Understanding Processing Overheads of Network Coding Based Content Distribution in VANETs

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

***Abstract*—Content distribution in vehicular networks, such as multimedia file sharing and software updates, poses a great challenge due to network dynamics and high speed mobility. In recent years, network coding has been shown to efficiently support distribution of content in such dynamic environments, thereby considerably enhancing the performance. However, the related work in the literature has mostly focused on theoretic or algorithmic aspects of network coding so far. In this paper, we provide an in-depth analysis on the implementation issues of network coding in wireless networks. In particular, we study the impact of resource constraints (namely CPU, disk, memory, and bandwidth) on the performance of network coding in the content distribution application. The contribution of this paper is two-fold. First, we develop an abstract model of a general network coding process and evaluate the validity of the model via several experiments on real systems. This model enables us to find the key resource constraints that influence the network coding strategy and thus to efficiently configure network coding parameters in wireless networks. Second, we propose schemes that considerably improve the performance of network coding under resource constrained environments. We implement our overhead model in the QualNet network simulator and evaluate these schemes in a large scale vehicular network. Our results show that the proposed schemes can significantly improve the network coding performance by reducing the coding overhead.**

***Index Terms*—Network coding, Content distribution, Coding overhead analysis, VANETs**

I. Introduction

Inter-vehicular communication has received a lot of attention recently due to safety concerns for drivers. Although the main focus of inter-vehicular communication has clearly been to improve the safety on the road, both industry and academia have also been seeking novel applications, ranging from mobile Internet to entertainment. In fact, the DSRC standard, a key enabling technology of inter-vehicular communications, has allocated several of its "service" channels specifically to non-safety usage [17].

One of the key applications will be content distribution among vehicles. We envision the distribution of shared multimedia files to deliver road/traffic conditions, to patch software installed in the vehicle (such as onboard satellite-navigation systems), to advertise local establishments, etc. If the content originates from an Internet server, vehicles passing by an open Access Point (AP) can opportunistically download it whenever they can establish a connection. Peer-to-Peer (P2P) technologies can further help disseminate the content, overcoming the constraint of the very short AP-vehicle contact time at highway speeds. Internet P2P content distribution schemes, however, cannot be directly applied to mobile ad hoc networks (MANETs) because of rapid changing network topology, and this has been a challenging issue of designing efficient MANET file swarming protocols [20], [8], [28].

Gkantsidis et al. [13] recently proposed Avalanche, a content distribution protocol built on top of BitTorrent in the wired Internet. In Avalanche, the original file is encoded using random linear network coding at the source, and coded "pieces" are exchanged and randomly mixed by intermediate peers. The original file can be recovered when a peer collects enough linearly independent coded pieces. Even with only a *local knowledge* of the network, network coding improves the performance of content distribution, because it increases the chance for a peer to pull the last "missing" piece [13], [5].

Recently, content distribution based on network coding has been introduced also in wireless networks [23], [31], [18], [4]. The major difference of this scenario compared to P2P Internet file sharing is that in wireless networks, nodes naturally communicate using multicast exploiting the *broadcast* nature of the wireless medium; while the Internet does not support network level multicast. Network coding in MANETs not only enables peers to fully utilize the broadcast capacity [1]; with proper redundancy, it also can effectively handle mobility, interference, and unreliable channel characteristics – all attributes common in VANETs [23], [31], [18].

While the benefits of network coding have been extensively demonstrated in theory [1], [24] and for a number of applications (though at a rather abstract level of protocol operations) [7], [13], [23], [31], [4], the practical issues of implementing and deploying network coding for content distribution in MANETs have not been well established. In this paper, we develop models that account for nodal resource constraints such as CPU consumption, memory access, and disk I/O. These factors are critical since network coding introduces significant processing overhead at the intermediate nodes. In our scenario, we assume that multiple applications can run on each "embedded" mobile system (e.g., onboard safety/navigation system, data access/entertainment systems, etc.); thus, the file sharing application is resource limited. This may create problems when users try to download large size data such as high resolution maps. The most critical resource in conventional file swarming is the *communication capacity* (i.e., the upload/download bandwidth). However, when network coding is used, other resources (i.e., CPU, memory, and disk) also play an important role in the encoding/decoding at intermediate peers.

The goal of this paper is to model and evaluate network coding resource consumption in content distribution applications. To this end, we abstract the overall behavior of the application, develop models for computation and disk I/O for a given network coding configuration, and integrate these models into an off-the-shelf discrete time network simulator. This allows us to better understand the impact of limited resources in large scale VANET scenarios. This paper extends our earlier work in this area [22] and makes the following contributions:

- Overhead models that can accurately estimate at each node the latency incurred by network coding. The models clearly reflect the relationship between the computation power and the coding rate. This is a major departure from previous results mainly focused on reducing network coding computation overhead [27], [26] or showing network coding feasibility via experiments [12], [36], [26]. Also, our disk I/O model takes the storage access pattern into account, thus, precisely modeling the case when all the necessary pieces have to be loaded back into main memory before encoding. We validate the accuracy of our models via extensive experiments in various platforms (e.g., servers, laptops, and Internet tablets). The model gives us a better insight into analyzing the goodput of content pulling applications. It helps identify the constraints that influence the choice of the best network coding configuration.
- Methods for improving the performance of network coding using the extended simulation environment. 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 [26], [27]. We perform extensive simulations to show the impact of overheads and the effectiveness of these enhancements. These properties are difficult to validate experimentally as they would require large scale testbeds. Our results show that network coding configuration has a great impact on the overall performance, thus resource constraints must be carefully considered to achieve the configuration that yields the best performance. For given resource constraints, we show that our proposed method significantly improves the performance.

Our models are not limited to content distribution scenarios – they can also be applied 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 [37].

This paper significantly enhances our earlier work [22] by including the study of resource constraints of embedded systems in vehicles (Section II-B), extensive testbed experiment results (Section V), and a complete set of simulation results (Section VII). The rest of the paper is organized as follows. In Section II-A, we review network coding based content distribution in VANETs. In Section III, we formulate the network coding configuration and discuss the importance of general resource constraints on the performance of network coding applications. 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 validate our models via experiments. In Section VI, we investigate performance enhancement features. In Section VII, we conduct simulations to show the impact of resource constraints and the effectiveness of enhancement features. Finally, we conclude the paper in Section VIII.

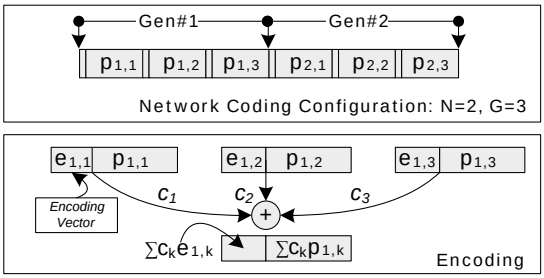


Fig. 1. An illustration of network coding: A file has six B KB pieces and has configured for coding with the number of generations $N = 2$ (i.e., the generation size $G = 3$). When GF(256) is used, the size of an encoding vector is 3 bytes (i.e., G dimension vector). The bottom figure shows how an encoded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

II. Background

A. Content Distribution using Network Coding in VANETs

In this section, we review CodeTorrent, a content distribution protocol using network coding in VANETs [23]. We extend the protocol to support "multi-generation" based network coding for content distribution.

We assume that a file can be uniquely identified with an ID. 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). When distributing a file using network coding, intermediate nodes exchange coded pieces instead of original pieces. For the sake of consistency, we assume that each original piece ℓ has a 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 operation takes 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 checks the received coded pieces for linear dependencies and only keeps linearly independent pieces, which it proceeds to combine into a new coded piece. 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*. Only pieces belonging to the *same* generation are used for encoding. For a given generation, after

Symbol	Description
N	Number of generations
B	Piece size (KB)
G	Generation size (or number of pieces per generation)
$\mathbf{p}_{i,j}$	jth piece in generation i
$p_{i,j,k}$	kth symbol of jth piece in generation i
c_k	Random coefficient drawn over GF (256)
θ	Avg. latency of reading a k-KB chunk from disk (s)
R_d	Storage access rate for encoding (B/s)
δ	Avg. latency of calculating a pair of GF mul/add (s)
T_e	Per symbol encoding time (s)
R_e	Per symbol encoding rate, $1/T_e$ (B/s)
R_b	Bandwidth share of a node in wireless networks (b/s)
N_r	Number of requests per generation

TABLE I
DESCRIPTION OF SYMBOLS

collecting G coded pieces that are linearly independent of each other, a node can recover the original data by solving a set of linear equations. This process repeats until the node collects all N generations. The list of symbols used in the paper is summarized in Table I.

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 containing 100 pieces, the size of a gossip message as large as $10KB$. To reduce the overhead, we use a bit vector to represent the availability of each generation. If a node inquires about a specific generation, the receiver returns the corresponding encoding vector. This allows the requester to determine whether the responding node would be helpful. If so, the requester starts pulling data without further negotiation. For generation selection, a node uses the *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).

For some mobile systems, it is possible that a peer is given a limited buffer (memory) space and the buffer size is smaller than the file size. If the system supports *application-controlled file caching* where the kernel allocates physical pages to an application, the applications can manage 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). If application-controlled file caching is not supported, we use the memory mapping for file access. Since we cannot control the buffer management policy of an Operating System (OS), we use the standard advisory function *madvise()* to give OS hints about access patterns such as WILLNEED and DONTNEED [25]. The application keeps track of the popularity of each generation which is then used to enforce the LRU policy. For instance, the application can evict part of mapped pages with an option DONTNEED. Yet, the overall memory reclamation depends on the OS's reclamation policy. Thus, the LRU policy will be loosely enforced.

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 from neighbors at most k-hops away. 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 [20] 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).

B. Resource Constraints of Embedded Systems in Vehicles

We now briefly review the system configuration of mobile embedded systems in vehicles such as satellite navigation (SatNav). Most mobile embedded systems use NAND flash memory to store operating systems and map data [19]. They typically use shadowing technique such that during system booting time, the entire code image of an OS and applications is copied from flash memory to DRAM for execution. SatNav systems are typically equipped with 64MB or 128MB DRAM and 1 or 2GB of NAND flash memory. For instance, Clarion EZD580 has 64MB DRAM and 2GB NAND flash memory; TomTom GO910 has 64MB DRAM and 1GB NAND flash memory. Recently, vendors have released the second generation SatNav systems that support video displays; e.g., a SatNav a user can plug in a USB memory stick to watch a movie. Thinkware iNavi K2 has a Texas Instrument OMAP 2 multimedia processor (ARM 1136, $<$528Mhz) [2], 256MB DRAM, 8GB of NAND flash memory, which even provides 3D map navigation. Clarion NAX980HD also has a similar configuration, but it has a 40GB HDD. Other portable multimedia players have the similar configurations. For instance, Microsoft Zune has a Freescale i.MX31L multimedia processor (ARM 1136, $<$528Mhz), 64MB DRAM, and 30GB of HDD [15].

In general, mobile embedded systems tend to keep the DRAM size minimal. This is mainly because DRAM is power hungry; i.e., each DRAM refresh cycle dissipates a few miliwatts per MB [19]. Moreover, it is known that for a given workload, there are some threshold values for DRAM and flash memory sizes such that increasing the size beyond those threshold values will not bring any further performance gain [11]. For instance, smartphones and pocket PCs based

on Windows Mobile or Palm OS such as Dell Axim and Palm Treo have about 64 or 128MB DRAM and 128MB or more NAND flash memory. The trend is also true in recent multimedia systems. For instance, Apple TV has a 1 GHz Pentium M Crofton Processor, 256MB DRAM, and a 40G HDD where GeForce Go 7300 video card uses 64MB of DRAM space. Note that a large portion of DRAM is typically used by an OS and the integrated video card; thus applications have limited availability of memory.

Besides, mobile embedded systems typically use low-power and -cost CPUs such as ARM and xScale whose clock speed is less than 1Ghz, or low-power/cost multimedia processors such as Texas Instruments OMAP 2/3 and RMI Alchemy Au series. Although Intel recently released a high performance mobile CPU called Atom (Silverthorne 1.6/8Ghz), its performance is about the same as 900Mhz Celeron M [35].

In summary, we observe that SatNav and embedded systems are limited in terms of DRAM and CPU power, by a few orders of magnitude, compared to standard desktop machines or servers. The recent trend suggests that such a computing/memory resource gap will continue exist between embedded mobile systems and regular desktop machines in the foreseeable future.

III. Problem Definition: Network Coding Configuration

The benefits of network coding based content distribution in VANETs can be attributed to the following: (1) network coding exploits the *broadcast* nature of the wireless medium; (2) network coding mitigates the peer and piece selection problem [5], which is extremely difficult to address in dynamic VANETs; and (3) network coding can effectively handle the random losses due to mobility and interference, which is common in VANETs [31], [23]. One of the most important performance factors in network coding is the "generation size" (i.e., the number of pieces per generation). Real time applications such as P2P streaming have a delay constraint and before the data can be played an entire generation must be received [31], [7]. Thus, the generation size must be small enough to comply with such a constraint. However, content distribution applications in general do not have a strict delay constraint, and this case. we can have a larger generation size.

A. Impact of Generation Size

We first show that one must reduce the number of generations (i.e., increase the generation size) to improve the network coding performance. Assuming that the bandwidth is equally shared by M neighboring nodes over a long time, a node can use a $1/M$ fraction of the channel for sending a request for a piece that it wants to download. As the number of neighbors increases, a node will spend more time overhearing the channel than requesting for the pieces that are needed. Assuming that the piece size is constant and a file has total N pieces, we can consider two extreme scenarios: one that uses a single generation and the other that uses N generations (no coding). In the first scenario, an overheard piece is useful if it is linearly independent of already received 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 $1 - k/N$, i.e., the probability decreases as we collect more generations (the coupon collection problem). 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. For a network coding configuration with a number of generations between 1 and N, the number of steps will fall somewhere inthe $\Theta(N)$ – $\Theta(N \log N)$ range. In this respect, we should choose a small number of generations for better performance. However, it is not always possible to have a few large generations because they adversely impact the delay for downloads. We now investigate practical issues of content distribution using network coding; namely we consider the impact of *communication*, *computation*, and *disk I/O* overheads.

Communication overhead: It is ideal scenario 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 the communication overhead. Recall that each packet must contain a global encoding vector. For instance, when distributing 100KB and 1000KB files using 1KB blocks, we generate 100 and 1000 blocks, respectively. Assuming that GF (256) is used (i.e., 8bit), the overhead is 100B (≈10%) and 1000B (≈100%). In order to reduce overhead, we need to create smaller size generations. In this case, however, packet-level network coding will have many small size generations, thus causing the coupon collection problem. To mitigate this problem, the size of an individual piece may need to scale proportionally to the file size, while considering various link-level statistics [33].

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, i.e. the generation size. Thus, using a small number of large generations (to avoid the coupon collection problem) may result in severe computational overhead that may outweigh the savings in communication. In this case, the encoding process may take more time than data transmission.

Disk I/O overhead: Since the main memory will be shared by a number of applications and the OS, the memory space that can be used for network coding may be limited. This will causes disk I/O overhead, especially when mobile users want to download large size data, e.g. multimedia files. 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 it is significant in VANETs because vehicles may make only short contacts with APs and other vehicles. For example, given a 250m (meter) wireless communication range, vehicles driving in opposite lanes with 50miles/hour have only

11 seconds to communicate with each other. If we 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.

In this paper, we model the overheads incurred by CPU and disk I/O, and use these overhead models to analyze the performance of network coding and determine the main constraints in the network coding performance.

B. Literature Review

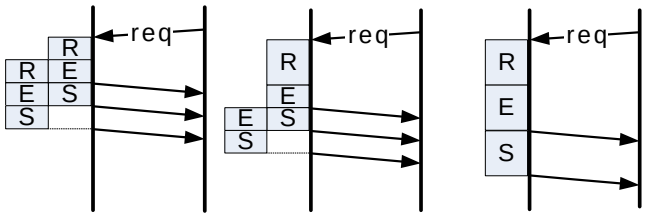
Recent studies on the feasibility of network coding in real testbeds [12], [26], [36] show that the measured performance varies widely depending on the system characteristics. However, the fundamental reason for such performance variation is not well understood. For instance, Gkantsidis et al. [12] reported that their Avalanche scheme incurs little overhead in terms of CPU and I/O using a large scale testbed. On the other hand, it has been empirically observed that that computation overhead degrades the performance, especially when the generation size is large [36], [26]. Various performance enhancement techniques have been proposed [9], [26], [27], [34]. Cooper et al. proposed a sparse network coding where each piece is selected for coding with a certain probability, thus reducing the number of pieces involved in the coding [9], [26]. Maymounkov et al. showed that one can decrease the generation size, yet can still effectively handle the coupon collection problem by using an erasure coding at the generation level [27]. Shojania et al. [34] used CPU acceleration techniques to improve the performance of Galois field operations. However, the impact of CPU overhead and disk I/O overhead was not studied in a unified framework to understand their impact on the network coding performance, especially in the context of VANET scenarios.

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. Finally, we perform experiments to measure the model parameters.

A. Request service procedure

If a node receives a request, it first checks its memory buffer. If the node has the data of the requested generation in the buffer, it can start an encoding process. Otherwise, the node 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, we do not have to worry about the interference between them. Thus we can exploit thread-level parallelism to speed up the overall process. Figure 2 shows three possible types of parallelism.



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

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

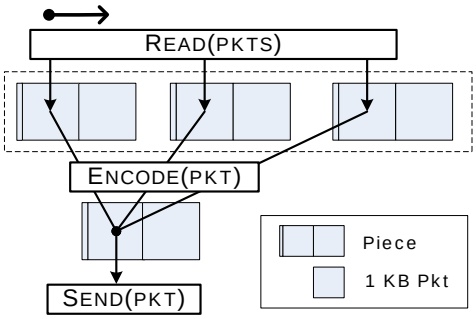


Fig. 3. Chunk-based reading example: $G = 3$ (generation size), $B = 2$KB (piece size). A coded piece is composed of 2 *independent* 1-KB coded packet. Each piece has a header composed of an encoding vector, generation number, etc.

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 used; and the 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 the case of E/S pipeline shown in Figure 2(b), all the pieces for a given generation are read at once and then only E/S steps are pipelined. In the case of no pipeline shown in Figure 2(c), all operations take place sequentially. We assume that a unit of data transfer is 1KB, but to minimize the overhead of system calls (or context switching time) and efficient file access (e.g., the access unit is a page of size 4KB in Linux) ,we can have a larger transfer unit.

B. Overhead models

We now present our disk I/O and computation O/H models, and analyze the goodput of the request handling procedure.

1) Disk I/O overhead model: Disk access involves mechanical motions and is inherently slow by orders 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 drives (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, we can use the same model for mechanical disks and SSDs. To generate a k-KB coded packet, we need to read all the corresponding k-KB data per piece as in Figure 3. We call this "chunk-based reading." 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 data takes $T_d = \theta \cdot G$. The seek latency may be quite prohibitive in the case of mechanical disks compared to SSDs, because the latency is proportional to the generation size. As an alternative, a node can sequentially read the entire piece at once (as in E/S pipeline). In this case, the disk I/O latency is given as $\frac{GB}{R_d}$ where B is the piece size and R_{d_seq} is the sequential data transfer access rate. Thus, the storage access rate R_d for encoding can be summarized as follows:

$$R_d = \begin{cases} \frac{1}{T_d} = \frac{1}{\theta} \cdot \frac{1}{G} \\ \frac{GB}{R_{d\_seq}} \end{cases} \quad (1)$$

2) Computation overhead model: For a given generation g, let $p'_{g,k}$ denote the kth code symbol in a coded piece, and $p_{g,i,k}$ 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 $p'_{g,k}$ is generated as follows:

$$p'_{g,k} = c_1 \cdot p_{g,1,k} + c_2 \cdot p_{g,2,k} + \cdots + c_G \cdot p_{g,G,k}$$

For each symbol ($p_{g,i,k}$) it requires a pair of multiplication (i.e., $c_i \cdot p_{g,i,k}$) and addition ($p'_{g,k}$ += $c_i \cdot p_{g,i,k}$). The per-symbol encoding 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:

$$R_e = \frac{1}{T_e} = \frac{1}{\delta} \cdot \frac{1}{G} \quad (2)$$

Equation 2 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.

C. Goodput analysis

In wireless networks, the bandwidth is shared by multiple nodes. When there are M nodes in a region, we assume that the bandwidth is fairly shared by the 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.

First we consider the goodput of the 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 $GB/R_d + B/R_e + N_r B/R_b$. 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.[1] When we have $R_e < R_b$, the total amount of time is $GB/R_d + N_r B/R_e + B/R_b$. Thus the goodput is given as

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

. In this case, for large N_r, the goodput is approximated to the encoding rate R_e. In order to fully utilize the wireless capacity, 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. The above equations also show that the effect of disk I/O disappears, as the number of requests per generation increases. In Section VI, 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.

V. Model Validation via Experiments

In this section, we validate our models via experiments using a few representative systems.

A. Disk I/O overhead measurement

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 6Y120P0 ATA disk (120GB, 7200rpm, 8MB cache), Pentium 4 2.2Ghz, 1G DRAM (2) Samsung OneNAND Flash SSD (256MB, 0.12μs), TI OMAP 2420, 128MB DRAM (Nokia N800). The measured maximum sequential data access rate of a disk and a SSD is 55.73MB/s (max data rate in spec: 133MB/s) and 14.69MB/s (max data rate in spec: 27MB/s), respectively.

We consider generating a k-KB coded packet by selectively reading all the corresponding k-KB data per piece, as shown

[1]Goodput is the application level throughput, i.e. the number of useful bits per unit of time forwarded by the network from a certain source address to a certain destination, excluding protocol overhead.

in Figure 3 (i.e., a sequence of seek/read pairs). Since the unit of file access is a page whose size is 4KB in typical operating systems such as Linux, k is a multiple of 4. We measure the access latency of reading all the necessary pieces to generate a k-KB coded piece with various k and piece sizes of 20KB and 40KB. Our measurement program scans a 100MB file and records the access time. For each run, we invalidate Linux file buffer cache that keeps a set of pages of a file recently accessed [21]. We report the average of 30 runs.

Figure 4 shows the average latency of reading k-KB chunk from each piece. The results of a disk show that the average latency of a 20KB scenario is much smaller than that of a 40KB scenario, because the larger the piece size, the longer is the seek time. Unlike a mechanical disk, flash memory has a very small seek time, and the impact of piece size on the latency is minimal. In fact, the access pattern of the chunk-based reading is not completely random, and a disk can get the benefit of pre-fetching (or read-ahead) that reads adjacent pages of data of a regular file in advance before they are actually requested, thus minimizing the access latency [38]. To validate this, we turn off the pre-fetching option in our measurement program using the file access advise interface, *posix_fadvise()* with POSIX_FADV_RANDOM option on. Figure 4 (with label "No-Pf") shows the difference between the cases with and without pre-fetching and the partial sequential access can exploit aggressive pre-fetching.

The figure also shows that the average latency increments linearly with chunk size approximately. In our model, the total reading latency is $\theta \cdot G$ where G is generation size. For instance, when the chunk size is $4KB$, the overall latency of a disk and a SSD with $G = 100$ and piece size of 20KB is given as 13.4ms (298KB/s) and 47.8ms (84KB/s) respectively. In contrast, reading all the necessary pieces into the memory (full sequential access) will take 35ms (55.73MB/s) and 136ms (14.69MB/s) respectively. If this rate is much faster than other processes (encoding or sending), we need to consider using the chunk-based reading. In general, we can decide which disk access method and parameters to used depending on the generation size and disk type.

B. Computation overhead measurement

We measure the per symbol encoding time (δ) in three different systems: a server (Intel Xeon Dual Core 5000 3.2GHz), representing a high speed machine, a laptop (Intel Pentium 4 M 1.73GHz), representing a relatively powerful mobile device, and a small mobile device (Nokia N800, 330 MHz TI OMAP 2420). We implement the Galois field operations based on a table lookup with the optimization techniques proposed in [16].[2] We ignore the effect of cache misses since the lookup table fits in the internal cache and the memory access pattern of network coding operation is sequential. We use a Galois field of size 256, and a 12MB file for this measurement. We increase the generation size G from 10 to 50 in the step of 10 blocks. We report the average of 1000 runs for each configuration.

[2]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 [34].

Figure 5 presents the per-symbol encoding latency. The figure shows that the encoding latency increases linearly as shown in Equation 2. In fact, the plots fit well with the lines with slope δ = 5.97ns, δ = 10.42ns, δ = 135ns for Xeon (server), P4 (laptop), and TI (mobile) respectively. Thus, the encoding rate equations are given as $\frac{166.9}{G}$MB/s, $\frac{95.9}{G}$MB/s, and $\frac{7.77}{G}$MB/s respectively. For a small generation size, e.g. $G = 10$, the server machine could generate code packets at the rate of 16.7MB/s, the laptop machine generates them at the rate of 9.6MB/s, and the mobile machine generates them at the rate of 777KB/s. For a relative large generation size, say $G = 100$, these rates drop to 1.67MB/s, 960KB/s, and 77.7KB/s respectively. For laptop and mobile machines, we see that the computation overhead can become the bottleneck compared to the network bandwidth, e.g., 11Mbps 802.11b vs. 7.68Mbps (or 960KB/s) encoding rate.

C. Goodput measurement

We now show the impact of the wireless bandwidth on the performance of network coding. 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 exclude other WiFi interferences. We use the following experiment environments.

- *IBM Thinkpad R52 Laptop*: Each laptop has Intel Pentium 4 M 1.73GHz and 512MB memory, and runs 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. 802.11g is configured as follows: ad hoc mode, no RTS threshold, and 54Mbps (fixed).
- *Nokia N800 Internet Tablet*: Each tablet has a TI OMAP 2420 processor with 128MB DRAM, and runs OS 2007 with Linux Kernel v2.6.18. Nokia N800 has Conexant's CX3110X 802.11b/g chipset. Prism54 softmac driver is used for wireless networking in Linux. In our tested environments, however, 802.11g is not supported; thus we use 802.11b for the measurements. 802.11b is configured as follows: ad hoc mode, no RTS threshold, and 11Mbps (fixed).

The measured goodput is reported in Figure 6. 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

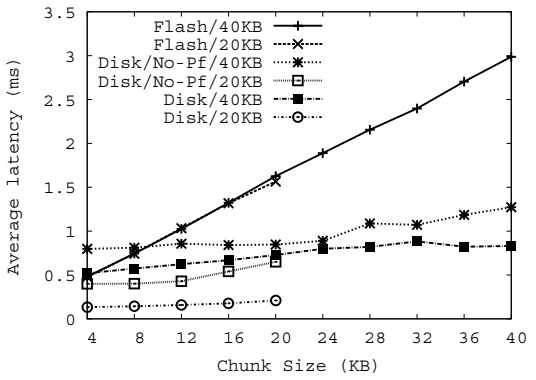


Fig. 4. Average latency of reading k-KB chunk from each piece

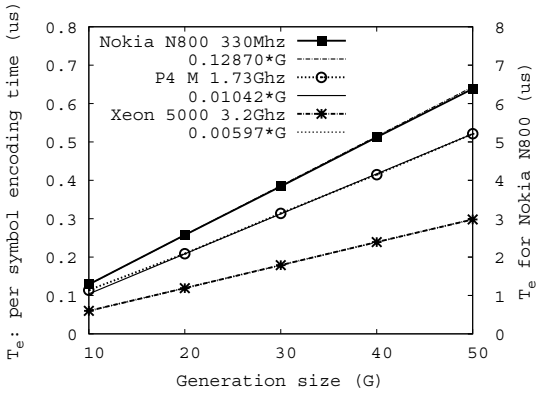
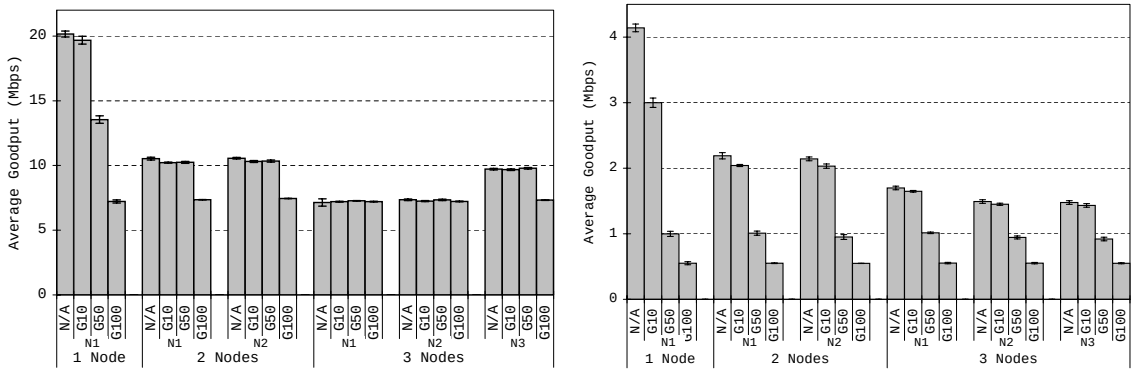


Fig. 5. Per symbol coding latency as a function of generation size G



(a) Thinkpad R52 (Pentium 4M 1.73Ghz, 802.11g) (b) Nokia N800 (TI OMAP 2420, 802.11b)

Fig. 6. Goodput with different generation sizes and interfering nodes. The baseline goodput without network coding is denoted as "N/A"

can support G=50 in the two node scenario and G=100 in the three node scenario.

When the generation size is large (i.e., piece size is small), the measured goodput is quite close to the estimated coding rate. For instance, the measured goodput of laptop and tablet with G=100 is 7.2Mbps and 550kbps respectively. The results are comparable to our model estimates for laptop, 7.68Mbps (960KB/s) and tablet, 621kbps (77KB/s). However, the measured goodput deviates as the generation size increases. For instance, the estimated goodput of Nokia N800 with G=10 is 6.21Mbps, whereas the measured goodput is about 3Mbps. This results from the fact that although packet transmission and coding processes can be parallelized, a piece must be processed in the networking stack in the kernel. One of the main causes is the MAC protocol overhead, because most 802.11 adapters implement part of the 802.11 MAC protocol in the kernel to reduce the cost [30].

VI. Performance Enhancement Features

In this section, we present a novel algorithm called remote buffer generation aware pulling to reduce disk access frequency and present the techniques that reduce computation overheads.

A. Remote Buffer Generation Aware Pulling

When a node uses a *rarest generation first* strategy, it chooses the least available generation measured in terms of the number of nodes. 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 significant 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 now propose the *Remote Buffer Generation Aware Pulling* mechanism 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.[3] The buffer status of a node can be included in periodic "gossip" messages.

[3]As discussed in Section II-A, an application may not have a complete control of memory management. In this case, it can still keep track of generation usage statistics such as popularity of a generation. This information can be gossiped to the neighbors.

Using the buffer status information of the neighbors, a node can search for the generation with the lowest rank among all the generations that are in the remote nodes' buffers. If none of the generations that are useful is present, the node simply sends a request for the rarest generation, which will in turn cause a disk access at a remote node.

B. Fast Network Coding

Sparse 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], [26]. This probabilistic approach, however, does not consider the 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 [23], [26].

Redundant Pre-Coding: The computation overhead can be reduced by decreasing the generation size, yet this will result in a large number of generations, leading to the coupon collection problem. Maymounkov et al. [27] propose Chunked Codes where they keep the generation size small to make the network coding computationally efficient, and use erasure coding at the generation level to circumvent the coupon collection problem. As illustrated in Section III, 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.

VII. Evaluation

In this section, we first describe the implementation details of the protocols that we consider for evaluation, and simulation setup in QualNet [32]. We then present the impact of disk I/O and computation overheads and then evaluate the proposed performance enhancement scheme.

A. Simulation setup

We use IEEE 802.11b PHY/MAC with 11Mbps data rate and Real-Track (RT) mobility model [29]. RT permits to model vehicle mobility in an urban environment more realistically than other simpler and more widely used mobility models such as Random Waypoint (RWP), by restricting the movement of the nodes. The road map input to the RT model is shown in Figure 7, 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 are special nodes called Access Points (APs), which possess 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, we believe that they are 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 while excluding the impact of routing overheads.

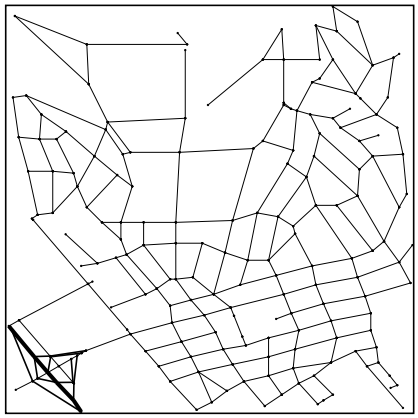


Fig. 7. Westwood area map used for the RT model

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=$\frac{48}{G}$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. We also test the configuration of Nokia N800 whose R_e=$\frac{7.77}{G}$MB/s. For this, we use the R/E/S pipeline for multi-threading due to its fast random access capability of a SSD. We use two file sizes (5MB and 10MB) for Nokia N800.

The disk and coding delays are scheduled based on the disk I/O and computation models respectively. When a request comes in, a node calculates the delay for which packet transmission is delayed in the network queue. We define

the "download delay" as the elapsed time for a node to finish downloading a file. How fast the overall downloading process finishes measures the efficiency of a scheme. For each configuration, we report the average value of 30 runs with the 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 the overheads are not considered. We also vary the availability of buffer space: 50% , 75%, 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 8 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 9, 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. The figure also shows that the processing capability is important; i.e., for a given file, Nokia 800 performs worse than Intel Pentium 4. For instance, for a 10MB file, it takes 561s and 1133s for Nokia N800 and Intel Pentium 4 respectively. As the number of generation increases, the impact of network coding overheads disappears, and thus, the delay difference between these machines decreases.

Now consider the cases where the buffer size is smaller than the file size (see Figures 10 and 11). The impacts of disk I/O can be clearly seen by comparing Figures 10 and 11 with Figure 9. Contrary to our common belief that network coding improves the file swarming performance [23], the download delay can be 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, which is as expected.

Remote Buffer Generation-Aware Pulling (RBGAP): Figure 12 and Figure 13 show the download delay and the number of pieces read from the disk with different buffer sizes, namely 100%, 75% and 50%. 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/Os, thereby reducing the total 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 14 presents the results. As the coding density decreases, the download delay also tends to decrease. 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.

Redundant Pre-Coding: The computation overhead can be reduced by decreasing the generation size (i.e., increasing the number of generations). However, our previous results of the *No Coding* scenario show that a large number of generations cause significant performance degradation. To show this impact more clearly, we evaluate the pre-coding mechanism as follows. We create $N' = \lfloor (1+\lambda/2)N \rfloor$. Nodes can recover the original content by collecting any subset of the generations, $(1 + \lambda/4)N$. We set $\lambda = 1$. For example, when an original file is divided into 50 generations, we need 62 out of 75 generations. Figure 15 shows the results of 50MB and 25MB files with the number of generations $N = 1, 5, 10, 50$. The figure shows that pre-coding increases the average download delay, mainly because the effectiveness of overhearing decreases with the number of generations. Moreover, nodes tend to download more number of generations than necessary, thus wasting valuable resources. Hence, we conclude that the generation level pre-coding is less efficient in our scenario.

VIII. Conclusion

The main focus of this paper has been to investigate the impact of practical resource constraints of mobile devices (namely disk I/O, computation overhead, memory constraints, and wireless bandwidth) on the performance of content distribution using network coding in a highly dynamic wireless network environment such as VANETs. We began our study by modeling the impact of these resource constraints on the network coding process and identified the key performance parameters that will mainly determine the goodput of network coding. We then validated our model by comparing them with the performance numbers that we obtained from real systems, including desktop machines, laptops, and handheld devices.

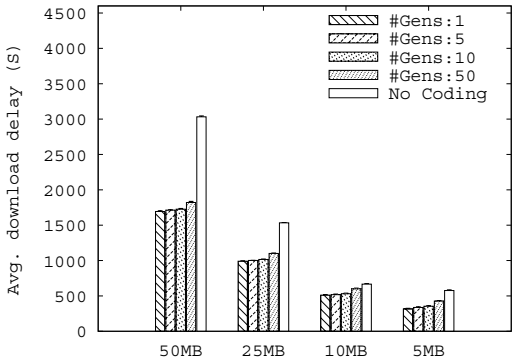


Fig. 8. Download delay without O/H

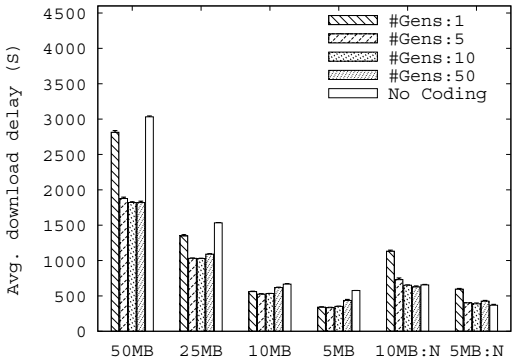


Fig. 9. Download delay with O/H: Buffer 100% (10MB:N and 5MB:N show the results of Nokia N800 configuration)

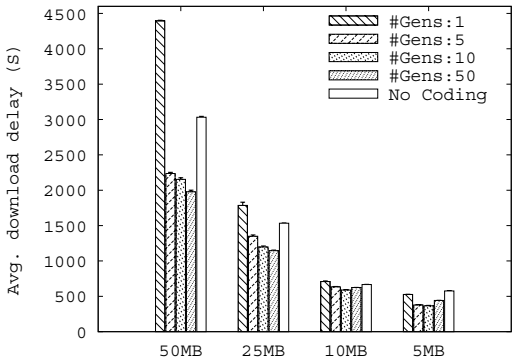


Fig. 10. Download delay with O/H: Buffer 75%

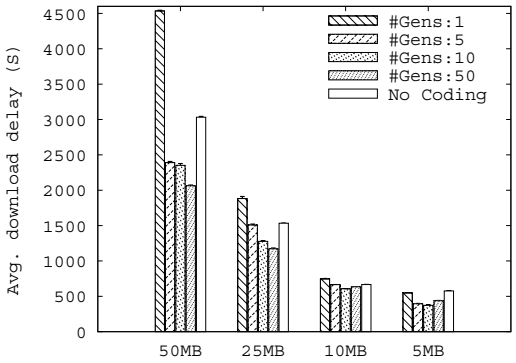


Fig. 11. Download delay with O/H: Buffer 50%

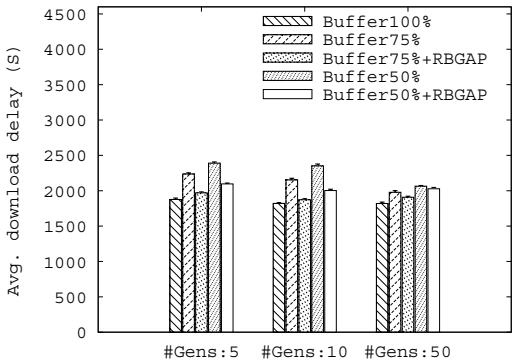


Fig. 12. Download delay with RBGAP (50MB file)

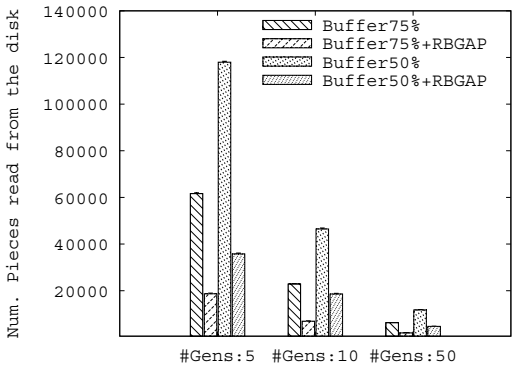


Fig. 13. Total number of pieces read from the disk (50MB file)

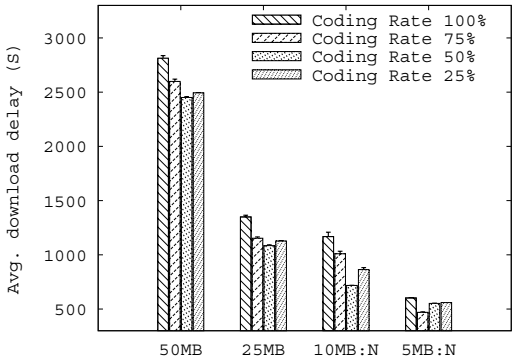


Fig. 14. Impact of sparse coding

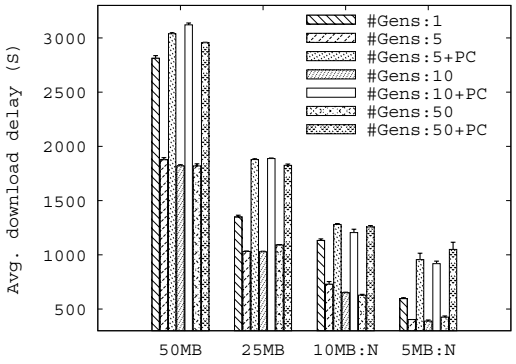


Fig. 15. Impact of pre-coding

Based on the intuition that we gained from this modeling and measurement exercise, we have designed a novel data pulling strategy called, the remote buffer generation aware pulling (RBGAP) that can significantly reduce disk I/O overhead at the remote node, thereby can reduce the overall delay of content distribution. To evaluate these ideas in a large scale network, we have implemented our overhead model and the data pulling scheme in the QualNet wireless network simulator. From the simulation study, we have obtained several new insights that will help improving the performance of applications based on network coding. They include: (1) resource constraints have a significant impact on the performance of network coding; (2) data pulling that considers the resource constraints of remote nodes can significantly improve the performance; (3) the benefit of sparse random network coding is not always obvious, and its parameter should be carefully chosen to perform well; and (4) generation level pre-coding is not as efficient in a highly dynamic environment as 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] Arm Architecture. http://en.wikipedia.org/wiki/ARM_architecture.

[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] F. Douglis, R. Cáceres, B. Marsh, F. Kaashoek, K. Li, and J. Tauber. Storage Alternatives for Mobile Computers. In *USENIX'94*, San Francisco, CA, Nov. 1994.

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

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

[14] P. Gupta and P. R. Kumar. The Capacity of Wireless Networks. *IEEE Transactions on Information Theory*, 46(2), 2000.

[15] Inside Zune – the Hardware. http://happymac.ch/happyMacEN/News/1C1515C1-A47C-4ADD-8A16-9B3A21C9AA5A.html.

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

[17] 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.

[18] M. Johnson, L. D. Nardis, and K. Ramchandran. Collaborative Content Distribution for Vehicular Ad Hoc Networks. In *Allerton'06*, Monticello, IL, Sep. 2006.

[19] M. G. Khatib, B.-J. van der Zwaag, P. H. Hartel, and G. J. M. Smit. Interposing Flash between Disk and DRAM to Save Energy for Streaming Workloads. In *ESTIMedia'07*, Salzburg, Austria, Oct. 2007.

[20] 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.

[21] A. Lawrence. Invalidating The Linux Buffer Cache. In *Linux Developer News*, Jan. 2007.

[22] S.-H. Lee, U. Lee, K.-W. Lee, and M. Gerla. Content Distribution in VANETs using Network Coding: The Effect of Disk I/O and Processing O/H. In *SECON'08*, San Francisco, CA, June 2008.

[23] 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.

[24] 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.

[25] R. Love. *Linux System Programming*. O'Reilly, 2007.

[26] 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.

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

[28] 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.

[29] 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.

[30] M. Neufeld, J. Fifield, C. Doerr, A. Sheth, and D. Grunwald. SoftMAC—Flexible Wireless Research Platform. In *HotNets-IV*, College Park, MD, Nov. 2005.

[31] 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.

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

[33] 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.

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

[35] Erster Benchmark von Intels Silverthorne. http://www.computerbase.de/news/hardware/prozessoren/intel/2008/maerz/erster_benchmark_intels_silverthorne.

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

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

[38] F. Wu, H. Xi, J. Li, and N. Zou. Linux Readahead: Less Tricks for More. In *Linux Symposium'07*, Ottawa, Ontario, June 2007.