Understanding Processing Overheads of Network Coding-Based Content Distribution in VANETs

Uichin Lee, *Member*, *IEEE*, Seung-Hoon Lee, *Member*, *IEEE*, Kang-Won Lee, *Senior Member*, *IEEE*, and Mario Gerla, *Fellow*, *IEEE*

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 twofold. 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

1 Introduction

Intervehicular communication has received a lot of attention recently due to safety concerns for drivers. Although the main focus of intervehicular 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 intervehicular communications, has allocated several of its "service" channels specifically to nonsafety usage [14].

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, and so on. 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 [19], [7], [31].

Gkantsidis and Rodriguez [12] 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 [12], [5].

Content distribution based on network coding-based content distribution has been also introduced also in wireless networks [23], [34], [15], [4], [24]. 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], [34], [15], [24].

- *U. Lee is with the Department of Knowledge Service Engineering, Korea Advanced Institute of Science and Technology, 291 Daehak-ro, Yuseong-gu, Daejeon 350-701, Korea. E-mail: uclee@kaist.ac.kr.*
- *S.-H. Lee is with the Department of Computer Science, University of California, Los Angeles, 420 Westwood Plaza Los Angeles, CA 90095. E-mail: shlee@cs.ucla.edu.*
- *K.-W. Lee is with IBM T.J. Watson Research Center, Wireless Networking Department, Yorktown Heights, NY 10598. E-mail: kangwon@us.ibm.com.*
- *M. Gerla is with the Department of Computer Science, University of California, Los Angeles, 420 Westwood Plaza Los Angeles, CA 90095. E-mail: gerla@cs.ucla.edu.*

Manuscript received 3 Aug. 2012; accepted 5 Oct. 2012; published online 19 Oct. 2012.
Recommended for acceptance by S. Guo.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2012-08-0699.
Digital Object Identifier no. 10.1109/TPDS.2012.306.

While the benefits of network coding have been extensively demonstrated in theory [1], [25] and for a number of applications (though at a rather abstract level of protocol operations) [6], [12], [23], [34], [4], [24], the practical issues of configuring, simulating, 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 limit, and disk access. 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 access 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 [28], [27] or showing network coding feasibility via experiments [11], [27], [42], [39], [41]. Also, our disk access 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 access overhead for local computation; and 2) we experiment with several computationally efficient network coding methods applicable for MANETs [27], [28]. 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.

The following items summarize the key distinctions from our earlier work [22]:

- Section 2.2: We include a brief survey of resource constraints of embedded systems for vehicles. Also, we report that a computing/memory resource gap will continue to exist between embedded mobile systems and regular desktop machines in the foreseeable future.
- Section 5: Compared to the earlier work, we perform extensive testbed experiments (e.g., disk I/O and computation overhead measurements). Further, we additionally evaluate the proposed models using Nokia N800, a portable smart device. The experiment results are now presented in a separate section.
- Section 7: We present a complete set of simulation results; compared to the earlier work, we additionally include the simulation results of Nokia N800 model, various buffer sizes (100, 75, 50 percent), disk I/O operations, and chunked coding. Further, we discuss other remaining issues such as decoding operations.
- Section 8: We perform an extensive survey of recent articles to validate the relevance of our work. We show that the proposed models can be extended to consider the recent performance enhancement techniques. Further, our models are not limited to content distribution scenarios, but they are applicable to the other related areas such as an opportunistic routing protocol with network coding [4] and network coding-based message dissemination in delay tolerant networks [43].

The rest of the paper is organized as follows: In Section 2.1, we review network coding-based content distribution in VANETs. In Section 3, we formulate the network coding configuration and discuss the importance of general resource constraints on the performance of network coding applications. In Section 4, we propose disk access and computation overhead models, and analyze the goodput of the overall content pulling procedure. In Section 5, we validate our models via experiments. In Section 6, we investigate performance enhancement features. In Section 7, we conduct simulations to show the impact of resource constraints and the effectiveness of enhancement features and discuss some of the remaining issues. In Section 8, we review the related literature on network coding experimentation and performance enhancement techniques. Finally, we conclude the paper in Section 9.

2 Background

2.1 Content Distribution Using Network Coding in VANETs

This section reviews CodeTorrent in VANETs [23]. In this paper, we extend the existing protocol to support

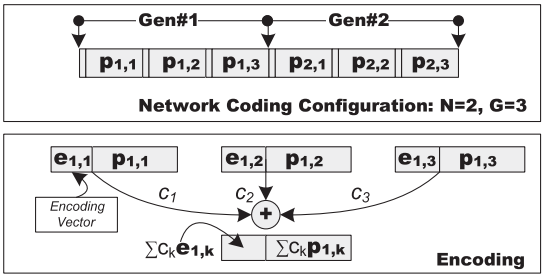


Fig. 1. An illustration of network coding: A file has six B KB pieces and has configured for coding with the number of generations $N = 2$ (i.e., the generation size $G = 3$). When GF(256) is used, the size of an encoding vector is 3 bytes (i.e., G dimension vector). The bottom figure shows how an encoded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

"multigeneration" based network coding for large-size 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., $\mathbf{p}_i = [\mathbf{p}_{i,1}, \mathbf{p}_{i,2}, \ldots, \mathbf{p}_{i,G}]^T$ for $i = 1, \ldots, N$ (see Fig. 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 $\tilde{\mathbf{p}}_{i,\ell} = [\mathbf{e}_\ell \ \mathbf{p}_{i,\ell}]$ (i.e., coefficients and coded block). For each generation i, the server creates a coded piece $\tilde{\mathbf{x}}_\mathbf{i}$ 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 an 8-bit field, GF(256). Each piece contains a unit vector at the source, and thus, the resulting encoding vector is the same as $[c_1 \ldots c_G]$. 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 i, after collecting G coded pieces ($\tilde{\mathbf{x}}_{i,k}$) that are linearly independent of each other, a node can recover the original data by solving a set of linear equations. These G coded pieces have a $G \times G$ matrix $\mathbf{C_i}$ for encoding coefficients and a $G \times B$ matrix $\mathbf{x_i}$ for coded data. The original pieces $\mathbf{p_i}$ can be recovered as: $\mathbf{p_i} = \mathbf{C_i}^{-1}\mathbf{x_i}$. This process repeats until the node collects all N generations. The list of symbols is summarized in Table 1.

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 10 KB. 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).

TABLE 1
Description of Symbols

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
$\tilde{\mathbf{p}}_{i,j}$	$\mathbf{p}_{i,j}$ with encoding vector attached
$\mathbf{C_i}$	$G \times G$ encoding coeff. matrix of G coded pieces from gen i
$\mathbf{x_i}$	$G \times B$ coded data matrix of G coded pieces from gen i
$p_{i,j,k}$	kth symbol of jth piece in generation i
$p'_{i,k}$	kth symbol of a coded piece from generation i
c_k	Random coefficient drawn over Galois Field, GF (256)
θ_k	Avg. latency of reading a k-KB chunk from storage (s)
R_d	Storage access rate for encoding (B/s)
δ	Avg. latency of computing a pair of GF mul/add operations (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

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 dependence 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 that it can choose appropriate read-ahead (reading expected pages ahead based on prediction) and caching techniques [26]. For instance, we use SEQUENTIAL when expecting page references in sequential order, WILLNEED when expecting page access in the near future—both SEQUENTIOAL and WILLNEED can be useful for disk read-ahead; and DONTNEED when we do not access pages in the near future and allow OS to free the cached pages. 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 [19] where k-hop limited controlled flooding of resource availability can be used as a *route discovery request* and a data pull request as a *route reply*.

Our protocol is a pull-based approach as in existing content distribution protocols in the Internet (e.g., BitTorrent, Avalanche). An alternative approach is to use a push-based approach as in multimedia streaming where received pieces are automatically pushed (or broadcast) to neighboring nodes [34], [24]. While a push-based approach works well when disseminating highly popular content or emergency messages, a pull-based approach is more preferable particularly when multiple files are distributed over the network, and each node may want to download a different set of files.

2.2 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 [17]. 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. This means that large portion of DRAM is typically used by an OS (and optionally, the integrated video card); thus, applications have limited availability of memory. SatNav systems are typically equipped with 128-MB DRAM and support various size of internal/external NAND flash memory. For instance, Clarion NX501 has 128-MB DRAM and supports up to 16-GB NAND external flash memory; TomTom GO 1015 LIVE has 128-MB DRAM and supports up to 8-GB NAND flash memory. Besides, mobile embedded systems typically use low-power and -cost CPUs such as ARM and xScale whose clock speed is around 1 Ghz, or low-power/cost multimedia processors such as Texas Instruments OMAP 2/3 and RMI Alchemy Au series. Those processors are significantly inferior to existing desktop CPUs; e.g., the performance of Intel Atom Silverthorne 1.6/8 Ghz, is comparable with Intel Celeron M 900 Mhz [40].

Recently, vendors released the second generation SatNav systems that support video displays; e.g., plugging in a USB memory stick to watch a movie. Thinkware iNavi K2 has a Texas Instrument OMAP 2 multimedia processor (ARM 1136, <528 Mhz) [2], 256-MB DRAM, 8 GB of NAND flash memory, which even provides 3D map navigation. Clarion NAX980HD also has a similar configuration, but it has a 40-GB HDD. Further, recent high-end smart devices (e.g., smartphones and tablets) are equipped with larger memory (up to 1 GB). Yet, the size of DRAM is one of the key limiting constraints; DRAM is power hungry as each refresh cycle dissipates a few milliwatts per MB [17]. Unlike dedicated devices (e.g., SatNav, multimedia play), these smart devices are designed to support general purpose applications and their workload demand would be much higher, thus requiring larger memory and computing power [10]. Due to power constraints, device manufacturers use low-power DRAM and multiple cores (e.g., 1.2-Ghz dual-core ARM CPU in Galaxy Nexus) [21].

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

3 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 [34], [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 [34], [6]. 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.

We first show that one must reduce the number of generations (meaning increasing 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

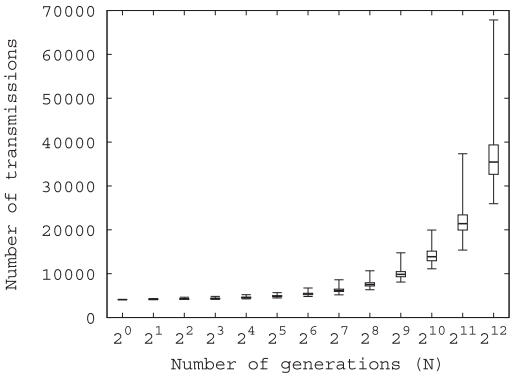


Fig. 2. Impact of the number of generations (N). For a file of size 2^{12} pieces, $N = 2^k$ means that there are 2^k generations, each of which has 2^{12-k} pieces.

more generations (the coupon collection problem). Given that an overheard piece is useful with high probability [9], 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 in the range of $[\Theta(N), \Theta(N \log N)]$. In this respect, we should choose a small number of generations to mitigate the coupon collection problem.

For illustration, we simulated the above scenario as follows: For a given file whose number of pieces is $2^{12} = 4{,}096$, we vary the number of generations N from 1 to 2^{12}. Here, $N = 2^k$ means that there are 2^k generations, each of which has 2^{12-k} pieces. We assume that a piece can be fit into a single packet. For network coding, we assume that the field size is large enough such that a randomly generated piece for a given generation is helpful with high probability. The overall process is similar to a bin/ball problem where there are 2^k bins, each of which has the capacity of 2^{12-k} balls, and a ball is thrown to a random bin. For each configuration, we measure the number of packet transmissions (balls thrown) to download the whole file (fill all bins) for 1,024 times and report the distribution using the box-and-whisker plot in Fig. 2.[1] The results show that as the number of generations increases, the distribution of the number of transmissions is dispersed with a larger median value. If the number of generations is small, the total number of transmissions is close to $2^{12} = 4{,}096$. In particular, we observe that the number of transmissions for the case of $N = 2^{12}$ is approximately $\log N$ factor greater than that for the case of $N = 1$. The above results clearly show that 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 due to network coding processing overhead.

We investigate practical issues of content distribution using network coding; namely we consider the impact of *communication*, *computation*, and *disk access* overheads.

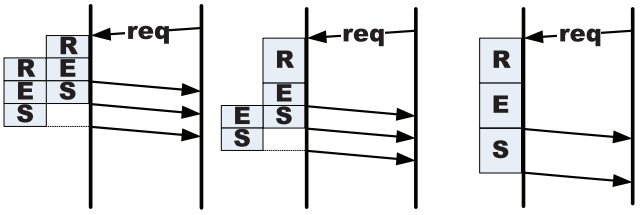
1. A boxplot depicts groups of numerical data through their five-number summaries: the smallest observation (sample minimum), lower quartile (Q1), median (Q2), upper quartile (Q3), and largest observation (sample maximum).

Communication overhead. It is the 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 100 KB and 1,000 KB files using 1 KB blocks, we generate 100 and 1,000 blocks, respectively. Assuming that GF (256) is used (i.e., 8 bit), the overhead is 100B ($\approx 10\%$) and 1000B ($\approx 100\%$). In order to reduce overhead, we need to limit the number of blocks per generation. If the number of blocks is small (i.e., the number of generations is large), we will suffer from the coupon collection problem. To mitigate this problem, the size of an individual piece needs to scale proportionally to the file size, while considering various link-level statistics [37].

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 access 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 access 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 network coding processing (i.e., computation and disk access) is huge, and it is significant in VANETs because vehicles may make only short contacts with APs and other vehicles. For example, given a radio range of 250 m, vehicles driving in opposite lanes with 50 miles/hour have only 11 s to communicate with each other. If we assume that the size of a generation is 40 MB. The nominal data transfer rate of hard disks or flash memory-based solid state disks is about 40 MB/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. Moreover, if the number of pieces per generation is large, it may take a long time to create a new coded piece, say several 100 ms. In this case, due to disk access and computation overheads, one cannot fully utilize one's wireless bandwidth for data transfer. We conclude that network coding processing overheads have a significant impact on the performance of network coding-based file swarming. In this paper, we model the overheads incurred by disk access and computation for content encoding to determine the main constraints of the network coding performance. Given that our goal is to maximize the generation size, we use such models to find the maximum generation size that can fully utilize the bandwidth share.



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

Fig. 3. Possible parallelism scenarios with piece size $B = 2$ KB.

4 Disk Access and Computation Overhead Models

We present the request processing procedure of a serving peer and model both disk access and computation overhead and analyze the throughput of the procedure. We then perform experiments to measure the model parameters.

4.1 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); the interference would be minimal. Thus, we can exploit thread-level parallelism to speed up the overall process. Fig. 3 shows three possible types of parallelism.

In Fig. 4, we consider an example with R/E/S pipeline when the generation size $G = 3$ and the piece size $B = 2$ KB. To generate an ith coded symbol, only three symbols (from each piece k denoted as p_i^k) are used (i.e., $\sum_{k=1}^{3} c_k p_i^k$); symbols with different index i are independent of each other. Assuming that the unit of data transfer is 1 KB, the communication thread sends the newly encoded packet as soon as a coded packet is ready. Note that if a piece size is B KB, one has to send a 1 KB coded packet B times. 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 access thread reads the necessary parts of the generation from the disk (READ), then signals the encoding thread. After the encoding is finished, the communication thread sends the newly generated encoded packet out to the requesting peer (SEND). In the case of E/S pipeline shown in Fig. 3b, 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 Fig. 3c, all operations take place sequentially. We assume that a unit of data transfer is 1 KB, 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 4 KB in Linux), we can have a larger transfer unit.

4.2 Overhead Models

We now present our disk access and computation overhead models, and analyze the goodput of the request handling procedure.

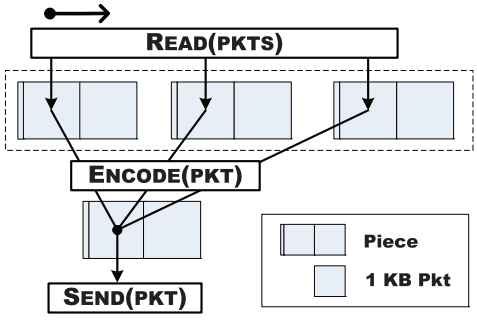


Fig. 4. Chunk-based reading example: $C = 1$ KB (chunk size), $G = 3$ (generation size), $B = 2$ KB (piece size). A coded piece is composed of two *independent* 1-KB coded packet. Each piece has a header composed of an encoding vector, generation number, and so on.

4.2.1 Disk Access 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 10-15 ms 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 40 MB/s (for desktop-grade disks) or 80 MB/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 0.1 ms of seek time and the read/write rates are 40 MB/s and 32 MB/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 C-KB coded packet, we need to read all the corresponding C-KB data per piece as in Fig. 4. We call this "chunk-based reading" where the size of a chunk is denoted as C. The access pattern will be a sequence of seek/read pairs. For a given chuck size C, let θ_C denote the average latency to perform a pair of seek/read operation. The overall time to read all the relevant data takes $T_d(C) = \theta_C \cdot G$. Thus, the disk access rate (KB/s) is given as

$$R_d(C) = \frac{C}{T_d} = \frac{C}{\theta_C G}. \quad (1)$$

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 at the sequential data transfer rate as in E/S pipeline; i.e., $R_d = R_{d\_seq}$ where R_{d_seq} the sequential data transfer access rate. In this case, the disk access latency is given as $\frac{GB}{R_{d\_seq}}$ where B is the piece size.

4.2.2 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

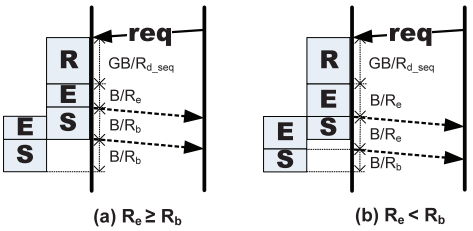


Fig. 5. Latency of the E/S pipeline with $N_r = 2$.

piece in the buffer. Let c_i for $i = 1, \ldots, G$ 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 8 bit). Each code symbol $p'_{g,k}$ is generated as follows:

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

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:

$$R_e = \frac{1}{T_e} = \frac{1}{\delta} \cdot \frac{1}{G}. \quad (3)$$

Equation (3) 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.

4.3 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.

Let us consider the goodput of the E/S pipeline. 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. Assume that there are total N_r requests of a specific generation from one's neighboring nodes. Recall that G is the generation size, B is the piece size, R_e is encoding rate, and $R_d = R_{d\_seq}$ is data transfer rate. If we have $R_e \geq R_b$, the latency is mainly dominated by the wireless bandwidth share (see Fig. 5a). The time to generate the first coded piece is B/R_e seconds. It takes B/R_b seconds for data transfer, during which the next coded packet can be generated. Thus, coded packets are sent one after another. The latency to transfer N_r pieces is simply $GB/R_d + B/R_e + N_rB/R_b$. 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. Similarly, when we have $R_e < R_b$, the total amount of time is $GB/R_d + N_rB/R_e + B/R_b$ (see Fig. 5b). 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.

Our goodput analysis shows that to fully utilize the wireless bandwidth share, the encoding rate R_e must be greater than or equal to the bandwidth share R_b, i.e., $R_e \geq R_b$. By replacing R_e with $\frac{1}{\delta G}$, we have the following inequality:

$$G \leq \frac{1}{\delta R_b}.$$

Thus, we can find the maximum generation size that satisfies the condition. The above equations also show that the effect of disk access disappears, as the number of requests per generation increases. In Section 6, 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.

5 Model Validation via Experiments

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

5.1 Disk Access Overhead Measurement

We investigate the impact of the pairwise access patterns (seek/read pairs) by measuring θ in real systems. We use two sets of scenarios: 1) Maxtor 6Y120P0 ATA disk (120 GB, 7,200 rpm, 8 MB cache), Pentium 4 2.2 Ghz, 1G DRAM 2) Samsung OneNAND Flash SSD (256 MB, $0.12\ \mu$s), TI OMAP 2420, 128-MB DRAM (Nokia N800). The measured average sequential data access rate R_{d_seq} of a disk and an SSD is 55.73 MB/s (max data rate in spec: 133 MB/s) and 14.69 MB/s (max data rate in spec: 27 MB/s), respectively.

We consider generating a C-KB coded packet by selectively reading all the corresponding C-KB data per piece, as shown in Fig. 4 (i.e., a sequence of seek/read pairs). Since the unit of file access is a page whose size is 4 KB in typical operating systems such as Linux, C is a multiple of 4. We measure the access latency of reading all the necessary pieces to generate a C-KB coded piece with various C and piece sizes of 20 KB and 40 KB. Our measurement program scans a 100-MB 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 [20]. We report the average of 30 runs.

Fig. 6 shows the average latency of reading C-KB chunk from each piece. The results of a disk show that the average latency of a 20-KB scenario is much smaller than that of a 40-KB scenario. The latter has higher latency in terms of data reading and seek time, because the larger the piece size, the longer is the read latency and the physical distance between pieces in the disk (larger seek time). When the chunk size is small (say, 4 KB), the latency is dominated by the seek latency. As we increase the chunk size, more data must be read from the disk and the latency linearly increases. Unlike a mechanical disk, flash memory has a

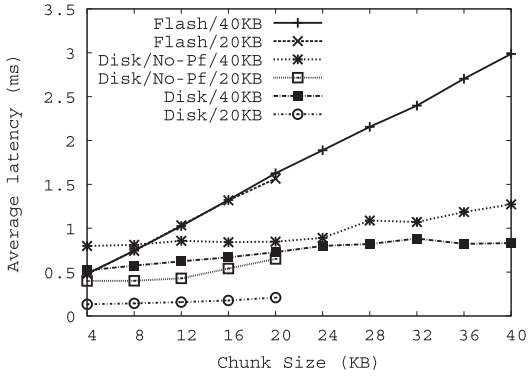


Fig. 6. Average latency of reading C-KB chunk from each piece.

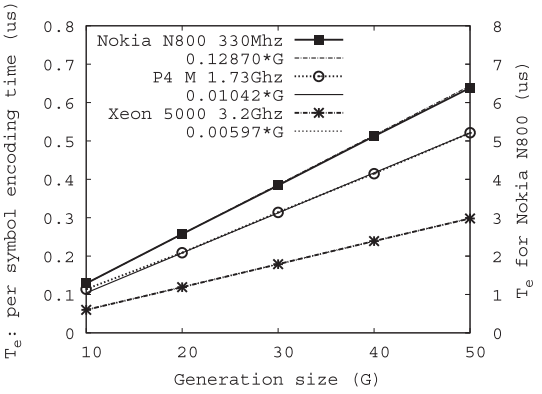


Fig. 7. Per symbol coding latency as a function of generation size G.

negligible seek time. Since the impact of piece size on the latency is minimal, we see that the latency linearly increases with the chunk size.

Note that the access pattern of the chunk-based reading is not completely random. A disk can get the benefit of prefetching (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 [44]. To validate this, we turn off the prefetching option in our measurement program using the file access advise interface, *posix_fadvise()* with POSIX_FADV_RANDOM option on. The Linux kernel will not perform prefetching when this option is on. Fig. 6 (with label "No-Pf") shows the difference between the cases with and without prefetching. It clearly shows that the partial sequential access can exploit aggressive prefetching to achieve better performance. When the chunk size is the same as piece size, its performance is close to the case with sequential access.

In our model, the total reading latency is $\theta_C \cdot G$ where G is the generation size. For instance, when the chunk size is 4 KB, the overall latency of a disk and an SSD with $G = 100$ and piece size of 20 KB is given as 13.4 ms ($R_d = 298$ KB/s) and 47.8 ms ($R_d = 84$ KB/s), respectively. In contrast, reading all the necessary pieces into the memory (full sequential access) at the speed of R_{d_seq} will take 35 ms and 136 ms, 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 be used depending on the generation size and disk type.

5.2 Computation Overhead Measurement

We measure the per symbol encoding time (δ) in three different systems: a server (Intel Xeon Dual Core 5000 3.2 GHz), representing a high-speed machine, a laptop (Intel Pentium 4 M 1.73 GHz), 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 [13].[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 12-MB 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 1,000 runs for each configuration.

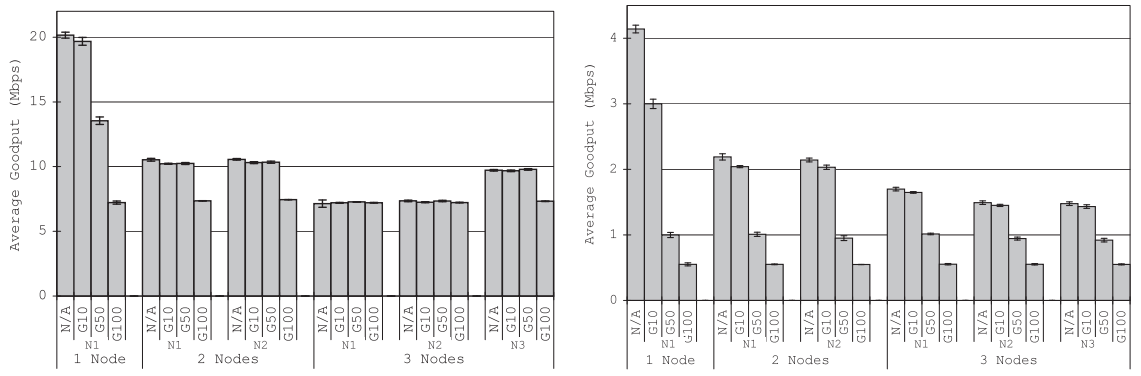
Fig. 7 presents the per-symbol encoding latency. The figure shows that the encoding latency increases linearly as shown in (3). In fact, the plots fit well with the lines with slope $\delta = 5.97$ ns, $\delta = 10.42$ ns, and $\delta = 135$ ns for Xeon (server), P4 (laptop), and TI (mobile), respectively. 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 16.7 MB/s, the laptop machine generates them at the rate of 9.6 MB/s, and the mobile machine generates them at the rate of 777 KB/s. For a relative large generation size, say $G = 100$, these rates drop to 1.67 MB/s, 960 KB/s, and 77.7 KB/s respectively. For laptop and mobile machines, we see that the computation overhead can become the bottleneck compared to the network bandwidth, e.g., 11 Mbps 802.11b versus 7.68 Mbps (=960 KB/s) encoding rate.

5.3 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 60 MB of data. We run each configuration 30 times and report the average goodput (i.e., application level throughput) with the 95 percent confidence interval. Readers can find the definition of goodput in Section 4.3. Data transfers of clients are initiated by the server via parallel SSH. We perform the experiment in the early morning (2-6 a.m.) to exclude other WiFi interferences. We use the following experiment environments:

- *IBM Thinkpad R52 Laptop.* Each laptop has Intel Pentium 4 M 1.73 GHz and 512-MB memory, and runs Fedora Core 5 with Linux Kernel v2.6.19. We use ORiNOCO 11b/g PC Cards (8471-WD) and the

2. Shojania and Li [38] 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.



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

Fig. 8. Goodput with different generation sizes and interfering nodes. The baseline goodput without network coding is denoted as "N/A."

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 54 Mbps (fixed).

- *Nokia N800 Internet Tablet.* Each tablet has a TI OMAP 2420 processor with 128-MB 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 11 Mbps (fixed).

The measured goodput is reported in Fig. 8. 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., for a laptop, a node can support $G = 50$ in the two node scenario and $G = 100$ in the three node scenario.

When the generation size G is large (i.e., the number of generations N 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 7.2 Mbps and 550 kbps respectively. The results are comparable to our model estimates for laptop, $7.68\ \text{Mbps}(= \frac{95.9}{G=100}\ \text{MB/s} = 959\ \text{KB/s})$ and tablet, $621\ \text{kbps}\ (= \frac{7.77}{G=100}\ \text{MB/s} = 77\ \text{KB/s})$. However, the measured goodput deviates as the generation size increases. For instance, the estimated goodput of Nokia N800 with $G = 10$ is 6.21 Mbps, whereas the measured goodput is about 3 Mbps. 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 [33].

6 Performance Enhancement Features

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

6.1 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 access. 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 access.

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 access 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. 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.

6.2 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 [8] has been proposed to achieve this: each piece is selected with probability $p \geq (\log_2 G + d)/G$

3. As discussed in Section 2.1, 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.

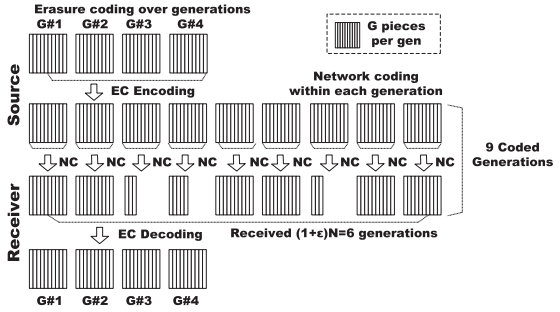


Fig. 9. Illustration of chunked code with $\epsilon = 1/2$.

where G is the generation size and d is a nonnegative constant [8], [27]. 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], [27].

Chunked code. 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. [28] propose chunked code 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 shown in Fig. 9, erasure codes encode N original generations into N *coded generations* $(N > N)$, so that the original data message can be reconstructed after receiving a subset of encoded generations, namely $(1+\epsilon)N$ where ϵ is a positive constant [29]. Network coding is performed within an encoded generation.

However, chunked code has the following limitations. Since a node pulls a random generation, there will be many outstanding or partially filled coded generations waiting for download completion. For instance, there are three incomplete coded generations in Fig. 19. The problem will be more pronounced, when the source distributes much larger number of coded generations. Moreover, as illustrated in Section 3, it cannot 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 validate these observations via extensive simulations.

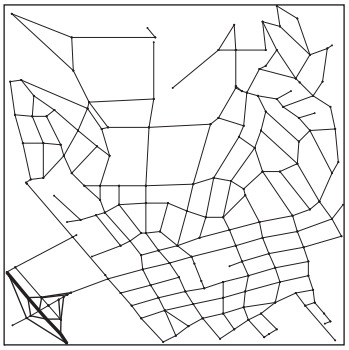


Fig. 10. Westwood area map used for the RT model.

7 Evaluation

In this section, we first describe the implementation details of the protocols that we consider for evaluation, and simulation setup in QualNet [36]. We then present the impact of disk access and computation overheads and then evaluate the proposed performance enhancement schemes.

7.1 Simulation Setup

We use IEEE 802.11b PHY/MAC with 11-Mbps data rate and real-track (RT) mobility model [32]. 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 Fig. 10, a street map of $2{,}400\ \mathrm{m} \times 2{,}400\ \mathrm{m}$ *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 percent 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, 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, 5 MB, 10 MB, 25 MB, and 50 MB. 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 20 KB. For the buffer replacement scheme, least recently used 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 s. The single hop pulling strategy is used to measure the performance of content distribution while excluding the impact of routing overheads.

We use the following hardware parameters to model disk access and computation overheads: a nominal hard disk of $R_d = 40\ \mathrm{MB/s}$, and a mobile device CPU of $R_e = \frac{48}{G}\ \mathrm{MB/s}$ (50 percent computing power of Intel Pentium 4 M 1.7 Ghz). We implement the E/S pipeline scheme for multithreading (see Fig. 3b): 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 $R_e = \frac{7.77}{G}\ \mathrm{MB/s}$. For this, we use the

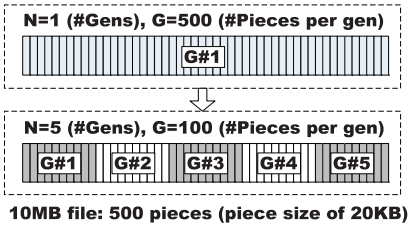


Fig. 11. Generation configuration example.

R/E/S pipeline for multithreading due to its fast random access capability of an SSD. We use two file sizes (5 MB and 10 MB) for Nokia N800.

The disk and coding delays are scheduled based on the disk access 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 percent confidence interval.

7.2 Simulation Results

Effects of disk access and computation overhead. 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 number of generations is the same as the total number of pieces for a given file. For instance, for a 10 MB file that has 500 20 KB pieces, if the number of generation is $N = 1$ and $N = 5$, the number of pieces per generation is 500 and 100, respectively, (see Fig. 11). The no coding case for a 10-MB file happens when each generation has a single piece ($N = 500$). The overall configuration is summarized in Table 2. 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 percent. Note that we can see the impact of "computational overhead" in the case of 100 percent buffer space, because a node can keep the entire file in the memory.

Fig. 12 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 Fig. 13, we show the case of buffer size $= 100$ percent 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., 50 M and 25 M). Yet it is still better than the *no coding* scenario where the generation size is 2,500 for a 50-MB file and 1,250 for a 25-MB file, and the corresponding encoding rates are 19.2 KB/s and 38.4 KB/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 25 MB and 50 MB files. For example, consider the plots of a 25 MB 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 10-MB file, it takes 1,133 s and 561 s 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.

TABLE 2
Network Coding Configuration with Various Numbers of Generations (N)

File Size	# Pieces	Number of pieces per gen (G)				No Coding
		N=1	N=5	N=10	N=50	
50MB	2500	2500	500	250	50	N=2500, G=1
25MB	1250	1250	250	125	25	N=1250, G=1
10MB	500	500	100	50	10	N=500, G=1
5MB	250	250	50	25	5	N=250, G=1

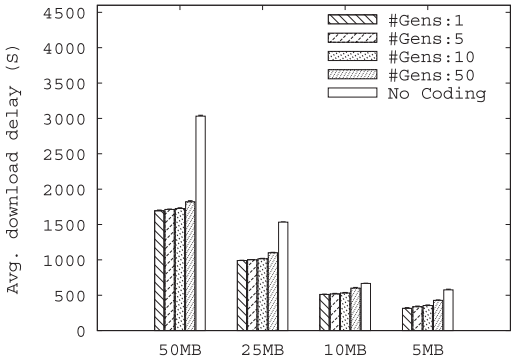


Fig. 12. Download delay without overhead (O/H).

Now consider the cases where the buffer size is smaller than the file size (see Figs. 14 and 15). The impacts of disk access can be clearly seen by comparing Figs. 14 and 15 with Fig. 13. 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. Figs. 16 and 17 show the download delay and the number of pieces read

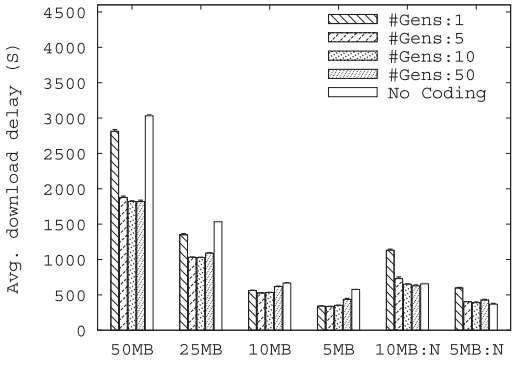


Fig. 13. Download delay with overhead: Buffer 100 percent (10 MB:N and 5 MB:N show the results of Nokia N800 configuration).

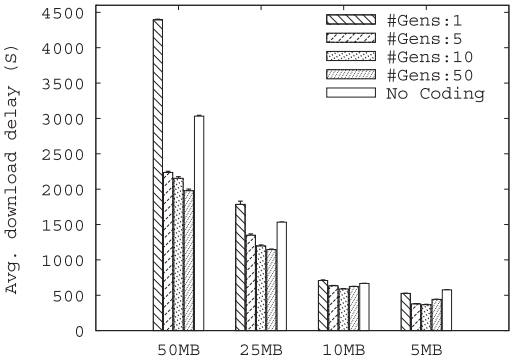


Fig. 14. Download delay with O/H: Buffer 75 percent.

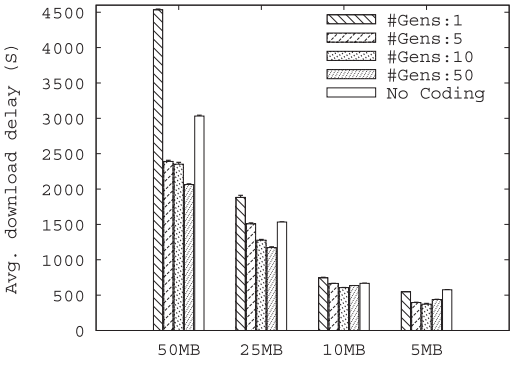


Fig. 15. Download delay with O/H: Buffer 50 percent.

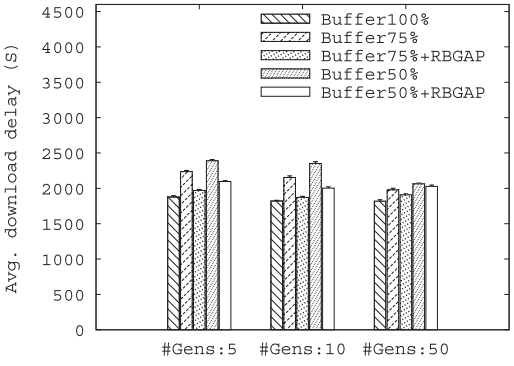


Fig. 16. Download delay with RBGAP (50-MB file).

from the disk with different buffer sizes, namely, 100, 75, and 50 percent. Note that the disk access 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 access, 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 percent increments. For instance, 25 and 50 percent coding density on a 50-MB file with $N = 1$ show that the maximum number of pieces used for encoding is 625 and 1,250 out of total 2,500 pieces, respectively. We simulate the following cases: a 50-MB file with $N = 1$ ($G = 2{,}500$) and a 25-MB file with $N = 1$ ($G = 1{,}250$). We use the buffer size of 100 percent (i.e., no buffer replacement overhead) to clearly see the benefits of sparse coding. Fig. 18 presents the results.

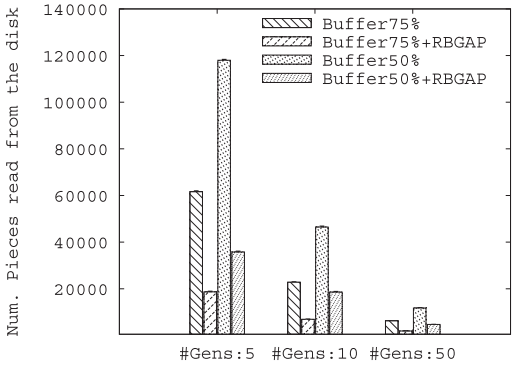


Fig. 17. Total number of pieces read from the disk (50-MB file).

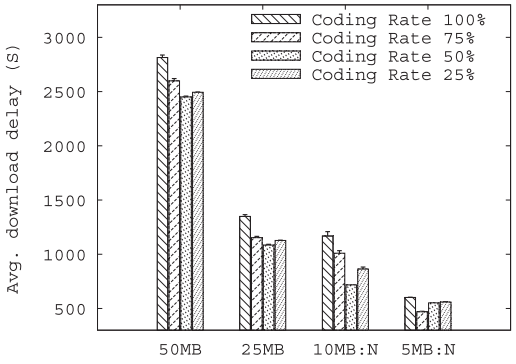


Fig. 18. Impact of sparse coding.

As the coding density decreases, the download delay also tends to decrease. For instance, when we lower the coding density to 75 percent, we observe a considerable delay reduction: from 2,814 s to 2,599 s for a 50-MB file and from 1,349 s to 1,153 s for a 25-MB 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.

Chunked code. 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 causes significant performance degradation. To show this impact more clearly, we evaluate the chunked code 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. Fig. 19 shows the results of 50 MB and 25 MB files with the number of generations $N = 1, 5, 10, 50$. The figure shows that chunked code 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 chunked code is less efficient in our scenario.

7.3 Discussion

In the paper, we mainly considered the encoding process since our goal is to find the maximum generation size that can fully utilize the bandwidth share—the overall delay for

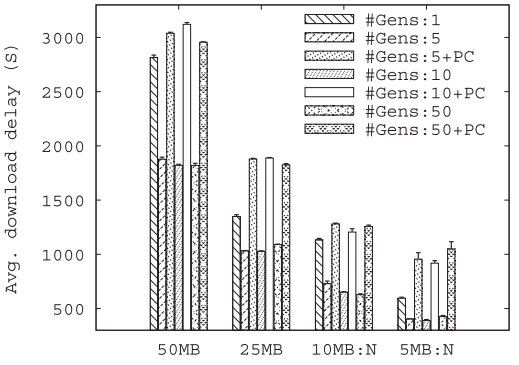


Fig. 19. Performance of chunked code.

downloading a file is dominated by the time for collecting linearly independent packets which is a slow process due to wireless resource limits and nodal mobility. Our approach can be easily extended to model the decoding process. Recall that the original pieces $\mathbf{p}$ can be recovered as $\mathbf{p} = \mathbf{C}^{-1}\mathbf{x}$ where $\mathbf{C}$ is a $G \times G$ matrix for encoding coefficients and $\mathbf{x}$ is a $G \times B$ matrix for coded data. Decoding basically uses Gaussian elimination that transforms a matrix to the reduced row-echelon form using elementary row operations, requiring $O(G \times GB)$ of multiplication and addition operations. To decode the entire file, i.e., $GB \times N$ symbols, it takes up to $G^2BN \cdot \delta$. From this, we find that the per symbol decoding rate is approximately equivalent to the encoding rate, namely $\frac{1}{\delta G}$. In our simulation settings, we used the encoding rates of 48/G MB/s and 7.77/G MB/s for ThinkPad and Nokia N800, respectively. The approximate decoding delay for ThinkPad and Nokia N800 is simply given as G $\times$ file size / 48 and G $\times$ file size / 7.7, respectively; for instance, decoding a 10-MB file takes about 10 s (ThinkPad) and 64 s (Nokia N800), which are far smaller than the overall download delay.

8 Related Work

In recent years, researchers performed various feasibility studies on network coding in real testbeds including smartphones, demonstrating that the measured performance actually varies widely depending on the system configurations [11], [27], [42], [39], [41]. For instance, Gkantsidis et al. [11] reported that their Avalanche scheme incurs little overhead in terms of CPU and access using a large scale testbed. On the other hand, it has been empirically observed that the computation overhead degrades the performance, especially when the generation size is large [42], [27].

Given that network coding overhead is a serious concern, researchers mostly focused on devising performance enhancement strategies which can be broadly categorized as hardware acceleration-based approaches [38], [35], [18] and efficient coding-based approaches [8], [27], [28], [30]. Hardware acceleration approaches include the use of special instruction sets and multiple cores. Shojania and Li [38] proposed to use SSE2 (Intel) and AltiVec (PowerPC) SIMD vector instructions and to parallelize coding operations over multiple CPU cores with multithreading. Given that heterogeneous workload assignment leads to poor performance, Park et al. [35] proposed balanced workload partitioning algorithms for parallelized network coding. Kim and Ro [18] demonstrated that hardware implementation of network coding using field programmable gate array (FPGA) can bring a major performance improvement over existing methods.

Also, researchers proposed various network coding strategies for computationally efficient content distribution [8], [27], [28], [30]. 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 [8], [27]. Maymounkov et al. [28] 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. Møller et al. proposed a systematic network coding method for content broadcasting that consists of two phases [30]; for a given generation, all pieces are first broadcast without coding and then coded pieces are broadcast. For clustered scenarios where users are mostly located within a single hop transmission range, the use of noncoded pieces in the first phase can significantly alleviate the coding overhead, but this approach is less suitable for highly mobile vehicular scenarios (i.e., only a small number of pieces can be delivered within a short contact period).

The main departure from existing work is that we model CPU and disk access overheads as a unified framework to better understand their impact on the network coding configuration, especially in the context of mobile wireless networking scenarios. This work mainly focuses on embedded devices that do not have special hardware for acceleration, but our model can be easily extended to consider hardware acceleration techniques. Note that researchers showed that symbol level network coding (as opposed to packet/piece level network coding) can improve the performance of content distribution if adequate wireless hardware/software capabilities are provided (e.g., symbol level encoding/decoding or partial packet recovery) [16], [45], [24], and our model can be easily extended to consider this approach. We leave these enhancements as a part of future work.

9 Conclusion

The main focus of this paper has been to investigate the impact of practical resource constraints of mobile devices (namely disk access, 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. 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 that can significantly reduce disk access 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 models and the data pulling scheme in the QualNet wireless network simulator. From the simulation study, we have obtained several new insights that will help improve 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 chunked code is not as efficient in a highly dynamic environment as VANETs.

ACKNOWLEDGMENTS

This research was sponsored in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (2012R1A1A1008858) and by the MSIP (Ministry of Science, ICT & Future Planning), Korea in the ICT R&D Program 2013. Also, research was sponsored in part by the US Army Research Laboratory and the UK Ministry of Defense and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies of the US Army Research Laboratory, the US Government, the UK Ministry of Defense, or the UK Government.

REFERENCES

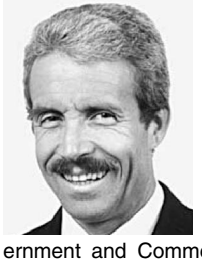
[1] R. Ahlswede, N. Cai, S.-Y.R. Li, and R.W. Yeung, "Network Information Flow," *IEEE Trans. Information Theory,* vol. 46, no. 4, pp. 1204-1216, July 2000.
[2] *Arm Architecture,* http://en.wikipedia.org/wiki/ARM_ architecture, 2013.
[3] P. Cao, E.W. Felten, and K. Li, "Application-Controlled File Caching Policies," *Proc. USENIX Summer Technical Conf. (USENIX '94),* June 1994.
[4] S. Chachulski, M. Jennings, S. Katti, and D. Katabi, "Trading Structure for Randomness in Wireless Opportunistic Routing," *Proc. ACM SIGCOMM '07,* Aug. 2007.
[5] D.M. Chiu, R.W. Yeung, J. Huang, and B. Fan, "Can Network Coding Help in P2P Networks?," *Proc. Second Workshop Network Coding (NetCod '06),* Apr. 2006.
[6] P.A. Chou, Y. Wu, and K. Jain, "Practical Network Coding," *Proc. Allerton Conf. Comm., Control, and Computing (Allerton '03),* Oct. 2003.
[7] M. Conti, E. Gregori, and G. Turi, "A Cross-Layer Optimization of Gnutella for Mobile Ad Hoc Networks," *Proc. ACM MobiHoc '05,* May 2005.
[8] C. Cooper, "On the Distribution of Rank of a Random Matrix over a Finite Field," *Random Structures and Algorithms,* vol. 17, nos. 3/4, pp. 197-221, 2000.
[9] S. Deb, M. Médard, and C. Chout, "Algebraic Gossip: A Network Coding Approach to Optimal Multiple Rumor Mongering," *IEEE/ ACM Trans. Networking* , vol. 14, no. SI, pp. 2486-2507, June 2006.
[10] F. Douglis, R. Cáceres, B. Marsh, F. Kaashoek, K. Li, and J. Tauber, "Storage Alternatives for Mobile Computers," *Proc. First USENIX Conf. Operating Systems Design and Implementation (USENIX '94),* Nov. 1994.
[11] C. Gkantsidis, J. Miller, and P. Rodriguez, "Comprehensive View of a Live Network Coding P2P System," *Proc. Sixth ACM SIGCOMM Conf. Internet Measurement (IMC '06),* Oct. 2006.
[12] C. Gkantsidis and P. Rodriguez, "Network Coding for Large Scale Content Distribution," *Proc. ACM INFOCOM '05,* Mar. 2005.
[13] C. Huang and L. Xu, "Fast Software Implementations of Finite Field Operations," technical report, Washington Univ. in St. Louis, Dec. 2003.
[14] D. Jiang, V. Taliwal, A. Meier, W. Holfelder, and R. Herrtwich, "Design of 5.9 GHz DSRC-Based Vehicular Safety Communication," *IEEE Wireless Comm.,* vol. 13, no. 5, pp. 36-43, Oct. 2006.
[15] M. Johnson, L.D. Nardis, and K. Ramchandran, "Collaborative Content Distribution for Vehicular Ad Hoc Networks," *Proc. Allerton Conf. Comm., Control, and Computing (Allerton '06),* Sept. 2006.
[16] S. Katti, D. Katabi, H. Balakrishnan, and M. Medard, "Symbol-Level Network Coding for Wireless Mesh Networks," *Proc. ACM SIGCOMM,* Aug. 2008.
[17] 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," *Proc. Fifth Workshop Embedded Systems for Real-Time Multimedia (ESTIMedia '07),* Oct. 2007.
[18] S. Kim and W.W. Ro, "Reconfigurable and Parallelized Network Coding Decoder for VANET," *Mobile Information Systems,* vol. 8, no. 1, pp. 45-59, 2012.
[19] A. Klemm, C. Lindemann, and O.P. Waldhorst, "A Special-Purpose Peer-to-Peer File Sharing System for Mobile Ad Hoc Networks," *Proc. IEEE 58th Vehicular Technology Conf. (VTC '03),* Oct. 2003.
[20] A. Lawrence, "Invalidating the Linux Buffer Cache," Linux Developer News, Jan. 2007.
[21] N. Leavitt, "Will Power Problems Curtail Processor Progress?," *Computer,* vol. 45, no. 5, pp. 15-17, May 2012.
[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," *Proc. IEEE Fifth Ann. Comm. Soc. Conf. Sensor, Mesh, and Ad Hoc Comm. and Networks (SECON '08),* June 2008.
[23] U. Lee, J.-S. Park, J. Yeh, G. Pau, and M. Gerla, "CodeTorrent: Content Distribution Using Network Coding in VANETs," *Proc. First Int'l Workshop Decentralized Resource Sharing in Mobile Computing and Networking (MobiShare '06),* Sept. 2006.
[24] M. Li, Z. Yang, and W. Lou, "CodeOn: Cooperative Popular Content Distribution for Vehicular Networks Using Symbol Level Network Coding," *IEEE J. Selected Areas in Comm.,* vol. 29, no. 1 pp. 223-235, Jan. 2011.
[25] J. Liu, D. Goeckel, and D. Towsley, "Bounds on the Gain of Network Coding and Broadcasting in Wireless Networks," *Proc. ACM INFOCOM '07,* May 2007.
[26] R. Love, *Linux System Programming.* O'Reilly, 2007.
[27] G. Ma, Y. Xu, M. Lin, and Y. Xuan, "A Content Distribution System Based on Sparse Linear Network Coding," *Proc. Second Workshop Network Coding (NetCod '07),* Mar. 2007.
[28] P. Maymounkov, N.J.A. Harvey, and D.S. Lun, "Methods for Efficient Network Coding," *Proc. Allerton Conf. Comm., Control, and Computing (Allerton '06),* Sept. 2006.
[29] P. Maymounkov and D. Mazieres, "Rateless Codes and Big Downloads," *Proc. Int'l Workshop Peer-to-Peer Systems (IPTPS '03),* Feb. 2003.
[30] J.H. Møller, M.V. Pedersen, F. Fitzek, and T. Larsen, "Network Coding for Mobile Devices - Systematic Binary Random Rateless Codes," *Proc. IEEE Int'l Conf. Comm. Workshops (ICC),* June 2009.
[31] A. Nandan, S. Das, M.Y. Sanadidi, and M. Gerla, "Cooperative Downloading in Vehicular Ad Hoc Wireless Networks," *Proc. Second Ann. Conf. Wireless On-Demand Network Systems and Services (WONS '05),* Jan. 2005.
[32] A. Nandan, S. Das, S. Tewari, M. Gerla, and L. Klienrock, "AdTorrent: Delivering Location Cognizant Advertisements to Car Networks," *Proc. Ann. Conf. Wireless On-Demand Network Systems and Services (WONS '06),* Jan. 2006.
[33] M. Neufeld, J. Fifield, C. Doerr, A. Sheth, and D. Grunwald, "SoftMAC—Flexible Wireless Research Platform," *Proc. Fourth Workshop Hot Topics in Networks (HotNets-IV),* Nov. 2005.
[34] J.-S. Park, M. Gerla, D.S. Lun, Y. Yi, and M. Médard, "CodeCast: A Network-Coding-Based Ad Hoc Multicast Protocol," *IEEE Wireless Comm.,* vol. 13, no. 5, pp. 76-81, Oct. 2006.

[35] K. Park, J.-S. Park, and W.W. Ro, "On Improving Parallelized Network Coding with Dynamic Partitioning," *IEEE Trans. Parallel and Distributed Systems,* vol. 21, no. 11, pp. 1547-1560, Nov. 2010.
[36] *Scalable Networks,* http://www.scalable-networks.com, 2013.
[37] P. Samar and S.B. Wicker, "On the Behavior of Communication Links of a Node in a Multi-Hop Mobile Environment," *Proc. ACM MobiHoc '04,* May 2004.
[38] H. Shojania and B. Li, "Parallelized Progressive Network Coding with Hardware Acceleration," *Proc. IEEE 15th Int'l Workshop Quality of Service (IWQoS '07),* Sept. 2007.
[39] H. Shojania and B. Li, "Random Network Coding on the iPhone: Fact or Fiction?," *Proc. 18th Int'l Workshop Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '09),* Sept. 2009.
[40] *Erster Benchmark Von Intels Silverthorne,* http://www.computerbase.de/news/hardware/prozessoren/intel/2008/maerz/erster_benchmark_intels_silverthorne, 2013.
[41] P. Vingelmann, F.H.P. Fitzek, M.V. Pedersen, J. Heide, and H. Charaf, "Synchronized Multimedia Streaming on the iPhone Platform with Network Coding," *IEEE Comm. Magazine,* vol. 49, no. 6, pp. 126-132, June 2011.
[42] M. Wang and B. Li, "How Practical Is Network Coding?," *Proc. IEEE 14th Int'l Workshop Quality of Service (IWQoS '06),* June 2006.
[43] J. Widmer and J.-Y.L. Boudec, "Network Coding for Efficient Communication in Extreme Networks," *Proc. (CHANTS '05),* Aug. 2005.
[44] F. Wu, H. Xi, J. Li, and N. Zou, "Linux Readahead: Less Tricks for More," *Proc. LINUX Symp.,* June 2007.
[45] Z. Yang, M. Li, and W. Lou, "CodePlay: Live Multimedia Streaming in VANETs Using Symbol-Level Network Coding," *Proc. IEEE 18th Int'l Conf. Network Protocols (ICNP '10),* Oct. 2010.

Uichin Lee received the BS degree in computer engineering from Chonbuk National University in 2001, the MS degree in computer science from Korea Advanced Institute of Science and Technology (KAIST) in 2003, and the PhD degree in computer science from the University of California at Los Angeles in 2008. He is an associate professor in the Department of Knowledge Service Engineering at KAIST. Before joining KAIST, he was a member of technical staff at Bell Labs, Alcatel-Lucent until 2010. His research interests include social computing systems and mobile/pervasive computing systems. He is a member of the IEEE.

Seung-Hoon Lee received the BS degree in computer science from the Illinois Institute of Technology in 2006, the MS degree in computer science from the University of California, Los Angeles (UCLA) in 2007, and the PhD degree in computer science from the UCLA in 2011. His research interests include mobile wireless networks, vehicular ad hoc networks (VANET), and network coding. He has published papers in the fields of content distribution in VANETs, inter-domain routing, and network coding. He is a member of the IEEE.

Kang-Won Lee received the PhD degree in computer science from UIUC and the MS and BS degrees in computer engineering from SNU. He is a Research Staff Member and Manager of Wireless Networking Department at the IBM T.J. Watson Research Center. From 2006 until 2011, he was an Industrial technical leader for the US-UK ITA program (total funding level $70M over 10 years). He is currently leading tech transition efforts of ITA, a critical requirement for the second phase of the project. He has developed significant technologies in the space of policy management (IBM TPC SAN configuration checker, Tivoli Netcool IMPACT filter analysis, PMAC policy middleware) and wireless networking (Inter-domain Routing for MANETs). He has been elected an IBM Master Inventor for his contribution to patent folio and creating high value patents. He has published more than 80 technical articles in leading IEEE/ACM conferences and journals. He is a distinguished scientist of ACM and a senior member of IEEE.



Mario Gerla received the engineering degree from Politecnico di Milano, Italy, and the PhD degree from the University of California, Los Angeles (UCLA). He is a professor in the computer science at the UCLA. At UCLA, he was part of the team that developed the early ARPANET protocols under the guidance of Prof. Leonard Kleinrock. At Network Analysis Corporation, New York, from 1973 to 1976, he helped transfer ARPANET technology to Government and Commercial Networks. He joined the UCLA Faculty in 1976. At UCLA, he has designed and implemented network protocols including ad hoc wireless clustering, multicast (ODMRP and CodeCast) and Internet transport (TCP Westwood). He has lead the $12M, 6 year ONR MINUTEMAN project, designing the next generation scalable airborne Internet for tactical and homeland defense scenarios. He is now leading two advanced wireless network projects under ARMY and IBM funding. His team is developing a Vehicular Testbed for safe navigation, urban sensing and intelligent transport. A parallel research activity explores personal communications for cooperative, networked medical monitoring (see http://www.cs.ucla.edu/NRL for recent publications). He is a fellow of the IEEE.