On hybrid network coding for visual traffic surveillance

Ling, Chih Wei; Datta, Anwitaman; Xu, Jun

2019

Ling, C. W., Datta, A. & Xu, J. (2019). On hybrid network coding for visual traffic surveillance. *Future Generation Computer Systems*, 100, 440-455.
<https://dx.doi.org/10.1016/j.future.2019.05.044>

<https://hdl.handle.net/10356/150612>

<https://doi.org/10.1016/j.future.2019.05.044>

© 2019 Elsevier B.V. All rights reserved. This paper was published in *Future Generation Computer Systems* and is made available with permission of Elsevier B.V.

Downloaded on 26 Apr 2023 07:13:35 SGT

On hybrid network coding for visual traffic surveillance

Chih Wei Ling^{a,b,c}, Anwitaman Datta^b, Jun Xu^a

^a*Western Digital Technology, Singapore*

^b*Nanyang Technological University, Singapore*

^c*cwling22@gmail.com*

Abstract

A large volume of data is generated by traffic surveillance devices such as cameras and sensors integrated into an intelligent transportation system (ITS), a subfield of the Internet of Things (IoT). We argue that network coding can be applied to leverage on an emerging fog architecture that relies on edge resources, to achieve higher throughput, saving up network bandwidth, and provide resilience to link failures, while also achieving simple obfuscation against wire-tapping attacks by linearly combining the source packets. There are two broad linear network coding paradigms in the literature - deterministic and random network coding, each with their own strengths and limitations. With the aid of software-defined network (SDN), we rethink about the possibility of applying a hybrid approach to deal with networks at different scales. Under network conditions that reflect expected network properties of an ITS, our simulation results show that the proposed hybrid approach performs better than other alternates.

Keywords: Internet of Things, Fog/Edge computing, Network Coding, Software-defined Network, Traffic Visual Surveillance.

1. Introduction

Video surveillance [3, 4, 5, 6] is one of the most important data sources of IoT. In particular, traffic visual surveillance is one of the essential components of ITS with the goal of monitoring and controlling the traffic flow and events over the entire transportation system of either a city, a state or a country, so that the whole transportation system can operate efficiently for the welfare of traffic users. During traffic surveillance, a vast amount of data distributed geographically across multiple regions is produced. Most

of the traffic surveillance devices are deployed at the network edges and the generated data has to be transmitted across the network from those devices to the back-end data centers for further processing.

A distributed multilevel storage architecture incorporating edge devices and networks for traffic visual surveillance in ITS [1] seems a natural way to operate an ITS at scale. To deal with the scale of ITS, the natural option for the back-end is to deploy cloud-scale infrastructure; however, as the volumes of data continue to increase, we argue that it is essential to augment the centralized back-end with fog/edge computing [7, 8]. Most of the cameras are installed at the network edges and the generated visual data has to be transmitted from the camera sources to the data center across the network for further processing which causes inefficient use of network resources. Moreover, the latency of responses to the requests of traffic users increases due to the back-and-forth transmissions if the requests are not processed locally on the spot. Fog computing extends the traditional cloud by delegating (some) tasks to the edge resources. Characteristics of fog computing, such as location awareness, low latency, geo-distribution, are suitable to meet the needs of ITS services.

The recent development in network streaming combines the idea of OpenFlow [23] and multimedia multicast streaming [31] so that streaming management can be handled efficiently. OpenFlow is one of the realizations of software-defined network (SDN) [20, 21, 22]. In the architecture of SDN, there is a controller (residing in the control plane, which can be logically centralized but physically distributed over several servers [25, 26, 27]) that can exert control logic through the control flow on the network switches (residing in the data plane) since it has the global view of the network by collecting the feedback from network devices. Additionally, the SDN controller can function as a *network operating system* [24] to control the underlying network resources, e.g., network links and switches, storage devices, computing nodes, etc.

A simplified view of the system is that we have multiple sources, from which data needs to be sent to a sink. Network coding has been shown to be more efficient than sending the raw data from each source through multiple intermediate nodes (store and forward) to the eventual sink [37, 38, 39, 40]. There are two variants of network coding - deterministic and randomized. Typically, there is no global view in a fast-evolving network, which is an important prerequisite when applying deterministic network coding effectively. Consequently, the research community advocates the use of random network

coding [39, 40, 41]. However, there are three issues associated with the random network coding approach:

1. There is bandwidth resource loss due to the probabilistic decoding nature of random network coding where the local coding coefficients are generated randomly.
2. There is an additional overhead (3%-6%) for tagging the generated random coefficients to the coded packets [41, 48] for constructing the decoding matrices at the sinks. For a large volume of data¹, as is the case for traffic surveillance applications, this additional overhead can be significant extra traffic in absolute.
3. Random network coding requires a larger field to assure higher decoding probability, which in turn increases the re-encoding complexity at the intermediate nodes [43] and decoding complexity at the sinks due to the matrix inversion [44]. The experiment results in [49] show that the overhead of random network coding cannot be neglected.

Hence, random network coding may not be appropriate for real-time visual surveillance; other than the original issue of packet loss over the network in real-world scenarios, there will be additional capacity wastage and computational complexity due to random network coding.

From the previous discussion, we can expect that both network coding paradigms perform differently in a distributed multilevel storage architecture when deployed on the networks at different scales. For the networks at small to medium scales, the probabilistic decoding nature of random network coding may burden the network load since a lot of retransmissions are required in the delivery scheme to combat the issue of linear dependency between coding coefficients. Moreover, we cannot neglect the tag overhead and the computational complexity in the case of traffic surveillance applications. It follows that the deterministic network coding is a more economical option for the networks at small to medium scales (with low dynamicity, and feasibility to acquire a reasonable estimate of the global state within a short latency); relying on the goodness of SDN, where the centralized controller has the global view of the network system, we can use the deterministic network coding to find the global optimum for the network system at this

¹Cisco predicts that a smart city of 1 million will generate 180 million gigabytes of data per day by 2019, where an intelligent transportation system will take a substantial amount of the data [2].

scale. Nonetheless, the synchronization between routers in the network for the deterministic network coding may be costly for large networks. Therefore, deterministic network coding may perform better on the networks at small to medium scales, while random network coding may perform better on the networks at medium to large scales (where it is unrealistic to assume almost real-time global information and coordination).

Hence, we advocate a hybrid approach of both network coding paradigms and software-defined framework for traffic visual surveillance in ITS over networks so that we can get the benefits provided by each network coding paradigm. Both deterministic and random variants are used in conjunction at different parts of the network for adapting to the network dynamics. Random network coding is robust and performs better in asynchronous network events but with the cost of higher bandwidth overhead and computational cost while deterministic network coding provides better throughput benefits but not suitable for asynchronous network events.

We leverage the fact that the internet is divided into a large number of different regions of administrative control commonly called *autonomous systems* (ASs). While a specific routing protocol such as Open Shortest Path First (OSPF), Routing Information Protocol (RIP) or Intermediate System to Intermediate System (IS-IS) is used within an AS, Border Gateway Protocol (BGP) is the de facto inter-autonomous system routing protocol [15]. It is used by BGP routers associated with different ASs to exchange the reachability information and determine the routes for packets traversing through multiple ASs. Several researchers have studied the problems of BGP, such as routing instability [16, 17], slow convergence [18], and scalability issues [19]. These problems can lead to packet loss across ASs. In [18], Labovitz et al. found that BGP convergence was delayed three minutes on average, up to fifteen minutes in extreme cases, due to path enumeration. During this time, data traffic suffered increased packet loss where the average packet loss increased from less than 1% to 30% at peak. In contrast, OSPF and IS-IS are fast-converging protocols, thus the network within an AS can achieve stability quickly. Since the failure rate of the infrastructures and devices in every AS is different, we focus on the aspect of protocols and assume that the data transmissions within an AS are more stable (small network change) than the data transmissions across multiple ASs.

By assuming that we have a collection of connected clusters of nodes (each cluster models an AS), we employ the deterministic network coding for the video transmissions within a cluster and the random network coding for

the video transmissions across the clusters. The rationale that we use the deterministic network coding for the video transmissions within a cluster in our hybrid approach is because of the small network change (say 1% or less) within clusters.

We assume that the size of the network is at the scale of a city (a small to medium-sized network), so the latency between the centralized controller and the switches is negligible. To that end, we have done several simulations to evaluate the performance of the hybrid approach under different network scenarios over a collection of clusters.

The main contribution of this work is demonstrating the practicality of a hybrid network coding approach facilitated by software defined networking (SDN) controllers, in which the data transmissions within a cluster leverages deterministic network coding while the data transmissions across clusters is realized by random network coding. We provide a theoretical model to describe (in Section 3) the hybrid network coding mechanism, and create a simulation platform using ns-3 (described in Section 4) to determine the efficacy of our approach. By exploring the parameter space, we identify a wide range of realistic scenarios where the benefits of our hybrid approach is significant, particularly, we observed upto 30% better network throughput than the random network coding approach in certain instances. Thus, the simulator not only helps in establishing the efficacy of the hybrid approach, but it also serves as a tool to determine when and whether to apply the hybrid approach, depending on the expected network conditions.

With this in mind, we next summarize related works broadly from the areas of network coding, cloud computing enabled intelligent systems and software-defined networking, several of which we leverage and put together to realize our proposed approach.

2. Literature Review

While the focus of the work is the application of hybrid network coding, its realization requires software defined networking, and pushing parts of the computation to the edges of a network. Accordingly, we next provide an overview of related works, as well as works that we leverage upon.

2.1. Network Coding

Network coding enables information symbols from different flows to be combined at intermediate nodes of the network to reduce the network band-

width, and so the application of network coding has the potential to save up a tremendous amount of network bandwidth and increase throughput. Ahlswede et al. [37] pioneered the field of network coding, studied the problem of single source multicast scenario and characterized the coding rate region for the network information flow. They showed that the source can multicast a specific information to all receivers at a rate approaching the source rate as long as the size of the block code is sufficiently large, which can be regarded as the generalization of Max-flow Min-cut theorem for information flow. Subsequently, Li et al. [38] used tools from graph theory and linear algebra to describe an optimal model for directed acyclic graphs and showed that linear coding, where the information symbols are combined linearly (finitely many symbols) at the nodes of the graph, can be used to achieve the optimum (source rate). Koetter et al. [39] opened up another formulation for network coding, which is purely algebraic and based on algebraic geometry and matrix theory. This work also provided the random network coding framework, where the coefficients of the linear combination can be chosen randomly. Jaggi et al. [40] provided the first polynomial time construction for deterministic network coding.

2.1.1. Realizing Network Coding

Chou et al. [41, 42] were the first to consider network coding under practical settings. They provided the first practical construction of random network coding by using the idea of tagging the random coefficients to coded packets, generation numbers from source and buffering of coded packets at nodes. Their scheme is decentralized, which obviates the need to know the global network topology and the need to synchronize the underlying network nodes, and thus it is robust against delays and link failures. In the realm of peer-to-peer content distribution, a group of Microsoft researchers [45] applied random network coding to the BitTorrent-like systems, which is named as *Avalanche*. By simulation study, they claimed that *Avalanche* had a throughput gain of 20%-30% over encoding at file server only.

Sundararajan et al [50] made the first attempt to combine network coding in the TCP layer with a minimal change and measured the performance of TCP without network coding. Subsequently, Kim et. al. [51] analyzed the performance of TCP and TCP with network coding (TCP/NC) in lossy networks by extending the framework built by [52]. Their analytical results showed that network coding masks random erasures from TCP, thus preventing TCP's performance degradation in lossy networks with significant

throughput gain for TCP/NC over traditional TCP.

2.1.2. Simulation of Network Coding

The work in [54] extended the work of Chou et al. [41] and implemented over SDN by using mininet [32]. They argued that the probabilistic nature of random network coding would hamper the decoding performance, so they chose to use a deterministic version of network coding [40] in their realization. They also used the idea of tagging, generation numbers and buffering at SDN's switches from [41] to enhance their system. However, their work lacked a comprehensive evaluation on their system prototype and did not have any theoretical analysis. Szabo et al. [55] did a similar work to [54] but they changed the underlying network coding strategy to random version and made comprehensive experiments and evaluations. However, their evaluation focused on very general data and network setting, and not specified to multimedia data.

The presented work builds on [54, 55] which effectuate network coding with SDN; however, we go one step further by realizing both deterministic and randomized network coding paradigms on different parts of the network concurrently, in particular, we implement deterministic variant within an AS and random variant across multiple ASs.

2.2. Fog Computing and Software-defined Concepts

Deploying ITSs over a cloud platform enables manipulating, provisioning and storing large amounts of data from diverse and disparate sources. However, this is accompanied with several challenges when deploying in real world scenarios, such as user admission control, resource scalability and optimal utilization and workload balancing across virtual machines. The work in [5] surveyed the many issues associated with an ITS. The scope of this work is limited to only managing the volume of traffic in the backend network over which an ITS would rely upon, and other design aspects of ITS are beyond the scope of this paper. Specifically, we anticipate an ITS where the back-end cloud is augmented by resources at the edges (local surveillance systems which are part of the overall system), to enable location awareness, exploit geographic proximity and reduce actuation latency, as well as process some data locally, and send only some of the processed data to back-end data centers, possibly using the network resources opportunistically (e.g., shipping a bulk of the data when the network is underutilized by other traffic).

The final missing puzzle in orchestrating the edge resources to realize our proposed (hybrid) network coding mechanism is software defined networking (SDN). Traditionally, the control plane (an abstraction layer, making decisions where the traffic is sent) and the data plane (an abstraction layer forwarding traffic to the selected destination decided by control plane) are combined together in each network node [20, 21, 22]. This has proven restrictive for flow management [20, 9, 10, 11], since the only way to make adjustments is reconfiguring the network devices once the initial forwarding policies have been defined. Thus, network operators are unable to respond quickly to the fast-paced changes of network traffic and the impact of big data [9, 10]. These have led to the advent of SDN [20, 21, 22, 23, 25, 26, 27], which addresses the dynamic nature of network functions and intelligent applications while lowering operating costs through abstraction of lower level functionality and simplified software, hardware and management. This is done by separating the control plane from the data plane, which allows network administrators to manage and provision network services through the control plane that is logically centralized. OpenFlow protocol [23] is one of the realizations of SDN. SDN is leveraged to design new systems. For example, NOX is a *network operating system* [24] built on top of OpenFlow switches that provides a uniform and centralized programmatic interface to observe and control the entire network, which is analogous to read and write access to various resources provided by computer operating systems. In the design of NOX, there is a southbound connection connecting the control plane NOX controller with the OpenFlow switches (OpenFlow protocol) in the data plane, and a northbound connection connecting the NOX controller with network services and applications in the application layer. We rely on the feedback about the network received from the network switches as facilitated by *network operating system* [24] to implement our hybrid network coding.

3. Hybrid Network Coding over a Hierarchical Topology

We model the network to comprise clusters of nodes where each cluster operates as an autonomous system (AS) with small to medium scale of (geographic) distribution (See Figure 1). The validity of this underlying assumption determines the efficacy of the deterministic network coding to be applied within individual clusters, since deterministic network coding would implicitly require low propagation latency and stability of the network so

that the network conditions can be correctly estimated for a period of time.² These clusters are interconnected and thus the collection of clusters occupies a larger geographical region. Therefore, the data has longer traveling time or hops (higher latency) when traveling across clusters than traveling within a cluster. Moreover, the cluster is stable where the events of network change (e.g., churn or nodes joining and leaving) within a cluster are relatively rare. We also assume that clusters have micro data-centers [35, 36] to provide computing or storage resources at the edge, catering to the need of network users, such as storage backup for video data in our case. The data sources (camera) can transmit the video data to the data-centers located in the same cluster for time sensitive analysis and action, as well as for the purpose of backup. The data may also be temporarily be further redundantly be backed up in neighboring clusters. Eventually, the data (possibly processed) is opportunistically shipped to back-end data-centers. In this paper, we simulate the data transmissions of the original video frames (MPEG’s IPBB structure of video [58]) from source cameras to the data centers without any processing at the edge nodes over the network by hybrid network coding to showcase the feasibility and advantages of hybrid approach. However, we can also send only the important video layers, which are processed at the edge nodes and have sufficient resolution for traffic surveillance, to the data centers where the traffic control center is located in proximity while keeping the trivial video layers on the micro-data centers which are attached locally to the network edge and near to the traffic surveillance devices. This idea is enabled by the layered structure of video [33, 34]. Note that the general network coding strategy itself is in principle independent of whether the original data is sent, or only some processed version is sent instead.

The proposed hybrid network coding approach can be deployed in general in such a hierarchical network topology. The proximate motivation of this work is visual traffic surveillance for intelligent transportation systems because, such an application naturally fits the assumed model in many ways (city-scale, multi-AS network, presence of edge-resources is essential to meet other ITS requirements such as low-latency response by localization), and even when there is only a small percentage improvement in terms of network traffic reduction or throughput increase, in absolute terms the benefit is still

²Our experiments explore the parameter space to capture scenarios where the network stability assumption is valid/violated.

huge since the total volume of data involved can be humongous.

Figure 1: A collection of interconnected clusters, forming a hierarchical topology. In our hybrid approach, we apply deterministic network coding for a *within-cluster multicast group* and apply random network coding for a *cross-cluster multicast group*. A within-cluster multicast group includes a source and all the collocated data centers; a cross-cluster multicast group includes a source and all the non-collocated data centers at the target cluster.

As mentioned earlier, the main idea of our hybrid approach is that we apply deterministic network coding for the video data transmissions *within a cluster* to have the benefit of higher throughput gain since no linear dependency issue incurred by random network coding. Furthermore, the hybrid approach provides a lower computational cost in terms of finite field computation since a smaller field size is applicable. For the data transmissions *across clusters*, we apply random network coding for the robustness to combat packet loss rates and dynamical network change, which deterministic network coding does not cope well with. With this set-up, we expect to get the benefits from both network coding paradigms.

The conventional OSI model for the network was developed before the development of the concept of network coding. So it is natural that the OSI model does not encompass the service of network coding in any layer. To gain the benefits provided by network coding, it is essential that any future enhancement to the existing models or any new specification of network protocols or models must include the network coding operations, be it in any of the layers. Therefore, it is an open-ended issue as to at which layer should network coding be implemented. We believe, with the enhanced computational power of network switches, it is feasible to apply the network coding

services in any of the layers above the network layer in the conventional architecture. Moreover, the current development of active network/programmable network has enabled the possibility of inserting new operations into network switches, so it is natural to explore whether we can realize the component above the network layer. The implementation of our network coding scheme in application layer is not particularly optimized, and the purpose of the presented study was to explore the conceptual foundations. We expect to build upon it in future work, to explore the practicability by prototyping and experimentation to determine which network layer should the network coding be implemented and optimized to find suitable trade-off between cost and benefit.

The multicasting model in a real-world scenario is most likely to be the intersession multicasting, where each data center may demand a *different* subset of the coded data from the source. While an intersession network coding scheme may help identify a better cost benefit trade-off, this problem is still under study and there is no general solution for it [46, 47]. In our current work, we thus assume intrasession multicasting both because of its simplicity but also because it is a better understood model in the literature, where the data centers demand the *same* set of coded data from the source as our model in the simulation. While the assumption means we may miss on a better solution that could potentially be identified with a intersession model, the current intrasession model nevertheless provides practical value by reducing the cost within individual sessions. We defer the exploration of intersession network coding for future work. First we define some terminologies for the discussion. For the camera sources and data centers located in the same cluster, we say that they are *collocated*; otherwise, they are *non-collocated*. For the video transmissions within a cluster, each source and all the collocated data centers form a *within-cluster multicast group* where all the data centers demand the same data from the source. Similarly, for the video transmissions across clusters, each multicast source and all the non-collocated data centers at the target cluster form a *cross-cluster multicast group* where all the data centers demand the same data as well. We denote a multicast group as $(s, t \in T_s \subset T)$ where T_s denotes all the data centers in the target cluster.

Given a clustered network $G = (V, E)$, we have a set of sources $s \in S \subseteq V$ to model the set of cameras and a set of receivers $t \in T \subseteq V$ to model the set of collocated or non-collocated data centers where $S \cap T = \emptyset$. In other words, we want to multicast video data from a set of cameras to data centers

connected by the clustered network. We denote the set of edge capacities of G as $C = \{c_{ij} : (i, j) \in E\}$. Every source s multicasts with rate R_s^t to sink t . Moreover, we denote $z_{ij}^{(t,s)}$ to be the allocated bandwidth across the edge (i, j) when the connection (t, s) is established. We also denote z_{ij}^s to be the *effective bandwidth* flowing through the edge (i, j) for the intrasession multicast of a multicast group $(s, t \in T_s \subset T)$. Each intrasession multicast of a multicast group $(s, t \in T_s \subset T)$ can then be described as follows:

$$\begin{aligned}
& \max \quad \chi, \\
& \text{s.t.} \quad R_s^t \geq \chi, \quad \forall t \in T_s \subset T, \\
& \quad c_{ij} \geq z_{ij}^s, \quad \forall (i, j) \in E, \\
& \quad z_{ij}^s \geq z_{ij}^{(t,s)}, \quad \forall (i, j) \in E, \forall t \in T_s \subset T, \\
& \quad \sum_{\{j|(i,j) \in E\}} z_{ij}^{(t,s)} - \sum_{\{j|(j,i) \in E\}} z_{ji}^{(t,s)} = \sigma_i^{(t,s)}, \quad \forall i \in V, \forall t \in T_s \subset T, \\
& \quad z_{ij}^{(t,s)} \geq 0 \text{ and } z_{ij}^s \geq 0, \quad \forall (i, j) \in E, \forall t \in T_s \subset T,
\end{aligned}$$

where

$$\sigma_i^{(t,s)} = \begin{cases} R_s^t & \text{if } i = s \in S; \\ -R_s^t & \text{if } i = t \in T_s \subset T; \\ 0 & \text{otherwise.} \end{cases}$$

The first constraint assures that the min-cut is at least χ for each multicast group. The second and third constraints reflect that the effective volume of data flowing through an edge is capped by the edge capacity. The fourth constraint on the summation of information flows through the network nodes that can be seen as a generalization of Kirchhoff's law; the source generates information flow (indicating positive value) to be consumed by sinks (indicating negative value), passing through intermediate nodes with no information loss (zero).

By using Fold-Fulkerson algorithm [13, 14], we can find the min-cut solution to all the connections (t, s) where $t \in T$; each connection (t, s) has the same min-cut value χ and a set of χ edge-disjoint paths from the source s to the sink t . The union of these sets of edge-disjoint paths (some of the paths from different sets may share the same edge) forms a directed acyclic graph H . If any path contains a cycle, the resulting flow around the cycle is zero (a loopback) and we can eliminate the cycle. The coding coefficients of either of deterministic or randomized network coding is then assigned accordingly.

Understanding the exact algorithm of assigning the coding coefficients is beyond the scope of this paper, however, we give a simple description about the algorithm, and refer the interested reader to the literature for in depth and rigorous expositions on the algorithm and proof of correctness [38, 40]. The description we give is high level and overlooks a number of technical details. The main idea of the algorithm is that the choice of the coding vector should preserve the property that the set of the information flows from the source to the sinks is of “ χ -degree of freedom” along the edge-disjoint paths; in other words, the χ flows should be linearly independent. This is possible by maintaining a set of linearly independent coding vectors for each sink $t \in T$ indexed by the edge-disjoint paths when transversing these paths in an upstream to downstream order. The algorithm visits the edges of the directed acyclic graph H in a topological order³ (thus ensuring that no edge is visited before any of its ancestors in the network) and assigns coding vectors to them. Each assigned coding vector preserves the aforementioned property for all downstream sinks up to the edge to which it is assigned.

While the model described above is general purpose supporting arbitrary location distribution of source and destinations, our simulation experiments use a simplified set-up, where we assume that each video source multicasts a video file to all the collocated data centers and also to all the non-collocated data centers located at another target cluster. Other possibilities, such as destinations being collocated, or only in non-collocated data centers can still be obtained as special cases from the simplified set-up by further imposing that there are no transmissions over certain edges within or across the networks. To elaborate further, suppose we have 4 clusters, each cluster has 50 sources, 150 intermediate nodes, and 2 sinks (edge data-centers). For the simplicity of the simulation, we assume that the sources located in the cluster i send the generated video to the collocated data-centers and to the non-collocated data-centers in cluster $i + 1$ (with warp). It follows that there are 200 within-cluster multicast groups (each group has 1 source and 2 collocated data-centers) and 200 cross-cluster multicast groups (each group has 1 source and 2 non-collocated data centers) in total. Next, in Section 4 we describe our simulation set-up in greater detail, before discussing the results in Section 5.

³A topological order is a partial order. Such an order exists for the edges of any acyclic graph.

4. Simulation System Architecture

As mentioned in Section 1, we propose that a distributed multilevel storage architecture (DMSI) can be realized by incorporating edge devices and networks for traffic visual surveillance in ITS, and in particular, we argue that hybrid network coding can be employed to achieve judicious management of the humongous network traffic in such systems.

Accordingly, in our simulation, we want to study the performance of the proposed hybrid network coding approach (Section 3). The performance can be quantified using different metrics such as network throughput, latency and computational cost at network devices. However, there are also other subtle aspects which are harder to capture, such as the amount of global knowledge required, the complexity of synchronization between network nodes, etc. Yet, capturing the influence of such implicit system behaviours is essential for a rigorous treatment, and there-in lies the challenges as well as our contributions in designing a suitable simulation framework.

A live deployment and debugging of every implementation iteration is not practical. The exploration of coding parameters to determine a good fit to specific network conditions before an actual deployment is also essential. We thus see our simulator not only as a means to validate the efficacy of the proposed approach, but also an essential tool that can be used to fine-tune implementations before real world deployment. Accordingly, we elaborate in detail the design of our simulation framework.

We will start by introducing the software tools we use, in Section 4.1. Then, we describe the software architecture of the simulation and its inner components in the following subsections. We will present and elaborate the network model (including network topology, software-defined network, delay analysis, and network link error model), network coding model (including network coding paradigms and transmission policy) and software architecture for our simulation in Section 4.2, Section 4.3 and Section 4.4 respectively. In addition, we give our assumptions for the simulation system and analyze the reasons behind these assumptions. We also highlight the limitations of the presented simulation system.

4.1. Software Tools

We first introduce the software packages and libraries we leverage upon.

4.1.1. *ns-3*

The ns-3 simulator [63] is a discrete-event network simulator targeted primarily for research and educational use. In brief, ns-3 provides the models of how packet data networks work and perform, and provides a simulation engine for users to conduct simulation experiments. ns-3 is mainly supported for Linux, OS X, and FreeBSD and is under the GNU GPLv2 license. Note that ns-3 is not backward compatible with its predecessor, ns-2.

The reason we choose to use ns-3 is that we want to investigate the video transmissions over large SDN network by using network coding methods, which is difficult to perform with real systems at scale. This difficulty arises both in terms of having access to such an infrastructure, and also from the impracticality of testing and debugging each implementation, idea or hypothesis over a real infrastructure in a controlled and reproducible manner. Moreover, ns-3 provides an additional module to set up SDN/Openflow network and an interface to integrate with BRITE [64, 65].

4.1.2. *BRITE*

BRITE (*Boston university Representative Internet Topology gEnerator*) [64, 65] is a universal topology generator which adheres to the following three design principles: representativeness, inclusiveness, and interoperability. Representativeness allows the users to construct synthetic topologies that accurately reflect many aspects of the actual Internet topology (e.g. hierarchical structure, degree distribution, etc.). Inclusiveness combines the powers of different generation models in one single generation tool. Interoperability provides interfaces to widely-used simulation applications such as ns-3 and OmNet++ as well as visualization applications.

By coupling BRITE with ns-3, we could create large Internet topologies quickly and efficiently, while taking advantage of ns-3's simulation capabilities. There are several works [66] that have leveraged on such a close integration to simulate large networks. We will further describe the assumptions and parameters we have chosen for the generation model in subsequent Section 4.2.

For the sake of self-containedness, we summarize the meaning of the parameters to generate BRITE network in the following two tables (Tables 1 and 2) for the ease of reading. A detailed description can be found in [65].

Table 1: Flat Topology (AS Only or Router Only) Parameters

Parameter	Meaning	Values
N_{AS}	Number of ASs	int $N_{AS} \geq 1$
N_{RT}	Number of nodes in a AS	int $N_{RT} \geq 1$
Model	Network generation model	Waxman or Barabasi-Albert
α	Waxman-specific exponent	$0 < \alpha \leq 1$
β	Waxman-specific exponent	$0 < \alpha \leq 1$
Node Placement	How nodes are placed in the plane	random or heavy-tailed
m	Number of links per new node	int $m \geq 1$
Growth Type	How nodes join the network	Incremental or random
BWdist	Bandwidth assignment to links	float ≥ 0

Table 2: Top-down Hierarchical topology Parameters

Parameter	Meaning	Values
Edge Connection Method	Method for interconnecting router topologies	random or smallest- k degree
Total Router Nodes	Number of total router nodes of generated network	$N_{AS} * N_{RT}$

4.1.3. Kodo

Kodo [56] is an open source C++ library for implementing erasure correcting codes, in particular, network codes. The library is intended to be used as a building block for reliable communication protocols and systems, and for research on the implementation of network coding methods. To provide a research-friendly library, Kodo provides a number of algorithms and building blocks, which includes random linear network coding (RLNC), Reed-Solomon codes (RS), etc. In general, the implementation of RLNC follows the idea from Chou et. al. [41]. Furthermore, it is possible to integrate Kodo and ns-3, and test the performance of the network codes over wired and wireless networks [57].

4.2. Network Model for Simulation

In this section, we describe the network model being used in our simulation.

4.2.1. Network Topology

A common property of many large networks (e.g., the Internet) is that the node connectivities follow a scale-free power-law distribution [70, 71]. This feature was found to be a consequence of two generic mechanisms: first,

networks expand continuously by the addition of new nodes, and second, new nodes attach preferentially to sites that are already well connected. Since the IP traffic cameras connect to the network edge and forward the generated video data packet through routers in the network backbone to the data center, we assume that the network topology of traffic cameras, edge devices, routers and data centers, is also a scale-free network.

We model a scale-free network consisting of two levels of router nodes, where the top level (AS-level) consists of gateways that interconnect autonomous systems (AS) and the bottom level (RT-level) consists of interconnected routers (RT) within each autonomous system. Moreover, some router nodes from different ASs are interconnected with each other by using edge connection method such as random or smallest- k degree. We use the hierarchical generation model of BRITE to create the two-level network of routers. The hierarchical generation model first generates the AS-level topology, and then generates the RT-level topologies and connects them to the AS-level. Each level (top or bottom) can use one of the two probabilistic graph models for edge connection, either Waxman’s model [70] or Barabasi-Albert’s model [71], to connect a new node to the set of previously connected nodes.

4.2.2. Software-defined Network

SDN designers now face new choices in the design of SDN network; in particular, how centralized or distributed should the network control plane be? Fully physically centralized control is inadequate because it limits (i) responsiveness, (ii) reliability, and (iii) scalability. Thus, designers resort to a physically distributed control plane, on which a logically centralized control plane operates. Large SDN networks are partitioned in multiple controller domains/subnets; each controller is responsible for one domain/subnet, and the controllers of adjacent domains/subnets may need to communicate to enforce global policies [29]. In doing so, however, they face trade-offs between different consistency models and associated liveness properties [28].

In our model, we accordingly assume that there is a logically centralized SDN controller, which has the global knowledge of the network model, while there are many physically distributed SDN controllers, each controls an individual cluster (a domain/subnet) by collecting the information of network states. The communication between the control plane and data plane can be implemented by using TCP protocol to assure the control and feedback delivery. We also assume that the geographical scale of the network in our simulation is at the scale of the city such as Singapore, so that the prop-

agation delay, which is of the order of tens of microseconds, is negligible. The detailed delay analysis is given next in Subsection 4.2.3. We defer the applicability of our approach in a system with much larger geographic spread than at city-scale to future work.

4.2.3. Delay Analysis

The network latency is composed of four components, which are node processing latency, queuing latency, propagation latency, and transmission latency. The first two components of latency are based on the power of processing node while for the latter two components, propagation latency is based on the scale of the network in terms of travel distance across network links, and the transmission latency is based on the link bandwidths. Queuing latency of a data packet also depends on the network traffic at the time when the data packet is enqueued. In general, the network latency can be summarized in the following equation:

$$l = n + q + p + t, \quad (1)$$

where l is network latency, n is node processing latency, q is queuing latency, p is propagation latency, and t is transmission latency.

In our simulation, there are two types of network latencies: first, measurement and monitoring latency between the controller in the control plane and the switches in the data plane, and second, video data transmission latency between the sources (traffic cameras) and sinks (data centers) through the network routers. To use deterministic network coding, the first type of latency is crucial since the SDN controller has to collect the network state information (e.g., bandwidth, delay, etc) across the network in order to compute disjoint edge paths from source to sinks based on the collected network states. The first type of latency may occur in any random network coding scheme which requires global state information to compute multicasting paths [40]. To simplify the simulation setup, we assume that the first type of latency is negligible since we are simulating the video transmission over the network at the geographical scale of a city (confined within a diameter of 10-100 Km, with much shorter individual network links). The second type of latency occurs in both network coding variants, either deterministic network coding or random network coding.

4.3. Network Coding Model and Delivery Scheme

In this section, we describe our network coding model and delivery scheme (including transmission policy) over the simulated BRITE network.

Figure 2: **Implementation of Network Coding Module in UDP/IP model.** In the figure, the network coding layer is placed between the application layer and the transport layer (third row). First, a network coding module of a camera source encodes a video frame received from the upper application layer and passes the encoded data packets to a lower network layer for transmission (second column). Then, an intermediate router receives the incoming encoded data packets from the UDP sockets and linearly combines the packets at the network coding module. After that, the network coding module passes the linearly-combined data packets to the lower network layer for outgoing transmission (third column). Finally, a sink listens to its UDP sockets and decodes the incoming packets from the lower network layer at the network coding module (fourth column).

4.3.1. Transmission Policy

We perform the network coding between the application and transport layer of the UDP/IP model as shown in Figure 2. Although other proposals⁴ [53, 50, 51] apply network coding between the transport and medium access control layer, we implement it below the application layer to not change other layers within the protocol stack and keep the simulation simple.

⁴Those network coding schemes implemented between the transport and network layers are targeted for TCP protocol, which emphasizes the data delivery reliability over the reduced latency, and are thus more suitable for applications such as record insertion to a database, file transfer, bank transactions, email, etc. The acknowledgments (ACKs) from the TCP layer are critical in realizing those network coding schemes and that is the reason why embedding those schemes between the transport and network layers is necessary. For video streaming, however, the reduced latency is a more important aspect than the data delivery reliability. Thus, we opt for the use UDP protocol for live streaming, and accordingly implement the network coding scheme in the application layer.

4.3.2. Network Coding Module

The network coding module (See Figure 2) executes the two paradigms (either random or deterministic) of network coding.

For random linear network coding, we adopt the framework of [41]. We also compute two edge-disjoint paths from each source to every sink (by using Edmonds-Karp Algorithm [14]) and route the data packets along these paths. We use Kodo [56] to assist our implementation, which is a random linear network coding library under the framework of [41]. Each data source is equipped with one Kodo encoder object, and each router or sink is equipped with multiple Kodo decoder objects to receive and re-encode/decode data packets from the corresponding source.

For deterministic network coding, we adopt the algorithm of [40]. For the first stage of [40], we implement Edmonds-Karp algorithm to find the edge-disjoint paths from each source to every sink. Then we implement the second stage of the algorithm in the network coding layer.

During the encoding at the source, the network coding module treats each video frame as an object to be encoded. The object is partitioned into disjoint collections of data packets, referred to as *generations*, and coding is done only within generations. The number of data packets per generation is called the *generation size*. We handle the computation of generation size and packet size for random and deterministic network codings differently since it is necessary to tag the random coefficients generated by sources and routers to the transmitted data packets along the paths for random network coding.

For random network coding, we fix the maximum packet size by maximum transferable unit (MTU), which is 1472 bytes, and determine the generation size h_r by the following formula:

$$h_r = \left\lceil \frac{S_{frame}}{MTU - 20} \right\rceil = \left\lceil \frac{S_{frame}}{1472 - 20} \right\rceil = \left\lceil \frac{S_{frame}}{1452} \right\rceil, \quad (2)$$

where S_{frame} denotes the size of a video frame and the 20 bytes are reserved for the coefficient tag. Due to the version of Kodo library we used, which only supports Galois fields $GF(2)$, $GF(16)$, and $GF(256)$, we limit to the latter two finite fields in our simulation. We don't consider $GF(2)$ since the probability that this finite field incurs linear dependency is high. If a Galois field $GF(256)$ of size 256 (corresponds to 1 byte) is used, the reserved bytes are able to handle up to a generation size of 20 data packets, which will incur a tag size of 20 bytes; while if a Galois field $GF(16)$ of size 16 (corresponds

to half-byte) is used, the reserved bytes are able to handle up to a generation size of 40 data packets, which will also incur a tag size of 20 bytes. Then, each encoded packet size p per generation to be transmitted is determined by the following formula:

$$p = \begin{cases} S_{frame} + S_{tag} & , \text{ if } S_{frame} \leq 1452; \\ (S_{frame} + h_r - r)/h_r + S_{tag} & , \text{ if } S_{frame} > 1452, \end{cases} \quad (3)$$

where r is the remainder of generation size h_r divides frame size, $h_r - r$ is the data to be zero padded and S_{tag} is the tag size to be attached to the randomly-encoded data packets. The tag size is computed as follows:

$$S_{tag} = h_r * |GF(\cdot)| \text{ in bytes.} \quad (4)$$

Finally, the data packets (of one generation) are randomly-encoded where the local coefficients are randomly generated at the source or at the routers along the routing paths.

For deterministic network coding, since we don't need to attach the coefficient tag to the data packets, it is unnecessary to reserve a space. Hence, we fix the maximum packet size by maximum transferable unit (MTU), which is 1472 bytes, and determine the generation size h_d by the formula:

$$h_d = \left\lceil \frac{S_{frame}}{MTU} \right\rceil = \left\lceil \frac{S_{frame}}{1472} \right\rceil, \quad (5)$$

where S_{frame} denote the size of a video frame. Then, each encoded packet size p per generation to be transmitted is determined by the following formula:

$$p = \begin{cases} S_{frame} & , \text{ if } S_{frame} \leq 1472; \\ (S_{frame} + h_d - r)/h_d & , \text{ if } S_{frame} > 1472, \end{cases} \quad (6)$$

where r is the remainder of generation size h_d divides frame size and $h_d - r$ is the data to be zero-padded. Finally, the data packets (of one generation) are encoded where the local coefficients are generated by Jaggi's construction [40].

At data sinks, the decoding step is the reverse process of encoding, namely, solving a system of linear equations.

4.3.3. Handling packet losses

Given a pair of nodes connected by a channel in the simulated network, we use a packet retransmission scheme to combat packet losses over the channel.

Intuitively, more packet retransmissions over the channel can combat with packet loss when the probability of packet loss is small. This is because, if the first transmission of an encoded packet is lost over the channel, we can just send it again. For an intermediate node in the network, the parent node of the intermediate node will keep sending the cached encoded-packets at a certain rate to the intermediate node as long as the elapsed time lies within the assigned time constraint of a video frame. The assigned time constraint of a video frame should be smaller than the generated time of a video frame. For a 24-fps video, the generated time of a video frame is about 40 milliseconds; for a 48-fps video, the generated time of a video frame is about 20 milliseconds.

For deterministic network coding, the intermediate node will wait until getting enough incoming packets from its parent nodes before linearly combining the received packets and forwarding the encoded packets, provided that the elapsed time lies within the time constraint of a video frame.

For the random network coding, the intermediate node linearly combines the cached packets and sends the encoded packets as long as the intermediate node receives a new incoming packet; while the sink nodes can decode the video frame provided that the sink nodes receive enough linearly independent packets.

Given this retransmission policy, we don't need to explicitly detect packet loss at either network layer. We defer the usage of erasure coding instead of retransmissions to future work.

4.4. *Simulation Architecture*

Figure 3 shows the high-level abstraction of the software integration of the video trace helper, network coding module (including both network coding paradigms), ns-3 event scheduler and BRITE-generated network in the ns-3 platform.

The video trace helper is a component that reads each entry of a video trace file[62] of some video in MPEG-4[58, 59, 60, 61] format (by the *struct* data structure in C/C++) to get the metadata of each video frame such as generated time, frame type and frame size. Then, the helper computes the generation and packet sizes according to Equation 2 to Equation 6 and passes these parameters to network coding module for further coding process.

After that, the network coding module linearly encodes the video frame according to the chosen paradigm (either random or deterministic), and the event scheduler schedules a multicasting event and transmits the encoded

instance, 150) of nodes nodes. We then connect the clusters in such a way that some set of AS nodes in the cluster i are connected to another set of AS nodes in the cluster $i + 1$ randomly (with warping of the index i). For each cluster, we also connect certain number of data sinks (for instance, 2) to the AS nodes and certain number of sources (for instance, 50) to some router nodes⁵ of the cluster. For the indicated instances mentioned above, it would follow that there would be 200 sources and 8 sinks in total. As a result, the network of interconnected clusters, including sources and sinks, would comprise 808 nodes. Note that the indicated numbers were in fact used to instantiate most of the experiment results reported later in the paper, unless otherwise stated. In Table 3, we accordingly provide the parameters for the component cluster of 150 nodes. All 4 component clusters have the same parameters. We also experimented with several other parameter choices with similar qualitative results, of which we further report results from experiments on component clusters generated using parameter choices indicated in Table 4. Furthermore, we explored the effect of different packet loss rates in said networks.

Table 3: Default experiment set-up: Parameters for the component cluster of 150 nodes generated with BRITE

BRITE Network	Edge Connection Method	AS Model	BWdist	N_{AS}	Node Placement	Growth Type	m
Component Cluster	3-degree	Waxman(0,15,0.2)	400Mbps	6	random	incremental	2
BRITE Network	Total Router Nodes	RT Model	BWdist	N_{RT}	Node Placement	Growth Type	m
Component Cluster	150	Waxman(0,15,0.2)	400Mbps	25	random	incremental	2

Table 4: Parameters for the Component Cluster of 225 Nodes Generated by BRITE

BRITE Network	Edge Connection Method	AS Model	BWdist	N_{AS}	Node Placement	Growth Type	m
Component Cluster	3-degree	Waxman(0,15,0.2)	400Mbps	9	random	incremental	2
BRITE Network	Total Router Nodes	RT Model	BWdist	N_{RT}	Node Placement	Growth Type	m
Component Cluster	225	Waxman(0,15,0.2)	400Mbps	25	random	incremental	2

In our simulation, we assume that each video source multicasts a video file to the collocated data centers and to all the non-collocated data centers of a target cluster. For example, suppose we have 4 clusters, each cluster has 50 sources, 150 intermediate nodes, and 2 sinks. Then, there are 200 within-cluster multicasting groups (each group has 1 source and 2 collocated data centers) and 200 cross-cluster multicasting groups (each group has 1 source and 2 non-collocated data centers) in total.

⁵Either a leaf node or a non-leaf node of the cluster.

In a network link, the propagation speed is about two-thirds of the speed of light in vacuum [12, 30]. By simple calculation, it is of the order of tens of microseconds for any single data packet transmission at the city scale as Singapore. For example, if a network link has propagation distance of 20km (from either from the west to the central of Singapore or from the center to the east of Singapore), then $20\text{km}/200000\text{kms}^{-1} = 100$ microseconds. In fact, we can assume that the propagation delay is less than 100 microseconds at all network links in our simulation, which is obviously a very generous assumption. Nonetheless, we set the propagation delay as 100 microseconds across all links.

Continuing with the example of Singapore, the network bandwidth provided by ISPs to consumers is about 1Gbps or more for the uplink/downlink speed [69]. Based on the proliferation of broadband (which is anyway a prerequisite for an intelligent transportation system) it is reasonable to assume that each camera source has at least 100Mbps network bandwidth connecting to the edge devices. Moreover, we assume that all the links in a cluster have bandwidth 400Mbps and the links connecting clusters have bandwidth 800Mbps.

A video frame of MP4 standard is of about a few hundred bytes up to a few KBs. Put alternatively, transmitting 1GB of video data in an hour would entail 17MB per minute (2.267Mbps) in average, so we can expect that the transmission delay to be of the order of milliseconds.

For a 24 fps video, the time between two frames is about $1/24 \approx 40$ milliseconds. Therefore, we schedule to send a frame object every 40 milliseconds at each data source, where the object is passed to the network coding module (See Section 4.3.2) between the application and transport layer. For both within-cluster multicasting and cross-clusters multicasting, we retransmit a transmitted data object every 2 milliseconds to combat the linear dependency issue incurred by the random coefficients for random network coding paradigm and packet loss for both network coding paradigms.

5.2. Network Coding Schemes

There are three different schemes in comparison, which are *hybrid*, *all-random* and *all-deterministic*. For the hybrid scheme, we apply deterministic network coding equipped with $GF(16)$ for within-cluster multicasting and random network coding equipped with $GF(256)$ for cross-cluster multicasting (see Section 3). For the all-random scheme, we apply random network coding,

either equipped with $GF(16)$ or equipped with $GF(256)$, for both within-cluster multicasting and cross-cluster multicasting. Similarly, for the all-deterministic scheme, we apply deterministic network coding equipped with $GF(16)$ for both within-cluster multicasting and cross-cluster multicasting. Note that we use a small finite field (e.g., $GF(16)$) for deterministic network coding, since the number of sinks (2 in each multicasting group) is less than 16 and thus $GF(16)$ is sufficient for the decoding purpose [40, Theorem 3]. Also, note that we exclude the case of using random network coding for within-cluster multicasting and deterministic network coding for cross-cluster multicasting.

5.3. Network Throughput under Network Change and Churn

In this subsection, we compare the performance of different schemes under the network change and churn. As mentioned before, the rationale that we use deterministic network coding for data transmission within a cluster in our hybrid approach is because of the network within an AS is more stable (small network change, say 1% or less). Therefore, we evaluate the following scenario where we assume that the network change and churn occur at the nodes connecting between clusters by setting that 5% of nodes are down/up per unit time, throughout the simulation.

To find the network throughput, we adopt the method described as follows. Given a network topology, we first fix the network bandwidth⁶ and packet loss rate⁷ across all links. Then we pump the data at a constant bit rate (CBR) from sources to sinks to find the network saturated point. If we cannot get the saturated network by the previous CBR, we increase the CBR; if we get a saturated network, we decrease the CBR to find a non-saturated network.

We first give the result for the network of 808 nodes and then the network of 1208 nodes in Figures 4 and 5, respectively.

In the clustered network of 808 nodes (See Table 3), we see that the hybrid scheme performs the best in terms of total throughput⁸ than other network coding schemes under the network change (See Figure 4(c)). Furthermore, the hybrid scheme performs about 5% better than the all-random

⁶Recall that we set 400Mbps as the network bandwidth for all the links inside a cluster and 800Mbps for all the links connecting between clusters.

⁷For simplicity, we set as 0.

⁸Total throughput = within-cluster throughput + cross-cluster throughput.

Figure 4: **Comparison of Network Throughput of Hybrid Network Coding against All-random and All-deterministic Schemes under Network Change for the Clustered Network of 808 Nodes.** In the setting, 5% of nodes go down randomly at the start of every second and come up at the end of the second, repeatedly.

scheme with $GF(256)$ in terms of total throughput. Figure 4(a) shows that both the hybrid and all-deterministic schemes have the highest throughput in the case of within-cluster data transmissions. However, the all-deterministic scheme performs badly in the case of cross-cluster data transmissions (See Figure 4(b)). This can be attributed to the synchronization of packets at the intermediate routers causing the delay and reduced throughput. In addition, all-random scheme with $GF(16)$ has the worst performance. This can

be explained by the linear dependency nature of all-random scheme since more retransmissions are required to combat the high probability of packet collision for the $GF(16)$ at the sinks.

Figure 5: **Comparison of Network Throughput of Hybrid Network Coding against All-random and All-deterministic Schemes under Network Change for the Clustered Network of 1208 Nodes.** In the setting, 5% of nodes go down randomly at the start of every second and come up at the end of the second, repeatedly.

In the clustered network of 1208 nodes (See Table 4), the hybrid scheme performs better than the remaining schemes and is about 5% better than the all-random scheme with $GF(256)$ (See Figure 5). In fact, we expect the total throughput increases with respect to the increased network size. However,

Figure 6: **Comparison of Network Throughput of Hybrid Network Coding against All-random and All-deterministic Schemes under Network Change for the Clustered Network of 1908 Nodes.** In the setting, 5% of nodes go down randomly at the start of every second and come up at the end of the second, repeatedly.

our current results do not show this trend. This is because of the sample of network size is too small (e.g., only 808 nodes and 1208 nodes) and cannot be a representative of the realistic scenario showing the increasing trend. All the simulations were run on a single campus PC; we do not use the idea of parallel and distributed simulator across multiple PCs in our simulation [67, 68]. The number of supported nodes is bounded by the available memory of PC. In [68], they used the plain vanilla version of ns-3 and fit 750 thousand nodes in

the 32GB RAM of a single PC, which is about 42KB per node. However, in our simulation, the scenario is more complicated; we need to bind the kodo objects and data buffers to each node, which can tremendously reduce the number of supported nodes. In our experience, a network of 808 nodes with a set of kodo objects and data buffers binding to each node can occupy about 1GB to 2GB RAM. In addition, a complete run of the simulation with the network of 808 nodes can take up to 8 hours. Due to the limitation of the simulator to simulate a network with a considerable number of nodes and large packet on a single PC, we are unable to simulate a network of more than 5000 nodes at the moment.

Nonetheless, we explore the parameter space by increasing the number of sources. We use the previous clustered network of 1208 nodes, but we increase the number of source nodes of each cluster from 75 to 250, so the total number of source nodes is 1000 and the total number of nodes is 1908. Moreover, we increase the bandwidth from 400Mbps to 4Gbps for all the links. Figure 6 shows the performance of each network coding approach. The hybrid scheme performs 35% better than the all-random scheme when we tune the number of sources and network bandwidth.

5.4. Our Proposal under Other Network Setting

To evaluate our proposal under more general network setting, we remove the assumption that we use in Section 5.3; that is, the network change and churn not only occur at nodes connecting clusters, but also can occur at nodes within these clusters. Similar as Section 5.3, we evaluate the hybrid approach against other schemes under the following network scenarios with some parameter changes: there is a specific percentage of nodes that are down/up per unit time, throughout the simulation. In our simulation we choose 5 nodes per cluster and the total is 20 nodes.

Under the new setting, Figure 7(c) shows that the hybrid scheme is no longer the best performer in terms of total throughput. The reason is the high network change rate in the local networks, which causes the deterministic network coding underperforming in the case of within-cluster data transmissions. Therefore, the total throughput is not so good and close to the total throughput of all-random scheme.

5.5. Network Throughput of Different Packet Loss Rates

In this subsection, we look at the comparison of the average network throughput of these network coding schemes over time under different packet

Figure 7: **Comparison of Network Throughput of Hybrid Network Coding against All-random and All-deterministic Schemes under Network Change.** In the setting, we assume that there are 5 nodes per cluster (the total is 20 nodes) that are down/up per second.

loss rates. The comparison of the average network throughput over time against packet loss rates is shown in Figure 8.

To complete the picture, we show the improvement of hybrid network coding against the other schemes (in ratio) in Table 5. The leftmost column lists the network coding schemes we want to compare while the first row lists the packet loss rates. Each entry in the table represents the ratio between the network throughput of hybrid network coding against the scheme in the

Figure 8: **Average Network Throughput over Time against Packet Loss Rates.**

leftmost column. The ratio 1.279 means the hybrid coding performs 27.9% better while 0.928 means the hybrid coding performs 7.2% less. A *Nil* in the table means that the corresponding network coding scheme fails to operate for the given system set-up and thus there is no further meaningful performance comparison.

From Figure 8, we can see that when there is no packet loss, the all-deterministic scheme performs the best. It is followed by the hybrid scheme and the all-random scheme with $GF(256)$; while the all-random scheme with $GF(16)$ performs the worst. From the second column of Table 5, which compares the performance of the hybrid scheme with the other schemes, we note the values 0.928, 1.057 and 1.279 respectively. Note that a value lower than 1 indicates a performance better than the hybrid scheme, and likewise, the larger (than 1) the value, the worse the scheme is relative to the hybrid scheme. This coincides with the conventional wisdom that the deterministic network coding scheme performs better than the random scheme when the network is stable. This is because all-deterministic scheme doesn't require retransmissions when there is no packet loss but both of the hybrid and the all-random schemes require retransmissions to combat the linear dependency

issue.

When the packet loss rate is 0.05, the hybrid scheme has the highest average network throughput over time, and is followed by the all-random scheme with $GF(256)$ and the all-deterministic scheme; while the all-random scheme with $GF(16)$ has the lowest average network throughput over time. We can also see this trend in the third column of Table 5, where the ratios from the highest to the lowest are 1.162, 1.058 and 1.017. The deterministic network coding for the video transmissions within a cluster is functioning well when the packet loss rate is small, e.g., 0.05 packet loss rate. This is because all-random scheme requires more retransmissions to combat linear dependency for within-cluster multicasting when the field size is small, and all-deterministic scheme requires more retransmissions to combat packet loss for cross-cluster multicasting when traveling for long distance and involves synchronization between upstream and downstream nodes.

The all-random scheme with $GF(256)$ performs the best as the packet loss rate increases, as seen with our experiment with a loss rate of 0.10, while the all-deterministic scheme performs the worst. Moreover, the all-random scheme with $GF(256)$ performs better than the hybrid scheme since both the random network coding in the cases of within-cluster multicasting and cross-cluster multicasting of the all-random scheme are more resilient to the packet loss than the deterministic component in the hybrid scheme. From the fourth column of Table 5, the ratio of the performance of hybrid network coding against all-deterministic scheme is 1.357, against all-random scheme with $GF(16)$ is 1.041, and against all-random scheme with $GF(256)$ is 0.869.

When the packet loss rate is 0.20, the all-random scheme with $GF(256)$ is the only network coding scheme that is still functioning while the other network coding schemes fail to deliver under this packet loss rate. This is because the deterministic network coding in both hybrid and all-deterministic schemes falls apart when the packet loss rate is too high. Moreover, the all-random scheme with $GF(16)$ has serious linear dependency issue, so the sinks cannot get enough linearly dependent packets to decode within the assigned time constraint. In Table 5, the last column is filled with the word *Nil*.

Overall, we see that the packet loss rate dictates the optimal coding strategy. While this observation in itself is on expected lines, our simulation framework provides a tool to choose the best suited strategy judiciously and systematically for different network scale and settings.

Table 5: Improvement of Hybrid Network Coding in ratio against Other Schemes

Packet Loss Rates	No Loss	0.05	0.10	0.20
RLNC $GF(256)$	1.057	1.017	0.869	Nil
RLNC $GF(16)$	1.279	1.162	1.041	Nil
DLNC $GF(16)$	0.928	1.058	1.357	Nil

5.6. Further Discussion on Throughput of Hybrid Network Coding w.r.t. Parameter Space

Here, we give a discussion on the throughput of the hybrid scheme against the other schemes with respect to different parameters over clustered networks. We assume that the network change occur on the nodes connecting between the clusters (See Section 5.3). The parameters we consider in this discussion are network bandwidth, packet loss rate, network size, network topology/random graph model, etc. For the clustered networks, the total throughput is the sum of the within-cluster throughput and the cross-cluster throughput, so we need to consider the throughput advantage of the component network coding of each scheme in the cases of within-cluster multicasting and cross-cluster multicasting separately.

When the network bandwidth increases, the total throughput of every scheme increases since more data can be sent per unit time. However, we expect that the hybrid scheme performs better than other schemes since the deterministic component of the hybrid scheme for the within-cluster will increase at a steeper rate. Even though the deterministic component of the all-deterministic scheme in the case of within-cluster multicasting will show a similar trend as in the hybrid scheme, however, the deterministic component in the case of cross-cluster multicasting will have bad performance. Therefore, the total throughput will not be better than the hybrid scheme.

When the number of nodes increases, resulting in more available links and thus there are more alternative paths from sources to sinks, which implies that more data can be sent from sources to sinks and thus increases the total throughput (see Figure 6). Also, we expect that the hybrid scheme will have the best performance in terms of total throughput as the deterministic network coding component in the case of within-cluster multicasting will contribute more throughput than other schemes.

When the packet loss rate increases, we expect the throughput of each scheme decreases (see Section 5.5) since more retransmissions are required to combat packet loss and this increases the operation latency and thus reduces

the total throughput. Under this setup, we expect the all-random scheme performs better than the hybrid scheme since both the random network coding in the cases of within-cluster multicasting and cross-cluster multicasting of the all-random scheme are more resilient to the packet loss. However, we predict the hybrid scheme performs better than all-random scheme when the packet loss rate is about 5% when extending to the clustered networks according to our results.

6. Conclusion and Future Work

In conclusion, our hybrid approach works well when the network change and churn happens to the nodes connecting the clusters; however when the network changes and churn happen within clusters, the hybrid approach is no longer the best performer. In our proposed hybrid approach, we apply deterministic network coding (with $GF(16)$) for data transmissions within a cluster to provide higher throughput and lower computational cost and apply random network coding (with $GF(256)$) for data transmissions across clusters to deal with dynamical network change. Our proposed hybrid approach performs about 30% better than the all-random scheme with $GF(256)$ when the number of sources is 1000 and the network bandwidth is 4Gbps for all the links in the clustered network (including sources) of 1908 nodes. This should fit into the realistic scenario where the connection of nodes within a local network is stable while the connection of nodes between different networks are unstable.

Overall, the hybrid approach is the best performer for certain ranges of parameter choices capturing the network dynamics and topology, while the randomized coding strategy is the sole viable strategy when the dynamicity in the network is high. Our simulation framework serves as a practical tool to determine the optimal strategy given a network set-up and conditions.

Acknowledgment

The authors would like to thank Junpeng Niu and Wen Jun Tan in assistance of programming advices. This work was done as part of Chih Wei Ling's M. Engg thesis work, which was supported by EDB-IPP grant.

A. Datta's work has been supported by Singapore MoE Tier 1 Grant No 2018-T1-002-076.

References

References

- [1] C. Ling, A. Datta, J. Xu, “A Case for Distributed Multilevel Storage Infrastructure for Visual Surveillance in Intelligent Transportation Networks,” in *IEEE Internet Computing*. To Appear.
- [2] Cisco Global Cloud Index: <http://www.dqindia.com/tag/cisco-global-cloud-index/>.
- [3] B. Tian, B. T. Morris, M. Tang, Y. Liu, Y. Yao, C. Gou, D. Shen, and S. Tang, “Hierarchical and networked vehicle surveillance in ITS: a survey,” in *IEEE Transactions on Intelligent Transportation Systems*, 2015.
- [4] M. Valera and S. A. Velastin, “Intelligent distributed surveillance systems: a review vision,” in *IEEE Proceedings on Image and Signal Processing*, 2005.
- [5] R. Nasim, A. Kasser, “Distributed architectures for intelligent transport systems: a survey,” in *Symposium on Network Cloud Computing and Applications*, 2012.
- [6] J. Zhang, F. Wang, K. Wang, W. Lin, X. Xu, C. Chen, “Data-driven intelligent transportation systems: a survey,” in *IEEE Transactions on Intelligent Transportation Systems*, 2011.
- [7] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” in *Proceedings of Mobile Cloud Computing*, 2012.
- [8] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, “Fog computing: a platform for internet of things and analytics,” in *Big Data and Internet of Things: A Roadmap for Smart Environments*, Springer, Studies in Computational Intelligence, 2014.
- [9] A. Das, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, C. Yu, “Transparent and flexible network management for big data processing in the cloud,” in *Proceedings of USENIX Conference Hot Topics Cloud Computing*, 2013.

- [10] A. Arefin, V. K. Singh, G. Jiang, Y. Zhang, and C. Lumezanu, “Diagnosing data center behavior flow by flow,” in Proceedings of IEEE ICDCS, 2013.
- [11] E. Keller, S. Ghorbani, M. Caesar, and J. Rexford, “Live migration of an entire network (and its hosts),” in Proceedings of ACM Workshop Hot Topics Networking, 2012.
- [12] G. Varghese, “Network Algorithmics,” Elsevier, 2005.
- [13] L. R. Ford Jr and D. R. Fulkerson, “Flows in networks,” in Princeton university press, 1962.
- [14] J. Edmonds and R. M. Karp, “Theoretical improvements in algorithmic efficiency for network flow problems,” in Journal of the ACM, 1972.
- [15] S. Paul, J. Pan, and R. Jain, “Architectures for the future networks and the next generation internet: a survey,” in Comp. Commun., 2011.
- [16] C. Labovitz, A. Ahuja, F. Jahanian, “Experimental study of internet stability and wide-area network failures,” in Proceedings of the International Symposium on Fault-Tolerant Computing, 1999.
- [17] C. Labovitz, R. Malan, and F. Jahanian, “Origins of Internet routing instability,” in Proceedings of IEEE INFOCOM, 1999.
- [18] C. Labovitz, A. Ahuja, A. Bose, F. Jahanian, “Delayed internet routing convergence,” in IEEE/ACM Transaction on Networking, 2001.
- [19] A. Elmokashfi, A. Kvalbein, and C. Dovrolis, “BGP churn evolution: a perspective from the core,” in Proceedings of IEEE INFOCOM, 2010.
- [20] S. Sezer, S. Scott-Hayward, P. Chouhan, B. Fraser, D. Lake, J. Foinnegan, N. Viljoen, M. Miller, and N. Rao, “Are we ready for SDN? Implementation challenges for software-defined networks,” in IEEE Communications Magazine, 2013.
- [21] B. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, “A survey of software-defined networking: past, present, and future of programmable networks,” in IEEE Communications Surveys Tutorials, 2014.

- [22] D. Kreutz, F. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: a comprehensive survey,” in *Proceedings of IEEE*, 2015.
- [23] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: enabling innovation in campus networks,” in *SIGCOMM Computer Communication Review*, 2008.
- [24] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, “NOX: towards an operating system for networks,” in *SIGCOMM Computer Communication Review*, 2008.
- [25] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, “Onix: a distributed control platform for large-scale production networks,” in *Operating Systems Design and Implementation, USENIX*, 2010.
- [26] A. Tootoonchian and Y. Ganjali, “HyperFlow: a distributed control plane for OpenFlow,” in *Proceedings of the Internet Network Management Conference on Research on Enterprise Networking, USENIX*, 2010.
- [27] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Holzle, S. Stuart, and A. Vahdat, “B4: experience with a globally-deployed software-defined WAN,” in *Proceedings of ACM SIGCOMM*, 2013.
- [28] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, “Logically centralized? state distribution trade-offs in software defined networks,” in *Proceedings of ACM SIGCOMM HotSDN*, 2012.
- [29] S. Schmid and J. Suomela, “Exploiting locality in distributed SDN control,” in *Proceedings of ACM SIGCOMM HotSDN*, 2013.
- [30] C. Yu et al., “Software-defined latency monitoring in data center networks,” in *Passive and Active Measurement, Springer*, 2015.
- [31] J. Yang, E. Yang, Y. Ran, “SDM²Cast: An OpenFlow-based, software-defined scalable multimedia multicast streaming framework,” in *IEEE Internet Computing*, 2015.

- [32] Mininet: <http://mininet.org/>
- [33] N. Shacham, "Multipoint communication by hierarchically encoded data," in INFOCOM, 1992.
- [34] M. Effros, "Universal multiresolution source codes," in IEEE Transactions on Information Theory, 2001.
- [35] S. Islam and J. Gregoire, "Network edge intelligence for the emerging next-generation internet," in Future Internet, 2010.
- [36] V. Valancius, N. Laoutaris, C. Diot, P. Rodriguez, and L. Massoulie, "Greening the Internet with Nano Data Centers," in Proceedings of ACM CoNEXT, 2009.
- [37] R. Ahlswede, N. Cai, S.-Y. Li, and R. W. Yeung, "Network information flow," in IEEE Transactions on Information Theory, 2000.
- [38] S.-Y. R. Li, R. W. Yeung, and N. Cai, "Linear network coding," in IEEE Transactions on Information Theory, 2003.
- [39] R. Koetter and M. Medard, "An algebraic approach to network coding," in IEEE/ACM Transactions on Networking, 2003.
- [40] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain and L. M. Tolhuizen, "Polynomial time algorithms for multicast network code construction," in IEEE Transactions on Information Theory, 2005.
- [41] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," in Proceedings of the Allerton Conference on Communication Control and Computing, 2003.
- [42] P. A. Chou and Y. Wu, "Network coding for the internet and wireless networks," in IEEE Signal Processing Magazine, 2007.
- [43] M. Langberg, A. Sprintson, and J. Bruck, "The encoding complexity of network coding," in IEEE International Symposium on Information Theory, 2005.
- [44] J. Heide and D. Lucani, "Composite extension finite fields for low overhead network coding: Telescopic codes," in IEEE International Conference on Communications, 2015.

- [45] C. Gkantsidis and P. Rodriguez, “Network coding for large scale content distribution,” in Proc. Infocom, 2005.
- [46] D. S. Lun, M. Medard, T. Ho and R. Koetter, “Network coding with a cost criterion,” in Proceedings of the International Symposium on Information Theory and Its Applications, 2004.
- [47] D. S. Lun, N. Ratnakar, M. Médard, R. Koetter, D. R. Karger, T. Ho, E. Ahmed, and F. Zhao, “Minimum-cost multicast over coded packet networks,” in IEEE Transactions on Information Theory, 2006.
- [48] A. K. Ramasubramanian and J. W. Woods, “Video multicast using network coding,” in Proceedings of SPIE VCIP, 2009.
- [49] M. Wang and B. Li, “How practical is network coding,” In IEEE International Workshop on Quality of Service, 2006.
- [50] J. K. Sundararajan, D. Shahand, M. Medard, S. Jakubczak, M. Mitzenmacher, and J. Barros, “Network coding meets TCP: Theory and implementation,” in Proceedings of IEEE, 2011.
- [51] M. Kim, T. Klein, E. Soljanin, J. Barros, and M. Medard, “Modeling network coded TCP: Analysis of throughput and energy cost,” in Mobile Networks and Applications, 2014.
- [52] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, “Modeling TCP throughput: A simple model and its empirical validation,” in Proceedings of ACM SIGCOMM, 1998.
- [53] S. Chachulski, M. Jennings, S. Katti, and D. Katabi, “Trading structure for randomness in wireless opportunistic routing,” in SIGCOMM, 2007.
- [54] S. Liu and B. Hua, “NCoS: A framework for realizing network coding over software-defined network,” in Local Computer Networks (LCN), IEEE 39th Conference on. IEEE, 2014.
- [55] D Szabó et al., “Network coding as a service,” arXiv.org, 2016.
- [56] M. Pedersen, J. Heide, and F. Fitzek, “Kodo: An open and research oriented network coding library,” in Networking 2011 Workshops, 2011.

- [57] N. Marcano, M. Pedersen, P. Vingelmann, J. Heide, D. Lucani, F. Fitzek, “Getting Kodo: Network coding for the ns-3 simulator,” in Proceedings of the Workshop on ns-3, 2016.
- [58] H. Schwartz, D. Marpe, T. Wiegand, “Overview of the scalable video coding extension of the H.264/AVC standard,” in IEEE Transactions on Circuits and Systems for Video Technology, 2007.
- [59] MPEG-4 Part 2: <https://www.iso.org/standard/39259.html>
- [60] MPEG-4 Part 3: <https://www.iso.org/standard/25035.html>
- [61] F.H.P. Fitzek and M. Reisslein, “MPEG-4 and H.263 video traces for network performance evaluation,” in IEEE Network, 2002.
- [62] Video trace file: <http://www-tkn.ee.tu-berlin.de/research/trace/ltvt.html>
- [63] ns3: <https://www.nsnam.org/>
- [64] BRITE: <http://www.cs.bu.edu/brite/>
- [65] <https://www.cs.bu.edu/brite/publications/usermanual.pdf>
- [66] B. Swenson, and G. Riley, “Simulating large topologies in ns-3 using BRITE and CUDA driven global routing,” in International Conference on Simulation Tools and Techniques, 2013.
- [67] K. Renard, C. Peri, and J. Clarke, “A performance and scalability evaluation of the ns-3 distributed scheduler,” in ICST 2012.
- [68] S. Nikolaev, P. D. Barnes, Jr., J. M. Brase, et al. Performance of distributed ns-3 network simulator. in ICST, 2013.
- [69] https://en.wikipedia.org/wiki/Internet_in_Singapore
- [70] B. Waxman, “Routing of multipoint connections,” in IEEE Journal of Selected Areas in Communications, 1988.
- [71] A.L. Barabasi and R. Albert, “Emergence of scaling in random networks,” in Science, 1999.

Biography

Chih Wei Ling received his B.S. in Mathematics and M. Engg in Computer Science from Nanyang Technological University (Singapore) in 2009 and 2018, respectively. He started his career as a Project Officer under the project entitled “InterCloudRAIDer” at Nanyang Technological University during the year 2011-2014. Since 2014, He has been participated in the Industrial Postgraduate Program, which is a collaboration between Western Digital and Nanyang Technological University. His research interests include coding theory and its application to the complex system. [E-mail: cwling22@gmail.com]

Anwitaman Datta obtained his PhD from EPFL, Lausanne (Switzerland) and he is currently a tenured Associate Professor in the School of Computer Engineering at Nanyang Technological University (NTU) Singapore. Anwitaman’s research interests span the topics of large-scale resilient distributed systems, information security and applications of data analytics. Some of his distinctive and pioneering research contributions in recent years include (i) the application of machine learning and stereotyping to computational trust, (ii) decentralized online social networks for privacy and censorship resistance, (iii) self-repairing (locally repairable) erasure codes for distributed storage systems. He has also made significant contributions to several other well established research topics, including cloud security, data center and NoSQL data stores, collaborative systems, social network analysis and decision support applications such as team recommendation, word sense disambiguation, entity and event identification. [E-mail: anwitaman@ntu.edu.sg]

Jun Xu got his B.S. in Mathematics and Ph.D. in Control, from Southeast University (China) and Nanyang Technological University (Singapore), respectively. He is a Lead Consultant Specialist in HongKong-Shanghai Corporation (HSBC) and was a Principal Engineer in Western Digital. Before that, he was with Data Storage Institute, Nanyang Technological University, and National University of Singapore for research and development. He has multi-discipline knowledge and solid experiences in complex system modeling and simulation, data analytics, data center, cloud storage, and IoT. He has published near 60 international papers and 15 US patents (applications) and 1 monograph. He is an editor of the journal Unmanned Systems and was a committee member of several international conferences. He is a senior member of IEEE and a certificated FRM. [E-mail: jun.xu99@gmail.com]