Incorporating Random Linear Network Coding for Peer-to-Peer Network Diagnosis

Elias Kehdi, Baochun Li
Department of Electrical and Computer Engineering
University of Toronto
{*elias, bli*}*@eecg.toronto.edu*

***Abstract*—Recent studies show that network coding improves multicast session throughput. In this paper, we demonstrate how random linear network coding can be incorporated to provide network diagnosis for peer-to-peer systems. We present a new trace collection protocol that allows operators to diagnose peer-to-peer networks. It is essential to monitor large-scale peer-to-peer applications by collecting measurements referred to as snapshots from the peers. However, existing solutions are not scalable and fail to collect measurements from peers that departed before the time of collection. We use progressive random linear network coding to disseminate the snapshots in the network, from which the server pulls data in a delayed fashion. We leverage the power of progressive encoding to increase block diversity and tolerate extreme block losses by introducing redundancy in the network. Peers cooperate by allocating cache capacity for other peers. Snapshots of departed peers can thus be retrieved from the network. We show how our protocol controls the redundancy introduced through progressive encoding and thus scales to large number of peers and tolerates high level of peer dynamics.**

I. Introduction

Peer-to-peer applications have been successfully deployed to provide many tasks such as content distribution, live streaming, distributed computation and collaborations. The main advantages of peer-to-peer architectures are scalability, resilience to failure and easiness of use. However, the challenge of large-scale peer-to-peer systems is the lack of internal knowledge of the architecture and the Quality of Service parameters at the participating peers, due to the nature of peer-to-peer communication. It is critical to monitor such systems by collecting measurements from the peers referred to as snapshots or traces. Such measurements consist of Quality of Service metrics that allow operators to diagnose large-scale peer-to-peer applications.

A common approach to characterize and diagnose peer-to-peer networks is to collect periodic statistics from the peers. Users periodically measure critical parameters and send them to logging servers. However, such periodic snapshots involve high traffic and consume large bandwidth when the number of peers is particularly large. UUSee Inc. [1] is a live peer-to-peer streaming provider that relies on logging servers to collect and aggregate snapshots periodically sent by each peer. Every five or ten minutes, each peer sends a UDP packet to the server containing vital statistics. However, the server bandwidth is not sufficient to handle such excessive amount of data. In fact, UUSee trace collection completely shuts down when it cannot handle the load of periodic snapshots. Certainly, it is not a scalable trace collection protocol.

This work is supported by NSERC Discovery, CRD and Strategic Grants (RGPIN 238994-06, CRDPJ 379623-08, STPGP 364910-08).

The efficiency of a trace collection protocol depends on the accuracy of the aggregated snapshots which is defined by the completeness of the measurements. The goal is to collect snapshots from all the peers even those who have left the session before the time of collection. In other words, an efficient trace collection protocol should be able to capture the dynamics of the peers which is a critical parameter for operators that allow monitoring of network performance. The most useful statistics are those collected from peers leaving the network due to Quality of Service degradation. The operators of peer-to-peer systems are highly interested in those valuable snapshots. However, they fail to capture accurate snapshots since the amount of data is limited to the server bandwidth. Indeed, operators tend to increase the time interval between snapshots or pull data from a small subset of the peers.

Since the trace collection is delay-tolerant, some designs propose to disseminate the traces produced by the peers in the network and allow the server to probe the peers in a delayed fashion. Such approach prevents peers from sending excessive simultaneous flows and shutting down the server. To tolerate traces losses due to peer dynamics, some redundancy is injected in the network. Network coding has been proposed to disseminate the traces in order to increase data diversity and to be resilient to losses [2], [3]. However, they did not demonstrate how they can control the redundancy introduced and thus did not prove to scale to large-scale peer-to-peer networks. The challenge is to utilize network coding in a way that allows the protocol to scale and, at the same time, to increase the diversity of the exchanged blocks.

In this paper, we present a new trace collection protocol that uses random linear network coding to exchange and store the snapshots in the network. Our protocol allows continuous trace generation and arbitrary trace dissemination by the peers. The peers disseminate coded snapshots and cache them in a decentralized fashion in the peer-to-peer network. The server periodically probes the peers using a small fixed bandwidth in order to reconstruct the collected snapshots. The peers cooperate in this process by allocating cache capacity to store snapshots generated by other peers. We use progressive encoding to control the redundancy introduced in the network and the storage cost. Progressive network coding increases the server decoding efficiency by progressively increasing

the blocks diversity. Thus, it guarantees resilience to large-scale peer departures and allows our design to adapt to peer dynamics and thus scale and handle flash crowds of peer arrivals. We show how our trace collection protocol is able to capture accurate snapshots by reporting the percentage of generated snapshots collected by the server under high levels of peer departures.

The remainder of the paper is organized as follows. In Section II, we discuss related work on trace collection protocols. In Section III, we present an overview of our protocol and discuss how progressive encoding guarantees efficient traces dissemination. We present the complete trace collection protocol in Section IV. We evaluate our design and demonstrate its tolerance to high level of peer dynamics through theoretical analysis and simulations in Section V and Section VI, respectively. Finally, we conclude the paper in Section VII.

II. Related Work

Little literature exists on protocols designed to collect measurements from peer-to-peer systems. Astrolabe [4] is a distributed information management system that aggregates measurements with gossip-based information exchange and replication. However, such information dissemination imposes significant bandwidth and storage costs. NetProfiler, proposed by Padmanabhan *et al.* [5], is a peer-to-peer application that enables monitoring of end-to-end performance through passive observations of existing traffic. The measurements are aggregated along DHT-based attribute hierarchies and thus may not be resilient to high peer churn rates. Several other tools have been developed for connectivity diagnosis such as pathchar [6] and tulip [7]. But these tools can be expensive and infeasible since they rely on active probing of routers. Stutzbach *et al.* [8] present a peer-to-peer crawlers to capture snapshots of Gnutella network. The goal of the crawler is to increase the accuracy of the captured snapshots by increasing the crawling speed. The protocol leverages the two-tier topology of Gnutella and thus is difficult to be generalized to other peer-to-peer systems.

On the other hand, Echelon [2] uses network coding to disseminate the snapshots in the network. Only coded snapshots are exchanged in the network. It utilizes the advantage of block diversity and failure tolerance brought by randomized network coding. However, the number of peers that produce the snapshots is limited since the block size grows with the amount of snapshot peers. Also, the snapshots generation is divided into epochs during which each peer is required to receive a coded block of all the snapshots. This limits the amount of traces generated or the set of peers that collect measurements. Niu *et al.* [3] present a theoretical approach on using network coding for trace collection. Also, in their protocol, the traces generation is divided into periods of time. The mechanism is a probabilistic gossip protocol that performs segment based network coding to buffer the snapshots in the network for the server to collect them in a delayed fashion. But such gossip protocol results in a significant redundancy which limits its scalability. The segment size factor plays an important role in controlling the coded blocks disseminated in the network and allowing the server to reconstruct the traces generated by the peers. Hence, the design of a scalable trace collection protocol that adapts to peer dynamics remains to be a major challenge. As in [2], [3], we leverage the power of network coding to collect snapshots. However, we present a more practical way to use network coding for the dissemination of the traces in the network. In fact, choosing a segment size equal to the number of generated blocks during an epoch, as in [2], limits the scalability of the protocol and, on the other hand, reducing the segment size to the number of generated blocks of a single peer during an epoch, as in [3], limits the diversity of the exchanged blocks. To solve this problem, we propose to use progressive encoding. First, we remove the periodic snapshot capturing restriction and allow arbitrary trace dissemination in order to collect measurements from departing peers. Second, we use progressive random linear network coding for trace dissemination to increase the block diversity and reduce storage cost. Finally, our protocol controls the redundancy introduced in the network and adapts to peer dynamics, and hence, unlike previously proposed schemes, it scales to large-scale peer-to-peer sessions.

First introduced by Ahlswede *et al.* [9], network coding has been shown to improve information flow rates in multicast sessions [9]–[11]. The main idea is to allow the nodes to perform coding operations, instead of simple replication and forwarding, in order to alleviate competition among flows at the bottlenecks. In a more practical setting, random linear network coding, first proposed by Ho *et al.* [12], has been shown to be feasible. For instance, Avalanche [13] uses randomized network coding for content distribution to reduce downloading times. Network coding is performed within segments to which a random linear code is applied. Another advantage of randomized network coding is its ability to increase the diversity of data blocks and improve resilience to block losses. Wu [14] also argues that network coding adapts to network dynamics, such as packet loss and link failures. In this paper, we leverage the power of network coding to diagnose large-scale peer-to-peer systems using vital statistics collected from the network.

III. Protocol Overview

We present a new trace collection protocol for large scale live peer-to-peer applications. The protocol allows continuous trace generation and arbitrary trace dissemination from participating peers. The server collects the snapshots by periodically probing the network. Depending on their Quality of Service experience, the peers generate vital statistics and exchange them with neighbors allowing the participating peers to cache them in a decentralized fashion. We do not assume a periodic fixed size trace exchange but rather the peers produce data independently at any time depending on their Quality of Service experience. Such trace generation allows the peers to produce more traces when experiencing important performance changes and to disseminate them even at the time they have to leave the session. On the other hand, the server allocates a small bandwidth to periodically pull data from a randomly selected set of peers during the streaming session.

By buffering the traces in the network, we prevent the peers from uploading their data simultaneously and hence we allow the server to collect excessive data traffic in a delayed fashion when the number of peers increases dramatically. In order to be tolerant to peer dynamics, the trace collection mechanism disseminates copies of the data generated in the network. By introducing redundancy in the network, the server can pull the traces produced by the peers even after they have left the session.

In our protocol, we use randomized linear network coding for traces exchange and storage. Network coding increases the diversity of the data blocks and the tolerance to block losses upon peer departures. Network coding is performed within segments, where each segment is defined by each peer as the set of blocks forming their snapshots. Random linear combinations are applied to each segment, *i.e.* to the blocks generated by the same peer. For instance, a peer that has received blocks, that belong to the same segment, generates new coded blocks by the linear combinations of the received blocks over a Galois field $GF(2^q)$, using randomly chosen coefficients. It sends those blocks to neighbors and stores one coded block to be sent to the server once probed. The segment size is a key factor in the design of the trace collection protocol. The best solution would require maximizing the segment size. As such, we maximize the blocks diversity and solve the caching capacity problem of the peers. When the segment size is small, the snapshots are less tolerant to peer dynamics and the peers inject more segment in the network. Therefore, neighbors would have to decide which segment to cache and which segment to send. In contrast, when the segment size is larger, the coded blocks can be better disseminated in the network without generating many dependencies. The goal is to propagate the coded blocks to as many peers as possible, in order to resist peer departures, without resulting in many block dependencies at the server.

However, if peers wait to generate additional snapshots in order to increase the number of blocks forming their segment, they might leave the session and hence, their measurements would be lost. We attempt to solve this problem by using progressive encoding in our protocol. The peers perform progressive encoding by grouping newly generated traces in a segment containing previous traces already disseminated in the network. Hence, the segment size is defined by each peer and increases depending on the Quality of Service experience. As the segment size increases, the blocks are disseminated further in the network to tolerate peer departures. Hence, peers control the redundancy introduced in the network by modifying the segment size. A cached block is replaced with its random linear combination with a newly received block that belongs to the same segment and which can contain more coefficients. Through progressive encoding, the peers effectively disseminate the coded blocks in the network. Also, the protocol takes advantage of the fact that the server is periodically probing the network since blocks containing different number of coefficients are utilized to reconstruct the snapshots and hence, are used to decode parts of the segment they belong to. Finally, our protocol adapts to peer dynamics by allowing the peers to adjust the amount of disseminated coded blocks depending on their local view of neighbors' departure rate. We study and present an in-depth view of our protocol in the subsequent sections.

IV. Trace Collection Protocol

In this section, we present our trace collection protocol which applies progressive random linear network coding on the snapshots disseminated in the network. We assume that all the peers allocate a cache capacity to store snapshots from other peers. The participating peers encode blocks that belong to the same segment, hence generated by the same peer. They generate and distribute snapshots independently without any time interval restrictions. We first present the data block format and then describe our trace collection protocol.

A. Data Block Format

The format of the coded data block, shown in Figure 1, addresses the progressive encoding used in our trace collection protocol. We include the ID of the peer that produced the trace associated with the data block. We also include an entry to indicate the segment number defined by the peer that has produced the block. The SF entry represents the spreading factor that defines how far should the blocks associated with this segment propagate. We also append the coefficients used in the encoding process and the payload of the data block.

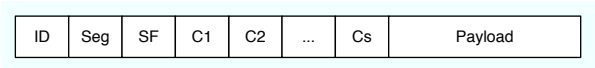


Fig. 1. Data block format.

The peers do not exchange any acknowledgements or requests, instead they only disseminate sufficient data blocks in the network so that the server can decode their traces. The server, on the other hand, does not send any acknowledgment to the peers, instead it periodically collects data blocks cached in the network and reconstructs original segments when possible. Otherwise, such messages would lead to significant overhead when used.

B. Protocol Description

Our goal is to control the redundancy introduced by the dissemination of the traces in the network and to efficiently store the coded blocks in the peers' caches until they are collected by the server. We start by discussing the spreading factor entry, shown in Figure 1, that corresponds to the group of blocks forming a segment k. Its value is set by the peer that has generated the segment and is modified whenever new blocks are added to the segment. The spreading factor determines the number of blocks that should be disseminated in the network. In other words, it is an indicator of the number of peers that should be reserving an entry in their caches for that segment k. In our trace collection protocol, the spreading factor of a segment depends on its size, defined by the number of blocks it contains, and the neighbors' departure rate. In fact, coded blocks belonging to a larger segment should be cached more frequently in the network in order for the server to decode them. Furthermore, a higher spreading factor

guarantees better tolerance to peer dynamics. Blocks should spread further in the network to resist losses due to high level of peer departures. In our protocol, peers choose the appropriate spreading factor for the newly generated blocks based on the local view of neighbor dynamics. Hence, the number of coded blocks disseminated in the network adapts to the dynamic of the peers.

The spreading factor of a segment k is calculated using a logarithmic function as shown in Equation (1). We use a logarithmic function to take into account previously disseminated blocks of segment k. Those blocks, with different number of coefficients, are used in the decoding process by the server. The spreading factor is defined as

$$SF = \alpha(d)\log_2(\beta n_c + 1), \tag{1}$$

where n_c is the number of coefficients of segment k. The parameters $\alpha(d)$ and β determine the redundancy or the number of coded blocks to be injected in the network. The variable d is the dynamic percentage rate measured using the local view of neighbor dynamics. In our protocol, $\alpha(d)$ is a step function as shown in Equation (2). A peer measures its neighbor dynamics and modifies the spreading factor based on its sensitivity indicated by the percentage levels p_l.

$$\alpha(d) = \begin{cases} \alpha_1 & d <= p_1 \\ \alpha_2 & p_1 < d < p_2 \\ \cdots & \\ \alpha_l & d > p_l \end{cases} \tag{2}$$

Our design objective is to effectively cache the traces in the network and communicate them in a manner that resists high level of peer dynamics and allows the server to reconstruct the traces by periodically pulling a fixed amount of arbitrarily blocks from the network. The protocol is implemented using progressive encoding as shown in Algorithm 1.

Upon experiencing Quality of Service changes, a peer produces new measurements to be collected by the server. Consequently, it generates traces and divides them into blocks of size 1 KB each which fits in a single UDP packet. Then, it adds those blocks to its segment k that was previously disseminated in the network. After modifying the segment and increasing its size, the peer recalculates the spreading factor SF using Equation (1). Accordingly, when it decides to inject those newly generated blocks in the network, it sends coded blocks from segment k. A peer can decide to switch to a new segment when n_c reaches a maximum value that leads to significant coefficient overhead. Note that the blocks exchanged in the network, that belong to the same segment, can have different number of coefficients. Such blocks are encoded together after appending a corresponding number of zeros to adjust their sizes.

Upon receiving blocks from a segment k, a peer retrieves the spreading factor from the messages. It then applies random linear combinations to the newly received blocks and the cached blocks that belong to that same segment k. According to the retrieved spreading factor, it sends coded blocks to neighbors and sets their appropriate SF entries. Finally, it stores a coded block by replacing previous cache entry of the segment k, if it exists. In case the cache is full, it replaces the oldest segment entry in the cache memory. Hence, as the segment size increases, the blocks propagate further in the network replacing previous versions of that same segment. The number of disseminated blocks is determined by the spreading factor set by the peer that has produced the traces. By controlling the number of disseminated blocks, the protocol allows the server to decode a large segment of snapshots and limits the number of linear dependent blocks in case of a small segment size. As such, our protocol decreases the chance of linear dependency when there are few coded blocks and increases the chance of decoding a segment when it contains a large number of coded blocks.

Progressive encoding is significant and effective when we

Algorithm 1 *Trace Collection Protocol.*

Sending original coded blocks
- $M \leftarrow$ collect measurements
- $\widehat{B} \leftarrow$ divide M and packetize it into block of 1KB each
- $k \leftarrow$ previously disseminated segment or new segment ID
- $\widehat{B}_k \leftarrow \widehat{B}_k \cup \widehat{B}$
- $SF = \alpha(d)\log_2(\beta \times sizeof(\widehat{B}_k) + 1)$
- **for** $u = 1$ to $sizeof(Neighbors)$
  - $SF_u \leftarrow$ Disseminate blocks $s.t. \sum SF_u = SF$
  - **for** $i = 1$ to SF_u
    - $C \leftarrow sizeof(\widehat{B}_k)$ random coefficients
    - $b = C \times \widehat{B}_k$
    - Send b to peer u
  - **end for**
- **end for**

Receiving coded blocks
- $\widehat{B} \leftarrow$ received coded blocks
- $SF \leftarrow$ retrieved message spreading factor
- $SF = SF - 1$
- $ID \leftarrow$ retrieved message source ID
- $k \leftarrow$ retrieved message segment ID
- $j \leftarrow$ Cache entry for segment k from peer ID
- $\widehat{B}' \leftarrow \widehat{B}_j \cup \widehat{B}$
- **for** $u = 1$ to $sizeof(Neighbors)$
  - $SF_u \leftarrow$ Disseminate blocks $s.t. \sum SF_u = SF$
  - **for** $i = 1$ to SF_u
    - $C \leftarrow sizeof(\widehat{B}')$ random coefficients
    - $b = C \times \widehat{B}'$
    - Send b to peer u
  - **end for**
- **end for**
- $C \leftarrow sizeof(\widehat{B}')$ random coefficients
- **if** $j == \emptyset$
  - **if** cache is full
    - $j \leftarrow$ oldest cache entry
  - **else**
    - $j \leftarrow$ new cache entry
  - **end if**
- **end if**
- $\widehat{B}_j = C \times \widehat{B}'$

avoid using acknowledgments and requests in the trace collection protocol. In fact, through the segment replacement mechanism, progressive encoding solves the problem of data block caching, where the peers have to decide wether to keep an existing block or replace it with a new one. The idea behind using progressive encoding is to increase blocks diversity in order to spread traces to as many peers as possible without introducing many block dependencies at the server. By encoding previously disseminated blocks with newly generated blocks, we increase blocks diversity in live trace generation. The server, on the other hand, collects blocks belonging to segments that are increasing in size as it periodically probes the network and deletes them from the peers' caches. With progressive encoding, such cached blocks are more meaningful since they help decoding segments that are increasingly containing newly generated blocks. In addition, we avoid the problem of decoding all or nothing since the server can decode part of the segment if it has collected enough blocks during its periodic probing.

V. Theoretical Analysis

In this section, we study the overhead generated by the dissemination of coded blocks and the relationship between the parameters of our trace collection protocol discussed in the previous sections.

A. Data Dissemination Overhead

In order for the design to scale, it should control the redundancy introduced by the dissemination of the snapshots in the network. The segment size is an important factor in regulating the overhead involved in our trace collection mechanism.

With progressive encoding, the coefficient overhead of a segment depends on its size defined by the number of coefficients n_c it contains. Network coding operations are performed over a Galois field 2^q. In a network of n peers the coefficient overhead is around $r \times q + \log_2 n$ bits. We limit the size of a block to $1KB$ which fits in a single UDP packet.

On the other hand, the communication overhead is defined by the redundancy ratio and the cache capacity allocated by the peers. Redundant blocks are not duplicates distributed in the network. Instead, when network coding is applied, redundant blocks are additional coded blocks exchanged. The server has to collect n_c blocks in order to decode a segment containing n_c coefficients. However, to tolerate peer dynamics, peers have to disseminate additional blocks which introduces redundancy in the network. Based on the protocol described in Algorithm 1, the redundancy ratio is defined as

$$RR = \frac{\alpha(d)\log_2(\beta n_c + 1)}{n_c}. \tag{3}$$

Note from Equation (3) that the redundancy adapts to the dynamics of the peers. Based on the local view of neighbor dynamics, a peer can adjust the spreading factor of its segments. Indeed, with higher level of peer dynamics, a peer should disseminate more blocks to tolerate the loss due to the departure of neighbors storing its data. Hence our trace collection protocol adapts to peer dynamics and prevents redundant blocks from flooding the network.

As previously discussed, the cooperation of the peers is determined by the cache capacity allocated to store blocks from other peers. Peers with long lifetime contribute the most to the system. In our design, we fix the cache capacity allocated for other peers and replace oldest entries when the cache is full. Peers store one copy of each segment received, to be sent to the server once probed. The server, on the other hand, pulls a fixed amount of data Q_s from the network every period of time T_s. Each block pulled from the network is deleted from the caches.

B. Decoding Condition

In order to simplify the analysis, we assume that the events occur at discrete time $t = 0, T, 2T, \ldots$. Consider a peer that has generated a segment k of size n_c and has disseminated its coded blocks to C peers in a network of size N. The server has to collect n_c coded blocks in order to decode and reconstruct the snapshots. However, the server uses a limited bandwidth to periodically probe the network consisting of dynamic peers that leave the session at a constant rate. We assume that the session terminates when all the peers depart.

We use the following parameters in our analysis. We denote by C_t the number of peers that store coded blocks from segment k at time t. Also, we denote the number of participating peers at time t by N_t. Assume that the server probes the network at a rate of one coded block from each of the R_p peers every period of time T. The peers, on the other hand, depart at rate R_d peers every period of time T. Hence, in this scenario, we have $N_t = N - t \times R_d$ and the redundancy ration $RR = C/n_c$.

Assume that the random variables C_t at different times t are independent. We define C_t as the number of peers that have cached blocks from segment k after the departure of $t \times R_d$ peers. The probability $P(C_t = l)$ is equivalent to the probability that $C - l$ peers that store coded blocks from segment k have left the session before time t. We thus have

$$P(C_t = l) = \frac{\binom{C}{C-l}\binom{N-C}{tR_d - C + l}}{\binom{N}{tR_d}}. \tag{4}$$

The expected value of the random variable C_t is

$$E(C_t) = \frac{1}{\binom{N}{tR_d}} \sum_{l=0}^{\min\{C, N - tR_d\}} l \binom{C}{C-l}\binom{N-C}{tR_d - C + l}. \tag{5}$$

Define the random variable x_t as the number of collected blocks by the server when it probes the network at time t. The variable x_t follows a binomial distribution with parameters R_p and $\frac{C_t}{N_t}$. The probability that the server collects l coded blocks from segment k at time t by probing l peers storing those blocks is

$$P(x_t = l) = \binom{R_p}{l}\left(\frac{C_t}{N_t}\right)^l \left(1 - \frac{C_t}{N_t}\right)^{R_p - l}. \tag{6}$$

Therefore, the expected value of x_t is $E(x_t) = R_p \times \frac{C_t}{N_t}$. In other words, the expected number of collected coded blocks

when the server probes the network at time t is equal to $R_p \times \frac{C_t}{N_t}$. Since the session terminates when all the peers leave the session at time $t = \frac{N}{R_d}$, we express the total number of collected blocks as the random variable X such that

$$X = \sum_{t=0}^{N/R_d} x_t = R_p \sum_{t=0}^{N/R_d} \frac{C_t}{N_t}.$$

We thus have

$$\begin{aligned} & E(X) \\ = \; & R_p \sum_{t=0}^{N/R_d} \frac{E(C_t)}{N_t} \\ = \; & \frac{R_p}{\binom{N}{tR_d}} \sum_{t=0}^{N/R_d} \left(\frac{1}{N_t} \sum_{l=0}^{\min\{C, N-tR_d\}} l \binom{C}{C-l} \binom{N-C}{tR_d - C + l} \right) \end{aligned}$$

In order for the server to reconstruct the snapshots from segment k, it has to collect at least n_c independent coded blocks. Hence, the expected number of collected blocks $E(X)$ should be at least equal to n_c. Therefore, we need to enforce the following condition:

$$\frac{R_p}{\binom{N}{tR_d}} \sum_{t=0}^{N/R_d} \left(\frac{1}{N_t} \sum_{l=0}^{\min\{C, N-tR_d\}} l \binom{C}{C-l} \binom{N-C}{tR_d - C + l} \right) \geq n_c, \quad (7)$$

where $N_t = N - tR_d$.

In our trace collection, the parameter C in condition (7) can be replaced by $RR \times n_c$ which refers to the number of coded blocks disseminated in the network, hence, the number of peers that store a coded block from segment k. It is evident that a larger segment size n_c requires more peers to store coded blocks. Note that the further the right hand side of condition (7) exceeds the segment size, the higher is the expected number of dependent blocks at the server. In our protocol, we approximate the rates and messaging intensity in order to satisfy condition (7).

We note from condition (7) that the most efficient way to disseminate the coded blocks is by spreading the blocks to as many peers as possible. In fact, increasing the number of coded blocks stored at the peers does not adjust the value of C to satisfy condition (7). Instead, C only depends on the number of peers storing coded blocks from segment k. By choosing the appropriate messaging intensity we satisfy the condition and hence tolerate the peer dynamics. This also applies to the number of segments generated by a single peer. To better resist network dynamics, a peer should maximize the segment size used and hence, minimize the number of injected segments to efficiently spread its coded blocks in the network. For this purpose, we implement progressive encoding in our trace collection protocol, through which, we increase the segment size and control the redundancy introduced in the network.

VI. Protocol Evaluation

In this section we study the efficiency of our trace collection protocol and its resilience to high level of peer dynamics. We simulate a live peer-to-peer session where the peers arbitrarily exchange data blocks as previously discussed in order to disseminate their snapshots in the network. The duration of the session is 600 minutes. We use event-driven simulations and model the peer dynamics using exponential distribution with a mean value e. The peers generate traces and send their blocks independently. We also model their behavior using exponential distribution. The mean of the distribution a_i defines the aggressiveness of a peer i. The server collects a fixed amount of data Q_s from the network every period of time T_s. The number of blocks generated by the peers during the session should be less than $Q_s \times T_s$ in order for the server to decode all the traces. We use the ratio of decoded blocks to the blocks generated as a metric to evaluate our protocol under different levels of peer dynamics. We also measure the redundancy collected or the linear dependent blocks collected by the server. Furthermore, we evaluate the messaging intensity defined as the average number of blocks sent by each peer during the session. We generate various random topologies and investigate parameters such as peer dynamics, associated with the mean of the exponential distribution e, the peers' cache capacity and the spreading factor SF. In the simulations, we fix the parameter β of SF to 0.25 and vary $\alpha(d)$.

A. Delayed Data Collection

Through simulations, we show how our protocol can scale to large-scale peer-to-peer networks. For this purpose, we report the percentage of generated blocks decoded by the server when it periodically pulls a fixed amount of data from the network. With this delayed trace collection, the server can handle large-scale peer-to-peer networks and prevent peers from sending simultaneous excessive data.

Figure 2 shows the number of decoded blocks in function of the spreading factor in a network consisting of 1000 peers. We fix β to 0.25 and vary the parameter $\alpha(d)$ of the spreading factor SF. In this scenario $\alpha(d)$ does not depend on neighbor dynamics. We model the peer dynamics by setting the mean of the exponential distribution e to 100 minutes. We notice that the peers generate around 30,000 blocks during the session. The server collects a fixed amount of 800 blocks every 10 minutes. By caching the coded blocks in the peer-to-peer network, the server is able to collect and reconstruct the traces under the specified rate of peer departures. We notice from Figure 2, that for small values of SF, the redundancy created in the network is not sufficient to tolerate the peer dynamics which limits the decoding capabilities of the server. On the other hand, as we increase the spreading factor, the blocks reach more peers and hence resist losses due to peer departures. Therefore, the server is able to collect enough packets to reconstruct most of the traces generated. We also observe that the number of dependent blocks collected by the server slightly increases. In fact, the redundancy disseminated in the network are coded blocks that are equally useful in the decoding process since progressive encoding increases the diversity of the blocks exchanged. This is one of the advantages of using progressive network coding for blocks dissemination. Moreover, we note that the message intensity is around 900 KB. It increases as we increase the spreading

factor. In fact, more redundancy is cached in the network and more data blocks are exchanged by the peers for larger values of $\alpha(d)$.

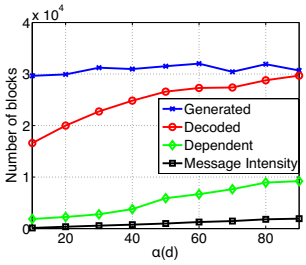


Fig. 2. Message intensity and decoding efficiency as a function of $\alpha(d)$.

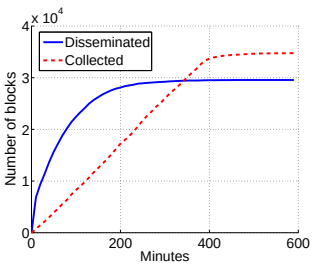


Fig. 3. Blocks dissemination and collection in function of time.

Furthermore, to clearly see how the server collects data in a delayed fashion, we report the blocks dissemination and collection in function of time in Figure 3. Note that the number of blocks collected follows a straight line since the server periodically pulls 800 blocks from the network, and as such prevents the peers from uploading excessive flows. The gap between the curves demonstrates how the blocks are disseminated first in the network and then collected by the server at a later time. Between time 150 and 400 the blocks generation slows down since the number of participating peers dramatically decreases. The remaining peers, with a long lifetime, store the snapshots of other peers that have already left. We observe from Figure 3, how the server is able to pull data from those peers until time 400, where all the participants have left the session. Note that in this scenario, the server was able to reconstruct more than 90% of the generated snapshots. The difference between the number of blocks collected and those disseminated, shown after 400 minutes, is equal to the number of dependent coded blocks collected by server.

In Figure 4, we show how the probing quantity Q_s affects the decoding efficiency. We fix the probing period T_s to 10 minutes. The parameter $\alpha(d)$ and β of the spreading factor are fixed to 50 and 0.25 respectively. Also the network size is set to 1000. Figure 4 reveals that the decoding is limited when the number of collected blocks is less than the number of generated blocks. We note a fast increase in the amount of decoded packets as $Q_s \times T_s$ approaches the number of blocks generated by the peers. A probing quantity Q_s equal to 800 is sufficient to allow the server reconstruct most of the traces. As we further increase Q_s, the number of decoded packets slightly increases but the amount of linear dependent blocks collected by the server increases significantly. Hence, a careful selection of probing quantity Q_s, can save the server bandwidth from dependent coded blocks.

Figure 5 demonstrates how the progressive encoding used in our trace collection protocol allows the server to reconstruct some snapshots from a segment received, even when it does not have enough blocks to decode all the segment. We report the number of decoded blocks collected from a randomly selected peer u, as the server is periodically probing the network. Figure 5(a) shows the distribution of the collected blocks from peer u as a function of the number of coefficients retrieved from the messages. On the other hand, Figure 5(b)

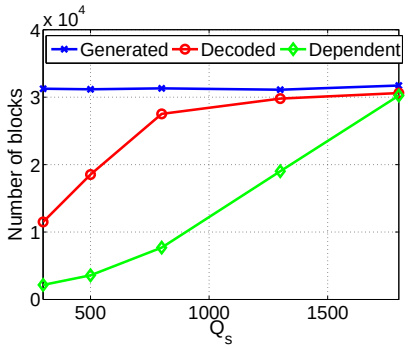
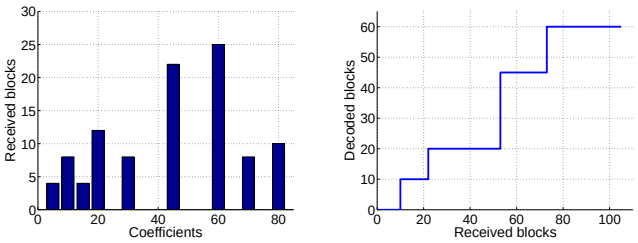


Fig. 4. Probing efficiency as a function of the periodic probing quantity Q_s.

shows how the number of decoded blocks increases as the server collects additional data. For instance, we observe that although the server collected 7 blocks containing 10 coefficients each, it was able to decode them using previously collected blocks containing only 5 coefficients. All the blocks reported in Figure 5 belong to the same segment. The server was able to decode 60 out of the 80 blocks forming the segment. Figure 5 shows that our protocol increases the diversity of the exchanged blocks through progressive encoding and allows the server to reconstruct snapshots from a segment without the need to completely decode it. Indeed, when Algorithm 1 is applied during a session, the size of the segments grows and the number of coefficients contained in the exchanged blocks increases. As the server periodically pulls data form the network, blocks containing different number of coefficients are used in the decoding process. As such, we avoid the problem of decoding all the segment or nothing.



(a) (b)

Fig. 5. Reconstructing blocks of a segment collected by the server.

Next, we fix the probing quantity Q_s to 800 and the probing period T_s to 10 minutes and investigate the effect of the peers' cache size on the protocol. We set the mean value of the exponential distribution that models peer dynamics to 80 minutes. Figure 6 shows that with a cache size of 100 KB the amount of decoded blocks is limited to 60% independent of the spreading factor. In fact, the peers that have a longer lifetime are supposed to cache the data blocks of the peers that have left the session in order for the server to pull their blocks in a delayed fashion. However, with a cache size of 100 KB the peers have to drop blocks previously buffered. The server fails to collect sufficient number of blocks to reconstruct additional snapshots under such rate of peer departures. On the other hand, observe that a cache size of 300 KB is sufficient to allow the server decode most of the blocks generated using an appropriate spreading factor. We note a slight decrease in the decoding efficiency for small cache sizes.

This is due to the fact that for large values of $\alpha(d)$, segments occupy peer caches more than needed by the server, creating additional dependency and preventing other generated blocks from getting buffered. In our protocol, progressive encoding decreases the number of segments injected in the network and through segment replacement mechanism it efficiently uses the neighbors' cache capacity.

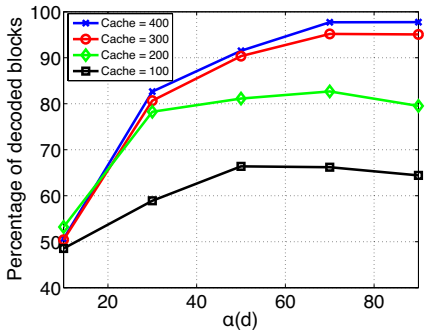


Fig. 6. The effect of cache size on the decoding efficiency.

B. Peer Dynamics Factor

Our trace collection protocol adapts to peer dynamics and controls the redundancy disseminated in the network. We study the effect of peer dynamics and show how our protocol can tolerate high level of peer departures. By modifying the spreading factor SF, based on their local view of neighbor dynamics, peers can determine the redundancy that should be disseminated in the network.

In order to reveal the relation between the spreading factor and the peer dynamics, we evaluate the percentage of decoded blocks under different values of the parameter e, as shown in Figure 7. The network size is set to 3000 peers. We note that under extreme level of peer dynamics, where e is equal to 30 minutes, the spreading factor parameter $\alpha(d)$ is required to be as high as 90 in order for the server to reconstruct most of the traces. With such rate of peer departures, the cached blocks losses prevent the server from reconstructing the generated snapshots. Hence, a segment should spread further in the network reaching more peers in order to tolerate such high level of peer dynamics. However, selecting a large value for $\alpha(d)$ under a low peer departures rate would result in many block dependencies at the server. Therefore, the spreading factor should be chosen according to the peer dynamics. In our protocol, peers determine the number of blocks to disseminate using Equation (1) which is a function of neighbors' dynamic percentage rate.

In the previous scenarios, the parameter $\alpha(d)$ did not depend on neighbor dynamics d. However, Figure 7 indicates that the peers should modify the spreading factor depending on the rate of peers' departure. Since a peer does not have a global knowledge of peer dynamics, it modifies its spreading factor based on the local view of neighbor dynamics. In our protocol, as the rate of neighbor departures augments, a peer increases the spreading factor of its segment to disseminate additional coded blocks in the network. As the blocks are disseminated further in the network, a peer can tolerate neighbor dynamics

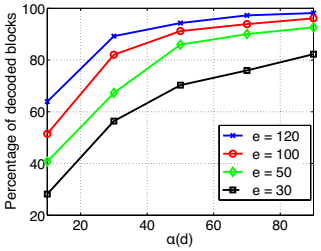


Fig. 7. Decoding efficiency as a function of the spreading factor under different level of peer dynamics. Network consists of 3000 peers.

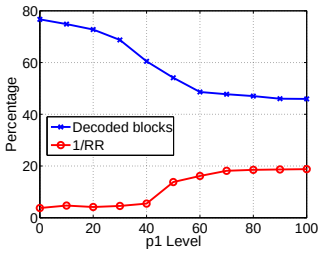


Fig. 8. Decoding efficiency and message intensity as a function of the dynamic sensitivity level.

and hence allow the server to collect its data and reconstruct its snapshots. The redundancy ratio RR of Equation (3) is an indicator of the level of redundancy injected in the network. In our simulations, we calculate RR by measuring the ratio of blocks disseminated in the network to blocks generated by the peers. We report in our results the percentage $\frac{1}{RR}$ where smaller values of $\frac{1}{RR}$ imply higher redundancy exchanged by the peers.

In the scenario of Figure 8, $\alpha(d)$ has a single sensitivity level p_1 at which the peers change their spreading factor according to Equation (8). The parameter β of SF is set to 0.25 and the peers vary $\alpha(d)$ depending on their neighbor departures rate. We fix the mean value e of the exponential distribution, that models peer dynamics, to 20 minutes.

$$\alpha(d) = \begin{cases} 50 & d <= p_1 \\ 80 & d > p_1 \end{cases} \quad (8)$$

We observe from Figure 8 that for small values of p_1, the server is able to reconstruct more snapshots. Indeed, when the peers are more sensitive to neighbor departures, they adjust their spreading factor faster and hence resist high level of peer dynamics by injecting additional coded blocks in the network. On the other hand, for large values of p_1, many coded blocks would be lost before the peers decide to adjust the spreading factor used. Note that when we reduce the sensitivity level p_1, the redundancy ratio increases. In fact, when p_1 is small, more peers would increase their spreading factor and hence increase the messaging intensity. Peers become more sensitive to their neighbor dynamics and more blocks are exchanged and cached in the network.

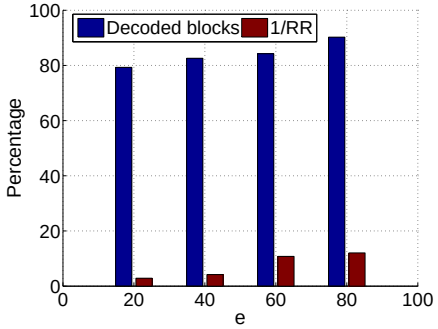


Fig. 9. Protocol's adaptability to peer dynamics in a network consisting of 5000 peers. The variable e is the mean of the peer dynamics distribution model.

Finally, we fix the sensitivity levels and vary the mean value e to study how our trace collection protocol adapts to high level of peer dynamics. For this purpose, we apply our protocol in a network consisting of 5000 peers under different levels of peer departures. Peers change the spreading factor they use according to Equation (9) and thus, adjust the messaging intensity.

$$\alpha(d) = \begin{cases} 30 & d <= 20 \\ 50 & 20 < d <= 35 \\ 80 & d > 35 \end{cases} \quad (9)$$

We report, in Figure 9, the decoding efficiency and the redundancy ration RR. We observe that the server is able to decode more than 80% of the generated snapshots independent of the mean value e. This demonstrates how our trace collection protocol adapts to peer dynamics. Depending on the neighbors departure rate, peers adjust the messaging intensity to allow the server reconstruct their snapshots. As such, the decoding efficiency does not drop even under an extreme level of peer dynamics with a mean value e equal to 20. We also note, from Figure 9, that the percentage $\frac{1}{RR}$ grows as the mean value e increases. Indeed, as neighbors become more dynamic, peers modify the spreading factor they use, hence, disseminate additional coded blocks of their segments in the network. The blocks spread further in the network in order to tolerate the losses due to peers leaving the session. As a result, the ratio of the number of blocks generated to blocks disseminated increases.

By adapting to neighbor dynamics, peers disseminate the appropriate amount of redundancy to resist the losses due to peers leaving the session. As such, they efficiently use the server bandwidth and reduce the number of collected blocks that are dependent. The ability to adapt to peer dynamics allows our protocol to collect the traces from the network under high rate of peers' departure and at the same time allows it to scale to large-scale peer-to-peer networks.

VII. Conclusion

In this paper, we have shown how random linear network coding can be used to help operators diagnose large-scale peer-to-peer networks. We presented a new trace collection protocol to monitor and diagnose large-scale peer-to-peer networks. We incorporate network coding to disseminate snapshots over the network. Peers cooperate by allocating cache capacity for snapshots from other peers. Since, the measurements are not time sensitive, the server periodically pulls a fixed amount of data from the network in a delayed fashion, in order to reconstruct the generated snapshots. We used progressive encoding to tolerate extreme level of peer dynamics and reduce storage cost. The protocol adapts to peer departures rate and controls the redundancy introduced in the network. The redundancy are coded blocks that allow the server decode the traces under extreme block losses. Through progressive encoding, we increase the blocks diversity as the traces are generated and efficiently use the storage capacity at the peers. We studied the performance of our trace collection protocol under different level of peer dynamics and demonstrated the benefits of progressive encoding. We used event-driven simulations to model the trace collection mechanism. Our protocol proved to scale and tolerate high level of peer departures. The results showed how it adapts to peer dynamics by disseminating the appropriate amounts of coded blocks in order for the server to reconstruct most of the traces. The protocol presented in this paper shows the benefits offered by random linear network coding in peer-to-peer architectures. We demonstrated how we can leverage the power of network coding once incorporated in our design.

References

[1] "UUSee Inc." http://www.uusee.com/.

[2] C. Wu and B. Li, "Echelon: Peer-to-peer network diagnosis with network coding," in *Proc. of IWQoS 2006*, New Haven, CT, USA, June 2006.

[3] D. Niu and B. Li, "Circumventing Server Bottlenecks: Indirect Large-Scale P2P Data Collection," in *Proc. of the 28th International Conference on Distributed Computing Systems (ICDCS 08)*, June 2008.

[4] 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.

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

[6] A. B. Downey, "Using Pathchar to Estimate Internet Link Characteristics," *SIGMETRICS Performance Evaluation Review*, vol. 27, no. 1, pp. 222–223, 1999.

[7] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson, "User-Level Internet Path Diagnosis," *SIGOPS Operating System Review*, vol. 37, no. 5, pp. 106–119, 2003.

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

[9] 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, 2000.

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

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

[12] T. Ho, R. Koetter, M. Medard, D. Karger, and M. Effros, "The Benefits of Coding over Routing in a Randomized Setting," in *Proc. of IEEE International Symposium on Information Theory (ISIT 2003)*, 2003.

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

[14] Y. Wu, "Network Coding for Multicasting," Princeton University, Tech. Rep., Nov. 2005.