Aggregation in Dynamic Networks

Alejandro Cornejo
MIT CSAIL
acornejo@csail.mit.edu

Seth Gilbert
National University of Singapore
seth.gilbert@comp.nus.edu.sg

Calvin Newport
Georgetown University
cnewport@cs.georgetown.edu

Abstract

The aggregation problem assumes that every process starts an execution with a unique token (an abstraction for data). The goal is to collect these tokens at a minimum number of processes by the end of the execution. This problem is particularly relevant to mobile networks where peer-to-peer communication is cheap (e.g., using 802.11 or Bluetooth), but uploading data to a central server can be costly (e.g., using 3G/4G). With this in mind, we study this problem in a dynamic network model, in which the communication graph can change arbitrarily from round to round.

We start by exploring *global* bounds. First we prove a negative result that shows that in general dynamic graphs no algorithm can achieve any measure of competitiveness against the optimal offline algorithm. Guided by this impossibility result, we focus our attention to dynamic graphs where every node interacts, at some point in the execution, with at least a p-fraction of the total number of nodes in the graph. We call these graphs p-clusters. We describe a distributed algorithm that in p-clusters aggregates the tokens to $O(\log n)$ processes with high probability.

We then turn our attention to *local* bounds. Specifically we ask whether its possible to aggregate to $O(\log n)$ processes in parts of the graph that locally form a p-cluster. Here we prove a negative result: this is only possible if the local p-clusters are sufficiently isolated from the rest of the graph. We then match this result with an algorithm that achieves the desired aggregation given (close to) the minimal required p-cluster isolation. Together, these results imply a "paradox of connectivity": in some graphs, increasing connectivity can lead to inherently worse aggregation performance.

We conclude by considering what it seems to be a promising performance metric to circumvent our lower bounds for local aggregation algorithms. However, perhaps surprisingly, we show that no aggregation algorithm can perform well with respect to this metric, even in very well connected and very well isolated clusters.

1 Introduction

In the aggregation problem every process starts the execution with a piece of information which is abstractly represented as a unique *token*. At the end of the execution every token must be uploaded to a central server. During the execution processes use peer-to-peer communication to consolidate the tokens at a minimum number of processes. The performance of an aggregation algorithm is described by the number of processes with tokens to upload at the end of the execution (the fewer the tokens the better the performance). This problem is motivated by the following observation: in many networks, local links between processes are far cheaper (in terms of expense, required infrastructure, and energy) than the long distance links needed to communicate with a central server. By using local links to aggregate tokens at a small number of processes, we reduce the total number of expensive long distance links required at the end of the execution to upload all the tokens.[1]

One setting in which this problem is particularly relevant is mobile networks. To give a concrete example, consider a traffic reporting system, such as CarTel [HBZ+06], which uses drivers' smartphones to upload traffic observations to a central traffic server. With major cellphone carriers such as AT&T eliminating their unlimited data plans, uploading these observations might deplete users' limited minutes. In addition, cellular links drain device batteries (a 3G link, for example, can use orders of magnitude more battery power than a local 802.11 or Bluetooth link [TRB+11]). A good aggregation algorithm in this setting could leverage free, low-energy local links (i.e., 802.11 or Bluetooth) to reduce the usage of expensive, high-energy long distance links (i.e., 3G).

Problem Setup. Aggregation is a problem of deep interest to practitioners. In this paper, however, we show that it also produces a set of interesting problems for theoreticians to tackle. We study both upper and lower bounds for randomized solutions to the aggregation problem in a synchronous dynamic network model. We model the communication topology as a *dynamic graph* where the edge set can change arbitrarily from round to round. We adopt this model for two reasons. First, it matches the unpredictable topology observed in real mobile networks, a setting where the aggregation problem is particularly relevant. Second, dynamic graphs generalize a diversity of different topology assumptions and results in this model, therefore, are widely applicable (c.f., the discussion in [KO11]).

Global Results. We start by showing a negative result which proves that no aggregation algorithm is competitive with the optimal offline algorithm in every dynamic graph (with respect to the number of processes required to upload all the tokens). Specifically, we show there are dynamic graphs where the optimal offline algorithm can aggregate to a single process, but with high probability any randomized algorithm will aggregate to $\Omega(n)$ processes, where n is the network size. Chastened by this impossibility result, we redirect our ambition to identifying natural structures that enable aggregation algorithms to perform well and which arise in real-world dynamic graphs. This search leads us to the notion of a *p-cluster* (where $p \in (0, 1]$). We say a dynamic graph is a p-cluster if by the end of the execution every node has interacted with at least p-fraction of all nodes. We highlight that we do not restrict when these connections occur or in what order. In particular we do not assume these connections are random, which would imply a graph with expander-like properties and would simplify significantly the task of aggregation. In contrast, we model these connections as being picked by an adaptive online adversary. We describe a randomized algorithm ClusterAggregate$_p$

[1] Implicit in this cost analysis is the assumption that, in practice, it consumes less bandwidth/energy to upload a collection of x tokens from a single source than to have x distinct sources each upload one token. There are many different rationales for this assumption. For example, each long distance connection induces a fixed overhead independent of the size of the packet data payload. This comes from both the handshaking required to connect with the central server, and the fixed size of the packet header. In addition, the data represented by abstract tokens in our model is often, in reality, either compressible or summarizable (e.g., the server only needs an average, max, or min over the values). This also leads to great bandwidth/energy reductions when we can collect the values at a small number of sources before uploading.

that with high probability aggregates all tokens to $\mathcal{O}(\log n)$ processes when executing on a p-cluster.

Local Results. In the second part of this paper, we turn our attention to proving performance guarantees that depend not on the "global" structure of the dynamic graph but on its "local" structure. For example, consider a dynamic graph that is not itself a p-cluster, but that includes many subsets of nodes that locally form p-clusters. We would like our aggregation algorithms to *discover these hidden structures* and locally aggregate well.

Consider, for example, a hypothetical aggregation algorithm which ensures that, for every subset of nodes that form a 1-cluster (i.e., clique) in the dynamic graph, there is at most one process that uploads tokens at the end of the execution (i.e., at most one "uploader"). Such an algorithm (which is, in fact, easy to obtain) would guarantee the following property: given a dynamic graph G, where η is the size of the minimum clique cover of G: for every execution of the algorithm, the number of uploaders is at most η. Such results having the interesting property that they leverage the existence of hidden structure, even while finding such structure may be NP-complete (as is finding the minimum clique cover).

Here, we focus on leveraging the existence of local p-clusters in the graph. We first prove that the ability to aggregate well in a local p-cluster depends on the isolation of this cluster with respect to the rest of the graph. In more detail, a subset of nodes is a (p, r)-cluster if: (a) the nodes form a p-cluster amongst themselves; and (b) a node in the cluster has at most r neighbors outside the cluster. We prove that for any aggregation algorithm that guarantees at most k uploaders in each (p, r)-cluster, r must be $O(k^2)$: in other words, more aggregation requires more isolation. Notice that this result has interesting echoes of classical complexity theory, where good polynomial-time approximations of the largest clique problem require sufficient isolation of the cliques in the graph.

On the positive side, we then extend our analysis of ClusterAggregate$_p$ showing that it guarantees, with high probability, to aggregate all the data in every $(p, O(\log n))$-cluster to $\mathcal{O}(\log n)$ nodes in that cluster. By the previous lower bound result, this performance is within a log factor of the optimal, both in terms of the amount of isolation required and on the total number of broadcasters.

Together, these results imply a "paradox of connectivity": in some graphs, increasing connectivity can lead to inherently worse aggregation performance. Intuitively we would expect the performance of aggregation protocols to improve as the networks become more connected, allowing more sharing of information. What we show, though, is that when clusters are relatively isolated, we can achieve good performance, limiting the number of uploaders in each cluster. By contrast, as connectivity increases, the performance worsens and it is impossible to achieve the same bounds on the number of uploaders per cluster.

To conclude we turn our attention to an alternative local performance metric. The metric used up to this point counts the number of processes in a cluster that upload. The alternative metric, by contrast, counts the number of processes that upload a token *on behalf* of a process in a cluster. Though the metric seems like a promising alternative to circumvent our previous lower bounds, we prove, perhaps surprisingly, that it is impossible to satisfy. In more detail, we show no aggregation algorithm can guarantee to perform well with respect to this metric, even if we require it to only work in very isolated and very well connected clusters (i.e., $(1, 1)$-clusters).

Related Work. Dynamic networks have received a lot of attention in recent years. Though many different dynamic network models have been proposed (see [KO11] for a good overview), the variant most relevant to our work was introduced in [KLO10], where the authors studied gossip in the context of a communication topology that could change arbitrarily from round to round. In subsequent work, prioritized gossip [CN10], consensus [KMO11], and random linear network coding [Hae10] were also studied in the same model. In [KLO10, KMO11, Hae10] the network was assumed to be connected at every round: such a constraint is similar to our global p-clusters property. Our study of local

p-clusters, by contrast, has some echoes of the study of prioritized gossip in [CN10], which also sidestepped global constraints and proved guarantees relative to local connectivity properties.

Our model differs from the existing models in [KLO10, CN10, KMO11, Hae10], however, in one crucial respect. These previous studies assume *broadcast communication*: in each round, each process can broadcast a message which is received by its arbitrary neighbor set. In this paper, by contrast, we assume *unicast communication*: in each round, each process is arbitrarily paired with at most one of other process with which it can interact. Such unicast communication has been previously studied, among other dynamic settings, in the context of gossip–e.g., [CHS10]—and population protocols (see, [AR07]). In the former, the choice of pair is typically random, not arbitrary, while in the latter, the protocols are assumed to have limited memory.

Finally, there exists a sizable corpus of work on aggregation in networks, much of it centered on the collection, combination, and uploading of data (or functions on the data, such as average value) in resource-constrained sensor networks; c.f., [MFHH02, IEGH02, KEW02]. The topic has proven applicable in other settings as well, such as gathering information from components in a telecommunications system [PPC99].

2 Model

We consider a synchronous network with unpredictable connectivity topology. Namely, we model communication using a *dynamic graph* $D = (V, \mathcal{E})$, where V is a static set of nodes, and $\mathcal{E} : \{1, ..., t\} \to 2^{V \times V}$ is a function mapping each round number $r \in \{1, ..., t\}$ to a set of undirected edges $\mathcal{E}(r)$ that captures the connectivity in that round. The parameter $t \geq 1$ is the *duration* of the dynamic graph. Different dynamic graphs can have different durations.[2] We constrain these graphs to behave as *interaction graphs*: in each round r, each process u is included in at most one edge in $\mathcal{E}(r)$, and there are no self-edges. Thus, for each round r, the static graph $(V, \mathcal{E}(r))$ is a matching. For each round $r \in \{1, ..., t\}$ in the execution, the edge set $\mathcal{E}(r)$ defines which pairs of processes *interact* by sending each other their state and performing some local computations.

To simplify definitions and notation, we assume V is fixed and known for all dynamic graphs. This allows us to define an *algorithm* $\mathcal{A}$ as a collection of $|V|$ randomized *processes*, one for each $u \in V$. It follows that processes have unique ids (as they can agree on a mapping from V to an id space) and know $n = |V|$.

Aggregation Problem. At the beginning of every execution α, each process u is passed a unique *token* $\alpha.\sigma[u]$. Similarly, at the end of any execution α each process u produces an output (potentially empty) $\alpha.\gamma[u]$. We call $\alpha.\sigma$ the *input assignment* of α, and $\alpha.\gamma$ the *output assignment* of α.

An *aggregation algorithm*, must ensure that every token in the input assignment is subsequently output exactly once in the output assignment. In other words, tokens are neither lost nor duplicated. Formally:

Definition 1 (Aggregation Algorithm)**.** An algorithm $\mathcal{A}$ is an aggregation algorithm if and only if at the end of every execution α of $\mathcal{A}$ we have: • *No Loss:* $\bigcup_{u \in V} \alpha.\gamma[u] = \bigcup_{u \in V} \alpha.\sigma[u]$. • *No Duplication:* $\forall u, v \in V, u \neq v : \alpha.\gamma[u] \cap \alpha.\gamma[v] = \emptyset$.

Uploaders. A natural metric for the performance of an aggregation algorithm is the total number of processes that end up broadcasting at least one token at the end of the execution. In the rest of

[2] In this paper we study algorithms that run for a fixed duration and then terminate. This follows because in practice aggregation is something that occurs over a fixed interval, as in "take a sensor reading, aggregate for t seconds, upload the data, then start over with a new reading." By giving durations to dynamic graphs, when we later say that an algorithm must work with all dynamic graphs, it follows automatically that must work for all durations.

the paper we call any such process an *uploader*. The smaller the number of uploaders, the better the performance of an aggregation algorithm. Formally, given an execution α, the set of uploaders is the set of nodes which upload at the end of the execution $\{u \mid u \in V \wedge \alpha.\gamma[u] \neq \emptyset\}$.

Token Ownership. We can leverage the conditions of Definition 1 to prove that every aggregation algorithm must satisfy a property we call token ownership. In the paragraph below we define this property informally. This should be sufficient for understanding the lower bound proofs that follow. The formal definitions and proofs appear in the appendix.

At the end of every execution prefix of an aggregation algorithm, each process u *owns* a set of tokens T_u such that the following two properties are true: (1) if we extend this execution prefix such that process u interacts with no other processes for the remainder of the execution, then with probability 1 it will output T_u at the end of this extension; and (2) for every token $x \in T_u$, either u received x as its input, or there is some other process v that received x as its input and there is a sequence of interactions in the prefix that starts with v and ends with u.

In other words, at the end of every round, every process has to commit to a set of tokens that it will definitely output if it finishes the execution in isolation. Furthermore, these tokens have to be tokens it actually heard about. To reiterate: these are not extra conditions that we impose on aggregation algorithms. These are instead conditions that we can show are true for any aggregation algorithm.

3 Global Bounds

3.1 The Impossibility of a Competitive Aggregation Algorithm

The amount of achievable aggregation in a given execution is affected by the dynamic graph for which the execution is defined. For example, if the dynamic graph isolates all nodes we cannot blame the algorithm for requiring n uploaders at the end of the execution. With this in mind, it makes sense to judge an algorithm's performance relative to the best achievable performance in the graph.

We could hope to find an algorithm that guarantees (with high probability) at most $k \cdot f(n)$ uploaders in any dynamic graph where the offline optimal algorithm (which knows G in advance) guarantees at most k uploaders. Ideally we would like $f(n)$ to a reasonably small function (i.e. $f(n) \in \text{polylog}(n)$). Our first result, however, proves that no algorithm can provide these guarantees.

Theorem 1. *For every aggregation algorithm $\mathcal{A}$ there is a dynamic graph G where the offline optimal algorithm guarantees 1 uploader but w.h.p., the system $(\mathcal{A}, G)$ produces an execution with $\Omega(n)$ uploaders.*

Proof. We define dynamic graph $G = (V, \mathcal{E})$ with a duration of two rounds, but it is straightforward to extend it to an arbitrary duration. Parition V into pairs of nodes (omitting one node if $|V|$ is odd). Define $\mathcal{E}(1)$ to consist of one edge $\{u, v\}$ for each pair $\{u, v\}$ from our partition. Next, for each pair, randomly select one node to be *isolated* and another to be *social*. In the next round we do not include any isolated node in an edge. Also, we choose a single *super*-social node to visit every other social node (and the node omitted in round 1 for the case where $|V|$ is odd). The fact that the offline optimal algorithm would guarantee 1 uploader follows from the possibility of aggregating all information at the super-social node.

Here we deploy the concept of *token ownership* which tells us that at the end of every round, every process must *own* a set of tokens (potentially empty) that it will output if it continues the execution in isolation. Furthermore, this set can only contain tokens it could have heard about at this point. It follows that after the first round of any execution in G, for each pair $\{u, v\}$, at

least one of these two processes owns at least one token. Call this the *committed process* (breaking ties arbitrarily in the case where both would output a token). When constructing G we choose the isolated node at random. For each pair, therefore, the probability that we isolate a committed process is $\frac{1}{2}$. Notice, any isolated committed process outputs at least one token. We thus expect $\Theta(n)$ committed processes that correspond to isolated nodes, and Chernoff tells us that with high probability we are not more than a constant factor away from this expectation. It follows that, w.h.p., $\mathcal{A}$ combined with G generates an execution where the number of uploaders is $\Omega(n)$. □

3.2 Aggregating in Well-Connected Dynamic Graphs

The previous lower bound tells us that it is impossible for an aggregation algorithm to be competitive with an optimal offline algorithm in all dynamic graphs. However this result relied on dynamic graphs in which a large fraction of the nodes had a minimum of connection with the rest of the graph. We should not be surprised that it is hard to aggregate well in the presence of such minimal and fleeting connections. Here we present an aggregation algorithm that is competitive as long as the graph is well-connected, which we formalize with the natural notion of a *p-cluster:*

Definition 2 (p-Cluster)**.** A dynamic graph G is a p-cluster for some $p \in (0, 1]$ iff every $u \in V$ is included in an edge in G with at least $\lfloor p|V| \rfloor$ different nodes.

We emphasize that we do not assume the connections of a node in a p-cluster to be random (which would simplify the task of aggregation). In contrast, we consider thew worst possible set of connections, modeling them as picked by an adaptive online adversary.

Below we present an algorithm tailored to aggregate well in p-clusters.

Algorithm. A process can be in one of two states, either *active* or *inactive*. At the end of the execution (i.e. all time t) all active processes will upload their tokens. Initially every process is active and therefore a potential uploader. When starting the algorithm each process u initializes a TAGCOUNT_u variable to zero, and selects an identifier ID_u of $\Theta(\log n)$ bits at random. Consider the interaction between processes u and v. If $\text{ID}_u \leq \text{ID}_v$ then process u does nothing. On the other hand, if $\text{ID}_u > \text{ID}_v$ then process u increases its TAGCOUNT_u variable. If in addition process v is active and $\text{TAGCOUNT}_u \geq c_p \log n$ then process u hands off all its tokens to process v *and* switches to the inactive state. Here $c_p \in \Theta(1/p)$ is a sufficiently large constant which depends on p which we will determine later.

Algorithm 1 $\text{ClusterAggregation}_p$ at process u

```
1: ACTIVE_u ← true
2: ID_u ← random string of Θ(log n) bits
3: TAGCOUNT_u ← 0
4: for each interaction with process v do
5:     if ID_u > ID_v then
6:         if ACTIVE_v and TAGCOUNT_u ≥ c_p log n then
7:             hand off tokens to process v
8:             ACTIVE_u ← false
9:         TAGCOUNT_u ← TAGCOUNT_u + 1
10: if ACTIVE_u then upload tokens
```

We reiterate that all active process broadcast data at the end of the execution, and a process only hands off its tokens and becomes inactive when encountering an active process. Therefore the ClusterAggregation algorithm satisfies the no loss and integrity conditions by construction, it

remains only to show that when the algorithm is executed in p-cluster with high probability the number of active nodes is at most $\mathcal{O}(\log n)$.

Moreover, also by construction it follows that the $c_p \log n$ processes with the smallest random identifier in the graph cannot get tagged $c_p \log n$ times, and therefore they cannot become inactive.

Fact 1. The processes with the smallest $c_p \log n$ identifiers remain active throughout the execution.

The following theorem captures the main result of this section.

Theorem 2. *Fix a constant $p \in (0, 1]$. For every dynamic graph G which forms a p-cluster the system* $(\textsc{ClusterAggregation}_p, G)$ *produces w.h.p., an execution with $\mathcal{O}(\log n)$ uploaders.*

Proof Outline of Theorem 2.

Since the node identifiers are chosen uniformly at random from a space of $\Theta(\log n)$ bits, the probability that they are not unique can be made less than n^{-c} for any constant c. Hence, we can use a union bound to show that w.h.p., the random identifiers are unique. Thus for simplicity and without loss of generality, in the rest of the proof we assume that the random identifiers generated are unique.

Let S be the set of $\mathcal{O}(\log n)$ processes with the smallest random identifiers. To prove theorem 2 it suffices to show that with high probability at the end of the execution only the processes in S will remain active. Since by assumption the graph is a p-cluster, it follows that any process will have met with at least $p|V|$ different processes. Specifically, we divide the interactions of every process in $V \setminus S$ into two groups of size $p|V|/2$. We will then show that with high probability a process is tagged $c_p \log n$ times in the first group of the interactions by a subset of the processes in S, and is subsequently deactivated in the second group of the interactions by a process in S (with which it had not interacted).

To prove theorem 2 we need two technical lemmas. The first lemma shows that, given a set $Q \subseteq V$ of processes selected according to some property of their random identifier (i.e., smallest, largest, etc.), during the first half of the interactions a fixed process $u \in V \setminus Q$ interacts with no more (and no less) than a constant fraction of the processes in Q. The proof (which appears in appendix B) follows from a generalization of the Chernoff bound to Hypergeometric random variables.

Lemma 3. *Let X_u be a random variable that counts the number of processes in Q a process $u \in V \setminus Q$ meets in the first $p|V|/2$ interactions.* $\Pr\left[X_u \leq \frac{p}{4}|Q|\right] \leq e^{-\frac{p}{16}|Q|}$ *and* $\Pr\left[X_u \geq \frac{3p}{4}|Q|\right] \leq e^{-\frac{p}{24}|Q|}$

Therefore, assuming $|Q|$ is "large", with high probability a process $u \in V \setminus Q$ will see no more and no less than a constant fraction of the process in Q during the first half of the interactions.

Our second technical lemma shows that w.h.p., a process $u \in V \setminus Q$ will encounter in the second half of the interactions a process with which it hadn't interacted before. To prove this (the proof appears in appendix B) we leverage lemma 3, and observe that at every interaction a process u has a probability of meeting a process in Q which is $\mathcal{O}(|Q|/|V|)$.

Lemma 4. *Assume $|Q| \geq 48/p$. The probability that a process $u \in V \setminus Q$ does not interact with a new process of Q in the last $p|V|/2$ interactions is less than $e^{-\frac{p}{48}|Q|}$.*

With these technical lemmas in place we are ready to prove theorem 2.

Proof of theorem 2. Let S be the set of $5c_p \log n$ processes with smallest identifier, and let A be the set of $c_p \log n$ processes with smallest identifier, clearly $A \subset S$. Define $B = S \setminus A$ as the remaining $4c_p \log n = |B|$ processes in S. By Fact 1 the processes in A remain active throughout the execution.

To prove the theorem it suffices to show that w.h.p., all processes in $V \setminus S$ become inactive. In particular we will show that the probability that a fixed process in $V \setminus S$ remains active is at most $1/n^2$, and then union bound over all processes.

Fix a process $u \in V \setminus S$. If process u meets $|B|/4$ processes of B in the first $p|V|/2$ interactions it will get tagged at least $c_p \log n$ times, and if in the remaining interactions it meets a process of A it will become inactive.

Instantiating lemma 3 with $Q = B$, the probability that process u *does not* interact with at least $|B|/4$ processes within the first $p|V|/2$ interactions is at most $e^{-p|B|/16}$. Similarly, instantiating lemma 4 with $Q = A$, the probability that in the remaining interactions process u *does not* meet a process of A is at most $e^{-p|A|/48}$. Therefore, the probability that process u remains active is at most $e^{-p|B|/16} + e^{-p|A|/48}$. Finally, by letting $c_p \geq 3 \cdot 48/p$ we have that the probability that node u remains active is at most $1/n^2$ which concludes the theorem.

□

We remark that by tweaking the constants, the same proof can be used to show the same statement holds with probability at least $1 - 1/n^c$ for any constant c. This fact will be useful later.

4 Local Bounds

In practice, a lot of the networks where aggregation is interesting are large. For example, consider a network of thousands of taxicabs spanning a large metropolitan city. In this setting, it is unlikely that the global network of taxicabs, spanning the whole city, form a p-cluster (some cabs will inevitably be stuck in the boondocks). At the same time, however, there are likely plenty of clusters of cabs *within the network* that locally form p-clusters (e.g., at areas of high density like downtown or at the airport). The question we pursue in this section is whether we can find algorithms that successfully find and leverage these local structures that are hidden within the larger global topology.

In doing so, we discover the following interesting *paradox of connectivity*. Intuitively it might seem that aggregation protocols perform only better on better connected networks: at worst, a protocol could ignore the extraneous communication links. In fact, however, we observe the following: when the local clusters are relatively isolated, we can develop aggregation protocols that work well, minimizing the number of uploaders in each cluster; by contrast, when the clusters have more communication with the rest of the network, it becomes impossible to achieve efficient aggregation.

We must begin, however, by generalizing our definitions of a cluster to describe a well-connected sub-graph within a larger graph.

Definition 3 ((p, r)-Cluster)**.** Fix some $p \in (0, 1]$ and $r \in \{0, ..., n\}$. We say a subset of $S \subseteq V$ forms a (p, r)-cluster in dynamic graph G, iff: 1) every $u \in S$ has a link in G to at least $\lfloor p|S| \rfloor$ different nodes in S; and 2) no node $u \in S$ has a link to more than r nodes in $V \setminus S$.

Observe that if $r = 0$ then we require clusters to be completely isolated from the rest of the graph, whereas, letting $r = n$ imposes no isolation requirements.

4.1 The Necessity of Cluster Isolation

Intuition. In this subsection we show that if an aggregation algorithm guarantees no more than k nodes in every (p, r)-cluster then r cannot be much bigger than k^2. In other words, *more aggregation requires more isolation.*

At a high-level, our lower bound works by construction a pair of dynamic graphs and arguing that any aggregation algorithm will fail (with constant probability) to guarantee k uploaders in every (p, r)-cluster in one of the two dynamic graphs if r is larger than $O(k^2)$.

To simplify the theorem statement and the proof we introduce the following definition.

Definition 4. Fix $p \in (0,1]$ and $r,k \in \{1,...,n\}$. Then $\mathcal{A}$ is a (p,r,k)**-cluster-aggregation algorithm** if in every dynamic graph G w.h.p., $(G,\mathcal{A})$ generates an execution where every (p,r)-cluster in G has at most k uploaders.

Theorem 5. *Fix some $p \in (1/2, 3/4)$ and $k \leq \sqrt{n/40}$. For any $r > 30k^2$ there does not exist a (p,r,k)-cluster-aggregation algorithm.*

Proof. Assume for contradiction that $\mathcal{A}$ is a (p,r,k)-cluster-aggregation algorithm for $r > 40k^2$.

We will construct a pair of dynamic graphs and we will argue that when $\mathcal{A}$ is executed on one of these two graphs, with constant probability there will exist a (p,r)-cluster with more than k uploaders, reaching a contradiction.

We begin by defining some constants $q = 5k$, $m = 2k$, and $\ell = m\left(\frac{p}{1-p}\right)$. A few observations regarding these constants: (1) $m < \ell < 3m$; (2) $\ell/(m+\ell) \geq p$; and (3) $q\ell < r$.

We partition the first $q\ell + qm$ nodes into sets $X_1, \ldots, X_q$ and $Y_1, \ldots, Y_q$, where for $i \in \{1, \ldots, q\}$ we have $|X_i| = \ell$ and $|Y_i| = m$, and $X = \bigcup_{i=1}^{q} X_i$. Notice that since $k \leq \sqrt{n/40}$ then $q(\ell + m) \leq (5k)(8k) \leq n$, and hence this partitioning is possible. In the following we only deal with the $q\ell + qm$ nodes partitioned and assume the other nodes are isolated in both graphs we construct.

In both graphs, the first set of rounds consist of nodes in X interacting with each other, while the remaining nodes are isolated. That is, the nodes X form a clique in the dynamic graph. It is at this point, after all nodes in X have interacted with each other, that we differentiate the graphs G_1 and G_2.

Constructing G_1. For each $i \in \{1, \ldots, q\}$ we schedule each of the m nodes of Y_i to interact with each of the ℓ nodes of X_i. We schedules these interactions in m *passes*, each consisting of ℓ interactions. In each pass, for each i, one node from Y_i interacts with every node in X_i in some arbitrary order. During the pass, there are no other interactions.

Thus, at the end of the duration G_1, the nodes in X have interacted in a clique, and for each i, the nodes (X_i, Y_i) interact in a complete bipartite graph. Notice that the nodes in Y_i have no direct interactions with each other.

Observe that the nodes in X_i and Y_i together form a (p,r)-cluster: (i) since $\ell/(m+\ell) \geq p$, every process in X_i and Y_i has interacted with at least a p fraction of the other nodes in X_i and Y_i; and (ii) each node in Y_i only interacted with nodes in X_i while each node has interacted with at most $q\ell$ nodes outside Y_i. Since $q\ell < (4k+4)(6k) < 48k^2 < r$ (by assumption on r), it follows that the nodes in $X_i \cup Y_i$ form a (p,r)-cluster.

In addition, observe that the nodes in X form a (p,r)-cluster: (i) every node in X interacts with every other node in X, and (ii) each node in X interacts with at most $m = 2k < r$ nodes not in X.

Constructing G_2. Graph G_2 is almost identical to G_1 with the following exception: for each $i \in \{1, \ldots, q\}$ we pick a random number n_i uniformly from the set $\{1, \ldots, m\}$. After the n_i^{th} pass in the interaction between X_i and Y_i we then isolate all nodes in X_i and Y_i for the remainder of the graph. In other words, G_2 is defined like G_1 with the exception that, for each i, it aborts the interactions between X_i and Y_i at a random point. We refer to this as the "abort point". As before, observe that X is a (p,r)-cluster.

Indistinguishability: We now analyze the behavior of $\mathcal{A}$ when executed on these two dynamic graphs. Consider an execution in G_1. Let A_i be the nodes in $X_i \cup Y_i$ whose processes upload tokens at the end of their interactions. Observe that if $\exists i$ such that $A_i \cap Y_i > k$ then the (p,r)-cluster formed by $X_i \cup Y_i$ has more than k uploaders. Let p' be the probability that that an execution of $\mathcal{A}$ in G_1 leads to this case. Since, with high probability, the number of uploaders is $\leq k$, we conclude that $p' \leq 1/n$.

Consider now the case that occurs with probability at least $(1-1/n)$ where $A_i \cap Y_i \leq k$, in which case the tokens of at least $m-k=k$ nodes in Y_i are at some point transferred to a node in X_i.

We now we turn our attention, to G_2. For the purpose of this discussion, fix some i. Notice that for processes in Y_i that interact with X_i before the "abort point," the executions G_1 and G_2 are indistinguishable. Recall that in G_1, for each Y_i, there are at least k processes that rely on some other process to upload their token. Thus, for each of these processes, if the abort point in G_2 occurs after their interaction with X_i, they will continue to rely on other processes to upload their token, as they cannot distinguish G_1 and G_2.

For a given execution on graph G_2, let $v \in Y_i$ be the node that interacts with X_i in the last pass before the abort point. If v chooses not to upload its token (i.e., if it is one of those $\geq k$ processes in Y_i that rely on some other process to upload its token), then some process in X_i must upload the token of v since there are no further interactions after the abort point at which its token might be aggregated. (This follows from the the no-loss property and the notion of token ownership defined in Section 2).

We thus conclude with the following analysis. With probability at least $(1-1/n) \geq 1/2$, for every Y_i there are at least k nodes that rely on a process in X_i to upload their token. Moreover, with probability at least $k/m = 1/2$ the abort point occurs immediately after one of these $\geq k$ nodes interacts with X_i. Thus, with probability $\geq 1/4$, at least one node in X_i needs to upload tokens. In total, then, the expected number of nodes in X that upload tokens is at least $q/4 > k$. We conclude that with constant probability, the number of nodes in X that upload tokens is greater than k, which contradicts our assumption that $\mathcal{A}$ is a (p,k,r)-cluster-aggregation algorithm. □

4.2 An Upper Bound for Isolated Cliques

Here we show that for any $r \in \mathcal{O}(\log n)$ the ClusterAggregation$_p$ algorithm aggregates the tokens down to $\mathcal{O}(\log n)$ uploaders in every subset $S \subseteq V$ which forms a (p,r)-cluster. This comes within a $\mathcal{O}(\log n)$ factor of the minimal amount of isolation established by theorem 5.

Theorem 6. *Fix a constant $p \in (0,1]$, $r \in \mathcal{O}(\log n)$. For every dynamic graph G the system* (ClusterAggregation$_p$, G) *produces w.h.p., an execution where every (p,r)-cluster has $\mathcal{O}(\log n)$ uploaders.*

The proof of this theorem follows the same spirit as theorem 2. The key difference is that, in the context of a (p,r)-cluster, Fact 1 is no longer true. Specifically, consider any set $S \subset V$ which forms a (p,r)-cluster. It still holds that the $c_p \log n$ processes with the smallest random identifier in a set S cannot be tagged $c_p \log n$ times by other processes inside S. However, the nodes in S *can be* tagged $c_p \log n$ times (or more) by processes which are outside S.

Nevertheless, observe that since $r = r_0 \log n$ for some sufficiently large constant r_0, then by letting $c_p \geq 2_r 0$ we can guarantee $c_p \log n \geq 2r$, which implies that $c_p \log n - r \geq \frac{1}{2} c_p \log n$. Therefore, if we consider the $c_p \log n$ processes with the smallest identifiers in S, only half of them can get tagged by processes outside S, and therefore we have the following fact (in the same spirit as Fact 1).

Fact 2. Assume $c_p \geq 2r_0$. In every subset $S \subseteq V$ which forms a (p,r)-cluster, the processes with the smallest $\frac{1}{2} c_p \log n$ identifiers remain active throughout the execution.

With this fact in place, we can essentially reuse the proof of theorem 2 to show that if we fix a (p,r)-cluster, then with high probability we aggregate to $\mathcal{O}(\log n)$ nodes in that cluster. Since trivially there are never more than $\mathcal{O}(n)$ (p,r)-clusters, we can union bound over all clusters and show that with high probability ClusterAggregation$_p$ aggregates to $\mathcal{O}(\log n)$ processes in all (p,r)-clusters of a dynamic graph.

4.3 The Impossibility of Cover Aggregation in Cliques

Motivated by considering clusters embedded within a larger dynamic graph, here we consider an alternative natural measure for the performance of an aggregation algorithm (rather than the number of uploaders in every local cluster).

Specifically we propose considering for every local cluster, the number of nodes that upload a token *on behalf* of a node in the cluster. We call such nodes, the *cover-uploaders* of a cluster. Observe that the number of *cover-uploaders* of a cluster is a trivial upper bound on the number of *uploaders* for the same cluster, but the reverse is not true.

It is possible, for example, that no process in a local custer uploads a token, but that many processes outside the culster upload tokens on their behalf (i.e., no uploaders in the cluster but plenty of cover-uploaders outside the cluster).

For a $(p, 0)$-cluster the cover-uploaders are equivalent to the uploaders, so we turn our attention to larger values of r. Our main results below shows that even for $r = 1$ (i.e., almost completely isolated clusters), there are no aggregation algorithms that can guarantee less than $n^{1/3}$ cover-uploaders in every (p, r)-cluster. This is a strong separation with the number of uploaders, where we just proved that for polylog r we can guarantee polylog uploaders in every (p, r)-cluster.

Theorem 7. *For every aggregation algorithm $\mathcal{A}$ there exists a dynamic graph G where the system $(\mathcal{A}, G)$ produces with constant probabilty an execution where one $(1, 1)$-cluster has $\Omega(n^{1/3})$ cover-uploaders.*

Proof. Fix any aggregation algorithm $\mathcal{A}$ that guarantees at most k cover-uploaders in every $(1, 1)$-cluster. To show that $k \geq n^{1/3}$, we will construct a dynamic graph G which contains a $(1, 1)$-cluster where with constant probability the number of cover-uploaders is at least $n^{1/3}$.

To simplify notation, in the following assume that $k + 1$ divides n (for the case where this does not hold, our bound on k differs only by a constant factor). To define G we first partition the nodes into $n/(k+1)$ clusters $C_1, \ldots, C_{n/(k+1)}$ where each cluster has exactly $k+1$ nodes. We define the random set U of size $n/(k + 1)$ which contains for each C_i a single node u_i which was selected uniformly at random. In the first half of the execution we let all nodes in each C_i interact with each other (i.e., each C_i is a complete graph), and in the remainder of the execution, the nodes in U remain isolated, while all nodes in $V \setminus U$ interact with each other (thus $V \setminus U$ is a $(1, 1)$-cluster).

With G defined, we can examine the execution of $\mathcal{A}$ in this graph. Observe that if we were to stop the execution after all the interactions in each C_i cluster, then every C_i is a $(1, 0)$-cluster. Moreover since $|C_i| > k$, we can assume that with constant probability some aggregation must have occurred in every cluster (otherwise if we extend the execution with all nodes being isolated we would have proved the theorem). It follows that every cluster C_i has at least one *responsible* process v_i that owns at least two tokens, in the sense that it will output these two tokens if isolated from this point forward (as established by the notion of token ownership from Section 2).

Since in the rest of the execution the nodes in U are isolated, each of the responsible processes in U will broadcast a token on behalf of a unique process from $V \setminus U$ (and recall that the processes in $V \setminus U$ form a $(1, 1)$-cluster). Because the nodes nodes in U are picked uniformly at random, we expect $|U|/(k + 1) = n/(k + 1)^2$ processes in U to upload at the end of the execution. Therefore with constant probability the number of cover-uploaders (nodes which upload tokens on behalf) of the $(1, 1)$-cluster defined by $V \setminus U$ is at least $n/(k + 1)^2$. Finally, this implies that with constant probability $k \geq n^{1/3}$, since if $k \leq n^{1/3} - 1$ then the number of (cover-)uploaders in U is at least $k+1$ (which would be a contradiction). □

References

[AR07] J. Aspnes and E. Ruppert. An Introduction to Population Protocols. *Bulletin of the European Association for Theoretical Computer Science*, 93:98–117, 2007.

[CHS10] K. Censor Hillel and H. Shachnai. Partial Information Spreading with Application to Distributed Maximum Coverage. In *Proceedings of the International Symposium on Principles of Distributed Computing*, 2010.

[CN10] A. Cornejo and C. Newport. Prioritized Gossip in Vehicular Networks. In *Proceedings of the International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, 2010.

[Hae10] B. Haeupler. Analyzing Network Coding Gossip Made Easy. *Arxiv preprint arXiv:1010.0558*, 2010.

[HBZ+06] Bret Hull, Vladimir Bychkovsky, Yang Zhang, Kevin Chen, Michel Goraczko, Allen Miu, Eugene Shih, Hari Balakrishnan, and Samuel Madden. CarTel: a Distributed Mobile Sensor Computing System. In *Proceedings of the Conference on Embedded Networked Sensor System*, 2006.

[IEGH02] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann. Impact of Network Density on Data Aggregation in Wireless Sensor Networks. In *Proceedings of the Conference on Distributed Computing Systems*, 2002.

[JLR00] S. Janson, T. Luczak, and A. Rucinski. *Random Graphs*. John Wiley & Sons Inc., 2000.

[KEW02] L. Krishnamachari, D. Estrin, and S. Wicker. The Impact of Data Aggregation in Wireless Sensor Networks. In *Proceedings of the Conference of Distributed Computing Systems*, 2002.

[KLO10] Fabian Kuhn, Nancy Lynch, and Rotem Oshman. Distributed Computation in Dynamic Networks. In *Proceedings of the Symposium on Theory of Computing*, 2010.

[KMO11] Fabian Kuhn, Yoram Moses, and Rotem Oshman. Coordinated Consensus in Dynamic Networks. In *Proceedings of the International Symposium on Principles of Distributed Computing*, 2011.

[KO11] F. Kuhn and R. Oshman. Dynamic networks: models and algorithms. *ACM SIGACT News*, 42(1):82–96, 2011.

[MFHH02] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. Tag: a Tiny Aggregation Service for Ad-Hoc Sensor Networks. *ACM SIGOPS Operating Systems Review*, 36(SI):131–146, 2002.

[PPC99] Robert Pinheiro, Alex Poylisher, and Hamish Caldwell. Mobile Agents for Aggregation of Network Management Data. In *Proceedings of the International Symposium on Mobile Agents*, 1999.

[TRB+11] A. Thiagarajan, L. Ravindranath, H. Balakrishnan, S. Madden, and L. Girod. Accurate, Low-Energy Trajectory Mapping for Mobile Devices. In *Proceedings of the Symposium on Networked Systems Design and Implementation*, 2011.

A Token Ownership

Fix some execution *prefix* β of an aggregation algorithm, and a process $u \in V$. We use $i(\beta, u)$ to describe every extension of β in which process u has no further communication with other nodes (that is, u is isolated in the extension of β). The following lemma formalizes property (1) of token ownership:

Lemma 8. *Fix some execution prefix β of an aggregation algorithm, and a node $u \in V$. For every $\alpha, \alpha' \in i(\beta, u)$, $\alpha.\gamma[u] = \alpha'.\gamma[u]$. That is, in all isolated extensions of β, u outputs the same tokens.*

Proof. The lemma follows from the *no loss* and *no duplication* conditions of Definition 1. If there existed a pair $\alpha, \alpha' \in i(\beta, u)$ such that $\alpha.\gamma[u] \neq \alpha'.\gamma[u]$, then there would be a probability greater than 0 that one of these two properties would be violated in some execution. □

We say $(u, r) \rightsquigarrow_G (v, r')$, for $u, v \in V$, rounds $r' \geq r \geq 1$, and dynamic graph $G = (V, \mathcal{E})$, if and only if: (a) $u = v$; or (b) there is a sequence of edges, $e_r = \{u, u_r\}, e_{r+1} = \{u_r, u_{r+1}\}, ..., e_{r'} = \{u_{r'}, v\}$, such that for every $i \in \{r, ..., r'\}$: $e_i \in \mathcal{E}(i)$, (Informally, $(u, r) \rightsquigarrow_G (v, r')$ indicates that there exists a sequence of edges by which u can pass information it knows at the beginning of r to v by r' in G.) The following lemma formalizes property (2) of token ownership:

Lemma 9. *Let α be an execution of an aggregation algorithm with dynamic graph G of duration t. For every $u \in V$ and token $x \in \alpha.\gamma[u]$: there exists $v \in V$ such that $x = \alpha.\sigma[v]$ and $(v, 1) \rightsquigarrow_G (u, t)$. That is, if u outputs a token x then there is a path from the process that started with x to u in G.*

Proof. If this lemma did not hold then some process v might output a token x even though there is no path from a process that started with x to process v. We consider two cases: First, if there is a path from every process to v in this execution, then the execution violates the no loss property of Definition 1. Second, if there are instead some processes that do not have paths to v, it is possible that one of these processes started with x, in which case process v may have made a lucky guess by outputting x. However, because there is no interaction with these processes and v, this execution is indistinguishable with respect to v from one in which none of these processes started with x: and in this execution the no loss property is once again violated. □

Notice that as a result of these lemmas, there is no benefit to *copying* tokens: each must be owned by a single process at any given time.

B Proofs for Section 3

Before we can prove lemma 3 we need to state a slight generalization of the traditional Chernoff bound.

Generalized Chernoff Bounds. *Let Z be a random variable with Binomial or Hypergeometric distribution, and let $\mu = \mathbb{E}[Z]$. Then* $\Pr[Z \geq (1+\delta)\mu] \leq e^{-\frac{\delta^2\mu}{3}}$ *and* $\Pr[Z \leq (1-\delta)\mu] \leq e^{-\frac{\delta^2\mu}{2}}$.

The Binomial distribution describes the number of successes in a sequence of draws *with* replacement, and can be expressed as a sum of independent random variables representing each draw. On the other hand, the Hypergeometric distribution describes the number of successes in a sequence of draws *without* replacement, and hence the draws are not independent.

Typically, the Chernoff bound applies to random variables with a binomial distribution. However, as noted by [JLR00], it seems reasonable to expect that drawing samples without replacement would produce smaller random fluctuations than sampling with replacement, so the fact that the Chernoff bound also holds for the Hypergeometric distribution is not entirely surprising. We omit the proof of this generalized bound since it can be found in Theorem 2.1 and 2.10 in [JLR00].

Now lemma 3 follows by a trivial application of this generalized Chernoff bound.

Proof of lemma 3. We have a total of $|V|$ processes, we are interacting with $p|V|/2$ of them (without replacement), and we want to count how many times we meet a process in $Q \subseteq V$. Observe that X_u has a Hypergeometric distribution, and hence $\mathbb{E}[X_u] = \frac{p}{2}|Q|$. The theorem follows from the generalized Chernoff bounds (with $\delta = \frac{1}{2}$). □

Proof of lemma 4. Fix $u \in V \setminus Q$ and let I_1 and I_2 correspond to the first $p|V|/2$ interactions, and the remaining interactions (at least $p|V|/2$) respectively.

Define X as the event that process u does not meet a single process of Q in the interactions of I_2. We want to show that the probability that X occurs is exponentially small (on $|Q|$). Define Y as the event that process u meets with more than $\frac{3p}{4}|Q|$ fraction of the processes in Q during the interactions I_1. By lemma 3 $\Pr[Y] \leq e^{-\frac{p}{24}|Q|}$.

Now consider $\Pr[X|\neg Y]$, that is the event that process u does not meet a single process in Q during the interactions of I_2, conditioned on the fact that there are at least $\frac{1}{4}|Q|$ processes with which it didn't interact in I_1. Observe that until process u meets one of those $\frac{1}{4}|Q|$ remaining processes in Q, then at every interaction of I_2 it will meet a process in Q with probability at least $\frac{1}{4}|Q|/|V|$. Therefore:

$$\Pr[X|\neg Y] \leq \left(1 - \frac{1}{4}\frac{|Q|}{|V|}\right)^{\frac{p}{2}|V|} \leq e^{-\frac{p}{8}|Q|}$$

Finally, by the law of total probability we have $\Pr[X] = \Pr[X|Y]\Pr[Y] + \Pr[X|\neg Y]\Pr[\neg Y] \leq \Pr[Y] + \Pr[X|\neg Y] \leq e^{-\frac{p}{8}|Q|} + e^{-\frac{p}{24}|Q|} \leq 2e^{-\frac{p}{24}|Q|} \leq e^{-\frac{p}{24}|Q|+1}$, and since $|Q| \geq 48/p$ then $\Pr[X] \leq^{-\frac{p}{48}|Q|}$ □