Content Distribution in VANETs using Network Coding: The Effect of Disk I/O and Processing O/H

Seung-Hoon Lee, Uichin Lee, Kang-Won Lee[†], Mario Gerla
University of California, Los Angeles IBM Thomas J. Watson Research Center[†]
{shlee,uclee,gerla}@cs.ucla.edu, kangwon@us.ibm.com[†]

***Abstract*—Besides safe navigation (e.g., warning of approaching vehicles), car to car communications will enable a host of new applications, ranging from office-on-the-wheel support to entertainment. One of the most promising applications is content distribution among drivers such as multi-media files and software updates. Content distribution in vehicular networks is a challenge due to network dynamics and high mobility, yet network coding was shown to efficiently handle such dynamics and to considerably enhance performance. This paper provides an in-depth analysis of implementation issues of network coding in vehicular networks. To this end, we consider general resource constraints (e.g., CPU, disk, memory) besides bandwidth, that are likely to impact the encoding and storage management operations required by network coding. We develop an abstract model of the network coding procedures and implement it in the wireless network simulator to evaluate the impact of limited resources. We then propose schemes that considerably improve the use of such resources. Our model and extensive simulation results show that network coding parameters must be carefully configured by taking resource constraints into account.**

I. Introduction

Propelled by navigation safety concerns, recently vehicle communications are becoming increasingly popular. Dedicated Short Range Communication (DSRC) is a key enabling technology for vehicular communications [14]. While the main objective has clearly been to improve the overall safety of vehicles, various industry consortiums and academia have been actively seeking for a host of new "killer" vehicle applications, ranging from mobile Internet to entertainment. In fact, the DSRC standard has allocated several "service" channels available for non-safety usage. One of the key applications will be content distribution to vehicles, where the content ranges from multi-media files to road conditions data and to updates/patches of software installed in the vehicle. Content distribution in VANETs is quite close to reality given that most commercial navigation systems can display multi-media data, and entertainment devices in vehicles are becoming popular such as Fiat's Blue and Me[1] and Vizualogic's mobile entertainment integration.[2]

In VANETs, we envision the following content distribution scenario. The original content is uploaded to the Internet server. Vehicles passing by an open AP can opportunistically download the content whenever the connection is available. By open APs we refer to the publicly available APs installed by the service providers, instead of unsecured APs ubiquitously available in urban environments [2]. For instance, a navigation system provider can install APs at the local gas stations for the purpose of map data distribution. P2P technology will help us to overcome the short contact duration with an AP due to high mobility, thus obviating the needs of installing APs every several hundred meters.

Internet-based P2P content distribution where peers with the same interest perform cooperative file swarming by forming an overlay network among peers, cannot be directly applied to wireless mobile ad hoc networks (MANET) since the network topology constantly changes due to mobility. Most file swarming protocols for MANETs have been focused on overcoming the discrepancy between a logical overlay and a physical topology of mobile nodes [15], [8], [20]. For example, ORION [15] builds an on-demand content-based overlay, closely matching the topology of an underlying network. Most protocols rely on flooding, not only to maintain the topology information, but also to distribute the content availability [15], [8], [20]. However, realizing content distribution in VANETs is particularly challenging due to *high mobility*. Since it necessitates more frequent flooding, flooding should be hop-limited for the sake of scalability. As a result, less information is collected, and thus, this makes the peer/piece selection problem, or scheduling problem hard.

Recently, Gkantsidis et al. proposed a content distribution scheme called Avalanche that uses network coding on top of BitTorrent like file sharing in the Internet [12]. Unlike BitTorrent where a file is divided into n pieces and each piece is distributed independently, in Avalanche the original pieces are encoded using a random linear network code at each peer, and coded pieces are exchanged with other peers. The original file can be recovered from any n linearly independent coded pieces. Network coding improves the performance of content distribution by mitigating the scheduling problem given only a *local knowledge* of the network. It helps increase the number of distinct pieces available in the network by generating many

Research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

[1] http://en.wikipedia.org/wiki/Blue&Me

[2] http://www.vizualogic.com

coded pieces, thus providing a higher chance for peers to pull useful pieces [12], [5].

Network coding based file swarming has also been considered in wireless networks. The major departure from P2P file sharing in the Internet where a true multicast via multicast-enabled routers is not supported, is that in wireless networks, nodes naturally communicate using multicast due to the *broadcast* nature of the wireless medium. Thus, network coding enables peers to fully utilize the broadcasting capacity [1]. It has been recently shown that network coding can also effectively handle mobility, interference, and unreliable channel characteristics (e.g., fading), particularly in VANETs [16], [22].

Although the benefit of network coding has been studied in theory [1], [17] and at an abstract level of protocol operations under various circumstances [7], [12], [16], [22], [4], the practical issues of enabling network coding particularly for content distribution have not been well investigated in the literature. In this paper, we consider the resource constraints of nodes, such as CPU consumption, memory access, and disk I/O[3] since network coding induces significant processing overhead at the intermediate nodes. In our scenario, we assume that there are multiple applications running on each "embedded" mobile system (e.g., onboard/safety navigation system), and thus, the file sharing application is allocated with limited resources. At the same time, the demand for resources by the file sharing application may be high because the users may exchange large files such as high quality videos. In conventional content sharing, the most critical resource is the *communication capacity* (i.e., the upload/download bandwidth). When network coding is used, other resource constraints (i.e., CPU, memory, and disk) also play an important role since they are used for encoding new data before sending to others and decoding the data received from other peers. However, standard discrete time network simulators such as ns-2 and QualNet generally allow one to model only network resources and constraints. Thus, the impact of more general resource constraints on network coding in VANETs must be investigated with other means.

To this end, we abstract the overall behavior of the protocol and develop models for computation and disk I/O operations for a given network coding configuration. In this way, we can accurately estimate the latency incurred by network coding procedures at each node. Our computation model clearly reflects the relationship between the computation power and the coding rate. This is a major departure from the previous researches that focused only on reducing the computation overheads of network coding [19], [18] or on showing the feasibility via experiments [11], [25], [18]. Also our disk I/O model takes the storage access patterns into account. This, for example, precisely models the case that all the necessary pieces have to be loaded into the memory before encoding. We validate our model via experiments. The model enables us to analyze the goodput of the overall content pulling procedure and to find the key constraints of determining the network coding configuration in wireless environments.

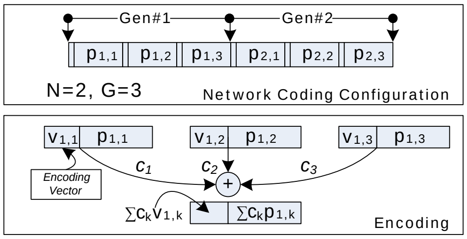


Fig. 1. An illustration of network coding: A file has six B KB pieces and has configured for coding with the number of generations $N = 2$ (i.e., the generation size $G = 3$). When GF(256) is used, the size of an encoding vector is 3 bytes (i.e., G dimension vector). The bottom figure shows how an encoded piece is created from the first generation.

The system modeling modules have been implemented in the content distribution application that we developed in the QualNet simulation environment. Using this "extended" simulation platforms, we investigate methods of improving the performance of network coding. More specifically, (1) we propose a novel "remote buffer aware" data pulling method that minimizes the disk I/O overhead for local computation; and (2) we experiment with recently published computationally efficient network coding methods [18], [19]. We perform extensive simulations to show the impact of overheads and the effectiveness of these enhancements. Our results show that network coding configuration has a great impact on the overall performance, and resource constraints must be carefully considered to achieve a better performance. For given resource constraints, we show that our proposed methods significantly improve the performance. Note that our models are not limited to content distribution scenarios, but they can also be applicable to other network coding based protocols requiring the network coding configuration such as an opportunistic routing protocol with network coding [4] and network coding based message dissemination in delay tolerant networks [26].

The rest of the paper is organized as follows. In Section II, we review the network coding based content distribution in VANETs. In Section III, we formulate the problem of network coding configuration and discuss the importance of general resource constraints on the performance of network coding based protocols. In Section IV, we propose disk I/O and computation overhead models, and analyze the goodput of the overall content pulling procedure. In Section V, we investigate performance enhancement features. In Section VI, we conduct simulations to show the impact of resource constraints and the effectiveness of enhancement features. Finally, we conclude the paper in Section VII.

II. Content Distribution using Network Coding in VANETs

In this section, we review CodeTorrent, a content distribution protocol using network coding in VANETs [16]. This protocol is extended to support content distribution with multi-generation based network coding.

We assume that a file can be uniquely identified with an ID.

[3] In this paper, a disk represents either a mechanical hard disk or a flash-memory based solid state drive.

The original file is divided into N generations. Each generation i has G pieces (which represents the *generation size*) and the *piece size* is fixed to B KB: i.e., $\mathbf{p}_{i,1}$, $\mathbf{p}_{i,2}, \cdots, \mathbf{p}_{i,G}$ for $i = 1, \cdots, N$ (see Figure 1). In network coding based content sharing, intermediate nodes exchange coded pieces instead of original pieces. For the sake of consistency, we assume that each original piece $\ell = 1, \cdots, G$ has ℓth unit vector $\mathbf{e}_\ell$ in the header which is called the *encoding vector*. The original piece ℓ of ith generation is then represented as $\tilde{\mathbf{p}}_{i,\ell} = [\mathbf{e}_\ell \; \mathbf{p}_{i,\ell}]$. For each generation i, the server creates a coded piece via *weighted random linear combination* of all the pieces: $\sum_{k=1}^{G} c_k \tilde{\mathbf{p}}_{i,k}$. Each coefficient c_k is randomly drawn over a finite field, e.g., Galois Field (GF), where the entire operations take place. We use a 8-bit field, GF(256). Each piece contains a unit vector at the source, thus the resulting encoding vector is the same as $[c_1 \cdots c_G]$. Each intermediate node similarly generates a coded piece by combining all the coded pieces collected so far for the generation and only keeps linearly independent coded pieces. If a received piece is linearly independent of other pieces, we call the piece *helpful* or *innovative* and similarly, the originator of the piece is considered *helpful* as well. The total number of linearly independent coded pieces is called *rank*. Note that each coded piece is marked with the *generation number*, and coded pieces belonging to the *same* generation are used for encoding. For a given generation, after collecting G coded pieces that are linearly independent of each other, a node can recover the original data by simply solving a set of linear equations. This process repeats until the node collects all N generations.

Each node periodically broadcasts or *gossips* its resource availability to its 1-hop neighbors. One of the simplest ways of representing the availability is to send an encoding vector of each generation (i.e., as a result of random linear combination of all the encoding vectors of the coded pieces in the buffer). Given this, the receiver can realize whether the originator has at least one linearly independent coded piece. This method is, however, impractical since the size of a gossip message increases with the file size. For instance, with 100 generations each of which contains 100 pieces, the size of a gossip message is $10KB$. To reduce the overhead, we use a bit vector to represent the availability of each generation. If a node requests for a specific generation, the receiver returns its encoding vector. This allows the requester to tell whether the receiver is helpful. If so, the requester starts data pulling without further negotiation. For generation selection, a node uses a *local rarest generation* first policy similar to the *rarest piece* first download policy in BitTorrent: a node chooses the least available generation measured in terms of the number of nodes having the generation (i.e., at least one piece).

We assume that a peer is given a limited buffer (memory) space and the buffer size is smaller than the file size because (1) the demands of applications are ever increasing (e.g., high quality video files) and (2) multiple applications are competing for the limited resource. The system supports *application-controlled file caching* where the kernel allocates physical pages to an application and it manages the pages using its own buffer replacement policy [3]. As shown later, the disk access pattern is per generation basis, and thus, we assume that the buffer replacement unit is a generation. The application replaces the generation that is Least Recently Used (LRU). A small fraction of space is reserved for keeping all the encoding vectors (to check the linear dependency of a request or coded piece) and receiving pieces from others (as receive buffer).

We assume that every transmission is MAC/link layer broadcasting, and a small random amount of wait time before each transmission called broadcast jitter is enforced to reduce collisions. Every node promiscuously listens to packets; i.e., a node receives a specific packet even if it is not the designated receiver, or the requester. If an overheard coded piece is linearly independent of the coded pieces in its local memory, then the node stores it. Our protocol can be configured to pull content at most k-hop neighbors. The resource advertisement is extended to k-hop. For data pulling, we can either use existing routing protocols (e.g., AODV, OLSR, etc.) or implement a customized routing protocol at the application layer as in ORION [15] where k-hop limited controlled flooding of resource availability can be used as a route discovery request (e.g., RREQ in AODV) and a data pull request as a *route reply* (e.g., RREP in AODV).

III. Problem Definition and Related Work

The benefits of network coding based content distribution in VANETs can be attributed to the following reasons: (1) network coding exploits the *broadcast* nature of wireless medium; (2) network coding mitigates the peer and piece selection problem [5], which is extremely difficult to address in dynamic VANETs; and (3) it has been recently shown that network coding in VANETs can effectively handle the random losses due to mobility and interference [22], [16]. One of the most important performance factors in network coding based content distribution is the “generation size” (i.e., the number of pieces per generation). Real time applications such as P2P streaming have a delay constraint because an entire generation must be received before data can be played out [22], [7]. Thus, the generation size must be small enough to comply with such a constraint. In contrast, content distribution does not have a strict delay constraint. Since its goal is to download the entire file as early as possible, without specific constraints on download rate, we can have a large generation size.

Given this, we show that one must reduce the number of generations (i.e., increase the generation size) to improve performance. Assuming that the bandwidth is equally shared by M neighboring nodes, a node can only use the channel $1/M$ fraction of the overall time by sending a request for a piece in a generation that has not been completely downloaded. As the average number of neighboring nodes increases, a node will spend more time overhearing the channel than requesting pieces of its own. Given that a piece size is fixed to B and a file has total N pieces, we can imagine two extreme scenarios: single generation and N generation (no coding) scenarios. In the single generation scenario, an overheard piece is useful if it is linearly independent of other pieces. On the other hand, in the N generation scenario the probability that an overheard packet is useful depends on the number of generations that a node has collected thus far. When a node has collected k generations, the probability is given as $1 - k/N$. The probability is getting smaller as we collect more generations; thus, the coupon collection problem will happen. Given that an overheard piece is useful with high probability [10], the single generation scenario will take $\Theta(N)$ steps to complete downloading. In contrast, the N generation scenario will take $\Theta(N \log N)$ steps due to coupon collection. For a given configuration with an arbitrary number of generations, the number of steps to complete ranges from $\Theta(N)$ to $\Theta(N \log N)$. As the number of generations decreases, the number of steps is getting closer to $\Theta(N)$. Thus, for better performance, we should decrease the number of generations. With this in mind, we now investigate practical issues that arise when distributing a large file using network coding; namely we consider *communication*, *computation*, and *disk I/O* overheads.

Communication overhead: It is an ideal scenario of network coding in a wireless network when the size of a piece is the same as the size of a packet since a packet loss (due to collision or channel errors) can be effectively masked via network coding. However, packet-level network coding becomes less efficient as the file size increases because it increases communication overhead. Recall that each packet must contain a global encoding vector. For instance, when distributing 100KB and 1000KB files, we generate 100 and 1000 1-KB blocks respectively. Assuming that GF (256) is used (i.e., 8bit), the overhead is 100B (≈10%) and 1000B (≈100%). Thus, we need to create smaller size generations to limit the overhead ratio. In this case, however, packet-level network coding (i.e., recovery from packet loss) will have many small size generations, thus causing the notorious coupon collection problem.[4] To mitigate this problem, the size of an individual piece should scale proportionally to the file size. In particular, the piece size must be carefully chosen based on the link duration statistics [23].

Computation overhead: Random linear network coding heavily relies on finite field operations. The computation overhead is roughly proportional to the number of pieces per generation, or generation size. Thus, using a small number of large generations to avoid the coupon collection problem may result in severe computational overhead such that encoding takes more time than transmission, and one's communication bandwidth is underutilized.

Disk I/O overhead: Since the main memory will be shared by a number of vehicular applications, the memory space dedicated to P2P applications may typically be limited compared to the size of a large multimedia file. For network coding, it may be necessary to read all the pieces belonging to the same generation from the storage device to generate a coded piece. If the memory is full, some pieces may have to be evicted to make room for the requested generation. The delay incurred for disk I/O is huge compared to memory access, and is especially significant in VANETs because vehicles have only short contact duration in general. For example, given a 250m wireless communication range, vehicles driving in opposite lanes with 50mph have only 11 seconds to communicate with each other. Let us assume that the size of a generation is 40MB. The nominal data transfer rate of hard disks or flash memory based solid state disks is about 40MB/s. If a miss happens (i.e., the requested generation is not in the memory), it will take one second to make the application ready for encoding, thus resulting in almost 10% performance loss. Therefore, it is important to design the file swarming algorithm so that disk access is minimized whenever possible.

Recent feasibility studies on network coding in real testbeds [11], [18], [25] show that the measured performance varies widely depending on the system characteristics, but the fundamental reason of performance variation is not well understood. For instance, Gkantsidis et al. [11] show that Avalanche, an Internet-based P2P file sharing with network coding, incurs little overhead in terms of CPU and I/O activity using their large scale testbed, whereas it is empirically observed that that computation overhead degrades the performance, especially when the generation size is large [25], [18]. Various performance enhancement techniques have been proposed [9], [18], [19], [24]. Cooper et al. propose the sparse network coding where each piece is selected for coding with a certain probability, thus reducing the number of pieces involved in coding [9], [18]. Maymounkov et al. show that one can decrease the generation size, yet can still effectively handle the coupon collection problem by using erasure coding at the generation level [19]. Shojania et al. [24] use CPU acceleration techniques to improve the performance of Galois field operations. However, it is still not clear how to find a proper network coding configuration, and how such improvement techniques influence the performance of network coding based content distribution in VANETs.

In this paper, we propose simple models for CPU and disk I/O overheads of network coding. The models enable us to analyze the goodput of the content pulling procedure and to find the key constraints of determining the network coding configuration for content distribution. Also, we propose novel algorithms to improve the performance of network coding for distribution of large contents in wireless environments. We implement the models and algorithms in a wireless network simulator and extensively evaluate the impact of network coding configuration.

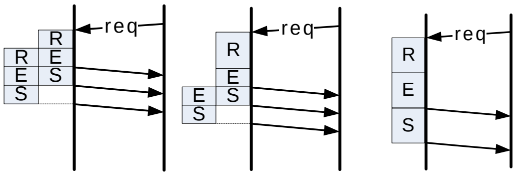
IV. Disk I/O and Computation O/H Models

In this section, we present the request processing procedure of a serving peer. We then model both disk I/O and computation O/H and analyze the goodput of the procedure. We perform experiments to measure the model parameters.

A. Request service procedure

If a node can serve a request, it first checks its buffer. If the node has the data of the requested generation in the buffer, it can start an encoding process. Otherwise, the node

[4] Note that random errors can also be effectively handled using a hybrid scheme using ARQ and erasure coding [6]



(a) R/E/S pipeline (b) E/S pipeline (c) No pipeline

Fig. 2. Possible parallelism scenarios with piece size $B = 2$KB

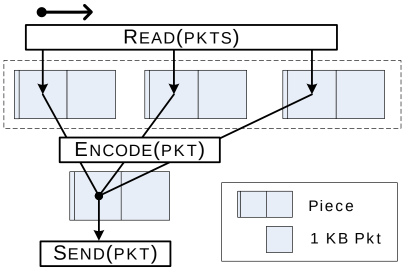


Fig. 3. Overall procedure example: $G = 3$ (generation size), $B = 2$KB (piece size). A coded piece is composed of 2 *independent* 1-KB coded packet. Each piece has a header composed of an encoding vector, generation number, etc.

must first read the generation from the disk before encoding. After the data has been properly encoded, the node sends the resulting coded piece to the requester. The overall procedure is composed of reading a generation (R), encoding the data (E), and sending the coded piece (S). Note that access to memory by disks and network interface cards are typically done via Direct Memory Access (DMA); therefore, in practice, we can ignore the interference between them. Thus we can exploit thread-level parallelism to speed up the overall process. Figure 2 shows possible scenarios of parallelism.

In Figure 3, we consider an example with R/E/S pipeline when the generation size $G = 3$ and the piece size $B = 2$KB. To generate a coded symbol, only 3 symbols (one from each piece) are involved; and rest of the symbols are independent of each other. Assuming that the unit of data transfer is 1KB, the communication thread sends the newly encoded packet as soon as it is ready. The server first checks its buffer to see whether a requested generation is present in the *working set*. If so, the encoding thread starts an encoding process (ENCODE); otherwise, the disk I/O thread reads the necessary parts of the generation from the disk (READ), then signals the encoding thread. After the encoding is finished, a communication thread sends the newly generated encoded packet out to the requesting peer (SEND). In E/S pipeline, i.e., Figure 2(b), all the pieces for a given generation are read at once and then only E/S are pipelined. In the case of no pipeline, i.e., Figure 2(c), all operations take place sequentially. Note that although we assume that a unit of data transfer is 1KB, but to minimize the overhead of system calls (or context switching time) we can have a larger transfer unit.

B. Overhead models

We present disk I/O and computation O/H models. We find the goodput of the request handling procedure.

1) Disk I/O overhead model: Disk access involves mechanical motions and is inherently slow by an order of magnitude compared to reading data from memory. Disk access delay consists of three factors: seek time, rotational latency, and transfer time. Seek time is the time to move disk heads to the disk cylinder to be accessed. Rotational latency is the time to get to a specific disk block in a cylinder. Transfer time is the time to actually read disk blocks. The total average latency for modern hard disks is in the range of 10-15msec and it varies from vendor to vendor. Disks are typically optimized for sequential access, and they can transfer large data files at an aggregate of 40MB/s (for desktop-grade disks) or 80MB/s (for enterprise server level disks). Recently, flash-based solid state disks (SSDs) are becoming popular. The main difference is that SSDs have much lower seek time and no rotational latency compared to the conventional disks. The transfer rate is still about the same as conventional disks. For instance, Transcend TS32GSSD25-M has 0.1ms of seek time and the read/write rates are 40MB/s and 32MB/s respectively.

Assuming that each generation is stored sequentially, we can safely ignore the rotation latency of disks. Thus, the analysis of mechanical disks and SSDs are the same. To generate a 1-KB coded packet, we need to read all the corresponding 1-KB data per piece as in Figure 3. The access pattern will be a sequence of seek/read pairs. Let θ denote the average latency to perform a pair of seek/read operation. The overall time to read all the relevant parts takes $T_d = \theta \cdot G$. Note that the seek latency is quite prohibitive in the case of mechanical disks compared to SSDs such that the overall latency is quite large, as the number of generations increases. As an alternative, a node can sequentially read the entire pieces at once (as in E/S pipeline). The disk I/O latency is given as $\frac{GB}{R_d}$ where B is the piece size and R_d is the data transfer rate.

We investigate the impact of the pair-wise access patterns (seek/read pairs) by measuring θ in real systems. We use two sets of scenarios: (1) Maxtor MaxLine Pro SATA-II HDD (500GB, 7200rpm, 8MB cache) with PERC 5i RAID (level 0); and (2) Transcend TS32GSSD25-M solid state drive (32GB). The measured maximum sequential data access rate of a disk and a SSD is given as 110MB/s and 38MB/s respectively. We first measure the pair-wise access latency (θ). The measured latency is given as θ=0.495ms (std. dev. 1.462ms) and θ=0.012ms (std. dev. 0.05ms) for a disk and a SSD respectively. Given generation size of G, the total latency is $\theta \cdot G$. For instance, given the generation size of 100, the overall latency of a disk and a SSD is given as 49.5ms and 1.2ms respectively.

2) Computation overhead model: Let b'_k denote the kth code symbol in a coded piece, and $b_{i,k}$ denote the kth symbol of the ith piece in the buffer. Let c_i for $i = 1, \cdots, G$ denote the ith encoding coefficient, which is randomly chosen over a Galois Field of size 256 once at the beginning of the entire procedure (i.e., symbol size is 8bit). Each code symbol b'_k is generated as follows:

$$b'_k = c_1 \cdot b_{1,k} + c_2 \cdot b_{2,k} + \cdots + c_G \cdot b_{G,k}$$

For each symbol ($b_{i,k}$) it requires a pair of multiplication (i.e., $c_i \cdot b_{i,k}$) and addition (b'_k += $c_i \cdot b_{i,k}$). The per-symbol encoding

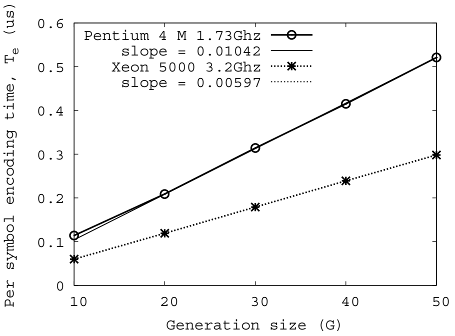


Fig. 4. Per symbol coding latency as a function of generation size G

time is proportional to the generation size G, i.e., $T_e = G \cdot \delta$ where δ is the time of executing the pair of operations. Let R_e denote the per-symbol encoding rate (byte/sec). Then, the rate is given as follows:

$$R_e = \frac{1}{T_e} = \frac{1}{\delta} \cdot \frac{1}{G} \qquad (1)$$

Equation 1 shows that the encoding rate is the function of δ and G. The value δ is purely dependent on the Galois field operation implementation and the processing power.

We measure the δ value in two different systems: a fast machine (Intel Xeon Dual Core 5000 3.2GHz), representing a high speed server and a slow machine (Intel Pentium 4 M 1.73GHz), representing a relatively powerful mobile device. We implement the Galois field operations based on a table lookup with the optimization techniques proposed in [13].[5] We ignore the cache miss since the lookup table fits in the internal cache and the memory access pattern of coding is sequential. We use a Galois field of size 256. We have compiled this code so that it is optimized for execution. We use a 12MB file to measure the value. We increase the generation size G from 10 to 50 with a gap of 10 blocks. We report the average of 1000 runs for each configuration. Per symbol encoding latency is reported in Figure 4. The figure shows that the encoding latency increases linearly as shown in Equation 1. In fact, the plots fit well with lines with slope δ = 5.97ns and δ = 10.42ns for Xeon and P4 respectively. Thus, the encoding rate equations are given as $\frac{166.9}{G}$MB/s and $\frac{95.9}{G}$MB/s for fast and slow where G denotes the generation size. For a small generation size, e.g. $G = 10$, the fast machine could generate code packets at the rate of 16.7MB/s whereas the slow machine generates them at the rate of 9.6MB/s respectively. For a relative large generation size, e.g. $n = 100$, these rates drop to 1.67MB/s and 960KB/s for the fast and the slow machines respectively. In the latter case, the computation overhead becomes the bottleneck compared to the network bandwidth (e.g., 11Mbps 802.11b vs. 7.68Mbps encoding rate).

C. Goodput analysis

In wireless networks, the bandwidth is shared by multiple nodes. Given M nodes are sharing the bandwidth, we assume that the bandwidth is fairly shared by M nodes. Let R_b denote the bandwidth share. In the following, we show that the goodput is mainly determined by the bandwidth share R_b and the encoding rate R_e. From the analysis, we show that for given resource constraints, we can find the maximum allowable generation size.

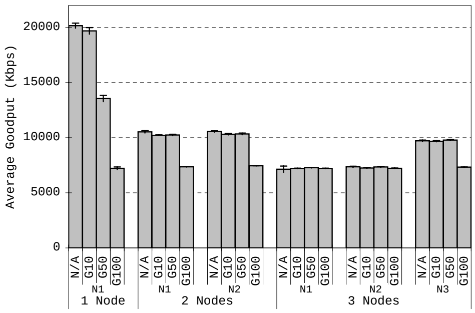


Fig. 5. Goodput with different generation sizes and interfering nodes. The baseline goodput without network coding is denoted as “N/A”

Let us find the goodput of E/S pipeline. Assume that there are total N_r requests of a specific generation. Recall that G is the generation size, B is the piece size, R_e is encoding rate, and R_d is data transfer rate. When we have $R_e \geq R_b$, the total amount of time to transfer N_r pieces is $GB/R_d + B/R_e + N_rB/R_b$. The goodput is given as

$$\frac{N_r}{G/R_d + 1/R_e + N_r/R_b}$$

For large N_r, the goodput can be approximated to the effective bandwidth R_b. When we have $R_e < R_b$, the total amount of time is $GB/R_d + N_rB/R_e + B/R_b$. The goodput is given as

$$\frac{N_r}{G/R_d + N_r/R_e + 1/R_b}$$

For large N_r, the goodput is approximated to the encoding rate R_e. The key constraint is that the encoding rate R_e should be greater than the bandwidth share R_b, i.e., $R_e \geq R_b$. By replacing R_e with $\frac{1}{\delta G}$, we have $G \leq \frac{1}{\delta R_b}$. Thus, this inequality enables us to find the maximum generation size that satisfies the condition ($R_e \geq R_b$). The equations also show that the effect of disk I/O disappears, as the number of requests per generation increases. In the following section, we propose a simple technique to increase the number of requests per generation. Note that the goodput of R/E/S pipeline is approximately the same as E/S pipeline when the number of requests per generation is large.

We now show the impact of the bandwidth share on the performance of a network coding configuration via experiments. Since the bandwidth share is mainly determined by the total number of nodes sharing the bandwidth (within their radio range), we vary the number of nodes (N_S=1-3) and measure the goodput of network coding with different generation sizes (G=10, 50, 100). We setup a server that receives all the blocks generated by other nodes. For each experiment, a client node continues to generate/send coded blocks to the server until it transfers 60MB of data. We run each configuration 30 times and report the average with the 95% confidence interval. Data transfers of clients are initiated by the server via parallel SSH. We perform the experiment in the early morning (2-6AM) to

[5] Shojania et al. showed that the Galois field operations can be further improved by using hardware acceleration techniques such as SSE2 and AltiVec SIMD vector instructions on x86 and PowerPC processors respectively [24].

exclude other WiFi interferences (e.g., wireless LAN). We use IBM Thinkpad R52 (Intel Pentium 4 M 1.73GHz, 512MB). All laptops have Fedora Core 5 with Linux Kernel v2.6.19. We use ORiNOCO 11b/g PC Cards (8471-WD) and the MadWifi v0.9.3.3 Linux Kernel device driver for the Atheros chipset to support wireless networking in Linux. We configure 802.11g as follows: ad hoc mode, no RTS threshold, and 54Mbps (fixed).

The measured goodput is reported in Figure 5. The figure clearly shows that if the generation size is too large (N_S=1, G=50/100), a node cannot fully utilize its bandwidth. The figure also shows that as the number of nodes increases, per node bandwidth share decreases accordingly. Interestingly, this allows a node to sustain a larger generation size; e.g., a node can support G=50 in the two node scenario and G=100 in the three node scenario. The measured goodput is quite close to the estimated coding rate based on our model. For instance, the measured goodput of G=100 is 7.2Mbps that is comparable to our model estimate of 7.68Mbps (960KB/s).

V. Performance Enhancement Features

In this section, we propose novel mechanisms to mitigate disk I/O and computation overhead for network coding for content distribution. We first present a remote buffer generation aware pulling technique to reduce disk access frequency. We then review the techniques that to reduce computation overheads.

A. Remote Buffer Generation Aware Pulling

A node uses a *rarest generation first* strategy where it chooses the least available generation measured in terms of the number of nodes having each generation. If the requested generation is in the buffer, it can start generating a coded piece; otherwise, the node has to read it from the disk. Many different nodes could send requests, each of which is likely to ask for a different generation because the topology keeps changing due to high mobility. The problem is that these requests are competing for the limited buffer space which may result in severe disk I/O. Given the fact that the overhead is proportional to the generation size, to circumvent this situation the serving peer should have enough buffer space to handle all requests (i.e., the buffer size should be larger than the *working set* size): i.e., $N_R \times G < S_b$ where N_R is the expected number of distinct generations requested, G is the generation size, and S_b is the buffer size. The relationship shows that the generation size should be limited to a certain threshold to avoid disk I/O.

We propose a *Remote Buffer Generation Aware Pulling* method where a requester considers the buffer status of a remote node (i.e., which generations are present in the buffer). The scheme mitigates the disk I/O by reducing the expected number of independent requests (a set of different generations). To realize this, given N generations we represent the buffer status of a node using an N-bit vector. The buffer status of a node can be included in a periodic "gossip" message so that other peers can learn about it. Using this buffer status information of a remote node, a node can search for the generation with the lowest rank among the generations that are in the remote nodes' buffers. If none of the generations that the node interested in pulling is in presence, the node simply sends a request of the rarest generation, thus causing a disk access.

B. Fast Network Coding

Since the computation overhead is proportional to the generation size one can reduce the overhead by decreasing the number of pieces used for coding. Sparse random linear coding [9] has been proposed to achieve this: each piece is selected with probability $p \geq (\log G + d)/G$ where G is the generation size and d is a non-negative constant [9], [18]. This probabilistic approach, however, does not consider computation capacity of a node, which can be measured by the maximum number of pieces that can be encoded without degrading the performance (denoted by γ). Since the number of pieces used for coding follows a binomial distribution, the average number of pieces used for coding is Gp, which is proportional to the generation size. Even with this, if the generation size is too large, there is a chance that the number of pieces may be greater than γ. To deal with this problem, we approximate the behavior of this probabilistic scheme by equating γ with the mean of the distribution. As a result, we have the following condition: $\gamma \geq \log_2 G + d$. This means that one has to control the generation size based on this condition, i.e., if G is too large, we need to create more generations. One caveat is that data dissemination occurs in a distributed fashion and the high mobility in VANETs creates cycles of dissemination, and thus it is hard to guarantee that encoded pieces from different peers are linearly independent [16], [18].

Note that on the one hand, the computation overhead can be reduced by decreasing the generation size. On the other hand, this will result in a large number of generations, leading to the coupon collection problem. Maymounkov et al. propose Chunked Codes where they keep the generation size small to make the network coding computationally efficient, and yet use erasure coding at the generation level to circumvent the coupon collection problem. However, this will not fully utilize the benefit of broadcasting in wireless networks, because the effectiveness of broadcasting decreases, as the number of generation increases. In the following section, we show this via extensive simulations.

VI. Evaluation

In this section, we first describe the implementation details of the protocols that we consider for evaluation, and simulation setup in QualNet.[6]. The impact of disk I/O and computation overhead is presented and then performance enhancement features are evaluated.

A. Simulation setup

We use IEEE 802.11b PHY/MAC with 11Mbps data rate and Real-Track (RT) mobility model [21]. RT permits to model vehicle mobility in an urban environment more realistically than other simpler and more widely used mobility models such

[6] Scalable Networks, http://www.scalable-networks.com

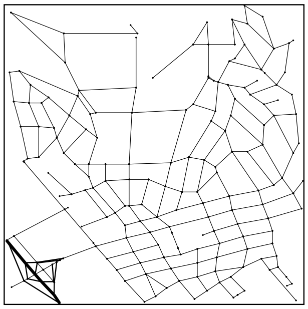


Fig. 6. Westwood area

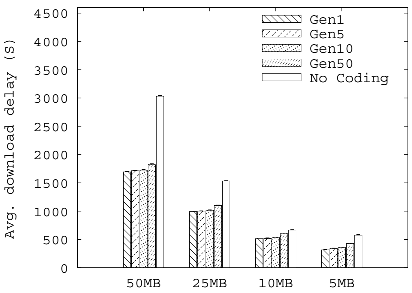


Fig. 7. Download delay without O/H

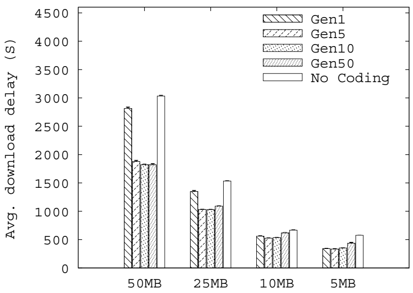


Fig. 8. Download delay with O/H: Buffer 100%

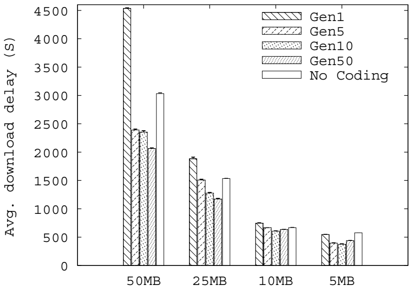


Fig. 9. Download delay with O/H: Buffer 50%

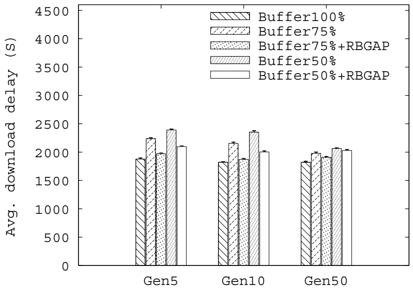


Fig. 10. Download delay with RBGAP (50MB)

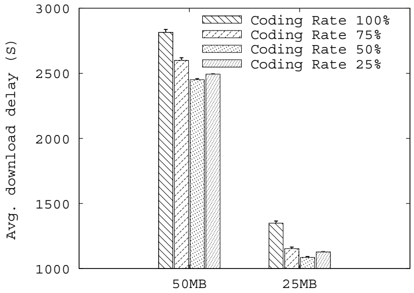


Fig. 11. Impact of sparse coding

as Random Waypoint (RWP), by restricting the movement of the nodes. The road map input to the RT model is shown in Figure 6, a street map of 2,400$m\times$ 2,400m $Westwood$ area in the vicinity of the UCLA campus. A fraction of nodes (denoted as popularity) in the network are interested in downloading the same file. In the simulations, 200 nodes are populated, and 40% of the nodes are interested in downloading the file (i.e., total 80 nodes). The speeds of nodes are randomly selected from [0,20]m/sec. There is a special type of node called an Access Point (AP) which possesses the complete file at the beginning of the simulation. Three static APs are randomly positioned on the roadside in the area. To evaluate the impact of file size, we use four different sizes of files, namely 5MB, 10MB, 25MB and 50MB. Although the file size is relatively small compared to multimedia files, which are in the order of 100MBs, we believe that it is large enough to evaluate the performance of various schemes. The piece size is set to 20KB. For the buffer replacement scheme, Least Recently Used (LRU) is used to evict an entire generation when the buffer is full. Buffer space size is represented using the ratio of the memory buffer size to the file size. A gossip message is sent to 1-hop neighbors in every 2 seconds. The single hop pulling strategy is used to measure the performance of content distribution by excluding routing overheads.

We use the following H/W parameters to model disk I/O and computation overheads: a nominal hard disk of R_d=40MB/s, and a mobile device CPU of R_e=$48/n$MB/s (50% computing power of Intel Pentium 4 M 1.7Ghz). We implement the E/S pipeline scheme for multi-threading (see Figure 2(b)): a missing generation is fully loaded into the buffer and then encoding (E) and sending (S) processes are pipelined. The disk and coding delays are scheduled based on the disk I/O and computation models respectively. We define the "download delay" as the elapsed time for a node to finish downloading a file. For each configuration, we report the average value of 30 runs with 95% confidence interval.

B. Simulation Results

Effects of Disk I/O and Computation O/H: We consider scenarios with various numbers of generations: N=1, 5, 10, 50 and *No Coding*. Here, *No Coding* denotes the case where network coding is not used (i.e., the generation size is 1). To show the impact of overheads, we present the ideal case where none of the overheads is not considered. We also vary the availability of buffer space: 50% and 100%. Note that we can see the impact of "computational overhead" in the case of 100% buffer space, because a node can keep the entire file in the memory.

Figure 7 shows the results of the ideal case. The figure shows that as the number of generations increases, the download delay also increases. This confirms that the number of generations must be kept as small as possible to achieve a good performance. In Figure 8, we show the case of buffer size = 100% to show the impact of computation overhead. Unlike previous results, we notice that the single generation scenarios perform worse than other scenarios, especially when the file size is large (i.e., 50M and 25M). Yet it is still better than the *No Coding* scenario where the generation size is 2500 for a 50MB file and 1250 for a 25MB file, and the corresponding encoding rates are 19.2KB/s and 38.4KB/s respectively. This clearly shows that the encoding rate is a bottleneck. As the number of generations increases, the effect of computation overhead reduces. However, if the number of generations is above a certain threshold, the download latency begins to increase. The figure shows a "U" shape delay curve for both 25MB and 50MB files. For example, consider the plots of a 25MB file case; the delay decreases until $N = 10$, and it increases thereafter.

Now consider the cases where the buffer size is smaller than the file size (see Figure 9). The impacts of disk I/O can be clearly seen by juxtaposing Fig. 9 with Figure 8. Contrary to our common belief that network coding improves the file swarming performance [16], the download delay is even worse than the conventional file swarming (i.e., the *No Coding* scenario). The larger the generation size, the higher the cost of loading a generation into the buffer; thus, the impact of overheads decreases as the number of generations increases.

Remote Buffer Generation-Aware Pulling (RBGAP): Figure 10 shows the download delay with different buffer sizes, namely 100%, 75% and 50%. The download delay, as the number of generations increases. Note that the disk I/O overhead is proportional to the number of pieces per generation, and the probability that the requested generation is not in the buffer is mainly determined by the buffer size. Thus, the impact of finite buffer decreases with the number of generations. The figure shows that RBGAP can effectively reduce unnecessary disk I/O overheads, thus reducing the downloading delay.

Sparse Coding: To show the effectiveness of a sparse random network coding, we vary the coding density (i.e., the fraction of the number of pieces used for encoding) with 25% increments. For instance, 25% and 50% coding density on a 50MB file with N=1 show that the maximum number of pieces used for encoding is 625 and 1250 out of total 2500 pieces respectively. We simulate the following cases: a 50MB file with $N = 1$ (G=2500) and a 25MB file with $N = 1$ (G=1250). We use the buffer size of 100% (i.e., no buffer replacement overhead) to clearly see the benefits of sparse coding. Figure 11 shows the results. As the coding density decreases, the download delay decreases. For instance, when we lower the coding density to 75%, we observe a considerable delay reduction: from 2814s to 2599s for a 50MB file and from 1349s to 1153s for a 25MB file. However, if the coding density is too low, it is likely that a linearly dependent coded piece is generated. As a result, a node may not be able to fully utilize its bandwidth and thus, the download delay increases.

VII. Conclusion

In this paper, we investigated the impact of resource constraints (i.e., disk I/O, computation, memory) on the performance of network coding based content distribution. We modeled the disk I/O and computation operations in network coding to consider the resource constraints. We analyzed the goodput of the content pulling procedure and found that one of the key constraints for network coding configuration is the available communication bandwidth for a node. We implemented the model in a wireless network simulator and extensively evaluated the impact of network coding configuration. To improve performance, we proposed a remote buffer aware pulling method that minimizes the disk I/O overhead and investigated computationally efficient network coding methods. We found that (1) resource constraints have a significant impact on the performance; (2) the proposed remote buffer aware pulling effectively reduced the disk I/O overheads; (3) the coding rate of sparse random network coding has to be carefully chosen based on the given bandwidth; (4) generation level pre-coding does not work well in VANETs.

References

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network Information Flow. *IEEE Transactions on Information Theory*, 46(4):1204–16, Jul. 2000.

[2] V. Bychkovsky, B. Hull, A. K. Miu, H. Balakrishnan, and S. Madden. A Measurement Study of Vehicular Internet Access Using In Situ Wi-Fi Networks. In *Mobicom'06*, Los Angeles, CA, Sep. 2006.

[3] P. Cao, E. W. Felten, and K. Li. Application-Controlled File Caching Policies. In *USENIX'94*, Boston, Massachusetts, Jun. 1994.

[4] S. Chachulski, M. Jennings, S. Katti, and D. Katabi. Trading Structure for Randomness in Wireless Opportunistic Routing. In *SIGCOMM'07*, Kyoto, Japan, Aug. 2007.

[5] D. M. Chiu, R. W. Yeung, J. Huang, and B. Fan. Can Network Coding Help in P2P Networks? In *NetCod'06*, Boston, MA, Apr. 2006.

[6] S. Choi and K. Shin. A Class of Adaptive Hybrid ARQ Schemes for Wireless Links. *IEEE Transactions on Vehicular Technology*, 50(3):777–790, May 2001.

[7] P. A. Chou, Y. Wu, and K. Jain. Practical Network Coding. In *Allerton'03*, Monticello, IL, Oct. 2005.

[8] M. Conti, E. Gregori, and G. Turi. A Cross-Layer Optimization of Gnutella for Mobile Ad hoc Networks. In *MobiHoc'05*, Illinois, USA, May 2005.

[9] C. Cooper. On the Distribution of Rank of a Random Matrix over a Finite Field. *Random Struct. Algorithms*, 17(3-4):197–221, 2000.

[10] S. Deb, M. Médard, and C. Chout. Algebraic Gossip: A Network Coding Approach to Optimal Multiple Rumor Mongering. In *Allerton'04*, Allerton, IL, Sep. 2004.

[11] C. Gkantsidis, J. Miller, and P. Rodriguez. Comprehensive View of a Live Network Coding P2P System. In *IMC'06*, Brasil, Oct. 2006.

[12] C. Gkantsidis and P. Rodriguez. Network Coding for Large Scale Content Distribution. In *INFOCOM'05*, Miami, FL, USA, Mar. 2005.

[13] C. Huang and L. Xu. Fast Software Implementations of Finite Field Operations. Technical report, Washington University in St. Louis, Dec. 2003.

[14] D. Jiang, V. Taliwal1, A. Meier, W. Holfelder, and R. Herrtwich. Design of 5.9GHz DSRC-based Vehicular Safety Communication. *IEEE Wireless Communications*, 13(5), Oct. 2006.

[15] A. Klemm, C. Lindemann, and O. P. Waldhorst. A Special-Purpose Peer-to-Peer File Sharing System for Mobile Ad Hoc Networks . In *VTC'03*, Orlando, FL, Oct. 2003.

[16] U. Lee, J.-S. Park, J. Yeh, G. Pau, and M. Gerla. CodeTorrent: Content Distribution using Network Coding in VANETs. In *MobiShare'06*, Los Angeles, CA, Sep. 2006.

[17] J. Liu, D. Goeckel, and D. Towsley. Bounds on the Gain of Network Coding and Broadcasting in Wireless Networks. In *INFOCOM'07*, Anchorage, AK, May 2007.

[18] G. Ma, Y. Xu, M. Lin, and Y. Xuan. A Content Distribution System based on Sparse Linear Network Coding. In *NetCod'07*, Miami, FL, USA, Mar. 2007.

[19] P. Maymounkov, N. J. A. Harvey, and D. S. Lun. Methods for Efficient Network Coding. In *Allerton'06*, Monticello, IL, Sep. 2006.

[20] A. Nandan, S. Das, M. Y. Sanadidi, and M. Gerla. Cooperative Downloading in Vehicular Ad Hoc Wireless Networks. In *WONS'05*, St. Moritz, SWITZERLAND, Jan. 2005.

[21] A. Nandan, S. Das, S. Tewari, M. Gerla, and L. Klienrock. AdTorrent: Delivering Location Cognizant Advertisements to Car Networks. In *WONS'06*, Les Menuires, France, Jan. 2006.

[22] J.-S. Park, M. Gerla, D. S. Lun, Y. Yi, and M. Médard. CodeCast: a Network-Coding-Based Ad Hoc Multicast Protocol. *IEEE Wireless Communications*, 13(5), Oct. 2006.

[23] P. Samar and S. B. Wicker. On the Behavior of Communication Links of a Node in a Multi-hop Mobile Environment. In *MobiHoc'04*, Tokyo, Japan, May 2004.

[24] H. Shojania and B. Li. Parallelized Progressive Network Coding with Hardware Acceleration. In *IWQoS'07*, Chicago, Illinois, Sep. 2007.

[25] M. Wang and B. Li. How Practical is Network Coding? In *IWQoS'06*, New Haven, CT, Jun. 2006.

[26] J. Widmer and J.-Y. L. Boudec. Network Coding for Efficient Communication in Extreme Networks. In *CHANTS'05*, Philadelphia, Pennsylvania, Aug. 2005.