Echelon: Peer-to-Peer Network Diagnosis with Network Coding

Chuan Wu, Baochun Li

***Abstract*— It is critical to monitor the performance and "health" of large-scale peer-to-peer applications. As an example, operators of peer-to-peer live streaming applications may be interested in observing performance bottlenecks, peer failures, and network topologies. In most cases, such observations are used to diagnose potential problems in the protocol design, to troubleshoot network outage, or to improve the Quality of Service of the peer-to-peer network in general. They are *not* time sensitive in nature, as delayed observations up to minutes or even hours are still valuable. However, such *historical* and delay-tolerant observations should include measurements of peers that have already failed or departed, as peer dynamics significantly affect the health of peer-to-peer applications. Such a delay-tolerant observation of peer-to-peer applications over a historical period of time is referred to as a *diagnosis*. In this paper, we present *Echelon*, a time-insensitive way to construct the *diagnosis* of a large-scale peer-to-peer application. Replacing the traditional wisdom of logging servers, we leverage the power of network coding to collect application-specific measurements on each peer, and disseminate them to other peers in a coded form. Over time, measurements of departed peers can still be recovered, simply by probing a small subset of peers in the network. Simulation studies have shown that *Echelon* is highly configurable, bandwidth efficient, and extremely tolerant of peer dynamics, thanks to the advantages of randomized network coding.**

I. Introduction

Peer-to-peer (P2P) architectures have been shown to offer high performance, better scalability, as well as superb resilience to peer failures and departures. It has been increasingly natural to design Internet applications using the peer-to-peer architecture, including bulk content distribution (*e.g.,* BitTorrent) and live media streaming (*e.g.,* IP Television). In such peer-to-peer applications, it is critical for operators to monitor the performance and "health" of live peer-to-peer sessions.

Any such monitoring starts with measurements of Quality of Service parameters at each peer. The set of parameters to be measured is certainly application specific. For example, in live media streaming applications, it is essential to monitor the achieved streaming rate, the number of upstream and downstream peers, the latency to neighbor peers, and resource usage such as bandwidth and CPU load. If bandwidth limits have been imposed by either Internet Service Providers or application-layer traffic shapers, it is ideal to also monitor residual bandwidth. In peer-to-peer online gaming applications, latency to a small set of gaming servers may be critical, as they dramatically affect the gaming experience. As these parameters are measured periodically (for example, every minute), the set of measurements in one time interval is hereafter referred to as a *snapshot* of the peer.

Chuan Wu and Baochun Li are affiliated with the Department of Electrical and Computer Engineering, University of Toronto. Their email addresses are {*chuanwu,bli*}*@eecg.toronto.edu*.

Operators of live peer-to-peer sessions are interested in system-wide characteristics of the entire peer-to-peer network, by collecting and aggregating per-peer snapshots. Such characteristics may include network topologies, regional performance bottlenecks, and patterns of peer departures or failures. In most cases, such observations are used to diagnose potential problems in algorithm design (such as convergence of optimization strategies), to troubleshoot network outage, or to improve the Quality of Service of the peer-to-peer session in general. For long-running peer-to-peer applications, most of such observations are *not* time sensitive in nature, as delayed collections of snapshots — up to minutes or even hours — are still valuable.

One specific requirement, however, is the ability to collect snapshots from peers that no longer exist at the time of collection, including those who have left the session or failed. Historical snapshots on these peers are particularly valuable, as peer dynamics is one of the most important characteristics that significantly affect the health of peer-to-peer sessions. Simply put, the operators may be interested in reconstructing a "postmortem report" of the peer-to-peer session, by collecting peer snapshots over a historical period of time, including all the peers that have already departed. For lack of a better word, in this paper, such a "postmortem report" is referred to as a *diagnosis* of the peer-to-peer session in question.

The traditional wisdom to collect peer snapshots is to rely on peers sending periodic reports to a *logging server*. While dedicated servers in peer-to-peer sessions are commonplace in real-world peer-to-peer applications, most are used for low-bandwidth one-time communication with participating peers, such as an authentication server to bootstrap a new peer. Periodic snapshots represent much higher traffic volume, especially when the number of peers in the session increases dramatically. In large-scale peer-to-peer applications, such periodic reporting essentially morphs into a *de facto* Distributed Denial of Service attack to the logging server. It is, without a doubt, not a scalable design. Due to such challenges of deploying centralized logging servers, operators resort to traditional remedies of either decreasing the frequency of obtaining snapshots, or reducing the amount of data to be reported in each snapshot.

In this paper, we present *Echelon*, our answer to the aforementioned challenges. The primary design objectives of *Echelon* are to be able to scale to large-scale peer-to-peer sessions, and to tolerate extreme levels of peer dynamics. In other words, peers may come and go, but the *diagnosis* of the network persists, regardless of the scale of the peer-to-peer session. In *Echelon*, we use a peer-to-peer overlay to disseminate snapshots produced at each peer, and cache them in a completely decentralized fashion in the peer-to-peer session. Due to the delay-tolerant nature of constructing

the diagnosis, such a peer-to-peer overlay is lightweight, and carries traffic at low bit rates using residual peer bandwidth. In order to be resilient to extreme peer dynamics and to include snapshots of peers that have already departed, we leverage the power of *randomized network coding*, such that only coded snapshots are exchanged among peers and cached. At the time of constructing a diagnosis, the operator only needs to probe an arbitrary and small subset of live peers in the session. The power of network coding effectively guarantees that *Echelon* is highly resilient to peer departures and failures, and is flexible, configurable, and simple to implement.

The remainder of this paper is organized as follows. In Sec. II, we motivate our design of *Echelon* and the use of network coding. In Sec. III, the complete spectrum of *Echelon* is formally presented. In Sec. IV, we show how *Echelon* may be refined and configured, with respect to its bandwidth usage and resilience to peer departures. An empirical study of *Echelon* based on simulations is presented in Sec. V. We discuss related work and conclude the paper with a future outlook in Sec. VI and Sec. VII, respectively.

II. Echelon: A Different Way to Use Network Coding

In this paper, we consider a large-scale peer-to-peer network with n peers. Each peer may participate in one or more peer-to-peer communication sessions. Peers may participate and leave the sessions at any time. In each session, a mesh overlay topology exists at any given time, defined by the selection of *neighbor peers*. Without loss of generality, we assume k out of a set ($\mathcal{N}$) of n peers periodically collect local application-specific measurements, called *snapshots*. As snapshots are captured periodically, the time interval between two successive snapshots is referred to as an *epoch*, with a length T. The peers that produce periodic snapshots are called *snapshot peers*, and form a set $\mathcal{S}$. We assume that all original snapshots captured at peers are of the same byte length L, sometimes also referred to as original data blocks. There exists a *snapshot collector*, C, who is responsible for collecting snapshots from peers, and for performing postmortem analysis and reaching the diagnosis on the "health" of the peer-to-peer sessions. An example for such a peer-to-peer network is illustrated in Fig. 1.

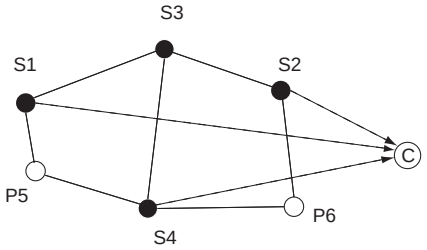

Fig. 1. Collecting snapshots: an example in a 6-peer network with 4 snapshot peers S_1, S_2, S_3 and S_4.

Our design objective in *Echelon* is to efficiently collect k original snapshots in each epoch, by probing just an arbitrary and small subset of the peers in the network. Such aggregation should always be successful, even when some of the snapshot peers are no longer in the network at the time of collection. In order to aggregate all snapshots from only a small number of peers, we leverage the storage capacities on each peer, and ask each peer to cache a number of data blocks generated by other peers. In order to be resilient to peer dynamics, we utilize the residual bandwidth of the peers to disseminate k data blocks to all n peers in the network in a coded form. To achieve bandwidth-efficient data dissemination and fully decentralized caching of each data block, we leverage the power of *randomized network coding* to exchange and store coded snapshot data blocks in the peer-to-peer network.

Network coding allows the encoding of received data blocks at intermediate nodes [1], [2], [3]. With *randomized* network coding [4], a node generates a new coded block by the linear combination of its received blocks (and possibly its own original blocks) over a Galois field $GF(2^q)$, with coefficients randomly chosen. In this way, every coded block in the network can be regarded as a linear combination of k original blocks. Decoding is performed by choosing k coded blocks with linearly independent coefficient vectors and inverting the combined coefficient matrix over $GF(2^q)$.

Network coding was first proposed to improve multicast session throughput. We believe, however, that the most important advantage of randomized network coding is to increase the *diversity* of data blocks, and to improve resilience against failures. By choosing coding coefficients randomly from a Galois field of a proper size, the coefficient vectors of any k coded blocks are linearly independent with high probability, and thus can be used to recover the k original blocks. Even after original data sources have departed, there is still a good chance for all the original blocks to be fully recoverable.

In *Echelon*, during each epoch, a snapshot peer sends out its original snapshot as an original data block to its neighbor peers. Its neighbors then code their received and original blocks (if they are snapshot peers themselves) belonging to the same epoch with a random linear code, and further distribute these new coded blocks. Fig. 2(A) illustrates a simple example of such *multi-source* network coding in a directed network, based on the network example given in Fig. 1. Four original data blocks, a, b, c, d, are generated at snapshot peers S_1, S_2, S_3, and S_4, respectively.

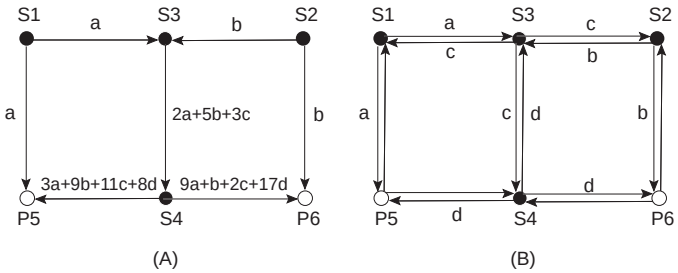

Fig. 2. Multi-source network coding: (A) an example in a direct network; (B) an example in a cyclic network.

The way we use network coding in *Echelon* is substantially different from most previous work. First of all, the dissemination of original snapshots (original data blocks) is time insensitive in nature, such that coded blocks may be sent using residual peer bandwidth. Second, our goal is not to broadcast all k original data blocks from their sources to all n peers

in the network, but to disseminate every original block to all n peers in its *coded form*. Therefore, cached coded blocks at each peer are linear combinations of all original blocks, but the number of coded blocks cached at a peer is dependent on its allocated cache capacity, usually much smaller than k. It is the snapshot collector's responsibility to collect k coded blocks from a subset of peers and to recover all original snapshots in an epoch.

We assume every peer caches coded blocks for E epochs, as the snapshot collector may be interested in collecting snapshots in a recent epoch. Let M_i be the total cache capacity allocated at peer i in terms of the number of coded blocks, and m_i be the number of blocks it caches for each epoch, $1 \leq m_i \leq k$. We have $M_i = E \cdot m_i$. Such a snapshot cache is implemented as a circular buffer, where the blocks for the oldest epoch are discarded when the buffer is full.

To implement randomized network coding in practical peer-to-peer networks, we need to consider bi-directional overlay links, and topologies with cycles. The convenience in the case of acyclic network topologies may no longer hold. For example, in Fig. 2(A), a peer codes and relays new coded blocks to its downstream peers when all the blocks from its incoming links have arrived. This is not possible in network topologies with cycles, *e.g.*, Fig. 2(B), since deadlocks occur as soon as peers are waiting for all incoming blocks from each other. We address this problem in subsequent sections in this paper.

III. Echelon: an In-Depth View

In this section, we present the *Echelon* protocol, which utilizes network coding in a novel and practical way. We limit our discussion to the snapshot dissemination protocol executed in one epoch, as it is trivial to extend it to more than one epoch.

Before going into the details of *Echelon*, let us revisit the practical network coding problem in Fig. 2(B). One possible solution for network coding in such cyclic networks is to ask each peer to encode currently available incoming blocks, after waiting for a certain period of time. However, we argue that even though this simple solution breaks deadlocks using a timeout mechanism, it does not completely solve the problem to our satisfaction. In *Echelon*, original data blocks are produced at different peers. If each peer only encodes incoming blocks (and sends coded blocks to downstream peers) *once*, cached blocks at a peer may only contain original blocks produced at nearby snapshot peers. We have not achieved the objective of disseminating the original snapshots over the entire network.

Echelon represents an *iterative* network coding approach, in that coded data dissemination in each epoch is implemented in a fully distributed and iterative fashion. In Echelon, randomized network coding at each peer is further divided into multiple *time slots* of length t, with $t \ll T$, the length of an epoch. In each time slot, a peer codes its cached blocks received in the previous time slots, and sends generated blocks to its neighbor peers. It then waits for time t before it codes and distributes snapshots again. We refer to the protocol execution in a time slot as a *round* of execution. In what follows, we first present the format of coded data messages in *Echelon*, and then discuss the detailed protocol for its coded dissemination.

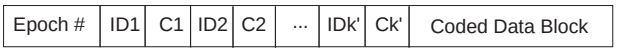

Fig. 3. Data message format: ID - block identifier, C - coding coefficient

A. Data message format

In traditional network coding implementation, k coding coefficients are delivered together with each coded data block, arranged in the order of their corresponding original blocks. However, in *Echelon*, original data blocks are produced in a completely distributed fashion, and thus represent no natural ordering among themselves. In addition, the total number of original blocks, k, is continuously changing with peer joins and departures, and is unknown to the peers in the network.

In *Echelon*, we design a new data message format to address the above problems. As each snapshot peer generates one snapshot in each epoch, we associate an identifier (ID) with each block, which is a r-bit integer ($2^r > k$) obtained by hashing its generator peer's IP address with a base hash function. We also generate an e-bit epoch number to represent each epoch, by hashing the starting time of the epoch. The block ID and the epoch number can be used to identify each original data block. In a coded data message, we include the epoch number and IDs of the original blocks from which the coded block is generated, together with the coding coefficients. An illustration of the message format is shown in Fig. 3. Note that we only include coefficients and IDs of the original blocks that are used to generate a coded block, and therefore, the byte length of the coefficient part (i.e., coding coefficients and corresponding block IDs) of different data messages varies.

B. Coded dissemination

At the beginning of an epoch, each snapshot peer collects its local measurements and generates an original snapshot. For example, in Fig. 4, at time 0, snapshot peers S_1, S_2, S_3 and S_4 produce and cache blocks a, b, c and d, respectively. Next, each snapshot peer sends its original snapshot to its neighbors. Therefore, in the first time slot $\bar{t} = 1$, every peer receives the original data blocks its snapshot neighbor peers produce. In Fig. 4(B), for example, S_3 obtains blocks a, b, d from its three neighbors S_1, S_2 and S_4, respectively.

In each of the following time slots $\bar{t} = 2, 3, \ldots$, a pull-based coded dissemination mechanism is employed based on block advertisement. Let $\widehat{D}_i(\bar{t})$ be the set of data blocks peer i is caching at the beginning of time slot $\bar{t}$. When peer i receives a coded data message, it retrieves the IDs of included original blocks, and adds them into a set $\widehat{B}_i$. Therefore, $\widehat{B}_i(\bar{t})$ includes IDs of the original blocks that are contained in any of the coded blocks in $\widehat{D}_i(\bar{t})$. If the ID of an original block b is included in $\widehat{B}_i(\bar{t})$, we say peer i *knows* or *has* block b by the $\bar{t}^{\text{th}}$ time slot.

The protocol executed in time slot $\bar{t}$ follows the following steps:

Step 1 **– *Advertise.*** Peer i sends an advertisement message to each of its neighbor peers, containing IDs of the original

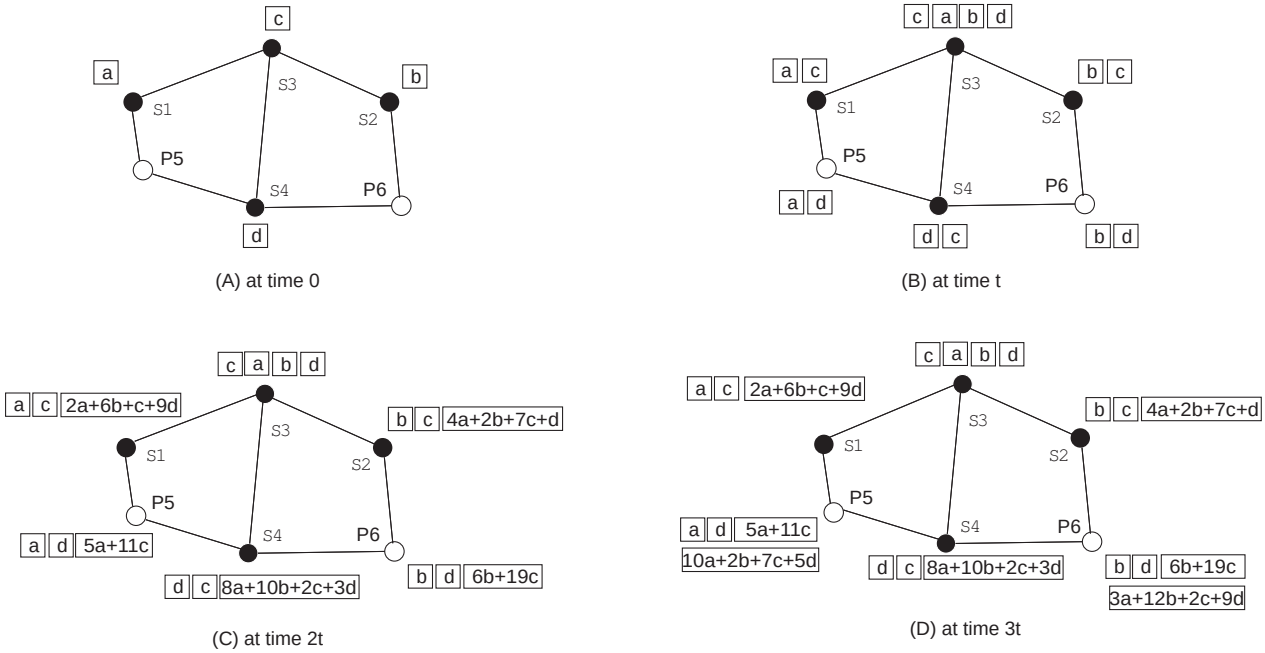

Fig. 4. Coded dissemination protocol: an example.

data blocks that it has newly learned from the coded blocks received in time slot $\bar{t}-1$, *i.e.*, any IDs in the set of $\widehat{B}_i(\bar{t}) - \widehat{B}_i(\bar{t}-1)$. If there is no such ID, peer i does not send any advertisement messages.

Step 2 **–** ***Request.*** At a receiver peer j, after retrieving the block IDs from a received advertisement message, it checks whether its neighbor i knows any original block that it has never heard of, *i.e.*, whose ID is not in $\widehat{B}_j(\bar{t})$. If there is such a block, peer j requests a coded block from peer i.

Step 3 **–** ***Code and Deliver.*** Upon the request from neighbor peer j, peer i generates a coded block by the linear combination of all its currently cached coded blocks in $\widehat{D}_i(\bar{t})$. Together with the epoch number, coding coefficients and corresponding original block IDs, it delivers the message to peer j.

Step 4 **–** ***Cache.*** After peer j receives a coded data message, if its cache is not full yet, it places the coded block directly into the cache; otherwise, it randomly selects one coded block in its cache and codes the received block with it with a random linear code.

The execution of the above iterative protocol at each peer naturally terminates when it no longer receives advertisement messages containing IDs of original data blocks that it does not know, *i.e.*, when each peer has virtually received all the original blocks in the current network in coded form.

Continuing with the example in Fig. 4, we illustrate the execution of the protocol. Each peer can cache up to 4 coded blocks per epoch. At the beginning of the second time slot shown in Fig. 4(B), for the example of peer S_1, its set $\widehat{B}_1$ contains IDs of original blocks a and c, among which c is newly learned in the previous time slot. S_1 sends an advertisement message containing c's ID to S_3 and P_5, respectively. After P_5 receives the advertisement message from S_3, it discovers that block c is new to itself, so it obtains a coded block $5a+11c$ from S_3. All the other peers follow the same protocol. After this round of dissemination, Fig. 4(C) illustrates the cache contents at the beginning of the third time slot. Similarly, in the third time slot, S_1 advertises the IDs of newly acquired blocks b and d to its neighbors. After P_5 identifies that b is new, it acquires a coded block $10a+2b+7c+5d$ from S_1. After this round, all peers hold all four original blocks in coded form, and the protocol execution ends in 3 rounds.

As we can see from the protocol, peer j requests a coded block from peer i only if the latter knows some original blocks that itself does not have yet. In this way, peer j is guaranteed to obtain *innovative* coded blocks for itself, with coding coefficient vectors linearly independent to those of its cached blocks. Therefore, (a) if peer j has available space in its cache and directly stores the received block, the rank of its coefficient matrix is increased; (b) otherwise, by coding the received block with one of the existing blocks, it will also increase the rank of the combined coefficient matrix, which the snapshot collector subsequently constructs during the decoding process. When coded dissemination terminates, each peer caches coded blocks that are linear combinations of virtually all the original blocks in the current network. As the coding coefficients are randomly chosen from $GF(2^q)$ ($q=16$ is enough in most cases), there is high probability for the combined coefficient matrix from any k or slightly more than k coded blocks in the network to have the rank of k, which can thus be used to decode k original blocks.

We make two additional remarks about the *Echelon* protocol:

First, the iterative protocol execution at each peer does not need to be carefully synchronized. While peers may be designated to capture snapshots at the same time of a day, *e.g.*, every minute/hour, they may actually collect measurements at different times due to clock difference. Furthermore, *Echelon* message transmissions may be delayed for various lengths of time, since only residual capacities at the peers are utilized. In *Echelon*, as long as each peer captures snapshots at the

same moment based on their local clocks and the clocks are roughly synchronized (*e.g.*, using the Network Time Protocol), the generated epoch numbers are unified, and snapshots of the same epoch can be correctly coded together. As long as every peer executes the protocol periodically with a period t, the iterative protocol in each epoch is bound to terminate successfully. *Time slots* are just introduced in the protocol description for better understanding and to facilitate analysis.

Second, *Echelon* provides excellent resilience to peer dynamics in collecting the network diagnosis. When a new snapshot peer joins the network, it can immediately participate in the coded dissemination process and disseminate its local measurements across the network. In the case of snapshot peer departures or failures, as long as the peer has sent out its captured snapshots and the network is still connected (which is generally guaranteed in practice by having isolated peers locate and connect to new neighbors), its snapshots are still able to propagate throughout the entire network in coded form. Therefore, they can still be successfully recovered by the snapshot collector.

The pseudocode that implements the *Echelon* protocol is summarized in Table I.

C. Overhead analysis

We now analyze the messaging and computational overhead in the coded dissemination process of each epoch. To simplify our analysis, we assume that all snapshot peers capture snapshots and start protocol execution at the same time.

First, we discuss the number of rounds the protocol executes before it terminates in each epoch. The number is decided by the maximum number of hops for an original data block to be distributed in its coded form, from the original snapshot peer to the farthest peer. This is actually the diameter of the network, usually $O(\ln n)$ for realistic richly connected peer-to-peer networks. For example, in Fig. 4, it maximally takes three hops for block b to reach peer P_5, and thus the protocol terminates in 3 rounds.

Let d be the average number of neighbors each peer has in a network. The total number of advertisement messages sent in each time slot across the entire network is at most $d \cdot n$. Therefore, the overall number of advertisement messages in each epoch is $O(n \ln n)$, and the number of coded data messages delivered in each epoch is at most $O(n \ln n)$ as well.

The coding coefficient overhead per data message depends on the number of original blocks included, the size of the Galois field (2^q), and the number of bits used for a block ID (r). In the extreme case that a coded block is the linear combination of all the original blocks, its coefficient overhead is $k(r+q)$ bits, where $r \approx \log_2 k$. When k scales up to thousands, such coefficient overhead is still less than a few KBytes, which is usually much smaller than the data size of snapshots.

Next, we analyze the computational overhead and coding delay at each peer, introduced by network coding. Such overhead is mainly determined by the size of each data block, and the number of coded blocks a peer combines to produce a new block. The former is application-specific, decided by individual diagnosis requirement. In *Echelon*, the number of coded blocks used in network coding is upper-bounded by the cache capacity at each peer, usually much smaller than k. Therefore, the computational overhead is limited. Since the snapshot collector aggregates snapshots and diagnoses peer-to-peer sessions in a "postmortem" way, the coding delay is not significant as well.

Finally, we discuss the number of peers the snapshot collector needs to probe to obtain k or slightly more than k coded blocks for decoding. It relies on the number of resulting coded blocks cached at each peer, which is decided by the number of rounds the protocol executes, the number of neighbors each peer has, and is upper-bounded by the cache capacity at a peer.

Based on the above analysis, we wish to further refine our protocol to reduce the communication and computational overhead of the protocol, while still guaranteeing large-scale dissemination of each original snapshot in coded form. In what follows, we investigate possible refinements to our *baseline Echelon* protocol presented in this section, and make our system highly configurable and flexible.

IV. Refining and Configuring Echelon

A. Refining the Advertising Step

In order to reduce the coded data traffic in the network, we refine step 1 in our baseline protocol, and disseminate the coded messages in a gossip-like manner. In each time slot, instead of advertising original blocks to all its neighbor peers, a peer randomly selects a subset of its neighbors and forwards the advertisement message to them. Let *NumNeighbor* be the maximum number of neighbors selected at each peer. Step 1 can be refined as follows:

Step 1 – *Advertise*. Peer i sends advertisement messages to *NumNeighbor* neighbors, selected uniformly at random from all its neighbor peers. The advertisement message to neighbor j contains IDs of all the original blocks which peer i has never advertised to peer j.

With the refined protocol, as we may not wish to disseminate every original block to every peer eventually, we introduce another parameter *MaxRound*. Instead of naturally terminating as in the baseline protocol, the refined protocol executed at a peer stops when the maximum number of rounds specified by *MaxRound* has been reached.

If *MaxRound* is set to a large value, the total number of messages involved in protocol execution may not be reduced eventually. Therefore, to actually reduce the bandwidth usage, *MaxRound* should be set to numbers at the same magnitude of that executed by the baseline protocol, $O(\ln n)$, or even smaller. We note that even in this way, each original block is still largely distributed over the network in coded form, which can be illustrated as follows: at peer i, if a neighbor peer j is not selected during the previous time slots but is chosen in time slot $\bar{t}$, the IDs of all peer i's known original data blocks are advertised to peer j, and a coded block combining all these original blocks is sent to j in this round.

Besides *MaxRound*, distribution of the original blocks also depends on the value of *NumNeighbor:* the more neighbors

TABLE I

Echelon PSEUDOCODE EXECUTED AT PEER i IN TIME SLOTS $\bar{t} = 1, 2, \ldots$

Notations:

$\widehat{D}_i$: the set of coded blocks currently cached at peer i
m_i: the cache capacity at peer i
$\widehat{B}_i$: the ID set of the original blocks which peer i knows
$\widehat{B'}_i$: the ID set of the original blocks newly learned in the previous time slot
Coeffs: combined coefficient matrix of coded blocks in $\widehat{D}_i$
Coeffs[v]: coefficient vector of the v^{th} coded block in $\widehat{D}_i$
Neighbors: the set of neighbor peers of peer i

Advertise block information:

1 **for** $u = 1$ to *Neighbors.size*
2 packetize $\widehat{B'}_i$ and send to *Neighbors*[u]
3 **end for**
4 $\widehat{B'}_i \leftarrow \phi$

Upon receiving an advertisement message from peer j:

1 $I \leftarrow$ original block IDs retrieved from the message
2 **if** there exists block ID *I*[u] $\notin \widehat{B}_i$, $\forall u = 1, \ldots,$*I.size*
3 send request for a coded block to peer j
4 **end if**

Upon receiving a request message from peer j:

1 $R \leftarrow \phi$
2 **for** $u = 1$ to $\widehat{D}_i$*.size*
3 $R[u] \leftarrow$a randomly chosen coefficient from $GF(2^q)$
4 **end for**
5 new coded block b = *multiplyVector*(R,$\widehat{D}_i$)
6 coefficient vector C = *multiplyMatrix*(R, *Coeffs*)
7 packetize C, corresponding original block IDs, b, and send the message to peer j

Upon receiving a coded data message:

1 $C' \leftarrow$ coefficient vector retrieved from the message
2 $I' \leftarrow$ original block IDs retrieved from the message
3 $b' \leftarrow$ coded block contained in the message
4 **for** $u = 1$ to I'*.size*
5 **if** $I'[u] \notin \widehat{B}_i$
6 $\widehat{B}_i \leftarrow \widehat{B}_i \cup I'[u]$
7 $\widehat{B'}_i \leftarrow \widehat{B'}_i \cup I'[u]$
8 **end if**
9 **end for**
10 **if** $\widehat{D}_i$*.size* $< m_i$
11 $\widehat{D}_i \leftarrow \widehat{D}_i \cup b'$
12 *Coeffs*[$\widehat{D}_i$*.size*]$\leftarrow C'$
13 **else**
14 $u \leftarrow$index of a randomly chosen block in $\widehat{D}_i$
15 $R' \leftarrow$ a size-2 vector of randomly generated coefficients from $GF(2^q)$
16 $\bar{b} \leftarrow (b', \widehat{D}_i[u])$
17 $\bar{C} \leftarrow$[C', *Coeffs*[u]]
18 new coded block $b'' =$ *multiplyVector*(R',$\bar{b}$)
19 new coefficient vector $C'' =$ *multiplyMatrix*(R',$\bar{C}$)
20 $\widehat{D}_i[u]$=b''
21 *Coeffs*[u]$\leftarrow C''$
22 **end if**

each peer is sending data messages to, the more peers each original data block is distributed onto in a coded form. When more peers in the network hold an original data block, the chances for its recovery in case of peer failures are higher.

To summarize, by configuring different values for *Num-Neighbor* and *MaxRound*, we can achieve different tradeoffs between messaging overhead and failure tolerance using the refined *Echelon* protocol.

B. Refining the Encoding Step

In order to reduce the coefficient overhead in the coded data messages, we aim to reduce the number of original blocks included in the linear combinations of the coded blocks. For this purpose, we refine the encoding step in our baseline protocol. At each peer, other than combining all the currently cached coded blocks to produce a new coded block, it now codes only those that contain the original blocks the neighbor is seeking. Step 2 and step 3 in our baseline protocol are modified in this regard:

Step 2 – *Request.* When peer j asks for a coded block from peer i, it sends a request *containing IDs of the original blocks that it is seeking from peer* i.

Step 3 – *Code and Deliver.* Upon the request from neighbor peer j, peer i generates a new coded block by the linear combination of its currently cached coded blocks, *which contain the original blocks peer* j *is seeking*.

With a simple example based on Fig. 4, we illustrate the effectiveness of this refinement. In the second time slot, after S_1 finds out S_3 knows original blocks b and d, it requests a coded block containing those two blocks from S_3. With the refined protocol, S_3 combines the two blocks and sends to S_1 a coded block in the form of $12b+7d$. Compared to the coded block $2a + 6b + c + 9d$ generated with the baseline protocol, delivering $12b+7d$ involves a much less coefficient overhead.

In addition, with this refinement, the computational overhead of network coding is reduced, as fewer coded blocks are combined to produce a new block. However, all the benefits come at the cost of possible reduced linear independence of coefficient vectors of the coded blocks cached at different peers, as they are now generated from fewer existing coded blocks. Therefore, this encoding refinement also introduces a tradeoff between the coefficient/computational overhead and failure tolerance of the system.

We introduce another boolean parameter *CodeAll* to indicate whether this refinement is incorporated in *Echelon* or not. A *true* value represents the baseline protocol, *i.e.*, a new coded block is generated by combining all existing blocks.

Altogether, we have introduced three parameters into the

Echelon protocol: *NumNeighbor*, *MaxRound* and *CodeAll*. Being highly configurable, the refined *Echelon* protocol can meet the requirements of different applications better. In the following section, with extensive simulation experiments, we demonstrate the effectiveness of these parameters.

V. Evaluations

In this section, we conduct an in-depth empirical study of the proposed baseline and refined *Echelon* protocols in simulated peer-to-peer environments. In our experiments, we simulate the coded dissemination in one epoch and evaluate the following performance metrics:

- ▷ **Rounds**: the maximum number of time slots the iterative protocol is executed at each peer
- ▷ **Decoding Efficiency**: the average number of coded blocks needed to obtain a $k \times k$ full-rank coefficient matrix for decoding, divided by k
- ▷ **Number of Peers to Probe**: the average number of peers the snapshot collector has to probe to obtain k coded blocks with linearly independent coefficient vectors.
- ▷ **Messaging Intensity**: the average number of messages sent by each peer in each time slot
- ▷ **Coefficient Overhead**: the average size of coefficient part (coefficients & original block IDs) in a data message

Decoding efficiency and number of peers to probe reflect the failure tolerance of the system: the fewer coded blocks acquired for decoding, the fewer peers the snapshot collector probes, the larger percentage of peer failures the system can tolerate. The other metrics represent messaging overhead in the coded dissemination of each epoch.

To evaluate the protocol in realistic network settings, we generate various random network topologies with the BRITE topology generator. The parameters to be investigated in our experiments include network parameters — network size (n), ratio of snapshot peers (k/n), average peer neighbor number (d), average peer cache capacity, and protocol parameters — *MaxRound*, *NumNeighbor*, *CodeAll*. As the general experimental setting, k/n is set to 0.8 in each network, $GF(2^{16})$ is used in network coding, and the average peer cache capacity for each epoch is 100.

A. Performance of base Echelon protocol

We first evaluate the baseline protocol itself, and then compare it with an uncoded random block dissemination scheme.

1) Dissemination speed: Fig. 5 shows the number of rounds the baseline protocol executes, until every peer has received every original block in coded form. The protocol stops quickly within number of rounds at the log scale of the network size, $O(\ln n)$, which conforms well to our analysis in Sec. III-C. When each peer has more neighbors in a network, the protocol terminates faster as the network diameter is smaller. The speed represents no difference with different snapshot peer percentages, since the network diameter remains the same.

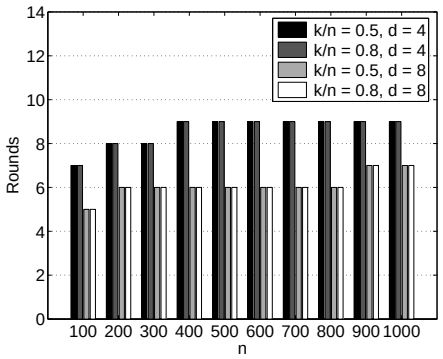

Fig. 5. Dissemination speed with the baseline *Echelon* protocol.

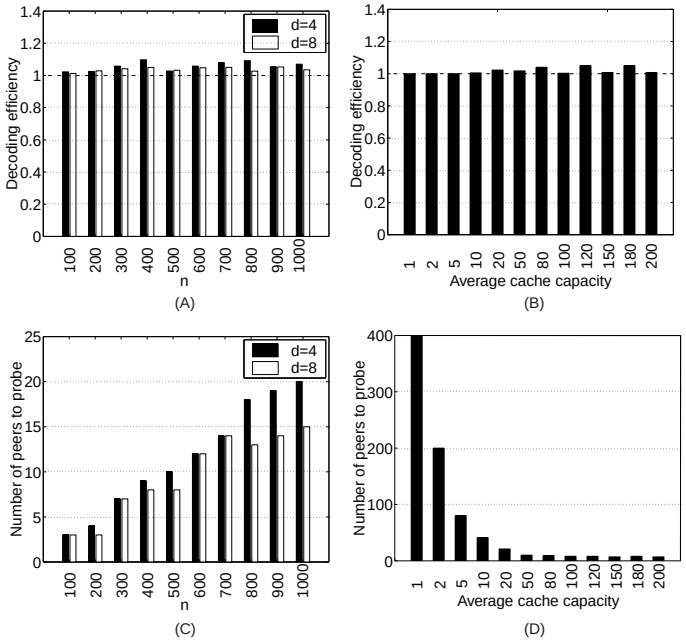

Fig. 6. Failure tolerance with the baseline *Echelon* protocol: (A) decoding efficiency and (C) number of peers to probe in networks of different sizes and neighbor numbers; (B) decoding efficiency and (D) number of peers to probe with different peer cache capacities in a network of $n = 500$, $d = 8$.

2) Failure tolerance: In each network, after the coded dissemination terminates, the snapshot collector randomly selects one peer after another, until it finds k coded blocks with linearly independent coding coefficient vectors. We repeat the experiment for 100 times, and report the average number of coded blocks aggregated for decoding and the average number of peers that provide these blocks.

Fig. 6(A) and (B) demonstrate the excellent linear independence of the resulting cached blocks in the network. With all parameter settings, any randomly selected k or slightly more than k coded blocks can be used for successful decoding. A careful study of Fig. 6(A) reveals that the decoding efficiency is even better when each peer has more neighbors in the network and thus receives more coded blocks in each round. In this case, its new blocks are generated from more existing blocks, leading to better linear independence in the system. Fig. 6(B) shows slightly better linear independence in the system with smaller peer cache capacity, as each resulting cached block contains more original blocks when received blocks are coded onto each other in the cache.

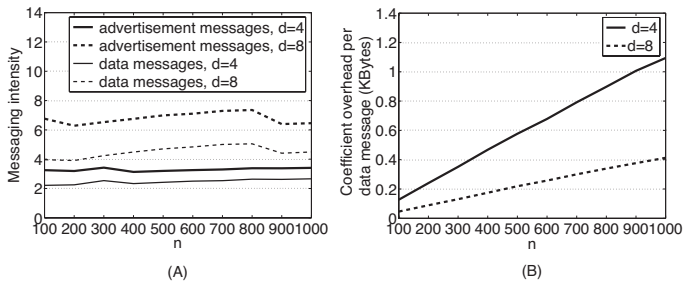

Fig. 7. Messaging overhead in the baseline *Echelon* protocol.

From Fig. 6(D), we observe that when the cache capacity is small, the number of peers to probe is $\frac{k}{\text{cache capacity}}$, as peers cache coded blocks as many as their cache capacity allows; when the caches are relatively large, the number remains consistent as now the number of cached blocks is decided by neighbor number and the number of rounds a peer executes the protocol. When the cache capacity is fixed at 100, Fig. 6(C) shows the slow increase of the probed peer number with the network size. When the average neighbor number d is larger, the number is much smaller, as each peer caches more blocks when the protocol terminates. Based on these observations, we find that the number of probed peers can be further reduced, if we incorporate the following protocol: if there is still available cache space at a peer after the protocol execution terminates, the peer can request more coded blocks from its neighbors to fill in the cache. However, this will introduce another tradeoff between messaging overhead and failure tolerance.

3) Messaging overhead: Fig. 7(A) demonstrates that the number of coded data messages is much smaller than that of advertisement messages, especially when d is larger. Thanks to network coding, a large number of original blocks can be disseminated with much fewer coded data messages.

In our experiments, each coding coefficient takes 2 bytes (chosen from $GF(2^{16})$), and a block ID is a 2-byte integer as well (identifying as many as 2^{16} peers). From Fig. 7(B), we observe that in a network with up to 1000 peers (800 original blocks as $k/n = 0.8$), the coefficient overhead is less than 1.1 KBytes, representing an average number of 275 original blocks included in the linear combinations. In addition, the coefficient overhead drops a lot when peers have more neighbors, which represents the case in realistic networks.

4) Comparison with uncoded random dissemination: In order to further validate the advantages of applying network coding, we compare our protocol with an uncoded random block dissemination scheme, which goes as follows:

In each time slot, every peer exchanges its block availability with its neighbors, and retrieves new original blocks from each other. When the cache at a peer has vacancy, a received block is directly stored; otherwise, the peer randomly selects an existing block in its cache and replaces it with the new block.

With this scheme, each peer also records the IDs of blocks that it has ever received. The protocol terminates when every peer has learned IDs of all the data blocks in the network. In this way, each peer eventually caches a random subset out of all the existing snapshots.

Comparing Fig. 8(A) with Fig. 5, we find the uncoded

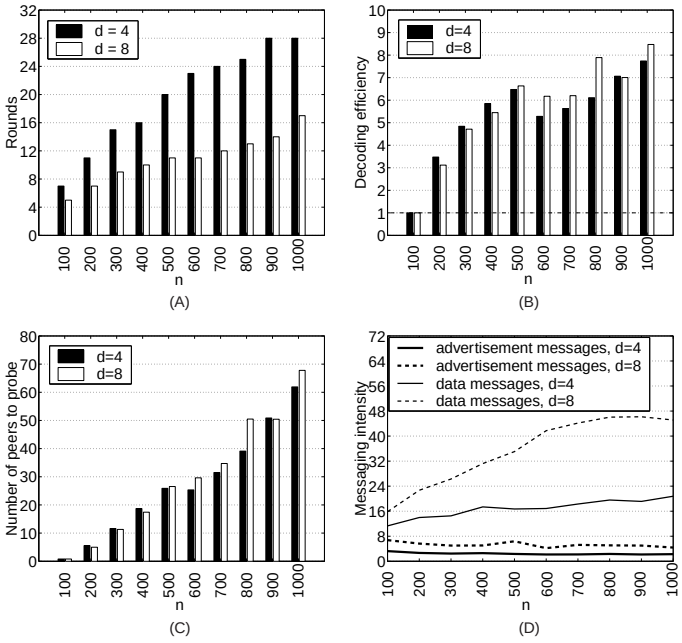

Fig. 8. Performance of an uncoded random dissemination scheme.

random dissemination terminates much slower than *Echelon*. Due to its random block replacement, the speed for one data block to propagate throughout the entire network has been much slowed down.

The comparisons between Fig. 8(B) and Fig. 6(A), Fig. 8(C) and Fig. 6(C) reveal the prominent advantage of network coding in enhancing failure tolerance of the network. With the uncoded scheme, the number of blocks aggregated by the snapshot collector in order to obtain all k snapshots becomes times larger than k, and the number of probed peers increases dramatically as well. For example, in the 200-peer network ($k = 160$ as $k/n = 0.8$), although each peer caches 100 different snapshots (cache capacity is 100), the snapshot collector still needs to probe 6 peers to get all the snapshots. Based on our observation, this is mainly caused by the "last block" problem, *i.e.*, the snapshot collector has to access lots of peers to find the last few missing blocks. This comparison result clearly demonstrates the effectiveness of network coding to avoid the "last block" problem.

In addition, comparing Fig. 8(D) with Fig. 7(A), we observe that the uncoded dissemination brings much higher messaging intensity. Considering the number of rounds it executes is also larger than that of *Echelon*, the total number of messages involved in the uncoded dissemination is much larger.

In summary, we conclude that network coding plays a significant role in *Echelon*, by providing much better scalability and failure tolerance with much less messaging overhead.

B. Effectiveness of advertising refinement

Next, we investigate the performance of *Echelon* with the advertisement refinement proposed in Sec. IV-A, with respect to different protocol parameters *MaxRound* and *NumNeighbor*. In these experiments, the average number of known neighbors at each peer in each network is 8.

We first study the impact of *MaxRound* on failure tolerance of the resulting system. As we know, the more rounds the

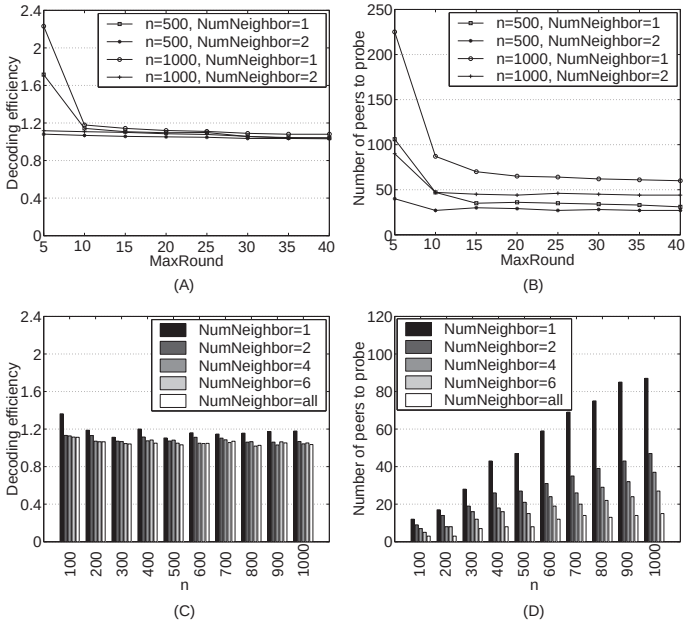

Fig. 9. Failure tolerance in *Echelon* with the advertising refinement.

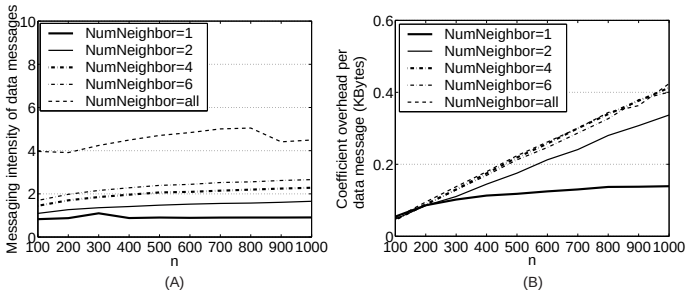

Fig. 10. Messaging overhead in *Echelon* with the advertising refinement.

protocol executes, the more peers each original block is distributed onto in coded form, the better failure tolerance the resulting system has. This is validated in Fig. 9(A) and (B). However, even in the extreme case that each peer only advertises to one neighbor in each round, the decoding efficiency quickly approaches 1 within 10 rounds, and the number of peers to probe drops quickly as well. This validates our analysis in Sec. IV-A that even when *MaxRound* is set to a small value, each original block can still be effectively distributed, thanks to the use of network coding.

Next, we set *MaxRound* to 10 and investigate the effect of the other protocol parameter, *NumNeighbor*.

Fig. 9(C) and (D) show that with the increase of *NumNeighbor*, failure tolerance quickly improves. Even a slight increase from 1 to 2 results in a great improvement. However, the best result is always achieved when the advertisement messages are forwarded to all neighbors, *i.e.*, the baseline protocol case.

Fig. 10 reveals that the messaging overhead is significantly reduced when the peers are not advertising to all their neighbors, not only in terms of data messaging intensity, but also the average coefficient size in each data message. This is because when *NumNeighbor* is smaller, each peer knows fewer original blocks, and therefore the number of coding coefficients included in each data message becomes smaller.

C. Effectiveness of encoding refinement

Finally, we examine the effect of encoding refinement discussed in Sec. IV-B. To this end, four schemes are compared, which are implemented respectively by setting the protocol parameters *CodeAll* and *NumNeighbor* to different values:

- *CodeAll = true, NumNeighbor = all*: the baseline *Echelon* protocol
- *CodeAll = false, NumNeighbor = all*: the *Echelon* protocol with encoding refinement only
- *CodeAll = true, NumNeighbor = 2*: the *Echelon* protocol with advertising refinement only
- *CodeAll = false, NumNeighbor = 2*: the *Echelon* protocol with both advertising and encoding refinements

Fig. 11(A) and (B) show slightly deteriorated decoding efficiency and increased number of probed peers when the encoding refinement is applied, due to increased linear dependence in the network by coding from fewer cached blocks. Fig. 11(C), on the other hand, demonstrates the coefficient overhead is much less when encoding refinement takes effect, as it aims to include in the new coded blocks only those original blocks requested by a neighbor. In addition, we observe that the effect of encoding refinement in reducing coefficient overhead is more significant than that of the advertising refinement, and the combination of both achieves the best results.

VI. Related Work

There exists little literature that touches upon the topic of diagnosing large-scale systems with peer-to-peer protocols. Stutzbach *et al.* [5] design a crawler to capture snapshots of Gnutella network, which focuses on accuracy of the captured snapshots and leverages the two-tier topology of such networks. NetProfiler [6] is a peer-to-peer infrastructure proposed for profiling wide-area networks, which aggregates information along DHT-based attribute hierarchies and thus may not adapt well to high peer churn rates.

Astrolabe [7] aggregates information for distributed system monitoring, with gossip-based information distribution and replication. Compared to the coded dissemination in *Echelon*, such uncoded dissemination and replication may involve large bandwidth and storage costs.

As data is disseminated and cached in a distributed fashion in *Echelon*, a reader may confuse *Echelon* with those distributed storage systems utilizing random linear coding [8], [9]. While both target at providing better failure tolerance for data aggregation, they represent fundamental differences. First, original snapshots in *Echelon* are produced in a completely decentralized fashion, while those systems disseminate parts of a centrally generated file. Second, the file is only coded at its origin in a distributed storage system. In *Echelon*, it is impossible to code from all the original blocks in such a centralized fashion, and a novel way of using randomized network coding is proposed to achieve distributed encoding.

Recently, random linear coding has also been utilized in sensor networks for failure tolerant aggregation of distributed data [10], [11], [12]. In both Dimakis *et al.* [10] and Rabbat *et al.* [11]'s work, original data is distributed and encoding is only

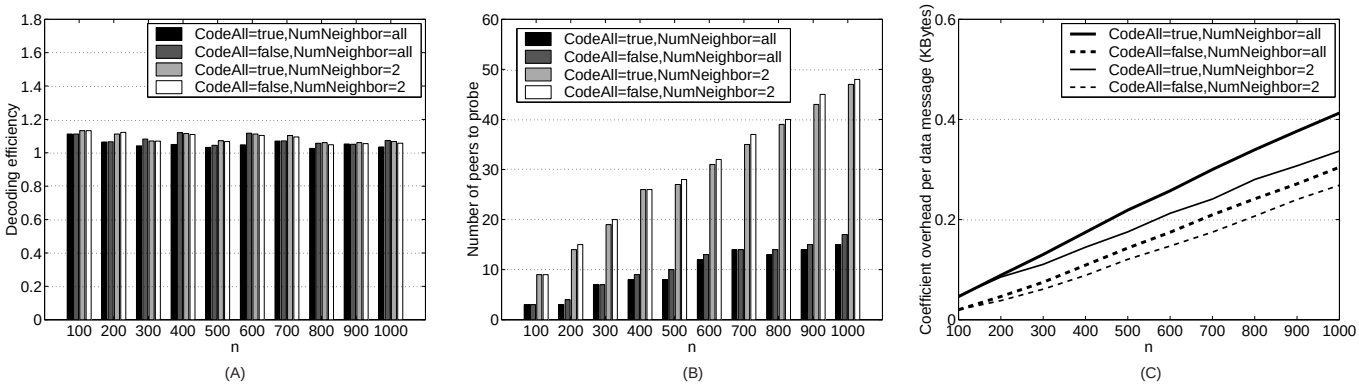

Fig. 11. Comparison of the baseline and refined *Echelon* protocols.

performed on the destination sensor nodes, in order to store them in a coded form. Compared to *Echelon*, such uncoded distribution involves larger messaging overhead. Wang *et al.* [12] propose a partial network coding scheme for continuous data collection, which removes obsolete original data from the stored coded data without the need of decoding. In contrast to these sensor network applications, *Echelon* utilizes network coding for diagnosing large-scale peer-to-peer networks, which represents a more practical scenario.

Network coding was first proposed to achieve the maximum capacity of a multicast network [1], [2], [4]. As mentioned earlier, other than bandwidth efficiency, *Echelon* also utilizes the significant advantages of block diversity and failure tolerance brought by randomized network coding.

In addition, most existing work in network coding is theoretical, and only a few discuss its practical implementation. Ho *et al.* [13] propose an asynchronous algorithm for network coding in cyclic networks, which requires the construction of an acyclic topology first. A practical network coding system is proposed by Chou *et al.* [14], which tackles cycles by encoding from already received blocks only. *Echelon* presents a more complicated scenario, and thus employs a practical iterative network coding approach. Avalanche [15] represents another practical content distribution system using randomized network coding. Different from *Echelon*, all its content originates from a same server, and each coded block is guaranteed to contain all the original blocks. The objective of *Echelon* is to disseminate the original blocks, which are produced in a completely distributed fashion, over the entire network in coded form. In this way, a complete copy of all original snapshots can be recovered from the live peers, even in cases of large-scale peer failures.

VII. Conclusion

We conclude this paper by reinforcing our belief that monitoring and diagnosis of large-scale peer-to-peer sessions represent a novel and critical application. Towards efficient diagnosis construction, we have presented *Echelon*, a light-weighted protocol to disseminate peer snapshots over the entire network with network coding. With such coded dissemination, the measurements collected at all peers, including those that have failed before the aggregation, can be effectively recovered by simply probing a random subset of the existing peers. Utilizing randomized network coding, the dissemination enjoys significant advantages of being bandwidth efficient, scalable and extremely failure tolerant. Motivated by the positive experimental results, we are confident in the effectiveness of *Echelon* in real-world scenarios. In our ongoing work, we are working on the implementation of *Echelon*, and look forward to leveraging it in the diagnosis of large-scale peer-to-peer applications over the Internet.

References

[1] R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung, "Network Information Flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, July 2000.

[2] R. Koetter and M. Médard, "An Algebraic Approach to Network Coding," *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 782–795, October 2003.

[3] S. Y. R. Li, R. W. Yeung, and N. Cai, "Linear Network Coding," *IEEE Transactions on Information Theory*, vol. 49, p. 371, 2003.

[4] T. Ho, M. Médard, J. Shi, M. Effros, and D. Karger, "On Randomized Network Coding," in *Proc. of 41st Annual Allerton Conference on Communication, Control, and Computing*, October 2003.

[5] D. Stutzbach and R. Rejaie, "Capturing Accurate Snapshots of the Gnutella Network," in *Proc. of IEEE Global Internet Symposium*, March 2005.

[6] V. N. Padmanabhan, S. Ramabhadran, and J. Padhye, "NetProfiler: Profiling Wide-Area Networks Using Peer Cooperation," in *Proc. of IPTPS 2005*, February 2005.

[7] R. van Renesse, K. Birman, and W. Vogels, "Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management and Data Mining," *ACM Transactions on Computer Systems*, vol. 21, no. 2, pp. 164–206, May 2003.

[8] S. Acedański, S. Deb, M. Médard, and R. Koetter, "How Good is Random Linear Coding Based Distributed Networked Storage," in *Proc. of Netcod 2005*, April 2005.

[9] M. K. Aguilera, R. Janakiraman, and L. Xu, "Using Erasure Codes Efficiently for Storage in a Distributed System," in *Proc. of DSN'05*, 2005.

[10] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Ubiquitous Access to Distributed Data in Large-Scale Sensor Networks through Decentralized Erasure Codes," in *Symposium on Information Processing in Sensor Networks (IPSN '05)*, April 2005.

[11] M. Rabbat, J. Haupt, A. Singh, and R. Nowak, "Decentralized Compression and Predistribution via Randomized Gossiping," in *Symposium on Information Processing in Sensor Networks (IPSN '06)*, April 2006.

[12] D. Wang, Q. Zhang, and J. Liu, "Partial Network Coding for Continuous Data Collection in Sensor Networks," in *Proc. of the Fourteenth IEEE International Workshop on Quality of Service (IWQoS 2006)*, June 2006.

[13] T. Ho, B. Leong, R. Koetter, and M. Mdard, "Distributed Asynchronous Algorithms for Multicast Network Coding," in *Proc. of Netcod 2005*, April 2005.

[14] P. A. Chou, Y. Wu, and K. Jain, "Practical Network Coding," in *Proc. of the 51st Allerton Conference on Communication, Control and Computing*, October 2005.

[15] C. Gkantsidis and P. Rodriguez, "Network Coding for Large Scale Content Distribution," in *Proc. of IEEE INFOCOM 2005*, March 2005.