CodedINT: Leveraging Network Coding to Improve the Visibility of In-band Network Telemetry (INT)

Zhihuang Ma, Shaofei Tang, Wenpeng Tao, Yuhan Xue, and Zuqing Zhu†
School of Information Science and Technology, University of Science and Technology of China, Hefei, China
†Email: {zqzhu}@ieee.org

***Abstract*—With momentum gained from programmable data plane (PDP), in-band network telemetry (INT) has been widely considered as a promising technique for realtime network monitoring. In this work, we leverage network coding (NC) to design CodedINT, for improving the visibility of INT in lossy networks. Specifically, we propose to encode the telemetry data in multiple packets with NC and distribute the encoded data over a group of packets. Then, among the group of packets, if we can receive enough ones that satisfy the decoding condition of NC, the whole original telemetry data can be recovered. We first explain the design of CodedINT to elaborate on its operation principle, packet format, and system implementation. Then, we implement and experimentally evaluate CodedINT in a real network testbed. Our experiments demonstrate that in a lossy network with packet loss rate at 80%, CodedINT ensures that 89.48% of the telemetry data carried by INT packets can be recovered successfully.**

***Index Terms*—In-band Network Telemetry (INT), Protocol-oblivious forwarding (POF), OpenvSwitch (OVS), Linear coding.**

I. Introduction

Recently, the rapid growth of network services and traffic generated by them have reshaped network systems dramatically [1], and thus introduced numerous new network elements and protocols in the Internet. For instance, mobile and access networks are being adapted to support 5G [2], metro and core networks are being developed based on elastic optical networking (EON) [3–5] for more flexibility, and programmability has been enhanced everywhere in network systems [6, 7] to unleash them from being restricted by the existing packet formats and forwarding protocols. However, these technical advances make network infrastructures increasingly complicated, which calls for more effective methods for network monitoring.

With the development of software-defined network (SDN) and programmable data plane (PDP) [6, 7], in-band network telemetry (INT) [8] has been proposed to provide a promising solution for realtime and fine-grained network monitoring. Specifically, INT is a PDP-enabled technique, with which the realtime status of each switch on a packet flow's forwarding path is recorded, encoded and inserted as specific INT header fields in the packets of the flow. Hence, the end-to-end performance of each flow can be monitored in a per-packet/per-hop manner. Compared with the traditional network monitoring approaches (*e.g.*, the simple network management protocol (SNMP) [9]), INT is more flexible and enables unprecedented visibility in the realtime and fine-grained way [10].

However, INT also has drawbacks: 1) it brings additional bandwidth overheads and increases the processing burden of switches, and 2) when there are packet losses, the telemetry data carried by the packets will be lost too. Previously, for the first issue, people have proposed and demonstrated several selective INT schemes, which sample packets in each flow to insert INT header fields [11–15], for relieving the overheads of INT. Nevertheless, the second issue (*i.e.*, how to improve the visibility of INT to overcome the impact of packet losses) has not been widely explored yet.

It is relevant and even necessary to protect telemetry data from being lost together with packets, and the rationale behind this is multi-fold. First of all, the telemetry data carried by a lost packet will be critical for the operator to reproduce the status of its forwarding path right before the packet loss. Therefore, obtaining the telemetry data will make troubleshooting much easier. Secondly, in a lossy network environment, it will be difficult for the operator to leverage INT to visualize the operation of a network service in the complete and realtime manner, especially when the traffic of the network service flows through many switches and/or does not use a line topology (*e.g.*, multicast, manycast and network virtualization [16–19]). Finally, the telemetry data collected by INT is frequently used to forecast future network status (*e.g.*, traffic prediction), while losing data samples due to packet losses will degrade the accuracy of time series prediction. Intuitively, telemetry data can be made more survivable by duplicating it and inserting the copies in multiple consecutive packets. However, this will increase the overheads of INT significantly and cannot protect against bursty packet losses.

In this work, we leverage the idea of network coding (NC) [20] to design an enhanced INT system, namely, CodedINT, which can improve the visibility of INT in lossy networks in a programmable and effective way. Specifically, we propose to encode the INT fields in multiple packets with NC and distribute the encoded fields over a group of packets. Then, among the group of packets, if we can receive enough packets that satisfy the decoding condition of NC, the whole original telemetry data can be recovered. Meanwhile, by adjusting the NC scheme of CodedINT, we can tune the tradeoff among data survivability, INT overheads, and decoding latency adaptively.

To verify our design of CodedINT, we implement it in an SDN system that enables PDP with protocol-oblivious forwarding (POF) [7], and demonstrate it experimentally with a real network testbed. The results validate the functionalities of CodedINT, and confirm that even when CodedINT is turned on, both the processing of packets on our switches and the

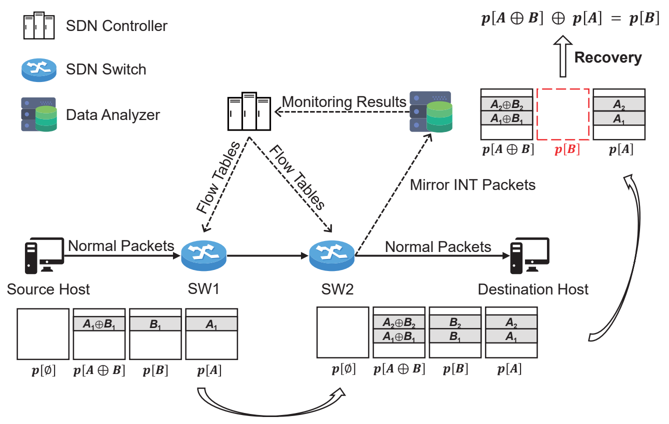

Fig. 1. Operation principle of CodedINT.

processing of telemetry data on our data analyzer (DA) can be accomplished in high throughputs (*i.e.*, above 2.00 million packets per second (Mpps) and 2.66 Mpps, respectively). We also conduct experiments to study the tradeoff among data survivability, INT overheads, and decoding latency.

The rest of this paper is organized as follows. We describe the design and implementation of CodedINT in Sections II and III, respectively. Experimental demonstrations are discussed in Section IV. Finally, Section V summarizes the paper.

II. Design of CodedINT

In this section, we present the design of CodedINT, including the operation principle, key network elements, NC schemes for telemetry data, and proposed packet format.

A. Operation Principle

We develop CodedINT by extending our previous work of Sel-INT [13]. Sel-INT samples the packets in a flow to insert INT fields, such that the tradeoff between network monitoring accuracy and INT overheads is balanced well. Specifically, in an SDN-based Sel-INT system, the controller first determines the INT sampling rate of a flow and installs corresponding flow tables in the switches on its forwarding path, and then the switches will select packets to insert INT fields accordingly. To select a packet for INT field insertion, the ingress switch encodes an INT header in the packet, and the subsequent switches will match to it to insert the required INT fields [13].

Therefore, we have two types of packets in a Sel-INT system, *i.e.*, those that carry INT fields (namely, **INT packets**) and the normal ones, while for each flow, the ratio of INT packets to the total ones is just the sampling rate. Meanwhile, the results in [13] suggested that for a flow whose rate is 1 Mpps or more, a sampling rate of a few percent can already guarantee the monitoring accuracy of fast-changing telemetry data (*e.g.*, bandwidth usage). However, if an INT packet gets lost in the Sel-INT system, the INT fields carried by it will be lost too. CodedINT addresses this issue by leveraging NC.

The example in Fig. 1 explains the operation principle of our proposed CodedINT. Here, the forwarding path of the flow goes through two switches (*i.e.*, *SWs* 1 and 2). If we assume that INT needs to collect two types of telemetry data (*i.e.*, types A and B) on each switch and the INT sampling rate is set as 50%, we can use two packets to collect the two types of telemetry data, respectively[1], and keep the two packets following them as normal ones. We denote the telemetry data of type A, which is collected on *SW* 1, as A_1, and so on.

Hence, without CodedINT, we encode four consecutive packets periodically as $p[A]$, $p[B]$, $p[\emptyset]$, and $p[\emptyset]$, where $p[A]$ denotes an INT packet carrying the telemetry data of type A and $p[\emptyset]$ refers to a normal packet. Then, if $p[A]$ or/and $p[B]$ get lost, we cannot recover the corresponding telemetry data. One way to relieve the impact of packet loss is to duplicate and encode the telemetry data repeatedly. For example, we can encode four consecutive packets periodically as $p[A]$, $p[B]$, $p[A]$, and $p[B]$, and then if any one of them gets lost, we can recover the whole telemetry data. Nevertheless, this will increase the overheads of INT, and in this particular example, the overheads saved by Sel-INT will be offset completely.

CodedINT addresses the aforementioned issues with NC. Specifically, as shown in Fig. 1, we can encode four consecutive packets periodically as $p[A]$, $p[B]$, $p[A \oplus B]$, and $p[\emptyset]$, where $A \oplus B$ means that we apply bitwise exclusive OR (XOR) operation on the INT fields for the telemetry data of types A and B. Therefore, if any one of the four packets gets lost, we can still recover the whole telemetry data, while comparing with the approach that simply duplicates INT packets, the CodedINT in this example saves the number of INT packets by 25%. Moreover, we hope to point out that CodedINT also has the flexibility to arrange the sequence of INT packets and normal packets, for protecting telemetry data against bursty packet losses. For instance, we can encode the packets in Fig. 1 as $p[A]$, $p[B]$, $p[\emptyset]$, and $p[A \oplus B]$, to reduce the probability of $p[A \oplus B]$ being lost together with $p[B]$.

B. NC Scheme in CodedINT

We refer to a set of consecutive packets on which CodedINT can encode telemetry data as an **NC group**, *e.g.*, the four packets in Fig. 1 are in the same NC group. The NC scheme of CodedINT can be represented with the following notations.

- M: the number of telemetry data types that CodedINT needs to collect on each switch (*e.g.*, $M = 2$ in Fig. 1).
- N: the number of packets in an NC group of CodedINT (*e.g.*, $N = 4$ in Fig. 1).
- X_j: the j-th type of telemetry data that CodedINT needs to collect on each switch ($j \in [1, M]$).
- Y_i: the coded INT field that needs to be inserted by each switch on the i-th packet in an NC group ($i \in [1, N]$).
- $a_{i,j}$: the boolean parameter that equals 1 if X_j should be considered in the coding scheme for Y_i, and 0 otherwise.
- $b_{i,j}$: the boolean parameter that equals 1 if Y_i should be used to decode X_j in the data analyzer of CodedINT, and 0 otherwise.

The NC scheme of each packet in an NC group is denoted as

$$Y_i = \sum_{j=1}^{M} a_{i,j} \cdot X_j, \quad \forall i \in [1, N], \tag{1}$$

[1]Here, for each packet, we only insert one INT field in it per hop, to avoid generating excessively long INT packets.

where, in this work, we use $\sum$ to denote the summation of bitwise XOR operation, *e.g.*, $Y_1 = \sum_{j=1}^{2} a_{1,j} \cdot X_j$ means $Y_1 = (a_{1,1} \cdot X_1) \oplus (a_{1,2} \cdot X_2)$. Meanwhile, the decoding scheme is

$$X_j = \sum_{i=1}^{N} b_{i,j} \cdot Y_i, \quad , \forall j \in [1, M]. \tag{2}$$

Note that, when the values of M and N are determined, we totally have $2^{N \cdot M}$ possible coding schemes, but not all of them are feasible or can be used by CodedINT to protect telemetry data. Therefore, we should carefully select the values of $\{a_{i,j}\}$ to design the NC scheme. Meanwhile, for a fixed set of $\{a_{i,j}\}$ (*i.e.*, an NC scheme of CodedINT), the values of $\{b_{i,j}\}$ that can lead to successful decoding are not unique. For instance, in Fig. 1, after we have encoded the packets as $p[A]$, $p[B]$, $p[A \oplus B]$, and $p[\emptyset]$, both the decoding schemes of $\{b_{1,1} = 1\}$ and $\{b_{2,1} = 1, b_{3,1} = 1\}$ can be leveraged to decode the telemetry data of type A. This actually explains why CodedINT introduces redundancy in INT.

To analyze CodedINT's tolerance to packet losses, we consider the four NC schemes in Fig. 2. Here, we need to collect three types of telemetry data (*i.e.*, types A, B and C) on each switch ($M = 3$). It can be seen that *Scheme* 0 actually does not apply NC on INT fields, which is the case in Sel-INT, and we use it as a benchmark of our analysis. The performance of the NC schemes in Fig. 2 is compared in Table I, which verifies that CodedINT can not only effectively improve the visibility of INT in lossy networks but also avoid introducing unnecessary INT overheads. For instance, with *Scheme* 3, we can recover the whole telemetry data as long as the packet losses in an NC group are 3 or less. However, if we want to achieve the same goal by encoding the telemetry data repeatedly, we need to repeat the 3 INT packets for telemetry data types of A, B and C for at least 4 times (*i.e.*, having 12 INT packets in each NC group). Finally, we would like to point out that when there is only one type of telemetry data to be considered ($M = 1$), the NC scheme will just duplicate and encode the telemetry data repeatedly in multiple INT packets, since bitwise XOR operation is not feasible in this case.

C. System Design

In addition to the operation principle, Fig. 1 also illustrates the key network elements in a CodedINT system. Here, the SDN switches should support PDP, and thus they can be developed based on P4 [6] or POF [7] and use either software-based or hardware-based platform. Note that, compared with the traditional INT, CodedINT only requires two additional functionalities on switches: 1) buffering a very limited number of INT fields, and 2) applying bitwise XOR operation on INT fields. These two functionalities can be easily supported on both software-based and hardware-based PDP switch platforms. In this work, we develop CodedINT-enabled switches based on our previous work of OVS-POF [21], which is a software-based switch that extends OpenvSwitch (OVS) to support POF.

The SDN controller needs to compute and install CodedINT-related flow tables in the switches, and we develop it based on ONOS [22]. Specifically, the flow tables will tell the related switches about: 1) the types of telemetry data to collect, 2) the INT sampling rate to use, and 3) the NC scheme to apply. As shown in Fig. 1, the egress switch of each flow will duplicates INT packets and send the copies to DA, where the telemetry data in INT fields gets parsed, extracted, decoded, and analyzed. After obtaining the network monitoring results from the INT fields, the DA will forward them to the controller. Meanwhile, the egress switch removes the INT fields on INT packets and converts them back to normal ones before forwarding them to the destination host. In this work, the DA is our homemade software system. Different from Sel-INT and other existing INT schemes, CodedINT makes the decoding of INT fields in DAs a stateful operation, which means that the DA needs to identify the NC groups accurately and apply correct decoding schemes when different combinations of INT packets in an NC group were received.

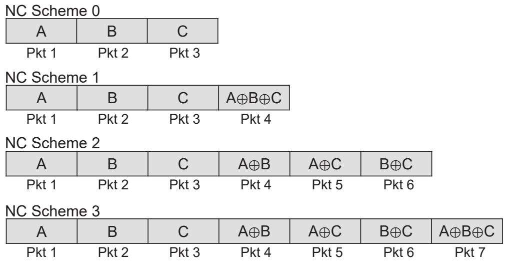

Fig. 2. NC Schemes for CodedINT ($M = 3$).

TABLE I
COMPARISON OF NC SCHEMES IN FIG. 2

NC Scheme	# of INT Packets in each NC Group	Tolerance to Packet Loss
0	3	0
1	4	25% (1/4)
2	6	33% (2/6)
3	7	43% (3/7)

In order to facilitate CodedINT, we design the packet format as shown in Fig. 3. Specifically, for each INT packet, CodedINT inserts an INT header in between the IP header and payload of the packet. In the INT header, the first five fields (*i.e.*, *Type*, *Length*, *MapInfo*, *GroupID*, and *CodeID*) are fixed ones, which means that they are created when the INT header is first inserted in a packet by its ingress switch and will not be removed until the egress switch. The five fixed fields are followed by a stack of *Metadata* fields, each of which is an INT field containing telemetry data. The definitions of the aforementioned fields are explained as follows.

Type is a 2-byte field that is filled with $0x0908$ to let the switches distinguish INT packets from normal ones. *Length* occupies one byte and records how many *Metadata* fields have already been encoded in the packet. *MapInfo* is a 2-byte bitmap that indicates the selected types of telemetry data to be collected on each switch and the selected NC scheme. In this work, we support 10 types of telemetry data (as shown in Fig. 3), each of which is represented by one among the ten lowest bits of *MapInfo*. The remaining 6 bits in *MapInfo* can be used to denote the selected NC scheme. *GroupID* stores the ID of the NC group that the packet belongs to, while *CodeID* tells the sequence ID of the packet in its NC group (*i.e.*, for NC). For instance, for the packets $p[A]$, $p[B]$, $p[A \oplus B]$, and

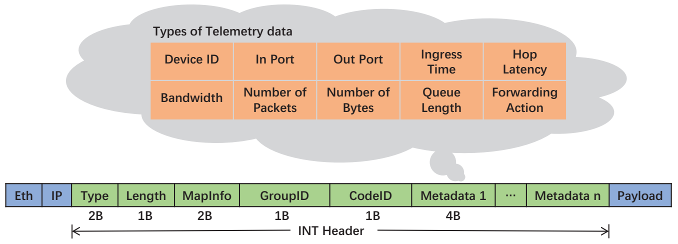

Fig. 3. Packet format used in CodedINT.

$p[\emptyset]$ in Fig. 1, their *GroupIDs* are the same, and the *CodeID* of $p[A]$ is 0, and so on. *Metadata* is a 4-byte field that carries the coded result of certain telemetry data. For example, for the INT packets in Fig. 1, the INT fields for A_1, $A_2 \oplus B_2$ and others are each encoded in a *Metadata*.

III. System Implementation

A. Packet Processing for CodedINT on Switches

Algorithm 1 explains the packet processing for CodedINT on switches. Specifically, for each packet, the procedures of CodedINT on its ingress, intermediate and egress switches are in *Lines* 3-8, 17-19, and 11-15, respectively. *Lines* 6, 12 and 18 show that a new *Metadata* field should be buffered locally if necessary, to enable the NC of *Metadata* fields. For instance, in Fig. 1, the *Metadata* of A_1 should be buffered after it has been encoded in $p[A]$, to enable the generation of the *Metadata* of $A_1 \oplus B_1$ for $p[A \oplus B]$. Note that, each *Metadata* only needs to be buffered until all the packets of the current NC group have been processed. In this work, the procedures in *Algorithm* 1 are translated into POF-based packet processing pipelines, and we implement them in our software-based switches accordingly.

B. Implementation of DA

As CodedINT makes the decoding of *Metadata* fields in DAs a stateful operation, we make each DA accumulate INT packets and decode the *Metadata* fields carried by them on-the-fly. However, this can make the DA use longer time to extract and record the telemetry data in INT packets, especially in a lossy network environment. Hence, we define the duration between when the DA receives the first INT packet of an NC group and when it has successfully decoded all the telemetry data carried by the NC group as the **decoding latency**.

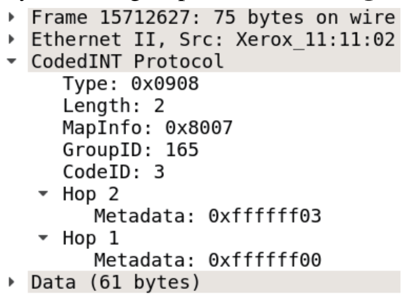
```
▸ Frame 15712627: 75 bytes on wire
▸ Ethernet II, Src: Xerox_11:11:02
▾ CodedINT Protocol
    Type: 0x0908
    Length: 2
    MapInfo: 0x8007
    GroupID: 165
    CodeID: 3
  ▾ Hop 2
      Metadata: 0xffffff03
  ▾ Hop 1
      Metadata: 0xffffff00
▸ Data (61 bytes)
```

Fig. 4. Wireshark capture of an INT packet generated by CodedINT.

IV. Experimental Demonstrations

To evaluate the performance of our proposed CodedINT, we conduct experiments in a network testbed whose configuration is the same as that in Fig. 1. The experimental setup consists

Algorithm 1: Procedure of CodedINT on Switches

```
1  while a packet is received do
2      if this is its ingress switch then
3          determine whether encoded as an INT packet
           based on the INT sampling rate from controller;
4          if the packet should be an INT packet then
5              obtain NC scheme, GroupID, and CodeID;
6              encode Metadata based on the NC scheme
               and CodeID, and buffer it if necessary;
7              generate and insert an INT header in packet;
8          end
9      else
10         if this is its egress switch then
11             if the packet is an INT packet then
12                 apply CodedINT according to its INT
                   header, and buffer Metadata if necessary;
13                 mirror the packet to DA;
14                 remove the INT header from the packet;
15             end
16         else
17             if the packet is an INT packet then
18                 apply CodedINT according to its INT
                   header, and buffer Metadata if necessary;
19             end
20         end
21     end
22     forward the packet to its next hop;
23 end
```

of two stand-alone SDN switches, an SDN controller, a DA, and two end-hosts. Each switch is developed by extending our previous work of OVS-POF [21], and runs on a Linux server equipped with 10 GbE ports. The controller is implemented based on ONOS, while the DA is homemade, and they also run on stand-alone Linux servers. The two end-hosts are emulated with commercial traffic generators/analyzers [23]. In the experiments, we let the egress switch (*SW* 2) randomly drop certain packets to emulate lossy network environments.

A. Feature Validation

We first try to verify the functionality of CodedINT. Here, we assume that CodedINT needs to collect the first three types of telemetry data (*i.e.*, *Device ID*, *In Port*, and *Out Port*) on each switch (*i.e.*, the NC has $M = 3$). Then, *Scheme* 2 in Fig. 2 is used to encode the INT packets, *i.e.*, each NC group has 6 INT packets as $p[A]$, $p[B]$, $p[C]$, $p[A \oplus B]$, $p[A \oplus C]$, and $p[B \oplus C]$, where A, B and C denote the telemetry data of *Device ID*, *In Port* and *Out Port*, respectively. We denote *Scheme* 2 with a bitmap of 100000, and the bitmap for the selected types of telemetry data is 0000000111. Hence, we should encode $0x8007$ in the *MapInfo* of INT packets. The *Device IDs* of *SWs* 1 and 2 are $0x$ffffff01 and $0x$ffffff02, respectively, and the used input and output ports on them are both 1 and 2.

Fig. 4 shows the Wireshark capture[2] of an INT packet re-

[2]The *Ethertype* is set as $0x0908$ to trigger Lua plugin for parsing customized CodedINT protocol, and therefore the IPv4 header is not parsed here.

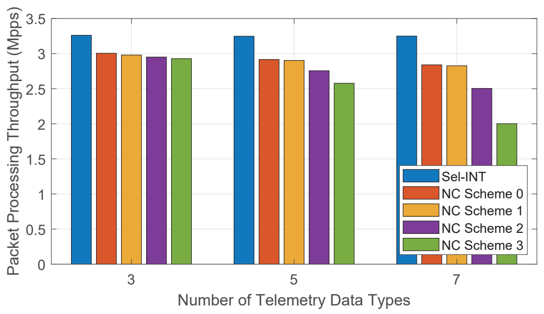

Fig. 5. Packet processing throughput of SDN switch.

ceived by the DA. Here, we have *Type* as $0x0908$ to indicate an INT packet. *Length=2* means that two *Metadata* fields are encoded (one for each hop), and *CodeID=3* suggests that the INT packet is the fourth one in an NC group (*i.e.*, $p[A \oplus B]$). Hence, each switch applies bitwise XOR operation on *Device ID* and *In Port* to get a *Metadata*. Specifically, the *Metadata* inserted by *SWs* 1 and 2 should be $0x$ffffff01$\oplus 0x$00000001$=0x$ffffff00 and $0x$ffffff02$\oplus 0x$00000001$=0x$ffffff03, respectively, which are exactly the results shown in Fig. 4.

B. Performance Benchmarking

As CodedINT requires more packet processing than Sel-INT, it might impact the throughput of SDN switches, especially the software-based ones. Hence, we conduct experiments to compare the packet processing throughput of CodedINT with that of Sel-INT. Specifically, we let the source host send 64-byte packets through *SW* 1, and record the maximal packet rate at which the switch can achieve 0 packet loss as the throughput. The experiments consider $M \in \{3, 5, 7\}$ types of telemetry data, and for each M, there are 4 NC schemes as listed in Table II. Specifically, for the NC group of each M, *Scheme* 0 does not apply NC, *Scheme* 1 adds an INT packet that applies XOR to all the *Metadata*, *Scheme* 2 adds INT packets that apply XOR to pairs of *Metadata*, and *Scheme* 3 adds INT packets that apply XOR to 2 and 3 *Metadata*.

We run each experiment for a minute, and plot the average throughputs of our switch in Fig. 5. We can see that compared with Sel-INT, the extra processing in CodedINT does decrease the switch's throughput, and the throughput degradation increases with the complexity of NC scheme. Note that, although NC is not applied in the *Scheme* 0 of each M, the throughput is still degraded in Fig. 5. This is because in this case, the switch still needs to process the *GroupID* and *CodeID* in each INT packet, which is not required in Sel-INT. Fortunately, except for the cases with $M = 7$, the throughput degradation is not significant, and even in the worse experimental scenario (*Scheme* 3 of $M = 7$), the switch's throughput is still above 2 Mpps. Meanwhile, we also measure the average throughputs of the DA for Sel-INT and CodedINT, which are 5.02 and 2.66 Mpps, respectively. Here, CodedINT is measured for its worst-case scenario ($M = 7$). Hence, even though the throughput of the DA is still larger than that of the switch, its throughput degradation is actually larger than that of the switch. This is caused by the fact that for CodedINT, the stateful decoding in DA is more complex than the packet encoding in switches.

TABLE II
NC SCHEMES USED IN EXPERIMENTS

	# of INT Packets in each NC Group		
Telemetry Data Types	3	5	7
NC Scheme 0	3	5	7
NC Scheme 1	4	6	8
NC Scheme 2	6	15	28
NC Scheme 3	7	25	63

C. Tradeoff Analysis

1) Recovery Ratio of Telemetry Data: To check CodedINT's recovery ratio of telemetry data in lossy network environments, we set the packet loss rate (PLR) within $[0, 100\%]$ and measure the ratio of the *Metadata* that can be recovered at the DA. Each recovery ratio is obtained after processing $1,000,000$ *Metadata*. Fig. 6 shows that the protection capability of *Scheme* 1 decreases with M, while by encoding many more INT packets in each NC group, *Schemes* 2 and 3 achieve a higher recovery ratio for a larger M. Specifically, in Fig. 6(c), in the lossy network with PLR at 80%, CodedINT can recover 89.48% of the telemetry data carried by INT packets.

2) Decoding Latency: Note that, for CodedINT, the better protection capability of a more complex NC scheme does not come without a price. There are actually two drawbacks of using a more complex NC scheme. Firstly, the bandwidth overheads of CodedINT will be larger, and this can be seen by checking the number of INT packets in each NC group in Table II. Secondly and more importantly, a more complex NC scheme will make the decoding latency longer, when there are packet losses. Therefore, we conduct more experiments to measure the decoding latency when the packet rate from the source host is 1 Mpps and the PLR is chosen within $[0, 90\%]$. In each experiment, we only record the decoding latency of the NC groups whose *Metadata* can be recovered at the DA, and the average decoding latency is obtained after the *Metadata* of $100,000$ NC groups having been decoded successfully.

Fig. 7 shows the results on average decoding latency. Note that, as the PLR increases, the ratio of the NC groups whose *Metadata* can be fully recovered will become extremely small for certain NC schemes. Hence, some data points of decoding latency are absent in Fig. 7. We find that the decoding latency of a more complex NC scheme generally increases faster with PLR, which implies that in a lossy network, the redundancy brought by NC can be heavily utilized for decoding.

Finally, by combining Figs. 6 and 7, we can see the tradeoff between recovery ratio and decoding latency clearly. Specifically, for each M, the protection capability of CodedINT to packet losses gets improved with a more complex NC scheme, while this can also prolong the average decoding latency (*i.e.*, degrading the timeliness of INT-based network monitoring), and *vice versa*. Therefore, when applying CodedINT in a real-world network, we should select the NC scheme according to the actual network status, for balancing this tradeoff well.

V. CONCLUSION

In this work, we leveraged NC to design CodedINT, for improving the visibility of INT in lossy networks in a programmable and effective manner. We first laid out the design

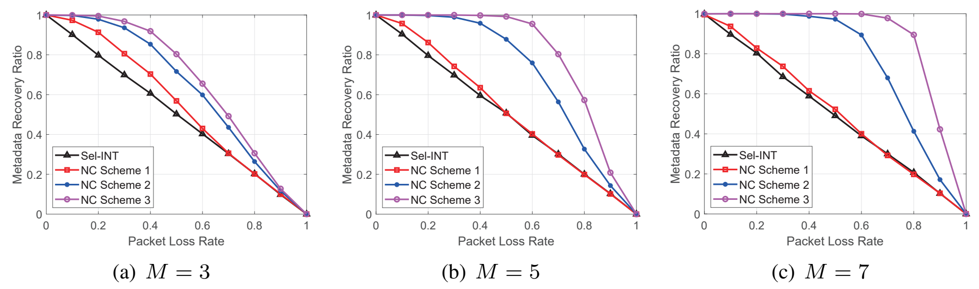

Fig. 6. Recovery ratio of telemetry data.

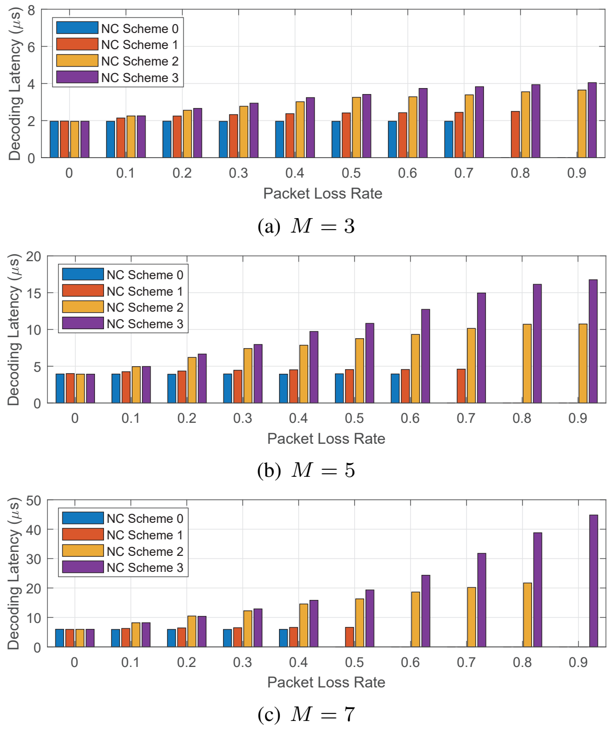

Fig. 7. Decoding latency of telemetry data.

of CodedINT to explain its operation principle, packet format, and system implementation. Then, to evaluate its performance, we implemented CodedINT in an SDN system based on POF, and demonstrated it experimentally with a real network testbed. The results validated the functionalities of CodedINT, and confirmed that even when CodedINT is turned on, both our software-based switches and our DA could maintain high throughputs (*i.e.*, above 2.00 Mpps and 2.66 Mpps, respectively). Meanwhile, the experiments also demonstrated that in a lossy network with PLR at 80%, CodedINT can successfully recover 89.48% of the telemetry data carried by INT packets.

Acknowledgments

This work was supported in part by the National Key R&D Program of China (2020YFB1806400), NSFC project 61871357, SPR Program of CAS (XDC02070300), and Fundamental Funds for Central Universities (WK3500000006).

References

[1] P. Lu *et al.*, "Highly-efficient data migration and backup for Big Data applications in elastic optical inter-datacenter networks," *IEEE Netw.*, vol. 29, pp. 36–42, Sept./Oct. 2015.

[2] M. Shafi *et al.*, "5G: A tutorial overview of standards, trials, challenges, deployment, and practice," *IEEE J. Sel. Areas Commun.*, vol. 35, pp. 1201–1221, Jun. 2017.

[3] Z. Zhu, W. Lu, L. Zhang, and N. Ansari, "Dynamic service provisioning in elastic optical networks with hybrid single-/multi-path routing," *J. Lightw. Technol.*, vol. 31, pp. 15–22, Jan. 2013.

[4] L. Gong *et al.*, "Efficient resource allocation for all-optical multicasting over spectrum-sliced elastic optical networks," *J. Opt. Commun. Netw.*, vol. 5, pp. 836–847, Aug. 2013.

[5] Y. Yin *et al.*, "Spectral and spatial 2D fragmentation-aware routing and spectrum assignment algorithms in elastic optical networks," *J. Opt. Commun. Netw.*, vol. 5, pp. A100–A106, Oct. 2013.

[6] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.

[7] S. Li *et al.*, "Protocol oblivious forwarding (POF): Software-defined networking with enhanced programmability," *IEEE Netw.*, vol. 31, pp. 12–20, Mar./Apr. 2017.

[8] INT dataplane specification. [Online]. Available: https://github.com/p4lang/p4-applications/blob/master/docs/INT_v2_1.pdf.

[9] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A simple network management protocol (SNMP)," *RFC 1098*, May 1990. [Online]. Available: https://tools.ietf.org/html/rfc1157.

[10] L. Tan *et al.*, "In-band network telemetry: a survey," *Comput. Netw.*, vol. 186, p. 107763, Feb. 2021.

[11] Y. Kim, D. Suh, and S. Pack, "Selective in-band network telemetry for overhead reduction," in *Proc. of CloudNet 2018*, pp. 1–3, Oct. 2018.

[12] B. Niu *et al.*, "Visualize your IP-over-optical network in realtime: A P4-based flexible multilayer in-band network telemetry (ML-INT) system," *IEEE Access*, vol. 7, pp. 82 413–82 423, Jun. 2019.

[13] S. Tang *et al.*, "Sel-INT: A runtime-programmable selective in-band network telemetry system," *IEEE Trans. Netw. Serv. Manag.*, vol. 17, pp. 708–721, Jun. 2020.

[14] B. Basat *et al.*, "PINT: Probabilistic in-band network telemetry," in *Proc. of ACM SIGCOMM 2020*, pp. 662–680, Aug. 2020.

[15] L. Tan *et al.*, "A packet loss monitoring system for in-band network telemetry: Detection, localization, diagnosis and recovery," *IEEE Trans. Netw. Serv. Manag.*, vol. 18, pp. 4151–4168, Nov. 2021.

[16] N. Xue *et al.*, "Demonstration of OpenFlow-controlled network orchestration for adaptive SVC video manycast," *IEEE Trans. Multimedia*, vol. 17, pp. 1617–1629, Sept. 2015.

[17] L. Gong and Z. Zhu, "Virtual optical network embedding (VONE) over elastic optical networks," *J. Lightw. Technol.*, vol. 32, pp. 450–460, Feb. 2014.

[18] L. Gong, H. Jiang, Y. Wang, and Z. Zhu, "Novel location-constrained virtual network embedding (LC-VNE) algorithms towards integrated node and link mapping," *IEEE/ACM Trans. Netw.*, vol. 24, pp. 3648–3661, Dec. 2016.

[19] J. Liu *et al.*, "On dynamic service function chain deployment and readjustment," *IEEE Trans. Netw. Serv. Manag.*, vol. 14, pp. 543–553, Sept. 2017.

[20] R. Ahlswede, N. Cai, S. Li, and R. Yeung, "Network information flow," *IEEE Trans. Inf. Theory*, vol. 46, pp. 1204–1216, Jul. 2000.

[21] OVS-POF with Sel-INT module developed by USTC. [Online]. Available: https://github.com/USTC-INFINITELAB/OpenvSwitch-pof/tree/fast-path.

[22] ONOS. [Online]. Available: https://onosproject.org/.

[23] BigTao220. [Online]. Available: http://www.xinertel.net/nam-portable/nams-(portable).html.