receiver runs a feedback algorithm on a different thread to send feedback for the performance of the current redundancy and performance of the various interfaces. Thus, we have two types of feedback:

Type-1 feedback for probability optimisation.

Type-2 feedback for redundancy optimisation.

The traffic splitting problem (**P.3**) is viewed in the framework of Multi-arm bandit problem [9]. Considering an interface as an arm and for the next packet generated from ffmpeg, our traffic splitter has to decide which interface (or arm) to choose to transmit the packet. We initialized the probabilities of choosing interfaces to a uniform probability distribution. Based on the performance score of the interfaces from feedback we are optimising the probabilities.

In the receiver side we are calculating the performance score in the following manner:

MA.1 Calculate the delay-jitter for each interface.

MA.2 For every 100 packets from an interface calculate the average jitter.

MA.3 Calculate the performance score for each interface using a reward function, which is defined as inverse of average jitter.

MA.4 Send feedback of performance score to the sender side.

On the sender side probabilities are optimised in the following manner:

1) Receive the feedback
2) Check if feedback is of Type-1.
3) For Type-1 feedback: extract the performance score of the interfaces from the message.
4) Apply softmax function on perfromance score to get the recommended probablities.
5) Update the probabilities.

V. Implementation on Openairinterface

The various modules developed as part of this activity are portable to different systems in a plug-n-play manner. We used the module pipeline in an openairinterface-based large network emulation system where multiple virtual machines were used to run individual UEs (softmodem) on one machine, and similarly, multiple virtual machines on another machine were used to run individual eNBs. The channel model between the different UE-eNB pairs were varied. One of the limitations

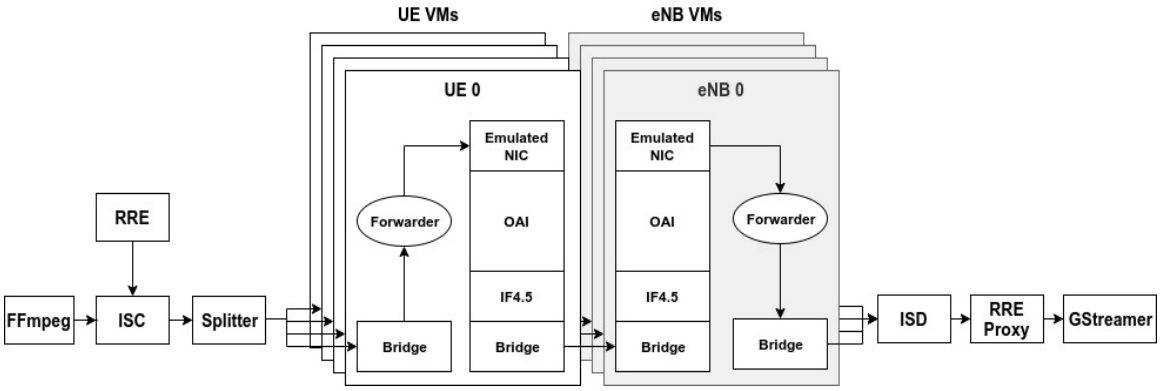

Fig. 3. The overall pipeline when using the openairinterface. The figure shows the use of ffmpeg as traffic source and gstreamer as the playout utility.

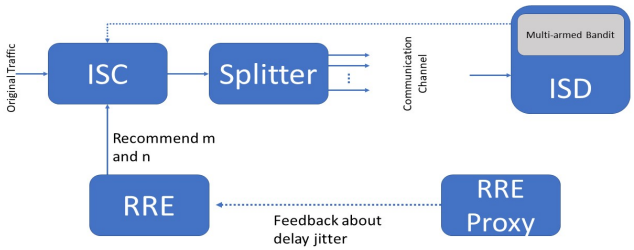

Fig. 4. The various blocks introduced for the adaptation logic.

of this system is that we have not been able to simulate multiple UEs connected to the same eNB. This is a known issue in openairinterface, and work on resolution of this is ongoing.

For our implementation, we choose to use the "siml1" option in the simulator which enables simulation of Layer 1 (PHY) [7] and connects UE-eNB pair through IF4.5 interface. Through siml1, even though all the virtual machines and host machines are connected through ethernet and virtual network bridges, they exchange only OFDM symbols [8]. The presence of abstraction till the resource elements in IF4.5 allows the control of SNR making it suitable for our testbench.

For the testbench, we made a total of five pairs of UE-eNB virtual machines (VM). The host computers of all eNBs and all UEs are connected through a physical ethernet cable and all the VMs are bridged to this network. The host containing the UEs act as a packet splitter and packets sent to it gets redirected to a particular VM based on the destination port. All UE VMs forward the packets to the emulated NIC interfaced made by the Openairinterface (OAI) script. All eNB VMs forward the incoming packets from OAI back to the eNB host which acts as an aggregator. This architecture is illustrated in Figure 3.

It is to be noted that the same modules developed for working with real world interfaces (cellular dongles) are used to create the end-to-end pipeline in the openairinterface setup as well. The developed modules thus seamlessly plug in to the ffmpeg and gstreamer pipelines.

VI. Results

We have implemented the proposed algorithm for transfer of real-time videos from a workstation that is connected to a video camera on one side and has multiple cellular dongle-based internet connectivity. The same setup also works over openairinterface. The RTP packets are captured using the Python's raw networking support and additional coded packets are generated. These packets (original and coded) are individually encapsulated inside a properly-formatted UDP packet and then sent across different cellular interfaces toward the end-device intended for reception of video stream. The values of n and m are dynamically configurable in the spirit of SD-WAN, using the adaptation logic provided in the paper.

When running the end-to-end video over openairinterface setup with multiple interfaces, we set the video stream configuration of ffmpeg to $30fps$, with rate of $500kbps$ and the total number of interfaces was set to $N = 5$. The adaptation algorithm uses a tolerance level of $30ms$ to avoid the ping-pong effect as mentioned earlier.

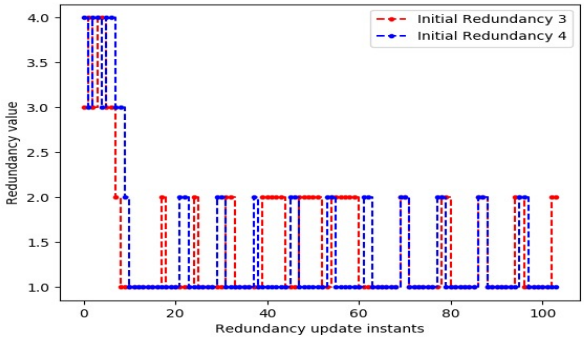

Fig. 5. Evolution of m in the less variable setup of openairinterface

In fig. 5, we see that target redundancy values vary as the update commands arrive. In the case of initial redundancy

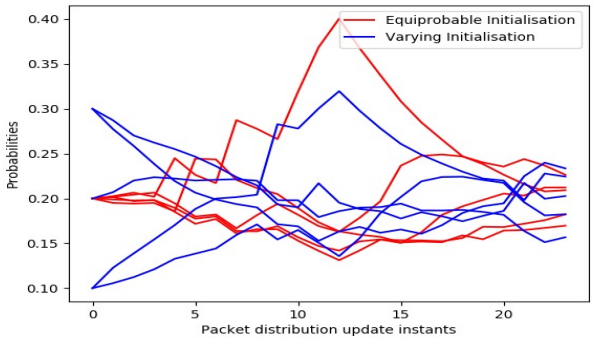

Fig. 6. Evolution of p in the less variable setup of openairinterface

3, the model explores the channel conditions by varying the values between 2 and 4 then later continues with 4 indicating convergence. With initial redundancy 2, we can see that the model tries to adapt to the best value but keeps oscillating between 1 and 3.

In fig. 6, we can see how the packet distribution probability vector changes with commands. The initial probability vectors for runs are (0.2,0.2,0.2,0.2,0.2) and (0.1,0.1,0.2,0.3,0.3) but the channel conditions were not changed. Hence, in the second run, the probability vector converges close to that of the first one even though the initial vectors are far.

Figure 7 provides the values of jitter obtained at various update instants in the real-world setup where $N = 7$ interfaces (from different operators in India) were used. The traffic generation rate was again varied using ffmpeg parameters for the live camera feed. We observe that for large rate, $2Mbps$, the adaptation algorithm indicates requirement for large number of redundant interfaces, while for $1Mbps$ traffic, small number of redundant interfaces were required. For very small data generation rate, i.e., $0.5Mbps$, we see that there was not enough penalty in the system to bring down the number of redundant interfaces.

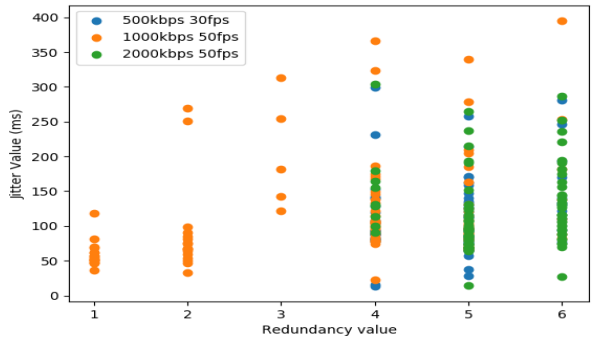

Fig. 7. Jitter vs n as obtained from real world implementation run for $N = 7$.

VII. Conclusion

We have presented, in the SD-WAN context, a configurable solution that uses availability of multiple access. The objective of the SD-WAN controller was set to limiting the end-to-end delay jitter for a real-time traffic. A two-time scale adaptation scheme was provided to decide on the number of redundant interfaces required and also finding the ideal traffic split into the available interfaces. A real-world implementation of the proposed solution is provided that works with real traffic sources and can be plugged in as pipeline to the ffmpeg sender or to LTE network simulator like openairinterface.

References

[1] https://www.networkworld.com/article/3284367/10-hot-sd-wan-startups-to-watch.html
[2] P. S. Schmidt, R. Merz, and A. Feldmann, "A first look at multiaccess connectivity for mobile networking," in Proceedings of the 2012 ACM Workshop on Capacity Sharing, ser. CSWS '12. New York, NY, USA: ACM, 2012, pp. 9–14.
[3] S. P. Rachuri, A. A. Ansari, D. Tandur, A. A. Kherani and S. Chouksey, "Network-Coded SD-WAN in Multi-Access Systems for Delay Control," in Proceedings of the 2019 4th International Conference on Contemporary Computing and Informatics (IC3I), Singapore, 2019. (IC3I), Noida, 2016, pp. 178-181.
[4] Byers, J. W. and Luby, M. and Mitzenmacher, M. and Rege, A., A Digital Fountain Approach to Reliable Distribution of Bulk Data, SIGCOMM Comput. Commun. Rev., Oct. 1998.
[5] M. Kim, J. Cloud, A. ParandehGheibi, L. Urbina, K. Fouli, D J. Leith M. Médard, Network Coded TCP (CTCP), ArXiv, 2012.
[6] Sesia, S., Toufik, I., Baker, M., LTE, The UMTS Long Term Evolution: From Theory to Practice, 2009, Wiley Publishing
[7] F. Kaltenberger., OpenAirInterface 5G Overview, Installation, Usage, 2018, available at https://www.openairinterface.org/, urldate = 2019-11-03
[8] EURECOM, France, Prototyping of Next Generation Fronthaul Interfaces (NGFI) using OpenAirInterface, available at https://www.openairinterface.org/?page_id=1695, urldate = 2019-11-03
[9] T. Lattimore and C. Szepesvari, Bandit Algorithms, available at https://tor-lattimore.com/ Cambridge University Press