

Maximizing the Data Utility of a Data Archiving & Querying System through Joint Coding and Scheduling

Junning Liu*, Zhen Liu†, Don Towsley* and Cathy H. Xia†

*Dept. of Computer Science, University of Massachusetts, Amherst, MA

Email: {liujn,towsley}@cs.umass.edu

†IBM Thomas J. Watson Research Center, Hawthorne, NY, USA

Email: {zhenl,cathyx}@cs.umass.edu

Abstract

We study a joint scheduling and coding problem for collecting multi-snapshots spatial data in a resource constrained sensor network. Motivated by a distributed coding scheme for single snapshot data collection [7], we generalize the scenario to include multi-snapshots and general coding schemes. Associating a utility function with the recovered data, we aim to maximize the expected utility gain through joint coding and scheduling.

We first assume *non-mixed coding* where coding is only allowed for data of the same snapshot. We study the problem of how to schedule (or prioritize) the codewords from multiple snapshots under two classes of utility models: an *archiving model* where data from all snapshots are of interests with additive utilities; and a *staleness model* where the most recent data of each sensor location is desired with non-additive utilities. For the archiving model, we formalize the scheduling problem into a Multi-Armed Bandit (MAB) problem. We derive the optimal solution using Gittins Indices, and identify conditions under which a greedy algorithm is optimal. For the staleness model, the scheduling problem is a Restless Bandit (RB) problem. We show that a greedy scheduling policy is optimal when the inter-arrival time between snapshots is not too short.

We then consider *random mixed coding* where data from different snapshots are randomly coded together. We generalize the growth codes in [7] to arbitrary linear-codes-based random mixed coding and show that there exists an optimal degree of coding. Various practical issues and the buffer size impact on performance are then discussed.

I. INTRODUCTION

In many distributed data streaming systems, we need to deal with two basic issues: the random environment and resource constraints. The random environment view is normally due to the lack of perfect information in a dynamic environment, e.g. random changing source and sink topologies in Delay Tolerant Networks (DTN) [5], random nodes failures in disaster monitoring networks [7], random fluctuations of link qualities in wireless networks, etc. This environment randomness motivates the employment of network coding [2], [1], [7]. Essentially, network coding helps to alleviate the coupon collector effect¹ [12]. On the other hand, we often have limited processing resources (communication, computation, storage, energy, time etc.), thus we are only able to recover some fraction of all the temporally and spatially labelled data. This raises the problem of resource allocation or scheduling among the sensed data.

In this paper, we study the interplay between coding and scheduling to improve data efficiency. We associate a *utility* function with the recovered data² and analyze the coding & scheduling schemes that maximize the total

¹Coupon collector effect refers to the long tail phenomena in collecting N distinct coupons by random sampling, it needs $N \log N$ samplings on average.

²We refer to the data decoded by the sink as recovered data and using recovering interchangeably with decoding.

utility associated with the recovered data subject to a limited communication resource budget. In general, the data is generated at multiple time stamps with different user specified utilities.

Our problem of maximizing data utilities through joint coding and scheduling is motivated by many real time data collecting & querying networks. One typical setting is that a number of sensor nodes are randomly deployed in a region to monitor disaster events such as earthquakes, fires, floods, etc. When a disaster happens, nodes may fail at any time. There are randomly deployed base-stations (sinks) in the network to collect the data. The sinks' number is much less than that of the sensors and their locations are unknown to the sensor nodes due to the random failures of both the nodes and transmissions. Thus, sensors have to use random gossiping type of routing. The goal is to recover as much data as possible before the whole network goes down.

Another setting common in practice is in the context of querying/stream-processing networks, where various types of data streams are generated online and transmitted to the backend base-stations for further processing or storage. Data are associated with different spatial & temporal labels and have different utilities to the application. For example, some applications may request data of all time stamps within a particular time window for archiving purposes, although the utilities of data from different time stamps may not be weighted uniformly. Some other applications may favor the most recent data, a more recent data for the same spot may cause the old one to become useless. Also data generated in a particular region at a particular time may have high utilities due to purposes such as event monitoring or target tracking.

In all above settings, the rate at which the network generates data may greatly exceed the communication capacity of the network. We assume the data samples are independent either by nature or pre-compressing. In wireless networks, for example, the bandwidth is often limited due to the wireless medium contention. The funnel effect [16][11] of bottleneck links around the sinks due to the “many to one” type of communication patterns causes the effective capacity to be even lower. The energy power is also limited especially for low cost cheap sensor networks. One other scenario in e.g. disaster monitoring networks, is that the network is not stable, nodes are prone to fail thus the communication resources left keep shrinking as time goes on. Another typical scenario is military communications in the combat field, parties want to turn on their radios as less as possible to reduce the chances of exposure to the adversaries. For all these applications, with the limited resource budget, there is a need to prioritize the forwarding of data of multiple time stamps, and decide what subset of data can be encoded together so as to maximize the data utility gain.

Without loss of generality, we assume all actions (sensing and transmitting) are taken in discrete time slots. A *snapshot* refers to the collection of data samples that are generated in the same time slot. To account for the various needs of different applications, we consider two generic data utility models: one is an *archiving* model where data of all snapshots are of interests with additive utilities, in other words, data of two snapshots about the same location both have utilities independent of each other; the other one is a *staleness* model where the most recent data is desired for each sensor location and the utilities are non-additive across different snapshots since obtaining an earlier snapshot data brings less utility if more recent snapshot data from the same location has already been recovered.

The most general setup that allows arbitrary coding and scheduling is a very complex open problem. In the presence of multiple locations and multi-snapshots, there is a huge number of possible spacial and temporal snapshot combinations to choose from. The coding and scheduling operations can be entangled together and it becomes difficult to determine the optimal solution. A greedy policy that simply delivers the current most “valuable”³ piece of data to the base-stations may not be optimal. As a first step, we focus on two generic coding schemes for simplicity of analysis. Namely, we consider *non-mixed coding*, where coding is only allowed for data of the same

³The utility may not be associated with any particular piece of data but rather a nonlinear function of a collection of data, also the random environment of such data retrieval system prevent precise control of which data is actually recovered (e.g. with random network coding).

snapshot, and a *random mixed coding* where data from different snapshots are randomly coded together.

Under an arbitrary non-mixed coding scheme, there is a need to schedule (or prioritize) the codewords from multiple snapshots so as to maximize the total utility gain. We show that under the archiving model, the scheduling problem can be modelled as a Multi-Armed Bandit (MAB) problem [17] and solved optimally using Gittins Indices [6], we also give an algorithm to calculate the indices specific to our underlying Markov process. We further show that greedy is optimal when a monotonicity property of the product of decoding probability and utility holds. Under the staleness model, the scheduling problem is a Restless Bandit (RB) problem [13] which is in general NP-hard (P-Space Hard [14]). We show that when the data arrival rate is not so large such that newer data will have at least twice the ‘utility’⁴ of the older data, a greedy algorithm that always favor the most recent snapshot is the optimal policy.

For random mixed codings, there is no scheduling issue since different snapshots data is drawn randomly. For linear codes based random mixed coding, we characterize the optimal coding *degree* (number of original data symbols to code together) to maximize the decoding probability at all the times. Based on this, we show that a greedy algorithm that always chooses the optimal coding degree is optimal. This is a generalization of [7]’s work where they characterize the optimal coding degree code called *growth code (GC)* within a class of random linear network codes for the single snapshot case.

In reality low cost sensors often have a constrained memory size to apply the joint coding & scheduling operations. So we also analyze the buffer size’s impact on the sensors’ coding strategies. We claim that the non-mixed coding is appropriate for sensors with relatively large buffer size while the random mixed-coding is good for low cost sensors with very limited buffer size and computation power. For growth codes applied in the single snapshot case, we show that performance is not sensitive to the buffer size in terms of the number of packets exchanged to recover the whole snapshot. The total expected number of packets to recover the whole snapshot for growth code is $\Theta(N)$ while for the non-coding approach it is $\Theta(N \log N)$ with N as the total number of data items in that snapshot. Our simulation shows that the expected number of packets is less than $2N$ even for sensors with very small buffers.

II. BACKGROUND AND RELATED WORK

The most relevant work to this paper is [7]. It considers a distributed sensor network environment for the purpose of disaster monitoring. Sensors are prone to fail at any time and have no knowledge of the sink location. [7] studies the data collection of a single snapshot with a *max data persistence* goal, aiming at maximizing the fraction of data that eventually reaches the sink. Without coding, data collection corresponds to a coupon collector process [12] where it takes in total $N \log N$ number of packets in order to recover N original data packets. Coding can help to combat the heavy tail coupon collector effect. Existing techniques include Reed-Solomon codes [8], Luby Transform Codes [9], Digital Fountain [4], Turbo Codes [3] and other LDPC codes [10]. However, all these traditional channel codes require accumulating a large number of initial codes before decoding, which is not desirable when resource is limited and nodes are subject to failure at anytime. The key idea of Growth Codes [7] is to intelligently control the coding degree (the number of symbols to code together) to maximize the decoding probability for each additional symbol reaching the sinks. In the beginning, data is not coded, when half of the original symbols are recovered, symbols are coded together with a degree 2, and the degree keeps increasing as the recovered fraction increases. The optimal coding degree is derived under a simplified centralized encoder model and then implemented in a distributed fashion.

There are two issues not considered in [7] that we want to address. First, they consider only a single snapshot; second, the spatial and temporal heterogeneity of the data is not considered, there is no notion of utilities associated with the data. In reality, events being monitored could happen across different times and each of them has some

⁴This is an ‘absolute utility’ we defined later for convenience of analysis.

application defined utilities to the end users. Even for one snapshot, different locations may be associated with different utilities depending on the users' interests. For such a data collection process with multi-snapshots and multiple utility levels, there is a need to combine coding and scheduling so as to maximize the systems utility.

One technique that we borrow is from the rich literature of classical Multi-Armed Bandit(MAB) problems. Details on the methodologies relevant to this class of problems can be referred to, e.g. [6], [18], [15], [13], etc. We will review necessary results along the discussion when they become needed.

The rest of the paper is organized as follows. The model under consideration is formalized in Sec. III. The scheduling problem under the archiving model and staleness model are studied in much detail respectively in Sec. IV and V. The optimal random mixed coding is explored in Sec. VI where the objective is to maximize the immediate return for next transmission. The buffer size impact on performance is addressed in Sec. VII. Some practical implementation issues are discussed in Sec. VIII. Concluding remarks and future works are finally given in Sec. IX.

III. MODEL FORMALIZATION

Throughout the paper we assume time is discretized into equal length slots, indexed $t = 1, 2, \dots$. There are N sensor nodes independently and randomly distributed in a connected region. Sensors are assumed to be synchronized, and at time t_i , each sensor $n = 1, \dots, N$ generates observation $Y_i^{(n)}$ (also referred to as data sample) of the field at its location. We assume the data samples are independent and refer to the collection of all observations $\vec{Y}_i = \{Y_i^{(1)}, \dots, Y_i^{(N)}\}$ at time t_i as the i -th snapshot of the field. We assume the interarrival times between two snapshots $t_i - t_{i-1} (\geq 1)$ are i.i.d. random variables.

Each data sample has b bits. We refer to this b -bit data unit as a *symbol* (or a packet), the original b -bit data sample as an *original symbol*, and a coded data of b -bits as a *coded symbol* (or a codeword). Each node has a buffer of size B that can store at most B data symbols.

There are l sinks that are randomly distributed in the network with their locations unknown to the sensor nodes. There are communication links among neighboring sensors with constrained bandwidths. The sinks aim to maximize the total utility (define concretely later) of the collected data for further processing. The majority of the nodes cannot transmit directly to the sink, and hence other nodes must act as intermediate forwarding agents of data. We assume the network is 'zero-configuration', the only topology information available to a node is its set of neighbors with whom it can communicate directly. At each time slot, a node can transmit b -bits to a randomly chosen neighbor. It can transmit to a neighbor either one original data symbol, or an encoded version of some subset of the symbols in its buffer. The sinks are assumed to be either wired up using high speed networks or have very large capacity links among them so that once the data reach any of the sinks it is considered being collected by a central server. We assume that each sink receives one packet per time slot on average. Upon receiving the encoded packets, the sinks may be able to decode a subset of the original symbols with certain probability depending on the amount of data that has already been recovered.

Since the rate at which the network generates data may greatly exceed the communication capacity of the network, there is a need to prioritize the forwarding of the multiple snapshots, and to decide what subset of data can be encoded together so as to increase delivery efficiency. We define a joint coding and scheduling scheme Ω as a uniform operation scheme for all sensor nodes that tells a node what to encode and which snapshot to send to its neighbors. In general, Ω can use any possible coding scheme, e.g., a node can select $d_1 \geq 0$ symbols from snapshot 1, $d_2 \geq 0$ symbols from snapshot 2, \dots , $d_j \geq 0$ symbols from snapshot j and coded them together.

Denote $X_t = \{Y_i^{(n)} | t_i \leq t, 1 \leq n \leq N, Y_i^{(n)} \text{ is decoded}\}$ to be the set of symbols recovered by the sinks at time t , and its sample path $X_{[1,t]} = \{X_s | 0 \leq s \leq t\}$. We assume there is a utility function associated with the sample

path of the recovered data set $U(X_{[1,t]})$. The goal is to find a joint coding and scheduling scheme Ω to maximize the expected total utility within a given time period T , i.e.

$$\max_{\Omega} E [U(X_{[1,T]})] .$$

In general U can be specified arbitrarily by the underlying user/application. In this paper, we focus on two specific utility models: *archiving model* and *staleness model*. In the archiving model, we assume users are interested in all data belong to a specific subset of snapshots, where different snapshots may have different importance (utilities) and data utilities of different snapshots are additive (independent). Such a scenario is typical in many archiving and querying applications, where all data of different time stamps within a given time window are of interests. The staleness model is also desired in many applications for purposes such as prediction. In this case, users would like to obtain the most recent observation of each location in the sensor network, thus the utilities are no longer additive. A piece of data may bring less utility if there is more recent snapshot data of the same location already recovered.

The most general setup that allows arbitrary coding and scheduling is a very complex problem. As a first step, we assume coding is only permitted for symbols from the same snapshot and call it as *non-mixed coding*. Under any non-mixed coding scheme, we study the problem of how to schedule (prioritize) the transmission of codewords from multiple snapshots so as to achieve the max system utility over a fixed time window. The archiving problem and staleness problem are studied respectively in the next two sections. We also apply these general results to the GC based non-mixed coding and discuss the specific solutions in Sec. 7.3.

IV. ARCHIVING PROBLEM

In this section we consider the archiving problem where users are interested in all snapshots with additive data utilities. We assume new snapshots & queries keep arriving and the problem is how should we schedule the data gathering to adapt to the dynamic utility map caused by the new snapshots & query updates. For example, an emergent query may ask for the three most recent snapshots' data in the next ten minutes, possibly with different utility preferences. We model this problem by the classic MAB problem and obtain the optimal scheduling policy.

A. Additive Utility Goal

Under an arbitrary non-mixed coding scheme, the utility gain at time t depends on the number of symbols one can decode at time t and the number of already recovered symbols before t , denoted respectively by $z_i(t)$ and $r_i(t)$ for snapshot i . In general, $r_i(t)$ is a random variable dependent on the coding & scheduling scheme Ω , and $z_i(t)$ is a random variable dependent on $r_i(t)$ and Ω . We use $u_i(r_i(t))$ to represent the utility gain of decoding a snapshot i symbol when r_i of them has been recovered. We further model the time influence on utilities using a time discounting factor $0 \leq a \leq 1$ to model the decay of data utility along with time. That is, all snapshot data's utility drops out exponentially after generated.⁵ Based on above, we can associate each snapshot i symbol decoded at time t a marginal utility gain $a^{t-t_i} u_i(r_i(t))$ where t_i is the time the snapshot i symbol is first generated. In other words, a new decoded symbol's contribution on utility is simply a function of the number of decoded symbols of that snapshot multiplying a discount factor. Therefore, if Ω schedules to transmit only codewords of snapshot i at time t , the expected utility gain for this time slot is: $\mathbf{E} a^{t-t_i} z_i(t) u_i \left(r_i(t) \right)$, where t_i is the i th snapshot generating time, $r_i(t)$ is the number of recovered symbols of snapshot i by time t .

⁵Note that when $a = 1$ there is no decaying.

As we mentioned earlier, the general scheme with arbitrary codings is hard to analyze, we focus on *incremental decoding* that is also considered in [7] where a packet will not be kept for later use if the sinks can not decode a new symbol out of it immediately.⁶ It is easy to see that for incremental decoding, the sinks either decode nothing or decode one symbol from one received packet. Denote the probability of decoding one symbol as $p_{r_i(t)}$, it is merely a function of the number of recovered symbols. W.l.o.g., we assume time is slotted fine grained enough s.t. in each slot t , any scheme Ω simply choose to schedule one of the snapshots $1 \leq \Omega_t \leq k$. Since each sink receive one packet per time slot, $\mathbf{E}z_{\Omega_t}(t) = lp_{r_{\Omega_t}(t)}$, or simply $\mathbf{E}z_{\Omega}(t) = lp_{r_{\Omega}(t)}$. Note that this is a conditional expectation conditioned on $r_{\Omega}(t)$ which is also a random variable dependent on Ω . For later convenience, we further absorb the a^{-t} term into u_i as a parameter. Based on above, for an arbitrary non-mixed coding scheme, we are interested in the best scheduling policy that maximizes the long-run average utility, i.e.

$$\max_{\Omega} \frac{1}{T} \mathbf{E} \sum_{t=1}^T a^t lp_{r_{\Omega}(t)} u_{\Omega} \left(r_{\Omega}(t) \right) \quad (1)$$

where $u_{\Omega}(r_{\Omega})$ is the undiscounted utility of decoding a snapshot Ω_t symbol and Ω_t is the index of the snapshot scheduled for time t by Ω .

B. Multi-armed Bandit Solution

A multi-armed bandit (MAB) problem is a scheduling problem for the operations among a number of machines. Each machine has an initial state and we denote the i th machine's initial state as $x_i(1)$. At any time t , we can choose one of the machines i to execute and obtain a reward $R(x_i(t))$, which is a function of the machine's state. As we mentioned earlier in general there is also a discount factor $0 \leq a \leq 1$ s.t. the actual reward we collect for operating machine i at time t is $a^t R(x_i(t))$. If a machine is chosen to be executed on, then its state evolves following some stationary Markov process. If not, the state remains unchanged $x_i(t+1) = x_i(t)$. The goal is to schedule the operations s.t. the expected reward collected within a time window T is maximized. The optimal policy is to, at each time step, execute the machine with the maximum Gittins index[6]. The Gittins Index is defined as:

$$v_i(x_i) = \max_{1 < \tau \leq T} \frac{\mathbf{E} \left(\sum_{t=1}^{\tau-1} a^t R_i(x_i(t)) | x_i(1) = x_i \right)}{\mathbf{E} \left(\sum_{t=1}^{\tau-1} a^t | x_i(1) = x_i \right)} \quad (2)$$

where x_i is the machine i 's current state. In the case of $T = \infty$, we just need to recalculate every step the index of the machine that has been just operated because other machines' indices remain unchanged.

Our archiving and querying problem can be modelled as a MAB in the following way. Each snapshot corresponds to a machine, to execute on a machine corresponds to exchange packets coded with symbols from that snapshot for one time step. Thus if at time t we have $S(t)$ snapshots not finished retrieving, then we have $S(t)$ machines to choose to operate. Let the arrival time of snapshot i be t_i , as mentioned earlier in (1) we absorb the a^{-t_i} term into the utility before discounting term $u_i(r_i)$, then every snapshot virtually starts discounting from the same time, bring a actual utility gain as $a^t u_i(r_i)$. When there are query updates that changes the utilities, we can always adjust $u_i(r_i)$ appropriately to reflect them.

⁶Even though in general it is possible to apply complex techniques to use cached packet to do joint decoding, for the simplicity of the analysis and the assumption of limited computation capability for the sensors, we assume those packets that are not decodable right away will be discarded. Also the benefit of keeping those packets is expected to be negligible compared to the effect of the degree optimization.

We then represent the state of each snapshot with the total number of symbols already recovered at current time $r_i(t)$ or simply r_i . In general we could apply different codes upon different snapshots. In that case the only difference between their state transition process is the transition probabilities, we still calculate the Gittins index with the same procedure. W.l.o.g., we assume the same type of coding is applied separately to all snapshots, i.e. they all share the same Markov state transition process described in the diagram of Fig. 1.

State index: number of symbols recovered \textcircled{r}

Marginal utility reward for snapshot i : $u_i(r_i)$

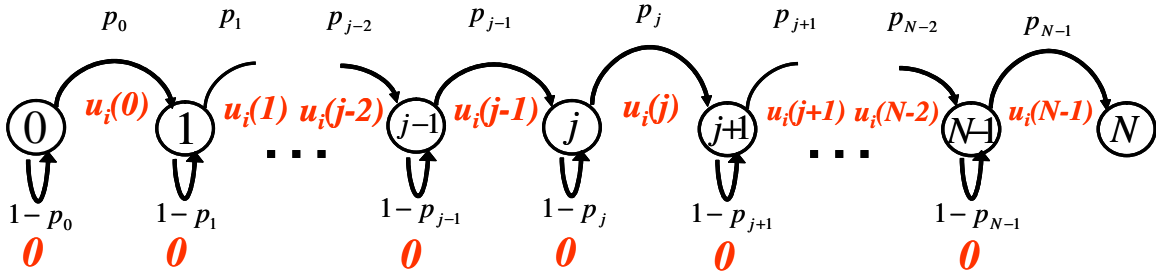


Fig. 1. The Markov states transition process

As we can see, the i th snapshot's state can be indexed with the number of recovered symbols $0 \leq r_i \leq N$, and the transitions are only between states with adjacent indices. With decoding probability p_{r_i} state r_i transits to $r_i + 1$, with probability $1 - p_{r_i}$ the state remains in r_i . Unlike the normal MAB problem, our reward is associated with the transitions instead the states. For each decoding transition of snapshot i , the sinks collected the symbol's marginal utility gain of $u_i(r_i)$ as the reward. We denote the Gittins index of a snapshot in state r as v_r .

1) *Calculating the Gittins Index:* (2) provides a general rule for calculating the index, however, the special state structure and Markovian process of our problem actually make it possible to further simplify the index calculation as shown in the following theorem.

Theorem 1: For any non-mixed coding, snapshot i 's Gittins index as a function of its state (the number of recovered symbols r) can be obtained by the following computation procedure,

$$i_1 = \operatorname{argmax}_{0 \leq r \leq N} p_r u_i(r), \quad v_{i_1} = p_{i_1} u_i(i_1),$$

$$\alpha_{i_1} = \frac{a p_{i_1} u_i(i_1)}{1 - a(1 - p_{i_1})}, \quad \beta_{i_1} = \frac{a}{1 - a(1 - p_{i_1})}.$$

Let $S_m = \{i_1, i_2, \dots, i_m\}$, $S_m(1) = \{r | r + 1 \in S_m, r \notin S_m\}$. Calculate the following from $m = 2$ to N .

If $\max_{r \notin S_{m-1}} p_r u_i(r) > \max_{r \in S_{m-1}(1)} p_r \frac{(u_i(r) + \alpha_{r+1})}{1 + p_r \beta_{r+1}}$, then

$$i_m = \operatorname{argmax}_{r \notin S_{m-1}} p_r u_i(r), \quad v_{i_m} = p_{i_m} u_i(i_m),$$

$$\alpha_{i_m} = \frac{a p_{i_m} u_i(i_m)}{1 - a(1 - p_{i_m})}, \quad \beta_{i_m} = \frac{a}{1 - a(1 - p_{i_m})}.$$

Otherwise

$$i_m = \operatorname{argmax}_{r \in S_{m-1}(1)} p_r \frac{(u_i(r) + \alpha_{r+1})}{1 + p_r \beta_{r+1}},$$

$$v_{i_m} = u_i(i_m) p_{i_m} \frac{(u_i(i_m) + \alpha_{i_m+1})}{1 + p_{i_m} \beta_{i_m+1}},$$

$$\alpha_{i_m} = \frac{a p_{i_m} (u_i(i_m) + \alpha_{i_m+1})}{1 - a(1 - p_{i_m})}, \quad \beta_{i_m} = \frac{a(1 + p_{i_m} \beta_{i_m+1})}{1 - a(1 - p_{i_m})}.$$

Proof: The proof follows the same spirit of Lemma 4.1, 4.2 and Theorem 4.1 in [15]. Let

$$\alpha_r = \mathbf{E} \sum_{t=1}^{\tau^*-1} a^t R(x(t)), \quad \beta_r = \mathbf{E} \sum_{t=1}^{\tau^*-1} a^t.$$

where τ^* is the optimal stopping time that achieves the index value of state r , it is a random variable conditioned on the initial state.

Our Markov process is different from the normal MAB problem where the reward is a deterministic function of the state. In our problem, the reward is a deterministic function of the transitions; when we transit from state r to $r+1$, or to say when we decode a new symbol, we gain utility $u_i(r_i)$ associated with the symbol. By the one step memoryless property of a Markov process and the special structure of our state process, we derive

$$\alpha_r = (1 - p_r) a \alpha_r + p_r (a u_i(r) + a \alpha_{r+1}),$$

$$\beta_r = (1 - p_r) a (1 + \beta_r) + p_r (a + a \beta_{r+1}).$$

Then apply Theorem 4.1 in [15] and we prove the theorem. ■

Queries are modelled by updates of the utility vector $u_1, \dots, u_{S(t)}$. For example, in a military sensor application, at time t a General may want get as much data of snapshot j to $j+h$ as possible in the next ten minutes, and also specifies the relative preference level of u_j to u_{j+h} as $c_j : c_{j+1} : \dots : c_{j+h}$. This emergent query is carried out by setting all $u_i = 0$ for $i \notin \{j, j+1, \dots, j+h\}$ and $u_j = c_j, u_{j+1} = c_{j+1}, \dots, u_{j+h} = c_{j+h}$. In addition to this, when calculating the Gittins index, instead of finding an optimal stopping time of infinite horizon, we need to find an optimal stopping time within ten minutes.

In some applications we may have some knowledge of the snapshot arrival process. We may know the arrival rate through past experience or some online learning/estimation techniques. And because of this, if the scheduling algorithm is searching the maximum index that could have its stopping time that achieves that index anywhere in the future, it may not be an optimal algorithm conditioned on the knowledge of the arrival process. For example, if we know some high utility snapshot is going to arrive within 5 minutes, then at current time we should not choose a snapshot with a max index achieved at time 1 hour later over a snapshot with the second largest index achieved within 5 minutes. However, if the product of the decoding probability and the utility is monotonically non-increasing, the following Lemma tell us a greedy algorithm is optimal.

Lemma 1: If $p_{r_i} u_i(r_i)$ is monotonically non-increasing as r_i increases for all snapshot i , then $v_i = p_{r_i} u_i(r_i)$ for all i and no matter what the new snapshots arriving process is, the optimal scheduling policy will be a greedy one that always chooses the current maximum index of $p_{r_i} u_i(r_i)$.

Proof: If there are no arrivals, by [17] the optimal policy will be the greedy policy. If there are arrivals, the only possible impact of the arrivals to the current machines' indices is to influence their time ranges to choose a stopping time to maximize the index. When the decoding probability is monotonically decreasing, the stopping time

that achieves the index is the minimum possible, one step. Thus, no matter what the arrival process is, a greedy policy will always produce an optimal performance incrementally. ■

The monotonicity property of the decoding probability is often true in reality, because in general the more symbols recovered, the more difficult it is to decode new symbols. For example in the growth codes case, even though the decoding probability may not be monotonically decreasing all the way to the end for the ideal case of growth codes with large buffer sensors, it is true for the recovered fractions between 0% to 97% under growth codes. Since the resources are limited, very often actually we are not able to recover more than 97% fraction of any snapshots due to new arrivals and limited-time queries. For many applications, the marginal utility gain $u_i(r_i)$ is also non-increasing for most of r .

Thus, greedy is in fact optimal for many real applications. We can also claim that greedy achieves the optimal max data persistence goal for lots of applications.

V. STALENESS PROBLEM

As mentioned earlier, applications may favor more recent data for purposes such as prediction or event monitoring. In this case, older data becomes useless once we have a more recent value for any spatial point. We model the data utility for such applications using a notion of staleness. The staleness of such data increases as time passes, but whenever we obtain fresher data, the staleness is reduced.

Now the utilities of various snapshots are no longer independent, the possible utility of an exchanged coded packet of a snapshot will not only depend on the recovered fraction of this snapshot, but also depend on the recovered fractions of other snapshots. When the machines' states are not independently evolving, this is a restless bandit problem rather than MAB problem. RB is in general NP-hard so there is no efficient solution. Nevertheless, we show that greedy is optimal under certain conditions of the decoding probability and inter-arrival times between adjacent snapshots.

A. Formulation of Staleness Problem

For the staleness problem, we use utility to quantify the staleness reduction of the spatial cells being monitored. The staleness of a sensor reading is a monotonically non-decreasing function of the time t , denoted by $\varphi(t)$. Thus we can associate an absolute utility value to each snapshot i as:

$$\mu_i(t) = \varphi(t) - \varphi(t - t_i). \quad (3)$$

$\mu_i(t)$ can be interpreted as the staleness reduction of obtaining snapshot i 's data at time t on that cell if that is the first symbol recovered for that cell.

Denote by $\delta_i(t)$ the fraction of snapshot i symbols that are already decoded by the sinks. For simplicity of presentation, let k equals $S(t)$, the current number of available snapshots. Then, for an arbitrary location, its staleness at time t is $\varphi(t - t_i)$ with probability $\delta_i(t) \prod_{j=i+1}^k (1 - \delta_j(t))$. That is, the data sample of snapshot i must have been decoded for that location, but the data of more recent snapshots have not been decoded yet for that location.

Combine further with the time discounting factor $0 \leq a \leq 1$, the staleness minimization can then be written as

$$\min_{\Omega} \frac{1}{T} \mathbf{E} \sum_{t=1}^T a^t N \sum_{i=1}^k \varphi(t - t_i) \delta_i(t) \prod_{j=i+1}^k (1 - \delta_j(t)) \quad (4)$$

We call this the *min integrated staleness problem*. Another possible goal is to minimize the final staleness at the stopping time T , i.e.

$$\min_{\Omega} \mathbf{E} N \sum_{i=1}^k \varphi(T - t_i) \delta_i(T) \prod_{j=i+1}^k \left(1 - \delta_j(T)\right) \quad (5)$$

we call this the *min staleness problem*.

For the min integrated staleness problem, when $\mu_i(t) = c(t_i)$ is a constant value independent of time t (some function of t_i). (4) can be equivalently written as

$$\begin{aligned} \max_{\Omega} \frac{1}{T} \mathbf{E} \sum_{t=1}^T a^t N \sum_{i=1}^k i(t) \left[\mu_i(t) \left(1 - \sum_{j=i+1}^k \delta_j(t) \prod_{\nu=i+1}^{j-1} (1 - \delta_{\nu}(t))\right) \right. \\ \left. - \sum_{j=1}^{i-1} \mu_j(t) \delta_j(t) \prod_{\nu \geq j, \nu \neq i} (1 - \delta_{\nu}(t)) \right] (T - t) \end{aligned} \quad (6)$$

where $\mu_i(t)$ is specified by (3). This is a specific Restless Bandit (RB) problem which is in general NP-hard.

B. Optimal Solution

Definition 1: A non-mixed coding satisfies *monotonic decoding* if its decoding probability is monotonically non-increasing (i.e. $p_{r_1} \geq p_{r_2}$ for all $r_1 \leq r_2$) and $p_r \geq 1 - \frac{r}{N}$ for all possible r .⁷

Definition 2: We call a snapshot arriving process *infrequent* if $\mu_j(t) \geq 2\mu_{j-1}(t)$ for all $t \geq t_j$ and $j > 1$.

Theorem 2: For the min staleness problem, a greedy scheduling of always exchanging coded bits from the most recent unfinished snapshot is optimal for any monotonic decoding non-mixed coding schemes with infrequent arrival snapshots.

To prove Theorem 2, we need to first prepare some lemmas. We will derive these lemmas based on a simplified scenario for simplicity of presentation then generalize them. Assume $\mu_i(t)$ is a constant of t and there are only two snapshot 1 and 2 with $t_1 < t_2$. Denote a policy that sticks to snapshot i all the time as Ω_i , and denotes the corresponding final staleness of a policy Ω as P_{Ω} .

Lemma 2: $P_{\Omega_2} \leq P_{\Omega_1}$.

Proof: The one step expected staleness reductions for Ω_1 and Ω_2 at time $t \geq 0$ are

$$\begin{aligned} \tilde{\mu}_2(t) &= N p_{r_2(t)} \left[\left(1 - \delta_1(0)\right) \mu_2(t) + \delta_1(0) \left(\mu_2(t) - \mu_1(t)\right) \right] \\ &= N p_{r_2(t)} \left(\mu_2(t) - \delta_1(0) \mu_1(t) \right) \\ &\geq N p_{r_2(t)} \left(2 - \delta_1(0) \right) \mu_1(t) \end{aligned} \quad (7)$$

$$\tilde{\mu}_1(t) = N p_{r_1(t)} \left(1 - \delta_2(0) \right) \mu_1(t) \quad (8)$$

At time zero, $p_{r_2} \geq (1 - \delta_2(0))$, $2 - \delta_1(0) \geq 1 \geq p_{r_1}$, so $\tilde{\mu}_2(0) \geq \tilde{\mu}_1(0)$. Since when $\mu_i(t)$ is a constant, $P_{\Omega_j} = N\varphi(t) - \sum_{t=1}^T \tilde{\mu}_j(t)$, we just need to show that $\tilde{\mu}_2(t) \geq \tilde{\mu}_1(t)$ for all t . This is true for the following three cases:

⁷Note that $1 - \frac{r}{N}$ is the naive non-coding approach's probability of decoding a new symbol.

a) When $r_2(0) = r_1(0)$, the two policy Ω_2 and Ω_1 have $p_{r_2(t)} = p_{r_1(t)}$ for any t . b) When $r_2(0) > r_1(0)$, easy to see $p_{r_2(t)} \geq p_{r_1(t)}$ for all t . c) When $r_2(0) < r_1(0)$, since p_r is monotonically non-increasing and $\frac{dr}{dt} = p_r$ we have $\frac{p_{r_2(t)}}{p_{r_1(t)}}$ increases with t , thus $\frac{p_{r_2(t)}}{p_{r_1(t)}} \geq \frac{p_{r_2(0)}}{p_{r_1(0)}}$.

Combined with (7),(8) and $\tilde{\mu}_2(0) \geq \tilde{\mu}_1(0)$ we show $\tilde{\mu}_2(t) \geq \tilde{\mu}_1(t)$ for all cases and any t thus finish the proof. \blacksquare

Lemma 3: Any scheduling policy that switches the snapshot exactly once can not perform better than Ω_2 .

Proof: Denote a scheduling policy that starts with snapshot 2 then switch to snapshot 1 as Ω_{21} and one that starts with 1 switches to 2 later as Ω_{12} .

By Lemma 2, we immediately see $P_{\Omega_{21}} \geq P_{\Omega_2}$ because after the switching point Ω_2 is consistently better than Ω_{21} while before that they are the same.

For Ω_{12} , denote the switching time as t_s , then we construct another policy Ω'_{21} that operates with snapshot 2 first from time 0 to $T - t_s$ then switches back to snapshot 1 till T . Since the coding operations between the two snapshots are independent in non-mixed coding, their orders do not affect the final $\delta_1(T)$ and $\delta_2(T)$. Thus $P_{\Omega'_{21}} = P_{\Omega_{12}}$. Compare Ω'_{21} with Ω_2 , by the same argument for Ω_{21} and Ω_2 above, we conclude $P_{\Omega_2} \leq P_{\Omega'_{21}} = P_{\Omega_{12}}$. \blacksquare

Lemma 4: Any scheduling policy that has more than one switchings performs no better than Ω_2 .

Proof: We shows this by recursively applying Lemma 3. Starting from the second last switching time, by Lemma 3, we can improve the performance from this point to T , by removing at least the last switching (possibly together with the second last switching). Work backward in time till time *zero*. By induction, we remove all the switchings and the performance only improves. Also by Lemma 3, after the last removing the policy has to be Ω_2 . \blacksquare

Lemma 5: Generalization of Lemma 2, 3 and 4 to $k > 2$ snapshots case.

Proof: When we have multi-snapshots, the expected one step staleness reduction for snapshot i and $i - 1$ are:

$$\begin{aligned} \tilde{\mu}_i(t) &= N p_{r_i(t)} (1 - c_{i+1}) \left(\mu_i(t) - \delta_{i-1}(0) \mu_{i-1}(t) - \Delta_{i-2} \right) \\ &\geq N p_{r_i(t)} (1 - c_{i+1}) \left(2\mu_{i-1}(t) - \delta_{i-1}(0) \mu_{i-1}(t) - \Delta_{i-2} \right) \\ &\geq N p_{r_i(t)} (1 - c_{i+1}) \left(2 - \delta_{i-1}(0) \right) \left(\mu_{i-1}(t) - \Delta_{i-2} \right) \end{aligned} \quad (9)$$

$$\tilde{\mu}_{i-1}(t) = N p_{r_{i-1}(t)} (1 - c_{i+1}) \left(1 - \delta_i(0) \right) \left(\mu_{i-1}(t) - \Delta_{i-2} \right) \quad (10)$$

Where c_{i+1} is the fraction of cells that have recovered at least one symbol from snapshot $i + 1, i + 2, \dots, k$ and Δ_{i-2} is a common constant need to be subtracted from both due to recovered fractions of earlier snapshots $1, 2, \dots, i - 2$.

Compare (9) and (10) we can see that $\tilde{\mu}_i(t) \geq \tilde{\mu}_{i-1}(t)$ if $p_{r_i(t)} \left(2 - \delta_{i-1}(0) \right) \geq p_{r_{i-1}(t)} \left(1 - \delta_i(0) \right)$. This can be shown in the exactly same way as in Lemma 2. Based on this, we can show the same results for Lemma 3 and Lemma 4 in the same way. \blacksquare

Finally we generalize all these results to the non-constant $\mu_i(t)$ case. This can be done by repeatedly use the non-decreasing property of $\varphi(t)$. First, Lemma 2 is still true because if $t_2 > t_1$ then $\varphi(t - t_2) \leq \varphi(t - t_1)$. Then after any time step, Ω_2 's staleness reduction is bigger than Ω_1 , which means its staleness curve is dragged behind

⁸Note now the switching can be between any two snapshots.

averagely, and for each step it is dragged more. So Ω_2 has a smaller staleness than Ω_1 at any time step. Based on Lemma 2, the no switching and other lemmas can be proved in the exact same way.

Based on the above results, we have proved Theorem 2.

Theorem 3: For the min integrated staleness problem, a greedy scheduling of always exchanging coded bits from the most recent unfinished snapshot is optimal for any monotonic decoding non-mixed coding schemes with infrequent arrival snapshots.

Proof: Since by Theorem 2, greedy is optimal for the min staleness problem under the same conditions, then a greedy algorithm achieves the minimum staleness at any time step, thus also achieves the minimum integration of the staleness with any possible discounting factor a . ■

VI. RANDOM MIXED CODING

A random mixed coding (RMC) randomly mixes up data symbols from all snapshots. RMC is primarily motivated by the scenario of small buffer sensors for which the delay of non-mixed coding can be non-negligible. While RMC does not control the fraction of coding symbols from any snapshot, in other words, it cannot choose to select d_1 symbols from snapshot 1, d_2 symbols from snapshot 2 and d_i symbols from snapshot i , it can only choose the total number of distinct symbols to select and the fraction of each snapshot in the selected symbols will be decided by their distributions. This random drawing of symbols from all snapshots without differentiating them is the simplest algorithm for the low cost sensors.

In this section, we analyze one type of RMC based on growth codes type of random linear codes with incremental decoding, we call it RMC-GC. We model this using the following procedure similar to [7]. That is, when encoding, a sensor node choose a degree d and then randomly draw d symbols from a central pool of original symbols from all available snapshots and *XOR* them together. The central pool model is a good abstraction of the collective effect of the distributed behavior of the sensors drawing symbols randomly from their own buffers, assuming the sinks are uniformly randomly distributed in the field and the original symbols are spatially uniformly mixed up after a short initial period.

Based on this model and strategy, we derive the optimal coding degree at each time step. Before which, we give some further notations.

A. Notations

More generally, in this section we allow different snapshots to have different number of total symbols. The i th snapshot contains N_i symbols. The total symbols of all snapshots at time t is $N(t) = \sum_{i=1}^{S(t)} N_i$. For each snapshot i , denote the fraction of symbols currently recovered by the sinks as $\delta_i(t)$. Denote the probability of randomly drawing a symbol of snapshot i from the pool as $f_i(t)$. Then $f_i(t) = \frac{N_i}{N(t)}$. In case that $N_1 = N_2 = \dots = N_{S(t)}$, we have $f_i(t) = \frac{1}{S(t)}$. For each snapshot i , there is a utility associated with each symbol as $u_i(r_i, t)$.⁹

B. Optimal Degree

Under RMC, the only thing a node can control is the coding degree. The question is whether there exists an optimal degree to achieve the max utility goal and the max data persistence goal. We answer the question in the following theorem.

Theorem 4: The optimal coding degree that achieves both the max utility and max data persistence goals for RMC is

$$d^*(t) = \lfloor \frac{1}{1 - \sum_{i=1}^{S(t)} f_i(t)\delta_i(t)} \rfloor. \quad (11)$$

⁹Note that here we use u to represent the final utility gain after possible time decaying.

When $N_1 = N_2 = \dots = N_{S(t)}$, we have

$$d^*(t) = \lfloor \frac{1}{1 - \bar{\delta}(t)} \rfloor \quad (12)$$

where $\bar{\delta}(t) = \frac{1}{S(t)} \sum_{i=1}^{S(t)} \delta_i(t)$. The corresponding maximum decoding probability is

$$p(t) = d^* \left(\sum f_i(t) \delta_i(t) \right)^{d^*-1} \left(1 - \sum f_i(t) \delta_i(t) \right). \quad (13)$$

Proof: Since the total number of symbols is normally much larger than the coding degree d , the random drawing of symbols can be approximated as a drawing with replacement procedure¹⁰.

For each symbol in a packet, the probability that it is already recovered by the sinks is

$$f(t) = \sum_{i=1}^{S(t)} f_i(t) \delta_i(t), \quad (14)$$

the probability that the symbol is new to the sinks is $1 - f(t)$. Since the symbols are independently randomly drawn, for each received packet encoded in degree d , the probability of decoding it is

$$\begin{aligned} Pr(d) &= Pr(\text{one new symbol, } d - 1 \text{ recovered symbol}) \\ &= \binom{d}{d-1} f(t)^{d-1} (1 - f(t)) \\ &= d \left(\sum f_i(t) \delta_i(t) \right)^{d-1} \left(1 - \sum f_i(t) \delta_i(t) \right) \end{aligned} \quad (15)$$

The expected utility gain for one packet equals the decoding probability multiplies the expected utility gain conditioned on decoding. The expected utility gain conditioned on decoding is the same for any degree encoding (independent of d) as below:

$$\tilde{u} = \mathbf{E}(u | \text{decoding}) = \sum \frac{f_i(t)(1 - \delta_i(t))}{1 - \sum f_i(t) \delta_i(t)} u_i(r_i, t) \quad (16)$$

This also means that no matter what $Pr(d)$ is, the $\vec{\delta} := (\delta_1(t), \delta_2(t), \dots, \delta_{S(t)}(t))$ follows the same sample path expectedly. Thus, to achieve the max utility and max data persistence goal, we simply need to choose $d^*(t)$ to maximize the decoding probability.

Let $c(d) = Pr(d)/Pr(d-1)$, we have $c(d) = \frac{f(t)}{(d-1)/d}$. Since $\frac{d-1}{d}$ is monotonically increasing, $c(d)$ is monotonically decreasing. Thus the maximum $Pr(d)$ is achieved at

$$d^*(t) = \lfloor \frac{1}{1 - f(t)} \rfloor. \quad (17)$$

Apply it to (15) we get the decoding probability of the optimal RMC (13). When $N_1 = N_2 = \dots = N_{S(t)}$, we have $f_i(t) = \frac{1}{S(t)}$ for all $i = 1, 2, \dots, S(t)$, then $f(t) = \frac{1}{S(t)} \sum_{i=1}^{S(t)} \delta_i(t) = \bar{\delta}(t)$, apply it to (17) we have (12). ■

Theorem 4 includes [7]'s optimal degree as a special case, when we have one snapshot as in [7], simply replace all δ_i with a common δ and Theorem 4 tells us the optimal degree of coding is $d^* = \lfloor \frac{1}{1-\delta} \rfloor$, this is consistent with [7]' result but we derive it in a simpler way and with a more general setup.

¹⁰The precise argument of using the selecting without replacement will result in the same conclusion.

VII. BUFFER SIZE AND GROWTH CODES PERFORMANCE

The random mixed coding is a good strategy for very cheap sensors with very constrained memory and computation budget. For sensors with relatively larger buffers and more computation power, they are able to afford the higher cost to control the scheduling between multi-snapshots data and thus also to enjoy the more query-adaptive freedom provided by such control. Namely, these sensors are able to manage a non-mixed way of coding the symbols such that symbols from different snapshots are kept separate and coding only happens within symbols of the same snapshot. This will not induce much overhead when the buffer size is relatively large because with large buffers it is not hard to gather a number of symbols of the same snapshot in a short time and encode them together with a desired degree.

Since GC is a simple and efficient random linear code that recently attracts a lot of interest especially due to its good data persistence performance, in this section we will analyze its performance under different buffer sizes and also discuss the specific solutions of the archiving and staleness problem for GC based non-mixed coding.

A. Large Buffer Case

Before solving the optimal scheduling for GC, we need a better understanding of the individual snapshot coding. In this subsection, we first present an approximate formula of the decoding probability for the coding of a single snapshot. Then with the help of this formula, we derive a lower bound for the minimum decoding probability and thus an upper bound for the maximum number of packets needed for decoding the whole snapshot using growth codes.

[7] assumes there is a central pool to draw the symbols and the sensor node is able to select any number of distinct symbols from the pool with zero overhead. Based on this, they show that the optimal degree is $\lfloor \frac{1}{1-\delta} \rfloor$ where δ is the fraction of recovered symbols. Then by Theorem 4 the probability of decoding with the best coding degree is a function of the recovered fraction δ

$$p_\delta = \lfloor \frac{1}{1-\delta} \rfloor \delta^{\lfloor \frac{1}{1-\delta} \rfloor - 1} (1-\delta) \quad (18)$$

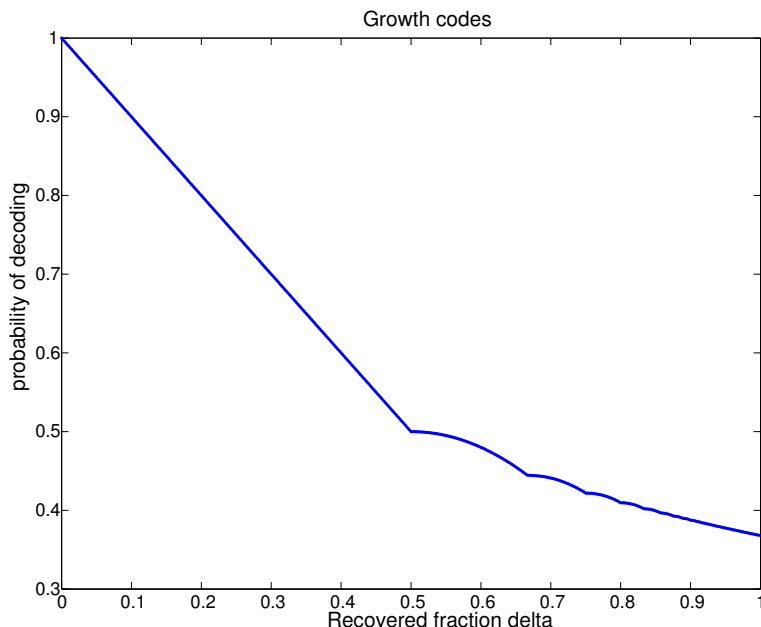


Fig. 2. Decoding probability vs. recovered symbols (approximate: drawing with replacement)

Fig.2 shows the curve of p_δ as a function of δ (the same shape with simple scaling if use the number of recovered symbols as the X axis). When δ increases to close to 1, this formula is not accurate, it should be updated with the selecting without replacement procedure [7].

$$p_r = \frac{\binom{r}{d-1}(N-r)}{\binom{N}{r}} \quad (19)$$

where r is the number of symbols already recovered, N is the total number of symbols, d is the best coding degree $d = \lfloor \frac{N}{N-r} \rfloor$.

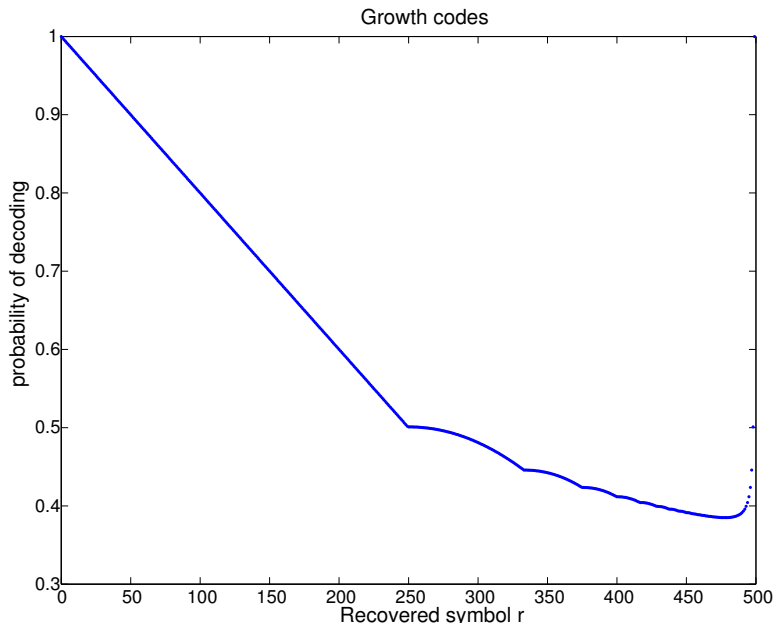


Fig. 3. Decoding probability (drawing without replacement) vs. recovered symbols

Fig.3 shows this curve of the decoding probability as a function of the symbols recovered (the same shape with simple scaling if use the recovered symbol fraction).

Comparing these two curves, we can see that (18) is a very good approximation of (19) for almost all values of δ . For $0 \leq \delta \leq 0.97$, (18) is almost the same as (19), only for the last about 2% of δ values when we almost recover the whole snapshot, (19) starts to increase drastically to 1 while (18) keeps decreasing. We analyze (18) first because it simplifies analysis and is a very good approximation for most part of the data retrieving process, especially for the situations when the bandwidth is constrained s.t. we never recover more than 97% data for any snapshot.

In addition to this, we also observe that there is a minimum decoding probability about 0.4 for both curves. More specifically, we show that the minimum decoding probability for both curves is indeed lower bounded by $\frac{1}{e}$ in Lemma 6. In fact, $\frac{1}{e}$ is the exact minimum value for (18) and a lower bound for (19). For large N , it is approximately the minimum value of (19) as well. So for (19) the decoding probability decreases from 1 to approximately $1/e$ before a point about $\delta = 0.97$, and after the point, the decoding probability increases drastically to 1 within a very short window. Since the decoding probability decides the speed of data recovering, qualitatively speaking, if we assume the same number of packets exchanged in every time slot then the data recovering speed keeps decreasing for the majority part of the data recovered fractions, and in the big scale it decreases slower and slower (roughly convex).

Lemma 6: The expected total number of packets that has to be exchanged for growth codes to decode a complete snapshot is upper bounded by $eN \approx 2.72N$ for large buffer size networks.

Proof: Between (18) and (19), we first show that $p_r \geq p_\delta$ ($\delta = r/N$). This is because a packet can be decoded iff there is one new symbol and $d - 1$ recovered symbols in the randomly selected d coding symbols, a selecting without replacement process of (19) certainly have a higher or at least equal chance than (18)'s selecting with replacement.

Next we show a lower bound for (18) and automatically a lower bound for (19) as well. Observe that p_δ in (18) is a continuous function and derivable almost anywhere except a finite set of points. These points divide $[0, 1]$ into a set of sections. Within each section, $p_\delta = c\delta^c(1 - \delta)$ where $c = \lfloor \frac{1}{1-\delta} \rfloor$ is a constant. Since $c \leq \frac{1}{1-\delta}$, $\frac{dp_\delta}{d\delta} = c\delta^{c-2}[(c-1) - c\delta] \leq 0$. Thus, we know p_δ in (18) is monotonically non-increasing and

$$\min p_\delta = \lim_{\delta \rightarrow 1} p_\delta = \lim_{\delta \rightarrow 1} \delta^{\frac{\delta}{1-\delta}} = \frac{1}{e}.$$

Then, we know that minimum decoding probability for both (18) and (19) satisfies $p \geq 1/e$. Consequently, eN is an upper bound of the expected total packets exchanged for decoding N symbols of a snapshot. ■

We define the *Data Efficiency* of a coding scheme as the expected total number of packets for decoding the whole snapshot. [7] proposes growth codes but provides no quantitative characterization of its data efficiency. We show here that its data efficiency is actually $\Theta(N)$, in other words, $\Theta(\log N)$ order better than the non-coding coupon collector process which on average needs $\Theta(N \log N)$ packets to decode a snapshot. In reality, our simulation shows that a snapshot is recovered with about $1.5N$ packets using growth codes.

This behavior of data gaining dynamics models accurately the case of infinite buffer size when we can merge packets in the buffer to any desired coding degree instantaneously at any time. We call this type of growth codes *Instantaneous Growth Coding (IGC)*. For it to be 100% accurate, we need a buffer size in the order of at least $\Theta(\log N)$. However, generally non-mixed coding performs quite well for networks with relatively large buffer sensors and sparse sinks. This is because the $\Theta(\log N)$ size buffer requirement is to guarantee a INC at all the times. In reality the coding degree only start to jump in the last 3% of δ so a relatively large buffer works fine for most of the time or even all the times (if we never reach that 3% region due to bandwidth constraint).

Lemma 7: The minimum buffer size to support IGC is $\Theta(\log N)$.

Proof: By the coupon collector process we know in order to cover N distinct symbols with independent random drawings, we need at least $N \log N$ random drawn symbols with high probability. Using growth codes, at the time that the $(N - 1)$ th symbol is recovered, the maximum degree of any packets in any node's buffer is $N/2$, from here on the degree needs to increase to N instantaneously. To be able to do this, each node's buffer should already cover all N symbols, thus the buffer size has to be at least $\frac{N \log N}{N/2} = 2 \log N$. ■

B. Small Buffer Case

For sensors with a very small buffer, they can not increase the coding degree instantaneously when needed, especially when that snapshot's data is almost all recovered and the optimal degree needs huge jumps in a short time. In this case, the sensors have to increase the degree as they exchange packets with neighbors and add in new symbols from the exchanging.

In this case, there is a discrepancy between the ideal coding degree and the actual possible coding degree thus the decoding probability will be lower than the one with the ideal degree. Fig.4 shows the simulation result of this scenario, decoding probability vs. number of received symbols, we can see that for sensors of small buffers the decoding probability is mostly a monotonically decreasing function.

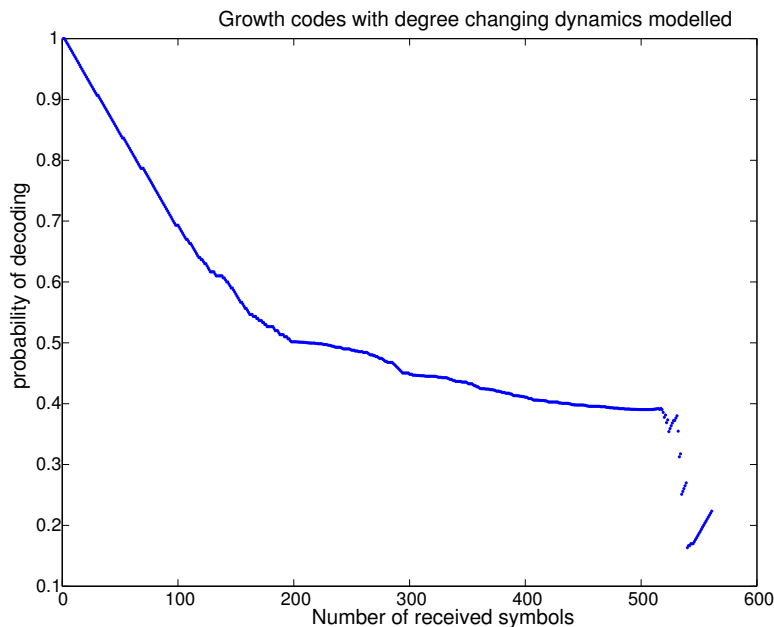


Fig. 4. Simulation result of Decoding probability vs received symbols for small buffer sensors

In addition, Fig.4 has $N = 300$ and a data efficiency less than 600. In fact, our simulation consistently shows that the average number of packets exchanged to decode a whole snapshot with small buffers is less than $2N$. Thus, combined with the large buffer analysis of Lemma 6, we know buffer sizes do not change the order of GC's data efficiency. The simulation result shows that the large buffer case has a data efficiency of about $1.5N$, thus small buffer causes a increase about 1.4. From the analysis and simulation result we conclude that GC's data efficiency performance is not sensitive to buffer sizes.

More than that, the delay for decoding the whole snapshot actually mainly comes from the recovering of the last 3% of the snapshot. For the first 97% of the retrieval, small buffers have very little impact since the optimal degree increases very slowly anyway.

C. Archiving and Staleness Problem with Growth Codes

Given growth code's advantage on data efficiency and low computation cost, it is worth analyzing the archiving problem on a growth code based non-mixed coding. To do so, we just need to apply Theorem 1's general result plugging in the decoding probability for growth codes we derived in this section. Further, if we assume there is a utility value associated with each original symbol independent of the symbols already recovered or $u_i(r) = u_i$, then we replace every $u_i(\cdot)$ in Theorem 1 with u_i . For large buffer sensors, replace the p_r in Theorem 1's with (19). Since (19) is not monotonically non-increasing, the optimal scheduling is not a greedy approach. If $u_i = u_j$ for all i, j , we can say that the optimal policy will favor either the least recovered snapshot or the most recovered snapshot within all the unfinished snapshots. For small buffers or the case of very limited bandwidths, p_r is approximately monotonically decreasing in the working range, then by Lemma 1, a greedy scheduling algorithm that always favor the snapshot with the maximum $p_{r_i} u_i$ is optimal. When $u_i = u_j$ for all i, j , the greedy algorithm will always choose the maximum p_{r_i} at each step, or equivalently the minimum r_i , the least recovered snapshot to exchange packets.

For the staleness problem, as we discussed earlier, GC mostly satisfies the monotonic property thus when the inter-arrival is infrequent, greedy is optimal for minimizing the data staleness under GC based non-mixed coding.

[7] already shows that for random linear codes with incremental decoding, GC is optimal for max utility and data

persistence of the single snapshot case, in other words, GC provides a uniform largest decoding probability at all times within such codings. Since scheduling is orthogonal to coding for non-mixed coding, we can see that GC is also the optimal one within such codings to provide the best performance. Because for the archiving problem, GC gives higher index value for all snapshots, thus the final scheduling policy will result in a higher value as well. For the staleness problem, when greedy is optimal, GC also gives larger staleness reductions due to higher decoding probability than other codings of the same category.

VIII. IMPLEMENTATION ISSUES

To implement the optimal scheduling algorithm in a real distributed random network, a key issue is how the sensors can obtain the optimal schedule. The sinks need to inform the sensors either the utility information or the final scheduling decisions. While one may argue that if there is a channel between the sinks and the sensors, then the sensor might learn the sinks' locations through it and use routing instead of random coding after it. In fact, we do aim at those applications that the sinks either use separate channels or the same channels among the sensor nodes to deliver query/scheduling information to the sensor nodes. However, since we assume the channels are not stable because both the sensor nodes and the sinks are prone to fail in a fairly dynamic environment, it will be too costly to maintain some path information for location based routing. In addition, since there are multiple sinks and we assume the sinks are uniformly distributed among the sensor nodes and fail independently, on average, random gossiping type of routing is not too bad compared with the shortest path routing.

We expect that the sinks do not need to transmit a lot of information to the sensor nodes. According to the optimal scheduling they compute, they just need to transmit the index of the snapshot that is scheduled to all the sensor nodes whenever the schedule changes. This broadcast task can be achieved through ad hoc transmissions from the sinks. The control level is adjustable between coarse scheduling (big chunks of data) and fine (packet level), even we schedule in the packet level, the snapshot index does not change for every packet transmitted (the coding degree can be decided based on an estimation of the retrieved fraction without frequent communication with the sinks), thus in general the control traffic is relatively small compared to the large volume of data transmissions.

IX. CONCLUSION AND FUTURE WORKS

We introduced a utility-based framework with joint scheduling and coding decisions for collecting multi-snapshots spatial data in a resource constrained sensor network. We investigated the problem of how to schedule (or prioritize) the codewords from multiple snapshots under the archiving model and the staleness model. For the archiving model, we mapped the scheduling problem into a Multi-Armed Bandit (MAB) problem, and derived the optimal solution using Gittins Indices. We also derived an efficient algorithm to calculate the indices based on the underlying Markov process. The scheduling problem under the staleness model was formulated as a Restless Bandit(RB) problem. A greedy scheduling policy was presented and proved to be optimal when the inter-arrival time between snapshots is not too short.

We then studied the optimal coding scheme when *random mixed coding* (for data in different snapshots) is allowed. We generalized the growth codes in [7] to arbitrary linear-codes-based random mixed coding and showed that there exists an optimal degree of coding. Various practical issues and in particular the buffer size impact on performance were also discussed, where we showed through both analysis and simulation that the expected number of packets to retrieve all data is less than twice the total number of data items.

One possible future work is to look at the implementation issues in practice and simulate the optimal scheduling's performance based on the model proposed in Sec. VIII. Particularly, check out the control overhead influence on the total data utility retrieved.

X. ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under grants EEC-0313747. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the National Science Foundation. This research is also continuing through participation in the International Technology Alliance sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defense, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

REFERENCES

- [1] V. P. A. Dimakis and K. Ramchandran. Decentralized erasure codes for distributed networked storage. *IEEE Transactions on Information Theory*, 2006.
- [2] R. Ahlswede, N. Cai, S. Li, and R. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, July 2000.
- [3] C. Berrou, A. Glavieux, and P. Thitimajshima. Near shannon limit error-correcting coding and decoding: Turbo-codes. In *Proceedings of IEEE International Communications Conference*, 1993.
- [4] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *SIGCOMM*, pages 56–67, 1998.
- [5] K. Fall. A delay tolerant network architecture for challenged internets, 2003.
- [6] J. Gittins and D. Jones. A dynamic allocation index for the sequential design of experiments. *Progress in Statistics*, 1, 1972.
- [7] A. Kamra, J. Feldman, V. Misra, and D. Rubenstein. Growth codes: Maximizing sensor network data persistence. In *Proceedings of ACM Sigcomm*, 2006.
- [8] S. Lin and D. Costello. *Error Control Coding: Fundamentals and Applications*. Prentice Hall, 1982.
- [9] M. Luby. Lt codes. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 271–282, 2002.
- [10] D. J. MacKay and R. M. Neal. Near shannon limit performance of low density parity check codes. In *Electronics Letters*, 1996.
- [11] D. Marco, E. Duarte-Melo, M. Liu, and D. L. Neuhoff. On the many-to-one transport capacity of a dense wireless sensor network and the compressibility of its data. In *IPSN*, 2003.
- [12] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge University Press, 1995.
- [13] J. Nino-Mora. Dynamic allocation indices for restless projects and queueing admission control: A polyhedral approach. *Math. Program*, 2002.
- [14] C. Papadimitriou and J. Tsitsiklis. The complexity of optimal queuing network control. *Math. of Operations Research*, 1999.
- [15] P. Varaiya, J. Walrand, and C. Buyukkoc. Extensions of the multiarmed bandit problem: The discounted case. *IEEE Transactions on Automatic Control*, 1985.
- [16] C. Wan, S. B. Eisenman, A. T. Campbell, and J. Crowcroft. Siphon: overload traffic management using multi-radio virtual sinks in sensor networks. In *SenSys*, 2005.
- [17] J. Warland. *An introduction to queueing networks*. Prentice Hall, 1988.
- [18] P. Whittle. Arm-acquiring bandits. *The Annals of Probability*, 9, 1981.