OpenSDC: A Novel, Generic Datapath for Software Defined Coalitions

Qiao Xiang[‡], Franck Le[◇], Yeon-sup Lim[◇], Vinod Mishra[○], Y. Richard Yang[‡], Hongwei Zhang[♭],
[‡]Yale University, [◇]IBM T.J. Watson Research Center,
[○]U.S. Army Research Labs, [♭]Iowa State University,
{qiao.xiang, yry}@cs.yale.edu, fle@us.ibm.com,
y.lim@ibm.com, vinod.k.mishra.civ@mail.mil, hongwei@iastate.edu

***Abstract*—With more and more success of Software Defined Networking (SDN) in academia and industry, people have started to explore how to integrate SDN into military coalitions, to realize a software-defined coalition (SDC) infrastructure. However, the integration of SDN into SDC is non-trivial due to the insufficiencies of the SDN datapath for expressing the data plane behavior in SDC systems. In particular, SDC systems operate in highly dynamic tactical networks (*e.g.*, wireless networks) and requires the data plane to support a wider range of events other than the traditional incoming packet event in wired SDN. In addition, SDC systems widely adopt in-network processing (INP) such as network coding, and supporting these functionalities in the data plane requires flexible storage for packets and complex operations on both packet headers and payload, which are not supported in the existing SDN datapath. In this paper, we tackle these issues by designing OpenSDC, a novel, generic SDC datapath which extends the current match-action primitive, with three new primitives: event, packet buffer, and packet INP block. The implementation of the proposed primitives can be optimized using a range of techniques to accelerate the event processing efficiency of OpenSDC. We implement a prototype of the proposed datapath, and demonstrate its ability to support network-coding-based 1+1 data delivery protection in dynamic tactical networks. Evaluation results show that compared with a state-of-the-art proactive protection system implemented using only match-action tables, our prototype significantly improves the efficiency and resiliency of tactical networks.**

I. Introduction

Software Defined Networking (SDN) have been deployed in many academic and industrial systems [12], [13], [22]. The success of such deployment is leading to efforts to explore the feasibility and benefits of integrating SDN into military coalitions to realize an efficient, agile, and optimal software-defined coalition (SDC) infrastructure [18], in which autonomous coalition members operate under harsh tactical network environments with resource constraints, such as limited power and processing capability, and dynamic connectivity.

One may think such an integration from SDN to SDC should be straightforward, in that SDN provides coalition members with greater control over networks' datapath, including the freedom to define different packet header fields and match-action rules that examine these fields and perform simple computations on them [5], [6], [24]. Though these features are all promising, they are insufficient for supporting SDC for several reasons.

First, in SDN, the main role of a device's data plane (*e.g.*, physical and virtual SDN switches) is to efficiently process packets. Therefore, its behavior is expressed as a *packet processing pipeline*. In such a datapath, which we refer to *the SDN datapath* in the remaining of the paper, the data plane only responses to *packet-in* events and process incoming packets through a user-defined pipeline of match-action tables. Other events (*e.g.*, link down and medium access contention) are handled either by the control plane, or by the underlying protocols (*e.g.*, data-link layer or physical layer protocols). In contrast, SDC requires devices to maintain communication capability even under disrupted, intermittent and lossy network environments (*e.g.*, wireless networks). Therefore, in SDC, a device's data plane needs to handle a wider range of events (*e.g.*, link up/down and timer expiration). The SDN datapath does not allow users to define these events or specify how these events are handled. For example, data transmission systems in SDC widely use TDMA-based mechanisms to avoid high interferences of transmissions. These mechanisms require the data plane to start a packet transmission only on pre-fixed time slots, which requires defining timers and corresponding handlers on the data plane but is not supported in the SDN datapath.

Second, to ensure the packet forwarding efficiency, the SDN datapath only allows simple operations on packet headers. In contrast, the SDC datapath requires complex operations for in-network processing (INP) techniques (*e.g.*, data aggregation, packet packing and network coding), which are widely used in SDC systems to perform data analytics in coalition networks [20], [30], or as means to improve the data delivery performance of SDC under the highly dynamic tactical network environments [29]. Realizing these INP techniques on the data plane require temporary, flexible storage for packets and complex operations on both packet headers and payload (*e.g.*, multiplication and division). However, none of these are supported in the SDN datapath.

To address these problems, we present OpenSDC, a novel and generic SDC datapath. In contrast to the packet processing pipeline model of the current SDN datapath, OpenSDC models the behavior of its data plane as an *event processing pipeline*, with three new primitives: event, packet buffer, and packet INP block, in addition to the existing match-action table primitive. To accelerate the event processing efficiency in OpenSDC, we adopt a series of algorithmic techniques, such as bit-wise operations and look-up tables.

Using the OpenSDC datapath, we design and implement a network-coding (NC)-based, proactive protection system which provides 1+1 data delivery protection to SDC networks. Evaluation results show that our implementation outperforms a state-of-the-art proactive protection system implemented using match-action tables in tactical networks, in terms of efficiency, reliability and resiliency.

The rest of the paper is organized as follows. In Section II

we give an overview of the SDN datapath and show its insufficiency by surveying representative SDC systems. We present the design of OpenSDC and our approach for accelerating its efficiency in Section III. In Section IV, we present the design and implementation of our NC-based protection system using the OpenSDC datapath. We evaluate the performance of our prototype in Section V. We discuss related work in Section VI before concluding this paper in Section VII.

II. Background and Motivation

In this section, we give an overview of the current SDN datapath [5], and then describe its insufficiencies for satisfying the requirements of SDC.

A. Overview of the SDN Datapath

The SDN datapath is a packet processing pipeline composed of *match-action* tables [5], [6], [9]. Users can program SDN datapaths using high-level languages such as P4 [5]. Figure 1 presents an abstract model for the SDN datapath. When a packet arrives at the data plane, the device first parses and extracts packet headers using a user-defined parser. The extracted header fields pass through a user-defined ingress pipeline of match-action tables. Each table matches on a subset of the extracted headers and applies simple computation primitives to these fields and packet metadata. By the end of this pipeline, the packet is pushed into a queue chosen by the ingress pipeline. Then a user-defined egress pipeline of match-action tables takes it for further processing before output.

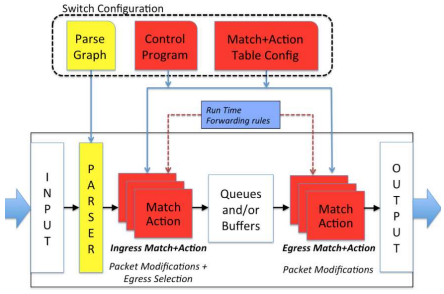


Fig. 1: The abstract model for the SDN datapath [5].

The SDN datapath provides the following primitives for packet processing:

Programmable Match-action table: This primitive is the core of the SDN datapath. Each table can match on any user-defined packet headers and take corresponding actions, such as forwarding, drop and modification of header fields and packet metadata.

Packet metadata: The SDN data plane devices provide a limited amount of memories that maintain states within a packet or across packets, such as counters, registers and meters.

Simple computation primitives on packet headers and metadata: On the SDN datapath, the device can perform simple computation on packet headers and metadata, such as addition and hashing.

Non-programmable packet storage: The SDN data plane devices provide a pre-defined data structure (*i.e.*, queue) to temporally store the packets between ingress and egress pipelines. These queues are used for scheduling algorithms that the device supports (*e.g.*, FIFO and round-robin).

B. Insufficiencies of the SDN Datapath for SDC

Though powerful, the existing SDN datapath is insufficient for expressing the desired behavior of devices' data plane in SDC systems. To this end, we survey three representative classes of SDC systems as follows.

TDMA-based data transmission systems: In coalition networks, many data transmission systems adopt TDMA-based mechanism to schedule the packet transmission between devices in a fair and efficient manner [23], which requires the SDC data plane to create and update different events (*e.g.*, timers) to decide when and where to forward a packet. However, the SDN datapath does not provide any primitive to allow user-defined events.

Data aggregation and packing systems: Data aggregation and packing [20], [30] are widely used for military data collection, such as intrusion detection. These systems process packets' payload at data plane devices instead of sending all packets back to a processing base station. In this way, they improve energy efficiency and data delivery performance of coalition networks by reducing network traffic load and thus channel contention. In such systems, the SDC data plane need to perform arithmetic operations, such as average, max and min, on the payload of packets. However, the SDN datapath does not allow such operations.

Network coding data delivery systems: Network coding data delivery systems [10] are important tools for improving the throughput, reliability, fairness, and management of coalition networks. The basic operations of these systems are encoding and decoding of a set of packets, which require temporary packet storage with random access and arithmetic operations (*i.e.*, add, subtract, multiplication and division) of packet payload on finite field. In addition, many network coding systems use complex scheduling mechanisms instead of simple FIFO or round-robin. To support such systems, the SDC data plane needs to provide a more complex packet storage model other than queue, perform arithmetic operations on packet payload, and create and update different events. None of these are allowed in the SDN datapath.

III. OpenSDC Overview

Having analyzed the insufficiency of the SDN datapath for expressing the behavior of SDC data plane, we present OpenSDC in this section, a novel, generic datapath for SDC.

A. OpenSDC High-Level Design

The key design decision of OpenSDC is to expand the data plane's capability from the processing of only packet-in events, to the processing of both hardware and user-defined events. In this way, the data plane behavior of an SDC system is expressed as an *event processing pipeline* in OpenSDC. Figure 2 shows an abstract model for OpenSDC. Specifically, OpenSDC keeps the match-action table primitives in the SDN datapath, but introduces three new primitives: event, packet buffer and packet INP block. Given an event processing pipeline, each *stage* consists of a match-action table, packet buffer or packet INP block, and events are passed from one stage to another.

Event: In OpenSDC, an event is a message passed between stages. The hardware defines a fixed set of events it can trigger by specifying their message format and semantics. And users also have the flexibility to define different events that can be triggered by stages in the pipeline. An event triggered by hardware will be sent to the head of the event processing pipeline, while an event triggered by a stage can be sent to any stage in the pipeline. For the completeness of presentation, we leave an illustration of the usage of event in Section IV-B.

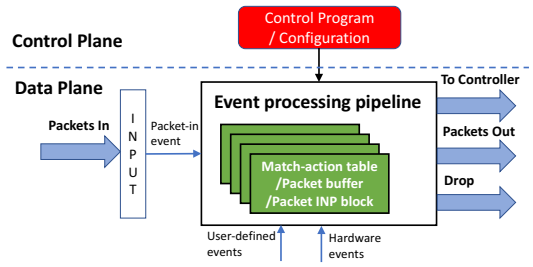


Fig. 2: The abstract model for the OpenSDC datapath.

Packet buffer: The packet buffer in OpenSDC is a key-value store providing temporary storage for packets. Given a buffer entry, its key is the hashing function of user-defined packet headers and its value is the entire packet, including both the headers and the payload. Different events are defined to provide usual interfaces of key-value store, such as insert, update, delete, and retrieve. It provides the random access of buffered packets based on user-defined packet headers, which enables a wide range of event processing functionalities of SDC systems (*e.g.*, scheduling).

Packet INP block: As suggested by the name, OpenSDC provides packet INP block to support in-network processing in SDC systems. Compared with the SDN datapath, OpenSDC allows arithmetic operations on not only user-defined packet headers, but packet payloads. Invoked by an event, a packet INP block retrieves packets from a packet buffer, performs arithmetic operations on these packets to create new packets, inserts them into a packet buffer or sends to the underlying protocols for transmission. As discussed in Section II, this primitive is crucial for efficient and resilient in-network analytics and data delivery in SDC.

Concurrency model: To ensure the correctness of event processing, a device whose data plane supports OpenSDC must obey certain concurrency model. Motivated by the packet processing concurrency model of the SDN datapath [5], [6], [25], we define the following concurrency model for OpenSDC.

***Definition** 1 (Concurrency model of OpenSDC):* Given an event processing pipeline in OpenSDC and an event e_i, any processing stage that modifies states visible to the next event e_{i+1} must finish execution before e_{i+1} reads the modified states.

B. OpenSDC Processing Optimization

The concurrency model defines the correctness of the OpenSDC datapath, but does not specify any constraints on its performance (*e.g.*, line-rate processing). One approach to fill this gap is to resort to the high-performance specialized hardware to support OpenSDC. This approach is not generally applicable to SDC, where devices are equipped with heterogeneous hardwares. Hence we leave it as future work. Alternatively, we adopt a series of algorithmic techniques, which are applicable to even low-end hardwares, to accelerate the event processing efficiency of OpenSDC. For simplicity, we present these techniques within the context of the packet INP block, but they can easily be extended to the match-action table as well.

Bit-wise arithmetic operation: A packet INP block that performs data aggregation on OpenSDC usually requires arithmetic operations on packet payload. The cost of such operations varies. For instance, multiplication and division has larger overhead than addition and subtraction. To accelerate the expensive operations, OpenSDC re-writes them as bit-wise expressions. For example, $\frac{a+b}{2}$ can be expressed as $(a+b) >> 1$, and $a * 4$ can be expressed as $a << 1$. And multiplication and division by a number not a power of 2 can be expressed as the sum of multiple bit-wise expressions. These operations are particularly useful for data aggregation systems performing in-network analytics on packet payload.

Look-up table: A packet INP block performing network coding related operations performs arithmetic operations on packet headers and payloads in Galois Field. In addition to bit-wise arithmetic operations, we exploit pre-computed look-up tables to further accelerate such blocks. Sine these operations are defined in a finite field, such look-up tables only needs a small space in memory. For example, for a finite field $GF(2^8)$, which is commonly used in current network coding based data delivery systems, the look-up table for multiplication only needs 64 KB of memory, and the look-up table for inversion only needs 256 bytes of memory.

We note that the techniques we adopt for OpenSDC is not platform-dependent, but rather common techniques for accelerating computations on heterogeneous hardware platforms (*e.g.*, general CPU, FPGA, Smart NIC and sensors), which suits the need for SDC. There are also other algorithmic techniques that can accelerate the event processing of the OpenSDC datapath, and we leave them as future work.

IV. OpenSDC Case Study: A Network-Coding-Based Proactive Protection System

To demonstrate the expressiveness and generality of the OpenSDC datapath, we design and develop an NC-based, proactive protection SDC system, which we refer to ProNCP in the remaining of the paper. For the completeness of presentation, we first give an overview of ProNCP, including how it computes routing configurations. Then we present the implementation of ProNCP on the data plane using the OpenSDC datapath.

A. Overview of ProNCP

ProNCP integrates the capability of network coding [3] for improving network throughput and the broadcast nature of wireless communication [7], to ensure efficient, reliable, real-time data delivery for military coalitions under the presence of disrupted, intermittent and lossy network environments (*e.g.*, wireless networks).

Figure 3 describes the architecture of ProNCP. Specifically, ProNCP provides 1+1 proactive protection to the SDC network by sending coded traffic from the source to the destination along two node-disjoint NC-based routing braids. On the control plane, a control program takes the topology information collected by data plane devices (*i.e.*, the link reliability of forwarding candidates of each device) as an input. Then it computes two node-disjoint NC-based routing braids for specific source-destination pairs, and sends the routing configuration (*i.e.*, the selected forwarders, the effective load, and the rate control parameters) to each corresponding device. On the data plane, each device stores received packets into a temporary buffer, generates coded packets by randomly mixing the received packets, broadcasts the coded packets to the neighbors, and stops the broadcast when receiving certain signals from the neighbors.

Routing configuration computation in ProNCP: The key challenge for implementing the control plane of ProNCP is

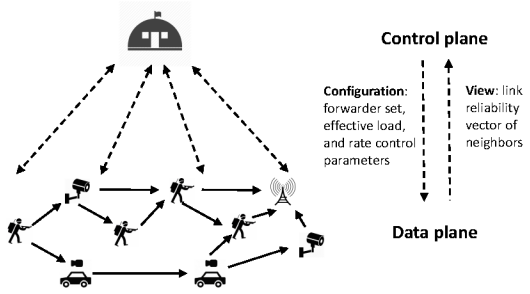


Fig. 3: The architecture and workflow of ProNCP.

to efficiently compute two node-disjoint NC-based routing braids that perform 1+1 delivery of coded packets for a given source-destination pair with a minimal cost. Though efficient algorithms have been developed for finding two node-disjoint routing paths with a minimal cost [26], they are not applicable to the context of node-disjoint NC-based routing braids. Using the analytic framework for estimating the cost of NC-based routing [29], we propose the following proposition.

***Proposition** 1:* Assume a directed acyclic graph $G = (V, E)$ with a source node S and a destination T, where each edge $(i, j) \in E$ is associated with a cost ETX_{ij} computed as the reciprocal of the link reliability of (i, j). It is NP-hard to find two node-disjoint NC-based routing braids B_1 and B_2 such that the total cost of delivering K linear independent packets from S to T along each braid is minimized.

We prove this proposition via a reduction from the classic two-partition problem [11]. During the proof, we also find and fix a mistake in the NP-hardness proof of the classic two commodity integral flow problem [8]. We omit the details due to the space limit and refer interested readers to [28].

Given this NP-hardness result, we develop an efficient heuristic algorithm in ProNCP. The algorithm first computes two node-disjoint paths with a minimal cost using the Suurballe algorithm [26]. Using the computed paths as references, it then alternatively assigns remaining nodes not chosen for forwarding to each path using the optimal single NC-based routing braid construction algorithm [29], and eventually constructs two node-disjoint NC-based routing braids. During this process, the forwarder set, the effective load and the rate control parameters of each node in the network are also computed.

B. Implementation of ProNCP on data plane using OpenSDC

Figure 4 presents the implementation of ProNCP on the data plane using OpenSDC. Specifically, ProNCP uses a key-value packet buffer for temporary packet storage. ProNCP registers a series of user-defined events, such as *send-data* and *send-ACK* events. ProNCP uses two match-action tables: packet and event classifiers, and four packet INP blocks: *linear independence test*, *collective receiving test*, *coded packet generator* and *coded ACK generator* blocks.

The ProNCP system uses an event classifier to respond to different events on the data plane. Specifically, a packet-in event is triggered when a packet arrives at the device. The packet is then passed from the event classifier to the packet classifier. In case of a data packet, the packet classifier sends a send-ACK event to the event classifier, and sends this packet to the linear independence test block. If the data packet is an innovative one, (*i.e.*, if it is a packet linear independent of the ones in the packet buffer), the packet is put to the buffer. Otherwise, it is dropped. In case of an ACK packet,

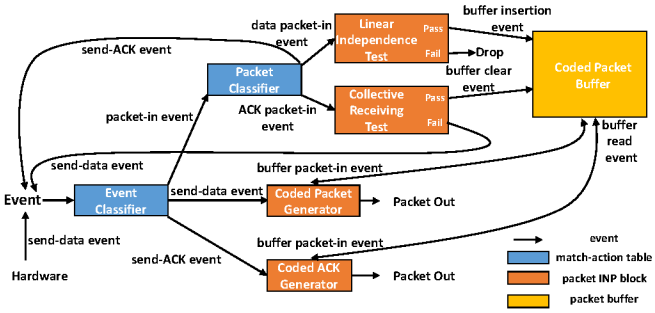


Fig. 4: Implementation of ProNCP on the data plane using OpenSDC.

the packet classifier sends it to the collective receiving test block to perform a collective-spacing test [29], [17] on this packet to test whether the device should transmit other coded data packets. If so, a send-data event is triggered. Otherwise, data packets related to the ACK will be cleared from the buffer.

A send-data event is forwarded to the coded packet generator block, which selects a flow that has a non-zero remaining effective load, reads the packets associated with this flow from the buffer, uses them to generate a coded packet, updates the effective load and then sends the coded packet to underlying devices (*e.g.*, network adapter or radio frequency front end) for actual transmission (*i.e.*, packet out). A send-ACK event is passed to the coded ACk generator block, which reads from the packet buffer the packets that has the same batch number as indicated in the ACK, generates a coded ACK packet, and sends it to the underlying device for actual transmission.

We implement the ProNCP data plane shown in Figure 4 on the TelosB sensor [2], a low-power, low-cost wireless device that is widely used for mission-critical data collection networks. Different from the data plane devices in SDN (*e.g.*, programmable switches) that are equipped with special, expensive hardware (*e.g.*, SRAM and TCAM) to help accelerate the match-action SDN datapath, the TelosB sensor is only equipped with a MSP430 16-bit CPU and a small amount of memory. Our implementation, which takes about 1,200 lines of nesC code, shows that the OpenSDC datapath supports the implementation of complex SDC systems on a wide range of devices that are used in SDC.

V. Performance Evaluation

In this section, we experimentally evaluate the performance of ProNCP. We first present the experimentation methodology and then the measurement results.

A. Methodology

Testbed: We use the *NetEye* wireless sensing and control testbed at Wayne State University [1]. In NetEye, 130 TelosB sensors are deployed in an indoor environment, where every two closest neighboring sensors are separated by 2 feet. Each sensor is equipped with a 3dB signal attenuator and a 2.45GHz monopole antenna. In our measurement study, we set the radio transmission power to be -15dBm such that multihop networks can be created. And we use the 802.15.4 MAC protocol in our experiment.

Topology: We focus on data collection scenarios as it is one of the most common scenarios in military coalitions. Out of the 130 sensors in NetEye, we randomly select 60 motes with uniform probability to form a random network. Among these 60 nodes, 10 are randomly selected as source nodes and one

as a data sink. Each source node periodically generates 40 data packets with an inter-packet interval uniformly distributed between 500 milliseconds and 3 seconds.

Systems studied: With the aim to explore the impact of in-network processing and the necessity and benefits of OpenSDC, we investigate the performance of the following proactive protection systems:

- ProNCP: the NC-based, proactive protection system we propose and implement in Section IV;
- TNDP: a state-of-the-art proactive 1+1 protection system that sends data traffic from a source along two shortest node-disjoint paths to a destination [26].

Both systems use the same architecture described in Figure 3, except that the data plane of TNDP is implemented as a pipeline of match-action tables as done in SDN. In the control plane, NC-based node-disjoint routing braids and node-disjoint routing paths are computed with a long-time sampling dataset of the link reliability of the experiment network. Then the forwarding configurations, such as the forwarder set and the effective load, are installed to the data plane of each system. In addition, we set the finite field $GF(2^8)$ and batch size 8 for ProNCP.

Performance metrics: For each system, we evaluate their behaviors based on the following metrics:

- *Delivery reliability*: the percentage of valid data packets correctly received by the data sink;
- *Delivery cost*: the average number of transmissions required for delivering a valid data packet from its source to the destination;
- *Goodput*: the number of valid data packets received by the sink per second;

where a data packet is *valid* if and only if the batch it belongs to can be successfully decoded by the data sink.

Network failure model: In our experiments, we deploy an additional periodic timer for all intermediate nodes in the network. Once the timer at node v_i expires, with a probability f, v_i enters a transient failure period, during which it cannot send or receive any packet. We comparatively study the performance of ProNCP and TNDP under different settings of f:

- $F0$: $f = 0$ for all intermediate nodes in the network; this is to represent the scenario where no failure happens in the network.
- $F10$ $f = 0.1$ for all intermediate nodes in the network; this represents the scenario where intermediate nodes have a 10% chance to stop working for a short period of time.
- $F20$ $f = 0.2$ for all intermediate nodes in the network; this represents the scenario where intermediate nodes have a 20% chance to stop working for a short period of time.

B. Measurement Results

We first present the measurement results for the no network failure scenario $F0$, then we discuss the cases of network failure patterns $F10$ and $F20$. For each setting of experiments, we run 10 experiments for ProNCP and TNDP and present means and 95% confidence intervals.

No network failure ($F0$): Figure 5 shows the delivery reliability, delivery cost and goodput of both systems. In Figure 5a, we observe that both ProNCP and TNDP achieve a delivery reliability close to 100%. However, the average transmission cost of ProNCP is only 50% of that of TNDP, as shown in Figure 5b. This shows that in-network processing of packets provides significant benefits for improving the efficiency of military networks, hence proving the necessity and benefits of OpenSDC its new primitives (*i.e.*, event, packet buffer, and packet INP blocks).

As shown in Figure 5c, the goodput of TNDP is slightly higher than that of ProNCP. However, this is an acceptable trade-off for providing proactive protection to networks, as ProNCP sends twice the traffic load than normal routing. And according to our experiment setting, the goodput of ProNCP and TNDP are both close to the capacity of the whole network.

Transient network failures ($F10$ and $F20$): Figure 6 shows the performance of ProNCP and TNDP under different failure models. As shown in Figure 6a, ProNCP keeps the delivery reliability close to 100% under both $F10$ and $F20$ failure models. In contrast, the delivery reliability of TNDP degrades to 91% under the $F10$ model and drops to 80% under the $F20$ model. Figure 6b shows that even under the existence of transient node failures, the average transmission cost of ProNCP is consistent at a low level, while the cost of $TNDP$ slightly increases in the $F10$ case, and drastically increases by 30% in the $F20$ case. Furthermore, the goodput of ProNCP maintains close to the capacity of the network under different failure models, while that of TNDP drops by 15% in the $F20$ case, as shown in Figure 6c.

All these result shows that in-network processing of packets significantly improves the resiliency and efficiency of coalition networks, and again proves the necessity and benefits of introducing new primitives for the OpenSDC datapath.

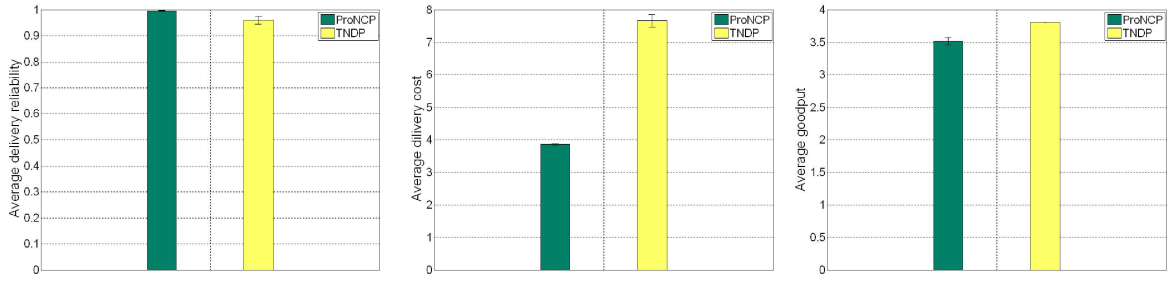
VI. Related Work

With SDN gaining the momentum in both academia and industry, people have started to investigate the programmability of the SDN datapath [14], [15], [16], [19], [21], [24], [25]. Some studies [14], [15], [16], [21], [24] realize specific resource allocation protocols and networked applications on programmable switches. Though these realizations show that the SDN datapath is expressive, we show by a thorough survey that it is insufficient for expressing the data plane behaviors of SDC systems. Domino [25] and Marple [19] investigate the design of high-performance hardware to enhance the programmability of the SDN datapath. However, in military coalition networks, data plane devices have heterogenous hardware support. Therefore, we adopt a series of algorithmic techniques to improve the efficiency of the OpenSDC. And we show by a proof-of-concept prototype of ProNCP system that even low-power wireless devices can support OpenSDC.

Software-defined radio (SDR) systems such as Sora [27] and Atomix [4] investigate the programming of radio systems and signal processing applications. OpenSDC is orthogonal with these systems, and the integration of SDR with OpenSDC is our future work.

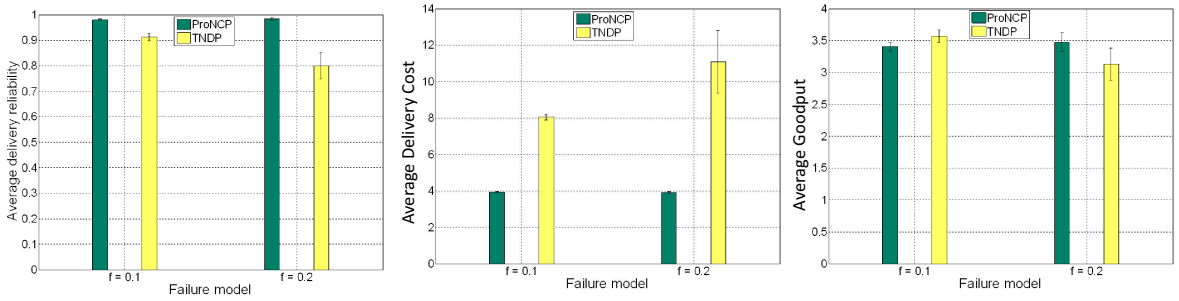
VII. Conclusion and Future Work

In this paper, we analyze the insufficiency of the SDN datapath for expressing the data plane behavior of SDC systems, and design OpenSDC, a novel, generic SDC datapath. In OpenSDC, the data plane behavior of an SDC system is expressed as an event processing pipeline. In addition to the match-action primitive, three new primitives: event, packet buffer and packet INP block, are introduced. We adopt a series of algorithmic techniques to accelerate the event processing efficiency of OpenSDC. We develop a prototype of



(a) Delivery reliability. **(b)** Delivery cost. **(c)** Goodput.

Fig. 5: Performance of ProNCP and TNDP when no network failure happens.



(a) Delivery reliability. **(b)** Delivery cost. **(c)** Goodput.

Fig. 6: Performance of ProNCP and TNDP in network with transient failures.

ProNCP, an NC-based proactive protection SDC system, using OpenSDC. Evaluation shows that it outperforms a state-of-the-art protection system implemented using the SDN datapath, in terms of resiliency, reliability and efficiency. As future work, we are investigating the high-performance hardware design for OpenSDC, and the integration of OpenSDC with SDR.

REFERENCES

[1] NetEye testbed. http://neteye.cs.wayne.edu/neteye/home.php.

[2] Telosb sensors. https://telosbsensors.wordpress.com/.

[3] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung. Network information flow. *Information Theory, IEEE Transactions on*, 2000.

[4] M. Bansal, A. Schulman, and S. Katti. Atomix: A framework for deploying signal processing applications on wireless infrastructure. In *NSDI*, pages 173–188, 2015.

[5] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[6] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. In *ACM SIGCOMM Computer Communication Review 2014*.

[7] S. Chachulski, M. Jennings, S. Katti, and D. Katabi. Trading structure for randomness in wireless opportunistic routing. In *SIGCOMM 2007*.

[8] S. Even, A. Itai, and A. Shamir. On the complexity of time table and multi-commodity flow problems. In *FOCS 1975*.

[9] O. N. Foundation. Openflow switch specification 1.4.0. Open Networking Foundation (on-line), Oct. 2013.

[10] C. Fragouli, D. Katabi, A. Markopoulou, M. Medard, and H. Rahul. Wireless network coding: Opportunities & challenges. In *MILCOM'07*.

[11] M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

[12] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven wan. In *SIGCOMM'13*.

[13] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al. B4: Experience with a globally-deployed software defined wan. In *SIGCOMM'13*.

[14] T. Jepsen, M. Moshref, A. Carzaniga, N. Foster, and R. Soulé. Life in the fast lane: A line-rate linear road. In *SOSR'18*, page 10. ACM, 2018.

[15] X. Jin, X. Li, H. Zhang, N. Foster, J. Lee, R. Soulé, C. Kim, and I. Stoica. Netchain: Scale-free sub-rtt coordination. In *NSDI'18*.

[16] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica. Netcache: Balancing key-value stores with fast in-network caching. In *SOSP'17*.

[17] D. Koutsonikolas, C.-C. Wang, and Y. C. Hu. Ccack: Efficient network coding based opportunistic routing through cumulative coded acknowledgments. In *INFOCOM 2010*.

[18] V. Mishra, D. Verma, C. Williams, and K. Marcus. Comparing software defined architectures for coalition operations. In *2017 International Conference on Military Communications and Information Systems (ICMCIS)*, pages 1–7, May 2017.

[19] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, and C. Kim. Language-directed hardware design for network performance monitoring. In *SIGCOMM'17*.

[20] R. Rajagopalan and P. K. Varshney. Data-aggregation techniques in sensor networks: A survey. *IEEE Communications Surveys Tutorials*, 8(4):48–63, Fourth 2006.

[21] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis. In-network computation is a dumb idea whose time has come. In *SOSR'17*.

[22] B. Schlinker, H. Kim, T. Cui, E. Katz-Bassett, H. V. Madhyastha, I. Cunha, J. Quinn, S. Hasan, P. Lapukhov, and H. Zeng. Engineering egress with edge fabric: steering oceans of content to the world. In *SIGCOMM'17*.

[23] A. Sgora, D. J. Vergados, and D. D. Vergados. A survey of tdma scheduling schemes in wireless multihop networks. *ACM Comput. Surv.*, 47(3):53:1–53:39, Apr. 2015.

[24] N. K. Sharma, A. Kaufmann, T. E. Anderson, A. Krishnamurthy, J. Nelson, and S. Peter. Evaluating the power of flexible packet processing for network resource allocation. In *NSDI*, pages 67–82, 2017.

[25] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown, and S. Licking. Packet transactions: High-level programming for line-rate switches. In *SIGCOMM'16*.

[26] J. Suurballe. Disjoint paths in a network. *Networks*, 4(2):125–145, 1974.

[27] K. Tan, H. Liu, J. Zhang, Y. Zhang, J. Fang, and G. M. Voelker. Sora: high-performance software radio using general-purpose multi-core processors. *Communications of the ACM*, 54(1):99–107, 2011.

[28] Q. Xiang. *In-network processing for mission-critical wireless networked sensing and control: A real-time, efficiency, and resiliency perspective*. Wayne State University, 2014.

[29] Q. Xiang, H. Zhang, J. Wang, G. Xing, S. Lin, and X. Liu. On optimal diversity in network-coding-based routing in wireless networks. In *INFOCOM'2015*.

[30] Q. Xiang, H. Zhang, J. Xu, X. Liu, and L. Rittle. When in-network processing meets time: Complexity and effects of joint optimization in wireless sensor networks. *IEEE TMC*, 10(10):1488–1502, Oct 2011.