**NCoS: A Framework for Realizing Network Coding Over Software-Defined Network**

Sicheng Liu* and Bei Hua†
University of Science and Technology of China(USTC),Hefei,Anhui,230027,China
Suzhou Institute for Advanced Study, USTC, Suzhou, Jiangsu, 215123, China
Email:*lsch@mail.ustc.edu.cn,†bhua@ustc.edu.cn

**Abstract**—Network coding is a transmission mechanism for improving the capacity of multicast applications. Practical problems arise when applying network coding in traditional wired networks: multipath multicast routing, backward compatibility with deployed base, and adding new functions in commercial routers. SDN network naturally solves these problems via logically centralized control plane and open APIs of Openflow switches. This paper proposes NCoS, a framework for realizing network coding over SDN networks. It gives a brief introduction on how to extend Openflow protocol to include new actions, and add network coding related functions in controller and switches.

**I. INTRODUCTION**

Network coding is a transmission mechanism firstly proposed in [1] for improving the capacity of multicast applications, and [2] proved that linear network coding can achieve the maximum multicast capacity. Network coding requires intermediate nodes to encode the received packets together before sending them out.

In a typical linear network coding scenario, a group of nodes is involved in moving data from a set of source nodes to a set of sink nodes. Each packet is considered to be a vector of N symbols on finite field of order q. Local encoding matrix is a matrix $V$ that directs the translation of input packets set $L$ to output packets set $U$. That is to say, $U = L \ast V$.

If the original packets generated by a source node are denoted as $o_1, o_2, \ldots$, then each packet arriving at a node is a linear combination of a series of original packets, denoted as $p = (o_1, o_2, \ldots, o_M) \ast A$, where $M$ is the number of packets encoded together and vector $A = (a_1, a_2, \ldots, a_M)^T$ (called global encoding vector) depends on the path the packet has taken. To recover the original packets, sink node collects sufficient number of arriving packets and solves the linear system of equations. To reduce the time of solving the linear equations, packets are divided into generations and only packets in the same generation can be encoded together.

So far, network coding has been applied to distributed storage system, wireless networks and wired networks. This paper focuses on applying network coding to wired networks, where two steps are needed [3]: 1) find $l$ edge-disjoint paths from the sender to each receiver; 2) design local encoding matrix for each node. [4] designed a packet format and a buffer model to make network coding practical in real networks. Nevertheless, problems still remain as summarized by [5]: the need for multipath routing to make network coding effective, the need for backward compatibility with a massively deployed base, and the need to keep computation and per-flow state in the core is hardly satisfied. All these problems stem from the connectionless nature of TCP/IP networks and difficulties of adding functions in commercial routers. Due to these difficulties, nowadays the only practical deployments of network coding in wired networks are on overlay networks, where only end systems are required to support network coding.

In this paper, we propose a framework (called NCoS) for realizing network coding over SDN networks. Benefited from centralized control logic and global network view, SDN architecture naturally solves the above problems. The NCoS framework includes functions in switch and controller and extensions to OpenFlow protocol. We also implement a prototype that can be used as a testbed to evaluate the theories and algorithms of network coding on wired networks. Different from former works, NCoS realizes network coding on network layer so that use of network coding is transparent to users. As far as we know, this is the first framework that implements network coding over SDN networks.

**II. DESIGN AND IMPLEMENTATION OF THE NCoS FRAMEWORK**

Our goal is to propose a framework that network coding is performed on network layer and is transparent to users. I.e., sender generates and sends out original IP packets, receivers receive these packets from network, and both of them are unaware of the encoding/decoding affairs.

To this end, for each multicast transmission, controller should determine whether an effective multipath multicast tree can be established. If not, packets are transmitted without network coding. Otherwise, controller first determines the functions that will be executed on each switch along the paths. There are four functions involved: 1) initialization function which adds NC (abbreviation of Network Coding) header to each encoded packet, 2) encoding function which buffers and encodes packets, 3) decoding function which recovers original packets from encoded ones by solving the system of linear equations and sends them to receivers, and 4) output function...
which just forwards packets. Then controller computes the local encoding matrix for each initialization and encoding function. Finally controller generates NC flow entries for each switch and dispatches them through controller-switch interface. Therefore, controller needs to compute multipath multicast tree, make encoding scheme, and generate NC flow entries. For Openflow switches, new actions should be defined to perform the initialization, encoding and decoding functions. Also Openflow protocol should be extended to include these new actions.

Fig.1 shows the functional modules of our NCoS framework, where modules marked by boldface rectangular frames are specific to this framework. It is worth noting that global knowledge of network topology, group membership, link statistics, etc. is needed for controller to build an effective multipath multicast tree. These services are commonly available in many open source SDN controllers, and can be reused in our framework. Three actions (init_coding, encoding, decoding) are defined and added to the switch.

A. Multipath Multicast Routing

Multipath multicast routing is to get a subgraph that contains l disjoint paths of minimum costs between each pair of (sender, receiver). In former works, this is done by two steps: 1) prune the network topology by omitting edges and nodes that do not meet the need of network coding multicast; 2) run a multipath routing algorithm (such as max-flow algorithm several times) to get l shortest disjoint paths for each pair of (sender, receiver). However, situation is a little bit different here: initialization and decoding function that are executed by hosts in former works must be executed by switches in our framework to provide transparency. That is, both sender and receivers in the multicast tree are switches.

Take the network topology in Fig.2a as an example, where all nodes are switches. Suppose that s connects to a sending host, and d1 – d5 each connects to a receiving host. Running max-flow algorithm for each (sender, receiver) pair may get a subgraph shown in heavy lines in Fig.2a, where s performs initialization function and d1 – d5 perform decoding function. As s, d1, d2, d3 and d5 each connects to the network through a single link, bandwidth on these link is wasted on NC headers. We then push initialization function one hop towards the network (i.e., push it into n1), and push decoding function one hop towards the network (e.g., push it from d2 into n4).

After this transformation, subgraph in Fig.2a may shrink to the subgraph shown in Fig.2b, which not only saves bandwidth consumption, but also reduces the number of encoding and decoding functions. This step is inserted between the above two steps.

B. Encoding Scheme Making

This module is to compute local encoding matrix for each node, and a main concern here is to guarantee that all original packets can be recovered at every receiver. In general, there are two methods to get a local encoding matrix: deterministic coding and random coding. Former works solely use random coding method due to lack of global information; however, in SDN networks both methods can be used. Since random coding may lose information, we use deterministic coding method.

For the local encoding matrix of initialization function, we randomly select a matrix and get a matrix $M_r$ for each receiver $r$. $M_r$ is composed of the global encoding vectors of all the packets that are encoded by initialization function and will be received by $r$. We will check whether $M_r$’s row rank is full. If not, we simply select another matrix and check again. For the local encoding matrix of encoding function, we implement a deterministic algorithm proposed in [3]. This algorithm finds a local encoding matrix that satisfies the following condition: the global encoding vectors of all the packets that are obtained by any downstream receiver of the encoding function are linearly independent.

C. NC Flow Entry Generation

Given the multicast tree and encoding scheme as input, this module generates NC flow entries for each node. Two steps are taken for each node on the tree: 1) map the assigned function to an action or an action list, 2) generate NC flow entries that would be carried in the Openflow messages.

In the scenario of network coding, packets belonging to the same NC flow (identified by source IP address and destination multicast IP address) but coming from different input links may be treated differently in a switch. That is, a function may translate into several actions. Take switch n1 in Fig.2a as an example, packets from link [s, n1] and link [n3, n1] need to be encoded and forwarded to node d1, and packets from link [n2, n1] may need to be forwarded to node d4. Then three actions
which is put between the IP packet header and the transport layer header, and uses 200 as its protocol identifier. *Header length* denotes the NC header length in bytes. *Generation id* is the generation the packet belonging to, and *generation size* gives the total number of packets in one generation. *Code type* specifies the finite field and coding method used. For now, the most significant bit (MS-bit) indicates the use of random coding (MS-bit=1) or deterministic coding (MS-bit=0), and the rest 7 bits (denoted as Q) indicate the finite field $GF(2^Q)$. *Code vector* carries the global encoding vector.

### 6. Buffer Management

Buffers are needed to perform network coding in switch. Two types of buffers are used: packet buffer and status buffer. Packet buffer is used to cache received packets that are to be encoded or decoded. Each packet buffer can hold $M$ packets, where $M$ is a configurable parameter. Status buffer is used to record generation id’s and counts the cached packets of each generation. Each status buffer is an array with $M$ elements, where each element contains a generation id and a packet counter.

Like other resources (e.g., TCAM, queues) of OpenFlow switch, buffers can be managed either by switch itself (switch-based method) or by controller (controller-based method). Switch-based method alleviates the buffer management overhead from controller, but controller may overwhelm switches with an inapposite encoding scheme. Therefore, we have controller to do the buffer management.

Both packet buffer and status buffer are managed in the same way. A huge space is pre-allocated in each switch to each type of buffer, whose usage is managed by controller. Whenever a buffer is needed, controller finds a usable one in the right space, marks it as used, and writes the buffer id in action field. Then the switch can use the buffer. Whenever a flow-entry (or flow-entries) that consume(s) a buffer is removed by a switch, the switch generates a flow-removed message. Whenever a flow-removed message is received, controller recycles the buffer by marking it as usable. To limit the communication overhead, only one flow-removed message is issued by the switch that removes a flow-entry with *init_coding* action.

### F. Implementation of Actions

This section is about how to perform network coding in the switch.

For each *init_coding* action, a packet buffer is allocated to cache the original packets, and a status buffer is allocated to record the current generation id and count the cached packets in the buffer. The generation id is initialized to 1 and increased in sequence. Once the packet counter exceeds the *generation size* given in the action, switch encodes these packets according to the *code vector* given in the action, and adds a NC header to the encoded packet, then increases current generation id by 1. The NC header is initialized as follows: *generation size* and *code vector* are copied from the action, *generation id* is copied from status buffer, *code type* is set to 2.
according to the finite field and coding method used, and header length counts the length of NC header in bytes.

For each encoding (or decoding) action, a packet buffer is allocated, and packet of generation \( g \) is stored in the \( i \)th slot of the buffer, where \( i = g \mod M \). It is worth noting that \( k \) packet buffers are allocated if packets from \( k \) input links will be encoded (or decoded) in a switch, each packet buffer corresponding to an input link. Apparently, packets of the same generation coming from different input links are stored in the corresponding slot position of the packet buffers, and records the generation id and number of allocated, each element corresponds to a slot position of the packet buffers, and records the generation id and number of packets currently stored in that slot in all packet buffers.

At the beginning of encoding, the status buffer is initialized to 0’s. Once a packet of generation \( g \) arrives, its packet slot is calculated as \( i = g \mod M \), and the generation id recorded in the \( i \)th element of the status buffer (say \( g_i \)) is checked. If \( g < g_0 \), which means this packet arrived too late, switch sends this packet immediately without encoding it. If \( g = g_0 \) or \( g_0 = 0 \), switch puts this packet in the \( i \)th slot, and sets the generation id to \( g \). If \( g > g_0 > 0 \), which means packets of this generation are not to be encoded as some of them may have lost or \( M \) is not big enough to absorb the delay between different paths, switch sends the packets in \( i \)th slot without encoding, and stores the received packet in slot \( i \). When any of the packet counters in the status buffer reaches input num given in the encoding action, switch generates a coded packet by combining all the packets of this generation according to the code vector given in the action, and sending it to the output port given in the action.

For decoding action, the status buffer is initialized to 0’s at the beginning. When a packet of generation \( g \) arrives, its packet slot is calculated as \( i = g \mod M \), and the generation id recorded in the \( i \)th element of the status buffer (say \( g_i \)) is checked. If \( g < g_0 \), which means this packet arrived too late, switch drops the packet. If \( g > g_0 > 0 \), which means some packets of generation \( g_0 \) have been lost, switch frees the packet slot, raises a warning, and inserts the received packet in slot \( i \). If \( g = g_0 \) or \( g_0 = 0 \), switch inserts the packet in slot \( i \). Before inserting a packet into a packet slot, Gaussian elimination is performed to test whether the packet carries useful information (i.e., code vectors of packets in this generation are linearly independent). If not, this packet is dropped. When any of the packet counters in the status buffer reaches input num in the action, switch recovers original packets from these encoded ones by solving the system of linear equations and sends them out to the output list given in the action.

III. IMPLEMENTATION AND EXPERIMENTS

We implement NCoS by extending OpenvSwitch 1.9.0 to include init coding, encoding and decoding actions, and extending POX 0.0.2 to include multipath multicast routing, encoding scheme making, NC flow generation and buffer management modules. We test NCoS on Mininet, one of the popular network simulation platforms.

One disadvantage of network coding is its high computational complexity, especially when random coding method is used. XOR is a simple and fast operation that can greatly reduce the computing overhead of \( p_i + b \times p_j \), if \( b = 1 \). Benefited from deterministic coding method and centralized control logic of SDN, we have the opportunity to get a XOR-preferential encoding scheme. Rather than selecting a random local encoding matrix using the algorithm proposed in [3], we first check if an all 1’s matrix meets the condition of a feasible local encoding matrix. If not, we then choose it as [3] describes.

We test our XOR-preferential encoding scheme on NCoS with 20, 40, 60, 80 nodes, and use TopGen [6] to generate trial topologies. At each trial, we randomly choose a certain proportion of nodes to form the multicast group, with one as the sender and others as the receivers. Iperf 2.0.5 is modified to send and receive packets. Our test shows that the probability of using all 1’s matrix as local encoding matrix rises from less than 1% to greater than 35%, and the computing overhead decreases more than 17.5% (XOR saves 50% CPU consumption than original method).

NCoS can be used to evaluate other network coding researches. For example, in former works multipath multicast routing and encoding scheme making are performed separately like what we have done in this paper. However, NCoS provides the possibility to jointly compute them.

IV. CONCLUSION

This paper gives a brief introduction on how to build a user-transparent network coding framework on SDN. NCoS is a first try of such work. However, NCoS is still a very preliminary work that does not solve problems such as packet loss, group membership change, etc. So far, NCoS has been implemented on Mininet solely; in the future we will implement it on China Environment for Network Innovations (CENI).

REFERENCES